**考核大作业**

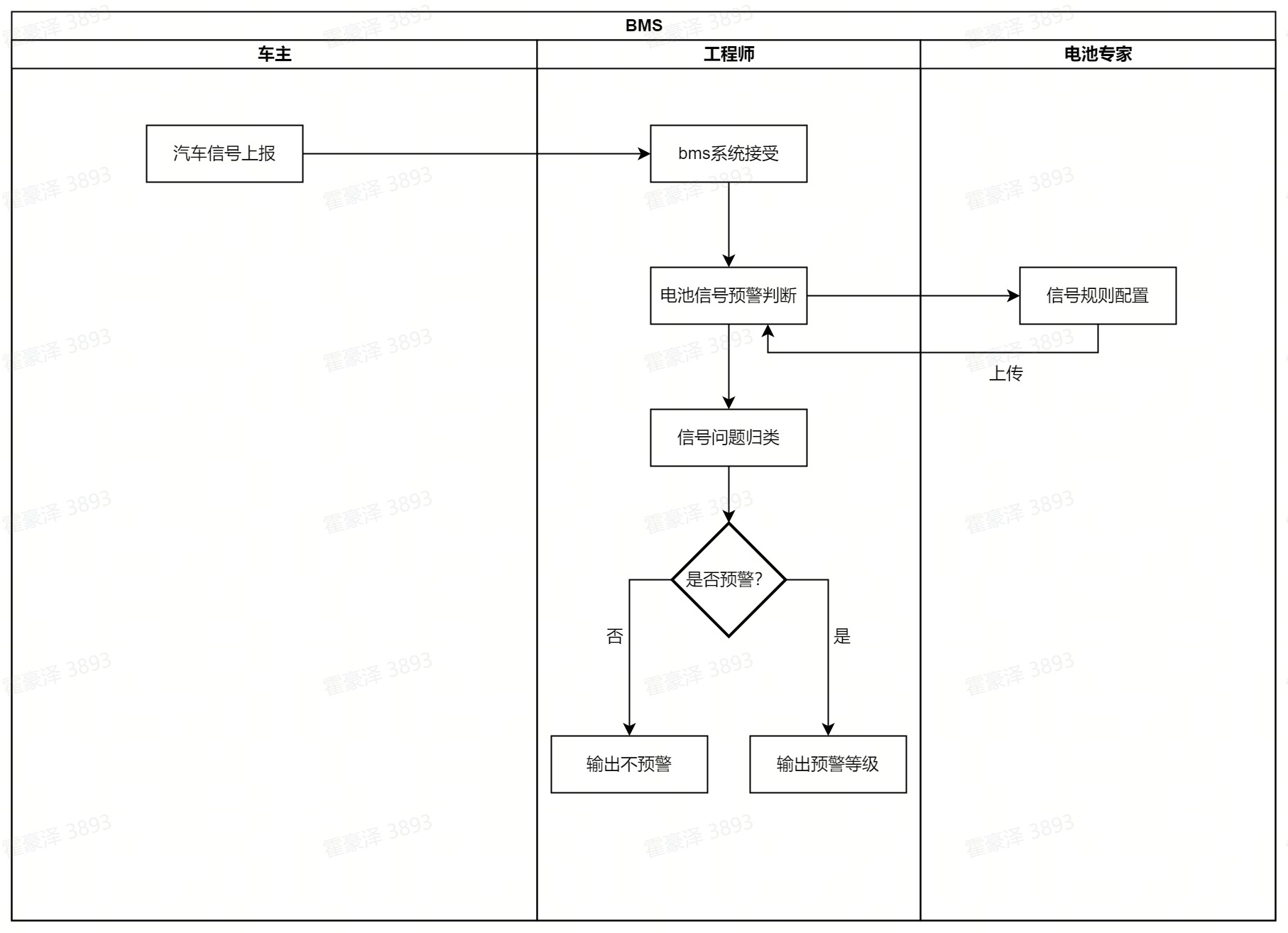
**一、系统设计**

该系统是一个 **BMS（电池管理系统）信号预警平台**，主要功能包括接收车辆上报的信号数据、结合规则计算预警等级，并将预警结果记录或推送。

项目采用技术栈：

* **Spring Boot**：基础框架
* **MyBatis**：数据库访问
* **Redis**：规则缓存（AlarmRuleCacheRedis）
* **RocketMQ**：异步预警消息推送
* **XXL-Job**：定时任务调度（XxlJobConfig）
* **MySQL**：数据持久化存储

