链滴

消息架构的设计难题以及应对之道

作者：jianzh5

原文链接：https://ld246.com/article/1604454930039

来源网站：链滴

概述

在微服务开发中我们经常会引入消息中间件实现业务解耦，执行异步操作， 现在让我们来看看使用消中间件的好处和弊端。

首先需要肯定是使用消息组件有很多好处，其中最核心的三个是：解耦、异步、削峰。

- **解耦**：客户端只要讲请求发送给特定的通道即可，不需要感知接收请求实例的情况。
- **异步**：将消息写入消息队列，非必要的业务逻辑以异步的方式运行，加快响应速度。
- **削峰**：消息中间件在消息被消费之前一直缓存消息，消息处理端可以按照自己处理的并发量从消息列中慢慢处理消息，不会一瞬间压垮业务。

当然消息中间件并不是银弹，引入消息机制后也会有如下一些弊端：

- **潜在的性能瓶颈**：消息代理可能会存在性能瓶颈。幸运的是目前主流的消息中间件都支持高度的横扩展。
- **潜在的单点故障**：消息代理的高可用性至关重要，否则系统整体的可靠性将受到影响，幸运的是大数消息中间件都是高可用的。
- **额外的操作复杂性**：消息系统是一个必须独立安装、配置和运维的系统组件，增加了运维的复杂度。

这些弊端我们借助消息中间件本身提供的扩展、高可用能力可以解决，但是要真正用好消息中间件我还需要关注可能会遇到的一些设计难题。

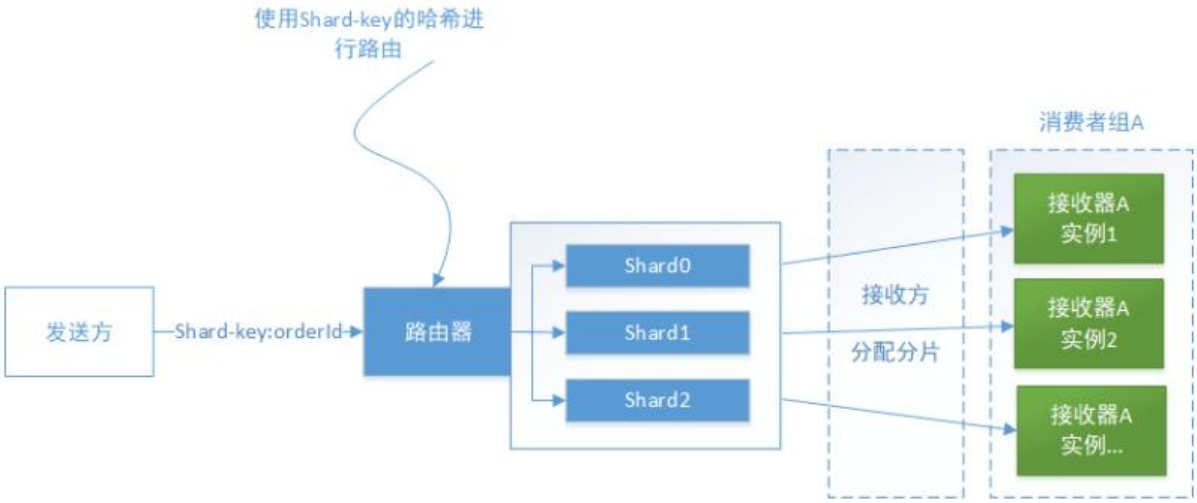
处理并发和顺序消息

在生产环境中为了提高消息处理的能力以及应用程序的吞吐量，一般会将消费者部署多个实例节点。么带来的挑战就是**如何确保每个消息只被处理一次，并且是按照他们的发送顺序来处理的。**

例如：假设有3个相同的接收方实例从同一个点对点通道读取消息，发送方按顺序发布了 Order Created、Order Updated 和 Order Cancelled 这3个事件消息。简单的消息实现可能就会同事讲每个消息不同的接收方。若由于网络问题导致延迟，消息可能没有按照他们发出时的顺序被处理，这将导致奇的行为，服务实例可能在另一个服务器处理 Order Created 消息之前处理 Order Cancelled消息。

