小米电池预警系统

1. 需求分析

1.1 项目背景

- BMS系统（电池管理系统）是智能化管理及维护各个电池单元，防止电池出现过充电和过放电、延长电池的使用寿命、监控电池状态的系统。在BMS系统中存在大量电池各种信号的规则管理以及监控，良好的信号处理，并且根据规则，生成相关预警信息，能够极大地提升用户体验。

1.2 功能需求

1.2.1 车辆信息存储

- 支持车辆信息（vid, 车架编号, 电池类型, 总里程(km), 电池健康状态(%)）的录入。
- 车辆信息录入的原因是：先有车才有电池，最后才会在车行驶中产生电流信号。
- 需要设计车辆信息存储的表结构，并将给定的信息存储到汽车信息表中。

1.2.2 规则配置

- 支持规则（包括：序号，规则编号，名称，预警规则，电池类型）的配置。
- 预警规则包含预警规则描述以及预警等级（0级最高响应）。
- 电池类型：不同类型电池对应规则不同。
- 信号：Mx（最高电压）, Mi（最小电压）、Ix（最高电流）, Ii（最小电流）。
- 需要设计车辆上报信号和规则的存储的表结构，并将给定的数据存储到规则表中。

1.2.3 上报电池信号功能

- 能通过接口上报电池信号状态，完成数据库的增删改查。

1. 查询电池信号功能
   - 查询电池信号状态，要求接口使用Redis做缓存，且保证缓存和数据库数据的一致性。

1.2.4 预警功能

- 通过定时任务扫描电池信号数据，通过发送MQ消息，消费MQ消息生成预警信息。
- 支持通过预警接口查询指定车辆的预警信息。

1.3 技术需求

1.3.1 技术栈

- Java技术栈，SpringCloudAlibaba，SpringBoot 2.0+，Mybyties，Mysql 5.7+，Redis，http/https，MQ。

1.3.2 技术方案

- 必须包含系统设计、数据库表设计、接口设计、缓存和数据库数据一致性、单元测试。
- 所有接口必须100%单测覆盖。

1.4 性能需求

- 系统每天处理信号量为百万甚至千万数据级别：考虑数据量对系统性能的影响，给出合理设计数据存储和查询方案。

2. 可行性分析

2.1 技术可行性

2.1.1 Java技术栈

- Java技术栈成熟稳定，SpringBoot框架提供了快速开发和部署的能力，Mysql数据库广泛应用于企业级应用，Redis作为缓存解决方案能够有效提升查询性能，MQ（如RabbitMQ、Kafka等）能够实现高效的异步消息处理。

2.1.2 性能优化

- 通过Redis缓存电池信号状态，减少数据库的直接访问，提高查询效率。
- 使用MQ进行异步处理，避免同步调用带来的性能瓶颈。
- 对规则解析和预警计算进行性能测试和优化，确保P99响应时间在1s以内。

2.2 经济可行性

2.2.1 开发成本

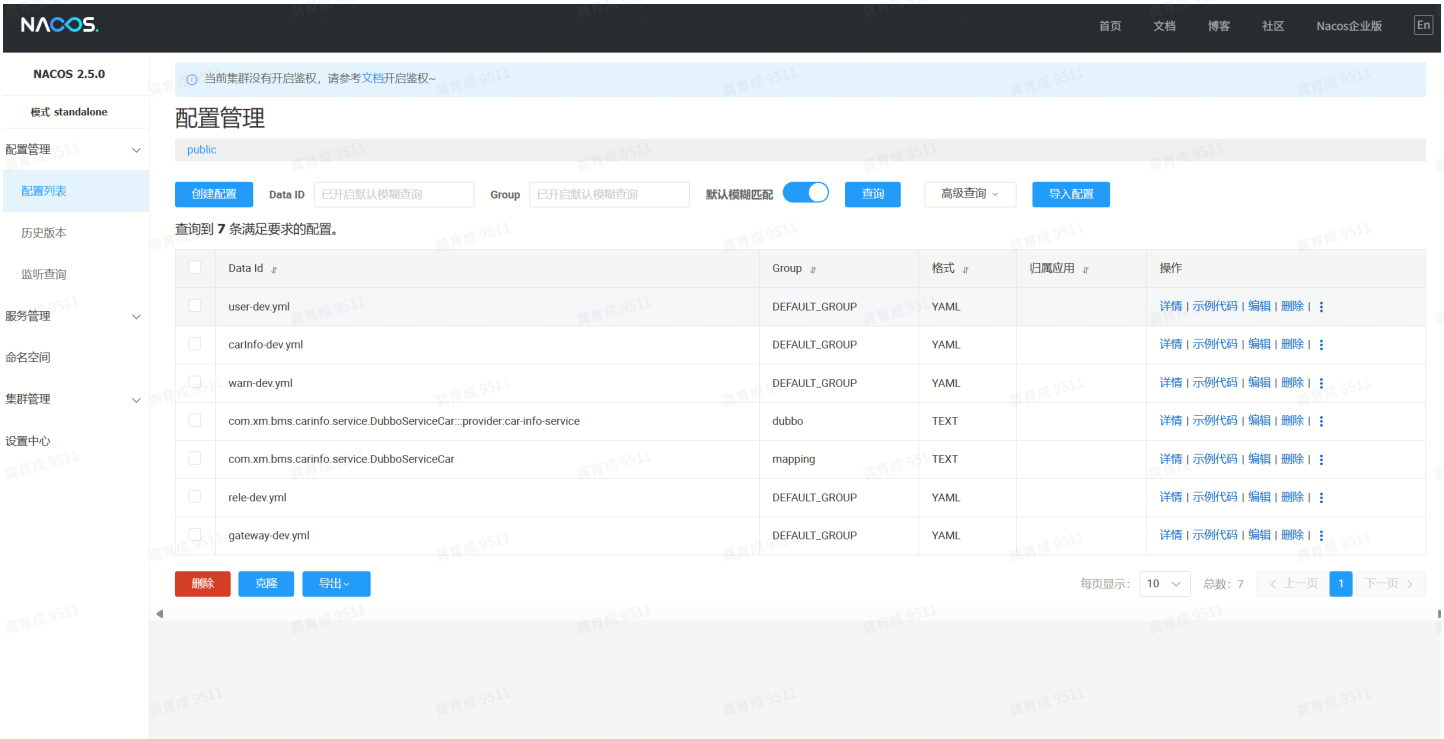
- 使用开源技术栈（SpringBoot、Mysql、Redis、MQ），开发成本较低。
- 项目规模适中，开发周期可控，人力成本在合理范围内。

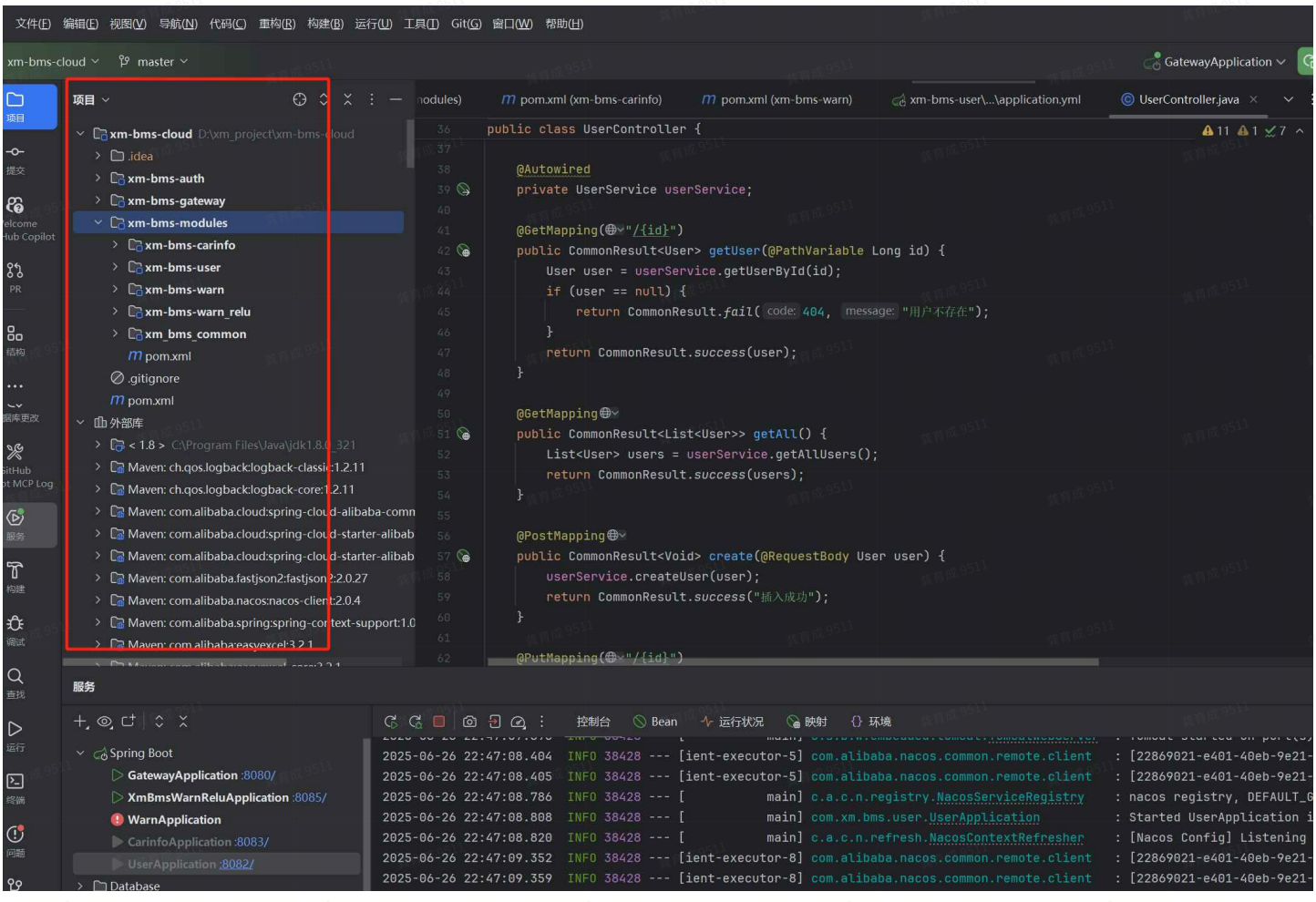
2.2.2 运营成本

- 系统基于成熟的开源技术栈，运维成本较低。
- 数据存储和查询方案合理，能够有效应对大规模数据量，减少硬件投入。

3. 总体设计

- 使用nacos作为注册中心和配置中心。





3.1 系统架构

3.1.1 分层架构

- 数据层：Mysql数据库存储车辆信息、用户信息、规则配置、电池信号数据、预警信息等。
- 缓存层：Redis缓存电池信号状态和用户信息，提高查询性能。
- 业务逻辑层：SpringBoot框架实现业务逻辑，包括用户信息管理、车辆信息管理、车辆预警信息管理、电池预警规则管理等。
- 消息队列层：MQ实现异步消息处理，提高系统性能。
- 接口层：提供RESTful API接口，供前端或其他系统调用。
- 网关层：使用Spring Cloud Gateway实现API网关，进行请求路由、权限控制等。
- 公共模块：用于统一异常处理和结果返回包装。

3.2 系统模块

3.2.1 网关模块（Gateway）

- 负责请求路由、权限验证、限流等功能。

3.2.2 权限控制模块（Auth）

- 负责用户认证、授权、权限管理等功能。

3.2.3 用户信息管理模块（User Management）

- 负责用户信息的录入、查询、更新和删除。（批量导入或更新，导出列表，导入模板下载）

3.2.4 车辆信息管理模块（Vehicle Management）

- 负责车辆信息和车辆状态信息的录入、查询、更新和删除。（批量导入或更新，导出列表，导入模板下载）

3.2.5 车辆预警信息管理模块（Vehicle Warn Management）

- 负责车辆预警信息的生成、查询和管理。（批量导入或更新，导出列表，导入模板下载）

3.2.6 电池预警规则管理模块（Battery Warn Rule Management）

- 负责电池预警规则的添加、查询、更新和删除。

3.2.7 公共模块（Common）

- 提供统一异常处理、结果返回包装等通用功能。

4. 详细设计

4.1 数据库表设计

4.1.1 用户信息表（user_info）

- `user_id` (INT, 主键)：用户ID。
- `username` (VARCHAR(50))：用户名。
- `password` (VARCHAR(100))：密码（加密存储）。
- `role` (VARCHAR(20))：用户角色（如管理员、工程师、车主等）。
- `create_time` (DATETIME)：创建时间。

