# 技术方案

## 一：系统设计

本项目构建于Spring Boot框架之上，旨在创建一个高效、可扩展的小米汽车电池预警系统。系统核心目标是接收、处理车辆的电池信号数据，并根据预设的规则生成并管理预警信息。系统设计遵循典型的分层架构模式，以提高模块化、可维护性和可测试性。

主要组成部分和分层说明：

#### 1.Controller 层 (org.xiaomi.controller)：

作为系统的入口点，负责接收来自外部客户端（如前端应用或其他服务）的 HTTP 请求。

处理请求参数的解析和验证。

调用 Service 层相应的业务方法来执行具体操作。

将Service 层返回的结果封装成统一的响应格式（例如使用 org.xiaomi.entity.Result 类），返回给客户端。

例如，VehicleInfoController 负责处理车辆信息的 RESTful API 请求。

#### 2.Service 层 (org.xiaomi.service 和 org.xiaomi.service.impl)：

包含系统的核心业务逻辑。

接口 (org.xiaomi.service): 定义了业务功能的契约，例如 VehicleInfoService 和 WarnInfoService。

实现类 (org.xiaomi.service.impl): 实现了 Service 接口中定义的业务逻辑，协调 Mapper、Redis、MQ 等组件完成复杂的业务流程。

负责事务管理、缓存处理、规则判断等。

例如，WarnInfoServiceImpl 包含了根据信号规则判断是否生成预警、调用 Mapper 存储预警、以及调用 Redis 清除缓存的逻辑。

#### 3.Mapper 层 (org.xiaomi.mapper)：

负责与数据库进行直接交互，使用 MyBatis 持久层框架。

定义数据库操作方法（如插入、查询、更新、删除）。

将数据库查询结果映射到 Entity 对象。

本项目使用 MyBatis，VehicleInfoMapper 和 WarnInfoMapper 定义了对车辆和预警信息的数据库操作。

#### 4.Entity 层 (org.xiaomi.entity)：

定义了系统中数据的领域模型，通常与数据库表结构一一对应。

使用 Lombok 简化 JavaBean 代码（如 getter, setter, constructors）。

例如，VehicleInfo、WarnInfo、SignalRule 分别代表车辆、预警和规则的数据结构。

#### 5.DTO/VO 层 (org.xiaomi.dto 和 org.xiaomi.dto)：

DTO (Data Transfer Object): 数据传输对象，用于在不同层之间传递数据，例如 WarnInfoDTO 用于接收外部输入的预警信号数据。

VO (View Object): 视图对象，用于封装返回给客户端的数据，通常是经过处理或聚合后的数据，例如 WarnInfoVO 用于表示返回给前端的预警信息列表项。

