

# 考核大作业

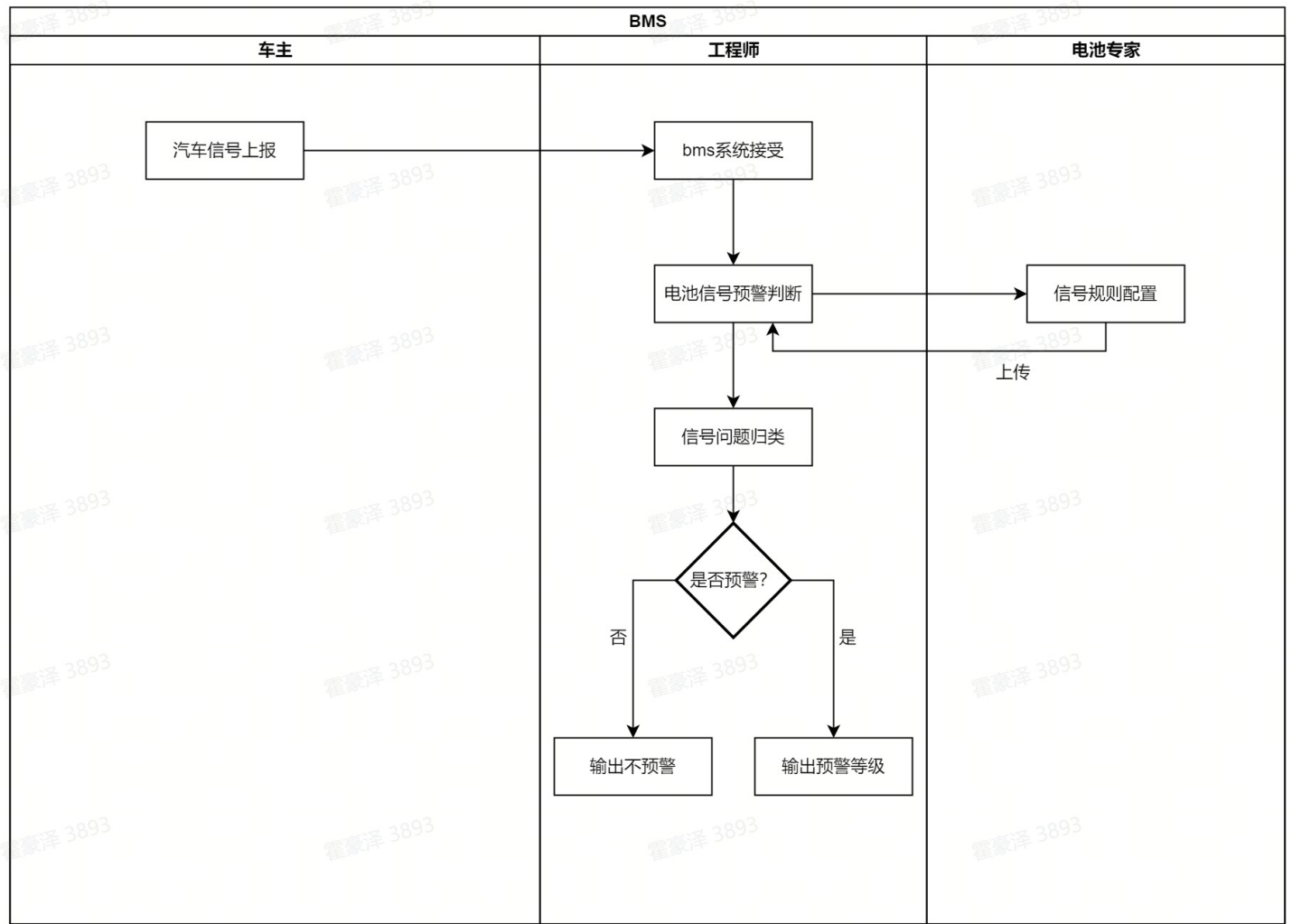
## 一、系统设计

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

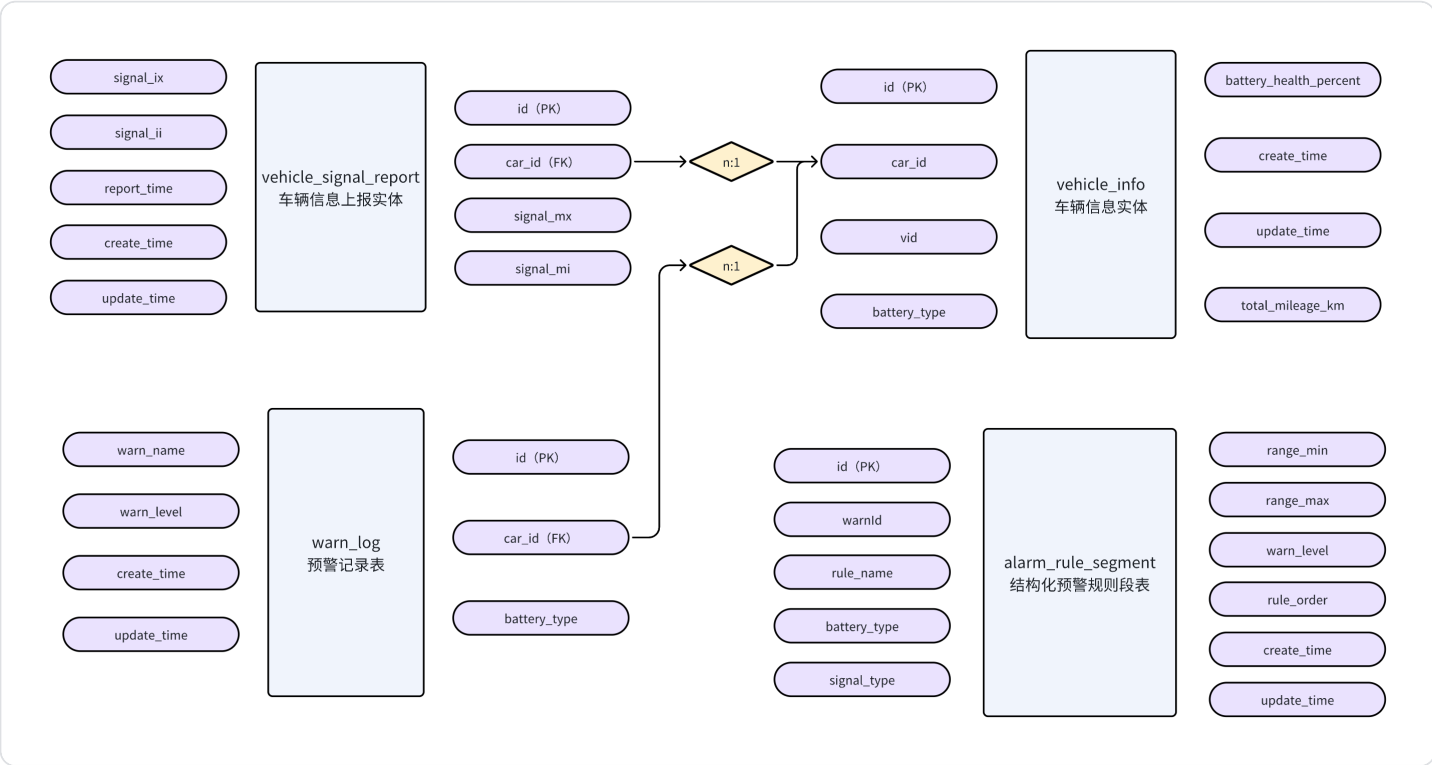


## 二、数据库设计

### 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

 **作业：**

- 设计车辆信息存储的表结构
- 将下面信息存储到汽车信息表中

(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：

```
1 CREATE TABLE vehicle_info
2 (
3     id                                INT UNSIGNED AUTO_INCREMENT PRIMARY KEY COMMENT '主
      键 ID',
4
5     vid                                CHAR(16)                                NOT NULL UNIQUE
      COMMENT '车辆识别码（16位）',
6     car_id                            INT UNSIGNED                                NOT NULL UNIQUE
      COMMENT '车辆编号（前端 carId，业务主键）',
7
8     battery_type                      ENUM ('三元电池', '铁锂电池') NOT NULL COMMENT '电池类
      型',
9     total_mileage_km                  INT UNSIGNED                                NOT NULL COMMENT '总里
      程（单位：公里）',
10    battery_health_percent             TINYINT UNSIGNED                        NOT NULL COMMENT '电池
      健康状态（0~100）',
11
12    create_time                       TIMESTAMP DEFAULT CURRENT_TIMESTAMP COMMENT '创建时
      间',
13    update_time                       TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
      CURRENT_TIMESTAMP COMMENT '更新时间',
14    INDEX idx_carId (car_id)
15 ) ENGINE = InnoDB
16 DEFAULT CHARSET = utf8mb4 COMMENT ='车辆信息表';
```

		vid	car_id	battery_type	total_mileage_km	battery_health_percent	create_time	update_time
1	1	A130B97182944724	10001	三元电池	1580	92	2025-06-25 00:2...	2025-06-26 14:31:31
2	2	E6E440D6E7E545B3	10002	铁锂电池	1200	85	2025-06-25 09:5...	2025-06-26 14:31:31
3	3	FBBDD03FB93649FC	10003	三元电池	2030	75	2025-06-25 09:5...	2025-06-26 14:31:31
4	5	2F22F9D42BD34513	10004	三元电池	100	100	2025-06-25 21:3...	2025-06-25 21:32:49
5	6	D27DDB805E9A4ABB	10005	铁锂电池	10000	100	2025-06-25 21:3...	2025-06-25 21:33:11

## (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

```
1 POST http://localhost:9081/vehicles/insert
2 Content-Type: application/json
3
```

```
4 {
5   "carId": 1,
