25 | 消费者组重平衡全流程解析

2019-07-30 胡夕

你好，我是胡夕。今天我要和你分享的主题是：消费者组重平衡全流程解析。

之前我们聊到过消费者组的重平衡流程，它的作用是让组内所有的消费者实例就消费哪些主题分区达成一致。重平衡需要借助Kafka Broker端的Coordinator组件，在Coordinator的帮助下完成整个消费者组的分区重分配。今天我们就来详细说说这个流程。

先提示一下，我会以Kafka 2.3版本的源代码开启今天的讲述。在分享的过程中，对于旧版本的设计差异，我也会显式地说明。这样，即使你依然在使用比较旧的版本也不打紧，毕竟设计原理大体上是没有变化的。

触发与通知

我们先来简单回顾一下重平衡的3个触发条件：

1. 组成员数量发生变化。
2. 订阅主题数量发生变化。
3. 订阅主题的分区数发生变化。

就我个人的经验来看，在实际生产环境中，因命中第1个条件而引发的重平衡是最常见的。另外，消费者组中的消费者实例依次启动也属于第1种情况，也就是说，每次消费者组启动时，必然会触发重平衡过程。

这部分内容我在专栏第15讲中已经详细介绍过了，就不再赘述了。如果你不记得的话，可以先去复习一下。

今天，我真正想引出的是另一个话题：**重平衡过程是如何通知到其他消费者实例的？答案就是，靠消费者端的心跳线程（Heartbeat Thread）**。

Kafka Java消费者需要定期地发送心跳请求（Heartbeat Request）到Broker端的协调者，以表明它还存活着。在Kafka 0.10.1.0版本之前，发送心跳请求是在**消费者主线程**完成的，也就是你写代码调用KafkaConsumer.poll方法的那个线程。

这样做有诸多弊病，最大的问题在于，**消息处理逻辑也是在这个线程中完成的**。因此，一旦消息处理消耗了过长的时间，心跳请求将无法及时发到协调者那里，导致协调者“错误地”认为该消费者已“死”。自0.10.1.0版本开始，社区引入了一个单独的心跳线程来专门执行心跳请求发送，避免了这个问题。

但这和重平衡又有什么关系呢？其实，**重平衡的通知机制正是通过心跳线程来完成的**。当协调者决定开启新一轮重平衡后，它会将“REBALANCE_IN_PROGRESS”封装进心跳请求的响应中，发还给消费者实例。当消费者实例发现心跳响应中包含了“REBALANCE_IN_PROGRESS”，就能立马知道重平衡又开始了，这就是重平衡的通知机制。

对了，很多人还搞不清楚消费者端参数heartbeat.interval.ms的真实用途，我来解释一下。从字面上看，它就是设置了心跳的间隔时间，但这个参数的真正作用是控制重平衡通知的频率。如果你想要消费者实例更迅速地得到通知，那么就可以给这个参数设置一个非常小的值，这样消费者就能更快地感知到重平衡已经开启了。

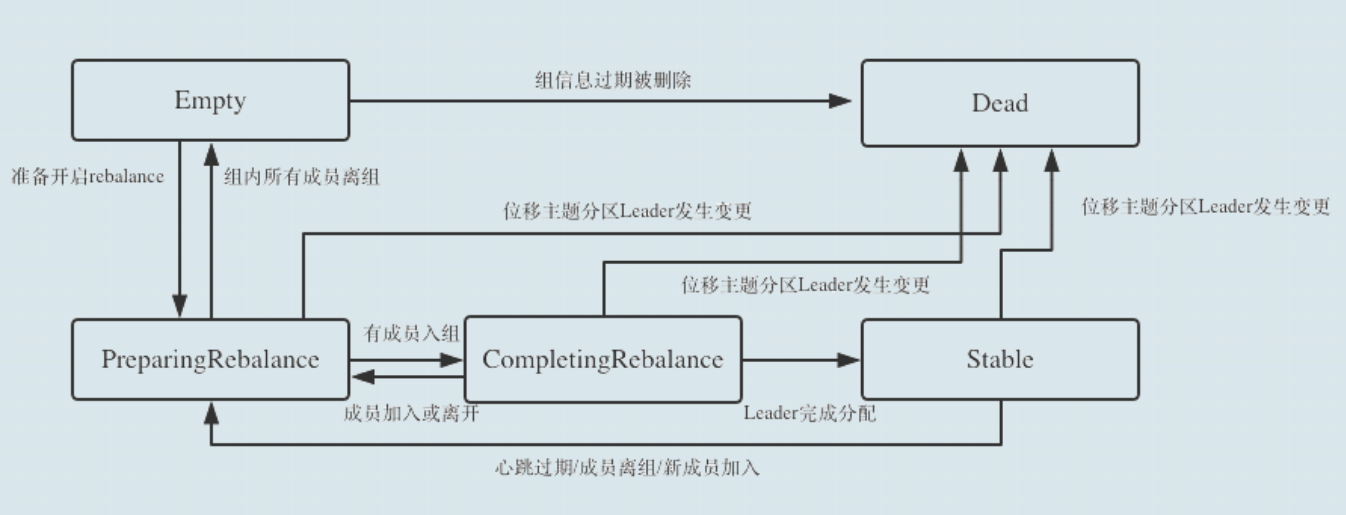
消费者组状态机

重平衡一旦开启，Broker端的协调者组件就要开始忙了，主要涉及到控制消费者组的状态流转。当前，Kafka设计了一套消费者组状态机（State Machine），来帮助协调者完成整个重平衡流程。严格来说，这套状态机属于非常底层的设计，Kafka官网上压根就没有提到过，但你最好还是了解一下，因为**它能够帮助你搞懂消费者组的设计原理，比如消费者组的过期位移（Expired Offsets）删除等**。