**整体业务图**

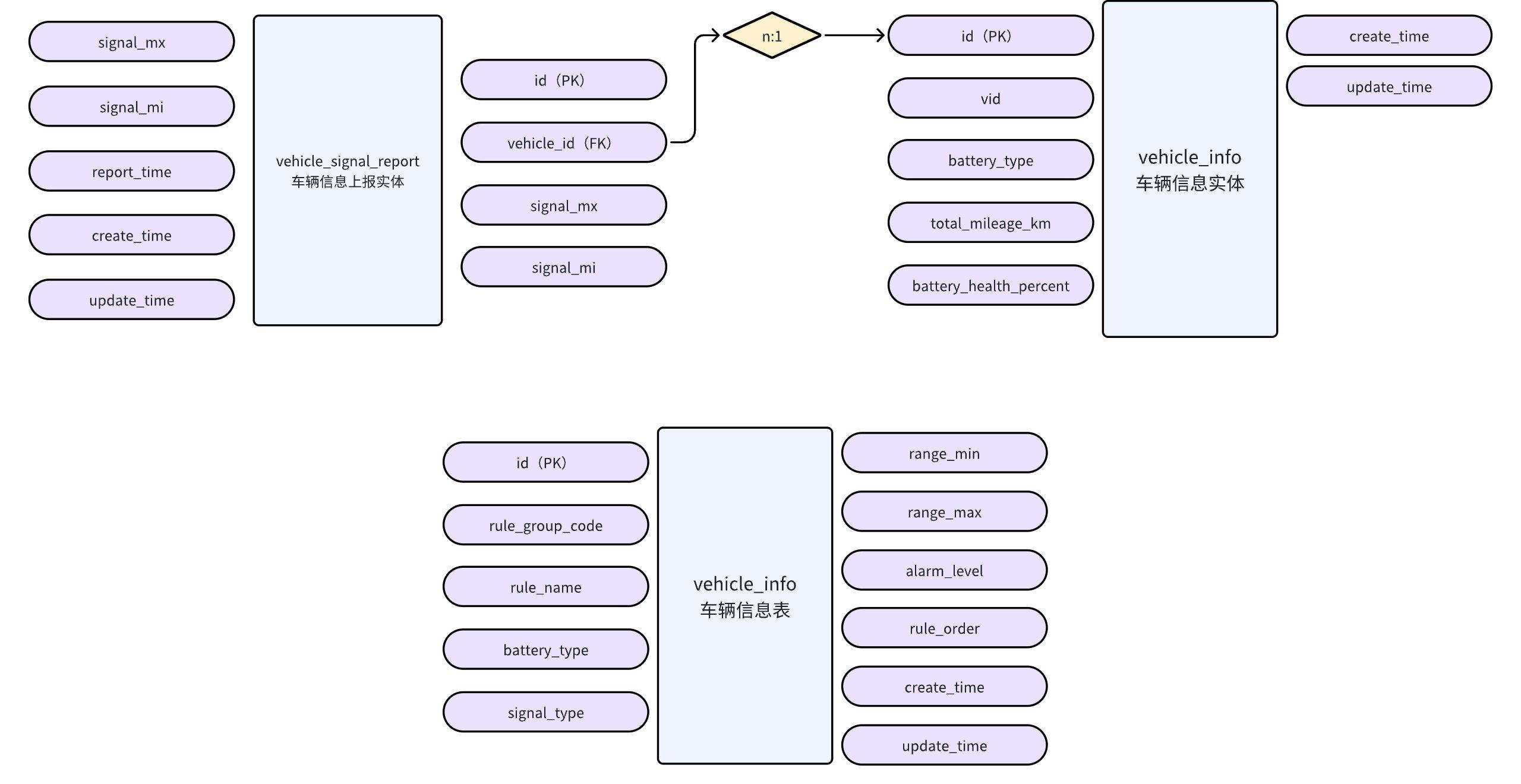


**二、数据库设计**

**1.数据库文件mbsv4.sql**

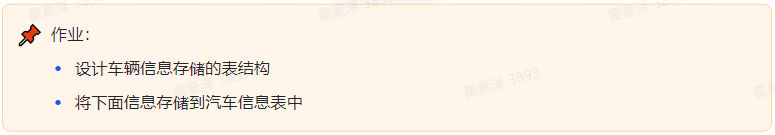
最后决定还是通过车辆信息上报表的car\_id与车辆信息表的car\_id进行关联，车辆编号可能会有字母

**2.ER关系图**



**三、功能模块设计**

**1、车辆信息模块**

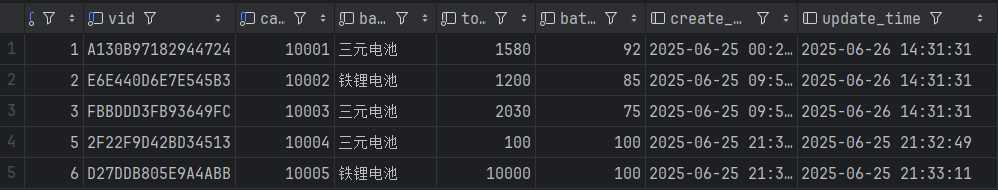


**（1）设计车辆信息表设计（vid,车架编号,电池类型,总里程(km),电池健康状态(%)）**

|  |
| --- |
| 车辆信息录入是因为：先有车才有电池，最后才会在车行驶中产生电流信号  vid: Vehicle Identification 车辆识别码，每辆车唯一，16位随机字符串  电池类型：三元电池、铁锂电池 |

车辆信息存储的表结构：

|  |
| --- |
| Plain Text CREATE TABLE vehicle\_info (  id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY COMMENT '主键 ID',   vid CHAR(16) NOT NULL UNIQUE COMMENT '车辆识别码（16位）',  car\_id INT UNSIGNED NOT NULL UNIQUE COMMENT '车辆编号（前端 carId，业务主键）',   battery\_type ENUM ('三元电池', '铁锂电池') NOT NULL COMMENT '电池类型',  total\_mileage\_km INT UNSIGNED NOT NULL COMMENT '总里程（单位：公里）',  battery\_health\_percent TINYINT UNSIGNED NOT NULL COMMENT '电池健康状态（0~100）',   create\_time TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP* COMMENT '创建时间',  update\_time TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP* ON UPDATE *CURRENT\_TIMESTAMP* COMMENT '更新时间',  INDEX idx\_carId (car\_id) ) ENGINE = InnoDB  DEFAULT CHARSET = utf8mb4 COMMENT ='车辆信息表'; |



**（2）接口设计**

**对请求的响应进行了统一规范的封装返回数据**

**A.新增车辆信息接口**

|  |
| --- |
| Plain Text POST http://localhost:9081/vehicles/insert *Content-Type*: application/json  {  "carId": 1,  "batteryType": "三元电池",  "totalMileageKm": 100,  "batteryHealthPercent": 100 } |

