大作业——刘博成

一.github代码：https://github.com/haiya23/bsm.git

二.数据库表设计

```sql
drop table if exists vehicle_info;
CREATE TABLE `vehicle_info` (
                                `id` bigint NOT NULL AUTO_INCREMENT,
                                `vid` varchar(16) NOT NULL COMMENT '车辆识别码',
                                `vin` int NOT NULL COMMENT '车架编号',
                                `battery_type` enum('三元电池','铁锂电池') NOT NULL COMMENT '电池类型',
                                `total_mileage` decimal(10,2) NOT NULL DEFAULT '0.00' COMMENT '总里程(km)',
                                `battery_health` decimal(5,2) NOT NULL DEFAULT '100.00' COMMENT '电池健康状态(%)',
                                `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
                                `update_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '更新时间',
                                PRIMARY KEY (`id`),
                                UNIQUE KEY `uk_vid` (`vid`),
                                UNIQUE KEY `uk_vin` (`vin`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='车辆信息表';
```

车辆信息表

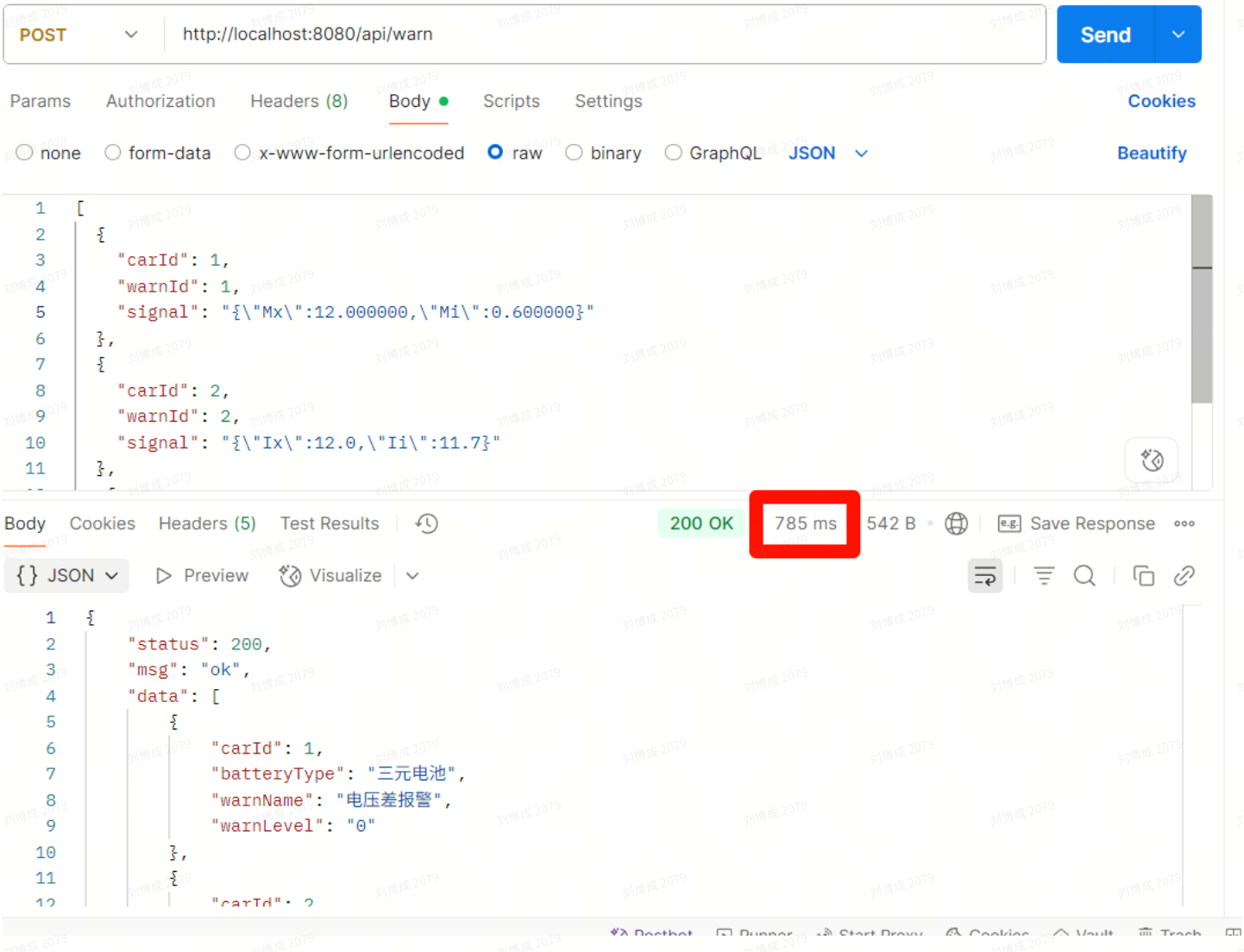
```sql
CREATE TABLE `bms_rules` (
                             `id` bigint NOT NULL AUTO_INCREMENT COMMENT '序号',
                             `rule_id` int NOT NULL COMMENT '规则编号',
                             `rule_name` varchar(50) NOT NULL COMMENT '规则名称',
                             `battery_type` enum('三元电池','铁锂电池') NOT NULL COMMENT '电池类型',
                             `rule_description` json NOT NULL COMMENT '规则描述(JSON格式)',
                             `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
```

```
                              `update_time` datetime NOT NULL DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '修改时间',
                              PRIMARY KEY (`id`),
                              UNIQUE KEY `uk_rule_battery` (`rule_id`,
`battery_type`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='BMS预警规则表';

-- 三元电池电压差报警规则
INSERT INTO `bms_rules`
(`rule_id`, `rule_name`, `battery_type`, `rule_description`)
VALUES
    (1, '电压差报警', '三元电池', '[
      {"max": null, "min": 5, "alert_level": 0},
      {"max": 5, "min": 3, "alert_level": 1},
      {"max": 3, "min": 1, "alert_level": 2},
      {"max": 1, "min": 0.6, "alert_level": 3},
      {"max": 0.6, "min": 0.2, "alert_level": 4},
      {"max": 0.2, "min": null, "alert_level": -1}
    ]');
```

