

# 小米大作业-方案设计

---

## 1. 系统设计

### 1.1. 需求分析

### 1.2. 具体思路

#### 1.2.1. 电池信息数据缓存设计

##### 1.2.1.1. 查询流程：

##### 1.2.1.2. 保存流程：

##### 1.2.1.3. 删除流程：

#### 1.2.2. 定时任务设计

#### 1.2.3. 预警规则解析设计

## 2. 数据库表设计

### 2.1. 车辆信息表

### 2.2. 电池信号记录表

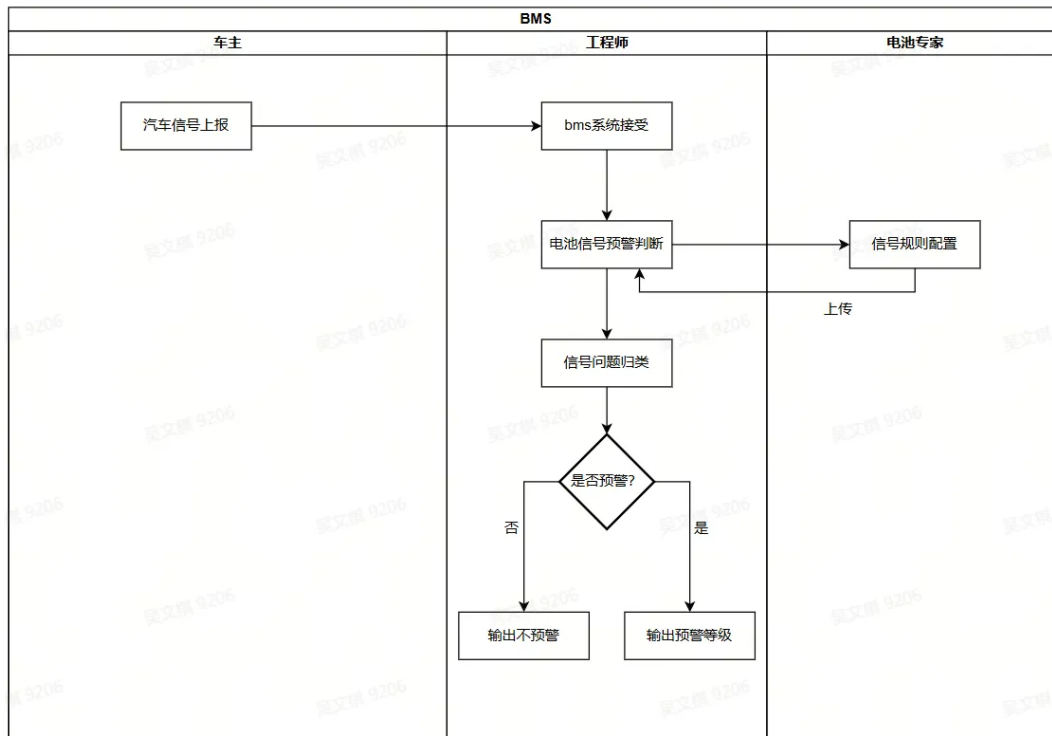
### 2.3. 预警规则表

### 2.4. 预警信息记录表

## 3. 接口设计

## 4. 具体实现（截图 + 代码）

# 1. 系统设计



## 1.1. 需求分析

1. 汽车信息模块，需要实现：
  - a. 车辆信息的上传，车辆vid、车架编号、电池类型、总里程、电池健康度。
2. 电池模块，需要实现：
  - a. 电池信息的上传，信息保存到数据库。
  - b. 电池信息查询，数据量大。需要使用分页查询，缓存查询结果，并保持数据一致。
  - c. 电池信息删除，根据信息记录 id 和车架 carId 进行删除，删除需判断信息所属的车辆。
3. 电池专家模块，需要实现：
  - a. 配置电池信号的预警规则，考虑可拓展性。
4. 预警模块，需要实现：
  - a. 定时扫描电池信号表，解析电池信号，转换为预警信息保存到数据库，需要注意：
    - i. 电池信号数据量较大，需要控制对数据库的 IO 操作。
    - ii. 定时任务的启动时间设定为系统压力较小的时间段。