Kafka 使用的解决方案是使用分片（分区）通道。整体解决方案分为三个部分：

1. 一个主题通道由多个分片组成，每个分片的行为类似一个通道。
2. 发送方在消息头部指定分片键如orderId，Kafka使用分片键将消息分配给特定的分片。
3. 将接收方的多个实例组合在一起，并将他们视为相同的逻辑接收方（消费者组）。kafka将每个分分配给单个接收器，它在接收方启动和关闭时重新分配分片。



如上图所示，每个Order事件消息都将orderId作为其分片键。特定订单的每个事件都发布到同一个分。而且该分片中的消息始终由同一个接收方实例读取，因此这样就能够保证按顺序处理这些消息。

处理重复消息

引入消息架构必须要解决的另一个挑战是处理重复消息。在理想情况下，消息代理应该只传递一次消，但保证消息有且仅有一次的消息传递的成本通常很高。相反，很多消息组件承诺至少保证成功传递次消息。

在正常情况下，消息组件只会传递一次消息。但是当客户端、网络或消息组件故障可能导致消息被多传递。假设客户端在处理消息后发送确认消息前，他的数据库崩溃了，这时消息组件将再次发送未确的消息，在数据库重新启动时向该客户端发送。

处理重复消息有以下两种不同的方法：

- 编写幂等消息处理程序
- 跟踪消息并丢弃重复项

编写幂等消息处理器

如果应用程序处理消息的逻辑是满足幂等的，那么重复消息就是无害的。程序的幂等性是指，即使这应用被相同输入参数多次重复调用时，也不会产生额外的效果。例如：取消一个已经取消的订单，就一个幂等性操作。同样，创建一个已经存在的订单操作也必是这样。满足幂等的消息处理程序可以被心的执行多次，只要消息组件在传递消息时保持相同的消息顺序。

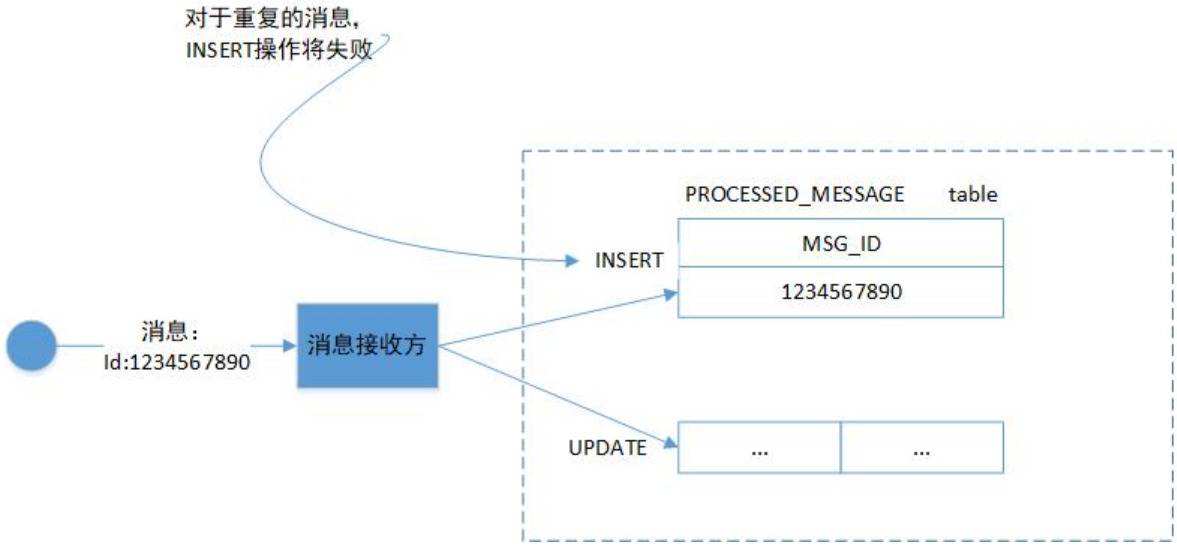
但是不幸的是，应用程序通常不是幂等的。或者你现在正在使用的消息组件在重新传递消息时不会保排序。重复或无序消息可能会导致错误。在这种情况下，你需要编写跟踪消息并丢弃重复消息的消息理程序。

跟踪消息并丢弃重复消息

考虑一个授权消费者信用卡的消息处理程序。它必须为每个订单仅执行一次信用卡授权操作。这段应程序每次调用时都会产生不同的效果。如果重复消息导致消息处理程序多次执行该逻辑，则应用程序行为将不正确。执行此类应用程序逻辑的消息处理程序必须通过检测和丢弃重复消息而让它成为幂等。

