11 | 无消息丢失配置怎么实现？

2019-06-27 胡夕

你好，我是胡夕。今天我要和你分享的主题是：如何配置Kafka无消息丢失。

一直以来，很多人对于Kafka丢失消息这件事情都有着自己的理解，因而也就有着自己的解决之道。在讨论具体的应对方法之前，我觉得我们首先要明确，在Kafka的世界里什么才算是消息丢失，或者说Kafka在什么情况下能保证消息不丢失。这点非常关键，因为很多时候我们容易混淆责任的边界，如果搞不清楚事情由谁负责，自然也就不知道由谁来出解决方案了。

那Kafka到底在什么情况下才能保证消息不丢失呢？

一句话概括，Kafka只对“已提交”的消息（committed message）做有限度的持久化保证。

这句话里面有两个核心要素，我们一一来看。

第一个核心要素是**“已提交的消息”**。什么是已提交的消息？当Kafka的若干个Broker成功地接收到一条消息并写入到日志文件后，它们会告诉生产者程序这条消息已成功提交。此时，这条消息在Kafka看来就正式变为“已提交”消息了。

那为什么是若干个Broker呢？这取决于你对“已提交”的定义。你可以选择只要有一个Broker成功保存该消息就算是已提交，也可以是令所有Broker都成功保存该消息才算是已提交。不论哪种情况，Kafka只对已提交的消息做持久化保证这件事情是不变的。

第二个核心要素就是“**有限度的持久化保证**”，也就是说Kafka不可能保证在任何情况下都做到不丢失消息。举个极端点的例子，如果地球都不存在了，Kafka还能保存任何消息吗？显然不能！倘若这种情况下你依然还想要Kafka不丢消息，那么只能在别的星球部署Kafka Broker服务器了。

现在你应该能够稍微体会出这里的“有限度”的含义了吧，其实就是说Kafka不丢消息是有前提条件的。假如你的消息保存在N个Kafka Broker上，那么这个前提条件就是这N个Broker中至少有1个存活。只要这个条件成立，Kafka就能保证你的这条消息永远不会丢失。

总结一下，Kafka是能做到不丢失消息的，只不过这些消息必须是已提交的消息，而且还要满足一定的条件。当然，说明这件事并不是要为Kafka推卸责任，而是为了在出现该类问题时我们能够明确责任边界。

“消息丢失”案例

好了，理解了Kafka是怎样做到不丢失消息的，那接下来我带你复盘一下那些常见的“Kafka消息丢失”案例。注意，这里可是带引号的消息丢失哦，其实有些时候我们只是冤枉了Kafka而已。

案例1：生产者程序丢失数据

Producer程序丢失消息，这应该算是被抱怨最多的数据丢失场景了。我来描述一个场景：你写了一个Producer应用向Kafka发送消息，最后发现Kafka没有保存，于是大骂：“Kafka真烂，消息发送居然都能丢失，而且还不告诉我？！”如果你有过这样的经历，那么请先消消气，我们来分析下可能的原因。

目前Kafka Producer是异步发送消息的，也就是说如果你调用的是producer.send(msg)这个API，那么它通常会立即返回，但此时你不能认为消息发送已成功完成。

这种发送方式有个有趣的名字，叫“fire and forget”，翻译一下就是“发射后不管”。这个术语原本属于导弹制导领域，后来被借鉴到计算机领域中，它的意思是，执行完一个操作后不去管它的结果是否成功。调用producer.send(msg)就属于典型的“fire and forget”，因此如果出现消息丢失，我们是无法知晓的。这个发送方式挺不靠谱吧，不过有些公司真的就是在使用这个API发送消息。

如果用这个方式，可能会有哪些因素导致消息没有发送成功呢？其实原因有很多，例如网络抖动，导致消息压根就没有发送到Broker端；或者消息本身不合格导致Broker拒绝接收（比如消息太大了，超过了Broker的承受能力）等。这么来看，让Kafka“背锅”就有点冤枉它了。就像前面说过的，Kafka不认为消息是已提交的，因此也就没有Kafka丢失消息这一说了。

不过，就算不是Kafka的“锅”，我们也要解决这个问题吧。实际上，解决此问题的方法非常简单：**Producer永远要使用带有回调通知的发送API，也就是说不要使用producer.send(msg)，而要使用producer.send(msg, callback)**。不要小瞧这里的

