

## 06-如何处理消费过程中的重复消息？

你好，我是李玥。上节课我们讲了如何确保消息不会丢失，课后我给你留了一个思考题，如果消息重复了怎么办？这节课，我们就来聊一聊如何处理重复消息的问题。

在消息传递过程中，如果出现传递失败的情况，发送方会执行重试，重试的过程中就有可能产生重复的消息。对使用消息队列的业务系统来说，如果没有对重复消息进行处理，就有可能导致系统的数据出现错误。

比如说，一个消费订单消息，统计下单金额的微服务，如果没有正确处理重复消息，那就会出现重复统计，导致统计结果错误。

你可能会问，如果消息队列本身能保证消息不重复，那应用程序的实现不就简单了？那有没有消息队列能保证消息不重复呢？

### 消息重复的情况必然存在

在MQTT协议中，给出了三种传递消息时能够提供的服务质量标准，这三种服务质量从低到高依次是：

- **At most once:** 至多一次。消息在传递时，最多会被送达一次。换一个说法就是，没什么消息可靠性保证，允许丢消息。一般都是一些对消息可靠性要求不太高的监控场景使用，比如每分钟上报一次机房温度数据，可以接受数据少量丢失。
- **At least once:** 至少一次。消息在传递时，至少会被送达一次。也就是说，不允许丢消息，但是允许有少量重复消息出现。
- **Exactly once:** 恰好一次。消息在传递时，只会被送达一次，不允许丢失也不允许重复，这个是最高的等级。

这个服务质量标准不仅适用于MQTT，对所有的消息队列都是适用的。我们现在常用的绝大部分消息队列提供的服务质量都是At least once，包括RocketMQ、RabbitMQ和Kafka 都是这样。也就是说，消息队列很难保证消息不重复。

说到这儿我知道肯定有的同学会反驳我：“你说的不对，我看过Kafka的文档，Kafka是支持Exactly once的。”我在这儿跟这些同学解释一下，你说的没错，Kafka的确是支持Exactly once，但是我讲的也没有问题，为什么呢？

Kafka支持的“Exactly once”和我们刚刚提到的消息传递的服务质量标准“Exactly once”是不一样的，它是Kafka提供的另外一个特性，Kafka中支持的事务也和我们通常意义理解的事务有一定的差异。在Kafka中，事务和Exactly once主要是为了配合流计算使用的特性，我们在专栏“进阶篇”这个模块中，会有专门的一节课来讲Kafka的事务和它支持的Exactly once特性。

稍微说一些题外话，Kafka的团队是一个非常善于包装和营销的团队，你看他们很巧妙地用了两个所有人都非常熟悉的概念“事务”和“Exactly once”来包装它的新的特性，实际上它实现的这个事务和Exactly once并不是我们通常理解的那两个特性，但是你深入了解Kafka的事务和Exactly once后，会发现其实它这个特性虽然和我们通常的理解不一样，但确实和事务、Exactly once有一定关系。

这一点上，我们都要学习Kafka团队。一个优秀的开发团队，不仅要能写代码，更要能写文档，能写

Slide (PPT)，还要能讲，会分享。对于每个程序员来说，也是一样的。

我们把话题收回来，继续来说重复消息的问题。既然消息队列无法保证消息不重复，就需要我们的消费代码能够接受“消息是可能会重复的”这一现状，然后，通过一些方法来消除重复消息对业务的影响。

## 用幂等性解决重复消息问题

一般解决重复消息的办法是，在消费端，让我们消费消息的操作具备幂等性。

**幂等 (Idempotence)** 本来是一个数学上的概念，它是这样定义的：

如果一个函数 $f(x)$ 满足： $f(f(x)) = f(x)$ ，则函数 $f(x)$ 满足幂等性。

这个概念被拓展到计算机领域，被用来描述一个操作、方法或者服务。一个幂等操作的特点是，**其任意多次执行所产生的影响均与一次执行的影响相同。**

一个幂等的方法，使用同样的参数，对它进行多次调用和一次调用，对系统产生的影响是一样的。所以，对于幂等的方法，不用担心重复执行会对系统造成任何改变。

我们举个例子来说明一下。在不考虑并发的情况下，“将账户X的余额设置为100元”，执行一次后对系统的影响是，账户X的余额变成了100元。只要提供的参数100元不变，那即使再执行多少次，账户X的余额始终都是100元，不会变化，这个操作就是一个幂等的操作。

再举一个例子，“将账户X的余额加100元”，这个操作它就不是幂等的，每执行一次，账户余额就会增加100元，执行多次和执行一次对系统的影响（也就是账户的余额）是不一样的。

如果我们系统消费消息的业务逻辑具备幂等性，那就不用担心消息重复的问题了，因为同一条消息，消费一次和消费多次对系统的影响是完全一样的。也就可以认为，消费多次等于消费一次。

从对系统的影响结果来说：**At least once + 幂等消费 = Exactly once。**

那么如何实现幂等操作呢？最好的方式就是，**从业务逻辑设计上入手，将消费的业务逻辑设计成具备幂等性的操作。**但是，不是所有的业务都能设计成天然幂等的，这里就需要一些方法和技巧来实现幂等。