4.1.2 车辆信息表（vehicle_info）

- `vid` (VARCHAR(16), 主键)：车辆识别码，每辆车唯一。
- `car_id` (INT)：车架编号。
- `battery_type` (VARCHAR(20))：电池类型（三元电池、铁锂电池）。
- `total_km` (INT)：总里程（km）。
- `battery_health` (INT)：电池健康状态（%）。
- `user_id` (INT)：所属用户ID。

4.1.3 预警规则配置表（rule_config）

- `rule_id` (INT, 主键)：规则编号。
- `rule_name` (VARCHAR(50))：规则名称。
- `battery_type` (VARCHAR(20))：电池类型。
- `warn_rule` (TEXT)：预警规则描述。
- `warn_level` (INT)：预警等级。

4.1.4 车辆信号表（battery_signal）

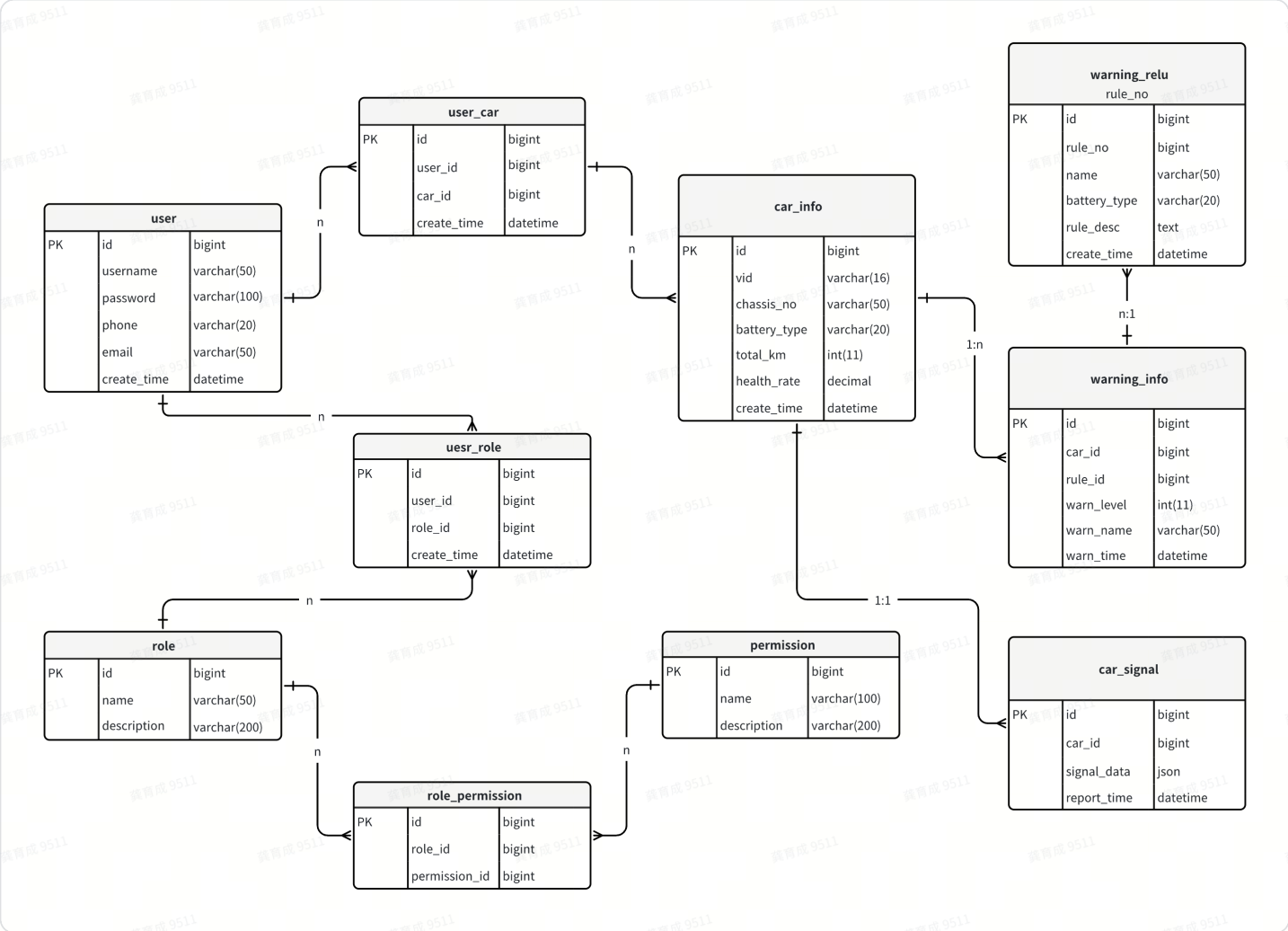
- `id` (INT, 主键)：信号编号。
- `car_id` (INT)：车架编号。
- `signal_data` (TEXT)：信号数据（JSON格式）。
- `create_time` (DATETIME)：信号上报时间。

4.1.5 预警信息表（warn_info）

- `warn_id` (INT, 主键)：预警编号。

- `car_id` (INT)：车架编号。
- `warn_name` (VARCHAR(50))：预警名称。
- `warn_level` (INT)：预警等级。
- `create_time` (DATETIME)：预警生成时间。