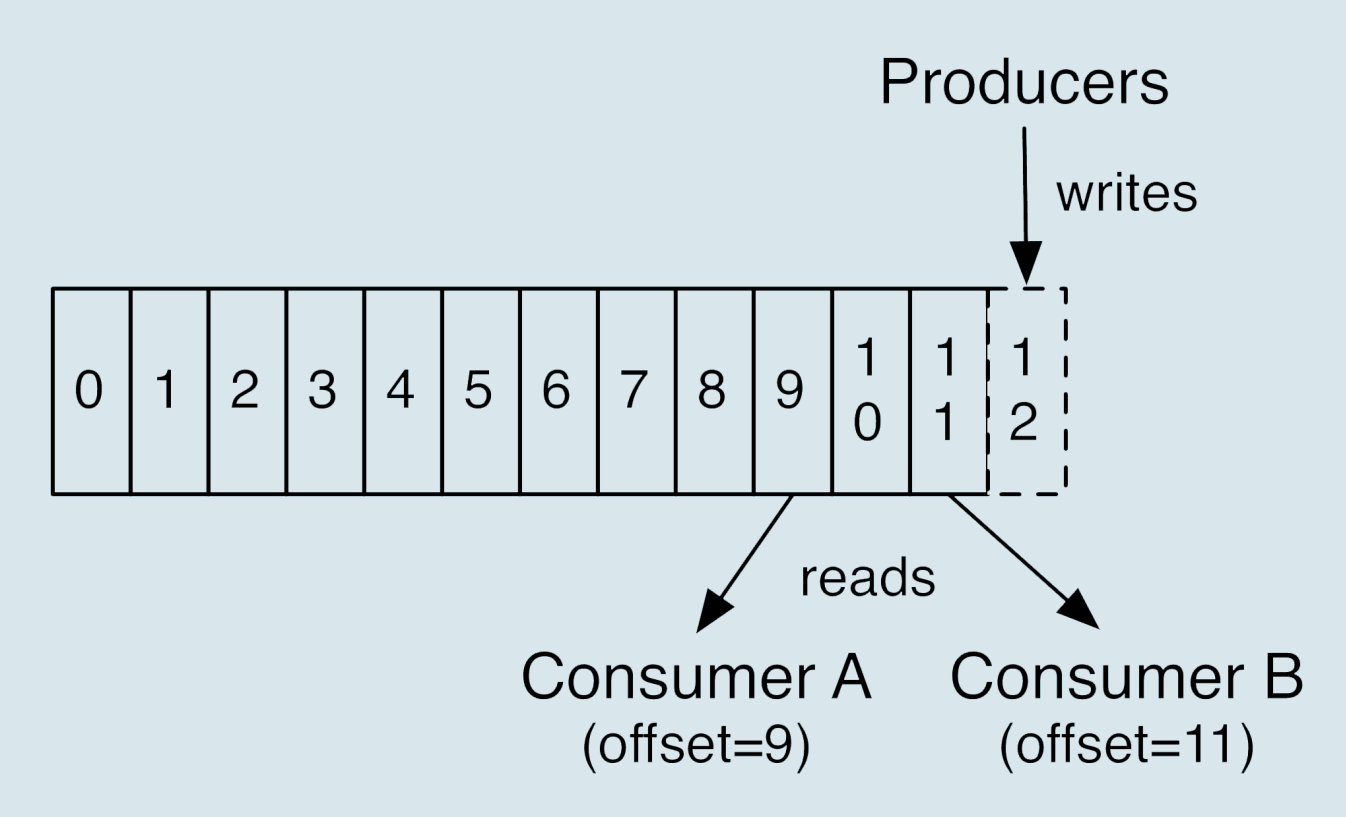
callback（回调），它能准确地告诉你消息是否真的提交成功了。一旦出现消息提交失败的情况，你就可以有针对性地进行处理。

举例来说，如果是因为那些瞬时错误，那么仅仅让Producer重试就可以了；如果是消息不合格造成的，那么可以调整消息格式后再次发送。总之，处理发送失败的责任在Producer端而非Broker端。

你可能会问，发送失败真的没可能是由Broker端的问题造成的吗？当然可能！如果你所有的Broker都宕机了，那么无论Producer端怎么重试都会失败的，此时你要做的是赶快处理Broker端的问题。但之前说的核心论据在这里依然是成立的：Kafka依然不认为这条消息属于已提交消息，故对它不做任何持久化保证。

