二

14 幂等生产者和事务生产者是一回事吗?

你好，我是胡夕。今天我要和你分享的主题是：Kafka 消息交付可靠性保障以及精确处理一次语义的实现。

所谓的消息交付可靠性保障，是指 Kafka 对 Producer 和 Consumer 要处理的消息提供什么样的承诺。常见的承诺有以下三种：

- 最多一次（at most once）：消息可能会丢失，但绝不会被重复发送。
- 至少一次（at least once）：消息不会丢失，但有可能被重复发送。
- 精确一次（exactly once）：消息不会丢失，也不会被重复发送。

目前，Kafka 默认提供的交付可靠性保障是第二种，即至少一次。在专栏[第 11 期]中，我们说过消息“已提交”的含义，即只有 Broker 成功“提交”消息且 Producer 接到 Broker 的应答才会认为该消息成功发送。不过倘若消息成功“提交”，但 Broker 的应答没有成功发送回 Producer 端（比如网络出现瞬时抖动），那么 Producer 就无法确定消息是否真的提交成功了。因此，它只能选择重试，也就是再次发送相同的消息。这就是 Kafka 默认提供至少一次可靠性保障的原因，不过这会导致消息重复发送。

Kafka 也可以提供最多一次交付保障，只需要让 Producer 禁止重试即可。这样一来，消息要么写入成功，要么写入失败，但绝不会重复发送。我们通常不会希望出现消息丢失的情况，但一些场景里偶发的消息丢失其实是被允许的，相反，消息重复是绝对要避免的。此时，使用最多一次交付保障就是最恰当的。

无论是至少一次还是最多一次，都不如精确一次来得有吸引力。大部分用户还是希望消息只会被交付一次，这样的话，消息既不会丢失，也不会被重复处理。或者说，即使 Producer 端重复发送了相同的消息，Broker 端也能做到自动去重。在下游 Consumer 看来，消息依然只有一条。

那么问题来了，Kafka 是怎么做到精确一次的呢？简单来说，这是通过两种机制：幂等性（Idempotence）和事务（Transaction）。它们分别是什么机制？两者是一回事吗？要回答这些问题，我们首先来说说什么是幂等性。

什么是幂等性（Idempotence）？

“幂等”这个词原是数学领域中的概念，指的是某些操作或函数能够被执行多次，但每次得到的结果都是不变的。我来举几个简单的例子说明一下。比如在乘法运算中，让数字乘以 1 就是一个幂等操作，因为不管你执行多少次这样的运算，结果都是相同的。再比如，取整函数（floor 和 ceiling）是幂等函数，那么运行 1 次 floor(3.4) 和 100 次 floor(3.4)，结果是一样的，都是 3。相反地，让一个数加 1 这个操作就不是幂等的，因为执行一次和执行多次的结果必然不同。

在计算机领域中，幂等性的含义稍微有一些不同：

- 在命令式编程语言（比如 C）中，若一个子程序是幂等的，那它必然不能修改系统状态。这样不管运行这个子程序多少次，与该子程序关联的那部分系统状态保持不变。
- 在函数式编程语言（比如 Scala 或 Haskell）中，很多纯函数（pure function）天然就是幂等的，它们不执行任何的 side effect。

幂等性有很多好处，**其最大的优势在于我们可以安全地重试任何幂等性操作，反正它们也不会破坏我们的系统状态。**如果是非幂等性操作，我们还需要担心某些操作执行多次对状态的影响，但对于幂等性操作而言，我们根本无需担心此事。

幂等性 Producer

在 Kafka 中，Producer 默认不是幂等性的，但我们可以创建幂等性 Producer。它其实是 0.11.0.0 版本引入的新功能。在此之前，Kafka 向分区发送数据时，可能会出现同一条消息被发送了多次，导致消息重复的情况。在 0.11 之后，指定 Producer 幂等性的方法很简单，仅需要设置一个参数即可，即 props.put(“enable.idempotence”, ture)，或 props.put(ProducerConfig.ENABLE_IDEMPOTENCE_CONFIG， true)。

enable.idempotence 被设置成 true 后，Producer 自动升级成幂等性 Producer，其他所有的代码逻辑都不需要改变。Kafka 自动帮你做消息的重复去重。底层具体的原理很简单，就是经典的用空间去换时间的优化思路，即在 Broker 端多保存一些字段。当 Producer 发送了具有相同字段值的消息后，Broker 能够自动知晓这些消息已经重复了，于是可以在后台默默地把它们“丢弃”掉。当然，实际的实现原理并没有这么简单，但你大致可以这么理解。

看上去，幂等性 Producer 的功能很酷，使用起来也很简单，仅仅设置一个参数就能保证消息不重复了，但实际上，我们必须要了解幂等性 Producer 的作用范围。