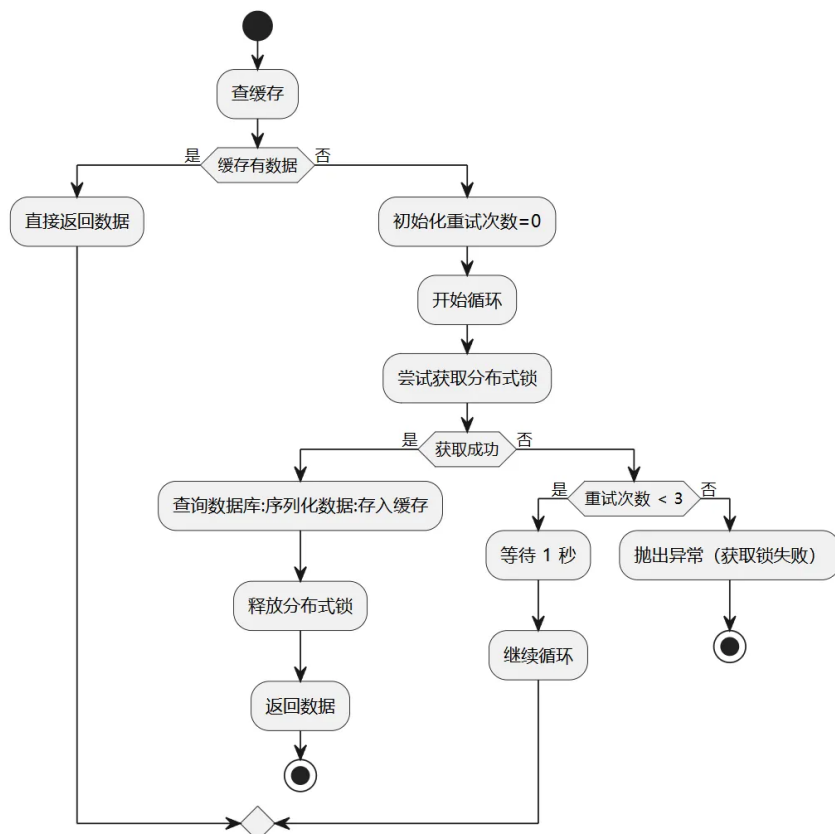
## 1.2. 具体思路

核心点分为：电池信息数据缓存、定时任务设计、预警规则动态解析

## 1.2.1. 电池信息数据缓存设计

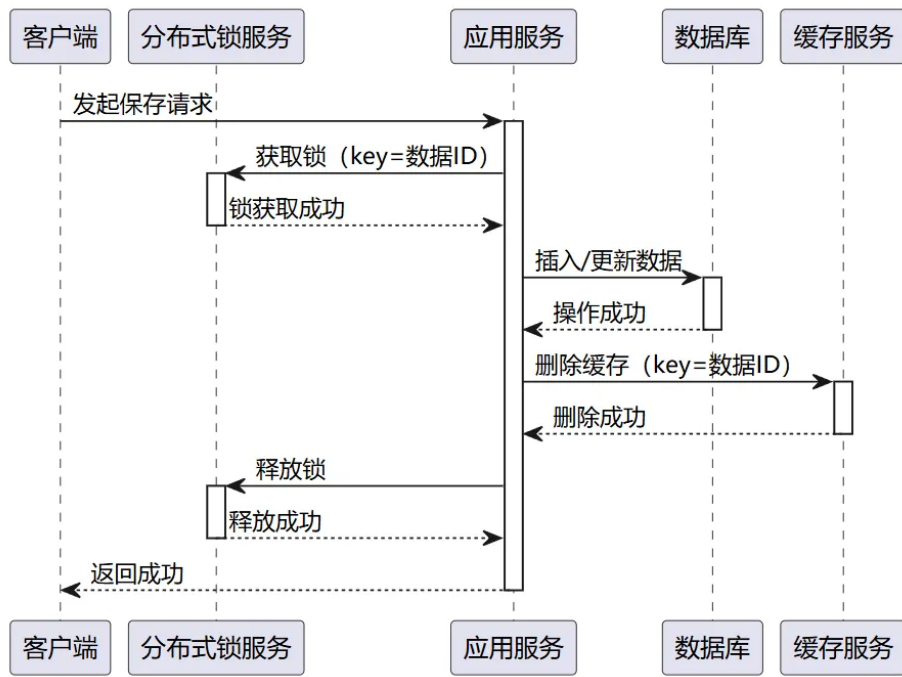
### 1.2.1.1. 查询流程：

1. 先查缓存，缓存有数据直接返回
2. 缓存没有，尝试获取分布式锁（失败则重复，重复三次，每一等待一秒）
3. 获取到分布式锁后，查询数据库，将数据进行序列化，存入缓存
4. 释放分布式锁