案例2：消费者程序丢失数据

Consumer端丢失数据主要体现在Consumer端要消费的消息不见了。Consumer程序有个“位移”的概念，表示的是这个Consumer当前消费到的Topic分区的位置。下面这张图来自于官网，它清晰地展示了Consumer端的位移数据。



比如对于Consumer A而言，它当前的位移值就是9；Consumer B的位移值是11。

这里的“位移”类似于我们看书时使用的书签，它会标记我们当前阅读了多少页，下次翻书的时候我们能直接跳到书签页继续阅读。

正确使用书签有两个步骤：第一步是读书，第二步是更新书签页。如果这两步的顺序颠倒了，就可能出现这样的场景：当前的书签页是第90页，我先将书签放到第100页上，之后开始读书。当

阅读到第95页时，我临时有事中止了阅读。那么问题来了，当我下次直接跳到书签页阅读时，我就丢失了第96～99页的内容，即这些消息就丢失了。

同理，Kafka中Consumer端的消息丢失就是这么一回事。要对抗这种消息丢失，办法很简单：**维持先消费消息（阅读），再更新位移（书签）的顺序**即可。这样就能最大限度地保证消息不丢失。

当然，这种处理方式可能带来的问题是消息的重复处理，类似于同一页书被读了很多遍，但这不属于消息丢失的情形。在专栏后面的内容中，我会跟你分享如何应对重复消费的问题。

除了上面所说的场景，其实还存在一种比较隐蔽的消息丢失场景。

我们依然以看书为例。假设你花钱从网上租借了一本共有10章内容的电子书，该电子书的有效阅读时间是1天，过期后该电子书就无法打开，但如果在1天之内你完成阅读就退还租金。

为了加快阅读速度，你把书中的10个章节分别委托给你的10个朋友，请他们帮你阅读，并拜托他们告诉你主旨大意。当电子书临近过期时，这10个人告诉你说他们读完了自己所负责的那个章节的内容，于是你放心地把该书还了回去。不料，在这10个人向你描述主旨大意时，你突然发现有一个人对你撒了谎，他并没有看完他负责的那个章节。那么很显然，你无法知道那一章的内容了。

对于Kafka而言，这就好比Consumer程序从Kafka获取到消息后开启了多个线程异步处理消息，而Consumer程序自动地向前更新位移。假如其中某个线程运行失败了，它负责的消息没有被成功处理，但位移已经被更新了，因此这条消息对于Consumer而言实际上是丢失了。

这里的关键在于Consumer自动提交位移，与你没有确认书籍内容被全部读完就将书归还类似，你没有真正地确认消息是否真的被消费就“盲目”地更新了位移。

这个问题的解决方案也很简单：**如果是多线程异步处理消费消息，Consumer程序不要开启自动提交位移，而是要应用程序手动提交位移**。在这里我要提醒你一下，单个Consumer程序使用多线程来消费消息说起来容易，写成代码却异常困难，因为你很难正确地处理位移的更新，也就是说避免无消费消息丢失很简单，但极易出现消息被消费了多次的情况。

最佳实践

看完这两个案例之后，我来分享一下Kafka无消息丢失的配置，每一个其实都能对应上面提到的问题。

1. 不要使用producer.send(msg)，而要使用producer.send(msg, callback)。记住，一定要使用带有回调通知的send方法。
2. 设置acks = all。acks是Producer的一个参数，代表了你对“已提交”消息的定义。如果设置成all，则表明所有副本Broker都要接收到消息，该消息才算是“已提交”。这是最高等级的“已提

交”定义。

3. 设置retries为一个较大的值。这里的retries同样是Producer的参数，对应前面提到的Producer自动重试。当出现网络的瞬时抖动时，消息发送可能会失败，此时配置了retries > 0的Producer能够自动重试消息发送，避免消息丢失。

4. 设置unclean.leader.election.enable = false。这是Broker端的参数，它控制的是哪些Broker有资格竞选分区的Leader。如果一个Broker落后原先的Leader太多，那么它一旦成为新的Leader，必然会造成消息的丢失。故一般都要将该参数设置成false，即不允许这种情况的发生。

5. 设置replication.factor >= 3。这也是Broker端的参数。其实这里想表述的是，最好将消息多保存几份，毕竟目前防止消息丢失的主要机制就是冗余。

6. 设置min.insync.replicas > 1。这依然是Broker端参数，控制的是消息至少要被写入到多少个副本才算是“已提交”。设置成大于1可以提升消息持久性。在实际环境中千万不要使用默认值1。