一个简单的解决方案是消息接收方使用 message id 跟踪他已处理的消息并丢弃任何重复项。例如，

数据库表中存储它消费的每条消息的 message id。



当接收方处理消息时，它将消息的 message id 作为创建和变更业务实体的事务的一部分记录在数据里。如上图所示，接收方将包含message id 的行插入 PROCESSED_MESSAGE表。如果消息是重复，则INSERT将失败，接收方可以选择丢弃该消息。

另一个解决方案是消息处理程序在应用程序表，而不是专门表中记录 message id。当时用具有受限务模型的NoSQL数据库时，此方法特别有用，因为 NoSQL数据库通常不支持将针对两个表的更新作数据库事务。

处理事务性消息

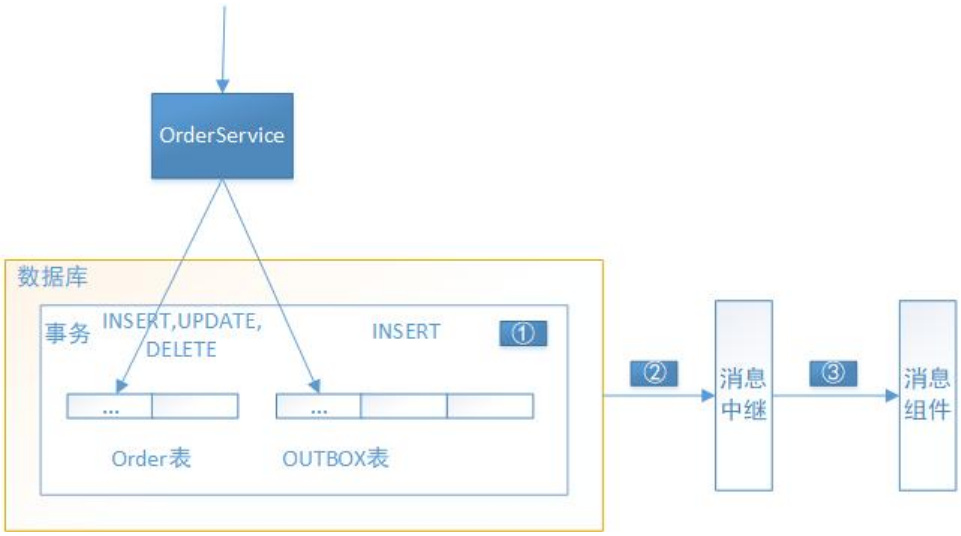
服务通常需要在更新数据库的事务中发布消息，数据库更新和消息发送都必须在事务中进行，否则服可能会更新数据库然后在发送消息之前崩溃。

如果服务不以原子方式执行者两个操作，则类似的故障可能使系统处于不一致状态。

接下来我们看一下常用的保证事务消息的两种解决方案，最后再看看现代消息组件RocketMQ的事务消息解决方案。

使用数据库表作为消息队列

如果你的应用程序正在使用关系型数据库，要保证数据的更新和消息发送之间的事务可以直接使用**事性发件箱模式，Transactional Outbox。**



此模式使用数据库表作为临时消息队列。如上图所示，发送消息的服务有个OUTBOX数据表，在进行INSERT、UPDATE、DELETE 业务操作时也会给OUTBOX数据表INSERT一条消息记录，这样可以保证子性，因为这是基于本地的ACID事务。

OUTBOX表充当临时消息队列，然后我们在引入一个消息中继（MessageRelay）的服务，由他从OUBOX表中读取数据并发布消息到消息组件。

消息中继的实现可以很简单，只需要通过定时任务定期从OUTBOX表中拉取最新未发布的数据，获取数据后将数据发送给消息组件，最后将完成发送的消息从OUTBOX表中删除即可。

使用事务日志发布事件

另外一种保证事务性消息的方式是基于数据库的事务日志，也就是所谓的数据变更捕获，Change Dat Capture，简称CDC。

一般数据库在数据发生变更的时候都会记录事务日志（Transaction Log），比如MySQL的binlog。务日志可以简单的理解成数据库本地的一个文件队列，它主要记录按时间顺序发生的数据库表变更记。