#### 6.Utils 层 (org.xiaomi.utils)：

存放通用的工具类和方法，与具体业务逻辑无关。

例如，SnowflakeIdGenerator 用于生成全局唯一的 ID。

#### 7.MQ Consumer (org.xiaomi.mq.WarnMsgConsumer)：

负责监听和消费来自 RocketMQ 特定主题的消息。

接收到消息后，解析消息内容（例如车辆信号数据）。

模拟电池信号生成，发送http请求。

#### 7.Config 层 (org.xiaomi.config)：

包含 Spring Boot 的配置类，用于配置应用程序的各个方面。

例如，RedisConfig 配置 Redis 连接和序列化器，RestTemplateConfig 配置 RestTemplate bean。

集成组件：

Spring Boot：简化了 Spring 应用程序的开发和部署，提供了自动配置、内嵌 Web 服务器等功能。

MyBatis：持久层框架，简化了数据库操作。

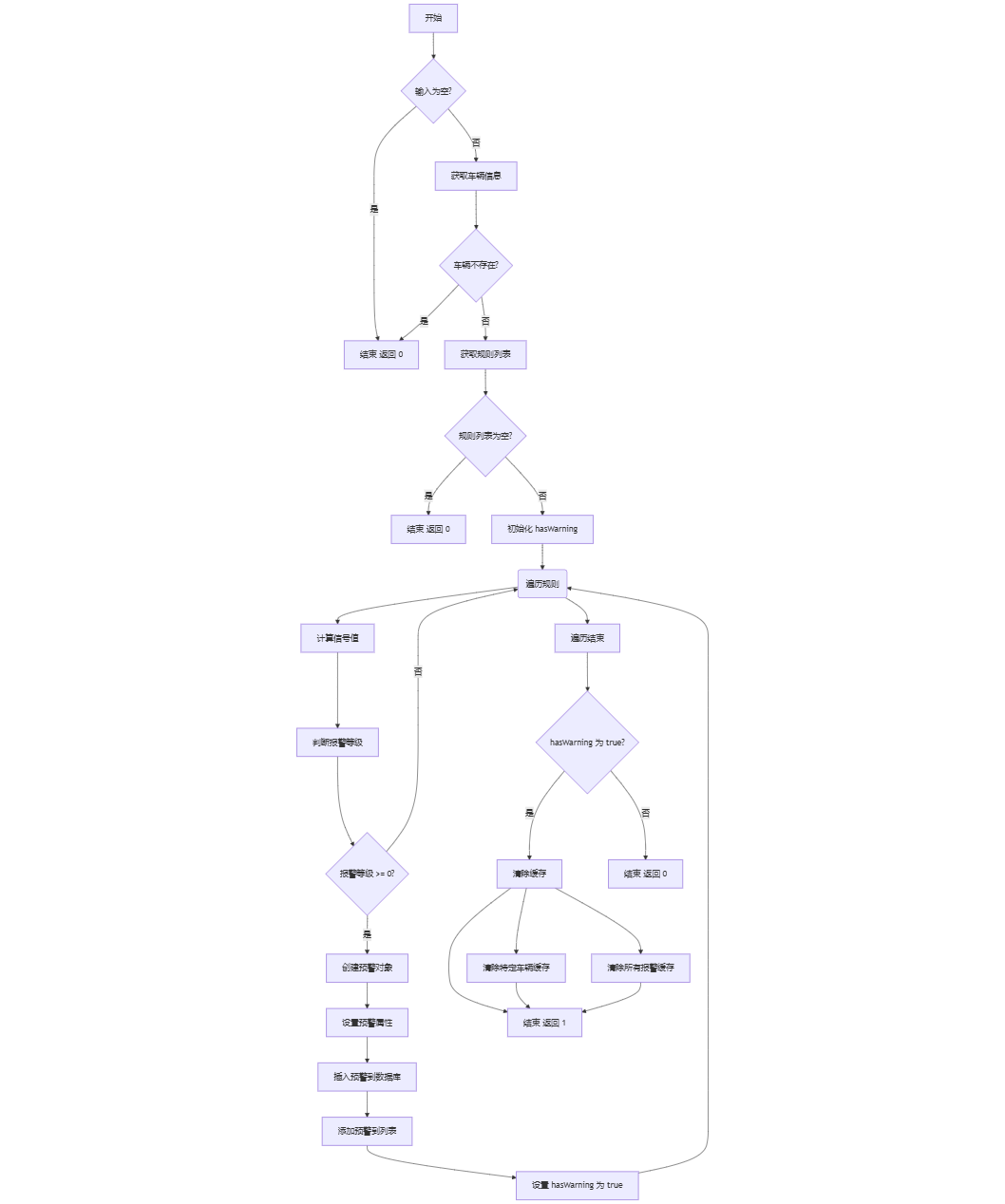
Redis：内存数据结构存储，本项目用作缓存，提高数据查询速度。

RocketMQ：使用定时任务配合，模拟生成电池信号。

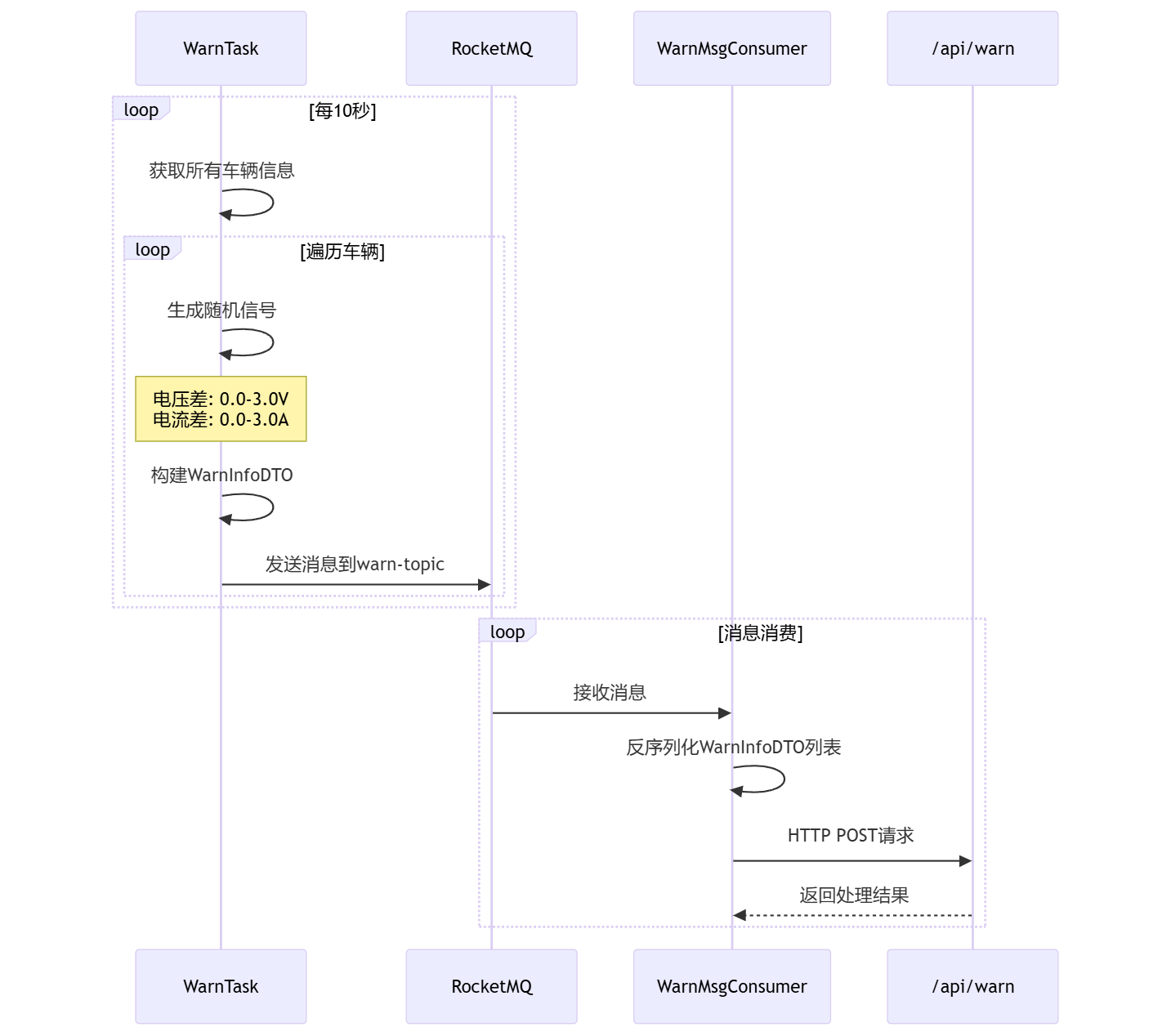
这种分层设计使得系统结构清晰，各层职责明确，便于开发、测试和维护。Service 层作为业务核心，不直接依赖 Controller 或 MQ Consumer，它们通过接口与 Service 层交互。