4.2 MYSQL放在云服务器上，减少本地主机压力



4.3 接口设计

4.3.1 网关模块（Gateway）

接口名：`/api/gateway`

接口方法：`POST`

Body：

- `path` (String)：请求路径。
- `token` (String)：用户令牌。

返回信息：

`code` (int)：状态码。

`message` (String)：请求成功或失败信息。

`data` (JSON)：路由后的响应数据。

4.3.2 用户信息管理模块（User Management）

1.用户信息查询接口

接口名： `/api/users/{id}` 接口方法： `GET` 功能描述：

根据用户ID查询用户信息。

如果用户存在，返回用户详细信息；如果用户不存在，返回404错误。

返回信息：

`code` (int)：状态码

`message` (String)：请求成功或失败信息。

`data` (User)：用户信息对象。

2.用户信息列表查询接口

接口名： `/api/users` 接口方法： `GET` 功能描述：

查询所有用户信息。

返回信息：

`code` (int)：状态码。

`message` (String)：请求成功或失败信息。

`data` (List<User>

3.用户信息创建接口

接口名： `/api/users` 接口方法： `POST` Body：

`username` (String)：用户名。

`password` (String)：密码。

`role` (String)：用户角色。

4.根据用户ID更新用户信息。

返回信息：

`code` (int)：状态码。

`message` (String)：请求成功或失败信息。

用户信息删除接口

接口名： `/api/users/{id}` 接口方法： `DELETE` 功能描述：

5.根据用户ID删除用户信息。

如果用户存在，删除成功；如果用户不存在，返回404错误。
返回信息：

`code` (int)：状态码。

`message` (String)：请求成功或失败信息。

6.用户数据导出接口

接口名： `/api/users/export` 接口方法： `POST` 功能描述：

导出所有用户信息为Excel文件。

设置响应头，使浏览器下载文件。
返回信息：

7.Excel文件流。

用户数据导入接口

接口名： `/api/users/import` 接口方法： `POST` 功能描述：

导入用户信息Excel文件。

读取Excel文件内容，批量插入或更新用户信息。
返回信息：

`code` (int)：状态码。

`message` (String)：请求成功或失败信息。

8.用户导入模板下载接口

接口名： `/api/users/importTemplate` 接口方法： `GET` 功能描述：

提供用户信息导入模板的下载。

设置响应头，使浏览器下载文件。
返回信息：

Excel模板文件流。

4.3.3 车辆信息管理模块（Vehicle Management）

车辆信息录入接口

接口名： `/api/carinfos/add`

接口方法： `POST`

Body：

`vid` (String)：车辆识别码。

`car_id` (int)：车架编号。

`battery_type` (String)：电池类型。

`total_km` (int)：总里程。

`battery_health` (int)：电池健康状态。

`user_id` (int)：所属用户ID。

返回信息：

`code` (int)：状态码。

`message` (String)：请求成功或失败信息。

车辆信息查询接口

接口名：`/api/vehicle/query`

接口方法：`GET`

参数：

`car_id` (int)：车架编号。

返回信息：

`code` (int)：状态码。

`message` (String)：请求成功或失败信息。

`data` (JSON)：车辆信息。

