一、背景介绍

1.1 MongoDB概念

MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。他支持的数据结构非常松散，是类似json的bson格式，因此可以存储比较复杂的数据类型。Mongo最大的特点是他支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。

1.2 MongoDB特点

它的特点是高性能、易部署、易使用，存储数据非常方便。主要功能特性有：

- 面向集合存储，易存储对象类型的数据。
- 模式自由。
- 支持动态查询。
- 支持完全索引，包含内部对象。
- 支持查询。
- 支持复制和故障恢复。
- 使用高效的二进制数据存储，包括大型对象（如视频等）。
- 自动处理碎片，以支持云计算层次的扩展性。
- 文件存储格式为BSON（一种JSON的扩展）。
- 可通过网络访问。

另外，需要注意的是，使用mongodb存在索引限制（参考关系型数据库索引）

1. 如果现有的索引字段的值超过索引键的限制，mongodb中不会创建索引。
2. 集合中索引不能查过64个
3. 索引名的长度不能超过125个字符
4. 复合索引最多可以有31个字段
5. 索引存储在内存（RAM）中，应该确保该索引的大小不超过内存限制
6. 如果文档的索引字段值超过了索引键的限制，mongodb不会将文档转换成索引的集合。

1.3 使用场景介绍

主要场景:

> 1）网站实时数据处理。它非常适合实时的插入、更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。

2）缓存。由于性能很高，它适合作为信息基础设施的缓存层。在系统重启之后，由它搭建的持久化缓存层可以避免下层的数据源过载。

3）高伸缩性的场景。非常适合由数十或数百台服务器组成的数据库，它的路线图中已经包含对MapReduce引擎的内置支持。

不适用的场景:

1）要求高度事务性的系统。

2）传统的商业智能应用。

3）复杂的跨文档（表）级联查询。

二、业务场景消息记录概述

2.1 nutown内部业务场景

- 各中心通过MQ异步的方式记录日志
- 各个中心发送MQ，消费MQ消息，均会记录日志。分布式雪花算法生成messageId 作为唯一标识。messageType 区分发送、消费方

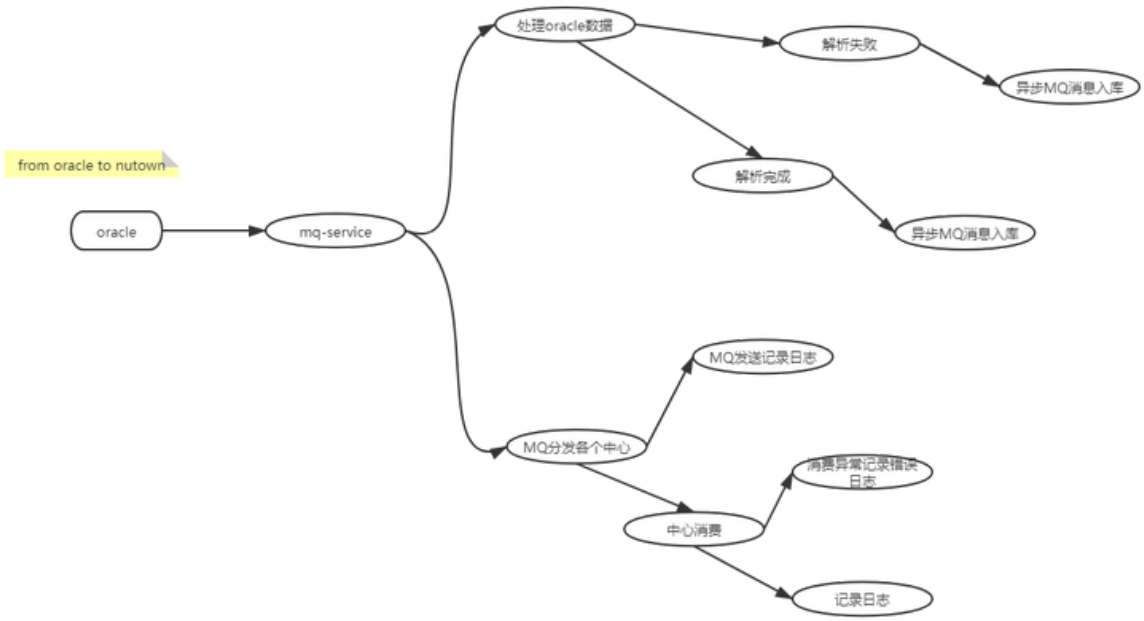
```
SEND_ERROR_LOG("send_error_log", "发送mq异常日志"),
RECEIVE_ERROR_LOG("receive_error_log", "接收mq异常日志"),
RECEIVE_LOG("receive_log", "接收mq日志"),
SEND_LOG("send_log", "发送日志"),
BEFORE_SEND_ERROR_LOG("before_send_error_log", "发送mq前异常日志"),
```