通过post请求，同时在后端通过UUID生成16位随机字符串，将数据写入数据库

|  |
| --- |
| JSON {  "code": 200,  "msg": "OK",  "data": "车辆新增成功." } |

**B.更新车辆信息接口**

|  |
| --- |
| Plain Text PUT http://localhost:9081/vehicles/update *Content-Type*: application/json  {  "vid": "A130B97182944724", //这个字段（可有可无--可以修改，也可以不修改），根据carId进行检索的  "carId": 10001,  "batteryType": "三元电池",  "totalMileageKm": 1000,  "batteryHealthPercent": 95 } |

根据传入的数据对数据库进行修改

|  |
| --- |
| JSON {  "code": 200,  "msg": "OK",  "data": "车辆更新成功" } |

**C.删除车辆信息接口**

|  |
| --- |
| Plain Text DELETE http://localhost:9081/vehicles/delete/10006 |

根据传入的车架编号对相关的车辆信息进行删除

|  |
| --- |
| JSON {  "code": 200,  "msg": "OK",  "data": "车辆删除成功" } |

**D.根据车架编号查找车辆信息接口**

|  |
| --- |
| JSON GET http://localhost:9081/vehicles/find/10005 |

根据唯一车架编号查询相关信息

|  |
| --- |
| JSON {  "code": 200,  "msg": "OK",  "data": {  "id": 6,  "vid": "D27DDB805E9A4ABB",  "carId": 10005,  "batteryType": "铁锂电池",  "totalMileageKm": 10000,  "batteryHealthPercent": 100,  "createTime": "2025-06-25T21:33:11",  "updateTime": "2025-06-25T21:33:11"  } } |

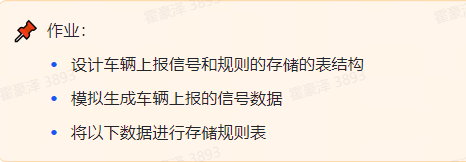
**E.查找所有车辆信息接口**

|  |
| --- |
| JSON GET http://localhost:9081/vehicles/findAll |

通过请求查看所有车辆

|  |
| --- |
| JSON {  "code": 200,  "msg": "OK",  "data": [  {  "id": 6,  "vid": "D27DDB805E9A4ABB",  "carId": 10005,  "batteryType": "铁锂电池",  "totalMileageKm": 10000,  "batteryHealthPercent": 100,  "createTime": "2025-06-25T21:33:11",  "updateTime": "2025-06-25T21:33:11"  },  {  "id": 5,  "vid": "2F22F9D42BD34513",  "carId": 10004,  "batteryType": "三元电池",  "totalMileageKm": 100,  "batteryHealthPercent": 100,  "createTime": "2025-06-25T21:32:49",  "updateTime": "2025-06-25T21:32:49"  },  {  .......  }  ] } |

**2、车辆信号上报模块**

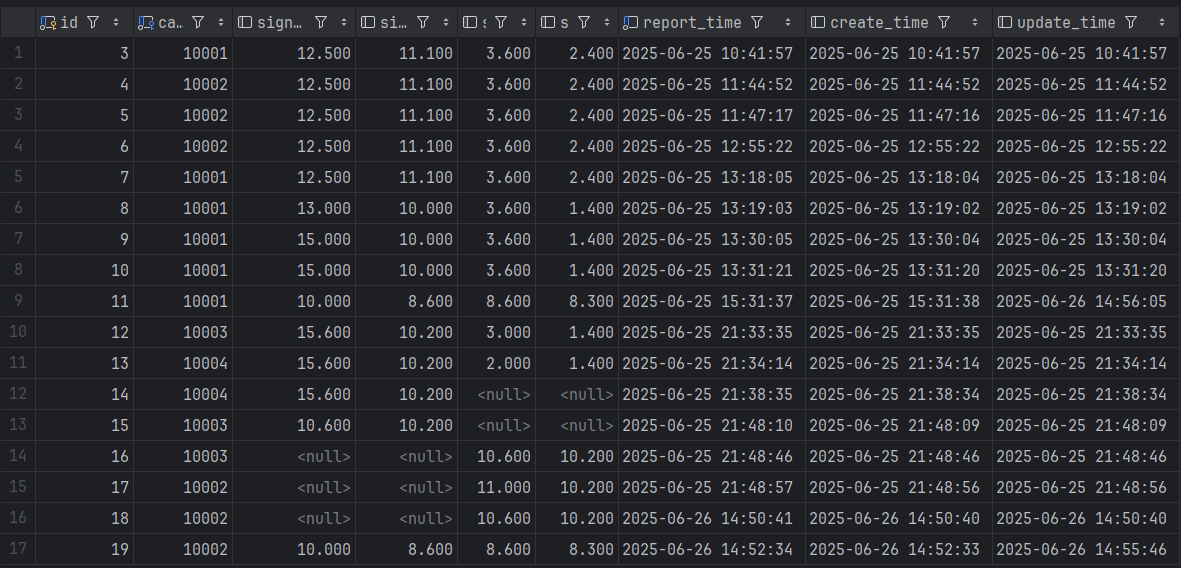


**规则**（包括：序号，规则编号，名称，预警规则，电池类型）

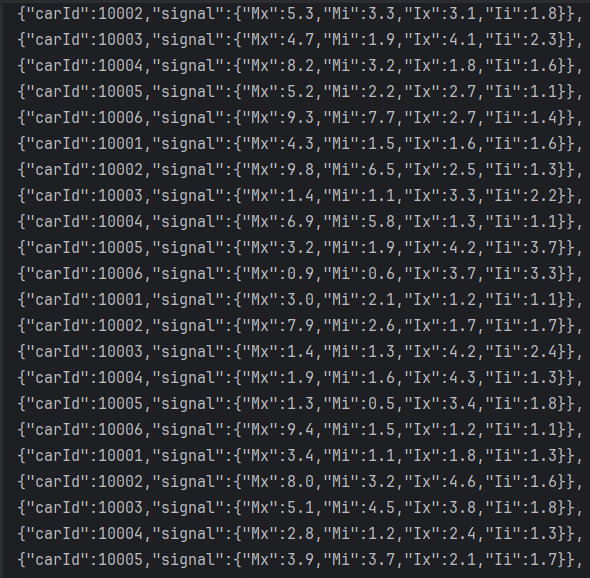
|  |
| --- |
| 预警规则：包含预警规则描述以及预警等级（0级最高响应）  电池类型：不同类型电池对应规则不同  信号：Mx（最高电压）,Mi（最小电压）、Ix（最高电流）,Ii（最小电流） |