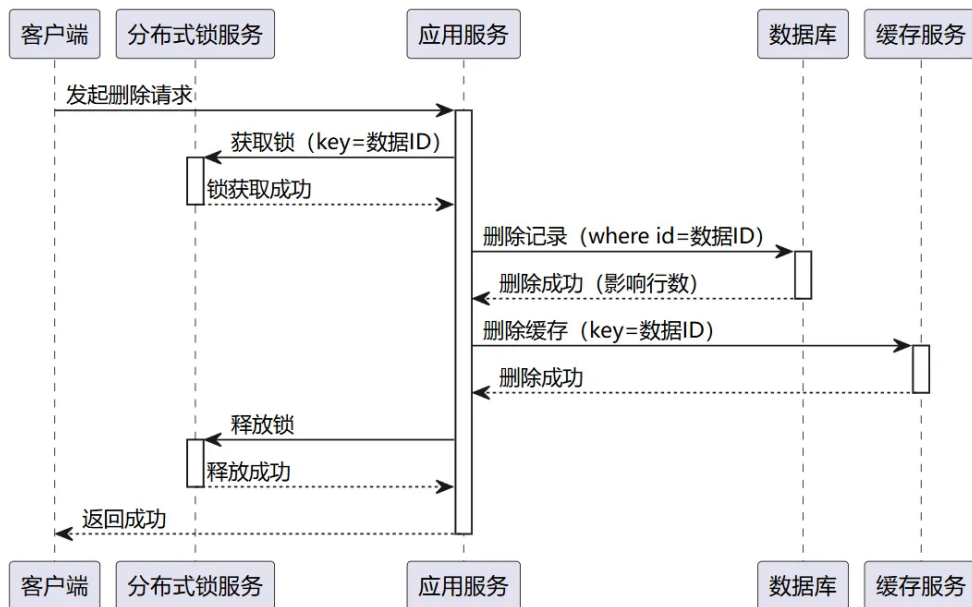
### 1.2.1.2. 保存流程：

1. 获取分布式锁
2. 将数据保存到数据库
3. 删除缓存
4. 释放分布式锁



### 1.2.1.3. 删除流程：

1. 获取分布式锁
2. 先删除 数据库，再删除缓存
3. 释放分布式锁



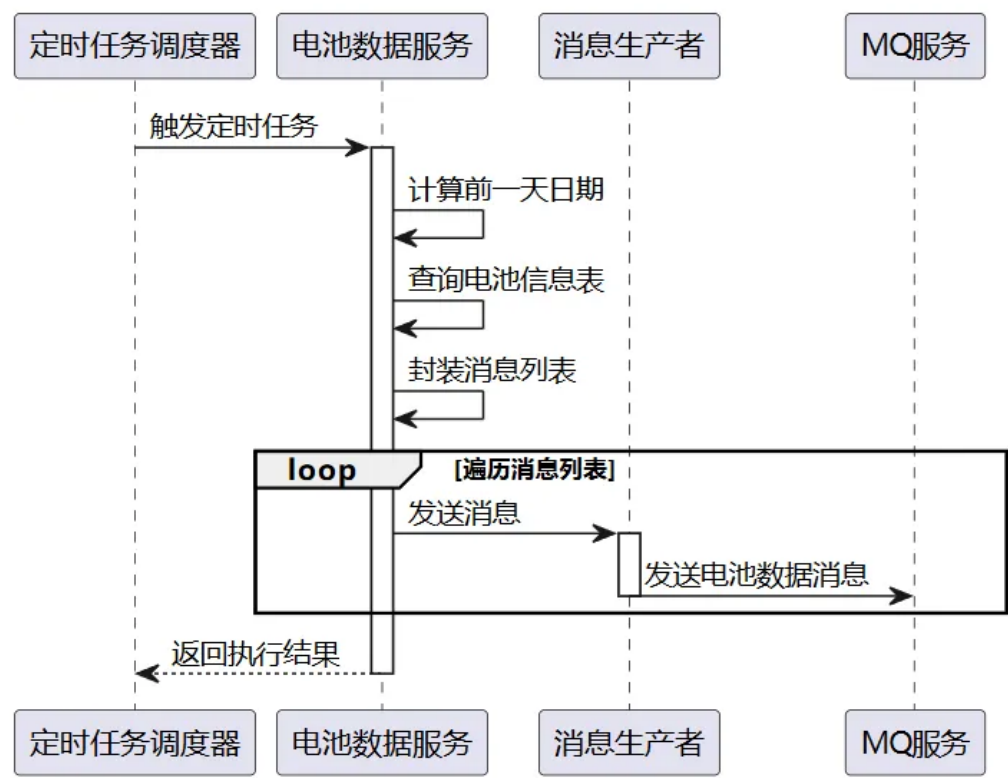
### 1.2.2. 定时任务设计

预警模块中定义一个定时任务，任务执行步骤：

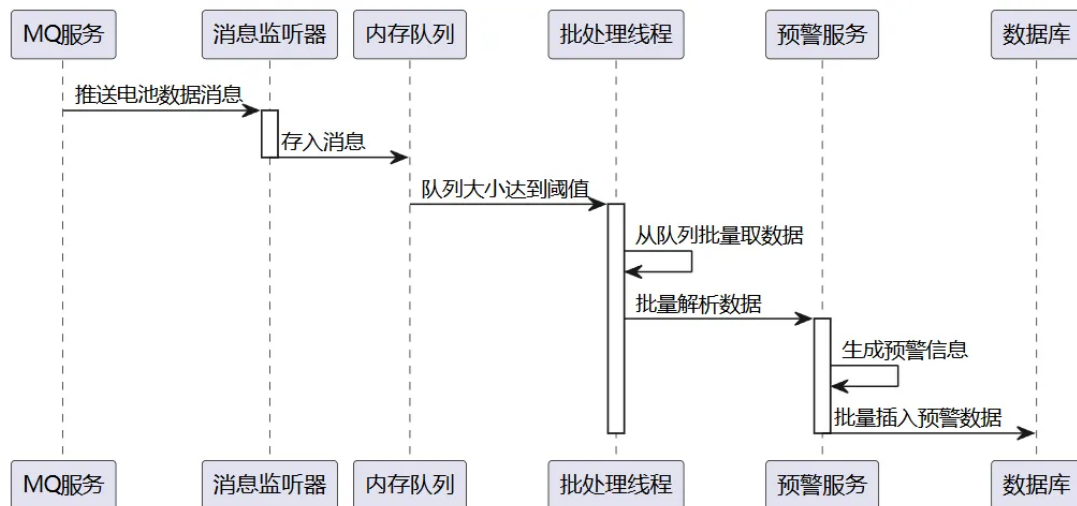
- 1. 根据当前日期查询前一天的电池信息表，获取电池信息
- 2. 通过 mq 进行推送。不考虑消息推送的顺序性，配置为 MessageListenerConcurrently 异步发送
- 3. 消费者端接受数据，设计一个队列存储接收到的数据，运行在另一个线程，确保消费者线程正常
- 4. 队列中的数据个数到达设定的批处理数量，即进行批处理
- 5. 批量解析电池数据获取预警信息
- 6. 将预警信息批量存储至数据库，减少对数据库的 IO 次数

