← BACK

解析 RocketMQ 业务消息——事务消息

作者 | 合伯

技术探索 功能特性

2022年8月3日

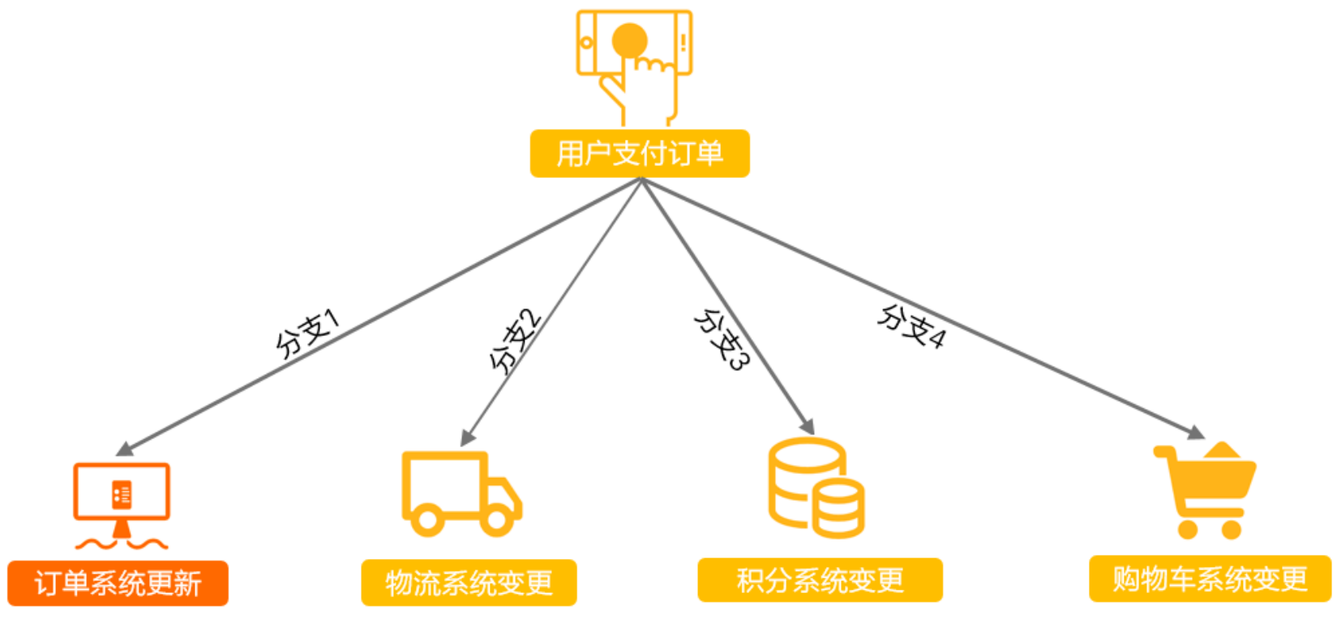
> 引言：在分布式系统调用场景中存在这样一个通用问题，即在执行一个核心业务逻辑的同时，还需要调用多个下游做业务处理，而且要求多个下游业务和当前核心业务必须同时成功或者同时失败，进而避免部分成功和失败的不一致情况出现。简单来说，消息队列中的“事务”，主要解决的是消息生产者和消费者的数据一致性问题。本篇文章通过拆解 RocketMQ 事务消息的使用场景、基本原理、实现细节和实战使用，帮助大家更好的理解和使用 RocketMQ 的事务消息。

点击下方链接，查看视频讲解： https://yqh.aliyun.com/live/detail/29199

场景：为什么需要事务消息

以电商交易场景为例，用户支付订单这一核心操作的同时会涉及到下游物流发货、积分变更、购物车状态清空等多个子系统的变更。当前业务的处理分支包括：

- 主分支订单系统状态更新：由未支付变更为支付成功；
- 物流系统状态新增：新增待发货物流记录，创建订单物流记录；
- 积分系统状态变更：变更用户积分，更新用户积分表；
- 购物车系统状态变更：清空购物车，更新用户购物车记录。



