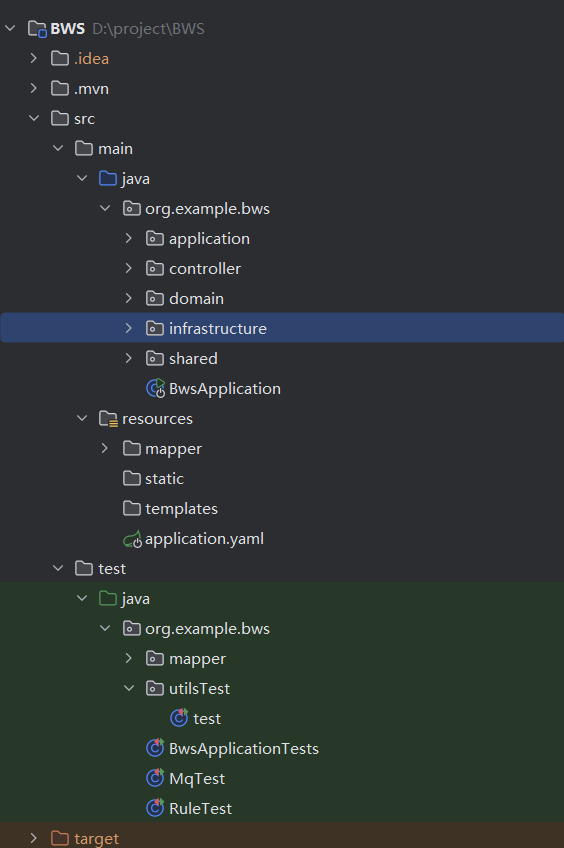
**大作业技术方案设计**

**1.系统设计**

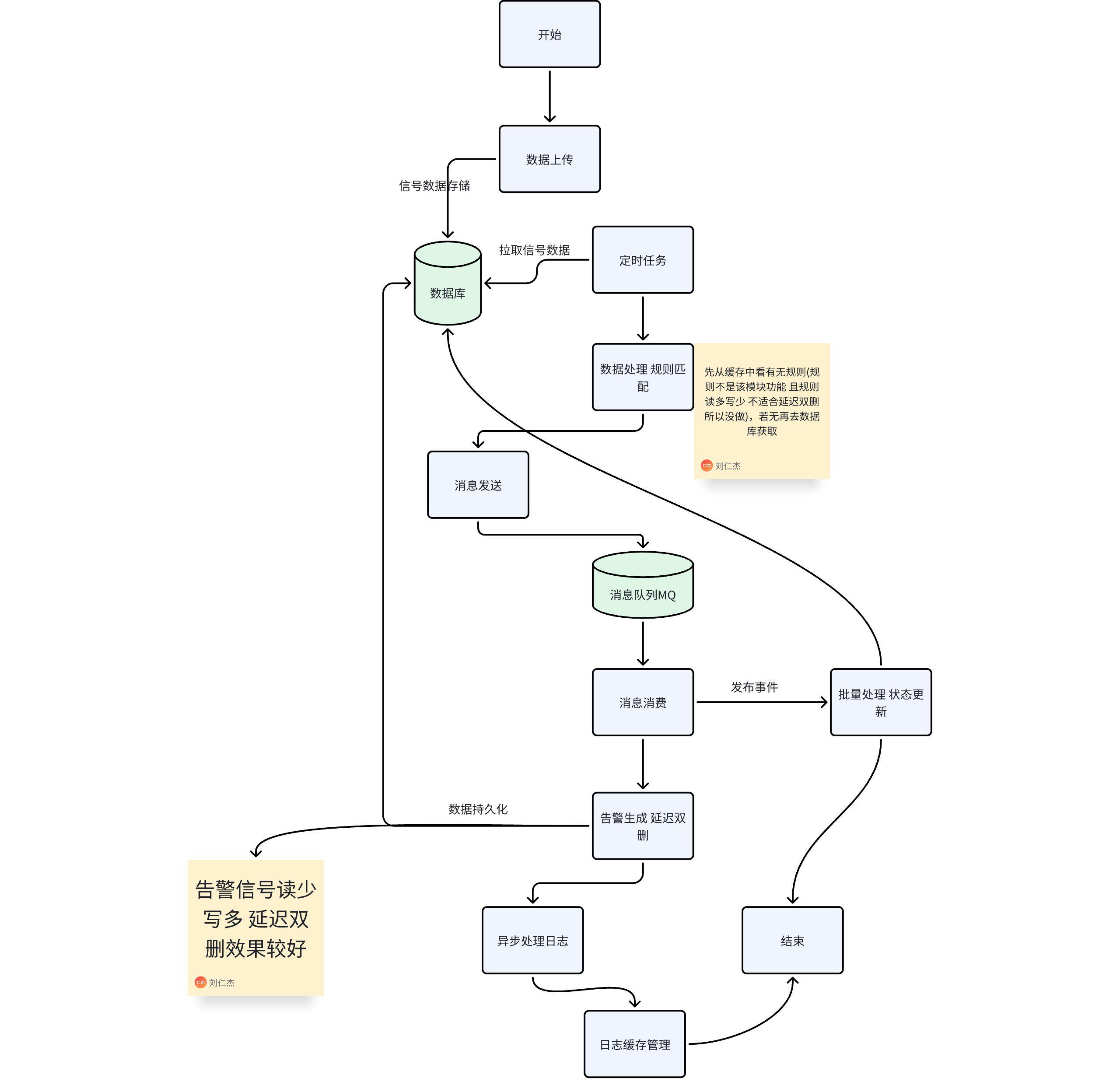
使用DDD领域驱动设计

* 表现层（Controller）：负责接收客户端请求，进行参数校验和请求分发，返回处理结果给客户端。
* 应用层（Service）：协调领域层的服务完成具体业务逻辑，处理事务和业务流程编排。
* 领域层（Domain）：包含业务领域的核心模型和业务逻辑，如规则处理、信号处理等。
* 基础设施层（Repository）：负责与数据库、缓存、消息队列等外部资源进行交互，提供数据持久化和数据访问服务。