目前，Kafka为消费者组定义了5种状态，它们分别是：Empty、Dead、PreparingRebalance、CompletingRebalance和Stable。那么，这5种状态的含义是什么呢？我们一起来看看下面这张表格。

消费者组的5种状态	
状态	含义
Empty	组内没有任何成员，但消费者组可能存在已提交的位移数据，而且这些位移尚未过期。
Dead	同样是组内没有任何成员，但组的元数据信息已经在协调者端被移除。协调者组件保存着当前向它注册过的所有组信息，所谓的元数据信息就类似于这个注册信息。
PreparingRebalance	消费者组准备开启重平衡，此时所有成员都要重新请求加入消费者组。
CompletingRebalance	消费者组下所有成员已经加入，各个成员正在等待分配方案。该状态在老一点的版本中被称为AwaitingSync，它和CompletingRebalance是等价的。
Stable	消费者组的稳定状态。该状态表明重平衡已经完成，组内各成员能够正常消费数据了。

了解了这些状态的含义之后，我们来看一张图片，它展示了状态机的各个状态流转。



我来解释一下消费者组启动时的状态流转过程。一个消费者组最开始是Empty状态，当重平衡过程开启后，它会被置于PreparingRebalance状态等待成员加入，之后变更到CompletingRebalance状态等待分配方案，最后流转到Stable状态完成重平衡。

当有新成员加入或已有成员退出时，消费者组的状态从Stable直接跳到PreparingRebalance状态，此时，所有现存成员就必须重新申请加入组。当所有成员都退出组后，消费者组状态变更为Empty。Kafka定期自动删除过期位移的条件就是，组要处于Empty状态。因此，如果你的消费者组停掉了很长时间（超过7天），那么Kafka很可能就把该组的位移数据删除了。我相信，你在Kafka的日志中一定经常看到下面这个输出：

> Removed □□□ expired offsets in □□□ milliseconds.

这就是Kafka在尝试定期删除过期位移。现在你知道了，只有Empty状态下的组，才会执行过期位移删除的操作。

消费者端重平衡流程

有了上面的内容作铺垫，我们就可以开始介绍重平衡流程了。重平衡的完整流程需要消费者端和协调者组件共同参与才能完成。我们先从消费者的视角来审视一下重平衡的流程。

在消费者端，重平衡分为两个步骤：分别是加入组和等待领导者消费者（Leader Consumer）分配方案。这两个步骤分别对应两类特定的请求：**JoinGroup请求和SyncGroup请求**。

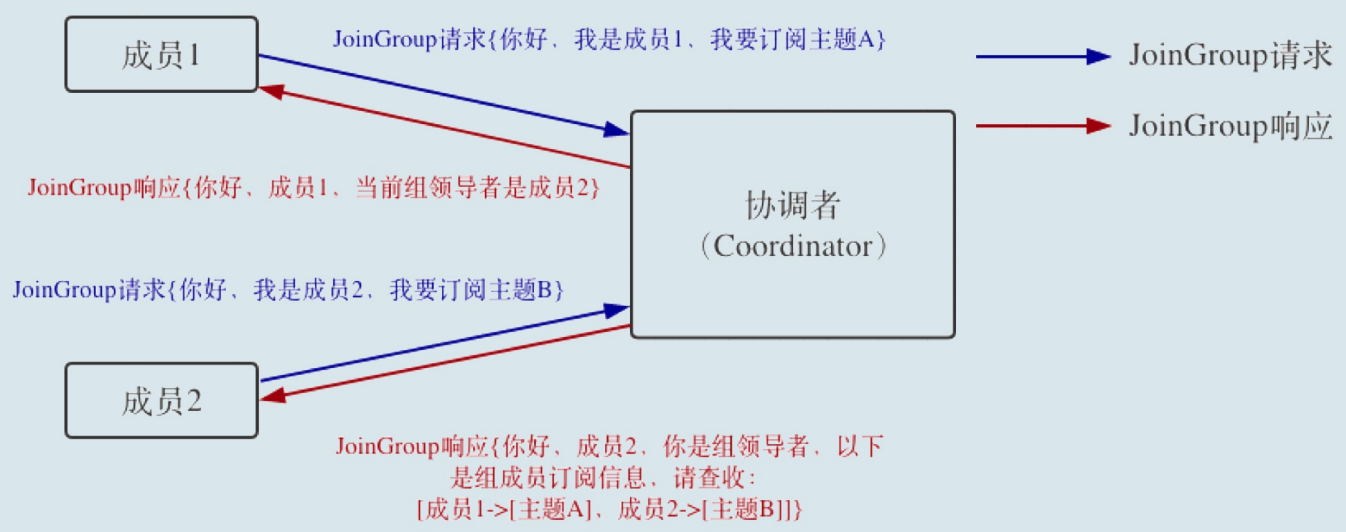
当组内成员加入组时，它会向协调者发送JoinGroup请求。在该请求中，每个成员都要将自己订阅的主题上报，这样协调者就能收集到所有成员的订阅信息。一旦收集了全部成员的JoinGroup请求后，协调者会从这些成员中选择一个担任这个消费者组的领导者。