**（1）设计车辆上报信号表**

|  |
| --- |
| JSON CREATE TABLE vehicle\_signal\_report (  id BIGINT UNSIGNED AUTO\_INCREMENT PRIMARY KEY COMMENT '主键 ID',   car\_id INT UNSIGNED NOT NULL COMMENT '车辆编号（对应 vehicle\_info.car\_id）',   signal\_mx DECIMAL(6, 3) DEFAULT NULL COMMENT '最高电压 Mx',  signal\_mi DECIMAL(6, 3) DEFAULT NULL COMMENT '最低电压 Mi',  signal\_ix DECIMAL(7, 3) DEFAULT NULL COMMENT '最大电流 Ix',  signal\_ii DECIMAL(7, 3) DEFAULT NULL COMMENT '最小电流 Ii',   report\_time DATETIME NOT NULL DEFAULT *CURRENT\_TIMESTAMP* COMMENT '上报时间',   create\_time TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP* COMMENT '创建时间',  update\_time TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP* ON UPDATE *CURRENT\_TIMESTAMP* COMMENT '更新时间',   INDEX idx\_car\_time (car\_id, report\_time),   CONSTRAINT fk\_signal\_car\_id FOREIGN KEY (car\_id)  REFERENCES vehicle\_info (car\_id) ) ENGINE = InnoDB  DEFAULT CHARSET = utf8mb4 COMMENT ='车辆上报信号表'; |



表中加入上报时间字段，可以通过定时器根据上报时间的时间差获取数据表中的数据



**模拟生成的一些车辆上报的信号数据**

**（2）设计预警规则表（序号，规则编号，名称，预警规则，电池类型，....）**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **规则编号** | **名称** | **电池类型** | **预警规则** |
| 1 | 1 | 电压差报警 | 三元电池 | 5<=(Ｍx－Mi),报警等级：0  3<=(Ｍx－Mi)<5,报警等级：1  1<=(Ｍx－Mi)<3,报警等级：2  0.6<=(Ｍx－Mi)<1,报警等级：3  0.2<=(Ｍx－Mi)<0.6,报警等级：4  (Ｍx－Mi)<0.2，不报警 |
| 2 | 1 | 电压差报警 | 铁锂电池 | 2<=(Ｍx－Mi),报警等级：0  1<=(Ｍx－Mi)<2,报警等级：1  0.7<=(Ｍx－Mi)<1,报警等级：2  0.4<=(Ｍx－Mi)<0.7,报警等级：3  0.2<=(Ｍx－Mi)<0.4,报警等级：4  (Ｍx－Mi)<0.2，不报警 |
| 3 | 2 | 电流差报警 | 三元电池 | 3<=(Ix－Ii),报警等级：0  1<=(Ix－Ii)<3,报警等级：1  0.2<=(Ix－Ii)<1,报警等级：2  (Ix－Ii)<0.2，不报警 |
| 4 | 2 | 电流差报警 | 铁锂电池 | 1<=(Ix－Ii),报警等级：0  0.5<=(Ix－Ii)<1,报警等级：1  0.2<=(Ix－Ii)<0.5,报警等级：2  (Ix－Ii)<0.2，不报警 |

|  |
| --- |
| JSON CREATE TABLE alarm\_rule\_segment (  id INT UNSIGNED AUTO\_INCREMENT PRIMARY KEY COMMENT '规则段主键',   warnId INT UNSIGNED NOT NULL COMMENT '规则编号（如 1）',  warn\_name VARCHAR(64) NOT NULL COMMENT '规则名称（如 电压差报警）',   battery\_type ENUM ('三元电池', '铁锂电池') NOT NULL COMMENT '适用电池类型',  signal\_type VARCHAR(32) NOT NULL COMMENT '信号类型（如 Mx-Mi, Ix-Ii）',   range\_min DECIMAL(7, 3) NOT NULL COMMENT '区间下限（闭区间）',  range\_max DECIMAL(7, 3) DEFAULT NULL COMMENT '区间上限（开区间，NULL 表示无上限）',   warn\_level TINYINT UNSIGNED NOT NULL COMMENT '报警等级（0 为最高响应）',  rule\_order TINYINT UNSIGNED NOT NULL COMMENT '匹配优先级（值越小优先级越高）',   create\_time TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP* COMMENT '创建时间',  update\_time TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP* ON UPDATE *CURRENT\_TIMESTAMP* COMMENT '更新时间',   INDEX idx\_rule\_lookup (warnId, battery\_type),  INDEX idx\_range (range\_min, range\_max),  INDEX idx\_rule\_full (warnId, battery\_type, signal\_type, range\_min, range\_max)  ) ENGINE = InnoDB  DEFAULT CHARSET = utf8mb4 COMMENT = '结构化预警规则段表'; |