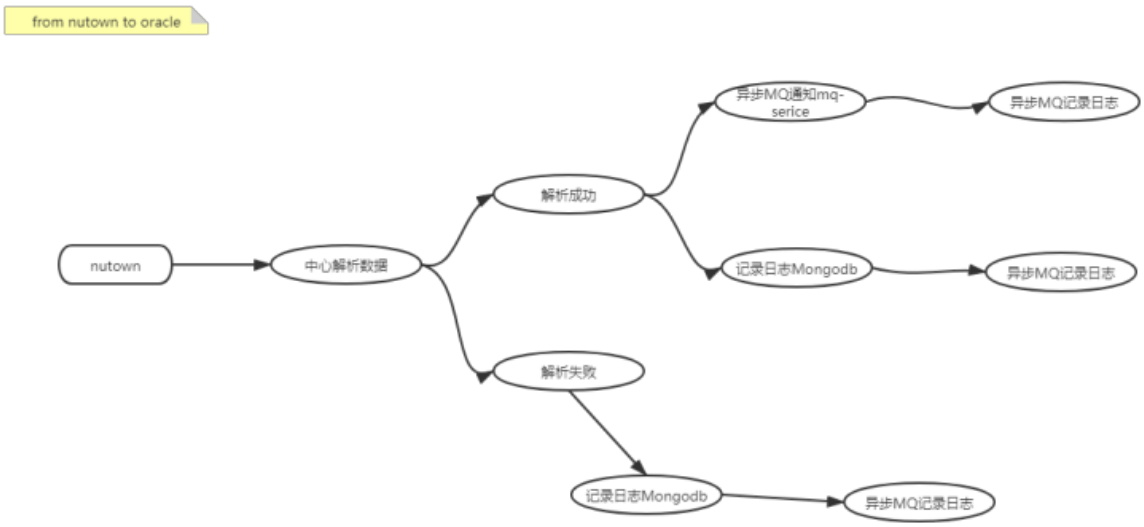
- 主要具体业务
    - 请翻阅应用层或中心对应具体场景详细设计的业务；

2.2 nutown与第三方交互监控场景

- 与第三方互相请求数据成功或失败的日志

2.3 用例图





三、 能力中心

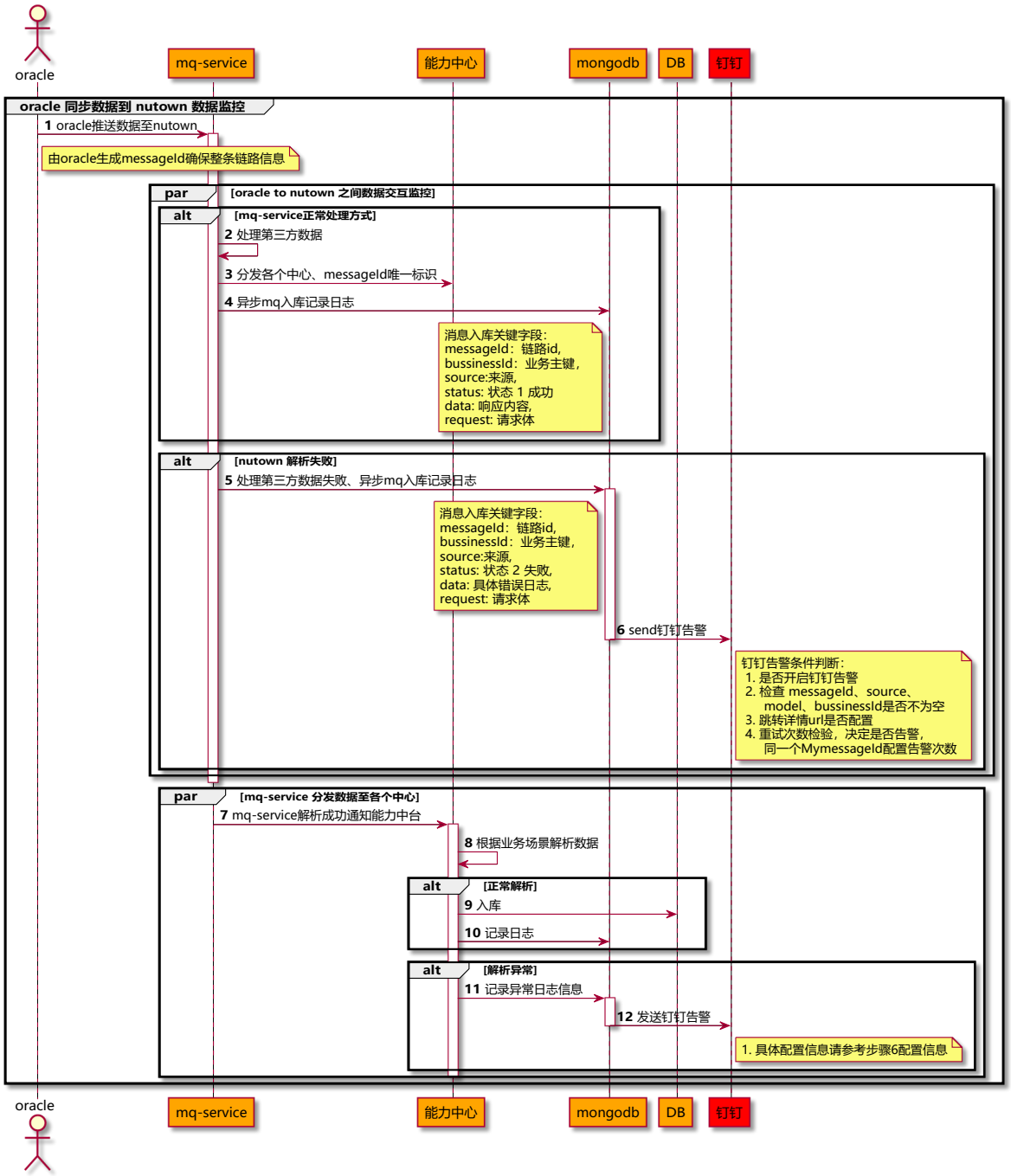
3.1 钉钉告警示意图

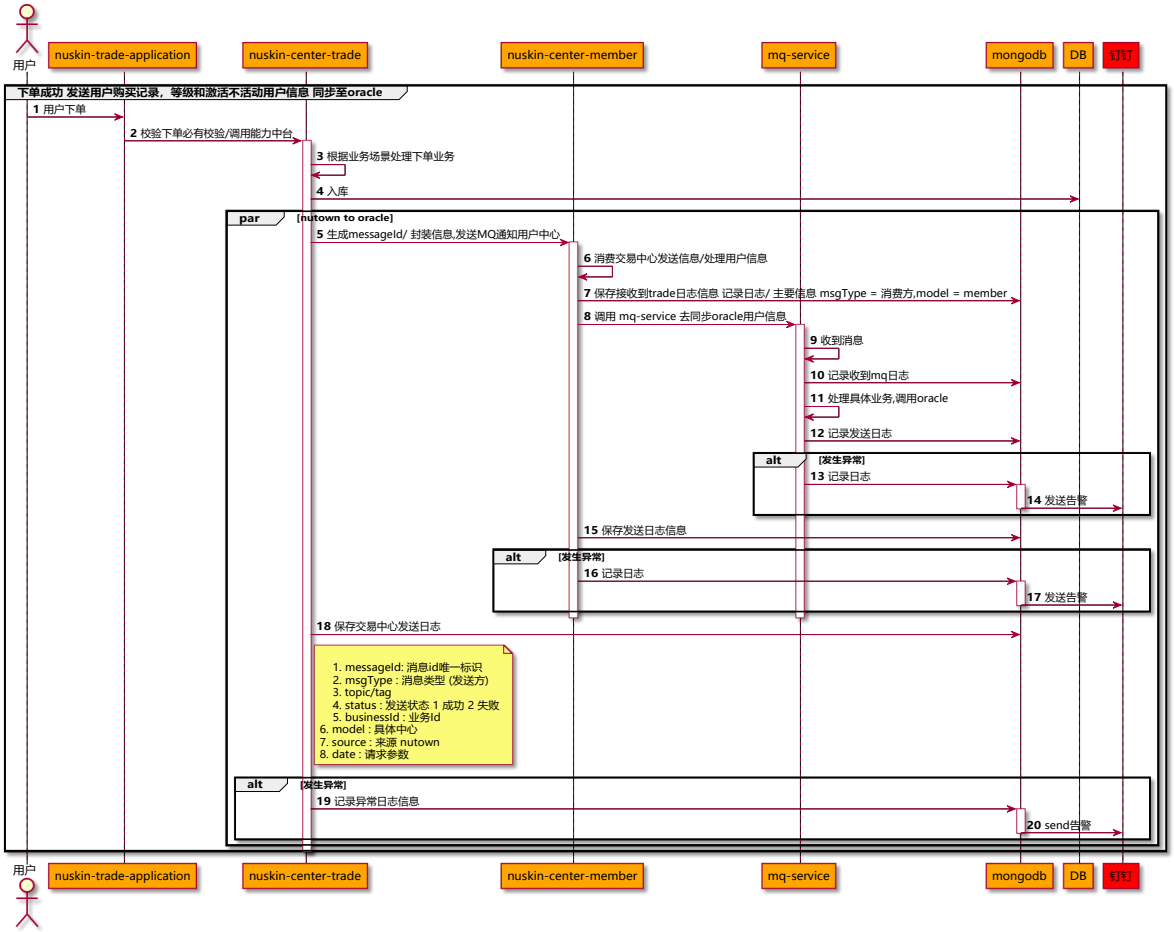
前言:

详细了解钉钉告警请参考 https://ding-doc.dingtalk.com/doc#/serverapi2/qf2nxq ;

钉钉告警触发场景:

时序图


oracle
mq-service
能力中心
mongodb
DB
钉钉
oracle 同步数据到 nutown 数据监控
1 oracle推送数据至nutown
由oracle生成messageId确保整条链路信息
par
[oracle to nutown 之间数据交互监控]
alt
[mq-service正常处理方式]
2 处理第三方数据
3 分发各个中心、messageId唯一标识
4 异步mq入库记录日志
消息入库关键字段:
messageId：链路id,
bussinessId：业务主键,
source:来源,
status: 状态 1 成功
data: 响应内容,
request: 请求体
alt
[nutown 解析失败]
5 处理第三方数据失败、异步mq入库记录日志
消息入库关键字段:
messageId：链路id,
bussinessId：业务主键,
source:来源,
status: 状态 2 失败,
data: 具体错误日志,
request: 请求体
6 send钉钉告警
钉钉告警条件判断:
1. 是否开启钉钉告警
2. 检查 messageId、source、
model、bussinessId是否不为空
3. 跳转详情url是否配置
4. 重试次数检验，决定是否告警，
同一个Mymessageld配置告警次数
par
[mq-service 分发数据至各个中心]
7 mq-service解析成功通知能力中台
8 根据业务场景解析数据
alt
[正常解析]
9 入库
10 记录日志
alt
[解析异常]
11 记录异常日志信息
12 发送钉钉告警
1. 具体配置信息请参考步骤6配置信息
oracle
mq-service
能力中心
mongodb
DB
钉钉



messageId: 使用分布式雪花算法生成的唯一标识

MymessageId: mq内置的messageId

3.2（oracle 至 mq-service / mq-service 至 oracle）之正常日志接口设计

方法名：saveOracleRequestLogMongodb(OracleRequestLogDto oracleRequestLogDto)
请求方式： mq形式.
参　　数：OracleRequestLogDto

request	请求数据
api	具体接口
createTime	创建时间
messageId	唯一请求标识
businessKey	业务key
businessValue	业务vakue
source	来源
messageType	MQ类型

