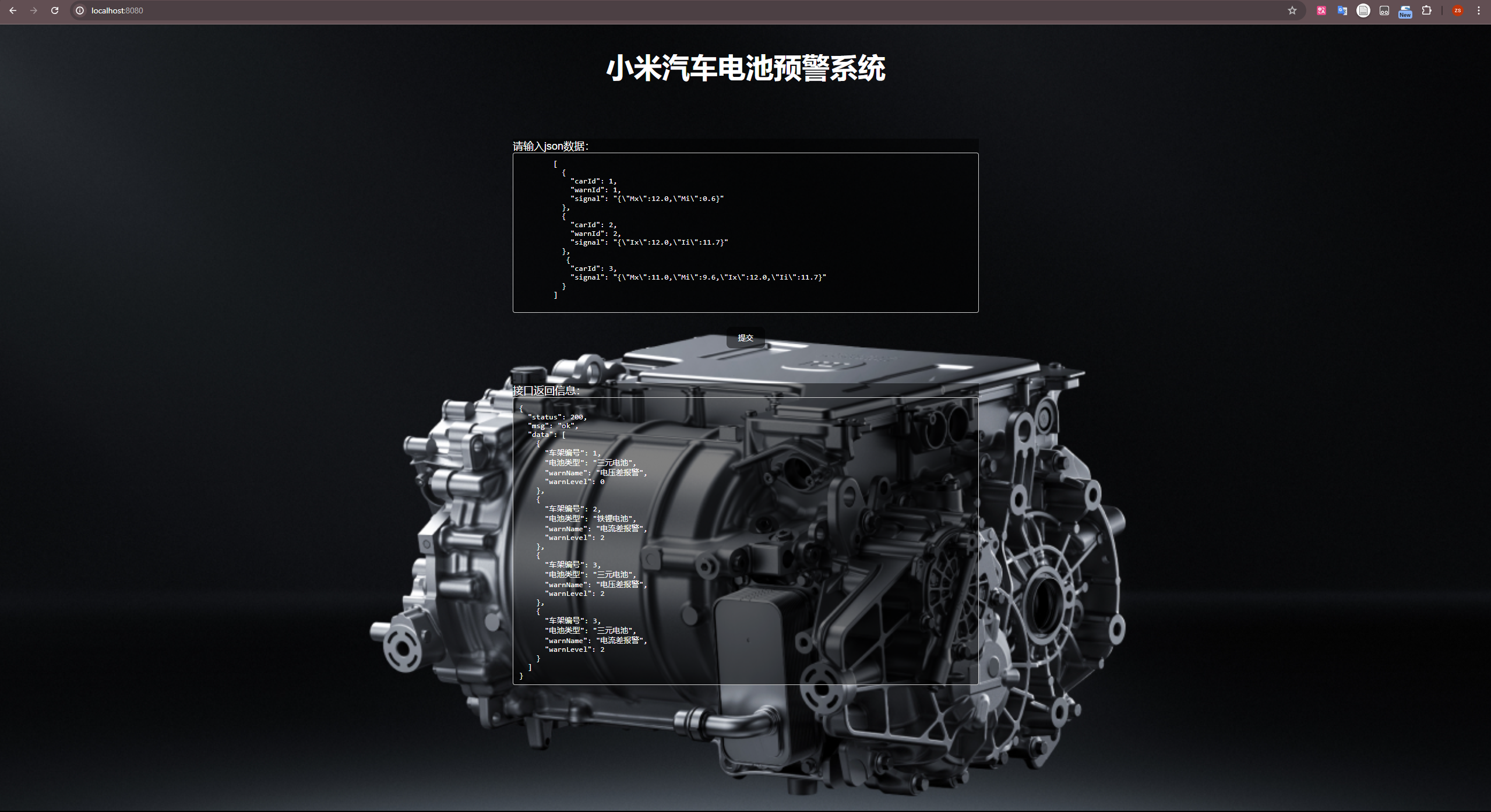
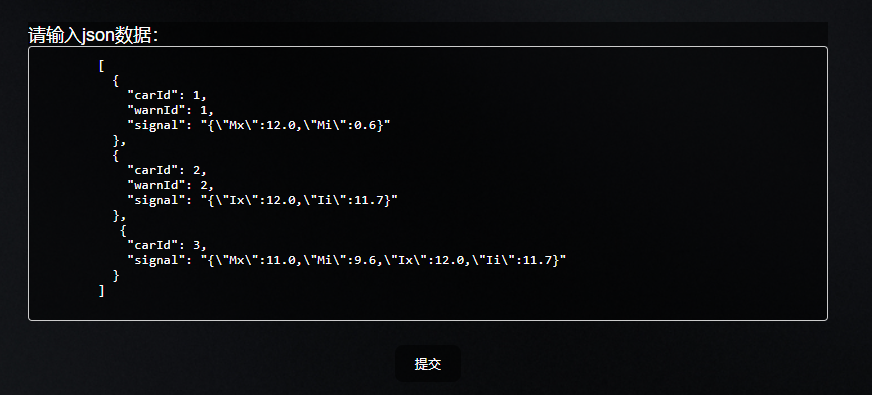
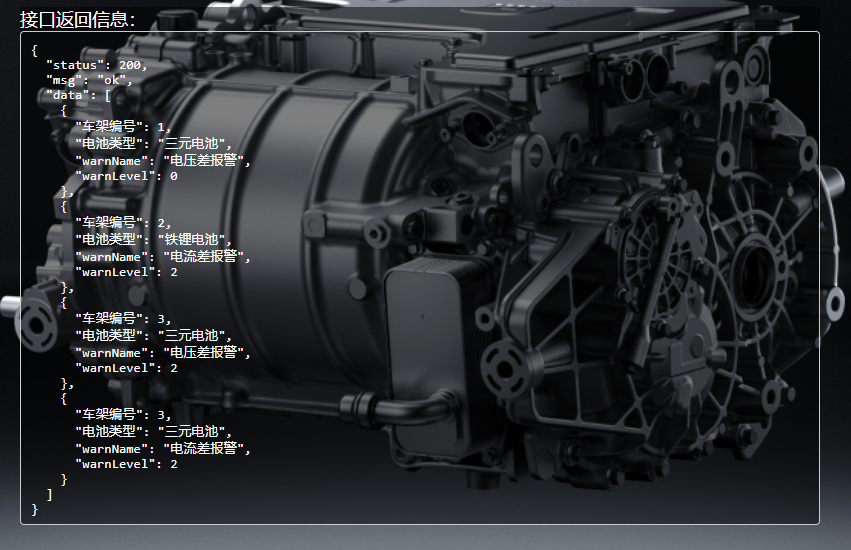
**README.zh-CN**