通常情况下，第一个发送JoinGroup请求的成员自动成为领导者。你一定要注意区分这里的领导者和之前我们介绍的领导者副本，它们不是一个概念。这里的领导者是具体的消费者实例，它既不是副本，也不是协调者。**领导者消费者的任务是收集所有成员的订阅信息，然后根据这些信息，制定具体的分区消费分配方案。**

选出领导者之后，协调者会把消费者组订阅信息封装进JoinGroup请求的响应体中，然后发给领导者，由领导者统一做出分配方案后，进入到下一步：发送SyncGroup请求。

在这一步中，领导者向协调者发送SyncGroup请求，将刚刚做出的分配方案发给协调者。值得注意的是，其他成员也会向协调者发送SyncGroup请求，只不过请求体中并没有实际的内容。这一步的主要目的是让协调者接收分配方案，然后统一以SyncGroup响应的方式分发给所有成员，这样组内所有成员就都知道自己该消费哪些分区了。

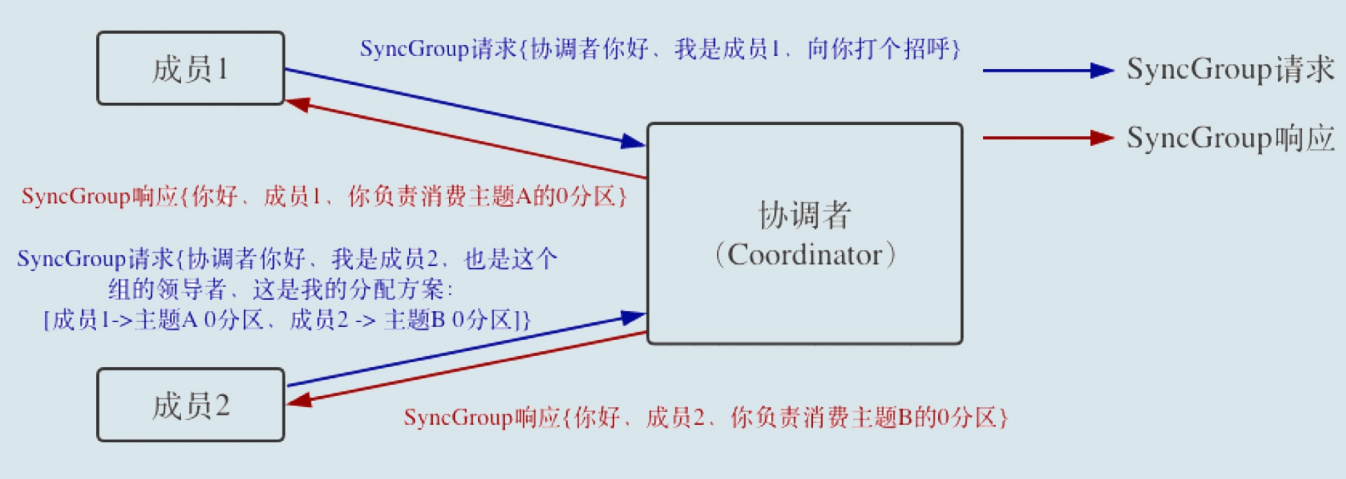
接下来，我用一张图来形象地说明一下JoinGroup请求的处理过程。



就像前面说的，JoinGroup请求的主要作用是将组成员订阅信息发送给领导者消费者，待领导者

制定好分配方案后，重平衡流程进入到SyncGroup请求阶段。

下面这张图描述的是SyncGroup请求的处理流程。



SyncGroup请求的主要目的，就是让协调者把领导者制定的分配方案下发给各个组内成员。当所有成员都成功接收到分配方案后，消费者组进入到Stable状态，即开始正常的消费工作。