> MQ类型 -> send_error_log 发送mq异常日志、receive_error_log 接收mq异常日志、RECEIVE_LOG、send_log 发送日志 、before_send_error_log 发送mq前异常日志

consumers 预期消费者
response 消息响应
parseStatus 解析状态
model 当前中心
status 解析状态 1 成功 2 失败
sendTimes 发送次数
hasConsumed 已经消费
MymesageId
返回值：void

接口说明:

1. 判断 messageId 、businessId、source、msgType 是否为空，为空发送钉钉告警由于参数不满足导致消息不入库原因, 以下业务均不执行。
2. 根据 messageId 查询 mongodb 是否有正常日志信息。
3. 有数据则累加 成功次数 + 1 ，失败次数 == 查询错误日志获取
4. 没有数据，则需要根据 messageId 查询错误日志信息获取失败次数，成功次数 == 1;

c. 根据messageId删除错误日志信息 （同一个messageId在mongodb中只能存在一条日志信息）

d. 保存本次日志至mongodb

3.3（oracle 至 mq-service / mq-service 至 oracle）之错误日志接口设计

方法名 ： saveOracleRequestErrorLog(OracleRequestErrorLogDto oracleRequestLogDto)
请求方式： mq形式.
参 数： OracleRequestErrorLogDto ()
request 请求数据
api 具体接口
exception 异常描述
messageId 唯一请求标识
businessKey 业务key
businessValue 业务vakue
method 请求方法
source 来源
messageType MQ类型

> MQ类型 -> send_error_log 发送mq异常日志、receive_error_log 接收mq异常日志、RECEIVE_LOG、send_log 发送日志 、before_send_error_log 发送mq前异常日志

consumers 预期消费者
response 消息响应
parseStatus 解析状态
model 当前中心
status 解析状态 1 成功 2 失败
sendTimes 发送次数
hasConsumed 已经消费
返回值：void

接口说明

1. 判断 messageId 、businessId、source、msgType是否为空，为空则告警
2. 根据 messageId 查询 mongodb 是否有错误日志信息。
    1. 有数据则累加 失败次数 + 1 ，成功次数获取正常日志
    2. 没有数据，则需要根据 messageId 查询正常日志信息获取成功次数，错误次数 == 1;
3. 删除正常日志信息、入库。
4. 发送钉钉告警

3.4 各中心链路信息MQ日志接口设计

方法名 ：saveMqLogMongodbDto(MqMsgLogDto mqMsgLogDto)
请求方式： mq形式.
参 数 ： MqMsgLogDto()
messageId: 消息id
businessType 业务类
msgType MQ类型
topic
tag
producerModel 生产者方
status 解析状态 1 成功 2 失败
sendTimes 成功次数
sendErrorTimes 失败次数
businessId 业务号
msg
errorMsg 错误信息
model 模块

source 来源
返回值：void

接口说明：

1. 该接口用于保存各中心MQ调用保存日志信息、此接口保存正常/错误日志
2. 校验必有参数。souces、 messageId、model、msgType
3. 封装日志信息保存mongodb
4. 判断MQ类型是否为接收mq异常日志、发送mq前异常日志、发送mq异常日志。满足则进行钉钉告警。
    1. 判断告警开发是否开启
    2. 获取MQ重试次数
    3. 封装信息
    4. 完成告警

3.4 后管日志详情接口设计

方法名 ： queryLoggerDetail()
参 数 ： messageId
请求方式 ： GET请求
messageId: 消息id
返回值 ： MonitoringLogDetailDto

接口说明

1. 根据messageId 获取整条链路日志信息，调用mongodb中心。
2. 根据messageId 获取mq-service正常日志，调用mongodb中心。
3. 根据messageId 获取mq-service错误日志，调用mongodb中心。
4. 一个messageId mq-serice日志正常或者错误只会存在一个；
5. 判断步骤2不为空
6. 判断步骤3不为空，满足则直接抛异常；
7. 调用步骤7
8. 判断步骤3不为空
9. 判断步骤2不为空，满足则直接抛异常；
10. 调用步骤7
11. 封装日志信息，返回前端。

3.5 后管日志列表接口设计

方法名　　：governanceLoggerList（）
请求方式　：　POST 请求
参　　数　：
messageId:　　消息id
businessValue:　　业务字段
pageNum
pageSize

返回值：PageInfo

接口说明：

1. 根据条件分页查询操作日志信息，按照操作时间倒序返回；

四、MQ改造详细设计

4.1 消息队列 RocketMQ

消息队列 RocketMQ 是一款阿里云基于 Apache RocketMQ 构建的一款低延迟、高可用、高并发、高可靠的一款分布式消息中间件，RocketMQ 可为分布式应用系统提供异步解耦和削峰填谷的能力，同时也具备互联网应用所需的海量消息堆积、高吞吐、可靠重试等特性。