@author techotaku@zs

@[GitHub仓库地址](https://github.com/numbbbbbplus/XiaomiEV-Battery-Warning-System)



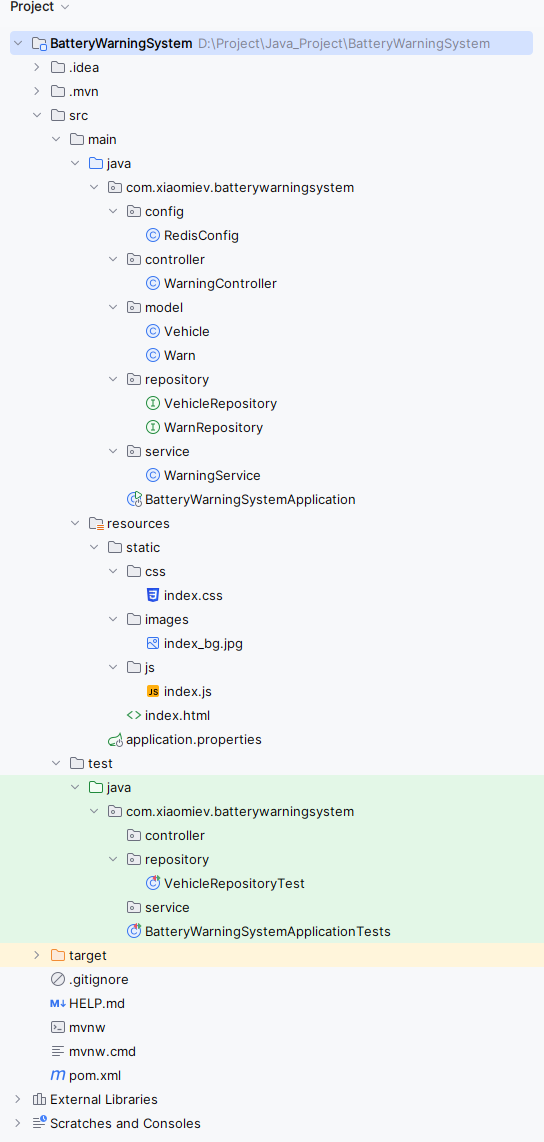




**1 考核标准**

**1.1 完成技术方案设计，技术方案至少包含系统设计、数据库表设计、接口设计、单元测试**

**系统设计**



项目结构

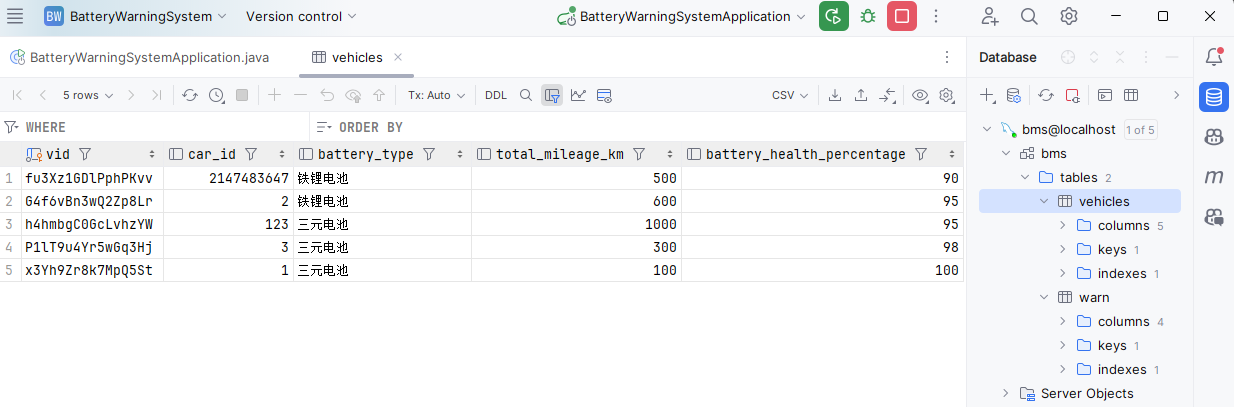
项目结构说明：

该项目的名称为 BatteryWarningSystem，基于 Spring Boot 框架构建，包含以下主要模块：

* config：包含 Redis 配置类 RedisConfig。
* controller：包含控制器类 WarningController，负责处理预警请求。
* model：包含实体类 Vehicle 和 Warn，用于定义数据库表结构。
* repository：包含数据访问接口 VehicleRepository 和 WarnRepository，用于与数据库交互。
* service：包含服务类 WarningService，实现业务逻辑。
* resources：包含静态资源文件（CSS、JS、图片）和配置文件 application.properties。
* test：包含单元测试类 VehicleRepositoryTest 和 BatteryWarningSystemApplicationTests。