这里我们利用alibaba开源的组件canal结合MySQL来说明下这种模式的工作原理。

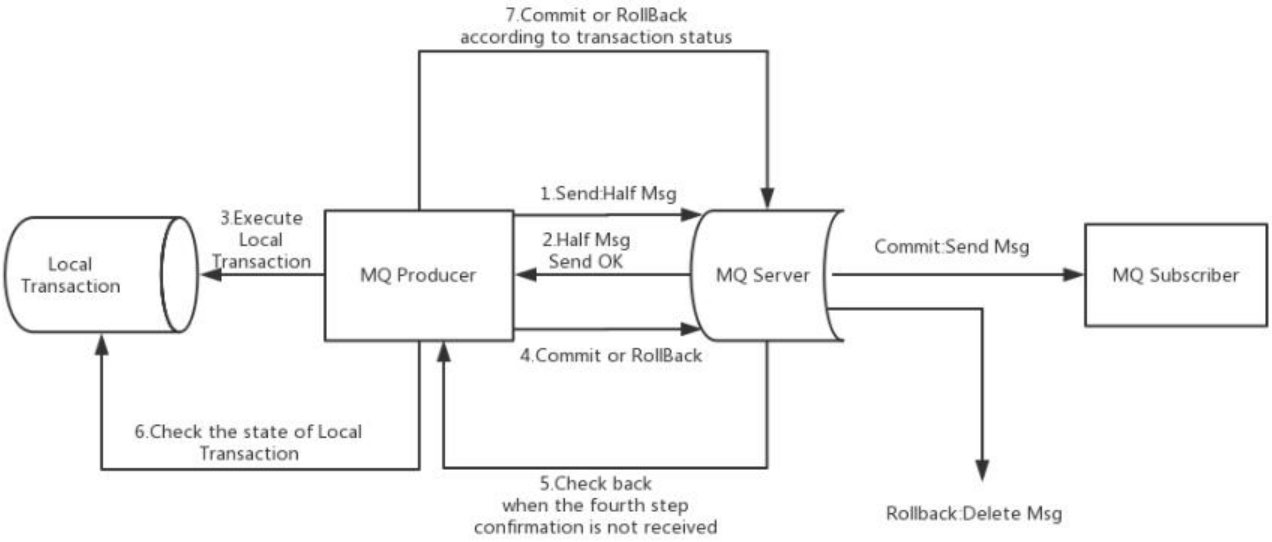
更多操作说明可以参考官方文档：https://github.com/alibaba/canal

canal工作原理

- canal 模拟 MySQL slave 的交互协议，把自己伪装成一个MySQL的 slave节点 ，向 MySQL master 发送dump 协议；
- MySQL master 收到 dump 请求，开始推送 binary log 给 slave (即 canal)；
- canal 解析 binary log 对象(原始为 byte 流)，然后可以将解析后的数据直接发送给消息组件。

RocketMQ事务消息解决方案

Apache RocketMQ在4.3.0版中已经支持分布式事务消息，RocketMQ采用了2PC的思想来实现了提事务消息，同时增加一个补偿逻辑来处理二阶段超时或者失败的消息，如下图所示。



RocketMQ实现事务消息主要分为两个阶段：正常事务的发送及提交、事务信息的补偿流程。

整体流程为：

- 正常事务发送与提交阶段

1、生产者发送一个半消息给MQServer（半消息是指消费者暂时不能消费的消息）

2、服务端响应消息写入结果，半消息发送成功

3、开始执行本地事务

4、根据本地事务的执行状态执行Commit或者Rollback操作

- 事务信息的补偿流程

1、如果MQServer长时间没收到本地事务的执行状态会向生产者发起一个确认回查的操作请求

2、生产者收到确认回查请求后，检查本地事务的执行状态

3、根据检查后的结果执行Commit或者Rollback操作

补偿阶段主要是用于解决生产者在发送Commit或者Rollback操作时发生超时或失败的情况。

在生产者使用RocketMQ发送事务消息的时候我们也会借鉴第一种方案即自建一张事务日志表，然后执行本地事务的时候同时生成一条事务日志记录，让本地事务与日志事务在同一个方法中，同时添加 Transactional 注解，保证两个操作事务是一个原子操作。**这样如果事务日志表中有这个本地事务的息，那就代表本地事务执行成功，需要Commit，相反如果没有对应的事务日志，则表示没执行成功需要Rollback。**