6   "batteryType": "三元电池",
7   "totalMileageKm": 100,
8   "batteryHealthPercent": 100
9 }
```

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

```
1 {
2   "code": 200,
3   "msg": "OK",
4   "data": "车辆新增成功."
5 }
```

## B.更新车辆信息接口

```
1 PUT http://localhost:9081/vehicles/update
2 Content-Type: application/json
3
4 {
5   "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
6   "carId": 10001,
7   "batteryType": "三元电池",
8   "totalMileageKm": 1000,
9   "batteryHealthPercent": 95
10 }
```

根据传入的数据对数据库进行修改

```
1 {
2   "code": 200,
3   "msg": "OK",
4   "data": "车辆更新成功"
5 }
```

## C.删除车辆信息接口

```
1 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

```
1 {
2   "code": 200,
3   "msg": "OK",
4   "data": "车辆删除成功"
5 }
```

#### D.根据车架编号查找车辆信息接口

```
1 GET http://localhost:9081/vehicles/find/10005
```

根据唯一车架编号查询相关信息

```
1 {
2   "code": 200,
3   "msg": "OK",
4   "data": {
5     "id": 6,
6     "vid": "D27DDB805E9A4ABB",
7     "carId": 10005,
8     "batteryType": "铁锂电池",
9     "totalMileageKm": 10000,
10    "batteryHealthPercent": 100,
11    "createTime": "2025-06-25T21:33:11",
12    "updateTime": "2025-06-25T21:33:11"
13  }
14 }
```

#### E.查找所有车辆信息接口

```
1 GET http://localhost:9081/vehicles/findAll
```

通过请求查看所有车辆

```
1 {
2   "code": 200,
3   "msg": "OK",
4   "data": [
5     {
6       "id": 6,
7       "vid": "D27DDB805E9A4ABB",
8       "carId": 10005,
9       "batteryType": "铁锂电池",
10      "totalMileageKm": 10000,
11      "batteryHealthPercent": 100,
12      "createTime": "2025-06-25T21:33:11",
13      "updateTime": "2025-06-25T21:33:11"
14    },
15    {
16      "id": 5,
17      "vid": "2F22F9D42BD34513",
18      "carId": 10004,
19      "batteryType": "三元电池",
20      "totalMileageKm": 100,
21      "batteryHealthPercent": 100,
22      "createTime": "2025-06-25T21:32:49",
23      "updateTime": "2025-06-25T21:32:49"
24    },
25    {
26      .....
27    }
28  ]
29 }
```

## 2、车辆信号上报模块



作业:

- 设计车辆上报信号和规则的存储的表结构
- 模拟生成车辆上报的信号数据
- 将以下数据进行存储规则表

**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）,li（最小电流）

(1) 设计车辆上报信号表