个人认为此场景不用考虑幂等消费问题，去掉幂等校验能减少数据库查询次数、简化业务逻辑

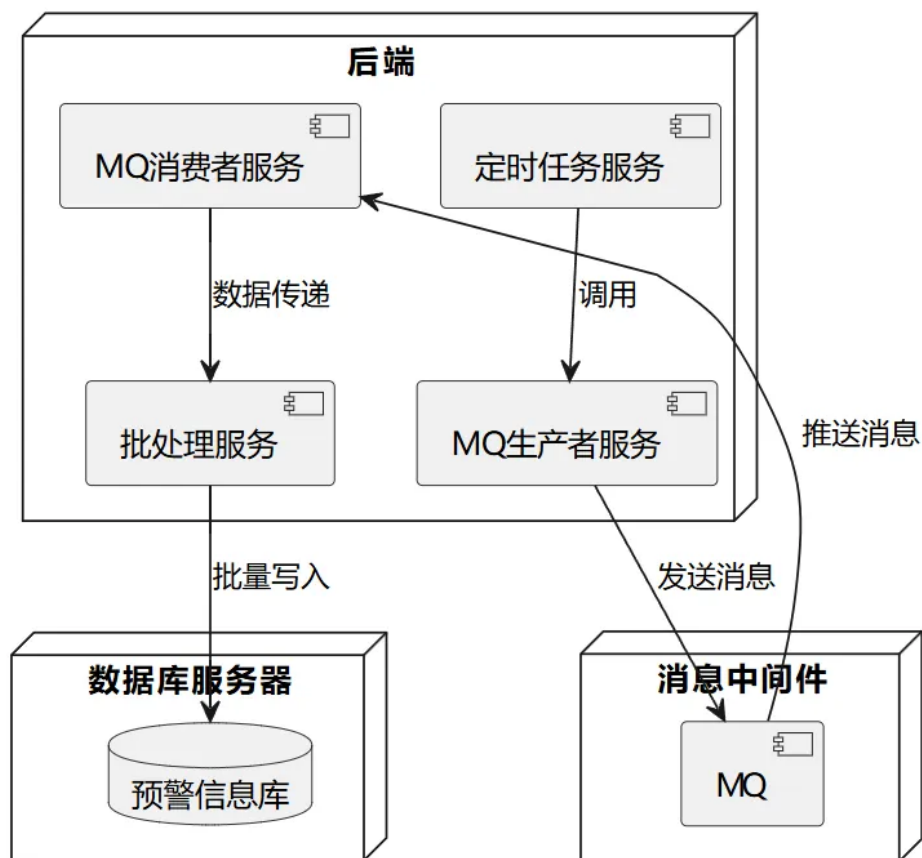
生产者端：



消费者端：

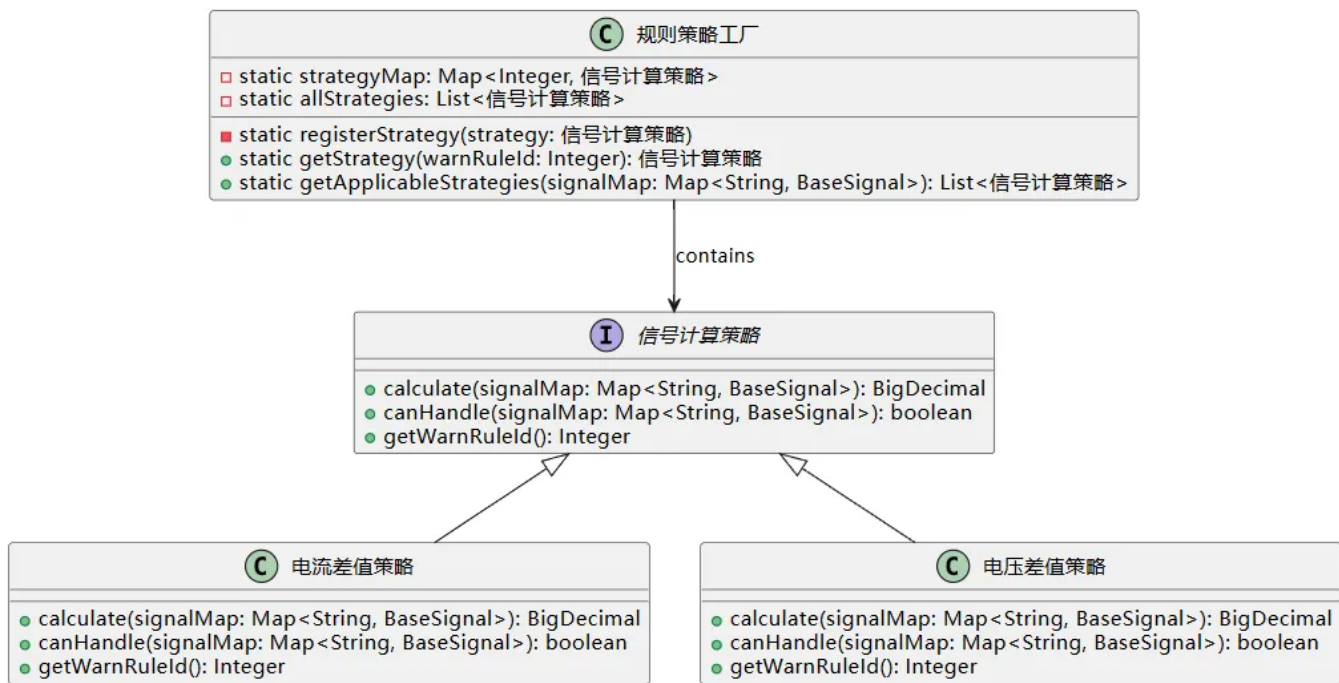


定时任务组件结构：

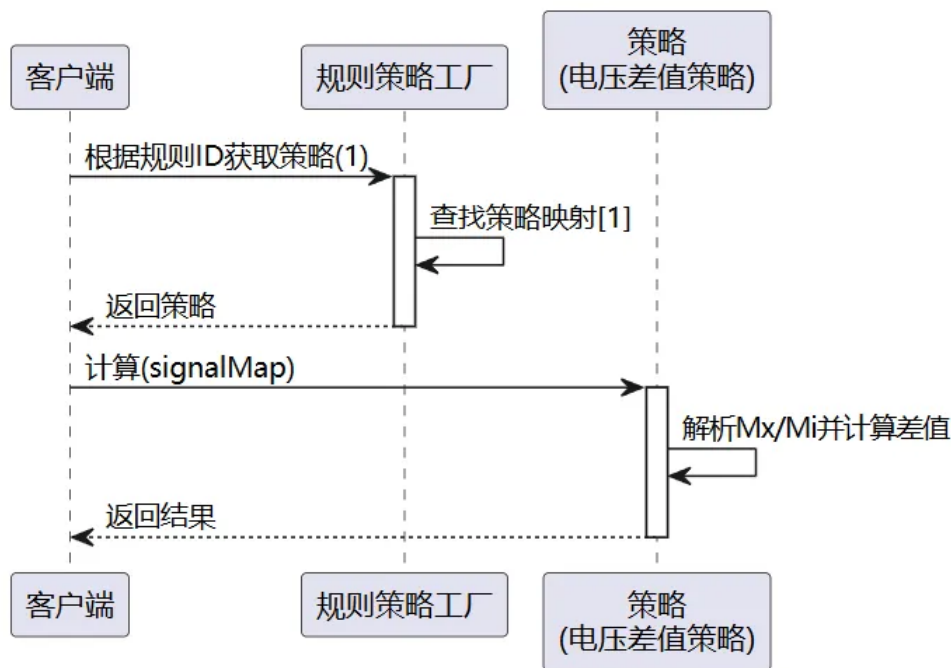


### 1.2.3. 预警规则解析设计

设计策略工厂模式，不同预警规则的解析实现放在具体的策略中。满足开闭原则，可拓展不同的预警规则。



规则解析时序图



## 2. 数据库表设计

总共设计四张表：车辆信息表、电池信号记录表、预警规则表、预警信息记录表

## 2.1. 车辆信息表

▼ 车辆信息表

SQL |

```
1 CREATE TABLE `car_message` (  
2   `vid` BIGINT ( 20 ) NOT NULL COMMENT '车辆识别码',  
3   `frame_number` VARCHAR ( 255 ) DEFAULT NULL COMMENT '车架编号',  
4   `battery_type` VARCHAR ( 255 ) DEFAULT NULL COMMENT '电池类型',  
5   `total_mileage` VARCHAR ( 255 ) DEFAULT NULL COMMENT '总里程',  
6   `battery_health_percent` INT ( 11 ) DEFAULT NULL COMMENT '电池健康度百分比 (%)',  
7   `is_deleted` TINYINT ( 1 ) DEFAULT '0' COMMENT '是否删除',  
8   `create_time` DATETIME DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',  
9   `update_time` DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '修改时间',  
10  PRIMARY KEY ( `vid` ),  
11  KEY idx_fNumber (frame_number)  
12 ) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4 COMMENT = '车辆消息表';
```

## 2.2. 电池信号记录表

创建索引：1.idx\_carId (car\_id) 2.idx\_cld\_rId (car\_id, warn\_rule\_id) 3.idx\_cTime (create\_time)

▼ 电池信号记录表

SQL |

```
1 CREATE TABLE battery_signal_records (  
2   id BIGINT(20) UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '主键ID',  
3   car_id BIGINT ( 20 ) NOT NULL COMMENT '车架编号',  
4   warn_rule_id INT NOT NULL COMMENT '预警规则ID',  
5   battery_signal TEXT NOT NULL COMMENT '电池信号json字符串',  
6   is_deleted TINYINT(1) UNSIGNED NOT NULL DEFAULT 0 COMMENT '是否删除(0-未删除,1-已删除)',  
7   create_time DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '记录创建时间',  
8   update_time DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '记录更新时间',  
9   PRIMARY KEY (id),  