4.3.4 车辆预警信息管理模块（Vehicle Warn Management）

预警信息生成接口

接口名：`/api/warns/generate`

接口方法：`POST`

Body：

`car_id` (int)：车架编号。

`signal` (String)：信号数据（JSON格式）。

返回信息：

`code` (int)：状态码。

`message` (String)：请求成功或失败信息。

预警信息查询接口

接口名：`/api/warn/query`

接口方法：`GET`

参数：

`car_id` (int)：车架编号。

返回信息：

`code` (int)：状态码。

`message` (String)：请求成功或失败信息。

`data` (JSON)：预警信息。

4.4 公共模块（Common）

4.4.1 统一异常处理

- 捕获系统中所有的异常，返回统一的错误信息。

4.4.2 结果返回包装

- 将接口的返回结果进行统一包装，包括状态码、消息和数据。

4.5 缓存设计

- 使用Redis缓存用户信息和电池信号状态，减少数据库的直接访问，提高查询效率。

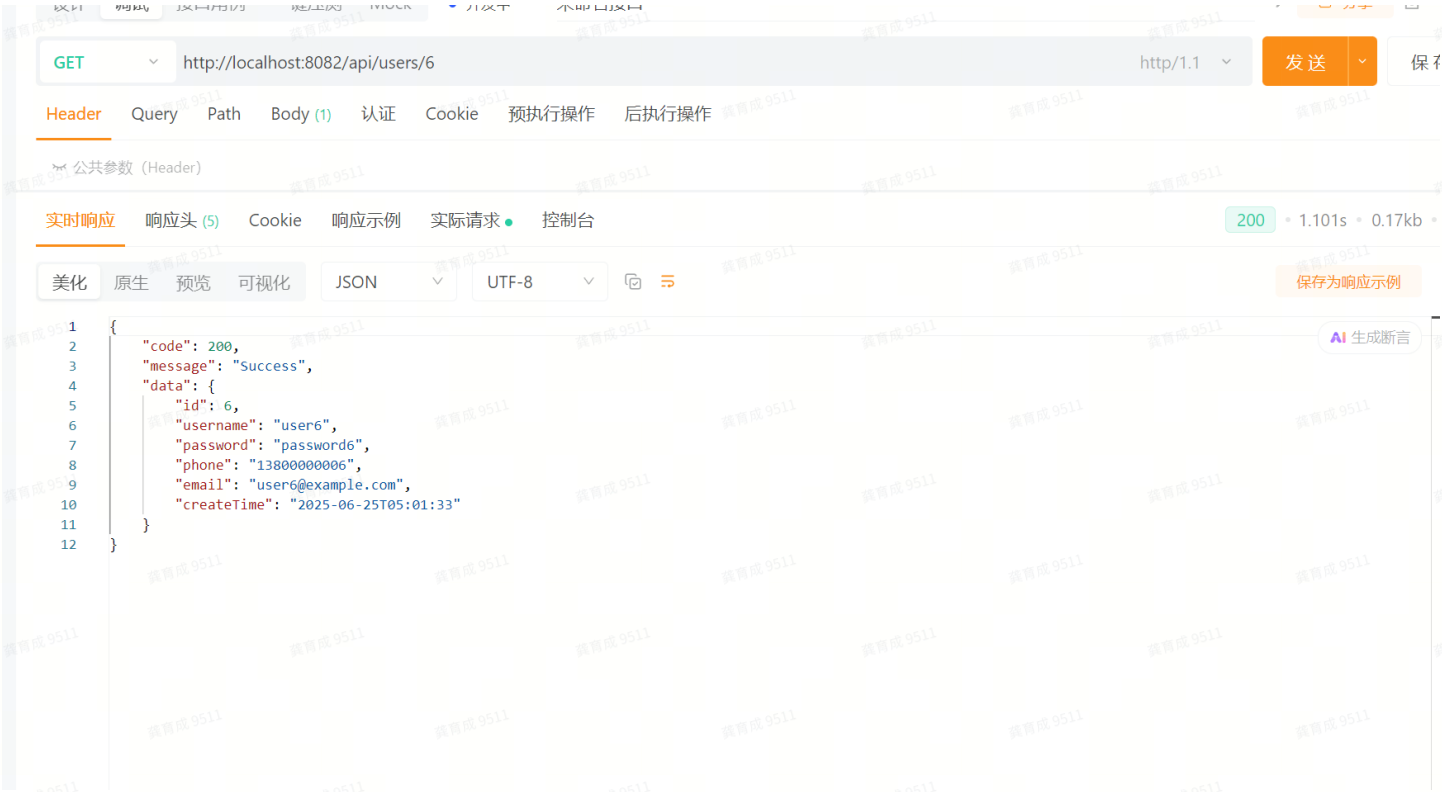
5. 编码和测试

> - 编码见github：https://github.com/gongyucheng0505/xm-bms-cloud.git

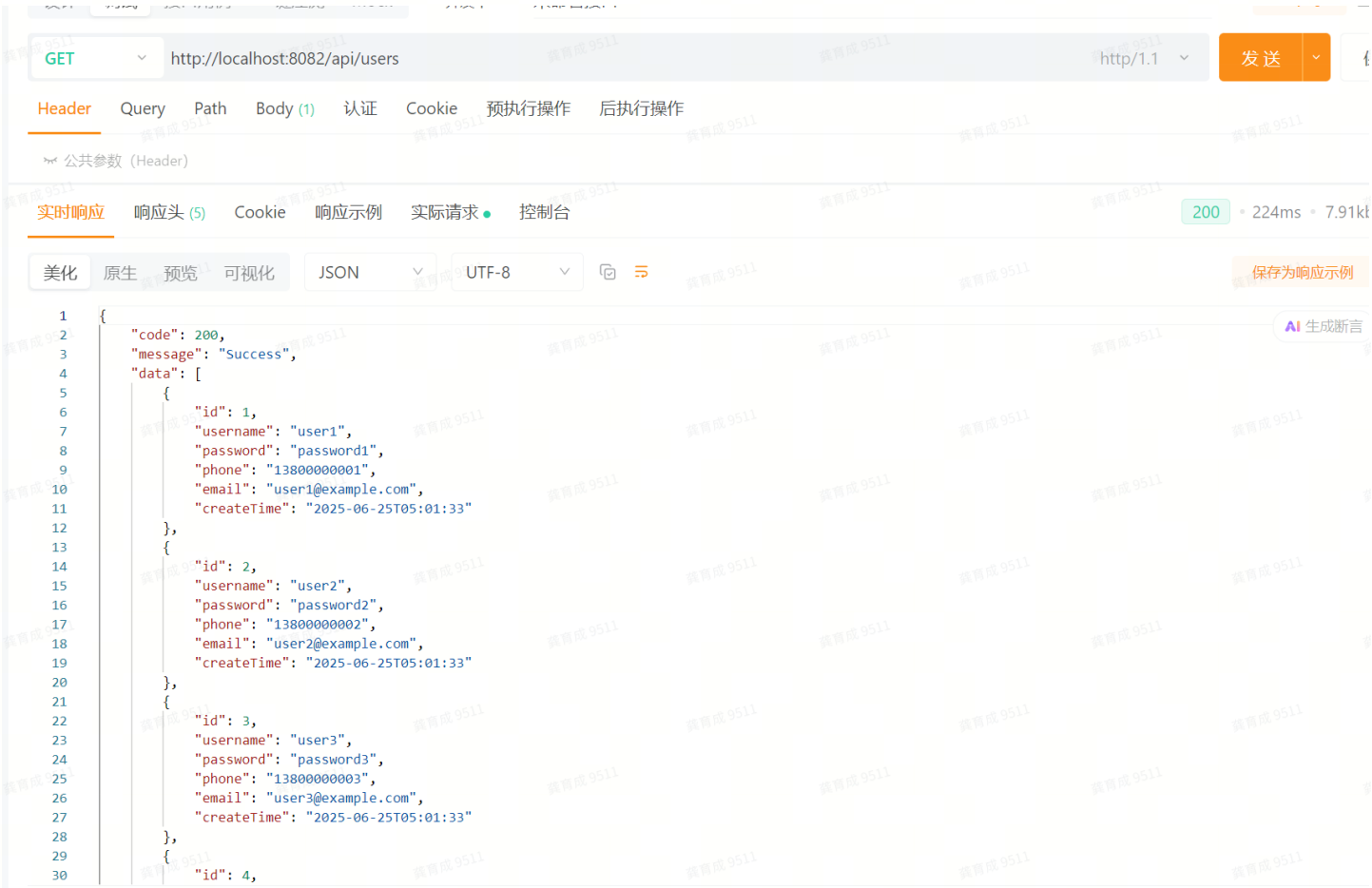
5.1 测试截图

5.1.1 用户信息模块

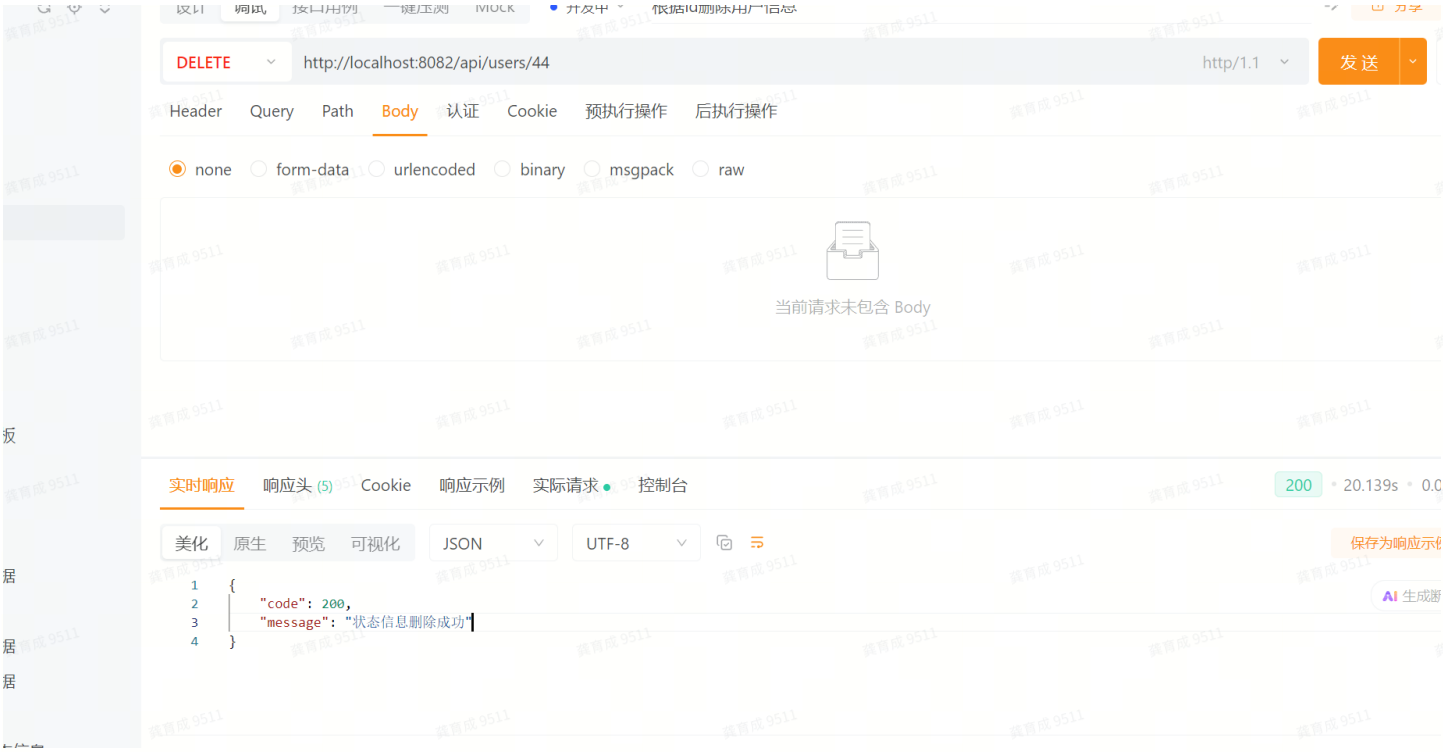
1.根据id查询用户信息



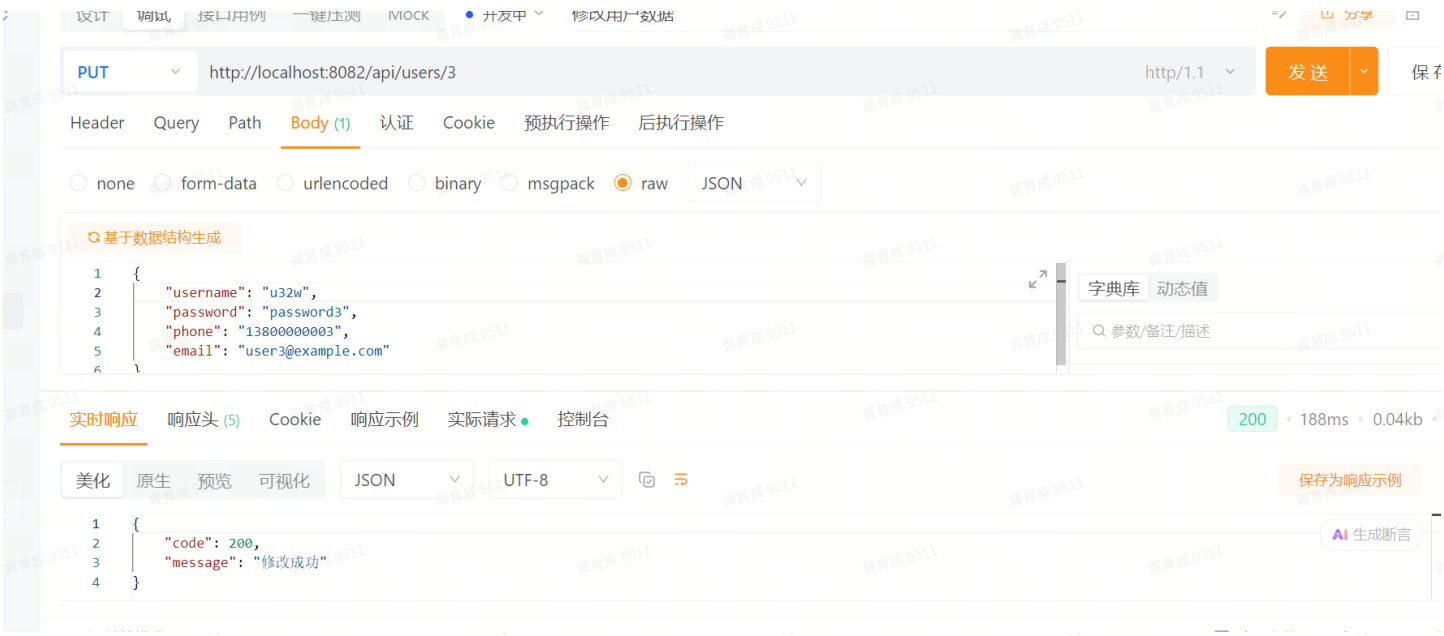
2.查询所有用户信息



3.根据id删除用户信息（删除成功不存在就返回记录不存在）

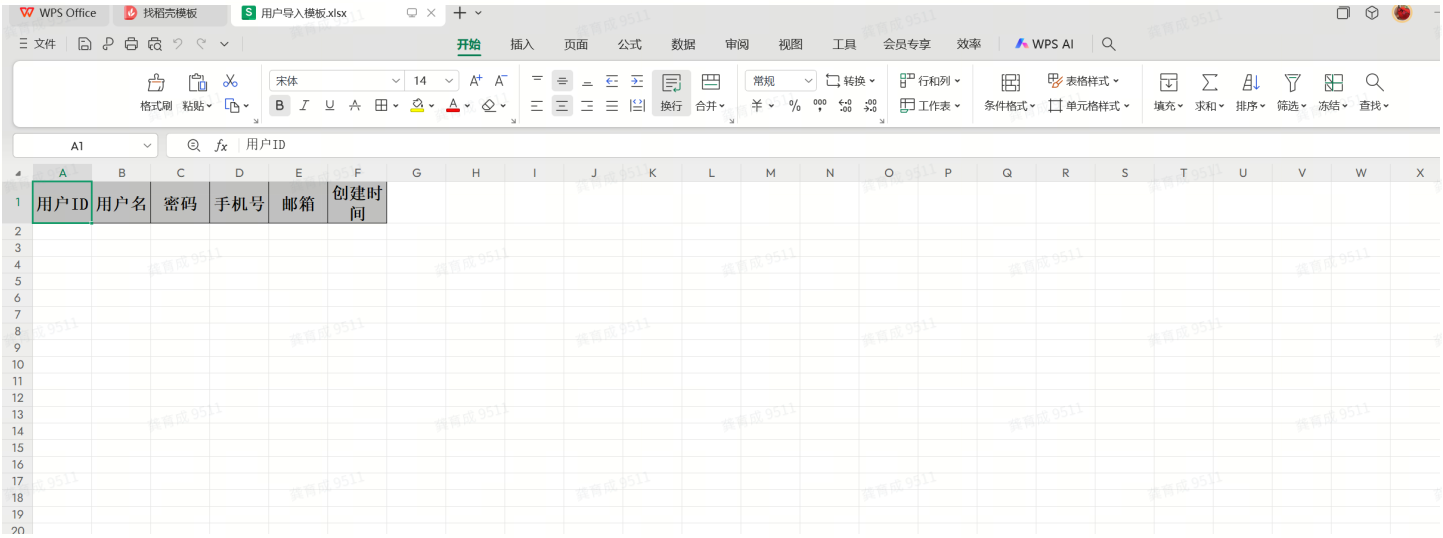


4.根据id修改用户信息



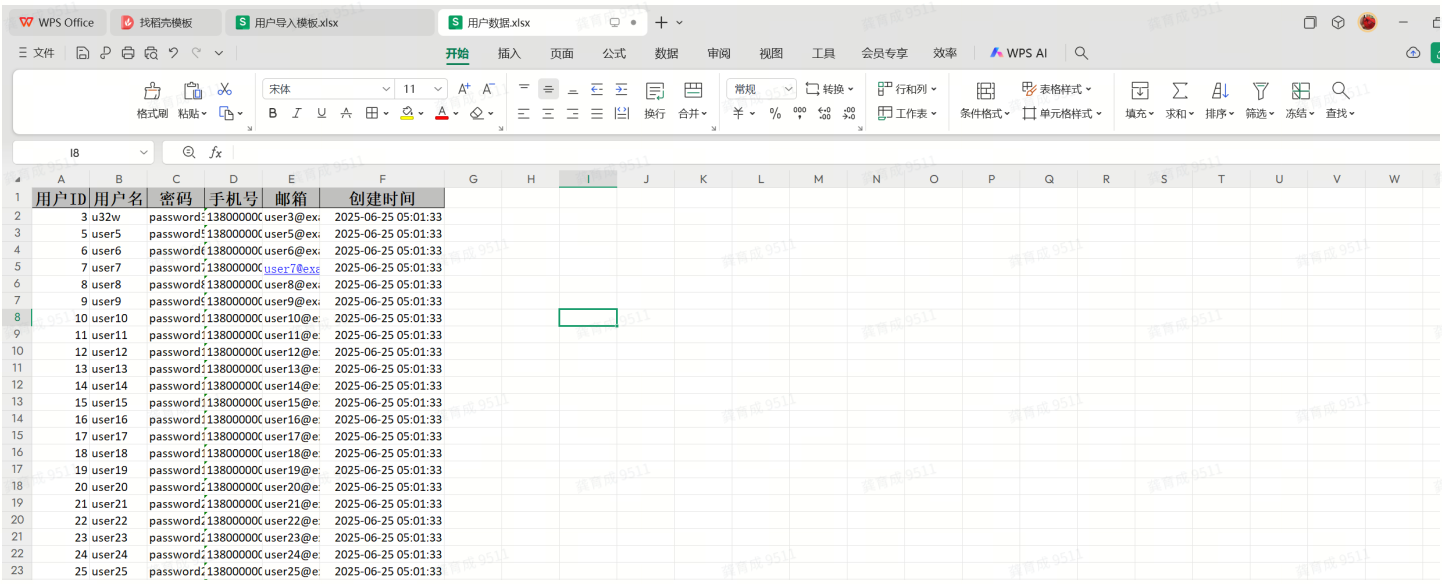
5.单条插入用户信息

6.下载用户信息模板



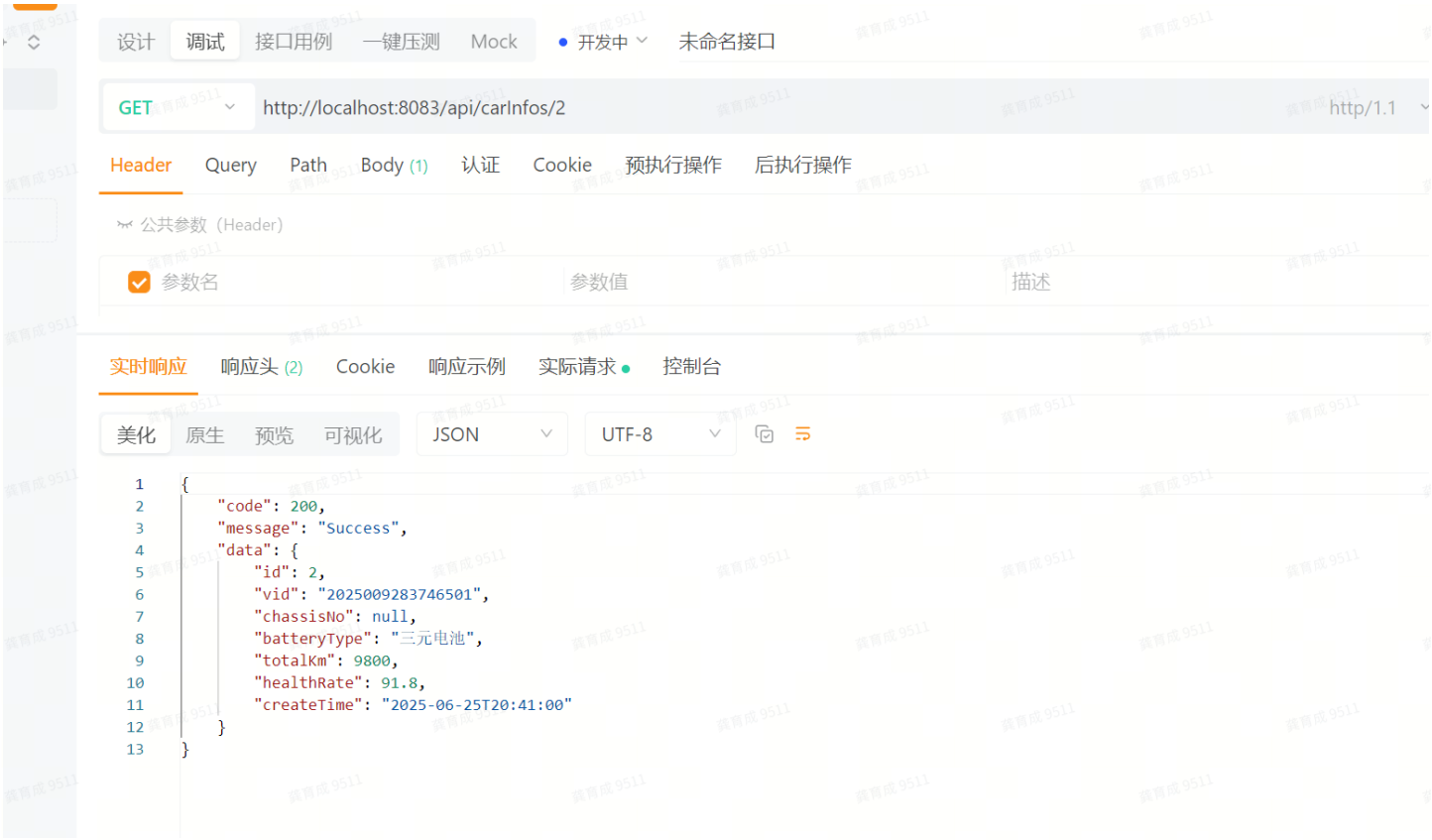
7.批量导入用户信息

8.导出车辆用户信息

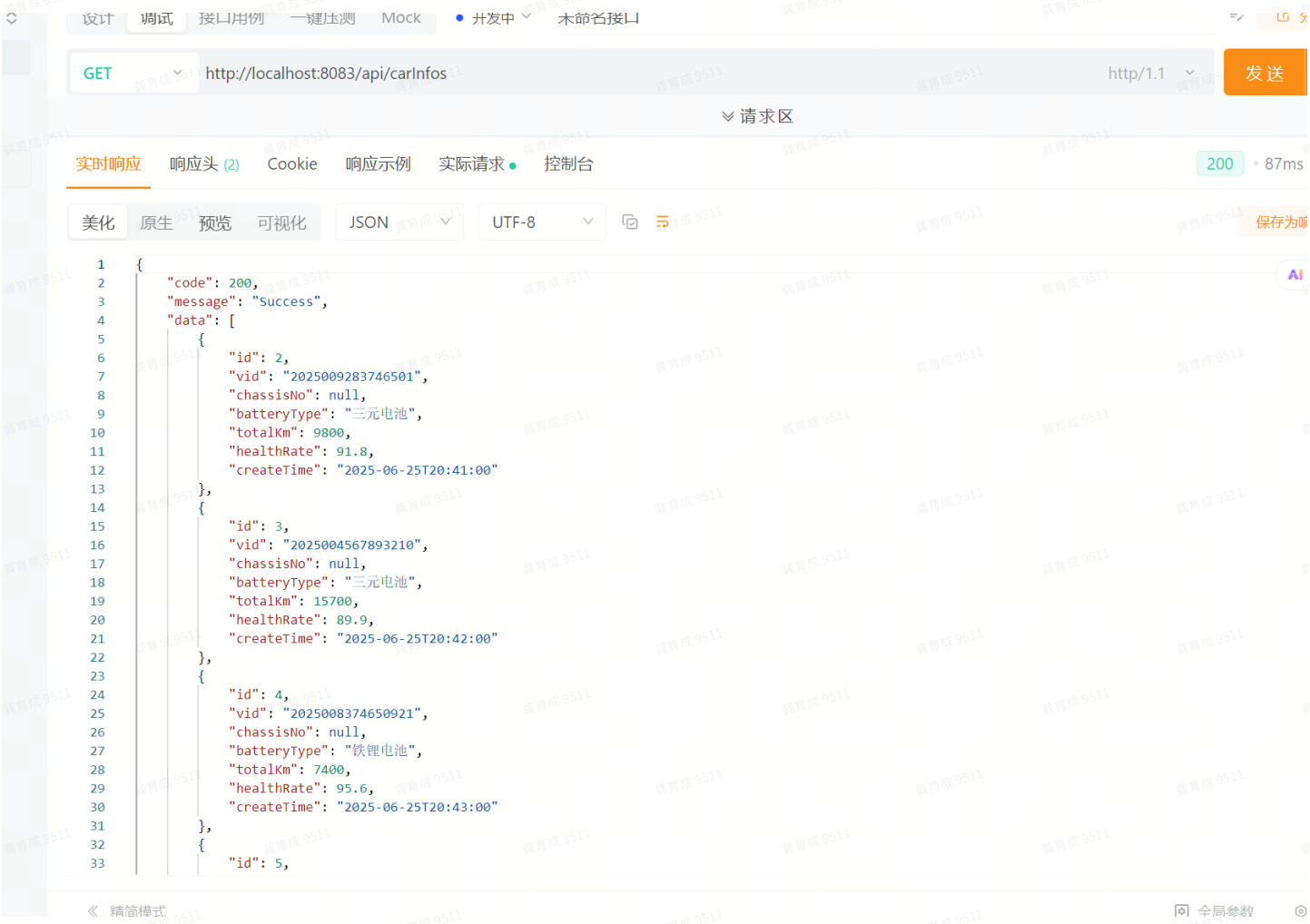


5.1.2 车辆信息模块

1.根据id查询车辆信息



2.查询所有车辆信息



读取缓存，不存在就查询数据库，再写入redis缓存

3.根据id删除车辆信息

首先删除 与车id关联的车辆状态信息。再实现延迟双删，先删除缓存，再更新数据库，等100ms删除缓存（两个缓存都要删掉，防止脏数据）

```java
@Transactional
@Override
public int deleteCarInfo(Long id) {
    // 1. 删除缓存，防止脏数据
    try (Jedis jedis = jedisClient.getJedis()) {
        jedis.del("car:info:" + id);  // 删除车辆信息缓存
        jedis.del("car:signal:" + id);  // 删除车辆状态信息缓存
    } catch (Exception e) {
        e.printStackTrace();  // 打印异常堆栈信息
    }

    // 2. 延时删除缓存，防止缓存未及时更新
    try {
        Thread.sleep(200);  // 延时100ms确保缓存删除操作完成
    } catch (InterruptedException ignored) {
        ignored.printStackTrace();  // 打印异常
    }

    // 3. 删除与车辆ID相关的状态信息
    carSignalMapper.deleteCarSignal(id);  // 确保你已在 Mapper 中定义该方法