```
1 CREATE TABLE vehicle_signal_report
2 (
3     id          BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY COMMENT '主键 ID',
4
5     car_id      INT UNSIGNED NOT NULL COMMENT '车辆编号 (对应
vehicle_info.car_id) ',
6
7     signal_mx   DECIMAL(6, 3)          DEFAULT NULL COMMENT '最高电压 Mx',
8     signal_mi   DECIMAL(6, 3)          DEFAULT NULL COMMENT '最低电压 Mi',
9     signal_ix   DECIMAL(7, 3)          DEFAULT NULL COMMENT '最大电流 Ix',
10    signal_ii   DECIMAL(7, 3)          DEFAULT NULL COMMENT '最小电流 Ii',
11
12    report_time  DATETIME                NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '上报时
间',
13
14    create_time  TIMESTAMP              DEFAULT CURRENT_TIMESTAMP COMMENT '创建时
间',
15
16    update_time  TIMESTAMP              DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP COMMENT '更新时间',
17
18    INDEX idx_car_time (car_id, report_time),
19
20    CONSTRAINT fk_signal_car_id FOREIGN KEY (car_id)
REFERENCES vehicle_info (car_id)
21 ) ENGINE = InnoDB
22 DEFAULT CHARSET = utf8mb4 COMMENT ='车辆上报信号表';
```

	id	car_id	signal_mx	signal_mi	signal_ix	signal_ii	report_time	create_time	update_time
1	3	10001	12.500	11.100	3.600	2.400	2025-06-25 10:41:57	2025-06-25 10:41:57	2025-06-25 10:41:57
2	4	10002	12.500	11.100	3.600	2.400	2025-06-25 11:44:52	2025-06-25 11:44:52	2025-06-25 11:44:52
3	5	10002	12.500	11.100	3.600	2.400	2025-06-25 11:47:17	2025-06-25 11:47:16	2025-06-25 11:47:16
4	6	10002	12.500	11.100	3.600	2.400	2025-06-25 12:55:22	2025-06-25 12:55:22	2025-06-25 12:55:22
5	7	10001	12.500	11.100	3.600	2.400	2025-06-25 13:18:05	2025-06-25 13:18:04	2025-06-25 13:18:04
6	8	10001	13.000	10.000	3.600	1.400	2025-06-25 13:19:03	2025-06-25 13:19:02	2025-06-25 13:19:02
7	9	10001	15.000	10.000	3.600	1.400	2025-06-25 13:30:05	2025-06-25 13:30:04	2025-06-25 13:30:04
8	10	10001	15.000	10.000	3.600	1.400	2025-06-25 13:31:21	2025-06-25 13:31:20	2025-06-25 13:31:20
9	11	10001	10.000	8.600	8.600	8.300	2025-06-25 15:31:37	2025-06-25 15:31:38	2025-06-26 14:56:05
10	12	10003	15.600	10.200	3.000	1.400	2025-06-25 21:33:35	2025-06-25 21:33:35	2025-06-25 21:33:35
11	13	10004	15.600	10.200	2.000	1.400	2025-06-25 21:34:14	2025-06-25 21:34:14	2025-06-25 21:34:14
12	14	10004	15.600	10.200	<null>	<null>	2025-06-25 21:38:35	2025-06-25 21:38:34	2025-06-25 21:38:34
13	15	10003	10.600	10.200	<null>	<null>	2025-06-25 21:48:10	2025-06-25 21:48:09	2025-06-25 21:48:09
14	16	10003	<null>	<null>	10.600	10.200	2025-06-25 21:48:46	2025-06-25 21:48:46	2025-06-25 21:48:46
15	17	10002	<null>	<null>	11.000	10.200	2025-06-25 21:48:57	2025-06-25 21:48:56	2025-06-25 21:48:56
16	18	10002	<null>	<null>	10.600	10.200	2025-06-26 14:50:41	2025-06-26 14:50:40	2025-06-26 14:50:40
17	19	10002	10.000	8.600	8.600	8.300	2025-06-26 14:52:34	2025-06-26 14:52:33	2025-06-26 14:55:46

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