##### 核心功能流程图：

1.添加报警信息



2.模拟生成车辆预警信号处理流程图



## 二.数据库设计

#### vehicle\_info (车辆信息表):

存储注册到系统中的车辆的基本信息。

vid: VARCHAR(255) - 车辆的唯一标识符 (Vehicle ID) NOT NULL且UNIQUE，确保每辆车有一个唯一的字符串标识。

car\_id: INT -主键，AUTO\_INCREMENT，自动递增的整数ID，作为车辆记录的主要标识。

battery\_type: VARCHAR(255)-车辆使用的电池类型

total\_mileage: DECIMAL(10, 2)-车辆的总行驶里程，总共最多10位数字，小数点后保留两位。

battery\_health\_percentage: DECIMAL(5, 2)-电池的健康度百分比，总共最多5位数字，小数点后保留两位。

#### signal\_rule (信号规则表):

存储不同电池类型和预警名称对应的预警触发规则。

id: INT -主键，AUTO\_INCREMENT，唯一标识规则记录。

warn\_id: INT-规则相关的预警ID。

warn\_name: VARCHAR(255)-预警的名称（如“电压差报警”、“电流差报警”）。

battery\_type: VARCHAR(255) - 此规则适用的电池类型。

warning\_rule: TEXT-存储预警判断的具体规则字符串。使用 TEXT 类型允许存储任意长度的规则文本。

#### warn\_info (预警信息表):

存储系统根据规则判断生成的电池预警记录。

id: INT-主键，AUTO\_INCREMENT，唯一标识预警记录。

car\_id: INT-产生预警的车辆的ID。NOT NULL。作为外键关联vehicle\_info表的car\_id字段。

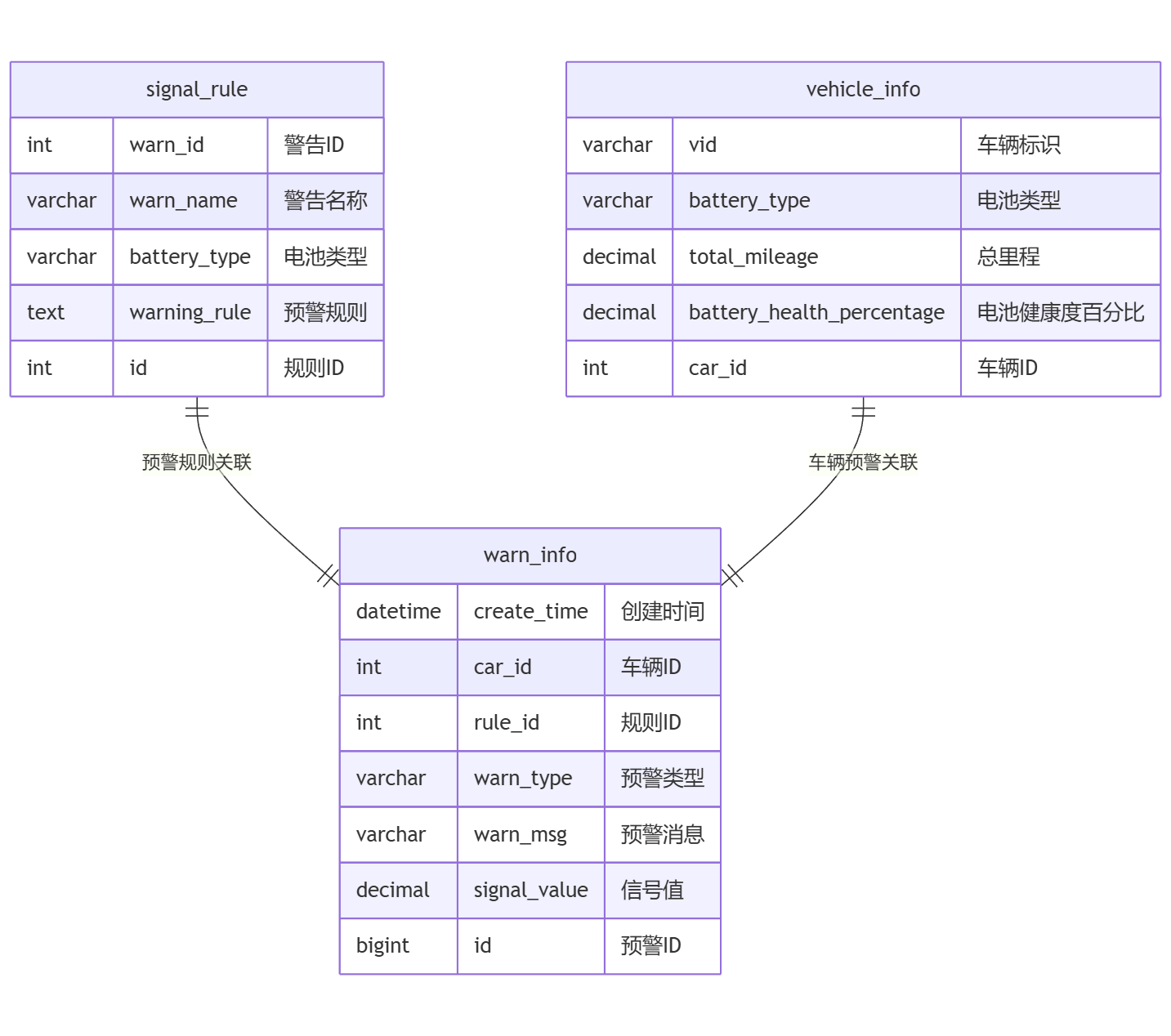
rule\_id: INT - 触发此预警的规则的ID。NOT NULL。作为外键关联signal\_rule表的id字段。

warn\_type: VARCHAR(32)-预警的类型。NOT NULL。

warn\_msg: VARCHAR(255)-具体的预警消息。NOT NULL。

signal\_value: DECIMAL(10, 2)-触发预警时的关键信号值 NOT NULL，总共最多10位数字，小数点后保留两位。

create\_time: DATETIME - 预警生成的时间。DEFAULT CURRENT\_TIMESTAMP，默认为记录插入时的当前时间。



## 三.接口设计

采用RESTFul风格

车辆信息接口 (/vehicle):

POST /vehicle/add: 添加车辆信息 (addVehicle)

GET /vehicle/list: 获取车辆列表 (listVehicles)

GET /vehicle/get/{id}: 获取单个车辆信息 (getVehicle)

POST /vehicle/update: 更新车辆信息 (updateVehicle)

DELETE /vehicle/delete/{id}: 删除车辆信息 (deleteVehicle)