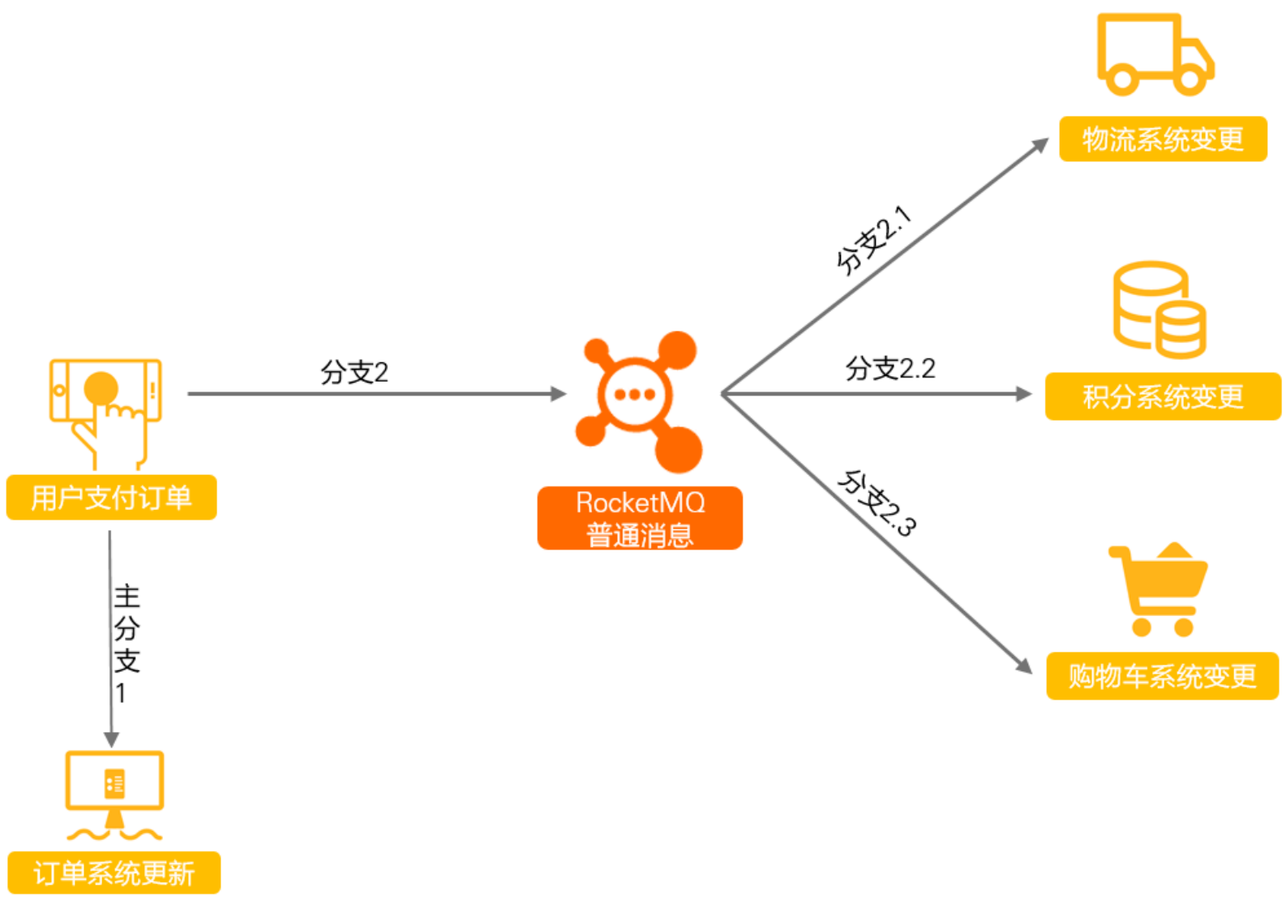
分布式系统调用的特点是：一个核心业务逻辑的执行，同时需要调用多个下游业务进行处理。因此，如何保证核心业务和多个下游业务的执行结果完全一致，是分布式事务需要解决的主要问题。

传统 XA 事务方案：性能不足

为了保证上述四个分支的执行结果一致性，典型方案是基于XA协议的分布式事务系统来实现。将四个调用分支封装成包含四个独立事务分支的大事务，基于XA分布式事务的方案可以满足业务处理结果的正确性，但最大的缺点是多分支环境下资源锁定范围大，并发度低，随着下游分支的增加，系统性能会越来越差。

基于普通消息方案：一致性保障困难

将上述基于 XA 事务的方案进行简化，将订单系统变更作为本地事务，剩下的系统变更作为普通消息的下游来执行，事务分支简化成普通消息+订单表事务，充分利用消息异步化的能力缩短链路，提高并发度。



该方案中消息下游分支和订单系统变更的主分支很容易出现不一致的现象，例如：

- 消息发送成功，订单没有执行成功，需要回滚整个事务；
- 订单执行成功，消息没有发送成功，需要额外补偿才能发现不一致；
- 消息发送超时未知，此时无法判断需要回滚订单还是提交订单变更。