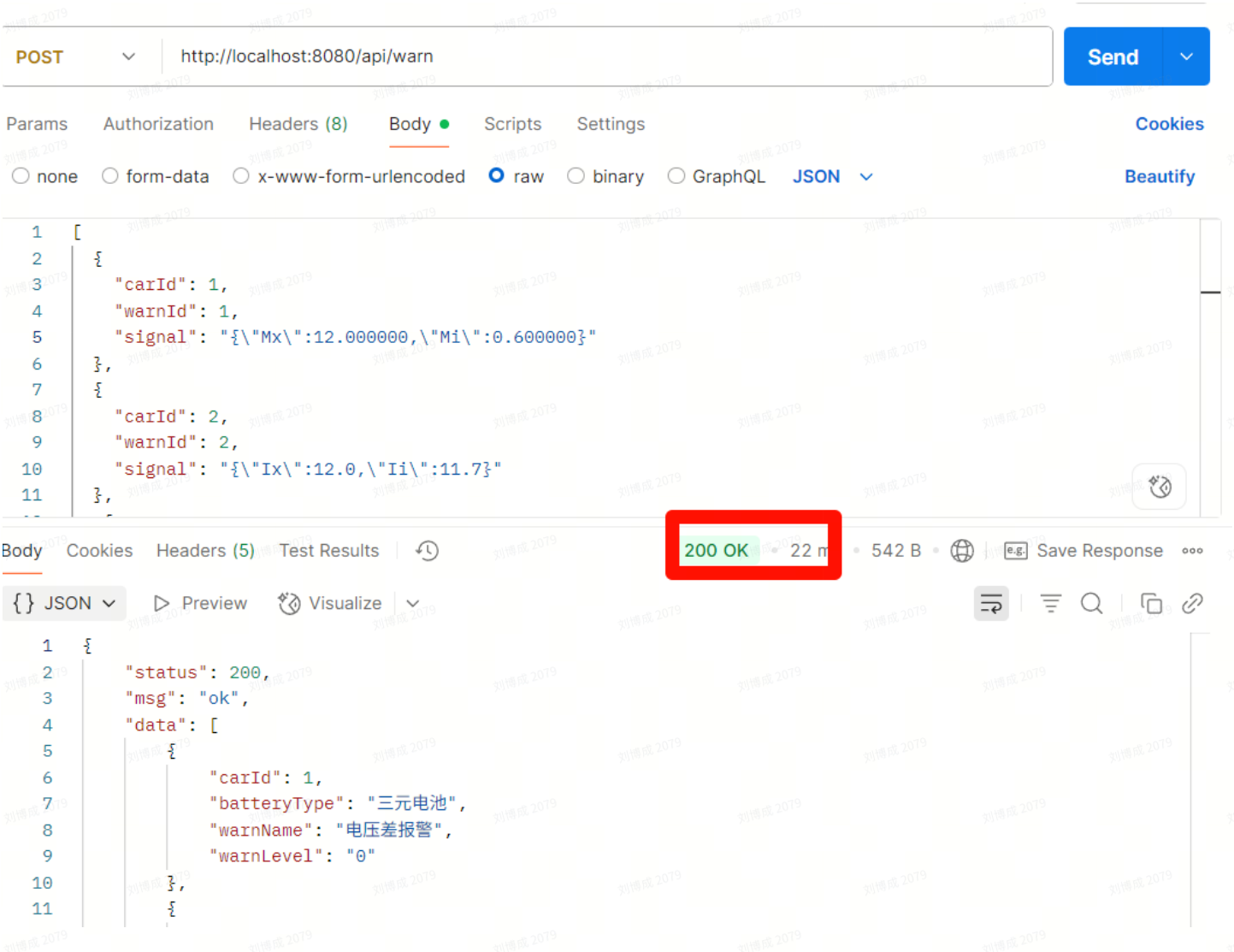
规则表

三：接口设计及测试

1.http://localhost:8080/api/warn



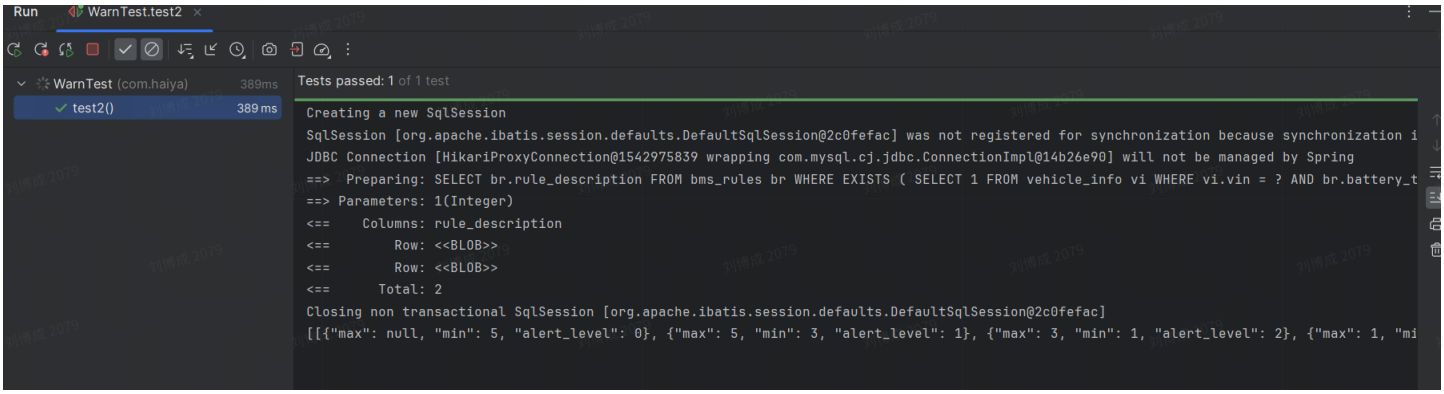
p1. 引入redis缓存之前接口响应时间



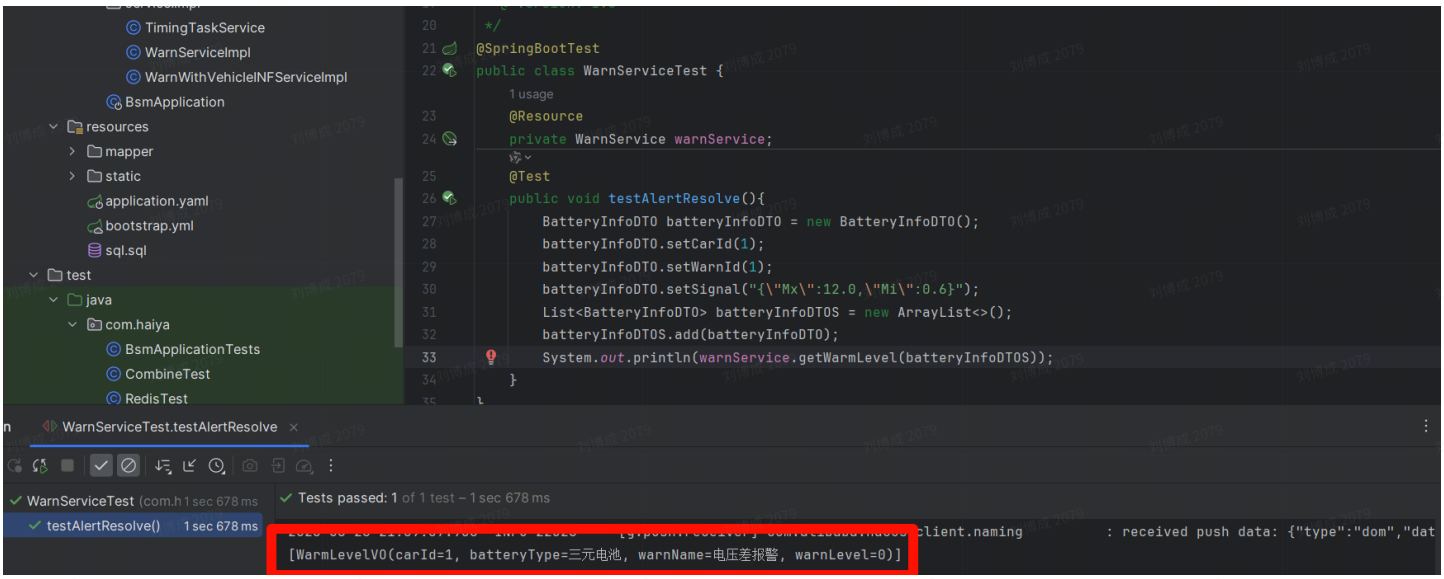
p1. 引入redis缓存之后接口响应时间

2.bsm_rules类mapper接口测试

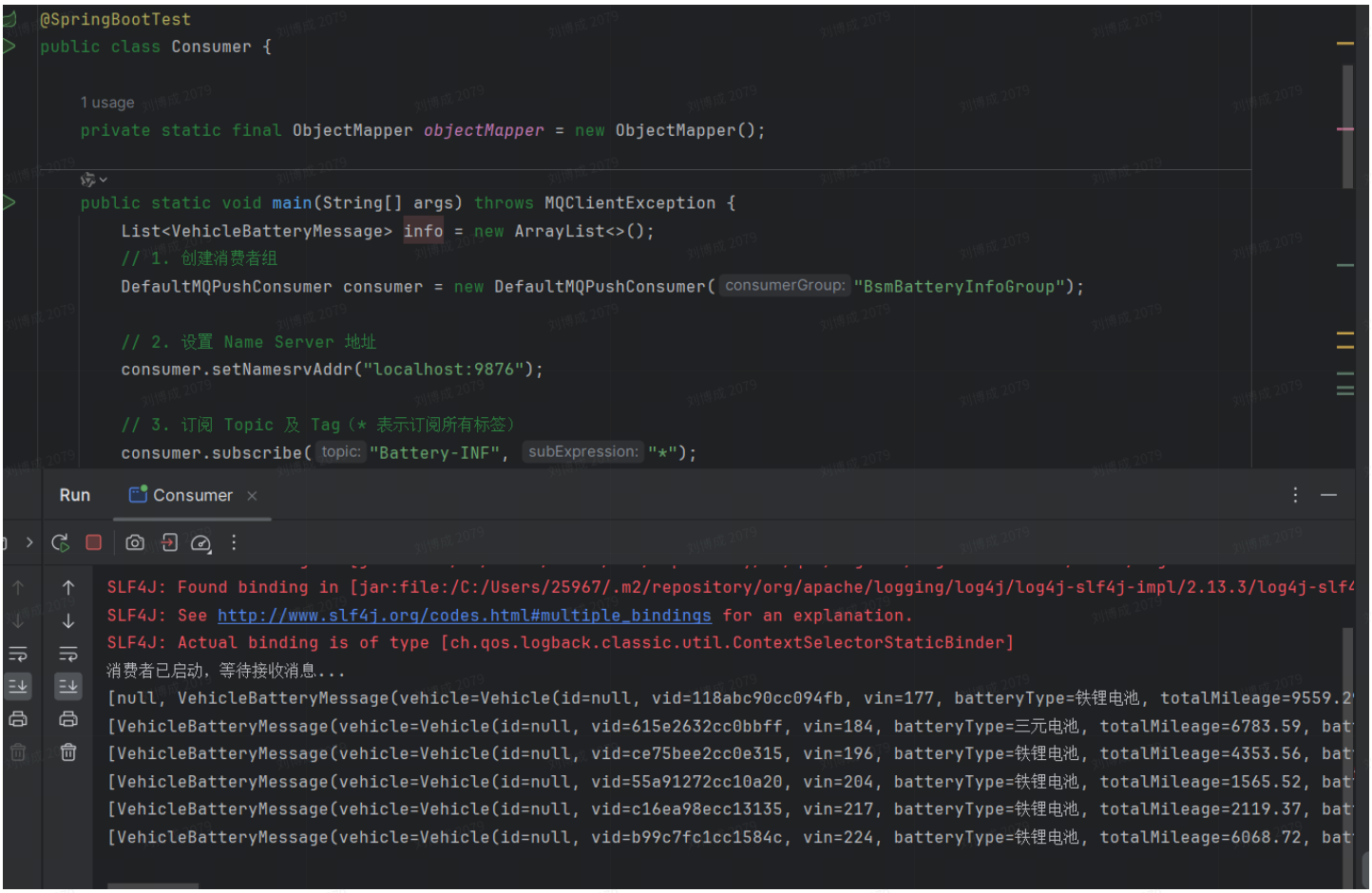
```
@SpringBootTest
public class WarnTest {
    2 usages
    @Resource
    private WarnMapper warnMapper;

    @Test
    public void test1(){
                                                    单表查询
        Map<String, Map<String, String>> rules = warnMapper.getRulesMap( ruleId: null, batteryType: "铁锂电池");
        System.out.println(rules);
    }

    @Test
    public void test2(){                            联表查询
        List<String> rulesByCarIdAndRuleId = warnMapper.getRulesByCarIdAndRuleId( ruleId: null,  carId: 1);
        System.out.println(rulesByCarIdAndRuleId);

    }


}
```