10  KEY idx_carId (car_id),  
11  KEY idx_cId_rId (car_id, warn_rule_id),  
12  KEY idx_cTime (create_time)  
13 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='电池信号记录表';
```



## 2.3. 预警规则表

▼ 预警规则表

SQL |

```
1 CREATE TABLE battery_warn_rules (
2     id BIGINT(20) NOT NULL COMMENT '主键',
3     warn_rule_id INT NOT NULL COMMENT '预警规则ID',
4     warn_name VARCHAR(50) NOT NULL COMMENT '预警规则名称',
5     warn_max_value DECIMAL(5,2) COMMENT '报警最大值',
6     warn_min_value DECIMAL(5,2) COMMENT '报警最小值',
7     warn_level INT NOT NULL COMMENT '报警等级',
8     battery_type VARCHAR(20) NOT NULL COMMENT '电池类型',
9     is_deleted TINYINT(1) DEFAULT '0' COMMENT '是否删除',
10    create_time DATETIME DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
11    update_time DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMES
12    TAMP COMMENT '修改时间',
13    PRIMARY KEY (id)
14 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='电池预警规则表';
```

## 2.4. 预警信息记录表

创建索引: 1.idx\_carId (car\_id) 2.KEY idx\_cId\_wlevel (car\_id, warn\_level)

▼ 预警信息记录表

SQL |

```
1 CREATE TABLE battery_warning_records (
2     id BIGINT(20) NOT NULL COMMENT '主键',
3     car_id BIGINT ( 20 ) NOT NULL COMMENT '车架编号',
4     battery_type VARCHAR(20) NOT NULL COMMENT '电池类型',
5     warn_name VARCHAR(50) NOT NULL COMMENT '预警规则名称',
6     warn_level INT NOT NULL COMMENT '报警等级(0-4)',
7     is_deleted TINYINT(1) DEFAULT '0' COMMENT '是否删除(0-未删除,1-已删除)',
8     create_time DATETIME DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
9     update_time DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMES
10    TAMP COMMENT '修改时间',
11    PRIMARY KEY (id),
12    KEY idx_cid (car_id),
13    KEY idx_cId_wlevel (car_id, warn_level)
14 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='电池预警信息记录表';
```