首先，它只能保证单分区上的幂等性，即一个幂等性 Producer 能够保证某个主题的一个分区上不出现重复消息，它无法实现多个分区的幂等性。其次，它只能实现单会话上的幂等

性，不能实现跨会话的幂等性。这里的会话，你可以理解为 Producer 进程的一次运行。当你重启了 Producer 进程之后，这种幂等性保证就丧失了。

那么你可能会问，如果我想实现多分区以及多会话上的消息无重复，应该怎么做呢？答案就是事务（transaction）或者依赖事务型 Producer。这也是幂等性 Producer 和事务型 Producer 的最大区别！

事务

Kafka 的事务概念类似于我们熟知的数据库提供的事务。在数据库领域，事务提供的安全性保障是经典的 ACID，即原子性（Atomicity）、一致性 (Consistency)、隔离性 (Isolation) 和持久性 (Durability)。

当然，在实际场景中各家数据库对 ACID 的实现各不相同。特别是 ACID 本身就是一个有歧义的概念，比如对隔离性的理解。大体来看，隔离性非常自然和必要，但是具体到实现细节就显得不那么精确了。通常来说，**隔离性表明并发执行的事务彼此相互隔离，互不影响**。经典的数据库教科书把隔离性称为可串行化 (serializability)，即每个事务都假装它是整个数据库中唯一的事务。

提到隔离级别，这种歧义或混乱就更加明显了。很多数据库厂商对于隔离级别的实现都有自己不同的理解，比如有的数据库提供 Snapshot 隔离级别，而在另外一些数据库中，它们被称为可重复读（repeatable read）。好在对于已提交读（read committed）隔离级别的提法，各大主流数据库厂商都比较统一。所谓的 read committed，指的是当读取数据库时，你只能看到已提交的数据，即无脏读。同时，当写入数据库时，你也只能覆盖掉已提交的数据，即无脏写。

Kafka 自 0.11 版本开始也提供了对事务的支持，目前主要是在 read committed 隔离级别上做事情。它能保证多条消息原子性地写入到目标分区，同时也能保证 Consumer 只能看到事务成功提交的消息。下面我们就来看看 Kafka 中的事务型 Producer。

事务型 Producer

事务型 Producer 能够保证将消息原子性地写入到多个分区中。这批消息要么全部写入成功，要么全部失败。另外，事务型 Producer 也不惧进程的重启。Producer 重启回来后，Kafka 依然保证它们发送消息的精确一次处理。

设置事务型 Producer 的方法也很简单，满足两个要求即可：

- 和幂等性 Producer 一样，开启 enable.idempotence = true。

- 设置 Producer 端参数 transctional. id。最好为其设置一个有意义的名字。

此外，你还需要在 Producer 代码中做一些调整，如这段代码所示：

```
producer.initTransactions();
try {
            producer.beginTransaction();
            producer.send(record1);
            producer.send(record2);
            producer.commitTransaction();
} catch (KafkaException e) {
            producer.abortTransaction();
}
```

和普通 Producer 代码相比，事务型 Producer 的显著特点是调用了一些事务 API，如 initTransaction、beginTransaction、commitTransaction 和 abortTransaction，它们分别对应事务的初始化、事务开始、事务提交以及事务终止。

这段代码能够保证 Record1 和 Record2 被当作一个事务统一提交到 Kafka，要么它们全部提交成功，要么全部写入失败。实际上即使写入失败，Kafka 也会把它们写入到底层的日志中，也就是说 Consumer 还是会看到这些消息。因此在 Consumer 端，读取事务型 Producer 发送的消息也是需要一些变更的。修改起来也很简单，设置 isolation.level 参数的值即可。当前这个参数有两个取值：

1. read_uncommitted：这是默认值，表明 Consumer 能够读取到 Kafka 写入的任何消息，不论事务型 Producer 提交事务还是终止事务，其写入的消息都可以读取。很显然，如果你用了事务型 Producer，那么对应的 Consumer 就不要使用这个值。
2. read_committed：表明 Consumer 只会读取事务型 Producer 成功提交事务写入的消息。当然了，它也能看到非事务型 Producer 写入的所有消息。

小结

简单来说，幂等性 Producer 和事务型 Producer 都是 Kafka 社区力图为 Kafka 实现精确一次处理语义所提供的工具，只是它们的作用范围是不同的。幂等性 Producer 只能保证单分区、单会话上的消息幂等性；而事务能够保证跨分区、跨会话间的幂等性。从交付语义上来看，自然是事务型 Producer 能做的更多。

不过，切记天下没有免费的午餐。比起幂等性 Producer，事务型 Producer 的性能要更差，在实际使用过程中，我们需要仔细评估引入事务的开销，切不可无脑地启用事务。

上一页 下一页