该项目通过 Maven 构建管理，依赖 Spring Boot、MySQL 和 Redis，支持前端页面交互。

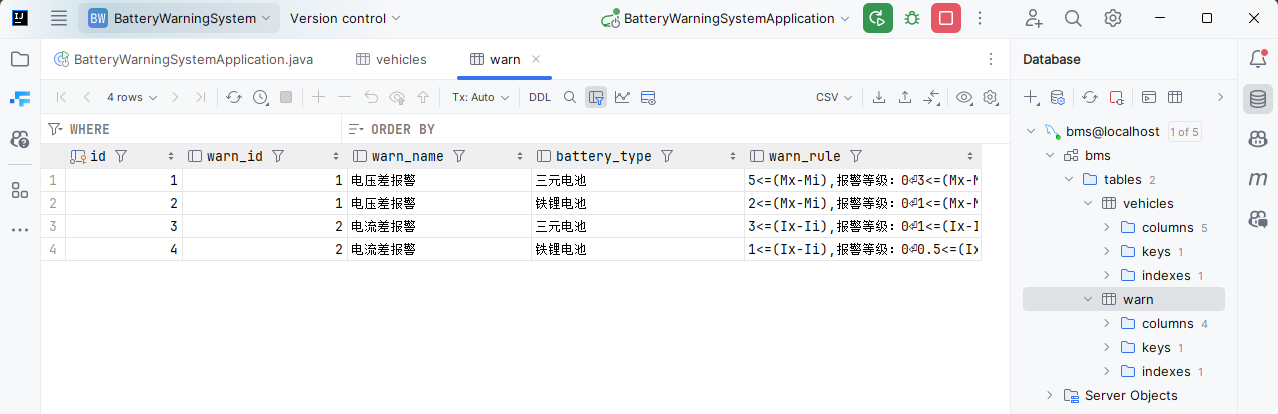
**数据库表设计**



vehicles表

vehicles表用于存储车辆的基本信息，每条记录表示一辆车辆的详细信息。

* vid (字符串，主键): 车辆唯一标识符，16位随机字符串。
* car\_id (整数): 车辆编号。
* battery\_type (字符串): 电池类型，如“三元电池”或“铁锂电池”。
* total\_mileage\_km (整数): 车辆的总行驶里程，以公里为单位。
* battery\_health\_percentage (整数): 电池健康百分比。



warn表

warn表用于存储预警规则信息，每条记录表示一种电池类型对应的一种预警规则。

* id (整数，主键): 规则唯一标识符。
* warn\_id (整数): 预警规则编号。
* warn\_name (字符串): 预警名称。
* battery\_type (字符串): 电池类型。
* warn\_rule (字符串): 预警规则，描述如何根据信号计算预警等级。

**接口设计**



WarnController.java - @RestController @RequestMapping("/api") @PostMapping("/warn")