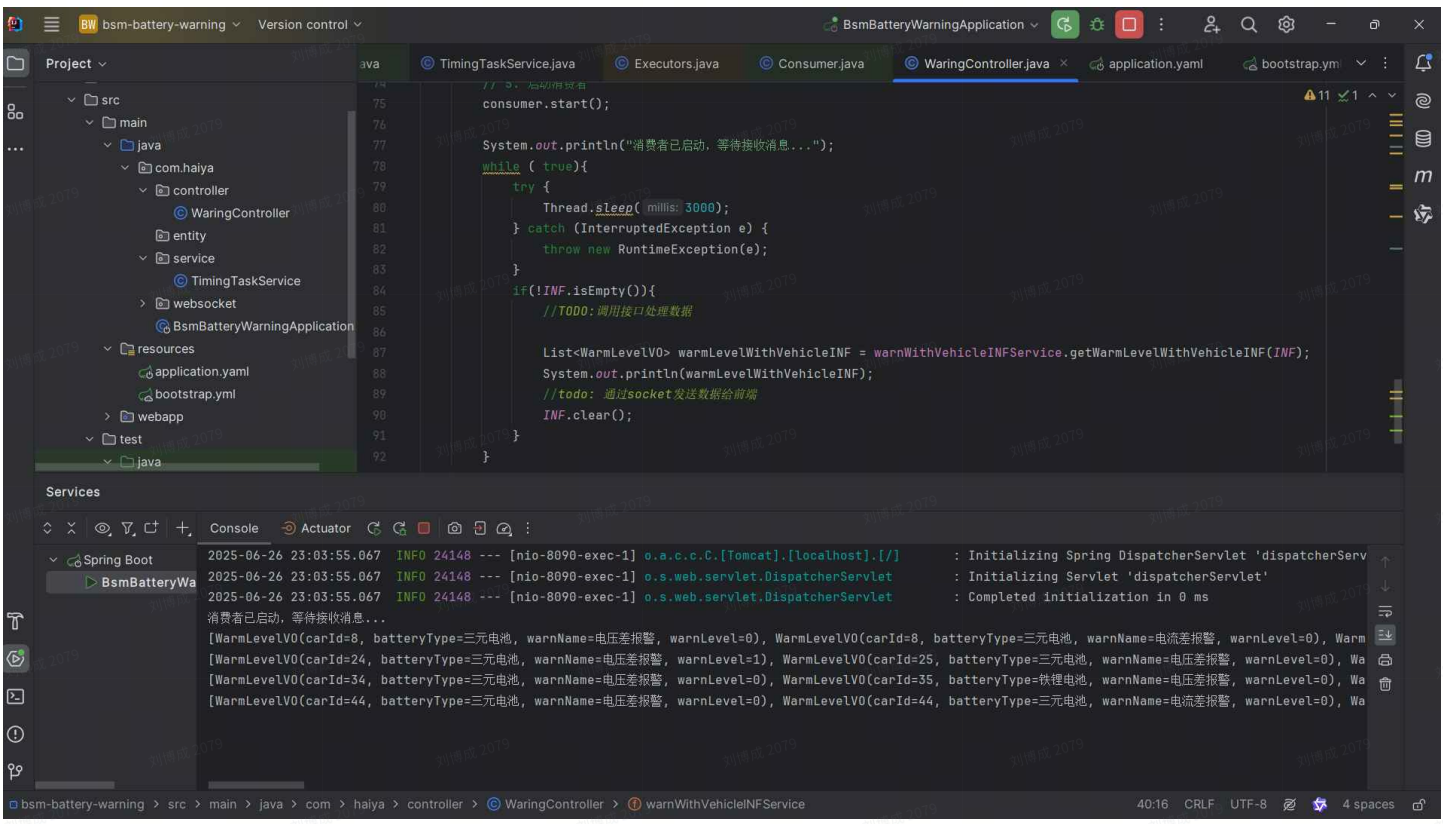
3.模拟查询指定车辆电池报警等级



4.定时任务发送及获取消息测试



5.调用bsm接口获取预警等级接口测试



四.实现细节

根据规则编号获取预警规则然后解析

1.定义bsm_rules表，将规则以json字符串的形式存储

```sql
CREATE TABLE `bms_rules` (
                            `id` bigint NOT NULL AUTO_INCREMENT COMMENT '序号',
                            `rule_id` int NOT NULL COMMENT '规则编号',
                            `rule_name` varchar(50) NOT NULL COMMENT '规则名称',
                            `battery_type` enum('三元电池','铁锂电池') NOT NULL COMMENT '电池类型',
                            `rule_description` json NOT NULL COMMENT '规则描述(JSON格式)',
                            `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
                            `update_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMME
                            PRIMARY KEY (`id`),
                            UNIQUE KEY `uk_rule_battery` (`rule_id`, `battery_type`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='BMS预警规则表';

-- 三元电池电压差报警规则
INSERT INTO `bms_rules`
(`rule_id`, `rule_name`, `battery_type`, `rule_description`)
VALUES
    (1, '电压差报警', '三元电池', '[
      {"max": null, "min": 5, "alert_level": 0},
      {"max": 5, "min": 3, "alert_level": 1},
      {"max": 3, "min": 1, "alert_level": 2},
      {"max": 1, "min": 0.6, "alert_level": 3},
      {"max": 0.6, "min": 0.2, "alert_level": 4},
      {"max": 0.2, "min": null, "alert_level": -1}
    ]');
```

rule_description中不需要存储Mx,Mi,Ix,Ii这些字段，只需要存储上界和下界。

因为warn_id就能代表是那种规则

1	电压差报警
1	电压差报警
2	电流差报警
2	电流差报警

2.解析规则

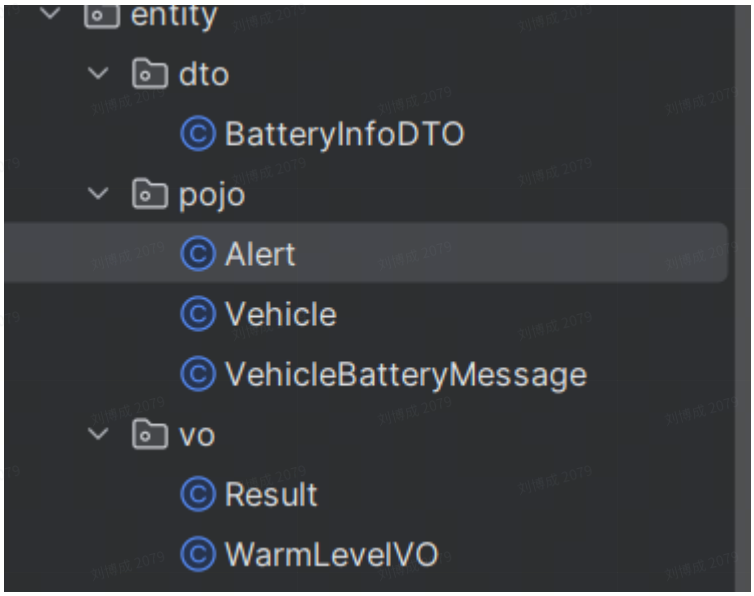
```
// 根据规则获取报警级别
Map<String, String> warmLevel = getLevel(batteryInfoDTO.getSignal(), rules);
```

```java
private static Map<String, String> getLevel(String signal, Map<String, Map<String, String>> rules) {
    Map<String, String> warmLevel = new HashMap<>();
    Map<String, Double> map = null;

    try {
        map = objectMapper.readValue(signal, Map.class);
    } catch (JsonProcessingException e) {
        throw new RuntimeException(e);
    }

    if (map.containsKey("Mx")) {
        processRule(map, rules, warmLevel, ruleName: "电压差报警", maxKey: "Mx", minKey: "Mi");
    }

    if (map.containsKey("Ix")) {
        processRule(map, rules, warmLevel, ruleName: "电流差报警", maxKey: "Ix", minKey: "Ii");
    }

    return warmLevel;
}
```

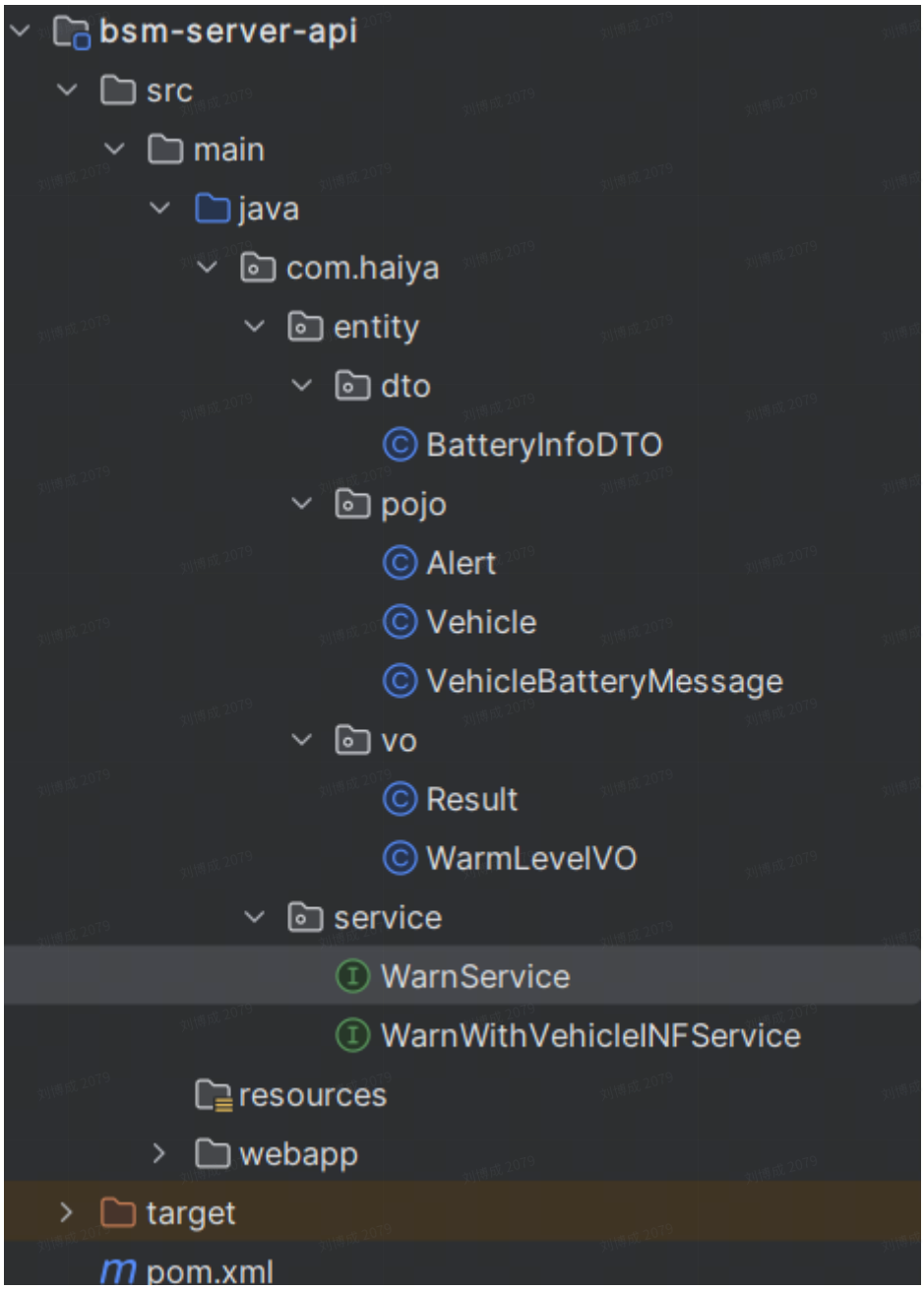
```java
private static void processRule(Map<String, Double> map, Map<String, Map<String, String>> rules,
                                Map<String, String> warmLevel, String ruleName, String maxKey, String minKey) {
    String ruleDescription = rules.get(ruleName).get("rule_description");
    List<Alert> alerts = null;

    try {
        alerts = objectMapper.readValue(ruleDescription, new TypeReference<List<Alert>>() {});
    } catch (JsonProcessingException e) {
        throw new RuntimeException(e);
    }

    double diff = map.get(maxKey) - map.get(minKey);
    if (diff > alerts.get(0).getMin()) {
        warmLevel.put(ruleName, String.valueOf(alerts.get(0).getAlert_level()));
    } else {
        for (int i = 1; i < alerts.size() - 1; i++) {
            if (diff > alerts.get(i).getMin() && diff < alerts.get(i).getMax()) {
                warmLevel.put(ruleName, String.valueOf(alerts.get(i).getAlert_level()));
                return;
            }
        }
        if (diff < alerts.get(alerts.size() - 1).getMax()) {
            warmLevel.put(ruleName, "不报警");
        }
    }
}
```

```java
@Getter
@Setter
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Alert {
        private Double max;
        private Double min;
        private Integer alert_level;

}
```

