消息传输和消费的有序性，是消息队列应用中一个非常重要的问题，在分布式系统中，很多业务场景都需要考虑消息投递的时序。例如，电商中的订单状态流转、数据库的 binlog 分发，都会对业务的有序性有要求。今天我们一起来看下，消息队列顺序消费的相关内容。

消息顺序消费有哪些困难

我们知道，消息队列中的队列是一个有序的数据结构，消息传递是顺序的，但在实际开发中，特别是在分布式场景下，消息的有序性是很难保证的，那么为什么实现有序性这么困难呢？下面进行拆解。

分布式的时钟问题

有序性可以分为业务上的有序和时间上的有序，先看一下时钟上的有序。在分布式环境下，消息的生产者、消费者和队列存储，可能分布在不同的机器上，不同的机器使用各自的本地时钟，由于服务器存在时钟偏斜等问题，本地时间会出现不一致，所以不能用消息发送和到达的时间戳作为时序判断标准。另一方面，分布式系统下缺乏全局时钟，这就使得绝对的时间顺序实现起来更加困难。

消息发送端和消费端的集群

在目前大多数消息队列的应用中，生产者和消费者都是集群部署，通过 ProducerGroup 和 ConsumerGroup 的方式来运行。

生产者如果存在多个发送实例，那么各个发送方的时间戳无法同步，所以消息发送端发送时的时序不能用来作为消息发送的有序判断。

同样的，消费端可能存在多个实例，即使队列内部是有序的，由于存在消息的分发过程，不同消费实例的顺序难以全局统一，也无法实现绝对的有序消费。

消息重传等的影响

我们知道，消息队列在传输消息时，可能会出现网络抖动导致的消息发送失败等，对这种场景的兼容，一般是通过进行合理地重传。消息的重传发生在什么时候是不可预知的，这也会导致消息传输出现乱序。

网络及内部并发

消息生产者集群或者消费端集群的方式，无法保证消息的绝对时序，如果只有一个消费端或者只有一个生产端呢？可以考虑这样一个场景，如果单纯地依靠消息队列本身来保证，那么在跨实例的情况下，因为网络传输的不稳定会有先后顺序，以及内部消费的并发等，仍然无法实现绝对有序。

通过上面的分析可以看到，保证消息绝对的有序，实现起来非常困难，除非在服务器内部，并且一个生产者对应一个消费者。但是这种情况的消息队列肯定是无法在实际业务中应用的，那么解决消息队列的有序性有哪些手段呢？下面从消息队列本身，以及业务设计上进行分析。

不同消息队列对顺序消费的保证

消息传输的有序性和不同的消息队列，不同业务场景，以及技术方案的实现细节等都有关系，解决消息传输的有序性，需要依赖消息队列提供对应的方式。

从消息队列自身的角度，可以分为全局有序和局部有序。当前大部分消息队列的应用场景都是集群部署，在全局有序的情况下，无法使用多分区进行性能的优化。在实际开发中，一般是应用局部有序，把业务消息分发到一个固定的分区，也就是单个队列内传输的方式，实现业务上对有序的要求。

以 Kafka 和 RocketMQ 为例，都实现了特定场景下的有序消息。

Kafka 顺序消息

Kafka 保证消息在 Partition 内的顺序，对于需要确保顺序的消息，发送到同一个 Partition 中就可以。单分区的情况下可以天然满足消息有序性，如果是多分区，则可以通过制定的分发策略，将同一类消息分发到同一个 Partition 中。

例如，电商系统中的订单流转信息，我们在写入 Kafka 时通过订单 ID 进行分发，保证同一个订单 ID 的消息都会被发送到同一个 Partition 中，这样消费端在消费的时候，可以保证取出数据时是有序的。

一个比较特殊的情况是消息失败重发的场景，比如同一个订单下的消息 1 和 2，如果 1 发送失败了，重发的时候可能会出现在 2 的后边，这种情况可以通过设置“max.in.flight.requests.per.connection”参数来解决，该参数可以限制客户端能够发送的未响应请求的个数，还可以在一定程度上避免这种消息乱序。

RocketMQ 顺序消息

RocketMQ 对有序消息的保证和 Kafka 类似，RocketMQ 保证消息在同一个 Queue 中的顺序性，也就是可以满足队列的先进先出原则。

如果把对应一个业务主键的消息都路由到同一个 Queue 中就可以实现消息的有序传输，并且 RocketMQ 额外支持 Tag 的方式，可以对业务消息做进一步的拆分，在消费时相对更加灵活。

从业务角度保证顺序消费

在我之前的项目中，消息消费的有序性，归根到底是一个业务场景的设计问题，可以在业务中进行规避，或者通过合理的设计方案来解决。

消息传输的有序性是否有必要

山不过来，我就过去，解决一个问题，如果从正面没有很好的解决方案，那么我们就可以考虑是否绕过它。考虑在你的业务中，是否必须实现绝对的消息有序，或者是否必须要有消息队列这样的技术手段。

比如在一个订单状态消息流转的业务场景中，订单会有创建成功、待付款、已支付、已发货的状态，这几个状态之间是单调流动的，也就是说，订单状态的更新需要保证有序性。考虑一下，如果我们要实现的功能是根据发货的状态，进行物流通知用户的功能，实际上因为这个状态是单调不可逆向的，我们可以忽略订单状态的顺序，只关注最后是否已发货的状态。

也就是说，在这个场景下，订单状态流转虽然是要考虑顺序，但是在具体的这个功能下，实际上不需要关注订单状态消息消费的时序。

业务中如何实现有序消费

除了消息队列自身的顺序消费机制，我们可以合理地对消息进行改造，从业务上实现有序的目的。具体的方式有以下几种。

- 根据不同的业务场景，以发送端或者消费端时间戳为准

比如在电商大促的秒杀场景中，如果要对秒杀的请求进行排队，就可以使用秒杀提交时服务端的时间戳，虽然服务端不一定保证时钟一致，但是在这个场景下，我们不需要保证绝对的有序。

- 每次消息发送时生成唯一递增的 ID

在每次写入消息时，可以考虑添加一个单调递增的序列 ID，在消费端进行消费时，缓存最大的序列 ID，只消费超过当前最大的序列 ID 的消息。这个方案和分布式算法中的 Paxos 很像，虽然无法实现绝对的有序，但是可以保证每次只处理最新的数据，避免一些业务上的不一致问题。

- 通过缓存时间戳的方式

这种方式的机制和递增 ID 是一致的，即当生产者在发送消息时，添加一个时间戳，消费端在处理消息时，通过缓存时间戳的方式，判断消息产生的时间是否最新，如果不是则丢弃，否则执行下一步。

总结

这一课时讨论了消息队列有序性的话题，消息的有序性可以分为时间上的有序和业务上的有序。

通过上面的分析可以看到，绝对的时间有序实现起来是非常困难的，即使实现了这样的消息队列，但在实际应用中的意义并不大。消息队列只是一个消息传输的解决方案，不是软件开发中的银弹，一般来说，我们可以通过业务中不同的场景，进行合理的设计，实现业务上的有序性。

现在你可以思考一下，在你的项目中，哪些场景要求消息传输和消费的有序性，具体是如何解决的？欢迎留言进行分享。

精选评论

haiki：

> 讲得太好了！和业务场景紧密结合，清晰易懂，希望赶紧更新接下来的内容😁

****雄：**

> 生成惟一自增id和缓存时间戳消费最新的数据，这样子不是会漏消息吗

讲师回复：

> 在消费端存储序列号，保证有序，可能会丢弃部分超时的消息