```

```
    // 4. 删除车辆信息
    int result = carInfoMapper.deleteById(id);

    // 5. 再次删除缓存（如果需要的话）
    try (Jedis jedis = jedisClient.getJedis()) {
        jedis.del("car:info:" + id);  // 再次删除缓存
        jedis.del("car:signal:" + id);  // 删除车辆信息缓存

    } catch (Exception e) {
        e.printStackTrace();  // 打印异常堆栈信息
        // 或者使用日志记录，比如：logger.error("延迟删除缓存失败", e);
    }

    return result;  // 返回删除结果
}
```

4.根据id修改车辆信息

先删除缓存，等100ms，再更新数据库，

```
public void updateCarInfo(CarInfo carInfo) {
    // 1. 更新数据库
    carInfoMapper.update(carInfo);
        // 2. 删除旧缓存
        try (Jedis jedis = jedisClient.getJedis()) {
            jedis.del("car:info:" + carInfo.getId());  // 删除缓存
```

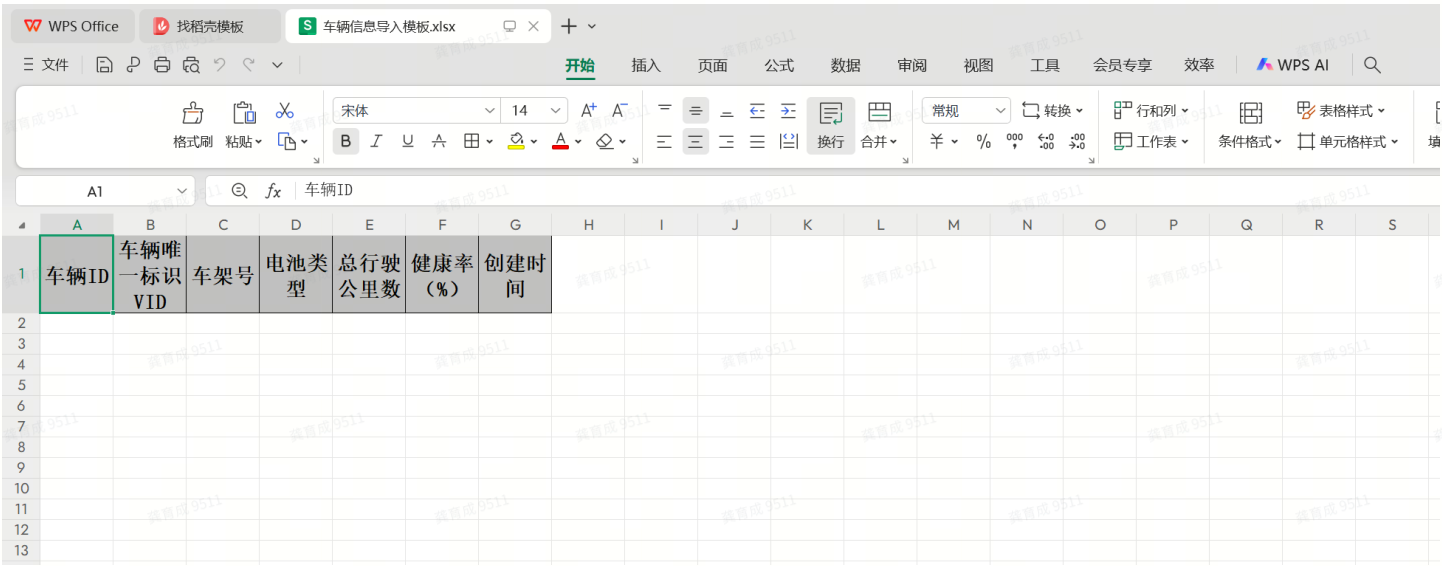
```
        } catch (Exception e) {
            // 如果删除缓存失败，记录日志
        }
        // 3. 等待缓存清理完成后再更新缓存
        try {
            Thread.sleep(100);  // 延迟100ms，确保缓存被清理
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    // 4. 将新数据写入缓存
    try (Jedis jedis = jedisClient.getJedis()) {
        String key = "car:info:" + carInfo.getId();
        jedis.setex(key, 1800, objectMapper.writeValueAsString(carInfo));  // 设置30分钟过期时间
    } catch (Exception e) {
        // 如果写入缓存失败，记录日志
    }
}
```

5.单条插入车辆信息

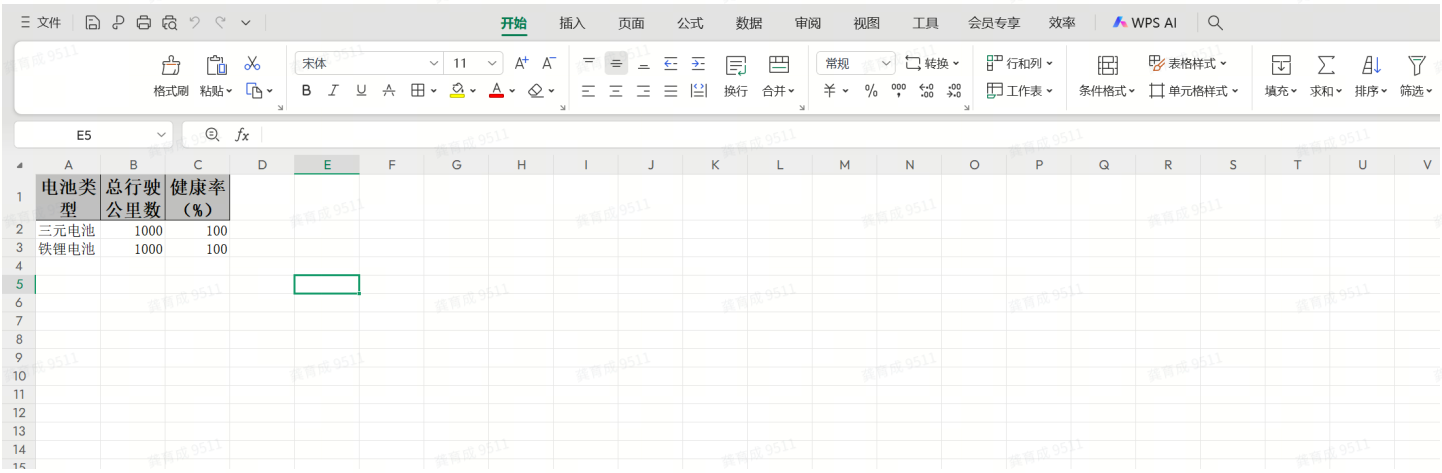

```
public void createCarInfo(CarInfo carInfo) {
    LocalDateTime now = LocalDateTime.now();
    carInfo.setCreateTime(now);
    // 生成16位VID：以"XM2025"开头 + 10位随机数字
    String vid = "XM2025" + RandomStringUtils.randomNumeric(10);
    carInfo.setVid(vid);

    carInfoMapper.insert(carInfo);
```

```
9 }
```

6.下载车辆信息导入模板

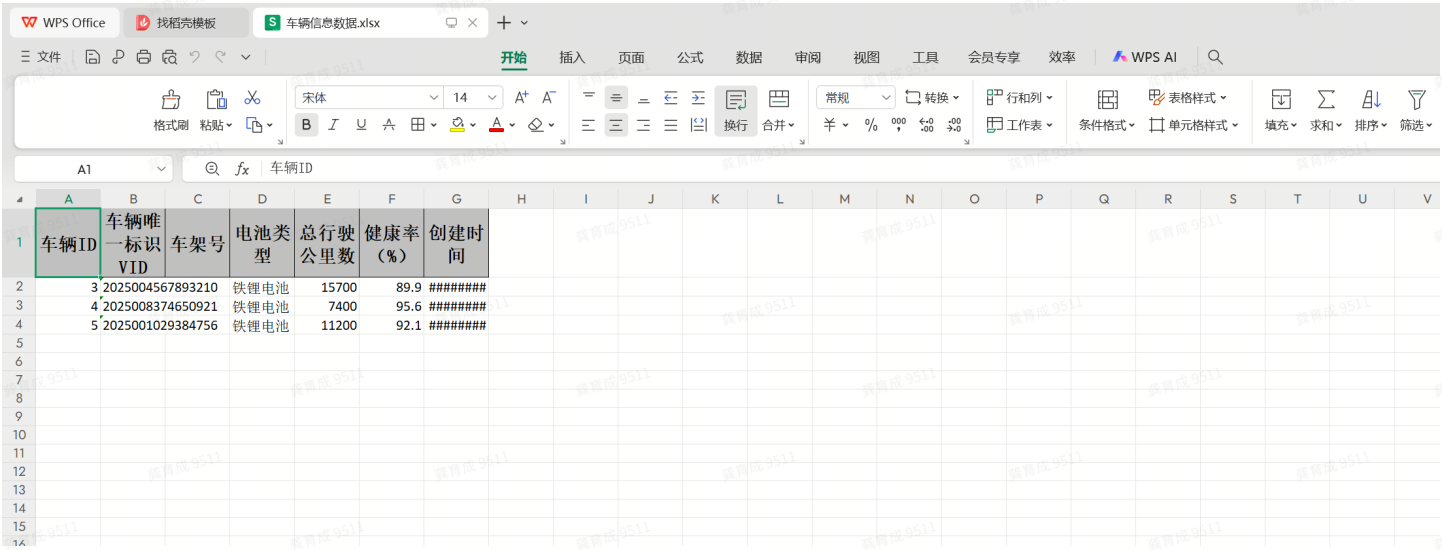


7.批量导入车辆信息



```java
LocalDateTime now = LocalDateTime.now();
for (CarInfo carInfo : carInfoList) {
    if (carInfo.getCreateTime() == null) {
        carInfo.setCreateTime(now);
    }
    // 生成16位vid，比如 "XM2025" + 10位数字随机数
    String vid = "XM2025" + RandomStringUtils.randomNumeric(10);
    carInfo.setVid(vid);
}
```

8.导出车辆信息列表



车辆ID	车辆唯一标识VID	车架号	电池类型	总行驶公里数	健康率（%）	创建时间
3	2025004567893210		铁锂电池	15700	89.9	########
4	2025008374650921		铁锂电池	7400	95.6	########
5	2025001029384756		铁锂电池	11200	92.1	########

5.1.3 车辆状态信息模块（改查删都用车id，新增需要判断车id是否存在）

1.根据id查询车辆状态信息

读取缓存，不存在就查询数据库，再写入redis缓存

```java

public CarSignal getCarSignalById(Long id) {
    String key = CACHE_PREFIX id;
    try (Jedis jedis = jedisClient.getJedis()) {
        // 1. 读缓存
        String json = jedis.get(key);
        if (json != null) {
            return objectMapper.readValue(json, CarSignal.class);
        }
        // 2. 未命中，查数据库
        CarSignal carSignal = carSignalMapper.selectById(id);
        if (carSignal != null) {
            // 3. 写缓存，设置30分钟过期
            jedis.setex(key, 1800, objectMapper.writeValueAsString(carSignal));
        }
        return carSignal;
    } catch (Exception e) {
        // Redis 出错，降级查询数据库
        return carSignalMapper.selectById(id);
    }
}
```