```
{
  "carId": 10002,
  "signal": {
    "Mx": 5.3,
    "Mi": 3.3,
    "Ix": 3.1,
    "Ii": 1.8
  }
},
{
  "carId": 10003,
  "signal": {
    "Mx": 4.7,
    "Mi": 1.9,
    "Ix": 4.1,
    "Ii": 2.3
  }
},
{
  "carId": 10004,
  "signal": {
    "Mx": 8.2,
    "Mi": 3.2,
    "Ix": 1.8,
    "Ii": 1.6
  }
},
{
  "carId": 10005,
  "signal": {
    "Mx": 5.2,
    "Mi": 2.2,
    "Ix": 2.7,
    "Ii": 1.1
  }
},
{
  "carId": 10006,
  "signal": {
    "Mx": 9.3,
    "Mi": 7.7,
    "Ix": 2.7,
    "Ii": 1.4
  }
},
{
  "carId": 10001,
  "signal": {
    "Mx": 4.3,
    "Mi": 1.5,
    "Ix": 1.6,
    "Ii": 1.6
  }
},
{
  "carId": 10002,
  "signal": {
    "Mx": 9.8,
    "Mi": 6.5,
    "Ix": 2.5,
    "Ii": 1.3
  }
},
{
  "carId": 10003,
  "signal": {
    "Mx": 1.4,
    "Mi": 1.1,
    "Ix": 3.3,
    "Ii": 2.2
  }
},
{
  "carId": 10004,
  "signal": {
    "Mx": 6.9,
    "Mi": 5.8,
    "Ix": 1.3,
    "Ii": 1.1
  }
},
{
  "carId": 10005,
  "signal": {
    "Mx": 3.2,
    "Mi": 1.9,
    "Ix": 4.2,
    "Ii": 3.7
  }
},
{
  "carId": 10006,
  "signal": {
    "Mx": 0.9,
    "Mi": 0.6,
    "Ix": 3.7,
    "Ii": 3.3
  }
},
{
  "carId": 10001,
  "signal": {
    "Mx": 3.0,
    "Mi": 2.1,
    "Ix": 1.2,
    "Ii": 1.1
  }
},
{
  "carId": 10002,
  "signal": {
    "Mx": 7.9,
    "Mi": 2.6,
    "Ix": 1.7,
    "Ii": 1.7
  }
},
{
  "carId": 10003,
  "signal": {
    "Mx": 1.4,
    "Mi": 1.3,
    "Ix": 4.2,
    "Ii": 2.4
  }
},
{
  "carId": 10004,
  "signal": {
    "Mx": 1.9,
    "Mi": 1.6,
    "Ix": 4.3,
    "Ii": 1.3
  }
},
{
  "carId": 10005,
  "signal": {
    "Mx": 1.3,
    "Mi": 0.5,
    "Ix": 3.4,
    "Ii": 1.8
  }
},
{
  "carId": 10006,
  "signal": {
    "Mx": 9.4,
    "Mi": 1.5,
    "Ix": 1.2,
    "Ii": 1.1
  }
},
{
  "carId": 10001,
  "signal": {
    "Mx": 3.4,
    "Mi": 1.1,
    "Ix": 1.8,
    "Ii": 1.3
  }
},
{
  "carId": 10002,
  "signal": {
    "Mx": 8.0,
    "Mi": 3.2,
    "Ix": 4.6,
    "Ii": 1.6
  }
},
{
  "carId": 10003,
  "signal": {
    "Mx": 5.1,
    "Mi": 4.5,
    "Ix": 3.8,
    "Ii": 1.8
  }
},
{
  "carId": 10004,
  "signal": {
    "Mx": 2.8,
    "Mi": 1.2,
    "Ix": 2.4,
    "Ii": 1.3
  }
},
{
  "carId": 10005,
  "signal": {
    "Mx": 3.9,
    "Mi": 3.7,
    "Ix": 2.1,
    "Ii": 1.7
  }
},
```

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	$3 \leq (I_x - I_i)$ ,报警等级：0 $1 \leq (I_x - I_i) < 3$ ,报警等级：1 $0.2 \leq (I_x - I_i) < 1$ ,报警等级：2 $(I_x - I_i) < 0.2$ ，不报警
4	2	电流差报警	铁锂电池	$1 \leq (I_x - I_i)$ ,报警等级：0 $0.5 \leq (I_x - I_i) < 1$ ,报警等级：1 $0.2 \leq (I_x - I_i) < 0.5$ ,报警等级：2 $(I_x - I_i) < 0.2$ ，不报警

```

1 CREATE TABLE alarm_rule_segment
2 (
3     id                INT UNSIGNED AUTO_INCREMENT PRIMARY KEY COMMENT '规则段主键',
4
5     warnId INT UNSIGNED                                NOT NULL COMMENT '规则编号 (如 1)',
6     warn_name        VARCHAR(64)                      NOT NULL COMMENT '规则名称 (如
电压差报警)',
7
8     battery_type      ENUM ('三元电池', '铁锂电池') NOT NULL COMMENT '适用电池类型',
9     signal_type       VARCHAR(32)                    NOT NULL COMMENT '信号类型 (如
Mx-Mi, Ix-Ii)',
10
11     range_min         DECIMAL(7, 3)                  NOT NULL COMMENT '区间下限 (闭
区间)',
12     range_max         DECIMAL(7, 3) DEFAULT NULL COMMENT '区间上限 (开区间, NULL 表
示无上限)',
13
14     warn_level        TINYINT UNSIGNED               NOT NULL COMMENT '报警等级 (0
为最高响应)',
15     rule_order        TINYINT UNSIGNED               NOT NULL COMMENT '匹配优先级
(值越小优先级越高)',
16
17     create_time       TIMESTAMP DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
18     update_time       TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP COMMENT '更新时间',
19
20     INDEX idx_rule_lookup (warnId, battery_type),
21     INDEX idx_range (range_min, range_max),
22     INDEX idx_rule_full (warnId, battery_type, signal_type, range_min,
range_max)
23
24 ) ENGINE = InnoDB
25 DEFAULT CHARSET = utf8mb4 COMMENT = '结构化预警规则段表';

```

插入的规则数据为：

```
1 -- 电压差报警：三元电池 (RUL-001)
2 INSERT INTO alarm_rule_segment
3 (warnId, warn_name, battery_type, signal_type, range_min, range_max,
4  warn_level, rule_order)
5 VALUES ('1', '电压差报警', '三元电池', 'Mx-Mi', 5.000, NULL, 0, 1),
6         ('1', '电压差报警', '三元电池', 'Mx-Mi', 3.000, 5.000, 1, 2),
7         ('1', '电压差报警', '三元电池', 'Mx-Mi', 1.000, 3.000, 2, 3),
8         ('1', '电压差报警', '三元电池', 'Mx-Mi', 0.600, 1.000, 3, 4),
9         ('1', '电压差报警', '三元电池', 'Mx-Mi', 0.200, 0.600, 4, 5);
9 -- 不报警 (<0.2), 可不插入
10 -- 电压差报警：铁锂电池 (1)
11 INSERT INTO alarm_rule_segment
12 (warnId, warn_name, battery_type, signal_type, range_min, range_max,
13  warn_level, rule_order)
14 VALUES ('1', '电压差报警', '铁锂电池', 'Mx-Mi', 2.000, NULL, 0, 1),
15         ('1', '电压差报警', '铁锂电池', 'Mx-Mi', 1.000, 2.000, 1, 2),
16         ('1', '电压差报警', '铁锂电池', 'Mx-Mi', 0.700, 1.000, 2, 3),
17         ('1', '电压差报警', '铁锂电池', 'Mx-Mi', 0.400, 0.700, 3, 4),
18         ('1', '电压差报警', '铁锂电池', 'Mx-Mi', 0.200, 0.400, 4, 5);
18
19 -- 电流差报警：三元电池 (2)
20 INSERT INTO alarm_rule_segment
21 (warnId, warn_name, battery_type, signal_type, range_min, range_max,
22  warn_level, rule_order)
23 VALUES ('2', '电流差报警', '三元电池', 'Ix-Ii', 3.000, NULL, 0, 1),
24         ('2', '电流差报警', '三元电池', 'Ix-Ii', 1.000, 3.000, 1, 2),
25         ('2', '电流差报警', '三元电池', 'Ix-Ii', 0.200, 1.000, 2, 3);
25 -- 不报警 (<0.2) 可不插
26
27 -- 电流差报警：铁锂电池 (2)
28 INSERT INTO alarm_rule_segment
29 (warnId, warn_name, battery_type, signal_type, range_min, range_max,
30  warn_level, rule_order)
31 VALUES ('2', '电流差报警', '铁锂电池', 'Ix-Ii', 1.000, NULL, 0, 1),
32         ('2', '电流差报警', '铁锂电池', 'Ix-Ii', 0.500, 1.000, 1, 2),
33         ('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