3.定义各种实体做数据转换



4.bsm-server-api模块

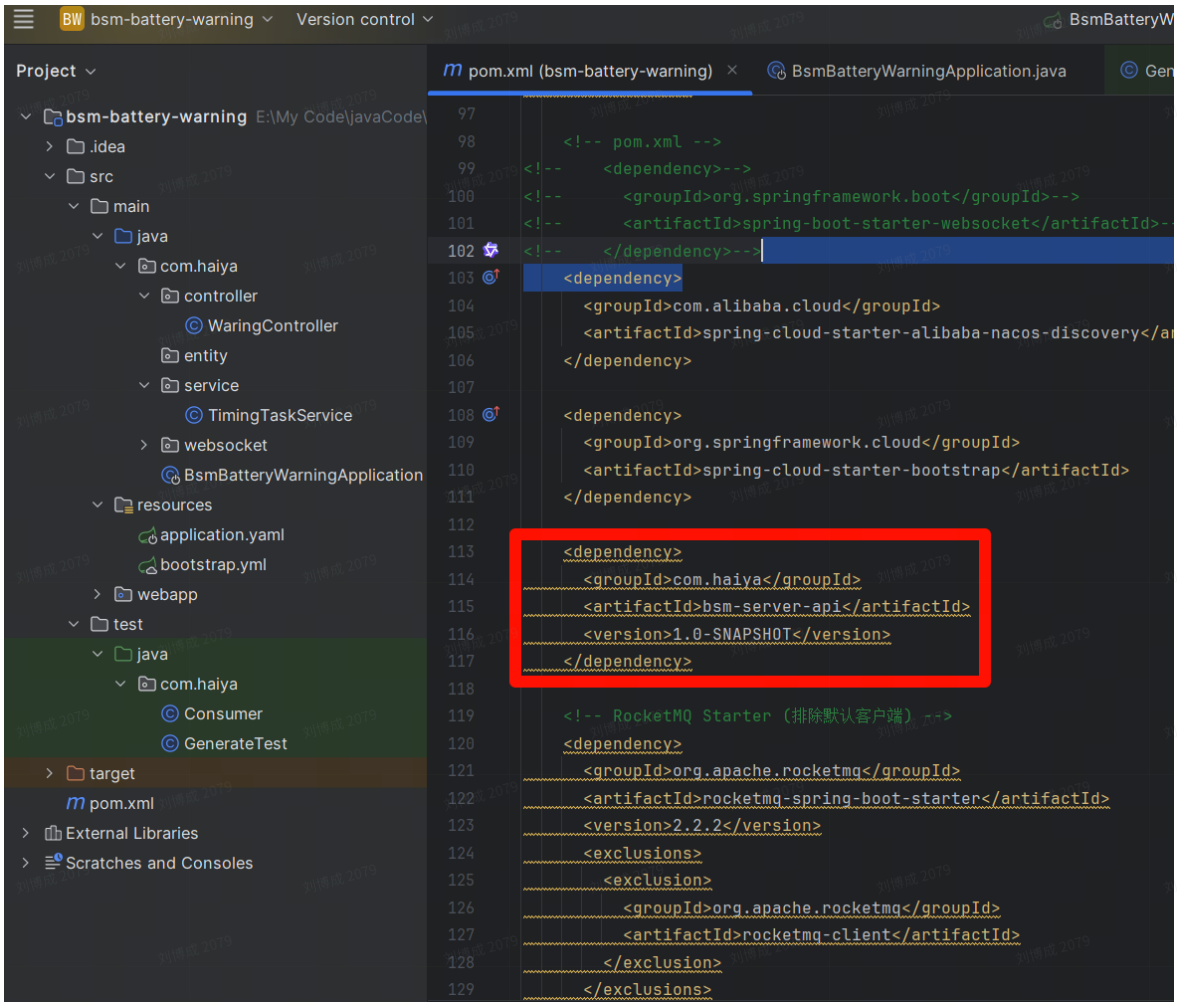


5.预警功能框架

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

另起一个dubbo服务，引入bsm-server-api模块，注册中心为nacos。



6.定时任务模拟车辆信息——电池信号数据生成

```
/**
 * 生成唯一16位随机字符串
 */
1 usage
private static String generateUnique16BitString() {
    // 使用UUID前8位 + 时间戳后8位组合成16位字符串
    String uuidPart = UUID.randomUUID().toString().replace( target: "-",  replacement: "").substring(0, 8);
    String timePart = Long.toHexString(System.currentTimeMillis()).substring( beginIndex: 4);
    return uuidPart + timePart;
//      //使用雪花算法
//      long id = nextId();
//      return encodeBase62(id);
}
```

uuid+时间戳生成vid

```java
public static String generateVehicleAndBatteryInfo() {
    // 生成VIN（int）和VID（String）
    int vin = vinCounter.getAndIncrement();
    String vid = generateUnique16BitString();

    // 构造 Vehicle 和 BatteryInfoDTO 对象
    Vehicle vehicle = new Vehicle();
    vehicle.setVid(vid);
    vehicle.setVin(String.valueOf(vin));
    vehicle.setBatteryType(Math.random() < 0.5 ? "三元电池" : "铁锂电池");
    vehicle.setTotalMileage(Math.round(Math.random() * 10000 * 100) / 100.0); // 随机里程 要求保留两位小数
    vehicle.setBatteryHealth(Math.round((70 + Math.random() * 30) * 100) / 100.0);  // 随机健康度 要求保留两位小数

    BatteryInfoDTO batteryInfoDTO = new BatteryInfoDTO();
    batteryInfoDTO.setCarId( vin); //
    // 随机状态 1 或 2 或 null共三种状态
    Integer warnId = Math.random() < 0.5 ? Integer.valueOf( i: 1) : Math.random() < 0.5 ? 2 : null;
    batteryInfoDTO.setWarnId(warnId);
    batteryInfoDTO.setSignal(generateSignal(warnId)); // 随机信号
    // 构建组合对象
    return toJson(vehicle, batteryInfoDTO);
}
```

