二

19 CommitFailedException异常怎么处理?

你好，我是胡夕。今天我来跟你聊聊 CommitFailedException 异常的处理。

说起这个异常，我相信用过 Kafka Java Consumer 客户端 API 的你一定不会感到陌生。**所谓 CommitFailedException，顾名思义就是 Consumer 客户端在提交位移时出现了错误或异常，而且还是那种不可恢复的严重异常。**如果异常是可恢复的瞬时错误，提交位移的 API 自己就能规避它们了，因为很多提交位移的 API 方法是支持自动错误重试的，比如我们在上一期中提到的**commitSync 方法。**

每次和 CommitFailedException 一起出现的，还有一段非常著名的注释。为什么说它很“著名”呢？第一，我想不出在近 50 万行的 Kafka 源代码中，还有哪个异常类能有这种待遇，可以享有这么大段的注释，来阐述其异常的含义；第二，纵然有这么长的文字解释，却依然有很多人对该异常想表达的含义感到困惑。

现在，我们一起领略下这段文字的风采，看看社区对这个异常的最新解释：

> Commit cannot be completed since the group has already rebalanced and assigned the partitions to another member. This means that the time between subsequent calls to poll() was longer than the configured max.poll.interval.ms, which typically implies that the poll loop is spending too much time message processing. You can address this either by increasing max.poll.interval.ms or by reducing the maximum size of batches returned in poll() with max.poll.records.

这段话前半部分的意思是，本次提交位移失败了，原因是消费者组已经开启了 Rebalance 过程，并且将要提交位移的分区分配给了另一个消费者实例。出现这个情况的原因是，你的消费者实例连续两次调用 poll 方法的时间间隔超过了期望的 max.poll.interval.ms 参数值。这通常表明，你的消费者实例花费了太长的时间进行消息处理，耽误了调用 poll 方法。

在后半部分，社区给出了两个相应的解决办法（即橙色字部分）：

1. 增加期望的时间间隔 max.poll.interval.ms 参数值。
2. 减少 poll 方法一次性返回的消息数量，即减少 max.poll.records 参数值。

在详细讨论这段文字之前，我还想提一句，实际上这段文字总共有 3 个版本，除了上面的这个最新版本，还有 2 个版本，它们分别是：

> Commit cannot be completed since the group has already rebalanced and assigned the partitions to another member. This means that the time between subsequent calls to poll() was longer than the configured session.timeout.ms, which typically implies that the poll loop is spending too much time message processing. You can address this either by increasing the session timeout or by reducing the maximum size of batches returned in poll() with max.poll.records.

> Commit cannot be completed since the group has already rebalanced and assigned the partitions to another member. This means that the time between subsequent calls to poll() was longer than the configured max.poll.interval.ms, which typically implies that the poll loop is spending too much time message processing. You can address this either by increasing the session timeout or by reducing the maximum size of batches returned in poll() with max.poll.records.

这两个较早的版本和最新版相差不大，我就不详细解释了，具体的差异我用橙色标注了。我之所以列出这些版本，就是想让你在日后看到它们时能做到心中有数，知道它们说的是一个事情。

其实不论是哪段文字，它们都表征位移提交出现了异常。下面我们就来讨论下该异常是什么时候被抛出的。从源代码方面来说，CommitFailedException 异常通常发生在手动提交位移时，即用户显式调用 KafkaConsumer.commitSync() 方法时。从使用场景来说，有两种典型的场景可能遭遇该异常。