讲完这里，**消费者端**的重平衡流程我已经介绍完了。接下来，我们从**协调者端**来看一下重平衡是怎么执行的。

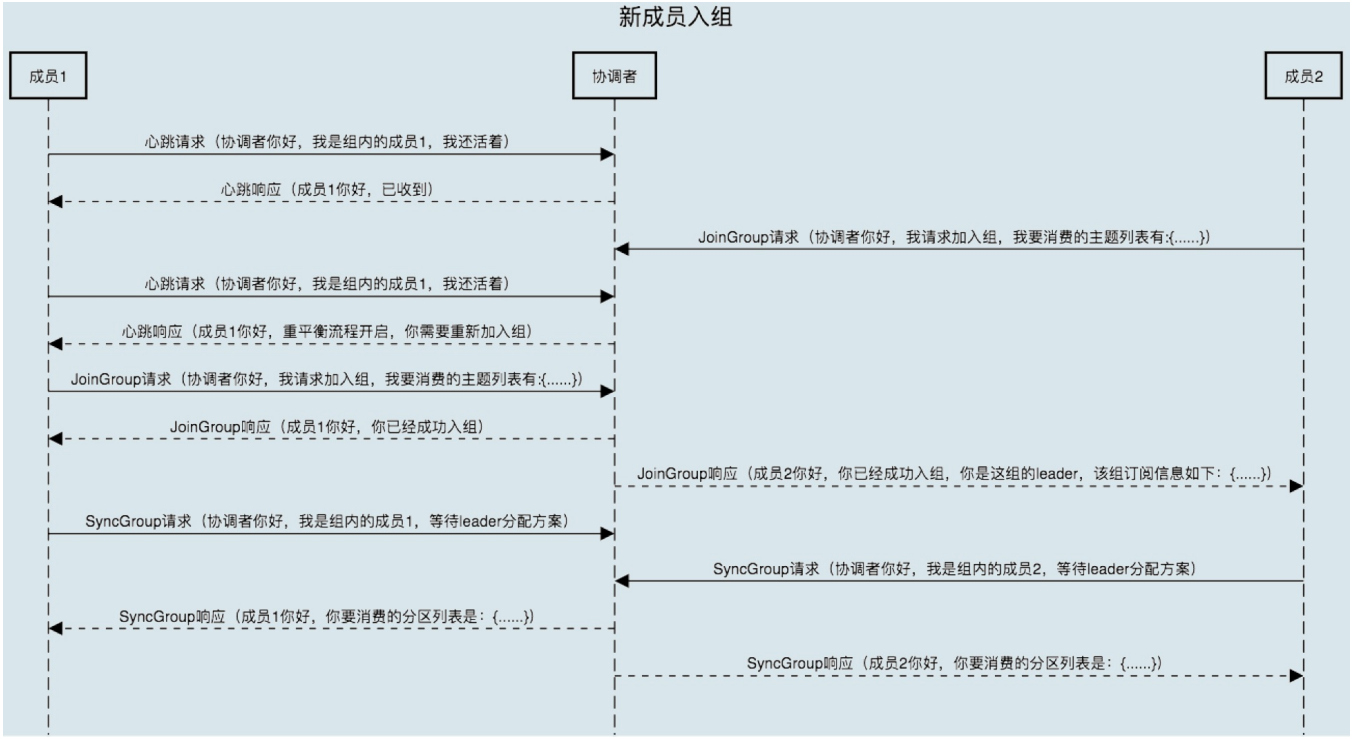
Broker端重平衡场景剖析

要剖析协调者端处理重平衡的全流程，我们必须要分几个场景来讨论。这几个场景分别是新成员加入组、组成员主动离组、组成员崩溃离组、组成员提交位移。接下来，我们一个一个来讨论。

场景一：新成员入组。

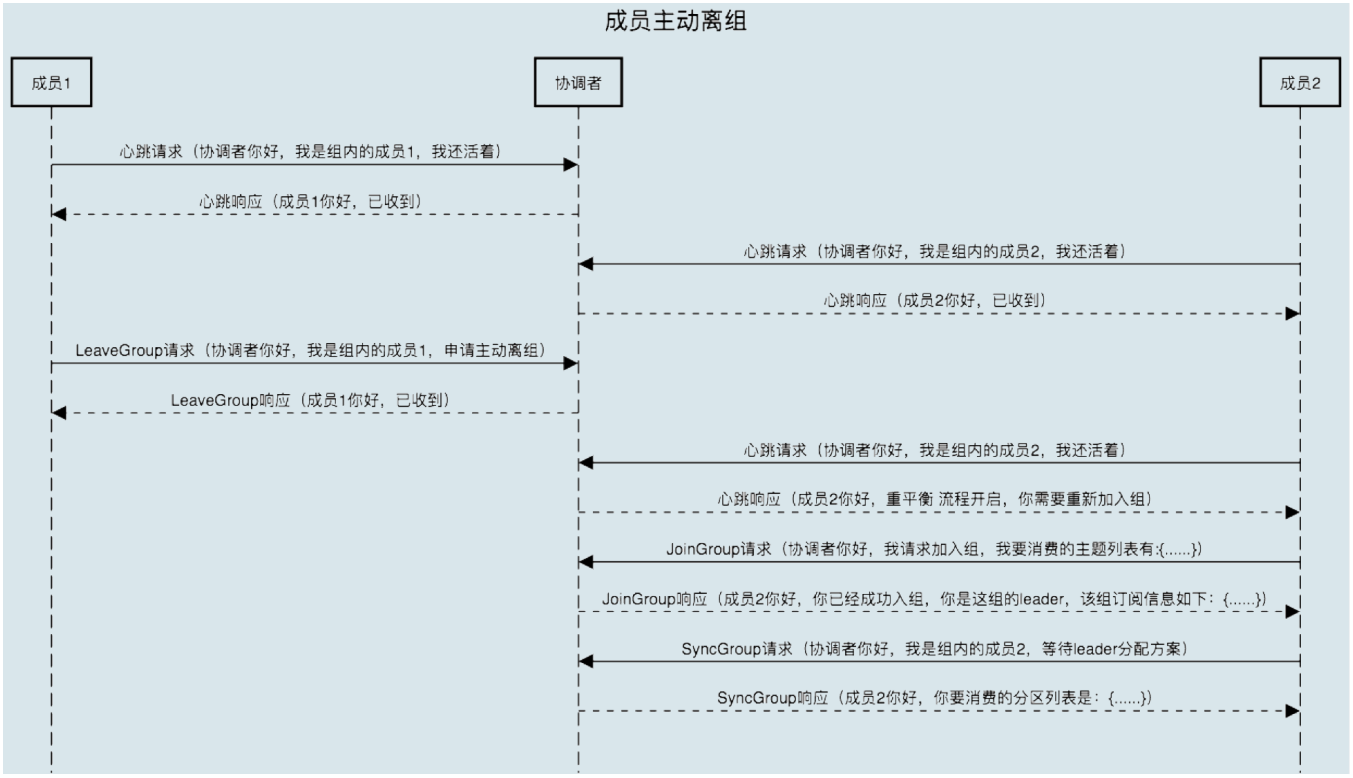
新成员入组是指组处于Stable状态后，有新成员加入。如果是全新启动一个消费者组，Kafka是有一些自己的小优化的，流程上会有些许的不同。我们这里讨论的是，组稳定了之后有新成员加入的情形。