i		warn...	ba...				wa.		create_time	update_time
1	1	电压差报警	三元电池	Mx-Mi	5.000	<null>	0	1	2025-06-24 23:33:04	2025-06-24 23:33:04
2	1	电压差报警	三元电池	Mx-Mi	3.000	5.000	1	2	2025-06-24 23:33:04	2025-06-24 23:33:04
3	1	电压差报警	三元电池	Mx-Mi	1.000	3.000	2	3	2025-06-24 23:33:04	2025-06-24 23:33:04
4	1	电压差报警	三元电池	Mx-Mi	0.600	1.000	3	4	2025-06-24 23:33:04	2025-06-24 23:33:04
5	1	电压差报警	三元电池	Mx-Mi	0.200	0.600	4	5	2025-06-24 23:33:04	2025-06-24 23:33:04
6	1	电压差报警	铁锂电池	Mx-Mi	2.000	<null>	0	1	2025-06-24 23:33:05	2025-06-24 23:33:05
7	1	电压差报警	铁锂电池	Mx-Mi	1.000	2.000	1	2	2025-06-24 23:33:05	2025-06-24 23:33:05
8	1	电压差报警	铁锂电池	Mx-Mi	0.700	1.000	2	3	2025-06-24 23:33:05	2025-06-24 23:33:05
9	1	电压差报警	铁锂电池	Mx-Mi	0.400	0.700	3	4	2025-06-24 23:33:05	2025-06-24 23:33:05
10	1	电压差报警	铁锂电池	Mx-Mi	0.200	0.400	4	5	2025-06-24 23:33:05	2025-06-24 23:33:05
11	2	电流差报警	三元电池	Ix-Ii	3.000	<null>	0	1	2025-06-24 23:33:05	2025-06-24 23:33:05
12	2	电流差报警	三元电池	Ix-Ii	1.000	3.000	1	2	2025-06-24 23:33:05	2025-06-24 23:33:05
13	2	电流差报警	三元电池	Ix-Ii	0.200	1.000	2	3	2025-06-24 23:33:05	2025-06-24 23:33:05
14	2	电流差报警	铁锂电池	Ix-Ii	1.000	<null>	0	1	2025-06-24 23:33:05	2025-06-24 23:33:05
15	2	电流差报警	铁锂电池	Ix-Ii	0.500	1.000	1	2	2025-06-24 23:33:05	2025-06-24 23:33:05
16	2	电流差报警	铁锂电池	Ix-Ii	0.200	0.500	2	3	2025-06-24 23:33:05	2025-06-24 23:33:05

(3) 接口设计

👉 上报电池信号功能

- 考核要求：
  - 能通过接口上报电池信号状态，完成数据库的增删改查

查询电池信号功能

- 考核要求：
  - 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

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）

```
1 POST http://localhost:9081/api/signal/report
2 Content-Type: application/json
3
4 [
5   {
6     "carId": 10005,
7     "signalMx": 20.2,
8     "signalMi": 19,
9     "signalIx": 27.2,
10    "signalIi": 27
11  },
12  {
13    "carId": 10006,
14    "signalMx": 20.2,
15    "signalMi": 19
16  }
17  {
18    "carId": 10006,
19    "signalIx": 20.2,
20    "signalIi": 19
21  }
22 ]
```

返回信号上报成功的信息

```
1 {
2   "code": 200,
3   "msg": "OK",
4   "data": "车辆信号上报成功"
5 }
```

## B.根据车架编号获取车辆信号接口

```
1 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

```
1 {
2   "code": 200,
3   "msg": "OK",
```

```
4  "data": [  
5    {  
6      "id": 19,  
7      "carId": 10002,  
8      "signalMx": 10.0,  
9      "signalMi": 8.6,  
10     "signalIx": 8.6,  
11     "signalIi": 8.3,  
12     "reportTime": "2025-06-26T14:52:34",  
13     "createTime": "2025-06-26T14:52:33",  
14     "updateTime": "2025-06-26T14:55:46"  
15   },  
16   .....  
17   {  
18     "id": 5,  
19     "carId": 10002,  
20     "signalMx": 12.5,  
21     "signalMi": 11.1,  
22     "signalIx": 3.6,  
23     "signalIi": 2.4,  
24     "reportTime": "2025-06-25T11:47:17",  
25     "createTime": "2025-06-25T11:47:16",  
26     "updateTime": "2025-06-25T11:47:16"  
27   },  
28   {  
29     "id": 4,  
30     "carId": 10002,  
31     "signalMx": 12.5,  
32     "signalMi": 11.1,  
33     "signalIx": 3.6,  
34     "signalIi": 2.4,  
35     "reportTime": "2025-06-25T11:44:52",  
36     "createTime": "2025-06-25T11:44:52",  
37     "updateTime": "2025-06-25T11:44:52"  
38   }  
39 ]  
40 }
```

## C.车辆信号更新接口