请求和响应均使用统一的 Result 封装。

预警信息接口 (/api/warn - 根据项目代码结构和功能推测):

POST /api/warn: 接收车辆信号数据并处理预警 (由 MQ 消费者调用或直接调用 addWarnInfo 方法)

GET /api/warn/car/{carId}: 根据车辆ID获取该车辆的所有预警信息 (getWarnsByCarId)

GET /api/warn/all: 获取系统中所有的预警信息 (getAllWarns)

请求和响应均使用统一的Result封装。

## 四.单元测试

##### 服务层测试：

VehicleInfoServiceTest：测试车辆信息的增删改查和缓存操作

WarnInfoServiceTest：测试预警信息的添加、查询和缓存管理

##### 工具类测试：

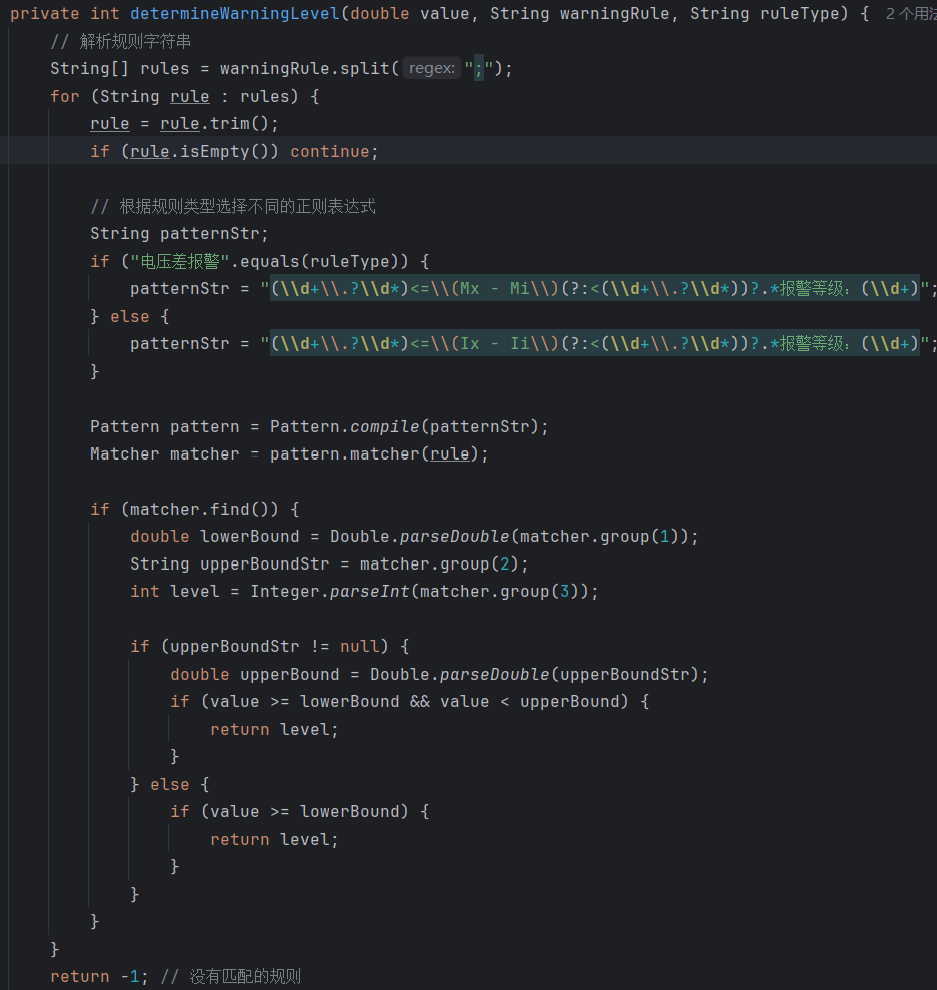
SnowflakeIdGeneratorTest：测试ID生成器的功能、边界条件和并发安全

##### 应用测试：

测试框架：使用JUnit 5 + Mockito进行测试，通过Mock对象隔离外部依赖，确保测试的独立性和可靠性。

## 加分项目：

## 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



通过这个方法利用正则表达式来解析规则，实现动态解析。

2.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

这个我测试时好像基本不超过一秒钟

3.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。

我觉得主要的数据压力在与 warn\_info这张表上，所以主要是对与这张表的处理。

第一点：按照时间分表分库，减轻负担。

第二点：因为这个信息的时效性高，所以对于一定时间前的数据可以做一些清理或者压缩之类的。

第三点：查询时限制分页大小，建立索引比如车辆id和时间的联合索引，查询时加上时间。