基于RocketMQ分布式事务消息：支持最终一致性

上述普通消息方案中，普通消息和订单事务无法保证一致的本质原因是普通消息无法像单机数据库事务一样，具备提

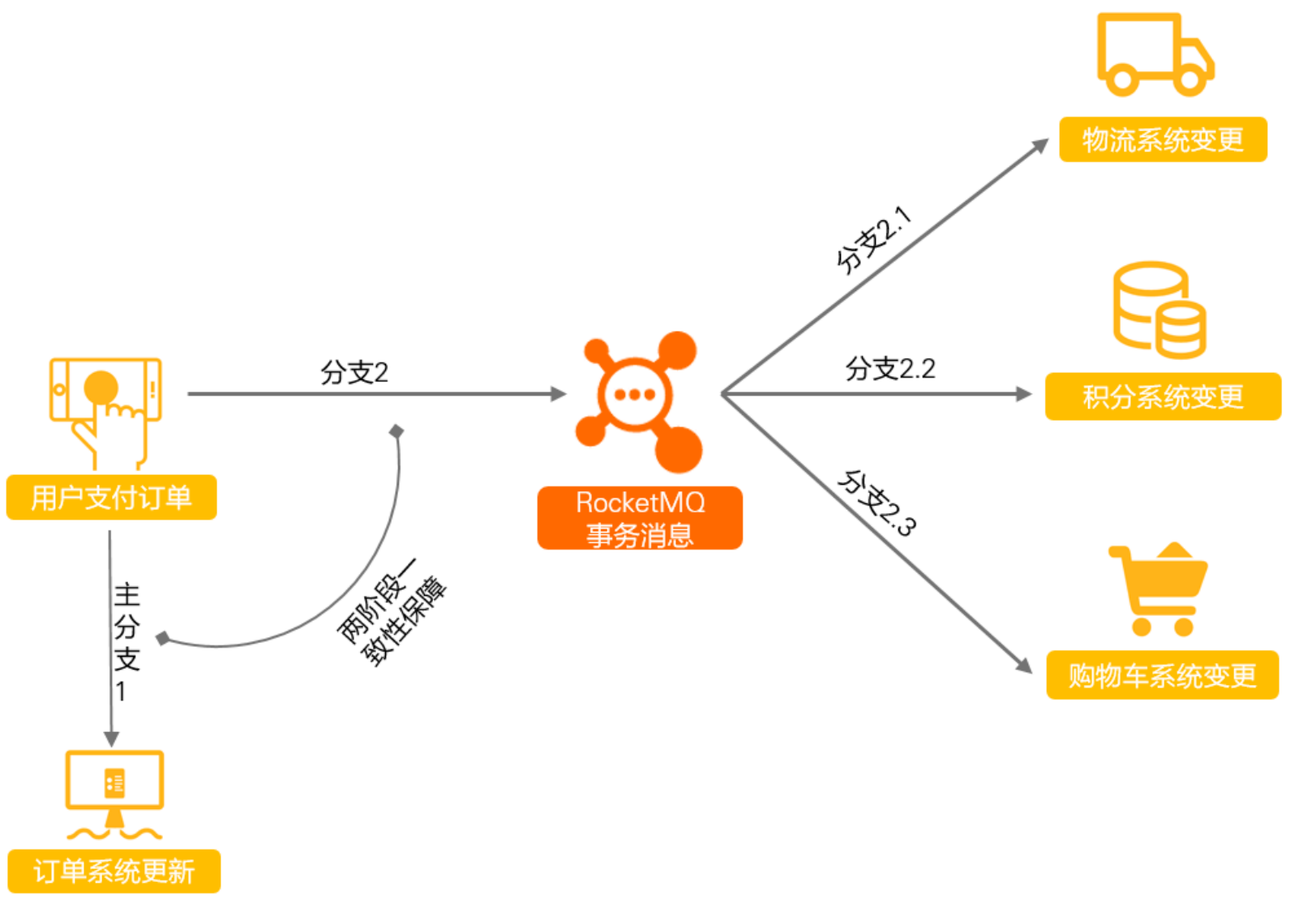
本页内容

热门标签

行业实践 技术探索 社区动态 最佳实践 功能特性 云原生 生态集成 事件驱动架构 物联网 可观测 高可用 微服务 流处理

交、回滚和统一协调的能力。

而基于消息队列 RocketMQ 版实现的分布式事务消息功能，在普通消息基础上，支持二阶段的提交能力。将二阶段提交和本地事务绑定，实现全局提交结果的一致性。



消息队列 RocketMQ 版事务消息的方案，具备**高性能、可扩展、业务开发简单**的优势。

基本原理

概念介绍

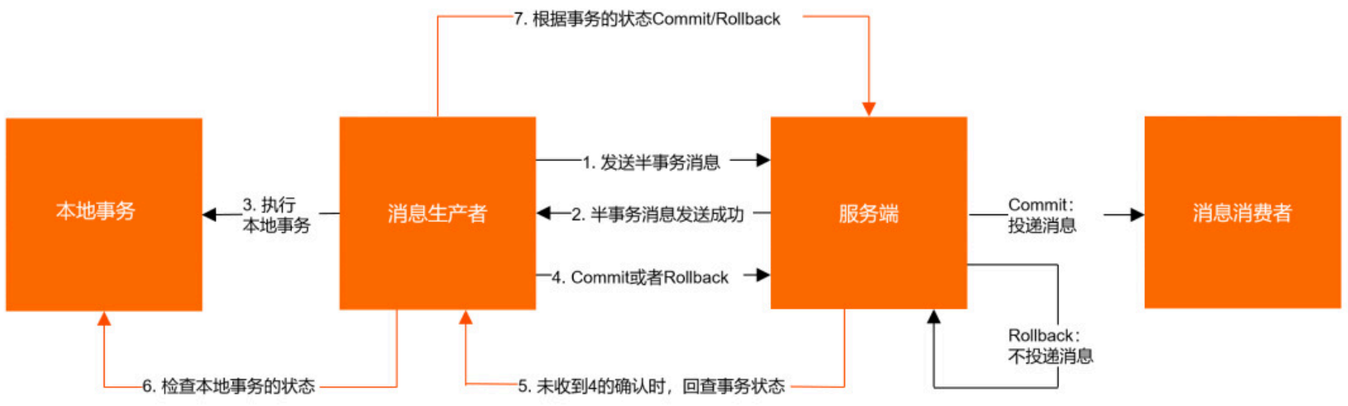
- **事务消息**：RocketMQ 提供类似 XA 或 Open XA 的分布式事务功能，通过 RocketMQ 事务消息能达到分布式事务的最终一致；
- **半事务消息**：暂不能投递的消息，生产者已经成功地将消息发送到了 RocketMQ 服务端，但是 RocketMQ 服务端未收到生产者对该消息的二次确认，此时该消息被标记成“暂不能投递”状态，处于该种状态下的消息即半事务消息；
- **消息回查**：由于网络闪断、生产者应用重启等原因，导致某条事务消息的二次确认丢失，RocketMQ 服务端通过扫描发现某条消息长期处于“半事务消息”时，需要主动向消息生产者询问该消息的最终状态（Commit 或是 Rollback），该询问过程即消息回查。

