11 | 无消息丢失配置怎么实现?

胡夕 · Kafka核心技术与实战

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。

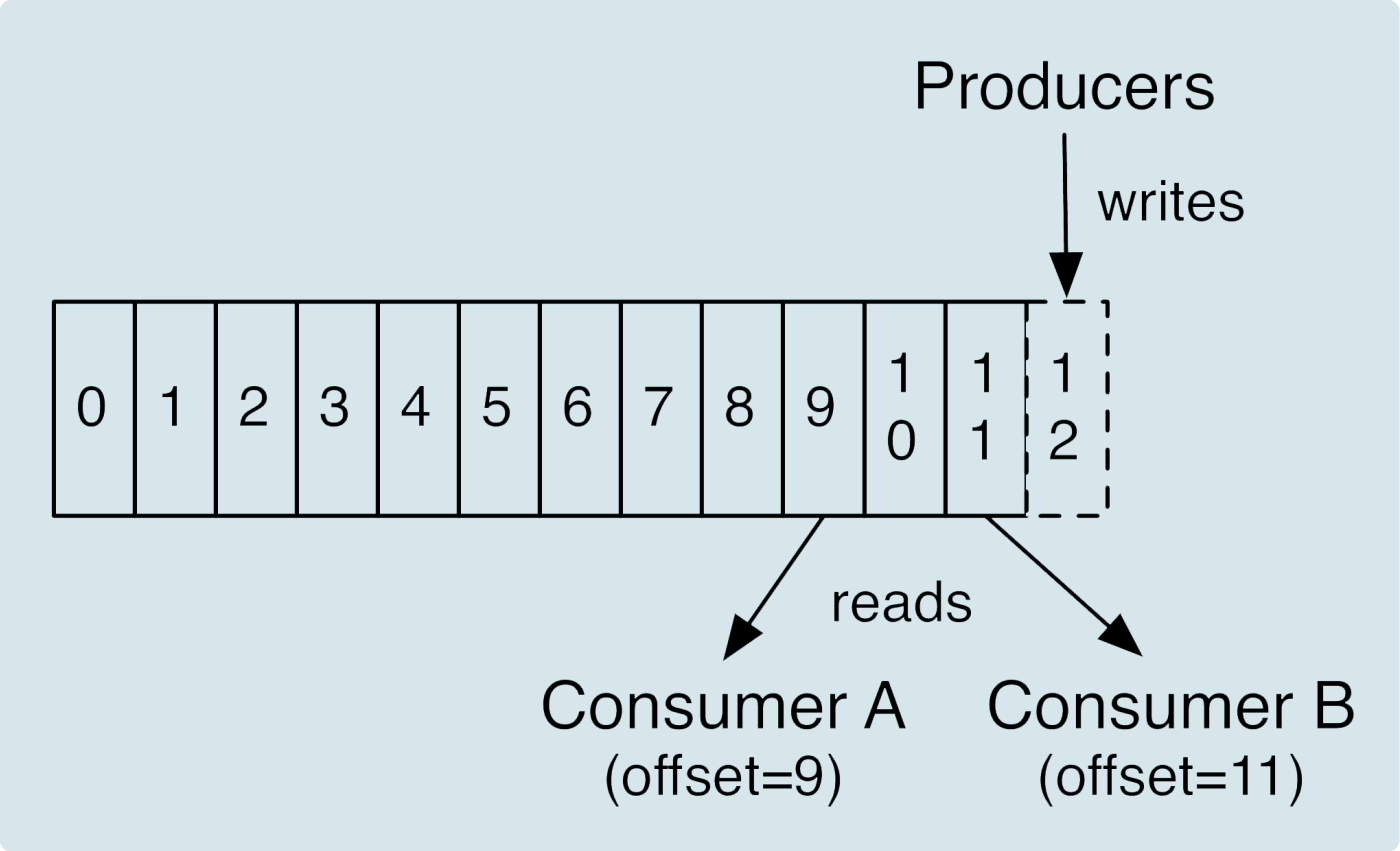
不过，就算不是 Kafka 的“锅”，我们也要解决这个问题吧。实际上，解决此问题的方法非常简单：**Producer 永远要使用带有回调通知的发送 API，也就是说不要使用 producer.send(msg)，而要使用 producer.send(msg, callback)**。不要小瞧这里的 callback（回调），它能准确地告诉你消息是否真的提交成功了。一旦出现消息提交失败的情况，你就可以有针对性地进行处理。

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。

无消息丢失配置怎么实现？

- 使用producer.send(msg, callback)。
- 设置acks=all。
- 设置retries为一个较大的值。
- 设置unclean.leader.election.enable=false。
- 设置replication.factor >=3。
- 设置min.insync.replicas>1。
- 确保replication.factor > min.insync.replicas。
- 确保消息消费完成再提交。

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。

AI智能总结

Kafka消息丢失一直备受关注，本文从Kafka对已提交消息的持久化保证出发，解释了Kafka在何种情况下能够保证消息不丢失。文章列举了两种常见的“消息丢失”案例：生产者程序丢失数据和消费者程序丢失数据，并提出了解决方法。此外，还提到了一种隐蔽的消息丢失场景，即增加主题分区后可能出现的问题。为了避免消息丢失，文章给出了一系列Kafka无消息丢失的配置建议，包括使用带有回调通知的发送API、设置acks为all、配置retries、设置unclean.leader.election.enable为false等。总结起来，本文帮助读者更好地理解Kafka消息丢失问题，并提供了解决方案。

全部留言 (142)

最新　精选

阳明
2019-06-27

总结里的的第二条ack=all和第六条的说明是不是有冲突

> 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。

共 29 条评论>　　102

曹伟雄
2019-06-30

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!