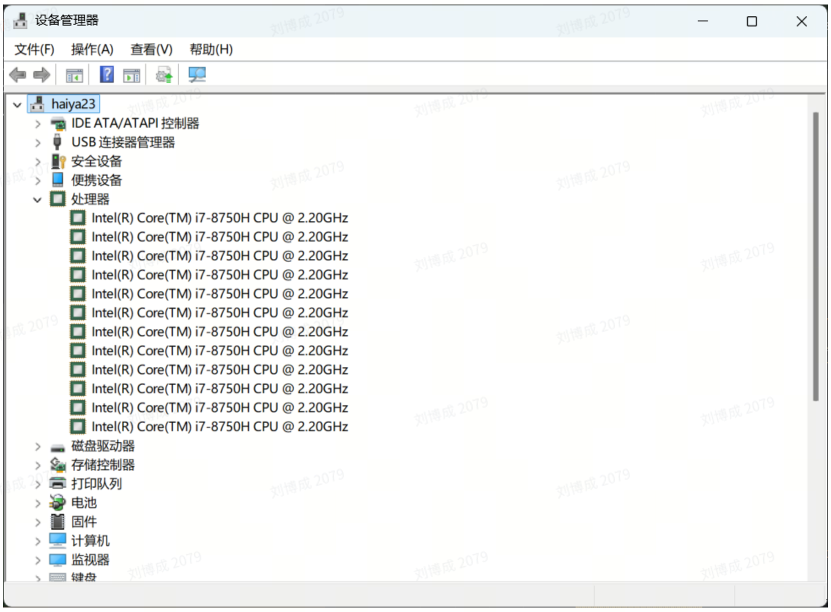
模拟数据生成

```java
    private static String generateSignal(Integer warnId) {
        // 随机生成浮点数值
        double mx = Math.round((5.0 + Math.random() * 10) * 100) / 100.0; // 5.0 ~ 15.0
        double mi = Math.round((0.0 + Math.random() * 5)  * 100) / 100.0; // 0.0 ~ 5.0
        double ix = Math.round((10.0 + Math.random() * 5) * 100) / 100.0; // 10.0 ~ 15.0
        double ii = Math.round((0.0 + Math.random() * 12)  * 100) / 100.0; // 0.0 ~ 12.0

        // 定义三种信号模板（使用 %f 作为浮点数占位符）
        String template1 = "{\"Mx\":%f,\"Mi\":%f}";
        String template2 = "{\"Ix\":%f,\"Ii\":%f}";
        String template3 = "{\"Mx\":%f,\"Mi\":%f,\"Ix\":%f,\"Ii\":%f}";

        if (warnId == null) {
            return String.format(template3, mx, mi, ix, ii);
        }

        switch (warnId) {
            case 1:
                return String.format(template1, mx, mi);  // 状态1 电压
            case 2:
                return String.format(template2, ix, ii);  // 状态2 电流
            default:
                return "";
        }
    }
}
```

singal字段生成

7.定时推送到mq

```java
@PostConstruct
public void init() throws MQClientException {
    producer = new DefaultMQProducer("BsmBatteryInfoGroup");
    producer.setNamesrvAddr("localhost:9876");
    producer.start();
}

@Scheduled(fixedRate = 1000 * 10)
public void sendBatteryInfoMessage() throws InterruptedException, JsonProcessingException {
    List<Future<String>> futures = new ArrayList<>();
    for (int i = 0; i < MESSAGES_PER_CALL; i++) {
        futures.add(executorService.submit(TimingTaskService::generateVehicleAndBatteryInfo));
    }

    for (Future<String> future : futures) {
        try {
            String messageBody = future.get(); // 获取线程结果
            Message msg = new Message(TOPIC, messageBody.getBytes(StandardCharsets.UTF_8));
            producer.send(msg);
        } catch (ExecutionException | MQBrokerException | InterruptedException | MQClientException |
                 RemotingException e) {
            e.printStackTrace();
        }
    }
}

@PreDestroy
public void destroy() {
    if (producer != null) {
        producer.shutdown();
    }
```



```java
6 usages
private DefaultMQProducer producer;
1 usage
static final String TOPIC = "Battery-INF";
2 usages
private static final int THREAD_COUNT = 13; // 线程数量
2 usages
private static final int MESSAGES_PER_CALL = 10; // 每次生成的消息条数
1 usage
private static final AtomicInteger vinCounter = new AtomicInteger( initialValue: 4); // VIN起始值
//    private static final ExecutorService executorService = Executors.newFixedThreadPool(THREAD_COUNT);
2 usages
private static final ObjectMapper objectMapper = new ObjectMapper();
2 usages
private static final ExecutorService executorService = new ThreadPoolExecutor(
        THREAD_COUNT,
        THREAD_COUNT,
        keepAliveTime: 0L,
        TimeUnit.MILLISECONDS,
        new LinkedBlockingQueue<>(MESSAGES_PER_CALL),
        new ThreadPoolExecutor.DiscardPolicy()
);
```

线程池设置

8.从mq中拉去数据调用api接口获取预警等级

GenerateTest.java | TimingTaskService.java | Executors.java | Consumer.java | WaringController.java | application.yaml

```java
// 4. 注册消息监听器
consumer.registerMessageListener((MessageListenerConcurrently) (msgs, context) -> {
    VehicleBatteryMessage received = null;
    for (MessageExt msg : msgs) {
        // 直接返回原始消息内容
        try {
            received = objectMapper.readValue(msg.getBody(), VehicleBatteryMessage.class);
            INF.add(received);
            // 新增：将收到的数据转换为 JSON 并发送给前端
            BatteryWebSocket.addDataAndNotifyFrontend(objectMapper.writeValueAsString(received));
        } catch (Exception e) {
            throw new RuntimeException("Error processing message", e);
        }
    }
    return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});

// 5. 启动消费者
consumer.start();

System.out.println("消费者已启动，等待接收消息...");
while ( true){
    try {
        Thread.sleep( millis: 3000);
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    }
    if(!INF.isEmpty()){
        //TODO:调用接口处理数据

        List<WarmLevelVO> warmLevelWithVehicleINF = warnWithVehicleINFService.getWarmLevelWithVehicleINF(INF);
        System.out.println(warmLevelWithVehicleINF);
        //todo: 通过socket发送数据给前端
        INF.clear();
```