4.2 核心概念

- Topic：消息主题，一级消息类型，生产者向其发送消息。
- 生产者：也称为消息发布者，负责生产并发送消息至 Topic。
- 消费者：也称为消息订阅者，负责从 Topic 接收并消费消息。
- 消息：生产者向 Topic 发送并最终传送给消费者的数据和（可选）属性的组合。
- 消息属性：生产者可以为消息定义的属性，包含 Message Key 和 Tag。
- Group：一类生产者或消费者，这类生产者或消费者通常生产或消费同一类消息，且消息发布或订阅的逻辑一致。

关于MQ详细介绍请点击 https://help.aliyun.com/document_detail/155952.html?spm=a2c4g.11186623.6.549.1c6b7e80hh4M0o

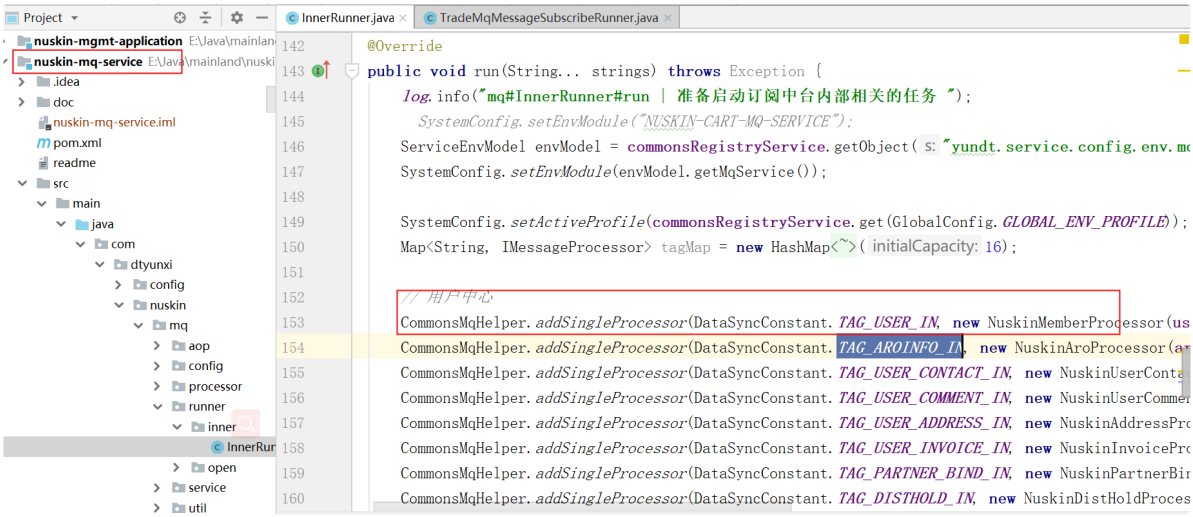
4.3 CommandLineRunner 介绍：

我们平常在使用springBoot时候，我们通常都会进行预加载数据，比如缓存预热、读取配置.. 等等，那么SpringBoot为我们提供了一个CommandLineRunner；

CommandLineRunner是一个接口我们需要时，只需实现该接口就行。如果存在多个加载的数据，我们也可以使用@Order注解来排序。

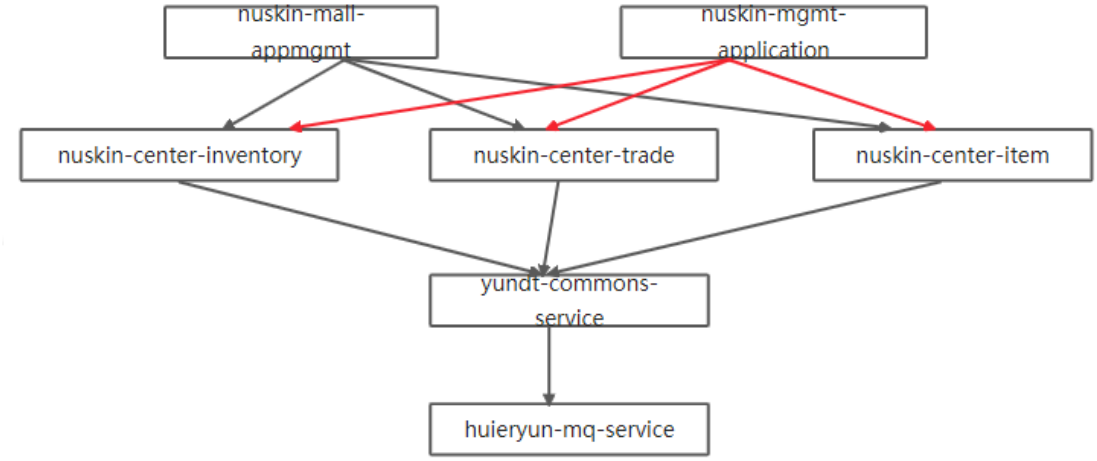
关于CommandLineRunner详细介绍，请翻阅资料

4.4 mq 消费者现状：



综上，如图所示，每个中心如果采用MQ进行业务处理，都必须定义一个类 implement CommandLineRunner 从而进行监听Broker服务器，从而定期的从 Name -service上实时拉取Topic的路由信息，进行订阅消息； 目前这样代码是没有任何问题，但是会出现我们经常说起的名词，高耦合、重复代码太多。为了减少代码的耦合度，本次改造将采用注解的方式。