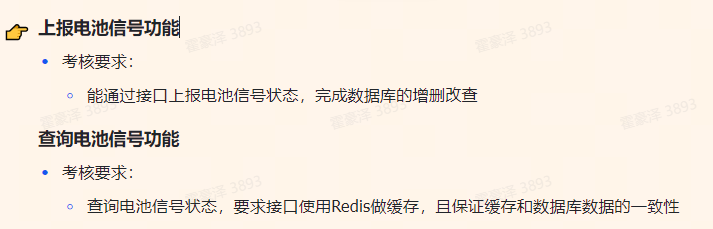
插入的规则数据为：

|  |
| --- |
| JSON -- 电压差报警：三元电池（RUL-001） INSERT INTO alarm\_rule\_segment (warnId, warn\_name, battery\_type, signal\_type, range\_min, range\_max, warn\_level, rule\_order) VALUES ('1', '电压差报警', '三元电池', 'Mx-Mi', 5.000, NULL, 0, 1),  ('1', '电压差报警', '三元电池', 'Mx-Mi', 3.000, 5.000, 1, 2),  ('1', '电压差报警', '三元电池', 'Mx-Mi', 1.000, 3.000, 2, 3),  ('1', '电压差报警', '三元电池', 'Mx-Mi', 0.600, 1.000, 3, 4),  ('1', '电压差报警', '三元电池', 'Mx-Mi', 0.200, 0.600, 4, 5); -- 不报警 (<0.2)，可不插入 -- 电压差报警：铁锂电池（1） INSERT INTO alarm\_rule\_segment (warnId, warn\_name, battery\_type, signal\_type, range\_min, range\_max, warn\_level, rule\_order) VALUES ('1', '电压差报警', '铁锂电池', 'Mx-Mi', 2.000, NULL, 0, 1),  ('1', '电压差报警', '铁锂电池', 'Mx-Mi', 1.000, 2.000, 1, 2),  ('1', '电压差报警', '铁锂电池', 'Mx-Mi', 0.700, 1.000, 2, 3),  ('1', '电压差报警', '铁锂电池', 'Mx-Mi', 0.400, 0.700, 3, 4),  ('1', '电压差报警', '铁锂电池', 'Mx-Mi', 0.200, 0.400, 4, 5);  -- 电流差报警：三元电池（2） INSERT INTO alarm\_rule\_segment (warnId, warn\_name, battery\_type, signal\_type, range\_min, range\_max, warn\_level, rule\_order) VALUES ('2', '电流差报警', '三元电池', 'Ix-Ii', 3.000, NULL, 0, 1),  ('2', '电流差报警', '三元电池', 'Ix-Ii', 1.000, 3.000, 1, 2),  ('2', '电流差报警', '三元电池', 'Ix-Ii', 0.200, 1.000, 2, 3); -- 不报警 (<0.2) 可不插  -- 电流差报警：铁锂电池（2） INSERT INTO alarm\_rule\_segment (warnId, warn\_name, battery\_type, signal\_type, range\_min, range\_max, warn\_level, rule\_order) VALUES ('2', '电流差报警', '铁锂电池', 'Ix-Ii', 1.000, NULL, 0, 1),  ('2', '电流差报警', '铁锂电池', 'Ix-Ii', 0.500, 1.000, 1, 2),  ('2', '电流差报警', '铁锂电池', 'Ix-Ii', 0.200, 0.500, 2, 3); |

表中的数据展示



**（3）接口设计**



**对请求的响应进行了统一规范的封装返回数据**

**A.上报电池信号状态接口（可以单独上报电流或电压）**

|  |
| --- |
| JSON POST http://localhost:9081/api/signal/report *Content-Type*: application/json  [  {  "carId": 10005,  "signalMx": 20.2,  "signalMi": 19,  "signalIx": 27.2,  "signalIi": 27  },  {  "carId": 10006,  "signalMx": 20.2,  "signalMi": 19  }  {  "carId": 10006,  "signalIx": 20.2,  "signalIi": 19  } ] |

返回信号上报成功的信息

|  |
| --- |
| JSON {  "code": 200,  "msg": "OK",  "data": "车辆信号上报成功" } |

**B.根据车架编号获取车辆信号接口**

|  |
| --- |
| JSON GET http://localhost:9081/api/signal/car/10002 |

获得数据表中所有关于车架编号为10002的电池状态信号，并根据上报的时间的远近返回

|  |
| --- |
| JSON {  "code": 200,  "msg": "OK",  "data": [  {  "id": 19,  "carId": 10002,  "signalMx": 10.0,  "signalMi": 8.6,  "signalIx": 8.6,  "signalIi": 8.3,  "reportTime": "2025-06-26T14:52:34",  "createTime": "2025-06-26T14:52:33",  "updateTime": "2025-06-26T14:55:46"  },  ......  {  "id": 5,  "carId": 10002,  "signalMx": 12.5,  "signalMi": 11.1,  "signalIx": 3.6,  "signalIi": 2.4,  "reportTime": "2025-06-25T11:47:17",  "createTime": "2025-06-25T11:47:16",  "updateTime": "2025-06-25T11:47:16"  },  {  "id": 4,  "carId": 10002,  "signalMx": 12.5,  "signalMi": 11.1,  "signalIx": 3.6,  "signalIi": 2.4,  "reportTime": "2025-06-25T11:44:52",  "createTime": "2025-06-25T11:44:52",  "updateTime": "2025-06-25T11:44:52"  }  ] } |

**C.车辆信号更新接口**

|  |
| --- |
| JSON PUT http://localhost:9081/api/signal/update *Content-Type*: application/json  {  "carId": 10006,  "signalMx": 20,  "signalMi": 20,  "signalIx": 20,  "signalIi": 20 } |

根据车架编号更新最近的一条车辆信号状态，并记录更新时间

|  |
| --- |
| JSON {  "code": 200,  "msg": "OK",  "data": "车辆信号更新成功" } |

**D.车辆信号删除接口**

|  |
| --- |
| JSON DELETE http://localhost:9081/api/signal/delete *Content-Type*: application/json  {  "carId": 10006 } |