事务消息生命周期

- **初始化**：半事务消息被生产者构建并完成初始化，待发送到服务端的状态；
- **事务待提交**：半事务消息被发送到服务端，和普通消息不同，并不会直接被服务端持久化，而是会被单独存储到事务存储系统中，等待第二阶段本地事务返回执行结果后再提交。此时消息对下游消费者不可见；
- **消息回滚**：第二阶段如果事务执行结果明确为回滚，服务端会将半事务消息回滚，该事务消息流程终止；
- **提交待消费**：第二阶段如果事务执行结果明确为提交，服务端会将半事务消息重新存储到普通存储系统中，此时消息对下游消费者可见，等待被消费者获取并消费；
- **消费中**：消息被消费者获取，并按照消费者本地的业务逻辑进行处理的过程。此时服务端会等待消费者完成消费并提交消费结果，如果一定时间后没有收到消费者的响应，RocketMQ 会对消息进行重试处理。具体信息，请参见消息重试；
- **消费提交**：消费者完成消费处理，并向服务端提交消费结果，服务端标记当前消息已经被处理（包括消费成功和失败）；RocketMQ 默认支持保留所有消息，此时消息数据并不会立即被删除，只是逻辑标记已消费。消息在保存时间到期或存储空间不足被删除前，消费者仍然可以回溯消息重新消费。
- **消息删除**：当消息存储时长到期或存储空间不足时，RocketMQ 会按照滚动机制清理最早保存的消息数据，将消息从物理文件中删除。