各中心依赖



综 1,2 图所示，yundt-commons-service 工程都会被我们能力中台，应用层所依赖，因此MQ注解的方式改造将定于yundt-commons-service 工程；

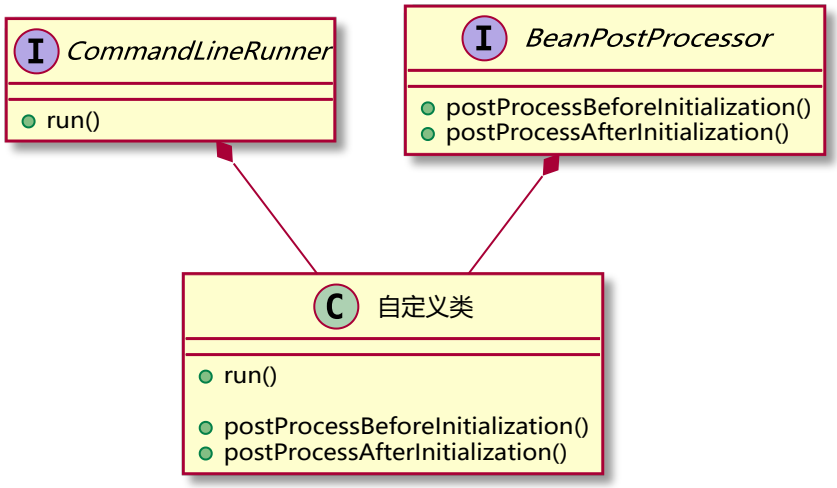
yundt-commons-service 添加以下代码

1、创建注解@MQDesc 属性包括 topic 、comsumer (中心名)、tag 、msgType

```java
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface MQDesc {

    String topic();

    String consumer();

    String tag();

    String msgType() default "single";
```

2、定义一个类 implement CommandLineRunner ， BeanPostProcessor (Bean 的后置处理器) 项目启动时开始订阅消息。

类图说明



1. Spring ioc 为我们提供了一个扩展性接口，也就是说项目启动时，创建srping容器，将带有注解，或者xml形式的对象纳入spring容器，那对创建对象之前，我们可以做什么，纳入容器之后 (Bean 对象)，我们又阔以做什么，本文将采用 spring ioc 的后置处理器，来收集项目中那些类被MQDesc注解标记。可以通过注解中的属性获取对应的属性。

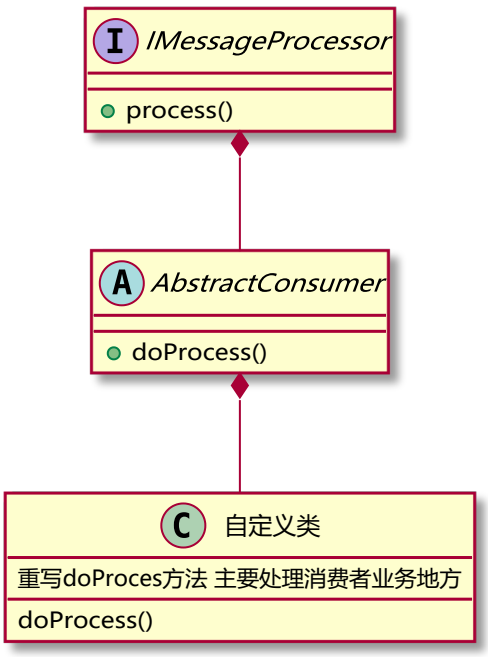
2. 综上所述，收集过程做完了之后，我们可以利用CommandLineRunner，项目启动的时候，我们可以监听 name -service 使用topic路由对borker进行订阅消息，从而达到MQ的正常消费。

本次将采用消息治理改造方案 @MQDesc方案 中的方案二consumer是以中心为纬度，每一个中心都会创建一个Consumer 来进行监听消息。

4.5mq 消费者改造：

综上所述，只是对冗余代码做了处理；使用注解方式替换了我们以前的方式，针对消费者记录日志，我们定义一个抽象类 AbstractConsumer ，我们在写mq消费业务只需自己定义一个类，extends AbstractConsumer 。

类图




```java
public abstract class AbstractConsumer implements IMessageProcessor<MessageVo> {

    @Autowired
    private ICommonsMqService commonsMqService;

    Logger logger = LoggerFactory.getLogger(AbstractConsumer.class);

    /**
     * 消费者业务相关实现
     *
     * @param message 消息内容
     * @return 消费者处理结果
     */
    public abstract MessageResponse doProcess(MessageVo message);
```