根据车架编号删除最近的一条车辆信号状态

|  |
| --- |
| JSON {  "code": 200,  "msg": "OK",  "data": "车辆信号删除成功" } |

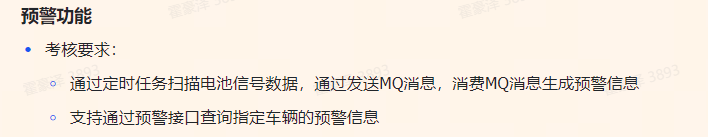
**在上传车架编号的时候，如果redis中有相同的车架编号id，则进行删除（这里采用了延迟双删）**

**在查询车辆信号数据的时候使用了redis作为缓存，如果缓存没有命中访问数据库，再将访问到的数据缓存到redis中。**

**在这些更新和删除车辆信号数据的时候，都先删除已经缓存的redis中的数据，然后再进行更新，之后再次删除缓存（延迟双删）**

通过上述做法，redis与mysql的缓存一致性得以保证。

**3、预警功能模块**



**（1）设计信号预警记录表**

|  |
| --- |
| JSON CREATE TABLE warn\_log (  id BIGINT UNSIGNED AUTO\_INCREMENT PRIMARY KEY COMMENT '主键 ID',  car\_id INT UNSIGNED NOT NULL COMMENT '车辆编号',  battery\_type VARCHAR(32) NOT NULL COMMENT '电池类型',  warn\_name VARCHAR(64) NOT NULL COMMENT '报警名称',  warn\_level TINYINT UNSIGNED NOT NULL COMMENT '报警等级',  create\_time TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP* COMMENT '创建时间',  update\_time TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP* ON UPDATE *CURRENT\_TIMESTAMP* COMMENT '更新时间',  CONSTRAINT fk\_warn\_car\_id FOREIGN KEY (car\_id)  REFERENCES vehicle\_info (car\_id),  INDEX idx\_car\_id (car\_id) ) ENGINE = InnoDB  DEFAULT CHARSET = utf8mb4 COMMENT = '预警记录表'; |



用于存储预警的消息日志

**（2）接口设计**

**A.预警功能接口**

|  |
| --- |
| JSON ### POST http://localhost:9081/api/warn *Content-Type*: application/json  [  {  "carId": 10001,  "signal": "{\"Ii\":2.4,\"Mx\":12.5,\"Mi\":11.1,\"Ix\":3.6}"  }, ......  {  "carId": 10003,  "warnId": 1,  "signal": "{\"Mx\":10.6,\"Mi\":10.2}"  },  {  "carId": 10003,  "warnId": 2,  "signal": "{\"Ii\":10.2,\"Ix\":10.3}"  },  {  "carId": 10002,  "warnId": 2,  "signal": "{\"Ii\":10.2,\"Ix\":11.0}"  } ] |

通过分析上报的数据，根据数据库（缓存/内存Map）中的规则返回数据。**这里通过mysql->redis->内存Map实现了缓存，将数据库中的信息，读取到redis和内存中，直接对信号数据进行判断。**

|  |
| --- |
| JSON {  "code": 200,  "msg": "OK",  "data": [  {  "carId": 10001,  "batteryType": "三元电池",  "warnName": "电压差报警",  "warnLevel": 2  }, ......  {  "carId": 10003,  "batteryType": "三元电池",  "warnName": "不报警"  },  {  "carId": 10002,  "batteryType": "铁锂电池",  "warnName": "电流差报警",  "warnLevel": 1  }  ] } |

**B.查询预警信息接口**

|  |
| --- |
| JSON GET http://localhost:9081/api/warn/10003 |

通过指定车辆查询预警信息，也可以放入redis中，实现方式和查询信号类似。

|  |
| --- |
| JSON {  "code": 200,  "msg": "OK",  "data": [  {  "carId": 10003,  "batteryType": "三元电池",  "warnName": "电压差报警",  "warnLevel": 0  },  {  "carId": 10003,  "batteryType": "三元电池",  "warnName": "电流差报警",  "warnLevel": 1  },  ......  {  "carId": 10003,  "batteryType": "三元电池",  "warnName": "电流差报警",  "warnLevel": 2  }  ] } |

**C.通过定时任务扫描电池信号数据，通过SocketMq发送消息，消费者消费消息**

首先导入xxl-job依赖，配置yaml文件

|  |
| --- |
| JSON *#配置调度中心属性* xxl:  job:  admin:  *#调度中心连接地址* addresses: http://127.0.0.1:8080/xxl-job-admin  *#连接调度中心 注册 发送请求 权限票据* accessToken: default\_token  executor:  *#执行器名称 admin存在一个默认执行器 需要在调度中心注册 需要ip和端口* appname: warning-info  address:  *#注册ip 本地就是空* ip:  *#注册端口 需要和admin通信* port: 21000  *#日志路径* logpath: /logs  *#日志保存天数* logretentiondays: 30 |

然后启动xxl-job-admain，通过xxl-job创建定时任务，将执行器与要调用的方法进行绑定

|  |
| --- |
| JSON @Component @Slf4j public class WarningInfoJobHandler {  @Autowired  private SignalReportProducerMapper signalReportProducerMapper;   @Autowired  private WarnMessageProducer warnMessageProducer;   @XxlJob("warningInfo")  public void sendWarningInfo() {  *log*.info("预警定时任务开始执行");  // 获取最近 1 天（24 小时）内上报的信号数据  ......   // 按 carId 分组  ......  /\*信号处理成字符串\*/  ......   // 将信号数据转换为 JSON 字符串  ......  }  // 发送预警消息  try {  ObjectMapper mapper = new ObjectMapper();  String json = mapper.writeValueAsString(warnReports);  warnMessageProducer.sendWarningMessage("warn-topic", json);  } catch (Exception e) {  *log*.error("预警消息发送失败", e);  }  } } |