当协调者收到新的JoinGroup请求后，它会通过心跳请求响应的方式通知组内现有的所有成员，强制它们开启新一轮的重平衡。具体的过程和之前的客户端重平衡流程是一样的。现在，我用一张时序图来说明协调者一端是如何处理新成员入组的。



场景二：组成员主动离组。

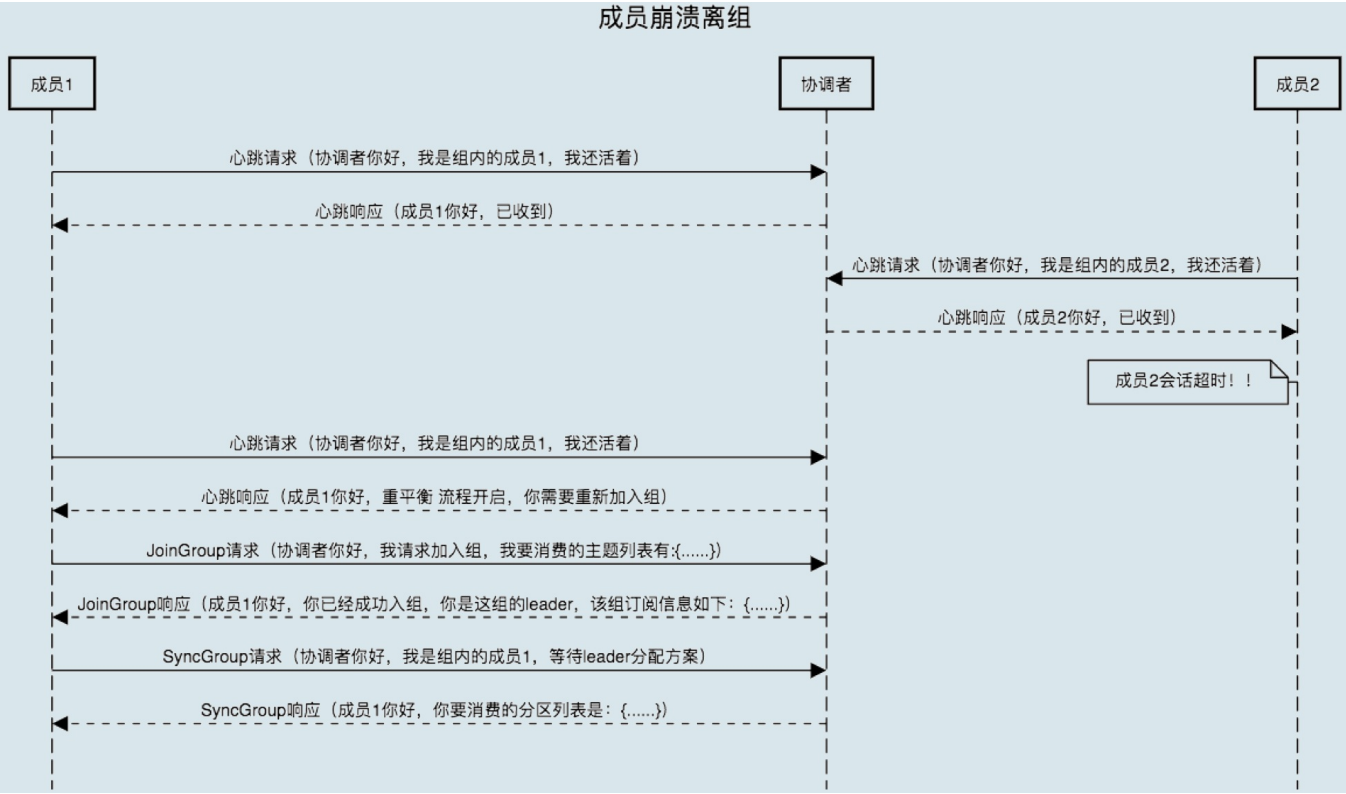
何谓主动离组？就是指消费者实例所在线程或进程调用close()方法主动通知协调者它要退出。这个场景就涉及到了第三类请求：**LeaveGroup请求**。协调者收到LeaveGroup请求后，依然会以心跳响应的方式通知其他成员，因此我就不再赘述了，还是直接用一张图来说明。