在WarnController.java中，使用@RestController注解标记该类为控制器，并通过@RequestMapping("/api")定义根路径。在该类中，@PostMapping("/warn")定义了处理预警请求的POST接口。通过该接口，系统可以接收JSON格式的预警请求，并调用WarningService进行处理，最后返回处理结果。



WarnController.java - Class WarningRequest

WarningRequest是WarnController的一个内部静态类，用于封装客户端发送的预警请求数据。该类包含三个字段：carId、warnId和signal，其中carId表示车辆ID，warnId表示预警ID，signal表示信号数据。这个类的实例通过@RequestBody注解接收JSON格式的请求数据。



WarnController.java - Class WarningResponse

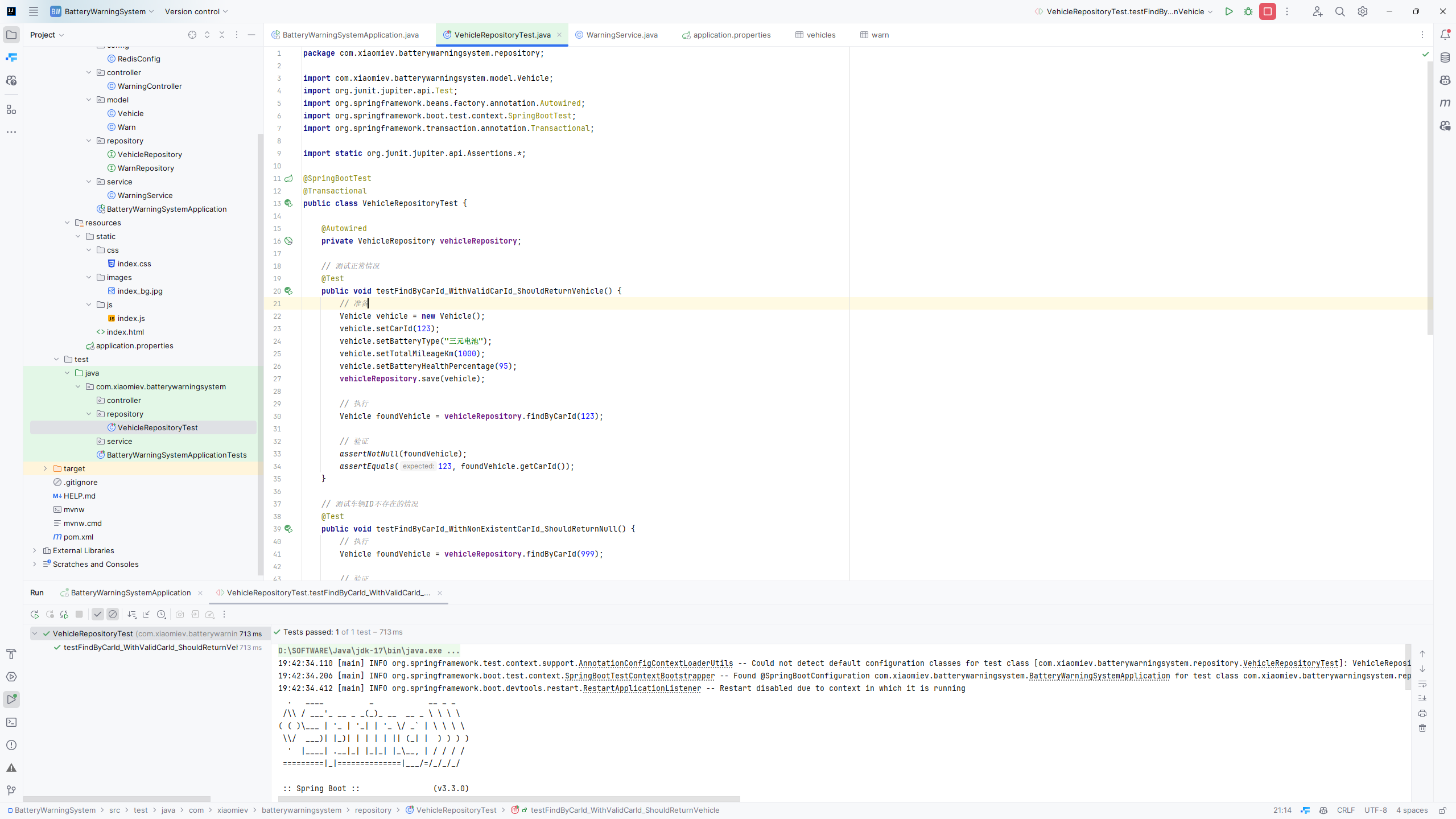
WarningResponse是WarnController的一个内部静态类，用于封装服务端返回的响应数据。该类包含三个字段：status、msg和data。status表示响应状态码，msg表示响应消息，data是一个包含WarningResult实例的列表，用于存储处理后的预警结果。通过这种方式，可以将处理结果以统一的格式返回给客户端。