2.查询所有车辆状态信息

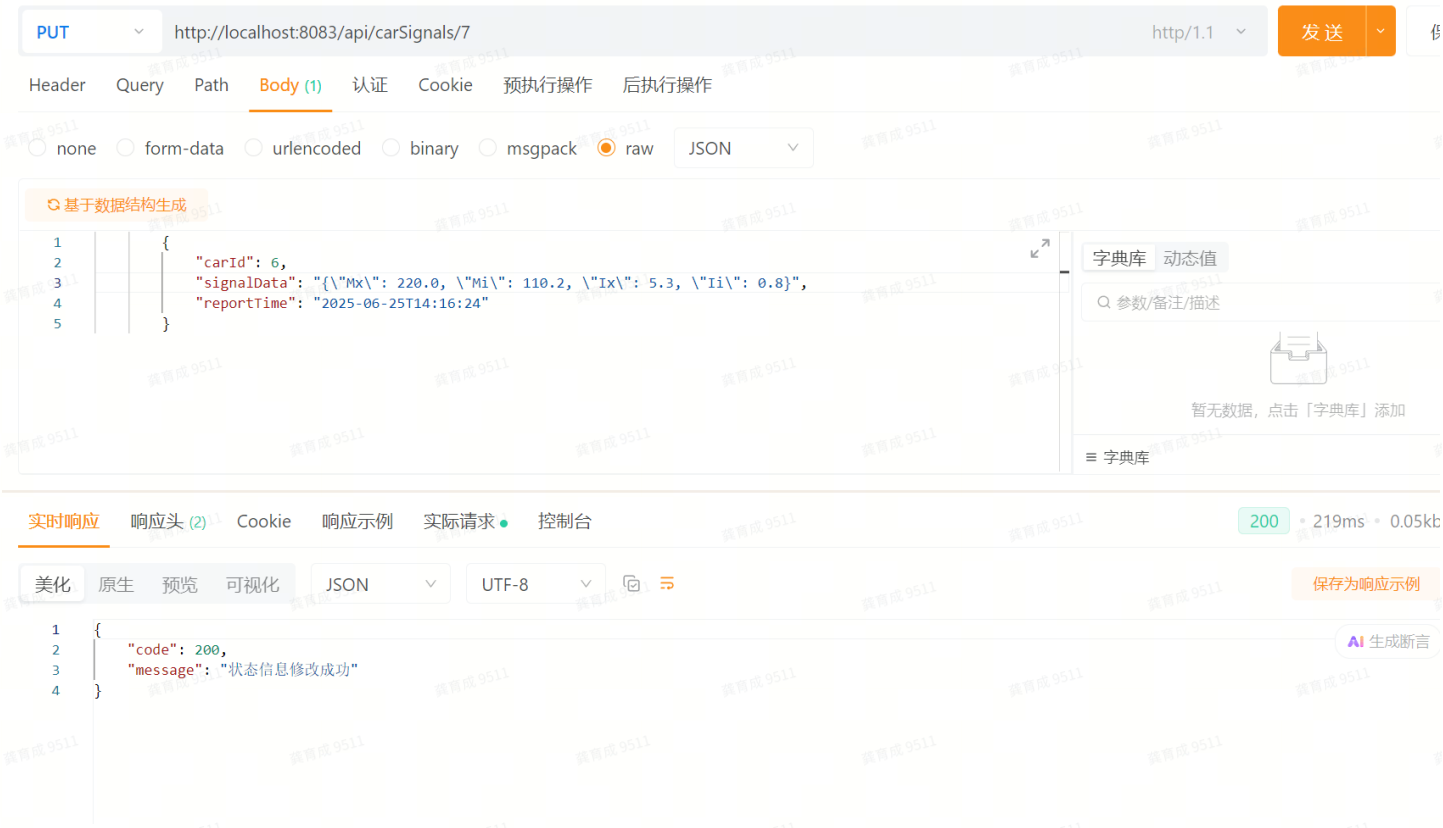
3.根据id删除车辆状态信息

延迟双删，先删除缓存，再更新数据库，等100ms删除缓存

```java
@Override
public void deleteCarSignal(Long id) {
    carSignalMapper.deleteById(id);
    // 再次删除缓存
    try (Jedis jedis = jedisClient.getJedis()) {
        jedis.del(CACHE_PREFIX + id);
    } catch (Exception e) {
        e.printStackTrace();  // 打印异常堆栈信息
    }
    try {
        Thread.sleep(100);
    } catch (InterruptedException ignored) {
```

```
13
14      }
15      // 再次删除缓存
16      try (Jedis jedis = jedisClient.getJedis()) {
17          jedis.del(CACHE_PREFIX + id);
18      } catch (Exception e) {
19          e.printStackTrace();  // 打印异常堆栈信息
20          // 或者使用日志记录，比如：logger.error("延迟删除缓存失败", e);
21      }
22
23  }
```

4.根据id修改车辆状态信息



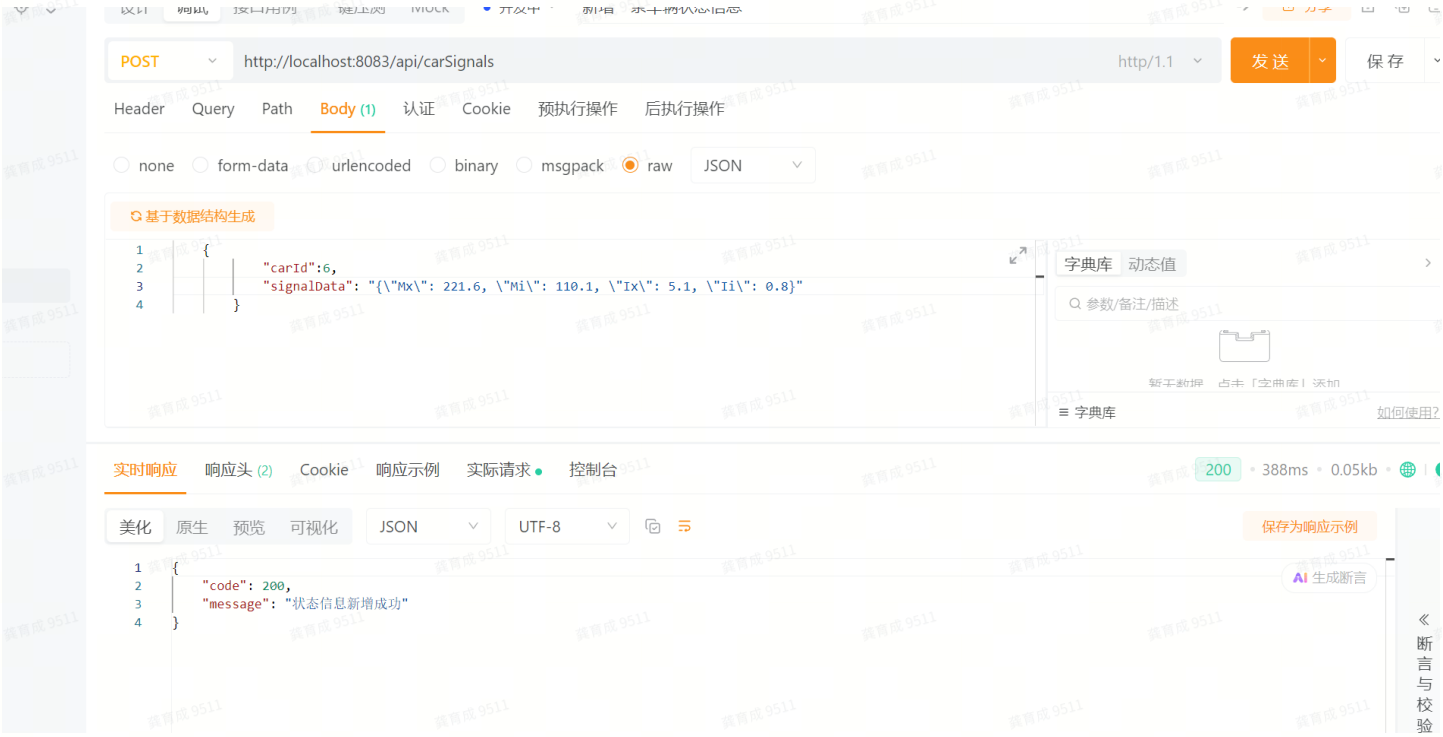
先删除缓存，等100ms，再更新数据库，

```
1  @Override
2  public void updateCarSignal(CarSignal carSignal) {
3      // 1. 更新数据库
4      carSignalMapper.update(carSignal);
5      // 2. 删除旧缓存
6      try (Jedis jedis = jedisClient.getJedis()) {
7          jedis.del(CACHE_PREFIX + carSignal.getId());  // 删除缓存
8      } catch (Exception e) {
9          // 如果删除缓存失败，记录日志
10     }
```

```
11
12      // 3. 等待缓存清理完成后再更新缓存
13      try {
14          Thread.sleep(100);  // 延迟100ms，确保缓存被清理
15      } catch (InterruptedException e) {
16          e.printStackTrace();
17      }
18      // 4. 将新数据写入缓存
19      try (Jedis jedis = jedisClient.getJedis()) {
20          String key = CACHE_PREFIX + carSignal.getId();
21          jedis.setex(key, 1800, objectMapper.writeValueAsString(carSignal));
    // 设置30分钟过期时间
22      } catch (Exception e) {
23          // 如果写入缓存失败，记录日志
24      }
25  }
```

5.单条插入车辆状态信息

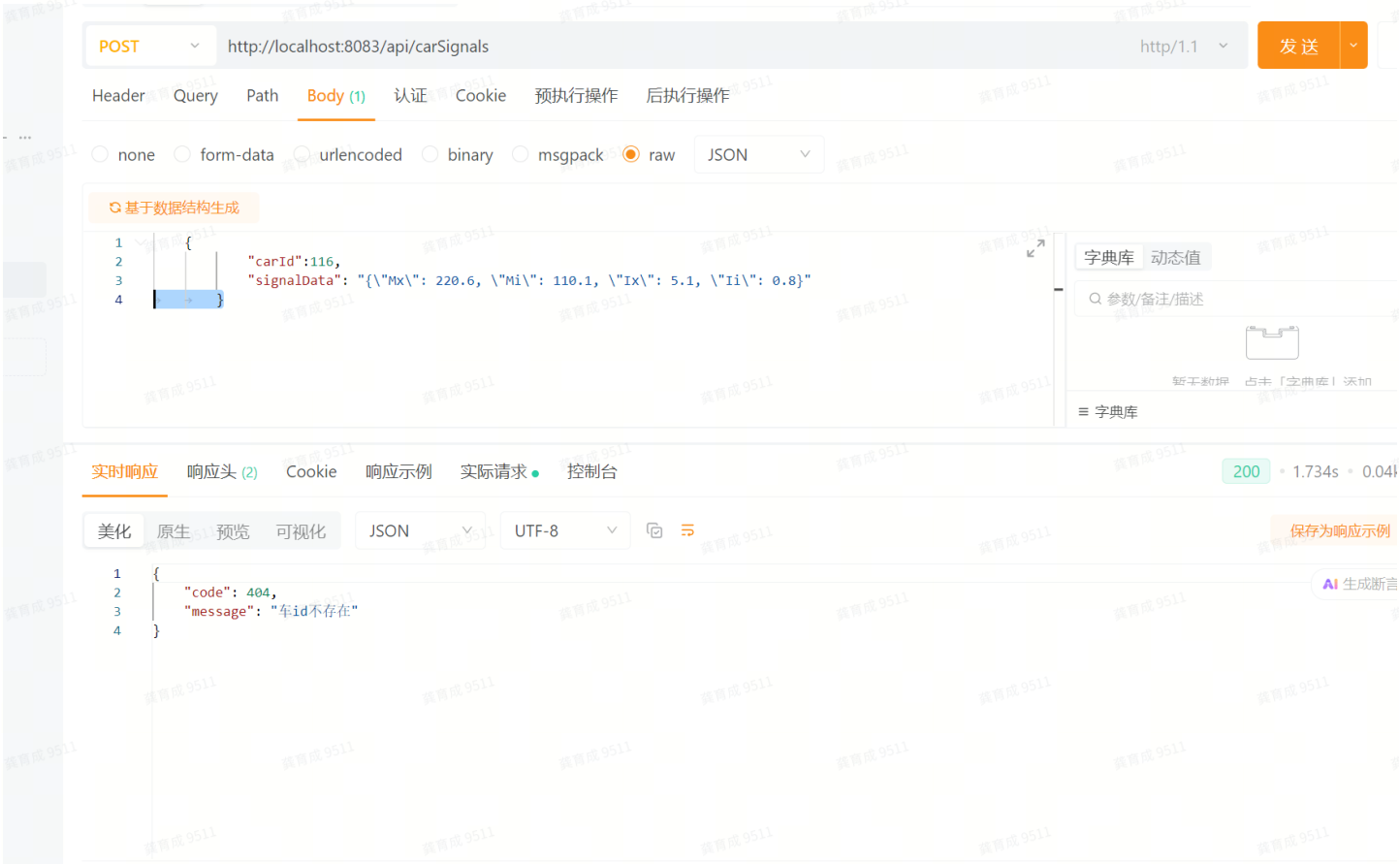
车id存在，允许插入




```
1  @Transactional
2  @Override
3  public int createCarSignal(CarSignal carSignal) {
4      // 查询车id
5      CarInfo carInfo = carInfoMapper.selectById(carSignal.getCarId());
6      // 判断车信息是否存在
```

```
    if (carInfo != null) {
        // 设置报告时间
        LocalDateTime now = LocalDateTime.now();
        carSignal.setReportTime(now);

        // 插入新的 carSignal 数据
        carSignalMapper.insert(carSignal);
        // 返回 1 表示操作成功
        return 1;
    } else {
        // 返回 0 表示车 ID 不存在
        return 0;
    }
}
```