场景三：组成员崩溃离组。

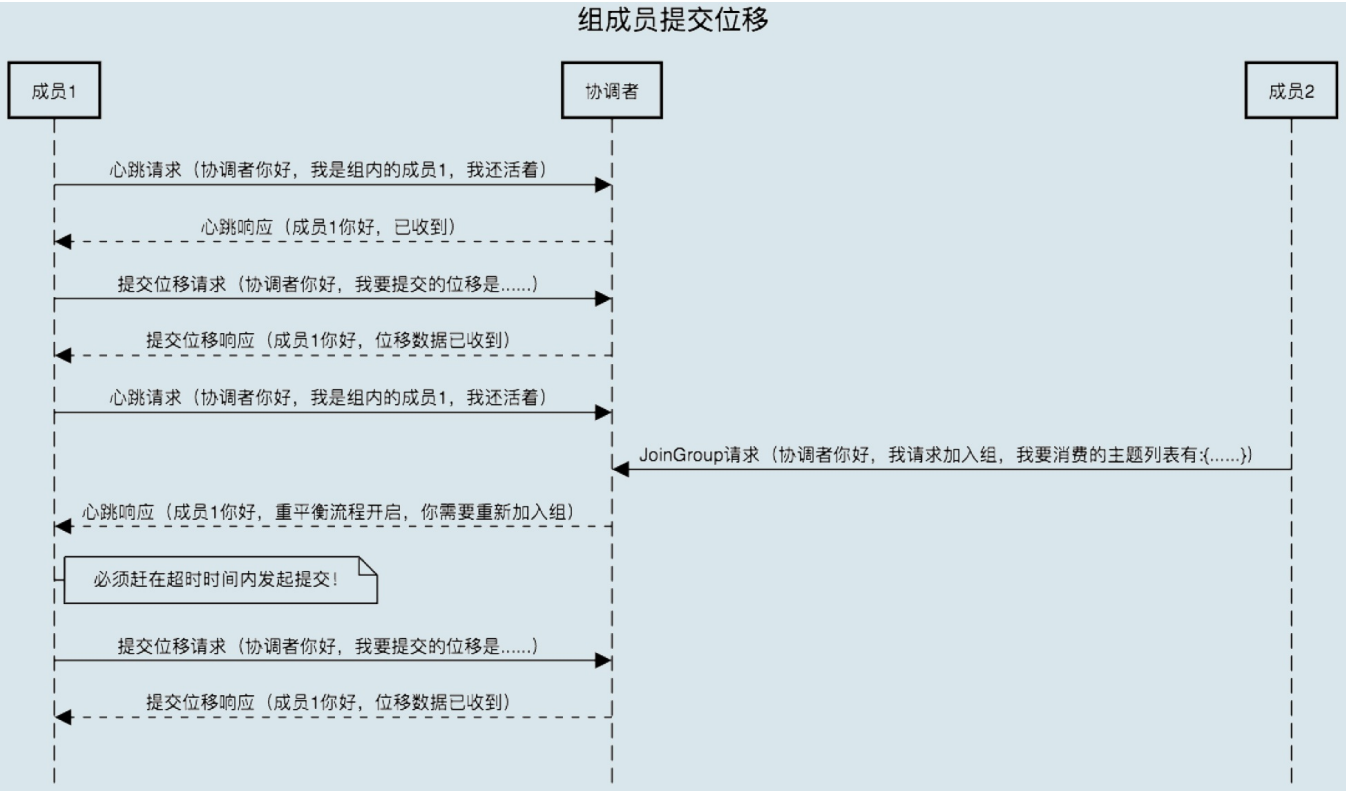
崩溃离组是指消费者实例出现严重故障，突然宕机导致的离组。它和主动离组是有区别的，因为后者是主动发起的离组，协调者能马上感知并处理。但崩溃离组是被动的，协调者通常需要

等待一段时间才能感知到，这段时间一般是由消费者端参数session.timeout.ms控制的。也就是说，Kafka一般不会超过session.timeout.ms就能感知到这个崩溃。当然，后面处理崩溃离组的流程与之前是一样的，我们来看看下面这张图。