index.js

在index.js中，使用了fetch函数向/api/warn发送POST请求。首先将输入的预警请求数据转换为JSON格式，然后通过fetch函数发送请求。在收到响应后，then方法处理响应数据，将结果显示在页面上。如果请求失败，则在catch方法中处理错误并显示错误信息。这段代码实现了与后端API的交互，将用户输入的预警请求发送给后端，并将响应结果展示给用户。

**单元测试**



VehicleRepositoryTest.java 单元测试

VehicleRepositoryTest.java 单元测试说明:

该测试类验证了 VehicleRepository 接口的正确性，通过多种查询场景，确保车辆信息的存取操作符合预期，包括有效、无效、最小和最大 carId 的处理。

**1.2 完成项目搭建，能本地运行，调试，并且提交代码到统一github仓库**

**1.2.1 本地运行录 Demo.mkv**

**[Demo.mkv]**

Demo.mkv 14s录屏

**1.2.2 GitHub仓库地址**

https://github.com/numbbbbbplus/XiaomiEV-Battery-Warning-System

**1.3 实现功能流程，截图保留上述作业所有功能实现点**

**技术点**

1. **规则解析不是写成固定在代码里面，而是根据规则编号获取预警规则然后解析**



WarningService.java private int evaluateWarningLevel(String signal, String warnRule, int warnId)