车id不存在，无法插入车状态信息



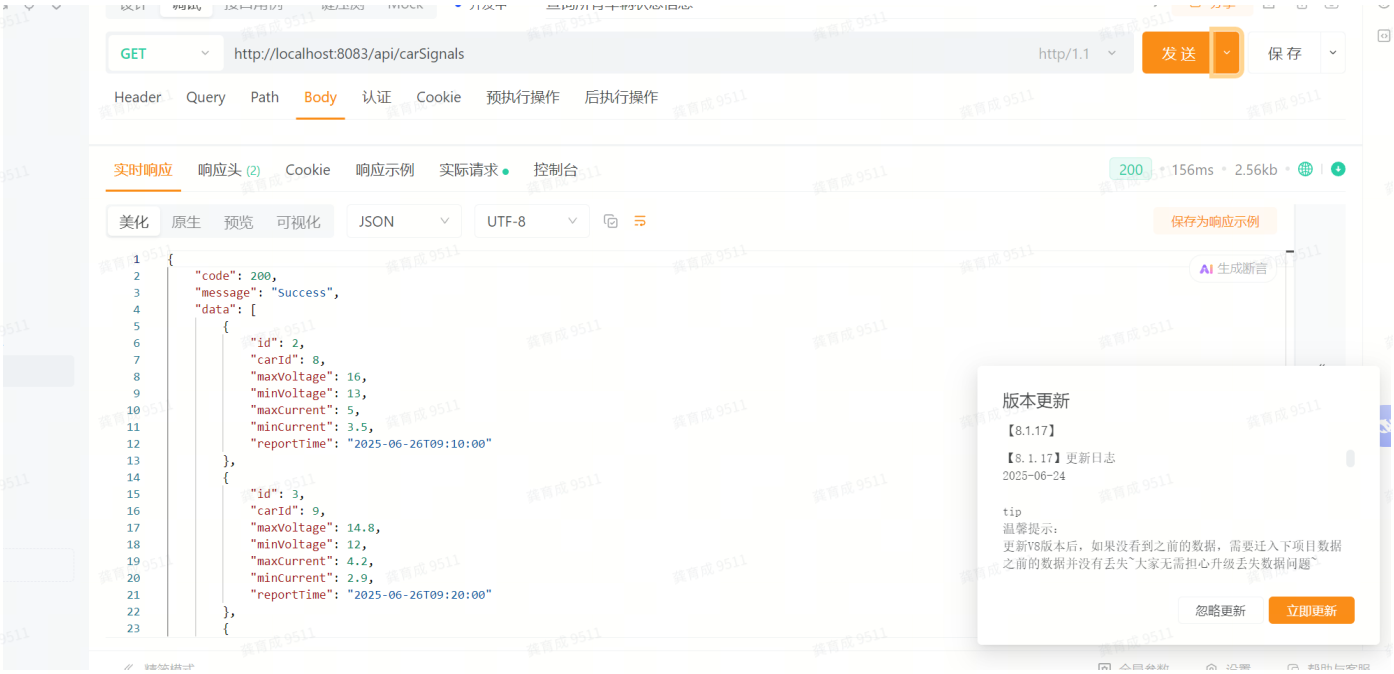
5.1.4 预警信息模块

1.生产者使用定时器每六十秒定时扫描所有车辆信息，取出signal_desc里的Mx Mi，Ix，Ii。

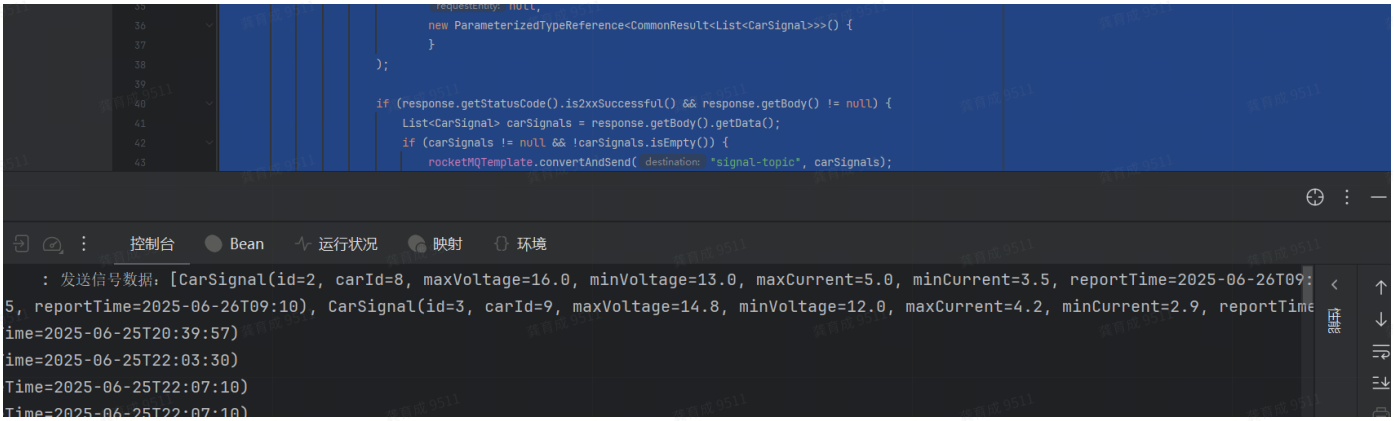
```
1
2 @Slf4j
3 @Component
```

```java
public class SignalDataScanTask {

    @Autowired
    private RocketMQTemplate rocketMQTemplate;
    @Autowired
    private RestTemplate restTemplate;
    @Autowired
    private CarSignalMapper carSignalMapper;

        @Scheduled(fixedRate = 60000)
        public void scanAndSendSignalData() {
            try {
                ResponseEntity<CommonResult<List<CarSignal>>> response = restTemplate.exchange(
                        "http://localhost:8083/api/carSignals",
                        HttpMethod.GET,
                        null,
                        new ParameterizedTypeReference<CommonResult<List<CarSignal>>>() {
                        }
                );

                if (response.getStatusCode().is2xxSuccessful() && response.getBody() != null) {
                    List<CarSignal> carSignals = response.getBody().getData();
                    if (carSignals != null && !carSignals.isEmpty()) {
                        rocketMQTemplate.convertAndSend("signal-topic", carSignals);
                        log.info("发送信号数据：{}", carSignals);
                    } else {
                        log.warn("未获取到任何车辆信号数据");
                    }
                } else {
                    log.error("请求 carSignals 接口失败：{}", response.getStatusCode());
                }
            } catch (Exception e) {
                log.error("扫描并发送信号数据时出错", e);
            }
        }
}
```

所有车辆信号信息



发送成功



消费者消费信息，生成预警信息，同步到warn_info表

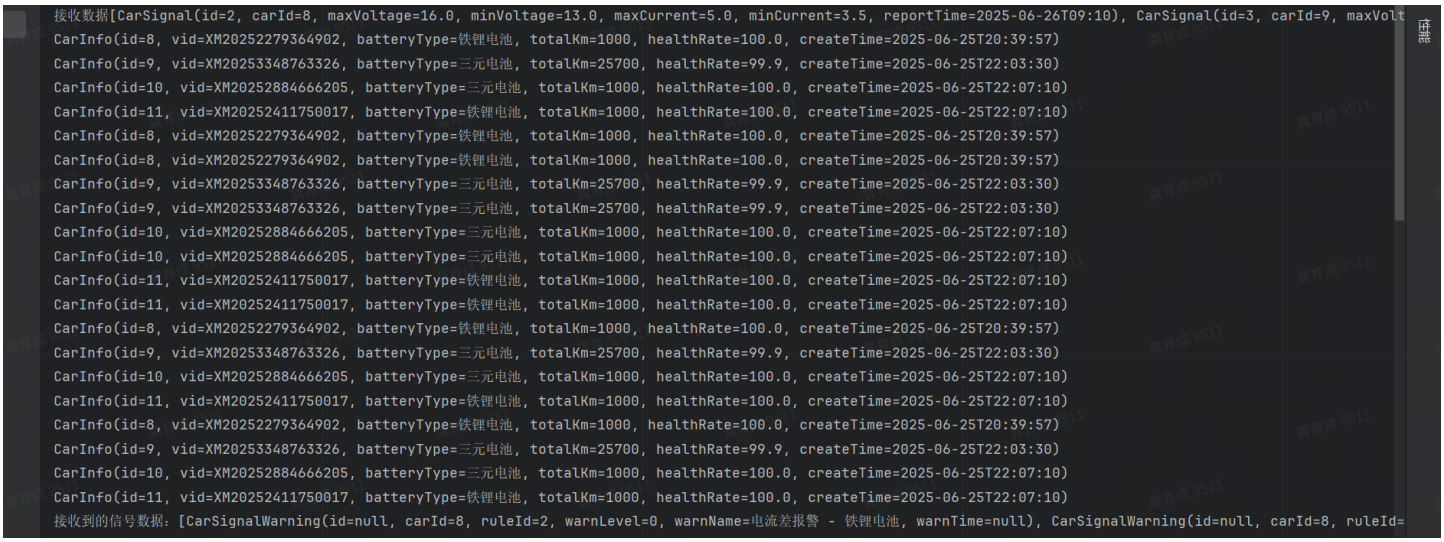
```java
package com.xm.bms.warn.MQ;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.xm.bms.carinfo.domain.CarInfo;
import com.xm.bms.carinfo.domain.CarSignal;
import com.xm.bms.carinfo.service.CarInfoService;
import com.xm.bms.common.web.CommonResult;
import com.xm.bms.warn.domain.CarSignalWarning;
import com.xm.bms.warn.domain.WarnRequest;
import com.xm.bms.warn.domain.WarnResponse;
import com.xm.bms.warn.mapper.CarSignalWarningMapper;
import com.xm.bms.warn.service.CarSignalWarningReluService;
import com.xm.bms.warn.service.CarSignalWarningService;
import com.xm.bms.warnrelu.domain.SignalData;
```

```java
import com.xm.bms.warnrelu.domain.WarningResult;
import lombok.extern.slf4j.Slf4j;
import org.apache.rocketmq.spring.annotation.RocketMQMessageListener;
import org.apache.rocketmq.spring.core.RocketMQListener;
import org.checkerframework.checker.units.qual.C;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpMethod;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.stream.Collectors;

@Slf4j
@Component
@RocketMQMessageListener(
        topic = "signal-topic",
        consumerGroup = "signal-consumer-group",
        selectorExpression = "*"
)
public class SignalDataConsumer implements RocketMQListener<List<CarSignal>> {
    @Autowired
    private RestTemplate restTemplate;

    @Autowired
    private CarSignalWarningReluService carSignalWarningReluService;
    @Autowired
    private ObjectMapper objectMapper;
    @Autowired
    private CarSignalWarningMapper carSignalWarningMapper;
    @Override
    public void onMessage(List<CarSignal> carSignals) {
        System.out.println("接收数据"+carSignals);
        List<CarSignalWarning> carSignalWarningList = new ArrayList<>();

        for (CarSignal carSignal : carSignals) {
            Long carId = carSignal.getCarId();
            CarSignalWarning carSignalWarning = new CarSignalWarning();
            SignalData signalData = new SignalData();
            signalData.setMx(carSignal.getMaxVoltage());
            signalData.setMi(carSignal.getMinVoltage());
            signalData.setIx(carSignal.getMaxCurrent());
            signalData.setIi(carSignal.getMinCurrent());
            String url = "http://localhost:8083/api/carInfos/" + carId;
```

```
63              ResponseEntity<CommonResult> response = restTemplate.exchange(
64                      url,
65                      HttpMethod.GET,
66                      null,
67                      CommonResult.class
68              );
69              Object data = Objects.requireNonNull(response.getBody()).getData();
70              CarInfo carInfo = objectMapper.convertValue(data, CarInfo.class);
71              System.out.println(carInfo);
72              if (carInfo == null) {
73                  // 处理无对应 carId 的情况（例如返回一个错误信息）
74                  carSignalWarningList.add(null);
75              } else {
76                  String batteryType = carInfo.getBatteryType();
77                  for (long ruleId = 1; ruleId <= 2; ruleId++) {
78                      if (ruleId == 1) {
79                          WarningResult warningResult =
carSignalWarningReluService.getWarningResult(ruleId, batteryType, signalData);
80                          carSignalWarning.setCarId(carId);
81
carSignalWarning.setWarnLevel(Math.toIntExact(warningResult.getWarnLevel()));
82
carSignalWarning.setWarnName(warningResult.getWarnName());
83                          carSignalWarning.setRuleId(1L);
84                          carSignalWarningList.add(carSignalWarning);
85                      }
86
87                      if (ruleId == 2) {
88                          WarningResult warningResult =
carSignalWarningReluService.getWarningResult(ruleId, batteryType, signalData);
89                          carSignalWarning.setCarId(carId);
90
carSignalWarning.setWarnLevel(Math.toIntExact(warningResult.getWarnLevel()));
91
carSignalWarning.setWarnName(warningResult.getWarnName());
92                          carSignalWarning.setRuleId(2L);
93                          carSignalWarningList.add(carSignalWarning);
94                      }
95                  }
96              }
97          }
98          // 1. 过滤 null 值
99          List<CarSignalWarning> validWarnings = carSignalWarningList
100                 .stream()
101                 .filter(Objects::nonNull)
102                 .collect(Collectors.toList());
103         System.out.println("接收到的信号数据: " + carSignalWarningList);
```