### 3. 接口设计

统一响应参数：

```

1  /**
2   * 通用响应结果
3   * @param <T>
4   */
5   @Data
6   @NoArgsConstructor
7   public class Response<T> {
8
9       private Integer code;
10      private String message;
11      private Boolean success;
12      private T data;
13
14      // 成功的构造方法
15      public Response(Integer code, String message, Boolean success, T data)
16      {
17          this.code = code;
18          this.message = message;
19          this.success = success;
20          this.data = data;
21      }
22
23      // 失败的构造方法
24      public Response(Integer code, String message) {
25          this.code = code;
26          this.message = message;
27          this.success = false;
28      }
29
30      // 成功-有数据响应
31      public static <T> Response<T> success(T data) {
32          return new Response<>(200, "success", true, data);
33      }
34
35      // 成功-无数据响应
36      public static <Void> Response<Void> success() {
37          return new Response<>(200, "success", true, null);
38      }
39
40      // 失败-带自定义状态码
41      public static <T> Response<T> fail(Integer code, String message) {
42          return new Response<>(code, message);
43      }
44
45      // 失败-默认400状态码
46      public static <T> Response<T> fail(String message) {
47          return new Response<>(400, message);
48      }
49  }

```

```

47     }
48 }

```

### • 汽车信息上传接口

请求路径：/carMessage/saveCarMessage

接口方法：POST

请求参数：

字段名	类型	是否必填	描述	约束条件
vid	Long	是	车辆唯一识别码	非空
frameNumber	String	是	车架编号	非空
batteryType	String	是	电池类型	不能为空；使用枚举值
totalMileage	String	否	总里程	
batteryHealthPercent	Integer	是	电池健康度百分比	$0 \leq \text{值} \leq 100$

响应数据示例：

```

1 {
2     "code": 200,    // 状态码
3     "message": "success", // 状态码，成功success，失败fail
4     "success": true, //是否成功
5     "data": null    // 返回数据，无数据时为空
6 }

```

- 电池信息上传接口

请求路径：/battery/uploadBatteryMessage

接口方法：POST

请求参数:

字段名	类型	是否必填	描述	约束条件
carId	Long	是	关联车辆的唯一标识	不能为空；需对应数据库中存在的车辆记录
warnId	Long	否	预警规则 id	为空的时候遍历所有预警规则
batterySignal	String	是	电池信号 JSON 字符串	需为合法 JSON 格式

响应参数:

字段名	类型	描述
code	Integer	状态码（200 = 成功，非 200 = 失败）
message	String	状态描述信息（成功 / 失败原因）
success	Boolean	请求是否成功
data	Object	请求响应数据

-----

-----

- 电池状态信息查询接口（分页）

请求路径：/battery/queryBatteryMessageByCarId

接口方法：POST

请求参数:

字段名	类型	是否必填	描述	约束条件
-----	----	------	----	------

carId	Long	是	车架编号（关联车辆的唯一标识）	不能为空；需对应数据库中存在的车辆记录
pageNum	Integer	是	页码（从 1 开始）	必须 $\geq 1$
pageSize	Integer	是	每页数据量	必须 $\geq 1$

响应参数示例:

```
1 {
2     "code": 200,    // 状态码
3     "message": "success",
4     "success": true,
5     "data": {      // 返回数据
6         "pageNum": 1, // 页码
7         "pageSize": 10, // 请求的每页数据量
8         "totalPage": 1, // 总页数
9         "carId": 10, // 车架唯一id
10        "batterySignalList": [ // 响应数据
11            {
12                "id": 13,
13                "warnId": 2,
14                "batterySignal": "{\"Ix\": 12.0, \"Ii\": 11.7}",
15                "recordTime": "2025-05-19T13:26:25"
16            },
17            {
18                "id": 25,
19                "warnId": null,
20                "batterySignal": "{\"Mx\":11.0,\"Mi\":9.6,\"Ix\":12.0,\"Ii
21                \":11.7}",
22                "recordTime": "2025-05-19T17:35:37"
23            }
24        ]
25    }
}
```

-----

-----

• 电池信息删除接口

请求路径: /battery/deleteBatteryMsgById

接口方法：POST

请求参数:

字段名	类型	是否必填	描述	约束条件
msgId	Long	是	电池信息记录 id	需对应数据库中存在的记录
carId	Long	是	车架编号（关联车辆的唯一标识）	与 id 关联的车辆记录一致

响应参数:

字段名	类型	描述
code	Integer	状态码（200 = 成功，非 200 = 失败）
message	String	状态描述信息（成功 / 失败原因）
success	Boolean	请求是否成功
data	Object	null

-----

-----

• 预警信息记录查询接口

请求路径：/warn/getWarnRecords

接口方法：POST

请求数据:

字段名	类型	是否必填	描述
carId	Long	是	车架编号（关联车辆的唯一标识）

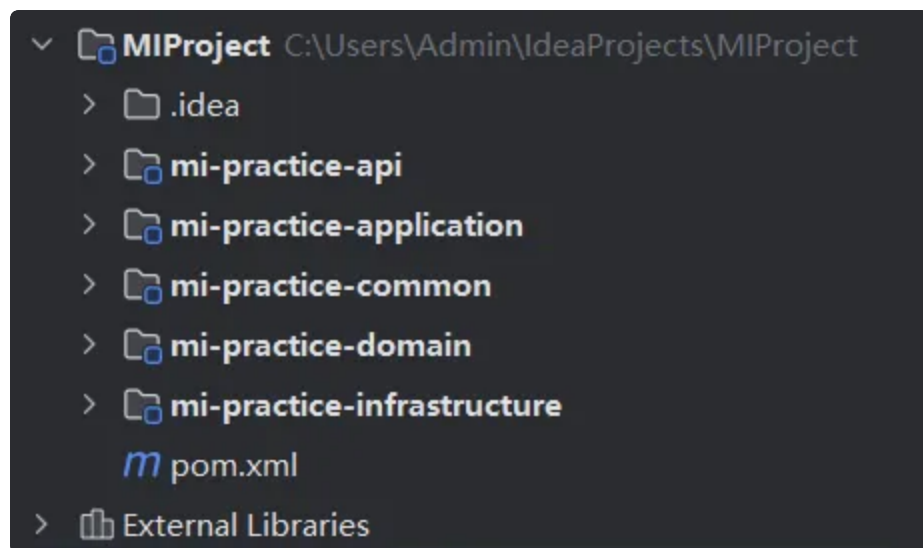
响应参数:

字段名	类型	描述
code	Integer	状态码（200 = 成功，非 200 = 失败）

message	String	状态描述信息（成功 / 失败原因）
success	Boolean	请求是否成功
data	Object	null

## 4. 具体实现（截图 + 代码）

项目结构：DDD 思想的多模块项目架构



单元测试，模拟电池专家添加预警规则，并动态解析



MIProject

MIProject C:\Users\Admin\IdeaProjects\MIProject

mi-practice-api

src

main

java

com.xiaomi.practice.api

base

battery

car

warn

test

java

com.xiaomi.practice.api

CarControllerTest

TestRuleTest

target

pom.xml

mi-practice-application

src

main

java

com.xiaomi.practice.web

battery

car

dto

job

HandleBatterySignalWarnJob

warn

WarnApplicationService

WarnApplicationServiceImpl

target

pom.xml

mi-practice-common

mi-practice-domain

src

```
21 public class TestRuleTest {
22
23     @Autowired
24     private BatteryWarnDomainServiceImpl batteryWarnDomainService;
25
26     @Test
27     public void testDynamicRule() {
28         // 1.从数据库中获取所有电池预警规则
29         RuleDomainService ruleDomainService = new RuleDomainServiceImpl();
30         List<BatteryWarnRule> allRules = ruleDomainService.getAllRules();
31
32         // 2.模拟电池专家 添加预警规则
33         BatteryWarnRule rule = new BatteryWarnRule();
34         rule.setWarnRuleId(1);
35         rule.setWarnName("电池电压差值--预警");
36         // 设置预警范围值
37         rule.setWarnMaxValue(new BigDecimal("100.00"));
38         rule.setWarnMinValue(new BigDecimal("80.00"));
39         // 设置报警等级
40         rule.setWarnLevel(6);
41         // 设置电池类型
42         rule.setBatteryType("超感电池");
43         allRules.add(rule);
44
45         System.out.println(allRules.size());
46
47         // 2.模拟电池信号数据
48         BatterySignalDTO signalDTO = new BatterySignalDTO();
49         signalDTO.setCarId(1L);
50         signalDTO.setBatteryType("超感电池");
51         signalDTO.setBatterySignal("{\"Mx\":\"200.0\",\"Mi\":\"110\",\"Ix\":\"12.0\",\"Ii\":\"11.7\"}");
52         List<BatterySignalDTO> signalDTOS = new ArrayList<>();
53         signalDTOS.add(signalDTO);
54
55         List<WarnResult> warnResultList = batteryWarnDomainService.parseSignals(signalDTOS, allRules);
56
57         System.out.println(warnResultList.get(0));
58     }
59 }
60
61
62
63 }
```

模拟电池专家添加预警规则

模拟电池信号

TestRuleTest (com.xiaomi.practice.api)

testDynamicRule 478 ms

Tests passed: 1 of 1 test - 478 ms

2025-05-19 20:24:21.902 INFO 40732 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...

2025-05-19 20:24:22.055 INFO 40732 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.

17

2025-05-19 20:24:22.104 INFO 40732 --- [main] c.x.d.w.s.BatteryWarnDomainServiceImpl : 开始解析电池信号列表, signalList1, allRu

WarnResult(carId=1, warnLevel=6, warnName=电池电压差值--预警, batteryType=超感电池)

输出预警信息

```

1  /**
2   * 测试动态配置预警规则并解析
3   */
4   @Test
5   public void testDynamicRule() {
6       // 1.从数据库中获取所有电池预警规则
7       // RuleDomainService ruleDomainService = new RuleDomainServiceImpl
8       ();
9
10      List<BatteryWarnRule> allRules = ruleDomainService.getAllRules();
11
12      // 2.模拟电池专家 添加预警规则
13      BatteryWarnRule rule = new BatteryWarnRule();
14      rule.setWarnRuleId(1);
15      rule.setWarnName("电池电压差值--预警");
16      // 设置预警范围值
17      rule.setWarnMaxValue(new BigDecimal("100.00"));
18      rule.setWarnMinValue(new BigDecimal("80.00"));
19      // 设置报警等级
20      rule.setWarnLevel(6);
21      // 设置电池类型
22      rule.setBatteryType("超威电池");
23      allRules.add(rule);
24
25      System.out.println(allRules.size());
26
27      // 2.模拟电池信号数据
28      BatterySignalDTO signalDTO = new BatterySignalDTO();
29      signalDTO.setCarId(1L);
30      signalDTO.setBatteryType("超威电池");
31      signalDTO.setBatterySignal("{\"Mx\":200.0,\"Mi\":110,\"Ix\":12.0,\"Ii\":11.7}");
32
33      List<BatterySignalDTO> signalDTOS = new ArrayList<>();
34      signalDTOS.add(signalDTO);
35
36      List<WarnResult> warnResultList = batteryWarnDomainService.parseSignals(signalDTOS, allRules);
37
38      System.out.println(warnResultList.get(0));
39  }