事务消息基本流程

事务消息交互流程如下图所示：

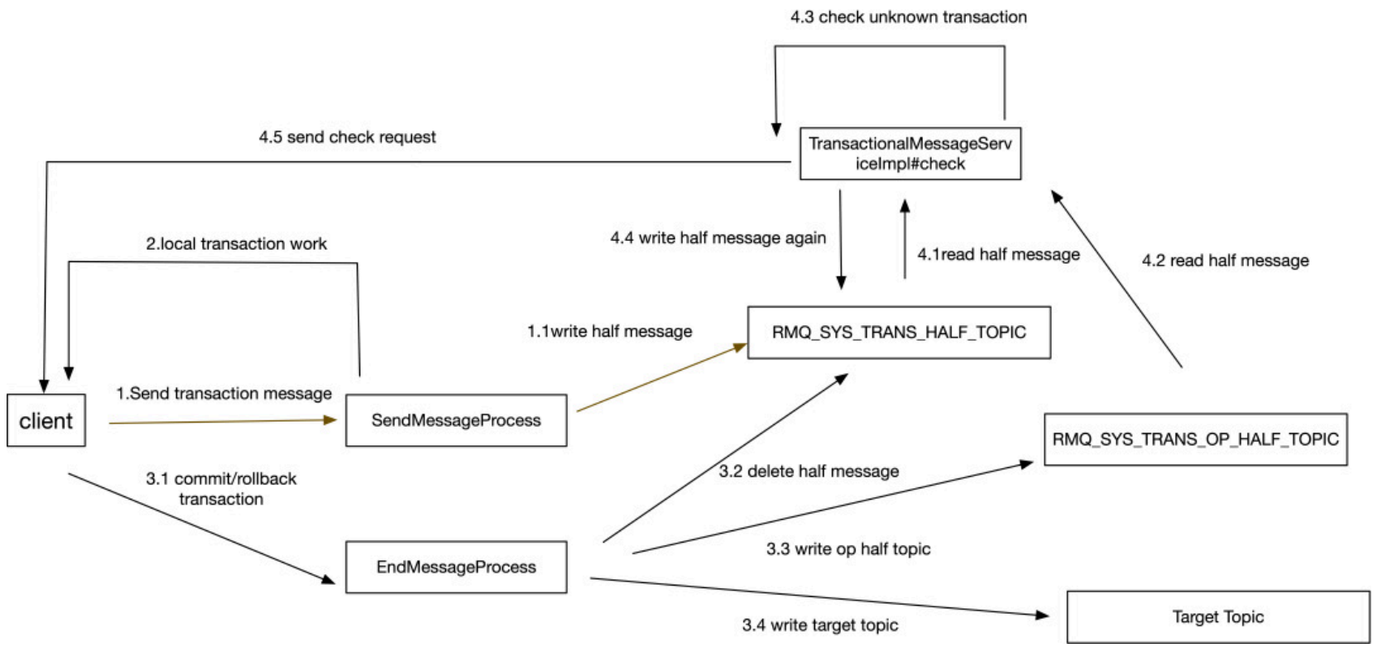


1. 生产者将消息发送至 RocketMQ 服务端；
2. RocketMQ 服务端将消息持久化成功之后，向生产者返回 Ack 确认消息已经发送成功，此时消息被标记为“暂不能投递”，这种状态下的消息即为半事务消息；

本页内容

3. 生产者开始执行本地事务逻辑；
4. 生产者根据本地事务执行结果向服务端提交二次确认结果（Commit 或是 Rollback），服务端收到确认结果后处理逻辑如下：
    - 二次确认结果为 Commit：服务端将半事务消息标记为可投递，并投递给消费者；
    - 二次确认结果为 Rollback：服务端将回滚事务，不会将半事务消息投递给消费者。
5. 在断网或者是生产者应用重启的特殊情况下，若服务端未收到发送者提交的二次确认结果，或服务端收到的二次确认结果为Unknown未知状态，经过固定时间后，服务端将对消息生产者即生产者集群中任一生产者实例发起消息回查；
6. 生产者收到消息回查后，需要检查对应消息的本地事务执行的最终结果；
7. 生产者根据检查到的本地事务的最终状态再次提交二次确认，服务端仍按照步骤 4 对半事务消息进行处理。