evaluateWarningLevel 方法实现逻辑：

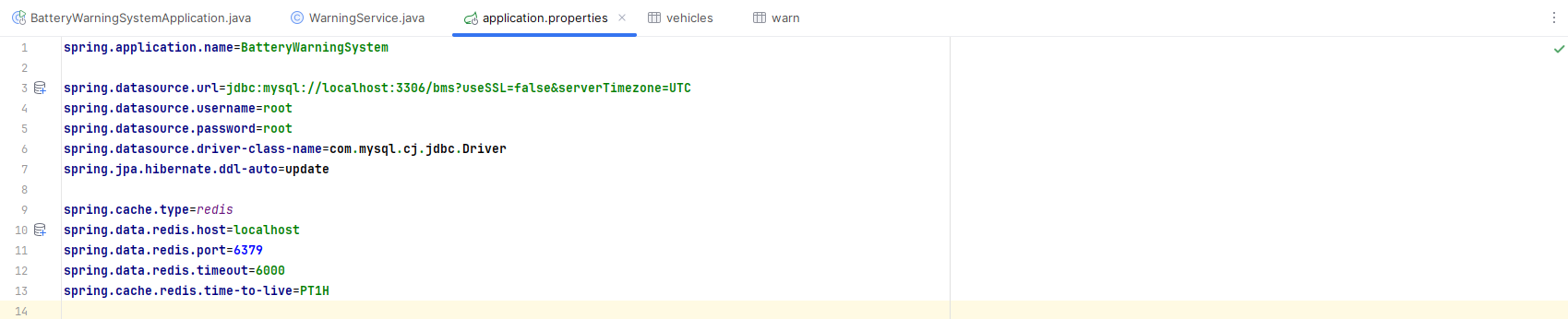
1. 解析信号数据：使用 objectMapper 将输入的 JSON 格式 signal 字符串解析为一个 Map，从中提取 Mx、Mi、Ix 和 Ii 的值。
2. 计算差值：
3. 如果 warnId 是 1，则计算电压差值 difference = Mx - Mi。
4. 如果 warnId 是 2，则计算电流差值 difference = Ix - Ii。
5. 从数据库获取预警规则：根据 warnId 和电池类型从数据库中获取对应的 warn\_rule 字符串。
6. 解析和替换规则：
7. 使用换行符分割 warn\_rule，得到每条规则。
8. 替换规则中的 (Mx-Mi) 或 (Ix-Ii) 为计算得到的 difference 值。
9. 评估条件表达式：
10. 对每条规则，提取条件表达式和报警等级。
11. 通过 evaluateCondition 方法，评估替换后的条件表达式是否成立。
12. 如果条件成立，返回对应的报警等级。
13. 默认返回值：如果所有条件都不满足，则返回默认值 -1，表示不报警。

该实现通过从数据库获取预警规则，而不是将规则写死在代码中，实现了规则的动态解析和灵活性。

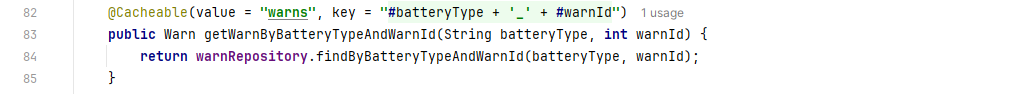
2. **单元测试中：信号响应的时间级别，规则成功预警率（不低于90%）**

3. **信号通过预警规则计算时候，实时规则的接口性能测试和优化，P99 响应时间在 1s以内**

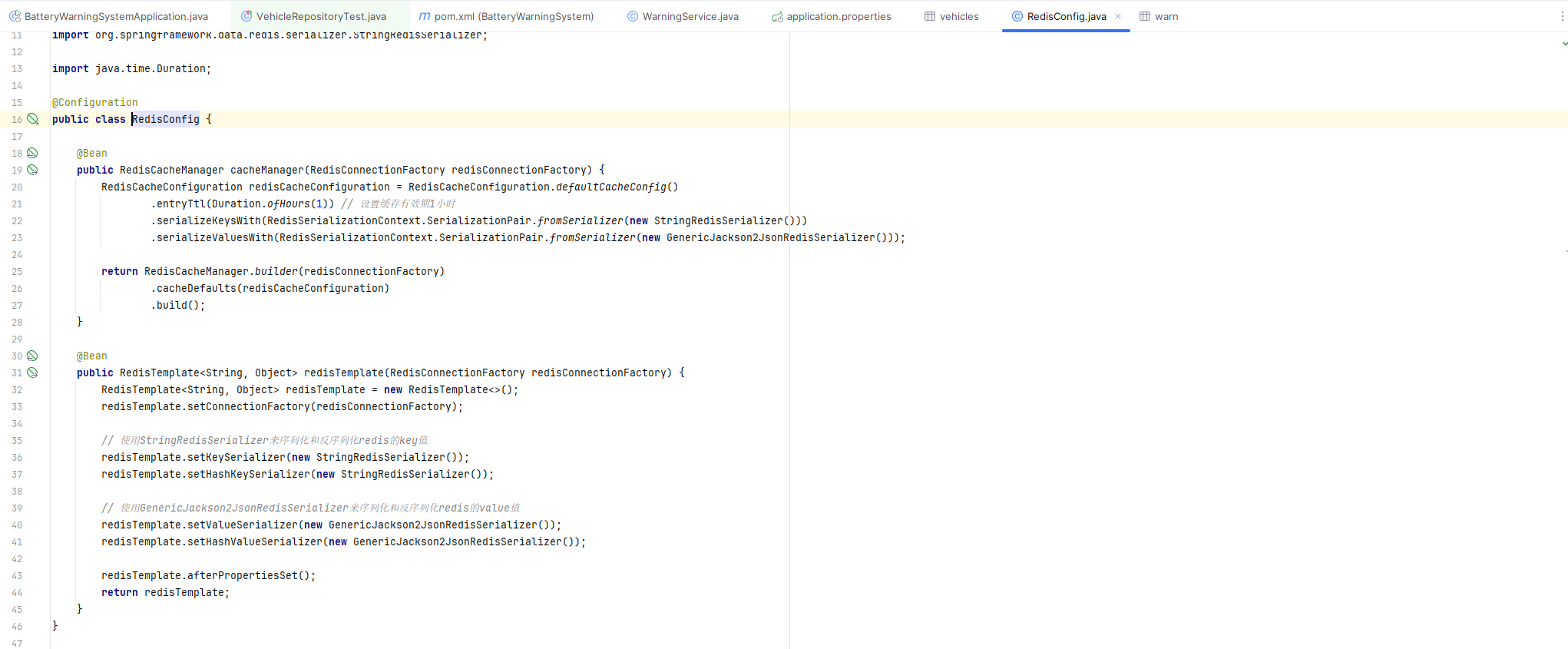
4. **系统每天处理信号量为百万甚至千万数据级别：考虑数据量对系统性能的影响，给出合理设计数据存储和查询方案**