doProcess 将是消费者处理业务的方法

- AbstractConsumer implements IMessageProcessor 因此需实现 IMessageProcessor中

process 方法 ， 我们将会再次对业务进行日志记录，值得注意，我们将会 process 方法中进行 try 、catch

如下图所示。 大家只要关心自己的业务，有异常请抛异常，不建议由于发生业务不正确，直接return 的情况

```
public MessageResponse process(MessageVo message) {
    try {
        logger.info("开始发送mq ---------------- | {}", JSONObject.toJSONString(message));
        MessageResponse response = doProcess(message);
        if (message.isLogReceiveMq()) {
            MqMsgLogDto mqMsgLogDto = CommonsMqHelper.getMqMsgLogDto(message, MessageLogTypeEnum.RECEIVE_LOG)
            MessageVo messageVo = new MessageVo();
            logger.info("mqMsgLogDto | {}", JSONObject.toJSONString(mqMsgLogDto));
            messageVo.setData(mqMsgLogDto);
            commonsMqService.sendSingleMessageAsync(MessageTopic.OPERATE_LOG_TOPIC, MessageQueue.NUSKIN_CO
        }
        return response;
    } catch (Exception e) {
        logger.info("MQ==》 消费失败 {},error:{}", message.toString(),ExceptionUtils.getStackTrace(e));
        if (message.isLogReceiveMq()) {
            MessageVo messageVo = new MessageVo();
            messageVo.setStatus(SapStatusEunm.ERROR.getStatus());
            MqMsgLogDto mqMsgLogDto = CommonsMqHelper.getMqMsgLogDto(message, MessageLogTypeEnum.RECEIVE_ERR
```

消费者业务处理地方

- 我们自己定义类，根据mq现状描述，需要在自定义加上 @MQDesc 注解、以及将对象纳入spring容器。

```
@Component
@MQDesc(topic = NuskinMessageTopic.FROM_NUTOWN_TO_SAP_TOPIC, consumer = NuskinMessageConsumer.GATEWAY,
        tag = NuskinMessageQueue.CREATE_SALES_ORDER_IN_NUTOWN_TRANSFER_TO_SAP_QUEUE)
public class CreateOrderTransferToSapConsumer extends AbstractConsumer {
```

- topic 可以不用写，但是不推荐
- consumer ： 中心名
- tag :
- msgType : publish / single 二选一

4.6 大陆 & 台湾 mq 发送者改造：

本次改造仅对于nutown 同步数据至 oracle || oracle 同步至 nutown ，比如 扫码购订单取消 需要同步订单信息至oracle，yundt-center-trade 需要发送mq 通知 mq-service 、mq-service 收到消息，处理业务，然后在发送至 oracle，那么这整个链路信息我们都需要记录日志， 如果采用普通的记录日志方式，没发次MQ然后手动封装下日志信息，调用rpc去记录日志，那么出现在很大的代码冗余，本文将针对mq发送时的改造来记录日志信息。

强制

- 发送MQ信息时参数类型我们将统一数据类型定义为 MessageVo
- 我们记录日志会根据messageVo中的属性 doRecord （*是否记录日志 0 不记录， 1 发送端记录日志， 2 消费端记录日志，3 发送消费都记录）默认* doRecord == 0 本次改造针对第三方同步数据 强制 doRecord == 3
- 当然为了确保整体一个链路信息，我们将考虑使用分布式雪花算法生成一个 messageId 来确保一条链路的完整性
- 为了我们可以根据日志查询快速定位到代码 method 全路径即可
- 为了确保那方是发送方 将采用 source == (nutown / oracle)
- bussinessId 将代表业务号，比如同步订单，bussinessId == orderId ，同步用户等级变更 bussinessId == memberId ..

```
MessageVo messageVo = new MessageVo(deliveryOrderHkDto);
messageVo.setBussinessId(deliveryOrderHkDto.get(0).getDeliveryOrderCode());
messageVo.setTraceId(deliveryOrderHkDto.get(0).getMessageId());
messageVo.setDoRecord(MqRecordEnum.SEND_AND_CONSUMER_RECORD.getCode());
messageVo.setSource(SourceEnum.SAP.getSource());
messageVo.setMethod(SapRequestEnum.GENERATE_DELIVERY_ORDER_TO_NUTOWN.getMethod());
commonsMqService.sendSingleMessageAsync(NuskinMessageTopic.FROM_SAP_TO_NUTOWN_TOPIC, NuskinMessageQueue.CRE
```

综上，如图所示，本次发送者的改造介绍完毕。（当然使用MQ非同步第三方则需要遵守 发送MQ信息时参数类型我们将统一数据类型定义为 MessageVo ）

4.7 MQ消费异常，统计日志失败次数、处理业务幂等性

- 累加日志失败次数

  RocketMQ支持消息的重试。可以满足顺序消息、无序消息、点对点消息。

重试次数

消息队列RocketMQ版默认允许每条消息最多重试16次，每次重试的间隔时间如下。

第几次重试	与上次重试的间隔时间	第几次重试	与上次重试的间隔时间
1	10秒	9	7分钟
2	30秒	10	8分钟
3	1分钟	11	9分钟
4	2分钟	12	10分钟
5	3分钟	13	20分钟
6	4分钟	14	30分钟
7	5分钟	15	1小时
8	6分钟	16	2小时

如图所示，RoketMQ 默认支持最多16次。

解决方案

因为我们发送MQ时会生成messageId作为整体链路的唯一性，然而 mongodb 会存储 正常次数、错误次数、因此可以通过messageId查询mongodb，从而进行累加。

- 关于消费失败、幂等性处理

消息幂等：当出现消费者对某条消息重复消费的情况时，重复消费的结果与消费一次的结果是相同的，并且多次消费并未对业务系统产生任何负面影响，那么这整个过程就可实现消息幂等。

例如，在支付场景下，消费者消费扣款消息，对一笔订单执行扣款操作，扣款金额为 100 元。如果因网络不稳定等原因导致扣款消息重复投递，消费者重复消费了该扣款消息，但最终的业务结果是只扣款一次，扣费 100 元，且用户的扣款记录中对应的订单只有一条扣款流水，不会多次扣除费用。那么这次扣款操作是符合要求的，整个消费过程实现了消费幂等。