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

共 2 条评论> 30

陈国林
2020-01-09

老师好，请教一个问题，ack=1的时候，min.insync.replicas还会生效吗？或者说还有必要吗，感谢 🤝

作者回复: 不生效，min.insync.replicas只有在acks=-1时才生效

共 5 条评论> 27

lmtoo
2019-06-27

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?

作者回复: 如果你配置了auto.offset.reset=latest就会这样的

共 3 条评论> 23

ban
2019-08-03

老师，
如果我有10个副本，isr=10，然后我配置ack=all，min.insync.replicas=5，
这时候这两个参数以谁为准，生产一个消息，必须是全部副本都同步才算提交，还是只要5个副本才算提交?

作者回复: min.insync.replicas是保证下限的。acks=all的含义是producer会等ISR中所有副本都写入成功才返回，但如果不设置min.insync.replicas = 5，默认是1，那么假设ISR中只有1个副本，只要写入这个副本成功producer也算其正常写入，因此min.insync.replicas保证的写入副本的下限。

共 7 条评论>

22

redis

2019-11-21

你好胡老师，想问一下 kafka是在落地刷盘之后，同步副本成功后，才能会被消费吗?

作者回复: 其实，有可能在落盘之前就被消费了。能否被消费不是看是否flush到磁盘，而是看leader副本的高水位是否越过了该条消息

共 6 条评论>

16

永光

2019-06-27

看了评论区回答还是不太理解，第二条ack=all与第六条min.insync.replicas 怎样协调工作的，总感觉是有冲突的。
问题是：
第二条的“已提交”和第六条的“已提交”是同一个意思吗？如果是同一个意思，那定义为什么不一样呀？

作者回复: acks=all表示消息要写入所有ISR副本，但没要求ISR副本有多少个。min.insync.replicas做了这样的保证

共 6 条评论>

15

美美

2019-11-24

胡老师 还有一种消息重复的情况希望帮忙分析下。producer发送消息后，broker成功写入消息了，但是ack因为网络问题没有到达producer，生产者可能会重试发送这条消息。
这种问题如何避免重复消费呢

作者回复: 使用幂等producer

共 6 条评论>

12

yang

2020-07-28

老师，我针对您的提问思考并查阅了一下相关资料，说一下我的思考哈：

我们假设有且仅有一个producer只在这个consumer感知到之前，新的partition分区只写了那么几条记录，不会再有其他producer写数据到这个新的partition中。

新增partition的情况，rebalance时由于我们默认offset.auto.reset=lastest，因此在使用了这个默认配置之下，producer较consumer先感知到新的partition将数据发送到新的partition，而consumer之后才感知到这个consumer，此时由于这个新的partition的offset是第一次消费，没有已提交的offset，所以使用latest从最新的位移开始读取，也就是producer写入消息offset + 1的那个位置开始读取，因此也就读取不到数据。

latest：有提交位移就从提交位移开始处理，没有提交位移就从最新的位移开始处理。

earlist: 有提交位移就从提交位移开始处理，没有提交位移就从最早的位移开始处理。

current: 从当前的提交位移开始处理。

因此，碰到上述情况，我们可以使用seekToBegin从这个新分区的开始位置读即可。

我能想到的办法是，实现一个ConsumerRebalanceListener，重写onPartitionsAssigned方法，在这个方法里我们每次都从自己维护的数据库系统里取offset，能取到说明这个partition是之前就存在的，按已有的offset继续消费就可以了，没有取到的记录表示是新增的partition，那么就从0开始消费并且保存当前offset到数据库。不论是不是新的，都可以seek到指定的位置，只是我们有没有维护到这个partition的offset记录的区别。也就不用针对这个新分区指定offset.auto.rset=earlist了吧?

希望得到胡老师的交流哈~

作者回复: 我觉得是很好的办法。值得一试：）

共 3 条评论> 10

浪迹人生
2019-11-01

请问消息的createTimestamp 是在生产者服务器上生成的，还是在进入不同partition 后生成的？我能不能根据这个时间戳来判断不同分区的消息原始全局顺序？谢谢🙏

作者回复: 在生产者服务器上生成的。个人感觉不可以，毕竟每个producer服务器上的时钟不是实时同步的。事实上，用时钟来保证同步性是一件非常不靠谱的事情

共 4 条评论>

10