7. 确保replication.factor > min.insync.replicas。如果两者相等，那么只要有一个副本挂机，整个分区就无法正常工作了。我们不仅要改善消息的持久性，防止数据丢失，还要在不降低可用性的基础上完成。推荐设置成replication.factor = min.insync.replicas + 1。

8. 确保消息消费完成再提交。Consumer端有个参数enable.auto.commit，最好把它设置成false，并采用手动提交位移的方式。就像前面说的，这对于单Consumer多线程处理的场景而言是至关重要的。

小结

今天，我们讨论了Kafka无消息丢失的方方面面。我们先从什么是消息丢失开始说起，明确了Kafka持久化保证的责任边界，随后以这个规则为标尺衡量了一些常见的数据丢失场景，最后通过分析这些场景，我给出了Kafka无消息丢失的“最佳实践”。总结起来，我希望你今天能有两个收获：

- 明确Kafka持久化保证的含义和限定条件。
- 熟练配置Kafka无消息丢失参数。

开放讨论

其实，Kafka还有一种特别隐秘的消息丢失场景：增加主题分区。当增加主题分区后，在某段“不凑巧”的时间间隔后，Producer先于Consumer感知到新增加的分区，而Consumer设置的是“从最新位移处”开始读取消息，因此在Consumer感知到新分区前，Producer发送的这些消息就全部“丢失”了，或者说Consumer无法读取到这些消息。严格来说这是Kafka设计上的一个小缺陷，

你有什么解决的办法吗？

欢迎写下你的思考和答案，我们一起讨论。如果你觉得有所收获，也欢迎把文章分享给你的朋友。

精选留言

阳明 6

总结里的的第二条ack=all和第六条的说明是不是有冲突

2019-06-27

作者回复

其实不冲突。如果ISR中只有1个副本了，acks=all也就相当于acks=1了，引入min.insync.replicas的目的就是为了做一个下限的限制：不能只满足于ISR全部写入，还要保证ISR中的写入个数不少于min.insync.replicas。

2019-06-27

cricket1981 4

consumer改用"从最早位置"读解决新加分区造成的问题

2019-06-27

nightmare 2

多线程消费这么确保手动提交offset管理不会丢失呢，期待老师给一个消费端最佳实践

2019-06-28

lmtoo

2

最后一个问题，难道新增分区之后，producer先感知并发送数据，消费者后感知，消费者的offset会定位到新分区的最后一条消息？消费者没有提交offset怎么会从最后一条开始的呢？

2019-06-27

作者回复

如果你配置了auto.offset.reset=latest就会这样的

2019-06-28

明翼

2

这个问题我想个办法就是程序停止再增加分区，如果不能停止那就找个通知机制了。请教一个问题min.insync.replicas这个参数如果设置成3，假设副本数设置为4，那岂不是只支持一台broker坏掉的情况？本来支持三台坏掉的，老师我理解的对不对

2019-06-27

作者回复

嗯嗯，是的。本来就是为了更强的消息持久化保证，只能牺牲一点高可用性了~~

2019-06-27

Alan

1

1 外部持久化每个topic的每个消费者组的每个patition的offset。

2 程序重启时继续上一次的offset

3 监控每个partition的offset，每次的from offset和to offset是不是线性连续的消费

4 允许重复消费，在消费端去重

2019-06-28

QQ怪

1

不知道是不是可以这样，生产者感知到了有新分区加入立即通知broke端下的消费者不能消费消息，直到消费端都感应到了加入的新分区之后，生产者和消费者才继续工作

2019-06-27

空知

1

老师问下
第7条 一个副本挂掉 整个分区不能用了 是因为每次都必须保证可用副本个数 必须跟提交时候一致 才可以正常使用,又没有冗余副本导致的嘛?