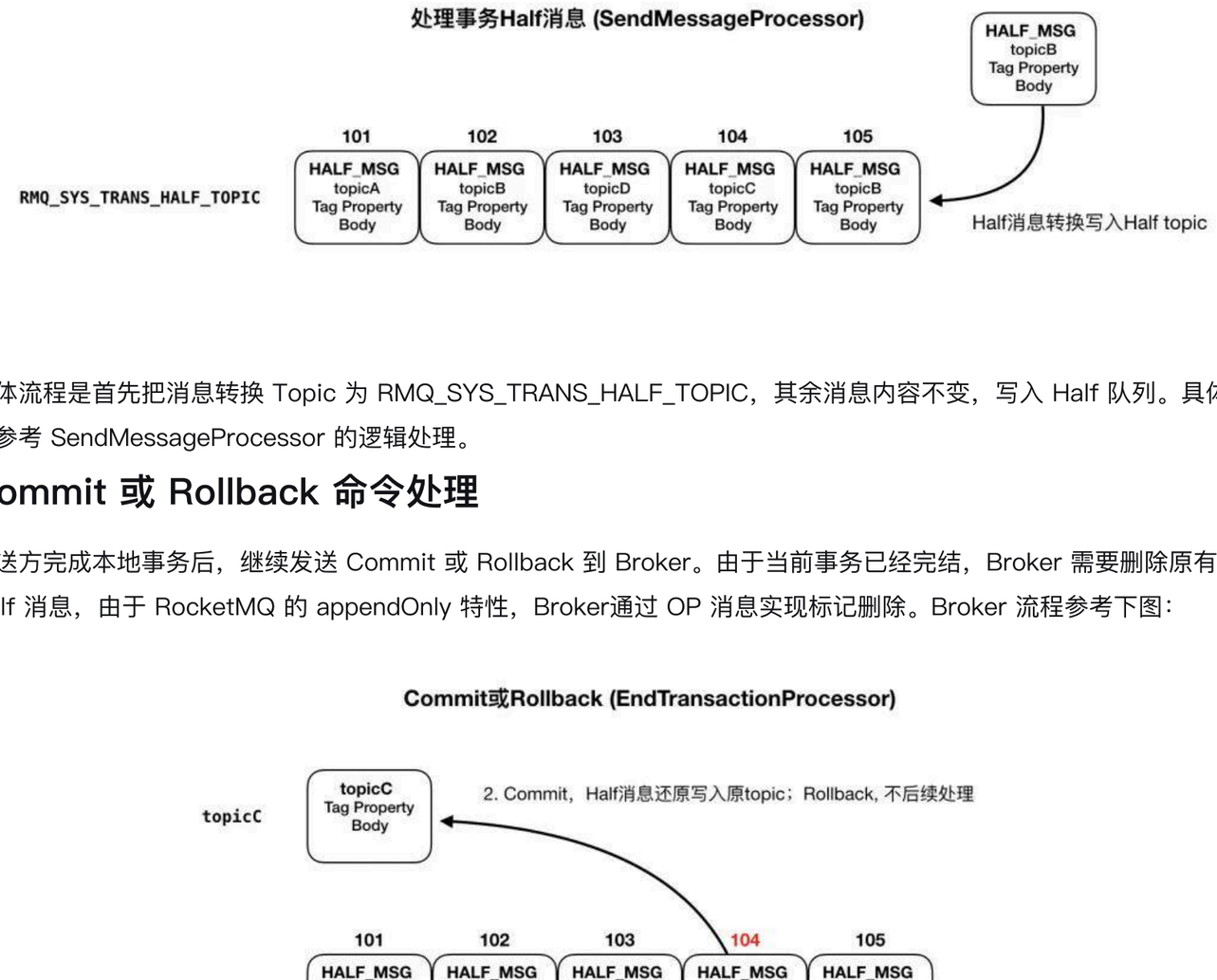
实现细节：RocketMQ 事务消息如何实现



根据发送事务消息的基本流程的需要，实现分为三个主要流程：接收处理 Half 消息、Commit 或 Rollback 命令处理、事务消息 check。

处理 Half 消息

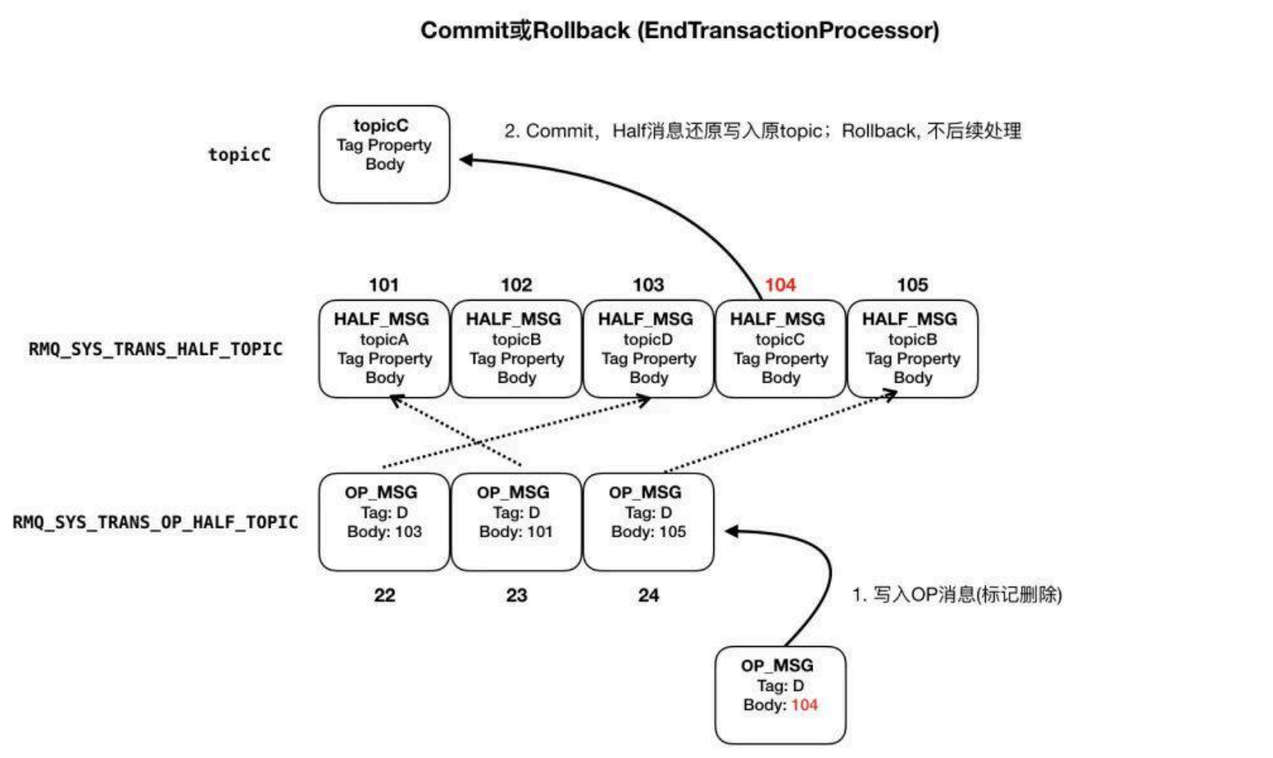
发送方第一阶段发送 Half 消息到 Broker 后，Broker 处理 Half 消息。Broker 流程参考下图：