application.properties



WarningService @Cacheable



RedisConfig.java

在电池预警系统中，通过 Redis 提升性能和可扩展性。配置文件 application.properties 定义了 Redis 主机、端口和缓存超时时间。RedisConfig 类设置了 RedisCacheManager 和 RedisTemplate，并使用 StringRedisSerializer 和 GenericJackson2JsonRedisSerializer 进行键值序列化。BatteryWarningSystemApplication 类通过 @EnableCaching 启用缓存功能。WarningService 类中使用 @Cacheable 注解缓存查询结果。这种缓存机制提高了系统响应速度，确保预警规则成功率，并高效处理大规模数据。

**2 本地运行**

**2.1 必要的软件**

* Java Development Kit (JDK) 17 或更新版本
* Apache Maven
* MySQL Server
* Redis

**2.2 导入数据库bms**

连接MySQL，Navicat图形化界面/命令行 运行 bms.sql，导入数据库bms

**[bms.sql]**

**2.3 修改application.properties配置文件**

**2.3.1 解压 JAR 文件：**

* 右键点击 BatteryWarningSystem-0.0.1-SNAPSHOT.jar。

**[BatteryWarningSystem-0.0.1-SNAPSHOT.jar]**

* 选择 7-Zip -> Extract to "BatteryWarningSystem-0.0.1-SNAPSHOT\"。

**2.3.2 修改 application.properties 文件：**

* 导航到解压后的目录BatteryWarningSystem-0.0.1-SNAPSHOT\BOOT-INF\classes\ application.properties。
* 打开并编辑 application.properties 文件(连接到本地MySQL和redis)，保存修改。

|  |
| --- |
| Plain Text spring.application.name=BatteryWarningSystem  spring.datasource.url=jdbc:mysql://localhost:3306/bms?useSSL=false&serverTimezone=UTC spring.datasource.username=root // your mysql username spring.datasource.password=root // your mysql password spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver spring.jpa.hibernate.ddl-auto=update  spring.cache.type=redis spring.data.redis.host=localhost spring.data.redis.port=6379 spring.data.redis.timeout=6000 spring.cache.redis.time-to-live=PT1H |

application.properties

**2.4 重新打包 JAR 文件：**

* 打开 7-Zip 文件管理器。
* 导航到 ..\BatteryWarningSystem-0.0.1-SNAPSHOT\。
* 按 Ctrl+A 选择所有文件和文件夹。
* 点击 添加（Add）。
* 在 添加到归档文件（Add to Archive）窗口中，设置归档格式为 zip，并将文件扩展名改为 .jar，如 BatteryWarningSystem-0.0.1-SNAPSHOT.jar。
* 点击 确定。

**2.5 本地运行打包好的jar包**

运行以下命令行：

|  |
| --- |
| Plain Text java -jar BatteryWarningSystem-0.0.1-SNAPSHOT.jar |