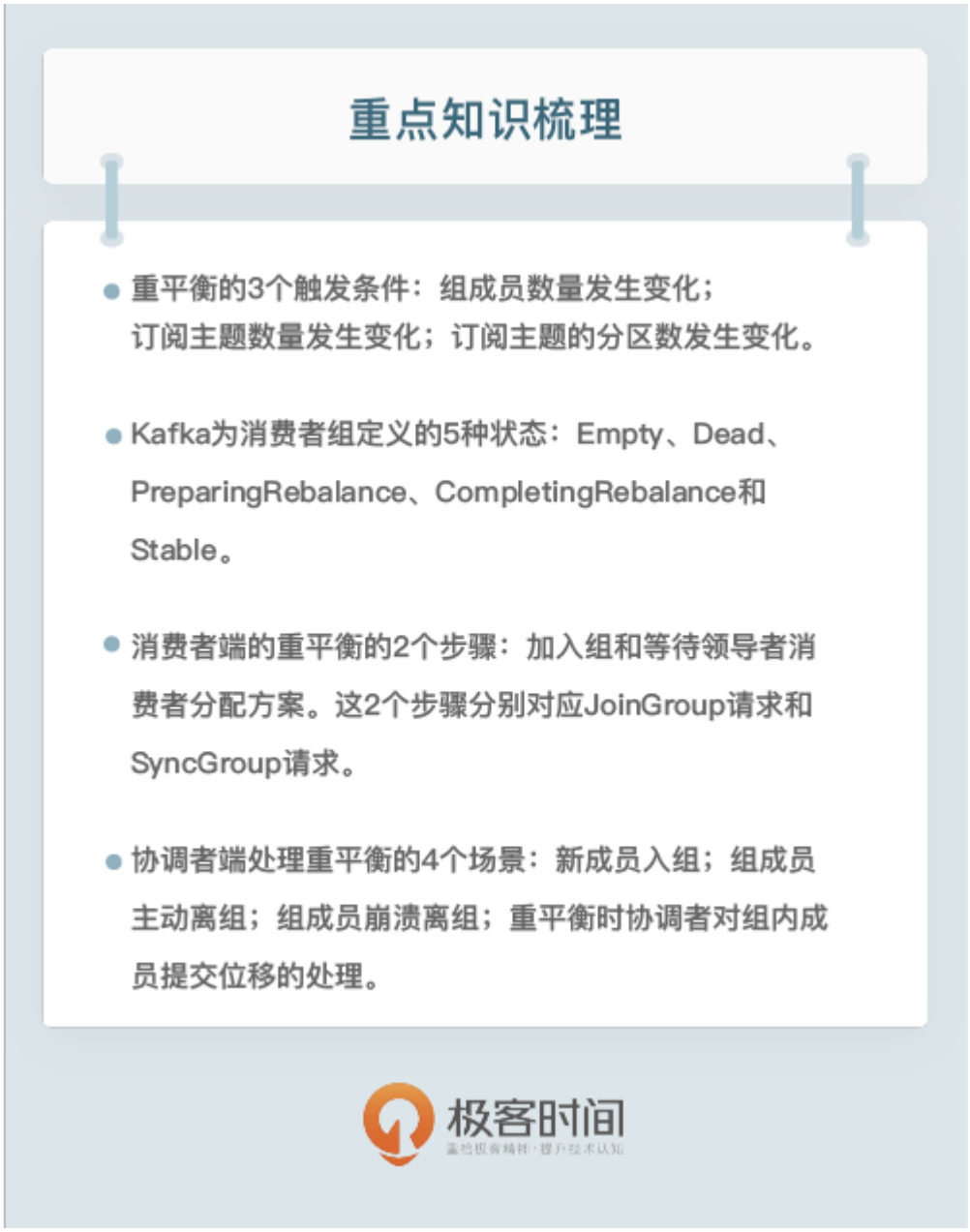
场景四：重平衡时协调者对组内成员提交位移的处理。

正常情况下，每个组内成员都会定期汇报位移给协调者。当重平衡开启时，协调者会给予成员一段缓冲时间，要求每个成员必须在这段时间内快速地上报自己的位移信息，然后再开启正常的JoinGroup/SyncGroup请求发送。还是老办法，我们使用一张图来说明。



小结

好了，消费者重平衡流程我已经全部讲完了。虽然全程我都是拿两个成员来举例子，但你可以很容易地扩展到多个成员的消费者组，毕竟它们的原理是相同的。我希望你能多看几遍今天的内容，彻底掌握Kafka的消费者重平衡流程。社区正在对目前的重平衡流程做较大程度的改动，如果你不了解这些基础的设计原理，后面想深入学习这部分内容的话，会十分困难。



开放讨论

在整个重平衡过程中，组内所有消费者实例都会暂停消费，用JVM GC的术语来说就是，重平衡过程是一个stop the world操作。请思考一下，针对这个问题，我们该如何改进这个过程？我们是否能允许部分消费者在重平衡过程中继续消费，以提升消费者端的可用性以及吞吐量？

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

cricket1981 2

SyncGroup请求处理流程图中怎么出现了JoinGroup请求？是不是笔误？另外，新成员入组流程图中成员2的SyncGroup请求不应该是“协调者你好，我是成员2，也是这个组的领导者，这是我的分配方案...”吗？

2019-07-30

作者回复

感谢纠正，已修改~~

2019-07-30

明翼 2

老师有两个问题请教下：

1）组状态在empty的时候，删除位移信息，这个时间间隔（文中7天）是否可以配置那，还是和普通的默认topic的消息存活时间一样吗？

2）这个设计我有点迷糊，都有协调者了为什么不让协调者统一做订阅分配那，让领导者做不是更麻烦吗？

2019-07-30

作者回复

1. 可以配置offsets.retention.minutes

2. 新版本consumer的一个改进就是把分区分配策略从server端移到consumer端来做。Client端代码演进的速度和容易程度要远胜于服务器端，算是一个优势吧

2019-07-30

ban 1

不会超过 session.timeout.ms 就能感知

老师，请问下，消费者已经崩溃了，不会发送心跳，协调者这时候怎么做到能到session.timeout.ms感知的。

2019-08-01

QQ怪 1

这一节学到了很多很多，开了视野，感谢

2019-07-30

Geek_edc612 1

胡老师有没有推荐的jvm 书籍？这块一直没有深入看过

2019-07-30

作者回复

我可不敢误人子弟：） 看看R大推荐的书单吧：https://www.douban.com/doulist/2545443/

2019-07-30

Stony.修行僧 1

有一个问题：Joingroup响应（成员2，你是这组的leader），Syncgroup请求（我是组员2，请求leader分配方案）。成员2已经是leader了，那么syncgroup的请求信息有点费解，明明已经是leader 还要请求leader分配方案？