具体流程是首先把消息转换 Topic 为 RMQ_SYS_TRANS_HALF_TOPIC，其余消息内容不变，写入 Half 队列。具体实现参考 SendMessageProcessor 的逻辑处理。

Commit 或 Rollback 命令处理

发送方完成本地事务后，继续发送 Commit 或 Rollback 到 Broker。由于当前事务已经完结，Broker 需要删除原有的 Half 消息，由于 RocketMQ 的 appendOnly 特性，Broker通过 OP 消息实现标记删除。Broker 流程参考下图：

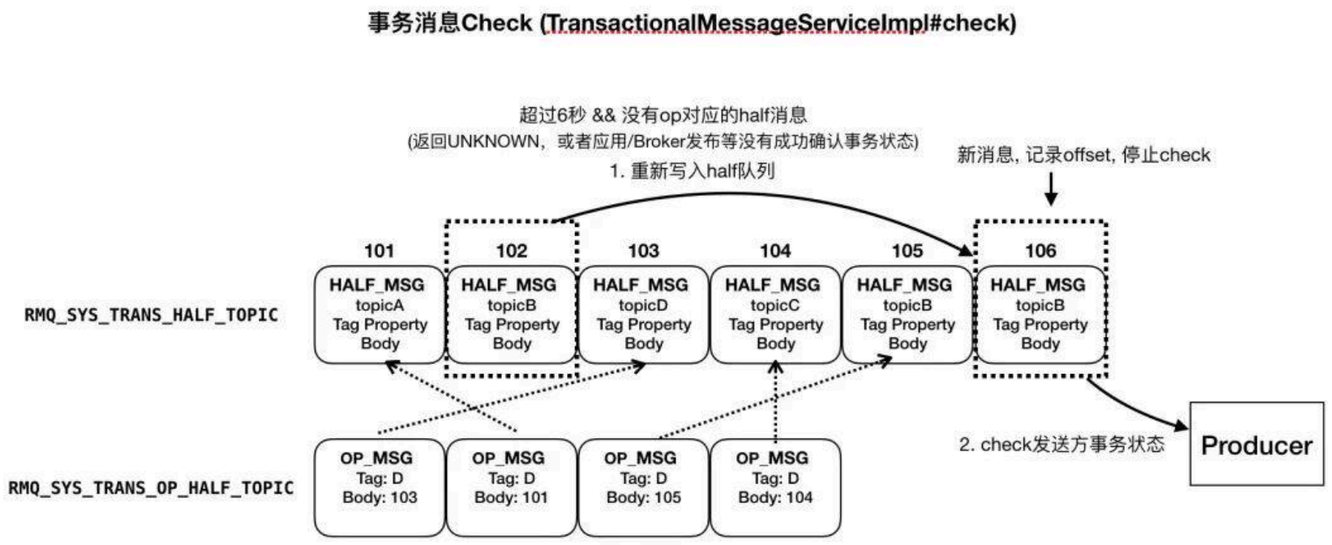


- **Commit**。Broker 写入 OP 消息，OP 消息的 body 指定 Commit 消息的 queueOffset，标记之前 Half 消息已被删除；同时，Broker 读取原 Half 消息，把 Topic 还原，重新写入 CommitLog，消费者则可以拉取消费；
- **Rollback**。Broker 同样写入 OP 消息，流程和 Commit 一样。但后续不会读取和还原 Half 消息。这样消费者就不会消费到该消息。

具体实现在 EndTransactionProcessor 中。

事务消息 check

如果发送端事务时间执行过程，发送 UNKNOWN 命令，或者 Broker/发送端重启发布等原因，流程 2 的标记删除的 OP 消息可能会缺失，因此增加了事务消息 check 流程，该流程是在异步线程定期执行（transactionCheckInterval 默认 30s 间隔），针对这些缺失 OP 消息的 Half 消息进行 check 状态。具体参考下图：