**目录如下**

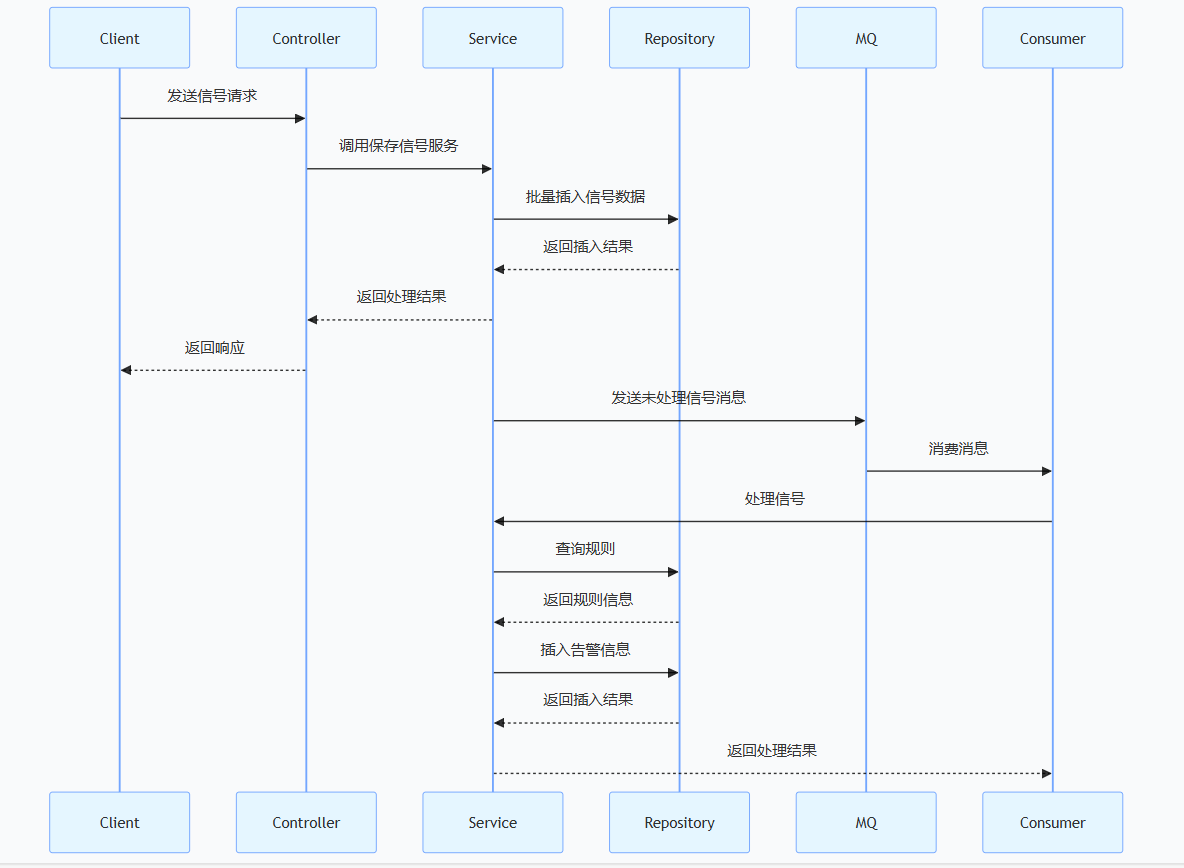


**主要业务流程图**



**模块划分**

* 信号处理模块：负责接收和保存电池信号，处理未处理的信号。
* 定时任务模块：负责扫描未处理的电池信号 ，并将其发送给mq
* 规则处理模块：根据规则对电池信号进行评估，判断是否触发告警，并返回告警信息。
* 消息队列模块：使用 RocketMQ 作为消息队列，处理并消费未处理信息 并保存
* 缓存模块：使用 Redis 作为缓存，提高规则查询的性能。
* 异常处理模块：全局异常处理，捕获并处理系统中出现的异常，返回统一的错误信息给客户端。