消息重复的场景如下

1. 发送时消息重复当一条消息已被成功发送到服务端并完成持久化，此时出现了网络闪断或者客户端宕机，导致服务端对客户端应答失败。 如果此时生产者意识到消息发送失败并尝试再次发送消息，消费者后续会收到两条内容相同并且 Message ID 也相同的消息。
2. 投递时消息重复消息消费的场景下，消息已投递到消费者并完成业务处理，当客户端给服务端反馈应答的时候网络闪断。为了保证消息至少被消费一次，消息队列 RocketMQ 版的服务端将在网络恢复后再次尝试投递之前已被处理过的消息，消费者后续会收到两条内容相同并且 Message ID 也相同的消息。

3. 负载均衡时消息重复（包括但不限于网络抖动、Broker 重启以及消费者应用重启）当消息队列 RocketMQ 版的 Broker 或客户端重启、扩容或缩容时，会触发 Rebalance，此时消费者可能会收到重复消息。

解决方案

我们在发送的时候无须关心消息的key来作为业务的幂等校验，因为我们发送消息，进行消息的投递，messageVo肯定包括这些信息，我们无须多次一举。从而进行消费的时候可以根据具体场景，具体业务做幂等；

- 关于MQ乱序问题

  场景分析:

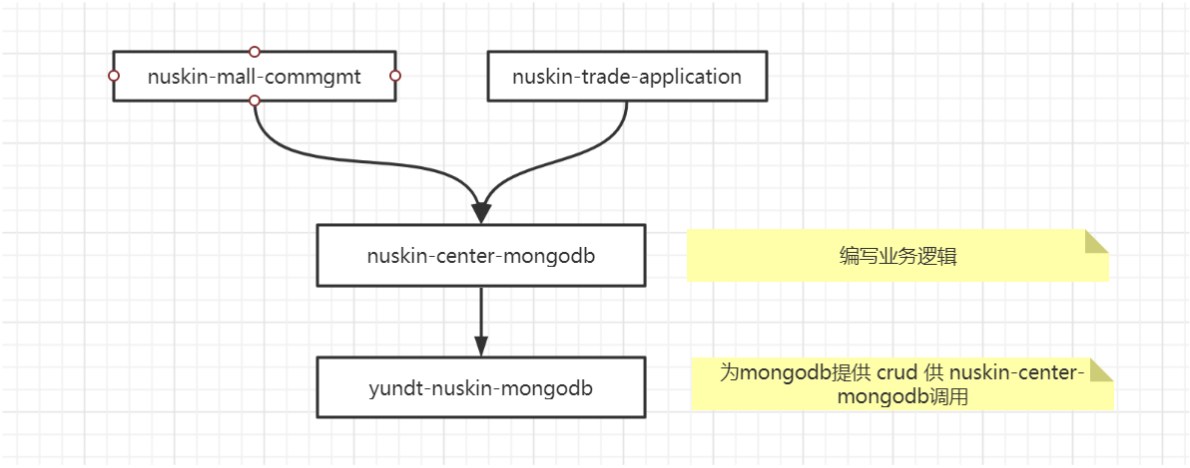
oracle 推送两笔订单信息到 nutown，那推送第一笔是错误的，第二笔是正确的，我们这边由于消息的堆积、或者一些不可控的因素导致 MQ乱序问题，那我们这边的数据先收到的是第二笔正确的，然后在收到第一笔错误的，目前没做任何处理。导致我们nutown 和 oracle 数据不同步。

解决方案:

需要添加一个时间戳来确保这条数据是否要被消费，时间戳需要入库，消费者可以通过业务ID查询原有的数据，如果当前时间戳小于数据库时间戳，那么该条消息属于旧消息，可以不消费。

五、 MongoDB 基础 jar 包说明

项目依赖



如图所示，目录三能力中心的接口详设都是在nuskin-center-mongodb中定义，yundt-nuskin-mongodb只是对nuskin-center-mongodb提供简单的crud

- 时间格式采用的是字符串，做区间查询比较慢

解决方案

- 将考虑将日期格式转化成字符串形式的毫秒值进行存储

六 、 针对改造待解决的课题疑问点方案设计

6.1 关于MyMessageId作为幂等性处理

使用业务字段进行幂等校验

6.2 关于MongoDB性能、日志入库问题

- mongodb性能

可以考虑针对于索引优化，+ 日志定期清理，提高用户的体验。根据条件全部是精确搜索，不采取模糊搜索。

- 日志入库

需要对整条链路信息进行日志存储、举例: 用户下单成功需要同步用户的购买记录到oracle 那么整条链路的信息就是 center-trade -> center-member-> mq-service 三个中心都要入库

6.3 关于ACK提交机制、以及MQ乱序问题

- ack提交机制

关于ack提交机制，主要是以业务为准，大家只要关心自己的业务*，有异常请抛异常，不建议由于发生业务不正确，直接return 的情况**导致消息不能够重试

- mq乱序问题

需要添加一个时间戳来确保这条数据是否要被消费，时间戳需要入库，消费者可以通过业务ID查询原有的数据，如果当前时间戳小于数据库时间戳，那么该条消息属于旧消息，可以不消费。