事务消息 check 流程扫描当前的 OP 消息队列，读取已经被标记删除的 Half 消息的 queueOffset。如果发现某个 Half 消息没有 OP 消息对应标记，并且已经超时（transactionTimeOut 默认 6 秒），则读取该 Half 消息重新写入 half 队列，并且发送 check 命令到原发送方检查事务状态；如果没有超时，则会等待后读取 OP 消息队列，获取新的 OP 消息。

另外，为了避免发送方的异常导致长期无法确定事务状态，如果某个 Half 消息的 bornTime 超过最大保留时间（transactionCheckMaxTimeInMs 默认 12 小时），则会自动跳过此消息，不再 check。

具体实现参考：

TransactionalMessageServiceImpl#check 方法。

实战：使用事务消息

了解了 RocketMQ 事务消息的原理后，我们看下如何使用事务。首先，我们需要创建一个“事务消息”类型的 Topic，可以使用控制台或者 CLi 命令创建。

事务消息相比普通消息发送时需要修改以下几点：

- 发送事务消息前，需要开启事务并关联本地的事务执行。
- 为保证事务一致性，在构建生产者时，必须设置事务检查器和预绑定事务消息发送的主题列表，客户端内置的事务检查器会对绑定的事务主题做异常状态恢复。

Java | 复制代码

```
//事务消息发送实例

//1.构造事务检查器
TransactionChecker checker = new TransactionChecker() {
    @Override
    public TransactionResolution check(MessageView messageView) {
        //异常状态的事务检查，返回本地事务结果。
        return TransactionResolution.COMMIT;
    }
};
ProducerBuilder producerBuilder = null;
//2.绑定事务检查器以及需要事务检查的主题列表。
Producer producer1 = producerBuilder.setTransactionChecker(checker)
    .setTopics("TransactionTopic").build();
//3.开启事务。
Transaction transaction = producer1.beginTransaction();
//3.1执行本地事务。
//....
//3.2发送事务消息。
Message message1 = messageBuilder.build();
SendReceipt sendReceipt1 = producer1.send(message1, transaction);
```

本页内容

热门标签

行业实践 技术探索 社区动态 最佳实践 功能特性 云原生 生态集成 事件驱动架构 物联网 可观测 高可用 微服务 流处理

专家答疑

```
//3.3提交本地事务。
//....
//3.4提交消息事务。
transaction.commit();
```

当事务消息 commit 之后，这条消息其实就是一条投递到用户 Topic 的普通消息而已。所以对于消费者来说，和普通消息的消费没有区别。

Java 复制代码

```
public static void main(String[] args) {
    ClientServiceProvider provider = ClientServiceProvider.loadService();
    StaticSessionCredentialsProvider staticSessionCredentialsProvider =
        new StaticSessionCredentialsProvider("AccessKey", "SecretKey");
    ClientConfiguration clientConfiguration =
ClientConfiguration.newBuilder()
        .setEndpoints("Endpoint")
        .setCredentialProvider(staticSessionCredentialsProvider)
        .build();

    try {
        PushConsumer pushConsumer = provider.newPushConsumerBuilder()
            .setClientConfiguration(clientConfiguration)
            .setConsumerGroup("ConsumerGroup")

.setSubscriptionExpressions(Collections.singletonMap("TransactionTopic", new
FilterExpression()))
            .setMessageListener(messageView -> {
                System.out.println("Receive Message " +
messageView.getMessageId());
                return ConsumeResult.SUCCESS;
            })
            .build();
    } catch (ClientException e) {
        // 捕获异常，进行异常处理
    }
}
```

注意：

1. 避免大量未决事务导致超时：在事务提交阶段异常的情况下发起事务回查，保证事务一致性；但生产者应该尽量避免本地事务返回未知结果；大量的事务检查会导致系统性能受损，容易导致事务处理延迟；
2. 事务消息的 Group ID 不能与其他类型消息的 Group ID 共用:与其他类型的消息不同，事务消息有回查机制，回查时服务端会根据 Group ID 去查询生产者客户端；
3. 事务超时机制：半事务消息被生产者发送服务端后，如果在指定时间内服务端无法确认提交或者回滚状态，则消息默认会被回滚。

活动推荐

阿里云基于 Apache RocketMQ 构建的企业级产品–消息队列RocketMQ 5.0版现开启活动：

1、新用户首次购买包年包月，即可享受全系列 85折优惠！ 了解活动详情：

https://www.aliyun.com/product/rocketmq

上一篇
← 云原生事件驱动引擎(RocketMQ-EventBridge)应用场景与技术解析

下一篇
RocketMQ 消息集成：多类型业务消息——定时消息 →

本页内容

热门标签

行业实践 技术探索 社区动态 最佳实践 功能特性 云原生 生态集成 事件驱动架构 物联网 可观测 高可用 微服务 流处理