场景一

我们先说说最常见的场景。当消息处理的总时间超过预设的 max.poll.interval.ms 参数值时，Kafka Consumer 端会抛出 CommitFailedException 异常。这是该异常最“正宗”的登场方式。你只需要写一个 Consumer 程序，使用 KafkaConsumer.subscribe 方法随意订阅一个主题，之后设置 Consumer 端参数 max.poll.interval.ms=5 秒，最后在循环调用 KafkaConsumer.poll 方法之间，插入 Thread.sleep(6000) 和手动提交位移，就可以成功复现这个异常了。在这里，我展示一下主要的代码逻辑。

```
…
Properties props = new Properties();
…
props.put("max.poll.interval.ms", 5000);
consumer.subscribe(Arrays.asList("test-topic"));
```

```
while (true) {
    ConsumerRecords<String, String> records =
                consumer.poll(Duration.ofSeconds(1));
    // 使用 Thread.sleep 模拟真实的消息处理逻辑
    Thread.sleep(6000L);
    consumer.commitSync();
}
```

如果要防止这种场景下抛出异常，你需要简化你的消息处理逻辑。具体来说有 4 种方法。

1. **缩短单条消息处理的时间。**比如，之前下游系统消费一条消息的时间是 100 毫秒，优化之后成功地下降到 50 毫秒，那么此时 Consumer 端的 TPS 就提升了一倍。
2. **增加 Consumer 端允许下游系统消费一批消息的最大时长。**这取决于 Consumer 端参数 max.poll.interval.ms 的值。在最新版的 Kafka 中，该参数的默认值是 5 分钟。如果你的消费逻辑不能简化，那么提高该参数值是一个不错的办法。值得一提的是，Kafka 0.10.1.0 之前的版本是没有这个参数的，因此如果你依然在使用 0.10.1.0 之前的客户端 API，那么你需要增加 session.timeout.ms 参数的值。不幸的是，session.timeout.ms 参数还有其他的含义，因此增加该参数的值可能会有其他方面的“不良影响”，这也是社区在 0.10.1.0 版本引入 max.poll.interval.ms 参数，将这部分含义从 session.timeout.ms 中剥离出来的原因之一。
3. **减少下游系统一次性消费的消息总数。**这取决于 Consumer 端参数 max.poll.records 的值。当前该参数的默认值是 500 条，表明调用一次 KafkaConsumer.poll 方法，最多返回 500 条消息。可以说，该参数规定了单次 poll 方法能够返回的消息总数的上限。如果前两种方法对你都不适用的话，降低此参数值是避免 CommitFailedException 异常最简单的手段。
4. **下游系统使用多线程来加速消费。**这应该算是“最高级”同时也是最难实现的解决办法了。具体的思路就是，让下游系统手动创建多个消费线程处理 poll 方法返回的一批消息。之前你使用 Kafka Consumer 消费数据更多是单线程的，所以当消费速度无法匹及 Kafka Consumer 消息返回的速度时，它就会抛出 CommitFailedException 异常。如果是多线程，你就可以灵活地控制线程数量，随时调整消费承载能力，再配以目前多核的硬件条件，该方法可谓是防止 CommitFailedException 最高档的解决之道。事实上，很多主流的大数据流处理框架使用的都是这个方法，比如 Apache Flink 在集成 Kafka 时，就是创建了多个 KafkaConsumerThread 线程，自行处理多线程间的数据消费。不过，凡事有利就有弊，这个方法实现起来并不容易，特别是在多个线程间如何处理位移提交这个问题上，更是极容易出错。在专栏后面的内容中，我将着重和你讨论一下多线程消费的实现方案。

综合以上这 4 个处理方法，我个人推荐你首先尝试采用方法 1 来预防此异常的发生。优化下游系统的消费逻辑是百利而无一害的法子，不像方法 2、3 那样涉及到 Kafka Consumer 端 TPS 与消费延时（Latency）的权衡。如果方法 1 实现起来有难度，那么你可以按照下面的法则来实践方法 2、3。