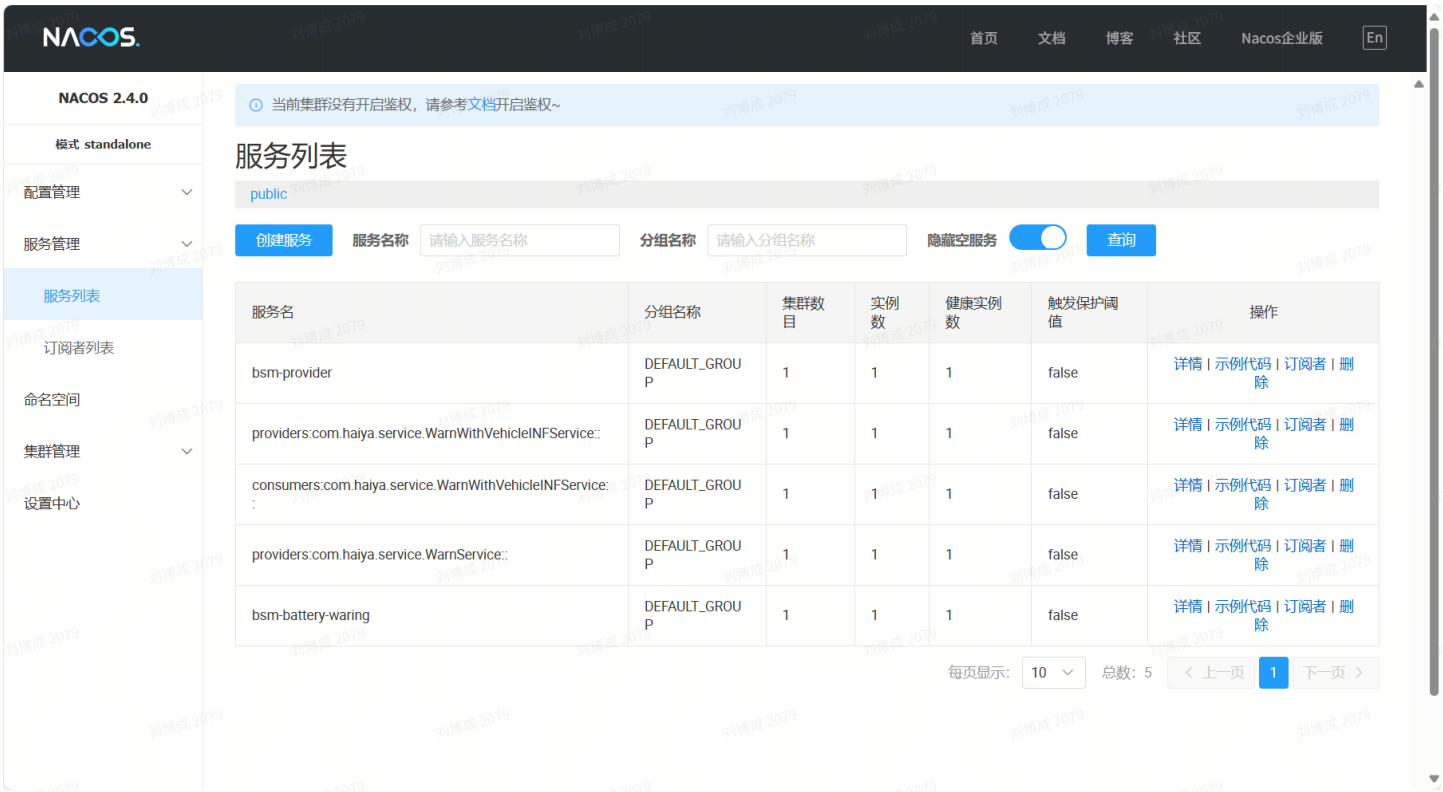
拉去mq的消息封装到VehicleBatteryMessage类中，车辆信息与电池信息意义对应。调用接口获取报警等级实时打印在控制台

```
14 usages
@Data
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class VehicleBatteryMessage implements Serializable {
    private Vehicle vehicle;
    private BatteryInfoDTO battery;

}
```

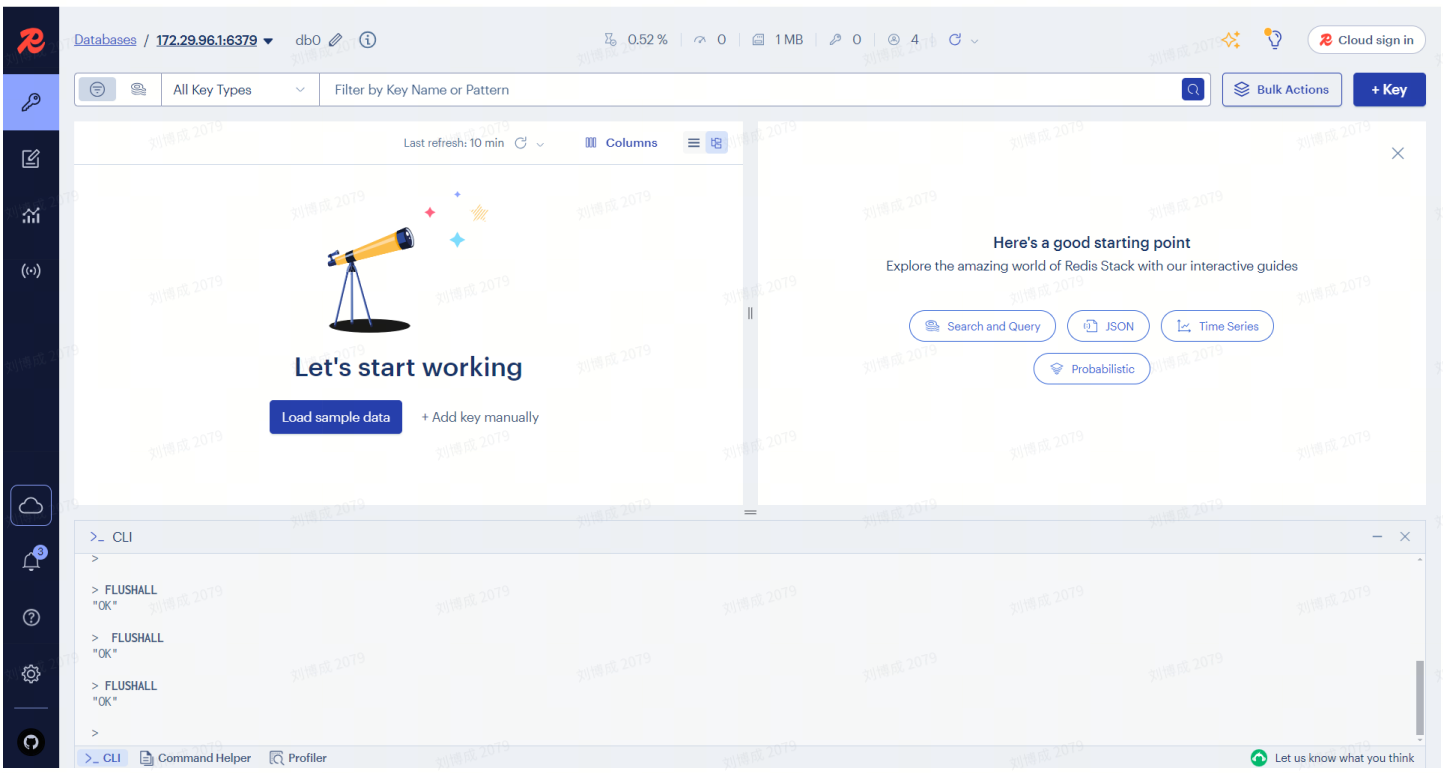
```
public class WarnWithVehicleINFServiceImpl implements WarnWithVehicleINFService {
    1 usage
    @Resource
    private WarnService warnService;
    1 usage
    @Resource
    private VehicleMapper vehicleMapper;

    no usages
    @Override
    public List<WarmLevelVO> getWarmLevelWithVehicleINF(List<VehicleBatteryMessage> vehicleBatteryMessages) {
        List<Vehicle> VEHICLES = new ArrayList<>();
        List<BatteryInfoDTO> BATTERIES = new ArrayList<>();
        for (VehicleBatteryMessage vehicleBatteryMessage : vehicleBatteryMessages) {
            if (vehicleBatteryMessage != null){
                VEHICLES.add(vehicleBatteryMessage.getVehicle());
                BATTERIES.add(vehicleBatteryMessage.getBattery());
            }
        }
        //1.车辆添加到数据库中
        for (Vehicle vehicle : VEHICLES) {
            //todo:添加布隆过滤器，判断是否在数据库中，在执行插入
            // 设置创建时间和更新时间
            vehicle.setCreateTime(new Date(System.currentTimeMillis()));
            vehicle.setUpdateTime(new Date(System.currentTimeMillis()));
            //插入数据库
            vehicleMapper.insert(vehicle);
        }
//          vehicleMapper.insetBatch(vehicles);
        //4.获取报警等级并返回返回数据
        return warnService.getWarmLevel(BATTERIES);
    }
}
```