```
1  PUT http://localhost:9081/api/signal/update  
2  Content-Type: application/json  
3  
4  {  
5    "carId": 10006,
```

```
6  "signalMx": 20,
7  "signalMi": 20,
8  "signalIx": 20,
9  "signalIi": 20
10 }
```

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

```
1 {
2   "code": 200,
3   "msg": "OK",
4   "data": "车辆信号更新成功"
5 }
```

## D.车辆信号删除接口

```
1 DELETE http://localhost:9081/api/signal/delete
2 Content-Type: application/json
3
4 {
5   "carId": 10006
6 }
```

根据车架编号删除最近的一条车辆信号状态

```
1 {
2   "code": 200,
3   "msg": "OK",
4   "data": "车辆信号删除成功"
5 }
```

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

#### 预警功能

- 考核要求：
  - 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) 设计信号预警记录表

```
1 CREATE TABLE warn_log
2 (
3     id          BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY COMMENT '主键 ID',
4     car_id      INT UNSIGNED      NOT NULL COMMENT '车辆编号',
5     battery_type VARCHAR(32)      NOT NULL COMMENT '电池类型',
6     warn_name   VARCHAR(64)      NOT NULL COMMENT '报警名称',
7     warn_level  TINYINT UNSIGNED NOT NULL COMMENT '报警等级',
8     create_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
9     update_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
10    CURRENT_TIMESTAMP COMMENT '更新时间',
11    CONSTRAINT fk_warn_car_id FOREIGN KEY (car_id)
12    REFERENCES vehicle_info (car_id),
13    INDEX idx_car_id (car_id)
14 ) ENGINE = InnoDB
15 DEFAULT CHARSET = utf8mb4 COMMENT = '预警记录表';
```

id	carId	ba.	war...		create_time	update_time
83	10001	三元电池	电压差报警	2	2025-06-25 22:16:22	2025-06-25 22:16:22
84	10001	三元电池	电流差报警	1	2025-06-25 22:16:22	2025-06-25 22:16:22
85	10002	铁锂电池	电压差报警	1	2025-06-25 22:16:22	2025-06-25 22:16:22
86	10002	铁锂电池	电流差报警	0	2025-06-25 22:16:22	2025-06-25 22:16:22
87	10002	铁锂电池	电压差报警	1	2025-06-25 22:16:22	2025-06-25 22:16:22
88	10002	铁锂电池	电流差报警	0	2025-06-25 22:16:22	2025-06-25 22:16:22
89	10002	铁锂电池	电压差报警	1	2025-06-25 22:16:22	2025-06-25 22:16:22
90	10002	铁锂电池	电流差报警	0	2025-06-25 22:16:22	2025-06-25 22:16:22
91	10001	三元电池	电压差报警	2	2025-06-25 22:16:22	2025-06-25 22:16:22
92	10001	三元电池	电流差报警	1	2025-06-25 22:16:22	2025-06-25 22:16:22
93	10001	三元电池	电压差报警	1	2025-06-25 22:16:22	2025-06-25 22:16:22
94	10001	三元电池	电流差报警	1	2025-06-25 22:16:22	2025-06-25 22:16:22
95	10001	三元电池	电压差报警	0	2025-06-25 22:16:22	2025-06-25 22:16:22
96	10001	三元电池	电流差报警	1	2025-06-25 22:16:22	2025-06-25 22:16:22
97	10001	三元电池	电压差报警	0	2025-06-25 22:16:22	2025-06-25 22:16:22
98	10001	三元电池	电流差报警	1	2025-06-25 22:16:22	2025-06-25 22:16:22
99	10001	三元电池	电压差报警	0	2025-06-25 22:16:22	2025-06-25 22:16:22

用于存储预警的消息日志

## (2) 接口设计

### A.预警功能接口

```
1 ###
2 POST http://localhost:9081/api/warn
3 Content-Type: application/json
4
5 [
6   {
7     "carId": 10001,
8     "signal": "{\"Ii\":2.4,\"Mx\":12.5,\"Mi\":11.1,\"Ix\":3.6}"
9   },
10  ....
11  {
12    "carId": 10003,
13    "warnId": 1,
14    "signal": "{\"Mx\":10.6,\"Mi\":10.2}"
15  },
16  {
17    "carId": 10003,
18    "warnId": 2,
```



```
19     "signal": "{\"Ii\":10.2,\"Ix\":10.3}"
20   },
21   {
22     "carId": 10002,
23     "warnId": 2,
24     "signal": "{\"Ii\":10.2,\"Ix\":11.0}"
25   }
26 ]
```

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。

```
1 {
2   "code": 200,
3   "msg": "OK",
4   "data": [
5     {
6       "carId": 10001,
7       "batteryType": "三元电池",
8       "warnName": "电压差报警",
9       "warnLevel": 2
10    },
11    .....
12    {
13      "carId": 10003,
14      "batteryType": "三元电池",
15      "warnName": "不报警"
16    },
17    {
18      "carId": 10002,
19      "batteryType": "铁锂电池",
20      "warnName": "电流差报警",
21      "warnLevel": 1
22    }
23  ]
24 }
```

## B.查询预警信息接口

```
1 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。

```
1 {
2   "code": 200,
3   "msg": "OK",
4   "data": [
5     {
6       "carId": 10003,
7       "batteryType": "三元电池",
8       "warnName": "电压差报警",
9       "warnLevel": 0
10    },
11    {
12      "carId": 10003,
13      "batteryType": "三元电池",
14      "warnName": "电流差报警",
15      "warnLevel": 1
16    },
17    .....
18    {
19      "carId": 10003,
20      "batteryType": "三元电池",
21      "warnName": "电流差报警",
22      "warnLevel": 2
23    }
24  ]
25 }
```

### 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

```
1 #配置调度中心属性
2 xxl:
3   job:
4     admin:
5       #调度中心连接地址
6       addresses: http://127.0.0.1:8080/xxl-job-admin
7       #连接调度中心 注册 发送请求 权限票据
8       accessToken: default_token
9     executor:
10      #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
11      appname: warning-info
12      address:
13        #注册ip 本地就是空
```

```
14      ip:
15      #注册端口 需要和admin通信
16      port: 21000
17      #日志路径
18      logpath: /logs
19      #日志保存天数
20      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