```

预警模块解析电池信号，消费者端

阻塞队列 + 批量处理

```

1  @Slf4j
2  @Component
3  @RocketMQMessageListener(consumerGroup = "MyConsumerGroup", topic = "TestTopic", consumeMode= ConsumeMode.CONCURRENTLY, messageModel= MessageModel.BROADCASTING)
4  public class BatteryMsgConsumer implements RocketMQListener<String> {
5
6      // 阻塞队列存储dto
7      private final BlockingQueue<BatterySignalDTO> dtoQueue = new LinkedBlockingQueue<>();
8
9      // 批量处理数量
10     private static final int BATCH_SIZE = 2;
11
12     // 预警规则列表
13     List<BatteryWarnRule> allRules;
14
15     .....
16
17     /**
18      * 消费者实例化后，启动后台线程，处理队列中的dto
19      */
20     public BatteryMsgConsumer() { new Thread(this::batchProcessLoop).start();}
21
22     /**
23      * 接收消息
24      */
25     @Override
26     public void onMessage(String message) {
27         try {
28             BatterySignalDTO dto = JSON.parseObject(message, BatterySignalDTO.class);
29             // 将消息放入队列中，后台异步线程批量处理，避免阻塞消费线程
30             dtoQueue.put(dto);
31         } catch (Exception e) {
32             log.error("消息消费失败: {}", message, e);
33         }
34     }
35
36     /**
37      * 后台线程循环批量处理队列中的dto，
38      */
39     private void batchProcessLoop() {
40         while (true) {
41             try {
42

```

```

43         List<BatterySignalDTO> dtoList = new ArrayList<>(BATCH_SIZ
44     E);
45     // 从队列中取出BATCH_SIZE个dto
46     for (int i = 0; i < BATCH_SIZE; i++) {
47         BatterySignalDTO dto = dtoQueue.take();
48
49         // 根据carId获取电池类型
50         String batteryType = getBatteryTypeByCarId(dto.getCarI
51     d());
52         dto.setBatteryType(batteryType);
53         dtoList.add(dto);
54     }
55     batchConsumerBatteryMessage(dtoList);
56 } catch (InterruptedException e) {
57     log.error("批量处理线程被中断", e);
58     // 恢复中断状态
59     Thread.currentThread().interrupt();
60 } catch (Exception e) {
61     log.error("批量处理失败", e);
62 }
63 }
64 .....
65 }

```

## 电池信息相关操作

redis缓存 + 缓存时间设置为区间范围内随机

```

1  /**
2   * 分页查询电池信息，并缓存
3   */
4   @Override
5   public BatteryMsgDTO queryBatteryMsgListByCarId(QueryBatteryMsgDTO queryBatteryMsgDTO) {
6       // 1.查缓存
7       Long carId = queryBatteryMsgDTO.getCarId();
8       Integer pageNum = queryBatteryMsgDTO.getPageNum();
9       Integer pageSize = queryBatteryMsgDTO.getPageSize();
10      // 1.1 获取缓存key, 根据carId、pageNum、pageSize生成
11      String batteryMsgPageKey = RedisKeyUtil.getBatteryMsgPageKey(carId
12      , pageNum, pageSize);
13      log.info("查询缓存{}", batteryMsgPageKey);
14      String batteryMsgPageCache = redisTemplateUtil.get(batteryMsgPageKey);
15      if (Objects.nonNull(batteryMsgPageCache)) {
16          log.info("缓存命中, carId{}, pageNum{}, pageSize{}", carId, pageNum, pageSize);
17          return JSON.parseObject(batteryMsgPageCache, BatteryMsgDTO.class);
18      }
19
20      // 2.缓存没有从数据库查询
21      String lockKey = RedisKeyUtil.getBatteryMsgLockKey(carId);
22      String lockValue = UUID.randomUUID().toString();
23      try {
24          // 尝试获取锁，失败则等待一秒，尝试三次
25          boolean isLock = redisTemplateUtil.tryLock(lockKey, lockValue,
26          2, TimeUnit.SECONDS, 3, 1);
27          if (isLock) {
28              BatteryMsgDTOList batteryMsgDTOList = batteryMsgRepository.queryBatteryMsgListByCarId(queryBatteryMsgDTO);
29              log.info("从数据库查询, carId{}, pageNum{}, pageSize{}", carId, pageNum, pageSize);
30
31              // 序列化后存入Redis
32              String cacheValue = JSON.toJSONString(batteryMsgDTOList);
33              // 缓存时间设置为区间随机值
34              int randomExpiry = ThreadLocalRandom.current().nextInt(30, 46);
35              redisTemplateUtil.set(batteryMsgPageKey, cacheValue, randomExpiry, TimeUnit.SECONDS);
36              return batteryMsgDTOList;
37          } catch (Exception e) {

```

```
38         log.error("获取锁失败, carId{}, pageNum{}, pageSize{}", carId
39 , pageNum, pageSize);
40     } finally {
41         // 释放锁时校验锁值
42         if (lockValue.equals(redisTemplateUtil.get(lockKey))) {
43             redisTemplateUtil.delete(lockKey);
44         }
45         log.info("释放锁成功");
46     }
47     return new BatteryMsgDTO();
48 }
```