2019-07-30

作者回复

至少这样能统一机制，因为目前非leader consumer依赖SyncGroup请求才能获取分配方案

2019-07-30

光辉 0

老师你好，
订阅主题数量发生变化是指什么？ 怎么才能触发这个场景的发生？

2019-08-01

ban 0

老师，这段话“但崩溃离组是被动的，协调者通常需要等待一段时间才能感知到，这段时间一般是由消费者端参数 session.timeout.ms 控制的。也就是说，Kafka 一般不会超过 session.timeout.ms 就能感知到这个崩溃。”

如果我的session.timeout.ms配置了10秒，是不是应该要10秒才能感知这个崩溃。但是我看你的原文“不会超过 session.timeout.ms 就能感知”，好像感觉是说10内就能感知，还没有超时应该不能告知到崩溃了把

2019-08-01

作者回复

最长不会超过10s，也有可能马上就能感知到

2019-08-01

rm -rf

0

老师，在Broker 端重平衡场景剖析这个第一个图里面，既然协调者说了成员2是这个组的leader，为啥成员2的SyncGroup请求会是”等待leader分配“？这是笔误吗？后面几幅图好像也这样。。。

2019-07-31

作者回复

只是想表明这是统一的一种机制。。。源代码中肯定没有这样的话。。。

2019-08-01

锦

0

我觉得协调者可以先自己重平衡，然后把结果同步给组成员，如果发现有些成员不在了，再把该成员的数据分配到其他成员

2019-07-31

雨夜听秋的孩子

0

老师，现需要内网生产消息，公网消费，配置advertise. listeners后生产消息有很长延时，可能是什么原因？

2019-07-31

wykkx

0

老师请假一个问题：客户端使用spring的组件写入kafka，很短的文本信息，但是报错如下：
WARN [SocketServer brokerId=0] Unexpected error from /192.168.x.x; closing connection (org.apache.kafka.common.network.Selector)
org.apache.kafka.common.network.InvalidReceiveException: Invalid receive (size = 1583156490 larger than 104857600)
不可能超过100M啊

2019-07-31

作者回复

通常可能是因为客户端、服务器端版本不匹配导致的，可以查看一下。

2019-08-01

金hb.Ryan 冷空氣駕到

0

如果始终有消费，那么过期的消息是不会清理的？这样么

2019-07-31

作者回复

没太懂。。。

2019-08-01

nightmare

0

重平衡组内位移提交的时候，一定要等到位移提交完成才能发生重平衡吗？ 我觉得是不是可以让分区分区比较多的消费者发生重平衡就行了，保持分区比较少或者压力较小的不发生重平衡

，比如在新的消费者加入消费组的时候

2019-07-30

Geek_25e177 0

胡老师，您好，我这里现在碰到这样一个问题，就是在kafka集群开启了认证后，我用客户端的AdminClient去获取所有主题client.listTopics()，需要大概20s左右才能返回结果，如果kafka未开启认证，则返回结果很快，想请教下老师，这有可能是什么原因造成的。

broker配置：
listeners=SASL_PLAINTEXT://host.name:port

security.inter.broker.protocol=SASL_PLAINTEXT

sasl.mechanism.inter.broker.protocol=SCRAM-SHA-512

sasl.enabled.mechanisms=SCRAM-SHA-512

acl

allow.everyone.if.no.acl.found=false

super.users=User:admin

authorizer.class.name=kafka.security.auth.SimpleAclAuthorizer

client代码：
client.listTopics().names().get()

kafka版本2.1.1

2019-07-30

作者回复

是在windows平台上吗？另外能否看下日志，看看SASL握手的时间是多少，主要是看看慢在哪里了？

2019-07-31

Li Shunduo 0

请问当重平衡开启时，协调者会给予提交位移的缓冲时间是多少？如果超过了会拒绝提交的位移吗？

2019-07-30

作者回复

没有具体的限制。反正如果consumer提交的位移请求到broker端时整个group已经从Preparing进化到Completing了，那么就晚了，broker会拒绝这个提交请求

2019-07-31

Li Shunduo

0

请问onPartitionsRevoked是在发送JoinGroup请求前触发的吗？
onPartitionsAssigned是在收到SyncGroup响应后触发的吗？

2019-07-30

作者回复

onPartitionsRevoked在发送JoinGroup之前；onPartitionsAssigned在收到SyncGroup之后

2019-07-30

许童童

0

我们是否能允许部分消费者在重平衡过程中继续消费，以提升消费者端的可用性以及吞吐量？我觉得可以选出领导者，让非领导者继续消费。领导者确定方案后，再让可用的非领领导者确定后的方案消费。

2019-07-30

南辕北辙

0

老师请教一下，在默认情况下，订阅主题的分区数量增加，会自动触发重平衡吗？

2019-07-30

作者回复

会的

2019-07-30

lmtoo

0

消费者新加入消费者组的时候，JoinGroup必须要携带订阅的主题信息吗？难道不是消费者组里的全部消费者消费一样的主题吗？
还有这个Coordinator组件是针对这个消费者组的，还是全部消费者组的，这个组件在哪个机器上？
控制消费者组位移数据删除时间的参数是哪个？

2019-07-30

作者回复

1. 消费者组里的消费者可能订阅不同的主题
2. 每个Broker上都有一个Coordinator组件，负责部分消费者组的协调工作
3. offsets.retention.minutes

2019-07-30