getWarmLevelWithVehicleINF接口

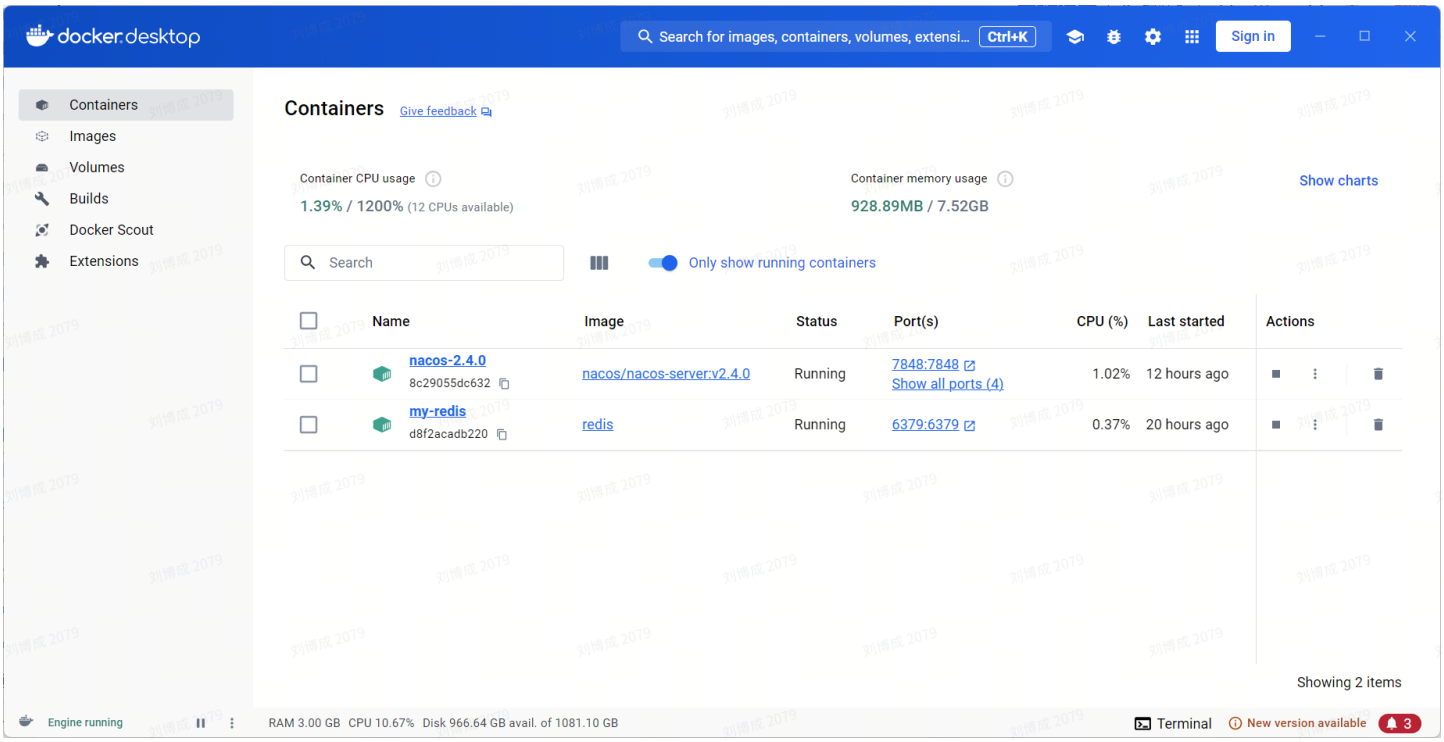
五.中间件服务



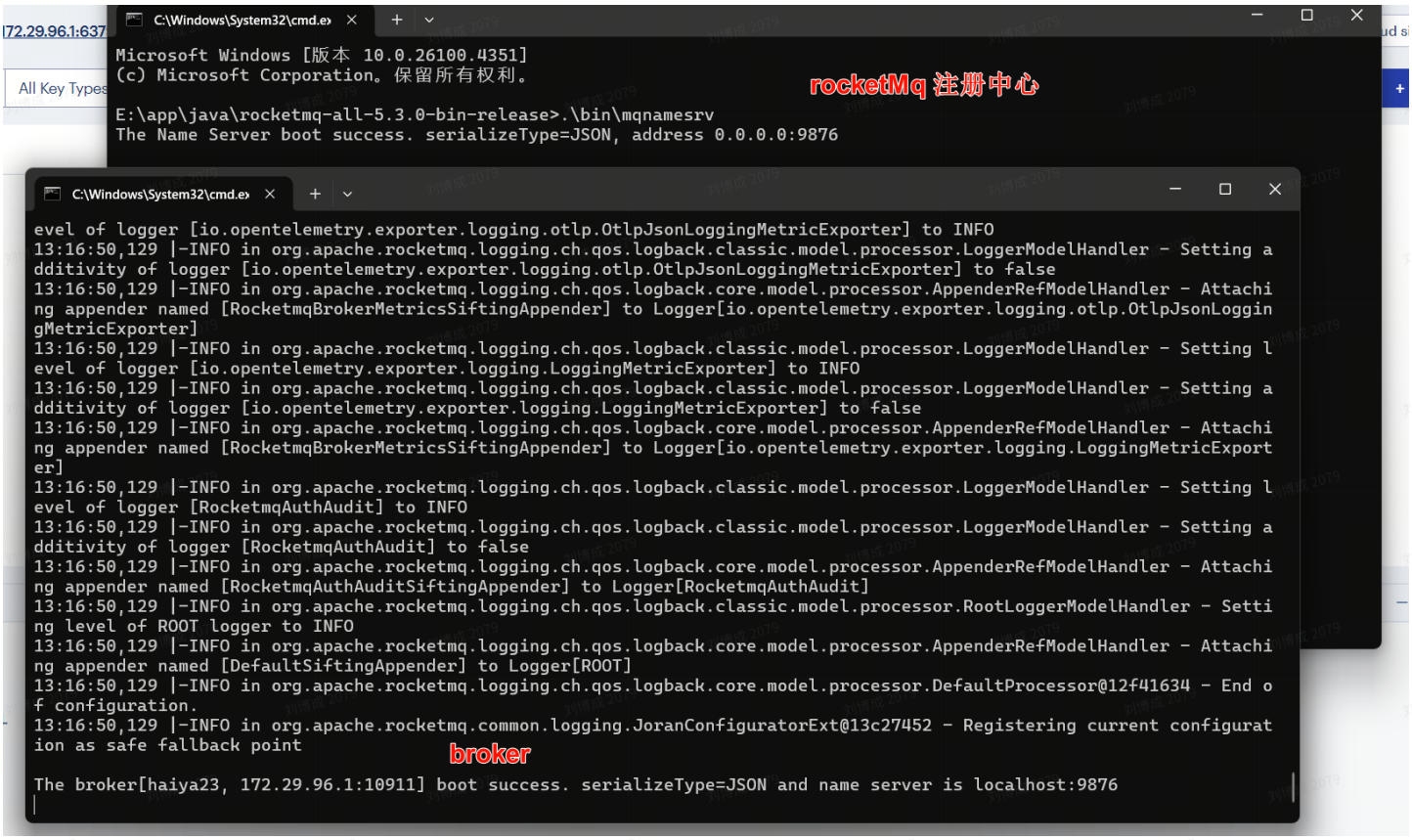
nacos注册中心



Redis insight可视化工具



redis、nacos服务



rocketMq