```
1  @Component
2  @Slf4j
3  public class WarningInfoJobHandler {
4      @Autowired
5      private SignalReportProducerMapper signalReportProducerMapper;
6
7      @Autowired
8      private WarnMessageProducer warnMessageProducer;
9
10     @XxlJob("warningInfo")
11     public void sendWarningInfo() {
12         log.info("预警定时任务开始执行");
13         // 获取最近 1 天（24 小时）内上报的信号数据
14         .....
15
16         // 按 carId 分组
17         .....
18         /*信号处理成字符串*/
19         .....
20
21         // 将信号数据转换为 JSON 字符串
22         .....
23     }
24     // 发送预警消息
25     try {
26         ObjectMapper mapper = new ObjectMapper();
27         String json = mapper.writeValueAsString(warnReports);
28         warnMessageProducer.sendWarningMessage("warn-topic", json);
29     } catch (Exception e) {
30         log.error("预警消息发送失败", e);
31     }
32 }
33 }
```

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

```
1 // 获取最近 1 天 (24 小时) 内上报的信号数据
2 LocalDateTime fromTime = LocalDateTime.now().minusDays(1);
3 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）

```
1 @Component
2 @Slf4j
3 @RocketMQMessageListener(topic = "warn-topic", consumerGroup = "warn-group")
4 public class WarnMessageConsumer implements RocketMQListener<String> {
5
6     @Autowired
7     private RestTemplate restTemplate; // 注入 RestTemplate
8
9     @Autowired
10    @Qualifier("warnExecutor") // 使用你配置的线程池
11    private Executor warnExecutor;
12
13    private final String targetUrl = "http://localhost:9081/api/warn";
14
15    @Override
16    public void onMessage(String message) {
17        warnExecutor.execute(() -> {
18            try {
19                // 打印接收到的原始消息
20                log.info("收到预警消息: " + message);
21
22                HttpHeaders headers = new HttpHeaders();
23                headers.setContentType(MediaType.APPLICATION_JSON);
24                HttpEntity<String> requestEntity = new HttpEntity<>(message,
25                    headers);
26
27                ResponseEntity<String> response =
28                    restTemplate.postForEntity(targetUrl, requestEntity, String.class);
29
30                log.info("HTTP响应状态码: {}", response.getStatusCode());
```

```

29         log.info("HTTP响应内容: {}", response.getBody());
30     } catch (Exception e) {
31         log.error("消息处理失败: ", e);
32     }
33     });
34 }
35 }

```

最后也会得到HTTP响应的内容：

```

1  {
2      "code": 200,
3      "msg": "OK",
4      "data": [
5          {
6              "carId": 10002,
7              "batteryType": "铁锂电池",
8              "warnName": "电流差报警",
9              "warnLevel": 2
10         },
11         {
12             "carId": 10002,
13             "batteryType": "铁锂电池",
14             "warnName": "电压差报警",
15             "warnLevel": 1
16         }
17         .....
18         {
19             "carId": 10005,
20             "batteryType": "铁锂电池",
21             "warnName": "电流差报警",
22             "warnLevel": 2
23         }
24     ]
25 }

```

## 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

```

1 //电压的规则解析代码

```

```

2 private void checkVoltage(Integer carId, String batteryType, Map<String,
  Double> signal,
3
4         List<WarnResultVO> result, List<WarnLog> logs) {
5     Double mx = signal.get("Mx");
6     Double mi = signal.get("Mi");
7     if (mx == null || mi == null) {
8         log.warn("车辆[{}] 缺少电压数据", carId);
9         return;
10    }
11
12    double diff = Math.round((mx - mi) * 1000.0) / 1000.0;
13    // AlarmRuleSegment rule =
14    alarmRuleSegmentMapper.findMatchedRule(batteryType, "Mx-Mi", 1, diff);
15    AlarmRuleSegment rule =
16    alarmRuleCacheRedis.findMatchedRule(batteryType, "Mx-Mi", diff);
17    if (rule != null) {
18        result.add(new WarnResultVO(carId, batteryType, rule.getWarnName(),
19        rule.getWarnLevel()));
20        logs.add(new WarnLog(null, carId, batteryType, rule.getWarnName(),
21        rule.getWarnLevel(), null, null));
22    } else {
23        result.add(new WarnResultVO(carId, batteryType, "不报警", null));
24    }
25 }

```

```

1 public AlarmRuleSegment findMatchedRule(String batteryType, String signalType,
  double diff) {
2     // 1. 先查本地内存
3     List<AlarmRuleSegment> rules = getFromLocalCache(batteryType, signalType);
4     if (rules != null && !rules.isEmpty()) {
5         log.info("已从本地缓存中匹配到规则: [{}]", batteryType + ":" + signalType);
6         return matchRule(rules, diff);
7     }
8
9     // 2. 查 Redis 并更新本地缓存
10    List<AlarmRuleSegment> redisRules = getFromRedis(batteryType, signalType);
11    if (redisRules != null && !redisRules.isEmpty()) {
12        log.info("已从 Redis 中匹配到规则: [{}]", batteryType + ":" + signalType);
13        putToLocalCache(batteryType, signalType, redisRules);
14        return matchRule(redisRules, diff);
15    }
16
17    // 3. 最后查数据库
18    List<AlarmRuleSegment> dbRules =
19    alarmRuleSegmentMapper.findByTypeAndSignal(batteryType, signalType);

```

```

19     if (dbRules != null && !dbRules.isEmpty()) {
20         log.info("已从数据库中匹配到规则: [{}]", batteryType + ":" + signalType);
21         putToLocalCache(batteryType, signalType, dbRules);
22         try {
23             String json = objectMapper.writeValueAsString(dbRules);
24             redisTemplate.opsForValue().set(getRedisKey(batteryType,
signalType), json, 1, TimeUnit.HOURS);
25             log.info("已写入 Redis 缓存: [{}]", getRedisKey(batteryType,
signalType));
26         } catch (Exception e) {
27             log.warn("写入 Redis 缓存失败: [{}]", getRedisKey(batteryType,
signalType), e);
28         }
29         return matchRule(dbRules, diff);
30     }
31
32     return null;
33 }