数据预处理：从数据库中获取数据，然后按carId进行分组处理，这样可以高效地处理每辆车的信号数据。

生产者：最后将整个数据转为字符串传到消息队列中（数据扫描范围是根据vehicle\_signal\_report表中report\_time字段确定，范围为一天）。后面数据量很大可以控制在几秒扫描一次

|  |
| --- |
| JSON // 获取最近 1 天（24 小时）内上报的信号数据 LocalDateTime fromTime = LocalDateTime.*now*().minusDays(1); List<SignalReportProducer> recentReports = signalReportProducerMapper.selectRecentReports(fromTime); |

消费者：将接受到消息进行格式转换，调用刚刚写好的 A.预警功能接口，最终入库（Mysql）

（为了防止消息消费不及时，在项目中配置了线程池，开启多个消费者进行消费）

|  |
| --- |
| JSON @Component @Slf4j @RocketMQMessageListener(topic = "warn-topic", consumerGroup = "warn-group") public class WarnMessageConsumer implements RocketMQListener<String> {   @Autowired  private RestTemplate restTemplate; // 注入 RestTemplate   @Autowired  @Qualifier("warnExecutor") // 使用你配置的线程池  private Executor warnExecutor;   private final String targetUrl = "http://localhost:9081/api/warn";   @Override  public void onMessage(String message) {  warnExecutor.execute(() -> {  try {  // 打印接收到的原始消息  *log*.info("收到预警消息: " + message);   HttpHeaders headers = new HttpHeaders();  headers.setContentType(MediaType.*APPLICATION\_JSON*);  HttpEntity<String> requestEntity = new HttpEntity<>(message, headers);   ResponseEntity<String> response = restTemplate.postForEntity(targetUrl, requestEntity, String.class);   *log*.info("HTTP响应状态码: {}", response.getStatusCode());  *log*.info("HTTP响应内容: {}", response.getBody());  } catch (Exception e) {  *log*.error("消息处理失败: ", e);  }  });  } } |

最后也会得到HTTP响应的内容：

|  |
| --- |
| JSON {  **"code"**: **200**,  **"msg"**: **"OK"**,  **"data"**: [  {  **"carId"**: **10002**,  **"batteryType"**: **"铁锂电池"**,  **"warnName"**: **"电流差报警"**,  **"warnLevel"**: **2**  },  {  **"carId"**: **10002**,  **"batteryType"**: **"铁锂电池"**,  **"warnName"**: **"电压差报警"**,  **"warnLevel"**: **1**  }  ......  {  **"carId"**: **10005**,  **"batteryType"**: **"铁锂电池"**,  **"warnName"**: **"电流差报警"**,  **"warnLevel"**: **2**  }  ] } |

**四、技术实现**

**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**

|  |
| --- |
| JSON //电压的规则解析代码 private void checkVoltage(Integer carId, String batteryType, Map<String, Double> signal,  List<WarnResultVO> result, List<WarnLog> logs) {  Double mx = signal.get("Mx");  Double mi = signal.get("Mi");  if (mx == null || mi == null) {  *log*.warn("车辆[{}] 缺少电压数据", carId);  return;  }   double diff = Math.*round*((mx - mi) \* 1000.0) / 1000.0; // AlarmRuleSegment rule = alarmRuleSegmentMapper.findMatchedRule(batteryType, "Mx-Mi", 1, diff);  AlarmRuleSegment rule = alarmRuleCacheaRedis.findMatchedRule(batteryType, "Mx-Mi", diff);  if (rule != null) {  result.add(new WarnResultVO(carId, batteryType, rule.getWarnName(), rule.getWarnLevel()));  logs.add(new WarnLog(null, carId, batteryType, rule.getWarnName(), rule.getWarnLevel(), null, null));  } else {  result.add(new WarnResultVO(carId, batteryType, "不报警", null));  }  } |

|  |
| --- |
| JSON public AlarmRuleSegment findMatchedRule(String batteryType, String signalType, double diff) {  // 1. 先查本地内存  List<AlarmRuleSegment> rules = getFromLocalCache(batteryType, signalType);  if (rules != null && !rules.isEmpty()) {  *log*.info("已从本地缓存中匹配到规则：[{}]", batteryType + ":" + signalType);  return matchRule(rules, diff);  }   // 2. 查 Redis 并更新本地缓存  List<AlarmRuleSegment> redisRules = getFromRedis(batteryType, signalType);  if (redisRules != null && !redisRules.isEmpty()) {  *log*.info("已从 Redis 中匹配到规则：[{}]", batteryType + ":" + signalType);  putToLocalCache(batteryType, signalType, redisRules);  return matchRule(redisRules, diff);  }   // 3. 最后查数据库  List<AlarmRuleSegment> dbRules = alarmRuleSegmentMapper.findByTypeAndSignal(batteryType, signalType);  if (dbRules != null && !dbRules.isEmpty()) {  *log*.info("已从数据库中匹配到规则：[{}]", batteryType + ":" + signalType);  putToLocalCache(batteryType, signalType, dbRules);  try {  String json = objectMapper.writeValueAsString(dbRules);  redisTemplate.opsForValue().set(getRedisKey(batteryType, signalType), json, 1, TimeUnit.*HOURS*);  *log*.info("已写入 Redis 缓存：[{}]", getRedisKey(batteryType, signalType));  } catch (Exception e) {  *log*.warn("写入 Redis 缓存失败：[{}]", getRedisKey(batteryType, signalType), e);  }  return matchRule(dbRules, diff);  }   return null; } |