下面我给你介绍几种常用的设计幂等操作的方法：

### 1. 利用数据库的唯一约束实现幂等

例如我们刚刚提到的那个不具备幂等特性的转账的例子：将账户X的余额加100元。在这个例子中，我们可以通过改造业务逻辑，让它具备幂等性。

首先，我们可以限定，对于每个转账单每个账户只可以执行一次变更操作，在分布式系统中，这个限制实现的方法非常多，最简单的是我们在数据库中建一张转账流水表，这个表有三个字段：转账单ID、账户ID和变更金额，然后给转账单ID和账户ID这两个字段联合起来创建一个唯一约束，这样对于相同的转账单ID和账户ID，表里至多只能存在一条记录。

这样，我们消费消息的逻辑可以变为：“在转账流水表中增加一条转账记录，然后再根据转账记录，异步操作更新用户余额即可。”在转账流水表增加一条转账记录这个操作中，由于我们在这个表中预先定义了“账户ID转账单ID”的唯一约束，对于同一个转账单同一个账户只能插入一条记录，后续重复的插入操作都会失败，这样就实现了一个幂等的操作。我们只要写一个SQL，正确地实现它就可以了。

基于这个思路，不光是可以使用关系型数据库，只要是支持类似“INSERT IF NOT EXIST”语义的存储类系统都可以用于实现幂等，比如，你可以用Redis的SETNX命令来替代数据库中的唯一约束，来实现幂等消费。

## 2. 为更新的数据设置前置条件

另外一种实现幂等的思路是，给数据变更设置一个前置条件，如果满足条件就更新数据，否则拒绝更新数据，在更新数据的时候，同时变更前置条件中需要判断的数据。这样，重复执行这个操作时，由于第一次更新数据的时候已经变更了前置条件中需要判断的数据，不满足前置条件，则不会重复执行更新数据操作。

比如，刚刚我们说过，“将账户X的余额增加100元”这个操作并不满足幂等性，我们可以把这个操作加上一个前置条件，变为：“如果账户X当前的余额为500元，将余额加100元”，这个操作就具备了幂等性。对应到消息队列中的使用时，可以在发消息时在消息体中带上当前的余额，在消费的时候进行判断数据库中，当前余额是否与消息中的余额相等，只有相等才执行变更操作。

但是，如果我们要更新的数据不是数值，或者我们要做一个比较复杂的更新操作怎么办？用什么作为前置判断条件呢？更加通用的方法是，给你的数据增加一个版本号属性，每次更新数据前，比较当前数据的版本号是否和消息中的版本号一致，如果不一致就拒绝更新数据，更新数据的同时将版本号+1，一样可以实现幂等更新。

## 3. 记录并检查操作

如果上面提到的两种实现幂等方法都不能适用于你的场景，我们还有一种通用性最强，适用范围最广的实现幂等性方法：记录并检查操作，也称为“Token机制或者GUID（全局唯一ID）机制”，实现的思路特别简单：在执行数据更新操作之前，先检查一下是否执行过这个更新操作。

具体的实现方法是，在发送消息时，给每条消息指定一个全局唯一的ID，消费时，先根据这个ID检查这条消息是否有被消费过，如果没有消费过，才更新数据，然后将消费状态置为已消费。

原理和实现是不是很简单？其实一点儿都不简单，在分布式系统中，这个方法其实是非常难实现的。首先，给每个消息指定一个全局唯一的ID就是一件不那么简单的事儿，方法有很多，但都不太好同时满足简单、高可用和高性能，或多或少都要有些牺牲。更加麻烦的是，在“检查消费状态，然后更新数据并且设置消费状态”中，三个操作必须作为一组操作保证原子性，才能真正实现幂等，否则就会出现Bug。

比如说，对于同一条消息：“全局ID为8，操作为：给ID为666账户增加100元”，有可能出现这样的情况：

- t0时刻：Consumer A 收到条消息，检查消息执行状态，发现消息未处理过，开始执行“账户增加100元”；
- t1时刻：Consumer B 收到条消息，检查消息执行状态，发现消息未处理过，因为这个时刻，Consumer A还未来得及更新消息执行状态。

这样就会导致账户被错误地增加了两次100元，这是一个在分布式系统中非常容易犯的错误，一定要引以为戒。

对于这个问题，当然我们可以用事务来实现，也可以用锁来实现，但是在分布式系统中，无论是分布式事务还是分布式锁都是比较难解决问题。

## 小结

这节课我们主要介绍了通过幂等消费来解决消息重复的问题，然后我重点讲了几种实现幂等操作的方法，你可以利用数据库的约束来防止重复更新数据，也可以为数据更新设置一次性的前置条件，来防止重复消息，如果这两种方法都不适用于你的场景，还可以用“记录并检查操作”的方式来保证幂等，这种方法适用范围最广，但是实现难度和复杂度也比较高，一般不推荐使用。

这些实现幂等的方法，不仅可以用于解决重复消息的问题，也同样适用于，在其他场景中来解决重复请求或者重复调用的问题。比如，我们可以将HTTP服务设计成幂等的，解决前端或者APP重复提交表单数据的问题；也可以将一个微服务设计成幂等的，解决RPC框架自动重试导致的重复调用问题。这些方法都是通用的，希望你能做到触类旁通，举一反三。

## 思考题

最后请你想一下，为什么大部分消息队列都选择只提供At least once的服务质量，而不是级别更高的Exactly once呢？欢迎在留言区与我分享讨论。

感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给你的朋友。



# 消息队列高手课

从源码角度全面解析 MQ 的设计与实现

李玥

京东零售技术架构部资深架构师



新版升级：点击「👤请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。