2019-06-27

作者回复

是因为不满足min.insync.replicas的要求了。比如该参数=2，当前ISR中只剩1个副本了，那么producer就没法生产新的消息了。

2019-06-28

没事走两步

1

如果consumer改用"从最早位置"读解决新加分区造成的问题，那会不会导致旧的分区里的已被消费过的消息重新全部被消费一次

2019-06-27

作者回复

只要位移没有越界以及有提交的位移，那么就不会出现这种场景。

2019-06-28

燃烧的M豆 0

最佳实践 5 这个 replication.factor 是 topic 级别的参数是可以在创建 topic 的时候带上的。
最佳实践 4 这个 unclean.leader.election.enable 应该才是 broker 级别的参数？

2019-07-01

天天向上 0

其实 auto.offset.reset这个配置也很重要 对于不自动提交，且忘记手动提交的场景来说 也是个困惑的因素

2019-07-01

Geek_jacky 0

你好，胡老师，我想每次隔10条记录进去消费一次，如果只有1个partition怎么做，如果有多个partition应该如何处理呢？个人认为可以手动在offset上进行处理，那么多个partition中的数据也只能保证每个partition中的消息间隔为10条，总感觉挺笨的，还有没有其他的办法？

2019-07-01

Geek_Sue 0

胡老师，您好，不知道您后面有没有计划针对单Consumer多线程处理的方式进行详细说明？我这边工作中恰好有这样的场景，现在是利用kafka stream来处理的，没有利用到多线程。

2019-07-01

作者回复

Kafka Streams后台就是多线程的处理机制（Stream Thread)，而且Kafka Streams的确是推荐方案之一。

2019-07-01

曹伟雄 0

老师你好，请教个问题。关于offset，有必要外部持久化记录吗？ 出问题后查出来继续消费。你说的更新offset是怎么处理? 是直接调用它的api更新吗？ 谢谢

2019-06-30

作者回复

你可以选择将offset保存在外部持久化设备中，不过更常见的做法是使用consumer API保存在Kafka里面。常见的API包括commitSync和commitAsync

2019-07-01

曹伟雄 0

单个 Consumer 程序使用多线程来消费消息说起来容易，写成代码却异常困难，因为你很难正

确地处理位移的更新，也就是说避免无消费消息丢失很简单，但极易出现消息被消费了多次的情况。
关于这个问题，老师能否提供个java代码的最佳实践？谢谢！

2019-06-30

作者回复

写过一两篇，https://www.cnblogs.com/huxi2b/p/7089854.html，

但总觉得不太完美。如果你想深入了解的话，推荐读一下Flink Kafka Connector的源码

2019-07-01

lmtoo 0

我还有一个疑问，broker接收到消息直接交给操作系统的虚拟内存了，然后Producer以为是提交成功了，这个时候突然断电，那在虚拟内存里的消息不就丢了吗？不管配置什么参数，也是没法保证消息不丢失的呀，如果用kill -9 杀掉broker的进程，是不是虚拟内存里的消息也就不持久化到磁盘了？正确停止broker而又不会丢失消息的命令是什么？

2019-06-29

作者回复

停止broker的命令是kafka-server-stop，你说的情况的确会发生，好在Kafka提供了副本机制在软件层面上保证消息持久性。

2019-07-01

光辉 0

老师，你好。仔细阅读文稿后，仍有一些困惑
1、如果只用 send()方法（fire and forget）， 即使配置retries，producer也是不知道消息状态，是不会重试的。所以说配置retries，要搭配send(msg, callback)，这么理解正确么？
2、配置了retries, producer是怎么知道哪条消息发送失败了，然后重试

2019-06-28

作者回复

1. 不是。如果配置了retries，即使调用send(msg)也是会重试的。这是Kafka producer自己实现的机制，不需要用户干预
2. Broker发送response给producer，里面会保存error信息以及那个(些)batch出错了

2019-07-01

趙衍 0

老师好！抱歉我之前表述的不够清楚，请问老师可以贴出官方社区对增加主题分区以后Consumer无法读取到部分新消息这个问题的解决方案吗，就是您在最后的开放讨论里提出的这个问题，我想更深入地学习一下，谢谢老师！

2019-06-28

作者回复

坦率来说，我是希望大家一起讨论，而且这个没有绝对的解决方案。能想到的一个简单方法是让consumer端缓存订阅信息，如果发现新的订阅分区出现，手动调整位移到最开始处执行（比如consumer.seekToBeginning）

2019-07-01

在路上

0

如果是多线程异步处理消费消息，Consumer 程序不要开启自动提交位移，而是要应用程序手动提交位移，这里的手动提交位移是什么意思啊，不太明白？

2019-06-28

光辉

0

老师，你好！

producer.send(msg, callback)中callback主要用来做什么？可以用来重新发送数据么？如果可以的话，跟producer的配置retries是不是功能重复了

2019-06-27

作者回复

可以。retries是producer自动帮你重试。callback中你可以做一些处理之后再重试。

2019-06-28