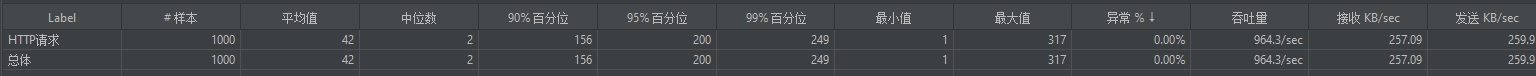
|  |
| --- |
| JSON //电流的规则解析代码 private void checkCurrent(Integer carId, String batteryType, Map<String, Double> signal,  List<WarnResultVO> result, List<WarnLog> logs) {  Double ix = signal.get("Ix");  Double ii = signal.get("Ii");  if (ix == null || ii == null) {  *log*.warn("车辆[{}] 缺少电流数据", carId);  return;  }   double diff = Math.*round*((ix - ii) \* 1000.0) / 1000.0; // AlarmRuleSegment rule = alarmRuleSegmentMapper.findMatchedRule(batteryType, "Ix-Ii", 2, diff);  AlarmRuleSegment rule = alarmRuleCacheaRedis.findMatchedRule(batteryType, "Ix-Ii", diff);  if (rule != null) {  result.add(new WarnResultVO(carId, batteryType, rule.getWarnName(), rule.getWarnLevel()));  logs.add(new WarnLog(null, carId, batteryType, rule.getWarnName(), rule.getWarnLevel(), null, null));  } else {  result.add(new WarnResultVO(carId, batteryType, "不报警", null));  }  } |

**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**

（采用jmeter工具进行分析）

**方案一：直接查询mysql数据库中的规则**

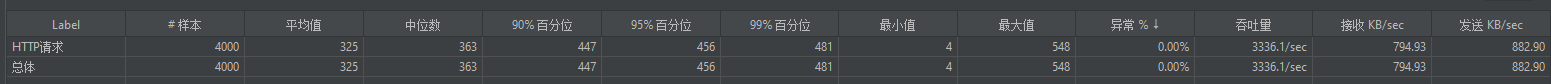
**3000并发量有极小概率会出现异常p99=249<1s**

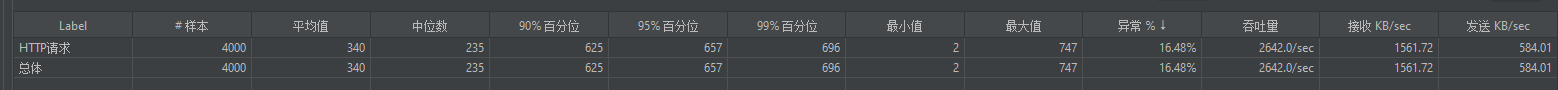






**4000 并发量有时会出现异常**







**方案二：提前将mysql数据库中的信息加载到redis缓存与本地内存中（规则信息量较少）**

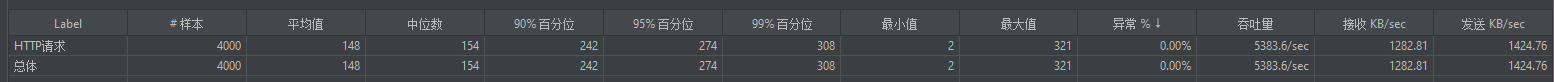
**3000并发量 保证p99=159ms<1s**





**4000并发量极少时间出现异常 仍然可以保证p99=212ms<1s**







**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。**

**数据库设计**

**1、alarm\_rule\_segment表的索引设计**

**索引设计**

* **PRIMARY KEY (id)**：保证了表的主键唯一性，并优化了基于 **id** 的查询。
* **idx\_range (range\_min, range\_max)**：这个复合索引非常适合 **范围查询**，即查询某个信号的范围（如 range\_min <= diff < range\_max）。当需要处理大量的信号数据时，这个索引可以大大提高查询效率，避免全表扫描。
* **idx\_rule\_full (warnId, batteryType, signalType, range\_min, range\_max)**：这个复合索引适合 **多条件查询**，尤其是涉及多个字段的查询（例如按 warnId、batteryType 和 signalType 查询）。
* **idx\_rule\_lookup (warnId, batteryType)**：这个索引在需要基于 **warnId** 和 **batteryType** 进行查询时提供了优化。它能够加速对这两个字段的查询，尤其是在过滤预警规则时。

**2、vehicle\_info 表的索引设计**

**索引设计**

* **PRIMARY KEY (id)**：这是必需的，确保表中的每条记录是唯一的，并且对该字段的查询非常高效。
* **UNIQUE (car\_id)**：确保每辆车有唯一的 car\_id，避免重复记录，并优化通过 car\_id 查询车辆信息的性能。
* **UNIQUE(vid)**：提高根据 vid 查询的性能，特别是当需要频繁访问车辆数据时。

**3、vehicle\_signal\_report 表的索引设计**

**索引设计**

* **PRIMARY KEY (id)**：保证每条信号报告是唯一的，并且快速查询。
* **idx\_car\_time (car\_id, report\_time)**：这个复合索引优化了按 **car\_id** 和 **report\_time** 查询信号数据的速度。它特别适合时间范围查询，例如查询特定车辆在特定时间段内的信号数据。

**4、warn\_log 表的索引设计**

**索引设计**

* **PRIMARY KEY (id)**：确保 warn\_log 表中的每条记录都是唯一的，并且在根据主键查询时非常高效。
* **idx\_car\_id (car\_id)**：优化了根据 car\_id 查询预警记录的速度。

**可以先将信号数据写入数据库（Mysql）中，然后再写入redis中，之后查redis中最近x秒的数据进行解析（易实现）**

**数据库分区（Sharding）**

* **水平分区**：将数据按照 **car\_id** 或 **时间戳（report\_time）** 等字段进行分区，有助于减少单个表的存储压力，并加快查询速度。例如，使用基于日期的分区可以有效加速时间范围查询。
* **更少的数据处理**：分区后，每个查询只涉及部分数据表，减少了数据库中需要扫描的数据量，提高查询效率。

**Elasticsearch**

**通过 ILM 策略，可以自动将不再频繁查询的数据从热存储迁移到冷存储，减少存储成本并提高数据查询效率。**