**思路：**

|  |
| --- |
| 通过消息队列使信号的处理 削峰 流量整形使得系统的流量平稳 |

|  |
| --- |
| 通过扫描数据库 可以一次获得更多数据减少了由于网络IO 和频繁的磁盘IO 导致处理变慢 |

|  |
| --- |
| 通过专门的状态更新模块 异步 收集被处理好的信号id 批量更新 理由同上 |

|  |
| --- |
| 本地写好处理日志 方便比对数据库 更正错误 也可以通过并发读取处理好的日志 发到es 更好的分类和聚合日志 也方便分析数据 |

|  |
| --- |
| 消费者消费数据时需要先获取全局唯一Id 防止重复消费 其他无法set的同时快速失败也可以减少性能损耗  由于是定时任务处理 且是从数据库拿 哪怕消费失败 也可以重新消费 |

Rule没有多版本 不能根据不同版本生效

**2.数据库设计**

**数据库表设计**

**车辆基础信息表(car\_info)**



**点击图片可查看完整电子表格**

vid和car\_id做索引

**信号报告表（signal\_report）**



**点击图片可查看完整电子表格**

is\_handled做索引 因为要依靠这个查找

**规则表（rule）**



**点击图片可查看完整电子表格**

用(rule\_id,battery\_type)做索引 因为需要这样查 查询时要满足最左前缀

**告警信息表（warn\_info）**



**点击图片可查看完整电子表格**

索引有两个为report\_id 和(car\_id,warn\_time)

**3.接口设计**

**上传接口**

* 上报接口

接口名：/api/warn

接口方法：POST

Body：格式为数组，数组内的每个元素包含以下字段。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段 | 类型 | 是否必传 | 含义 | 示例 | 备注 |
| carId | int | 必须 | 车架编号 | 1 |  |
| warnId | int | 非必须 | 规则编号 | 1 | 不传时候，遍历所有规则 |
| signal | String | 必须 | 信号 | {"Mx":1.0,"Mi":0.6} |  |

Body 示例：

|  |
| --- |
| JSON [  {  "carId": 1,  "warnId": 1,  "signal": "{\"Mx\":12.0,\"Mi\":0.6}"  },  {  "carId": 2,  "warnId": 2,  "signal": "{\"Ix\":12.0,\"Ii\":11.7}"  }，  {  "carId": 3,  "signal": "{\"Mx\":11.0,\"Mi\":9.6,\"Ix\":12.0,\"Ii\":11.7}"  } ] |

响应示例：

|  |
| --- |
| JSON {  "status": 200,  "msg": "ok" } |

**查询接口**

* 上报接口

接口名：/api/warn/query

接口方法：POST

Body：格式为数组，数组内的每个元素包含以下字段。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字段 | 类型 | 是否必传 | 含义 | 示例 | 备注 |
| carId | int | 必须 | 车架编号 | 1 |  |

Body示例：

|  |
| --- |
| JSON {  "carId":1 } |

响应示例：

|  |
| --- |
| JSON {  "status": 200,  "msg": "ok",  "data": [  {  "batteryType": "三元电池",  "warnName": "电压差报警",  "warnLevel": 0  },  {  "batteryType": "三元电池",  "warnName": "电压差报警",  "warnLevel": 0  },  {  "batteryType": "三元电池",  "warnName": "电压差报警",  "warnLevel": 0  }  ] } |

**4.单元测试设计**

**mq单元测试**

@Test  
public void test() throws JsonProcessingException, InterruptedException {  
 List<SignalReport> signalReports = signalReportRepository.selectByHandledStatus(false);  
 signalReports.forEach(System.*out*::println);  
  
 for (SignalReport signalReport : signalReports) {  
 String json = objectMapper.writeValueAsString(signalReport);  
 System.*out*.println(json);  
  
 warmMessageProducer.sendMessage("warm-message", json);  
 }  
 Thread.*sleep*(5000);  
}

**规则解析器单元测试**

示例

@Test

public void testEvaluateWarningLevel() {

Rule rule = new Rule(); // 初始化 Rule 对象

rule.setAlertRules(Arrays.asList(

new AlertRule("3 <= (Mx - Mi)", 1),

new AlertRule("0.2 <= (Ix - Ii) <1", 1)

));

RuleParser parser = new RuleParser();

AlertRule alertRule = parser.evaluateWarningLevel(rule, 0.0, 0.0, 12.0, 11.7);

assertTrue("警告级别应为1", alertRule != null && alertRule.getWarnLevel() == 1);

}

@Test  
void evaluateSimpleArithmeticExpression() {  
 ExpressionEvaluator evaluator = new ExpressionEvaluator();  
 Map<String, Double> variables = new HashMap<>();  
 variables.put("x", 5.0);  
 variables.put("y", 3.0);  
  
 boolean result = evaluator.evaluate("x + y > 7", variables);  
 *assertTrue*(result, "5+3=8 >7 应返回true");  
  
 result = evaluator.evaluate("x \* y < 16", variables);  
 *assertTrue*(result, "5\*3=15 <16 应返回true");  
}