首先，你需要弄清楚你的下游系统消费每条消息的平均延时是多少。比如你的消费逻辑是从 Kafka 获取到消息后写入到下游的 MongoDB 中，假设访问 MongoDB 的平均延时不超过 2 秒，那么你可以认为消息处理需要花费 2 秒的时间。如果按照 max.poll.records 等于 500 来计算，一批消息的总消费时长大约是 1000 秒，因此你的 Consumer 端的 max.poll.interval.ms 参数值就不能低于 1000 秒。如果你使用默认配置，那默认值 5 分钟显然是不够的，你将有很大概率遭遇 CommitFailedException 异常。将 max.poll.interval.ms 增加到 1000 秒以上的做法就属于上面的第 2 种方法。

除了调整 max.poll.interval.ms 之外，你还可以选择调整 max.poll.records 值，减少每次 poll 方法返回的消息数。还拿刚才的例子来说，你可以设置 max.poll.records 值为 150，甚至更少，这样每批消息的总消费时长不会超过 300 秒（150*2=300），即 max.poll.interval.ms 的默认值 5 分钟。这种减少 max.poll.records 值的做法就属于上面提到的方法 3。

场景二

Okay，现在我们已经说完了关于 CommitFailedException 异常的经典发生场景以及应对办法。从理论上讲，关于该异常你了解到这个程度，已经足以帮助你应对应用开发过程中由该异常带来的“坑”了 。但其实，该异常还有一个不太为人所知的出现场景。了解这个冷门场景，可以帮助你拓宽 Kafka Consumer 的知识面，也能提前预防一些古怪的问题。下面我们就来说说这个场景。

之前我们花了很多时间学习 Kafka 的消费者，不过大都集中在消费者组上，即所谓的 Consumer Group。其实，Kafka Java Consumer 端还提供了一个名为 Standalone Consumer 的独立消费者。它没有消费者组的概念，每个消费者实例都是独立工作的，彼此之间毫无联系。不过，你需要注意的是，独立消费者的位移提交机制和消费者组是一样的，因此独立消费者的位移提交也必须遵守之前说的那些规定，比如独立消费者也要指定 group.id 参数才能提交位移。你可能会觉得奇怪，既然是独立消费者，为什么还要指定 group.id 呢？没办法，谁让社区就是这么设计的呢？总之，消费者组和独立消费者在使用之前都要指定 group.id。

现在问题来了，如果你的应用中同时出现了设置相同 group.id 值的消费者组程序和独立消费者程序，那么当独立消费者程序手动提交位移时，Kafka 就会立即抛出 CommitFailedException 异常，因为 Kafka 无法识别这个具有相同 group.id 的消费者实例，于是就向它返回一个错误，表明它不是消费者组内合法的成员。

虽然说这个场景很冷门，但也并非完全不会遇到。在一个大型公司中，特别是那些将 Kafka 作为全公司级消息引擎系统的公司中，每个部门或团队都可能有自己的消费者应用，谁能保证各自的 Consumer 程序配置的 group.id 没有重复呢？一旦出现不凑巧的重复，发生了上面提到的这种场景，你使用之前提到的哪种方法都不能规避该异常。令人沮丧的是，无论是刚才哪个版本的异常说明，都完全没有提及这个场景，因此，如果是这个原因引发的

CommitFailedException 异常，前面的 4 种方法全部都是无效的。

更为尴尬的是，无论是社区官网，还是网上的文章，都没有提到过这种使用场景。我个人认为，这应该算是 Kafka 的一个 bug。比起返回 CommitFailedException 异常只是表明提交位移失败，更好的做法应该是，在 Consumer 端应用程序的某个地方，能够以日志或其他方式友善地提示你错误的原因，这样你才能正确处理甚至是预防该异常。

小结

总结一下，今天我们详细讨论了 Kafka Consumer 端经常碰到的 CommitFailedException 异常。我们从它的含义说起，再到它出现的时机和场景，以及每种场景下的应对之道。当然，我也留了个悬念，在专栏后面的内容中，我会详细说说多线程消费的实现方式。希望通过今天的分享，你能清晰地掌握 CommitFailedException 异常发生的方方面面，从而能在今后更有效地应对此异常。

上一页 下一页