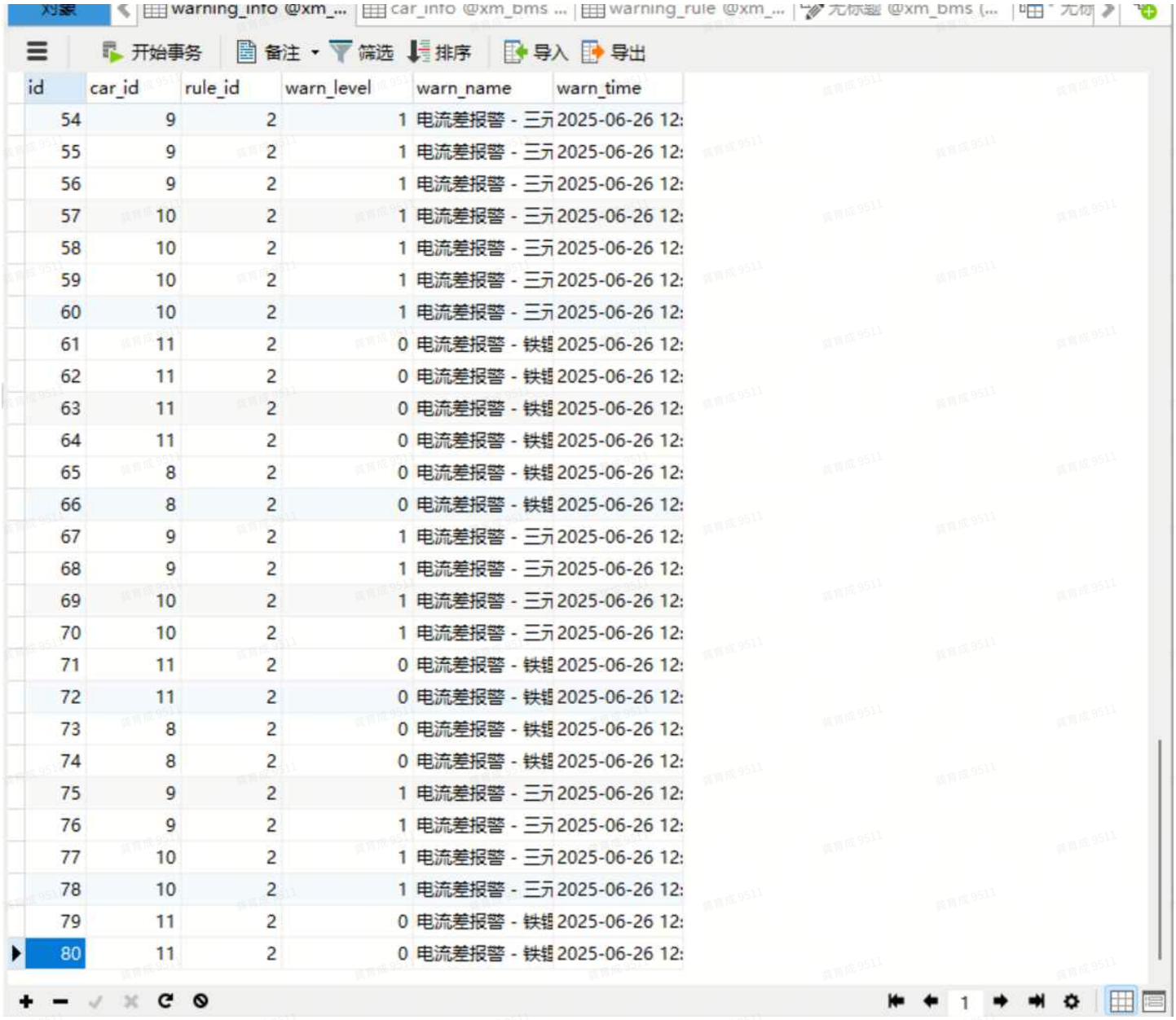
```
104             try {
105                 int result =
    carSignalWarningMapper.batchInsertOrUpdate(validWarnings);
106                 log.info("执行成功，影响记录数：{}", result);
107             } catch (Exception e) {
108                 log.error("批量插入或更新失败", e);
109             }
110
111         }
112 }
```

2.消费数据成功



同步到数据库表warn_info

同步成功



id	car_id	rule_id	warn_level	warn_name	warn_time
54	9	2	1	电流差报警 - 三元	2025-06-26 12:
55	9	2	1	电流差报警 - 三元	2025-06-26 12:
56	9	2	1	电流差报警 - 三元	2025-06-26 12:
57	10	2	1	电流差报警 - 三元	2025-06-26 12:
58	10	2	1	电流差报警 - 三元	2025-06-26 12:
59	10	2	1	电流差报警 - 三元	2025-06-26 12:
60	10	2	1	电流差报警 - 三元	2025-06-26 12:
61	11	2	0	电流差报警 - 铁锂	2025-06-26 12:
62	11	2	0	电流差报警 - 铁锂	2025-06-26 12:
63	11	2	0	电流差报警 - 铁锂	2025-06-26 12:
64	11	2	0	电流差报警 - 铁锂	2025-06-26 12:
65	8	2	0	电流差报警 - 铁锂	2025-06-26 12:
66	8	2	0	电流差报警 - 铁锂	2025-06-26 12:
67	9	2	1	电流差报警 - 三元	2025-06-26 12:
68	9	2	1	电流差报警 - 三元	2025-06-26 12:
69	10	2	1	电流差报警 - 三元	2025-06-26 12:
70	10	2	1	电流差报警 - 三元	2025-06-26 12:
71	11	2	0	电流差报警 - 铁锂	2025-06-26 12:
72	11	2	0	电流差报警 - 铁锂	2025-06-26 12:
73	8	2	0	电流差报警 - 铁锂	2025-06-26 12:
74	8	2	0	电流差报警 - 铁锂	2025-06-26 12:
75	9	2	1	电流差报警 - 三元	2025-06-26 12:
76	9	2	1	电流差报警 - 三元	2025-06-26 12:
77	10	2	1	电流差报警 - 三元	2025-06-26 12:
78	10	2	1	电流差报警 - 三元	2025-06-26 12:
79	11	2	0	电流差报警 - 铁锂	2025-06-26 12:
80	11	2	0	电流差报警 - 铁锂	2025-06-26 12:

3.先远程调用得到所有车辆状态信息。根据规则查询，先在carInfo模块查找是否有车辆信息（RPC），再根据有无规则id生成预警信息，并同步到数据库表warn_info

无预警id

有预警id


```java
public List<WarnResponse> processWarning(List<WarnRequest> warnData) {
    List<WarnResponse> carSignalWarningList = new ArrayList<>();

    for (WarnRequest warnRequest : warnData) {
        Long carId = warnRequest.getCarId();
        Long warnId = warnRequest.getWarnId();
        String signal = warnRequest.getSignal();
        SignalData signalData = new SignalData();
        WarnResponse carSignalWarning = new WarnResponse();
        try {
```

```
            JsonNode rootNode = objectMapper.readTree(signal);
            // 手动将 JsonNode 中的字段值提取出来并设置到 SignalData 对象
            signalData.setMx(rootNode.get("Mx").asDouble());
            signalData.setMi(rootNode.get("Mi").asDouble());
            signalData.setIx(rootNode.get("Ix").asDouble());
            signalData.setIi(rootNode.get("Ii").asDouble());
            System.out.println(signalData);
        } catch (Exception e) {
            e.printStackTrace();
            // 错误处理：无法解析信号数据
        }

        String url = "http://localhost:8083/api/carInfos/" + carId;
        ResponseEntity<CarInfoResponse> response = restTemplate.exchange(
                url,
                HttpMethod.GET,
                null,
                CarInfoResponse.class // We expect the response to be wrapped in CarInfoResponse
        );

        // Get the data (CarInfo object) from the response
        CarInfo carInfo = response.getBody().getData();
        System.out.println(carInfo);
        if (carInfo == null) {
            // 处理无对应 carId 的情况（例如返回一个错误信息）
            carSignalWarningList.add(null);
        } else {
            String batteryType = carInfo.getBatteryType();
            if (warnId != null) {
                // 根据指定的 warnId 计算报警等级
                WarningResult warningResult = carSignalWarningReluService.getWarningResult(warnId, batteryType, signalData);
                carSignalWarning.setCarId(carId);
                carSignalWarning.setWarnLevel(warningResult.getWarnLevel());
                carSignalWarning.setWarnName(warningResult.getWarnName());
                carSignalWarning.setRuleId(warnId);
                carSignalWarningList.add(carSignalWarning);
            } else {
                for (long ruleId = 1; ruleId <= 2; ruleId++) {
                    if (ruleId == 1) {
                        WarningResult warningResult = carSignalWarningReluService.getWarningResult(ruleId, batteryType, signalData);
                        carSignalWarning.setCarId(carId);

carSignalWarning.setWarnLevel(warningResult.getWarnLevel());
```

```
carSignalWarning.setWarnName(warningResult.getWarnName());
                        carSignalWarning.setRuleId(1L);
                        carSignalWarningList.add(carSignalWarning);
                    }

                    if (ruleId == 2) {
                        WarningResult warningResult =
carSignalWarningReluService.getWarningResult(ruleId, batteryType, signalData);
                        carSignalWarning.setCarId(carId);

carSignalWarning.setWarnLevel(warningResult.getWarnLevel());

carSignalWarning.setWarnName(warningResult.getWarnName());
                        carSignalWarning.setRuleId(2L);
                        carSignalWarningList.add(carSignalWarning);
                    }


                }

            }
        }
    }
    return carSignalWarningList;
}
```

4.规则逻辑可先根据电池类型和reluNo去数据查找对应描述，返回一个list用 case进行判断。

```
@Override
public WarningResult getWarningResult(Long reluNo, String batteryType,
SignalData signalData) {
    List<BatteryWarningRule> rules =
judgeByRulesMapper.selectByRuleNoAndBatteryType(reluNo, batteryType);
    if (rules == null || rules.isEmpty()) {
        return new WarningResult(-1L, "无规则");
    } else {
        case1:
        case2:
        case3:
        .....
    }
}
```

6. 维护

7. 思考

- 系统每天处理信号量为百万甚至千万数据级别：考虑数据量对系统性能的影响，给出合理设计数据存储和查询方案。

- 定时扫描 上传车辆状态信息，同步到数据库，根据车id和ruleId唯一标识车辆预警信息，设为key存入Redis。用户主动去查询返回缓存，缓存失效，数据库再去兜底。（车辆预警信息应该是更新的很快的）