```

```

1 //电流的规则解析代码
2 private void checkCurrent(Integer carId, String batteryType, Map<String,
Double> signal,
3                             List<WarnResultVO> result, List<WarnLog> logs) {
4     Double ix = signal.get("Ix");
5     Double ii = signal.get("Ii");
6     if (ix == null || ii == null) {
7         log.warn("车辆[{}] 缺少电流数据", carId);
8         return;
9     }
10
11     double diff = Math.round((ix - ii) * 1000.0) / 1000.0;
12 //     AlarmRuleSegment rule =
alarmRuleSegmentMapper.findMatchedRule(batteryType, "Ix-Ii", 2, diff);
13     AlarmRuleSegment rule =
alarmRuleCacheaRedis.findMatchedRule(batteryType, "Ix-Ii", diff);
14     if (rule != null) {
15         result.add(new WarnResultVO(carId, batteryType, rule.getWarnName(),
rule.getWarnLevel()));
16         logs.add(new WarnLog(null, carId, batteryType, rule.getWarnName(),
rule.getWarnLevel(), null, null));
17     } else {
18         result.add(new WarnResultVO(carId, batteryType, "不报警", null));
19     }
20 }

```

## 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

Label	# 样本	平均值	中位数	90% 百分位	95% 百分位	99% 百分位	最小值	最大值	异常 % ↓	吞吐量	接收 KB/sec	发送 KB/sec
HTTP请求	1000	42	2	156	200	249	1	317	0.00%	964.3/sec	257.09	259.9
总体	1000	42	2	156	200	249	1	317	0.00%	964.3/sec	257.09	259.9

Label	# 样本	平均值	中位数	90% 百分位	95% 百分位	99% 百分位	最小值	最大值	异常 % ↓	吞吐量	接收 KB/sec	发送 KB/sec
HTTP请求	3000	214	253	279	285	299	4	335	0.00%	3921.6/sec	934.44	1037.84
总体	3000	214	253	279	285	299	4	335	0.00%	3921.6/sec	934.44	1037.84

Label	# 样本	平均值	中位数	90% 百分位	95% 百分位	99% 百分位	最小值	最大值	异常 % ↓	吞吐量	接收 KB/sec	发送 KB/sec
HTTP请求	3000	262	316	372	406	428	3	486	0.00%	3003.0/sec	715.56	794.74
总体	3000	262	316	372	406	428	3	486	0.00%	3003.0/sec	715.56	794.74

4000 并发量有时会出现异常

Label	# 样本	平均值	中位数	90% 百分位	95% 百分位	99% 百分位	最小值	最大值	异常 % ↓	吞吐量	接收 KB/sec	发送 KB/sec
HTTP请求	4000	325	363	447	456	481	4	548	0.00%	3336.1/sec	794.93	882.90
总体	4000	325	363	447	456	481	4	548	0.00%	3336.1/sec	794.93	882.90

Label	# 样本	平均值	中位数	90% 百分位	95% 百分位	99% 百分位	最小值	最大值	异常 % ↓	吞吐量	接收 KB/sec	发送 KB/sec
HTTP请求	4000	340	235	625	657	696	2	747	16.48%	2642.0/sec	1561.72	584.01
总体	4000	340	235	625	657	696	2	747	16.48%	2642.0/sec	1561.72	584.01

Label	# 样本	平均值	中位数	90% 百分位	95% 百分位	99% 百分位	最小值	最大值	异常 % ↓	吞吐量	接收 KB/sec	发送 KB/sec
HTTP请求	4000	275	289	391	403	429	2	477	0.00%	3636.4/sec	866.48	962.36
总体	4000	275	289	391	403	429	2	477	0.00%	3636.4/sec	866.48	962.36

### 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

Label	# 样本	平均值	中位数	90% 百分位	95% 百分位	99% 百分位	最小值	最大值	异常 % ↓	吞吐量	接收 KB/sec	发送 KB/sec
HTTP请求	3000	106	110	149	156	159	1	161	0.00%	5725.2/sec	1364.21	1515.16
总体	3000	106	110	149	156	159	1	161	0.00%	5725.2/sec	1364.21	1515.16

Label	# 样本	平均值	中位数	90% 百分位	95% 百分位	99% 百分位	最小值	最大值	异常 % ↓	吞吐量	接收 KB/sec	发送 KB/sec
HTTP请求	3000	103	110	144	151	162	2	165	0.00%	5494.5/sec	1309.24	1454.11
总体	3000	103	110	144	151	162	2	165	0.00%	5494.5/sec	1309.24	1454.11

4000并发量极少时间出现异常 仍然可以保证p99=212ms<1s

Label	# 样本	平均值	中位数	90% 百分位	95% 百分位	99% 百分位	最小值	最大值	异常 % ↓	吞吐量	接收 KB/sec	发送 KB/sec
HTTP请求	4000	182	206	276	311	369	3	389	0.00%	4603.0/sec	1096.81	1218.17
总体	4000	182	206	276	311	369	3	389	0.00%	4603.0/sec	1096.81	1218.17

Label	# 样本	平均值	中位数	90% 百分位	95% 百分位	99% 百分位	最小值	最大值	异常 % ↓	吞吐量	接收 KB/sec	发送 KB/sec
HTTP请求	4000	148	154	242	274	308	2	321	0.00%	5383.6/sec	1282.81	1424.76
总体	4000	148	154	242	274	308	2	321	0.00%	5383.6/sec	1282.81	1424.76

Label	# 样本	平均值	中位数	90% 百分位	95% 百分位	99% 百分位	最小值	最大值	异常 % ↓	吞吐量	接收 KB/sec	发送 KB/sec
HTTP请求	4000	146	163	202	206	212	2	214	0.00%	5161.3/sec	1229.84	1365.93
总体	4000	146	163	202	206	212	2	214	0.00%	5161.3/sec	1229.84	1365.93

## 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。