15 | 消费者组到底是什么？

2019-07-06 胡夕

你好，我是胡夕。今天我要和你分享的主题是：Kafka的消费者组。

消费者组，即Consumer Group，应该算是Kafka比较有亮点的设计了。那么何谓Consumer Group呢？用一句话概括就是：**Consumer Group是Kafka提供的可扩展且具有容错性的消费者机制**。既然是一个组，那么组内必然可以有多个消费者或消费者实例（Consumer Instance），它们共享一个公共的ID，这个ID被称为Group ID。组内的所有消费者协调在一起来消费订阅主题（Subscribed Topics）的所有分区（Partition）。当然，每个分区只能由同一个消费者组内的一个Consumer实例来消费。个人认为，理解Consumer Group记住下面这三个特性就好了。

1. Consumer Group下可以有一个或多个Consumer实例。这里的实例可以是一个单独的进程，也可以是同一进程下的线程。在实际场景中，使用进程更为常见一些。
2. Group ID是一个字符串，在一个Kafka集群中，它标识唯一的一个Consumer Group。
3. Consumer Group下所有实例订阅的主题的单个分区，只能分配给组内的某个Consumer实例消费。这个分区当然也可以被其他的Group消费。

你应该还记得我在专栏第1期中提到的两种消息引擎模型吧？它们分别是**点对点模型和发布/订阅模型**，前者也称为消费队列。当然，你要注意区分很多架构文章中涉及的消息队列与这里的消息队列。国内很多文章都习惯把消息中间件这类框架统称为消息队列，我在这里不评价这种提法是否准确，只是想提醒你注意这里所说的消息队列，特指经典的消息引擎模型。

好了，传统的消息引擎模型就是这两大类，它们各有优劣。我们来简单回顾一下。传统的消息队列模型的缺陷在于消息一旦被消费，就会从队列中被删除，而且只能被下游的一个Consumer消费。严格来说，这一点不算是缺陷，只能算是它的一个特性。但很显然，这种模型的伸缩性（scalability）很差，因为下游的多个Consumer都要“抢”这个共享消息队列的消息。发布/订阅模型倒是允许消息被多个Consumer消费，但它的问题也是伸缩性不高，因为每个订阅者都必须要订阅主题的所有分区。这种全量订阅的方式既不灵活，也会影响消息的真实投递效果。

如果有这么一种机制，既可以避开这两种模型的缺陷，又兼具它们的优点，那就太好了。幸运的是，Kafka的Consumer Group就是这样的机制。当Consumer Group订阅了多个主题后，组内的每个实例不要求一定要订阅主题的所有分区，它只会消费部分分区中的消息。

Consumer Group之间彼此独立，互不影响，它们能够订阅相同的一组主题而互不干涉。再加上Broker端的消息留存机制，Kafka的Consumer Group完美地规避了上面提到的伸缩性差的问题。可以这么说，**Kafka仅仅使用Consumer Group这一种机制，却同时实现了传统消息引擎系统的两大模型**：如果所有实例都属于同一个Group，那么它实现的就是消息队列模型；如果所有实例分别属于不同的Group，那么它实现的就是发布/订阅模型。

在了解了Consumer Group以及它的设计亮点之后，你可能会有这样的疑问：在实际使用场景中，我怎么知道一个Group下该有多少个Consumer实例呢？**理想情况下，Consumer实例的数量应该等于该Group订阅主题的分区总数。**

举个简单的例子，假设一个Consumer Group订阅了3个主题，分别是A、B、C，它们的分区数依次是1、2、3，那么通常情况下，为该Group设置6个Consumer实例是比较理想的情形，因为它能最大限度地实现高伸缩性。

你可能会问，我能设置小于或大于6的实例吗？当然可以！如果你有3个实例，那么平均下来每个实例大约消费2个分区（6 / 3 = 2）；如果你设置了8个实例，那么很遗憾，有2个实例（8 - 6 = 2）将不会被分配任何分区，它们永远处于空闲状态。因此，在实际使用过程中一般不推荐设置大于总分区数的Consumer实例。设置多余的实例只会浪费资源，而没有任何好处。

好了，说完了Consumer Group的设计特性，我们来讨论一个问题：针对Consumer Group，Kafka是怎么管理位移的呢？你还记得吧，消费者在消费的过程中需要记录自己消费了多少数据，即消费位置信息。在Kafka中，这个位置信息有个专门的术语：位移（Offset）。

看上去该Offset就是一个数值而已，其实对于Consumer Group而言，它是一组KV对，Key是分区，V对应Consumer消费该分区的最新位移。如果用Java来表示的话，你大致可以认为是这样的数据结构，即Map<TopicPartition, Long>，其中TopicPartition表示一个分区，而Long表示位移的类型。当然，我必须承认Kafka源码中并不是这样简单的数据结构，而是要比这个复杂得多，不过这并不会妨碍我们对Group位移的理解。

我在专栏第4期中提到过Kafka有新旧客户端API之分，那自然也就有新旧Consumer之分。老版本的Consumer也有消费者组的概念，它和我们目前讨论的Consumer Group在使用感上并没有太多的不同，只是它管理位移的方式和新版本是不一样的。

老版本的Consumer Group把位移保存在ZooKeeper中。Apache ZooKeeper是一个分布式的协调服务框架，Kafka重度依赖它实现各种各样的协调管理。将位移保存在ZooKeeper外部系统的做法，最显而易见的好处就是减少了Kafka Broker端的状态保存开销。现在比较流行的提法是将服务器节点做成无状态的，这样可以自由地扩缩容，实现超强的伸缩性。Kafka最开始也是基于这样的考虑，才将Consumer Group位移保存在独立于Kafka集群之外的框架中。

不过，慢慢地人们发现了一个问题，即ZooKeeper这类元框架其实并不适合进行频繁的写更新，而Consumer Group的位移更新却是一个非常频繁的操作。这种大吞吐量的写操作会极大地拖慢ZooKeeper集群的性能，因此Kafka社区渐渐有了这样的共识：将Consumer位移保存在ZooKeeper中是不合适的做法。

于是，在新版本的Consumer Group中，Kafka社区重新设计了Consumer Group的位移管理方式，采用了将位移保存在Kafka内部主题的方法。这个内部主题就是让人既爱又恨的__consumer_offsets。我会在专栏后面的内容中专门介绍这个神秘的主题。不过，现在你需要记住新版本的Consumer Group将位移保存在Broker端的内部主题中。

最后，我们来说说Consumer Group端大名鼎鼎的重平衡，也就是所谓的Rebalance过程。我形容其为“大名鼎鼎”，从某种程度上来说其实也是“臭名昭著”，因为有关它的bug真可谓是此起彼伏，从未间断。这里我先卖个关子，后面我会解释它“遭人恨”的地方。我们先来了解一下什么是Rebalance。

Rebalance本质上是一种协议，规定了一个Consumer Group下的所有Consumer如何达成一致，来分配订阅Topic的每个分区。比如某个Group下有20个Consumer实例，它订阅了一个具有100个分区的Topic。正常情况下，Kafka平均会为每个Consumer分配5个分区。这个分配的过程就叫Rebalance。

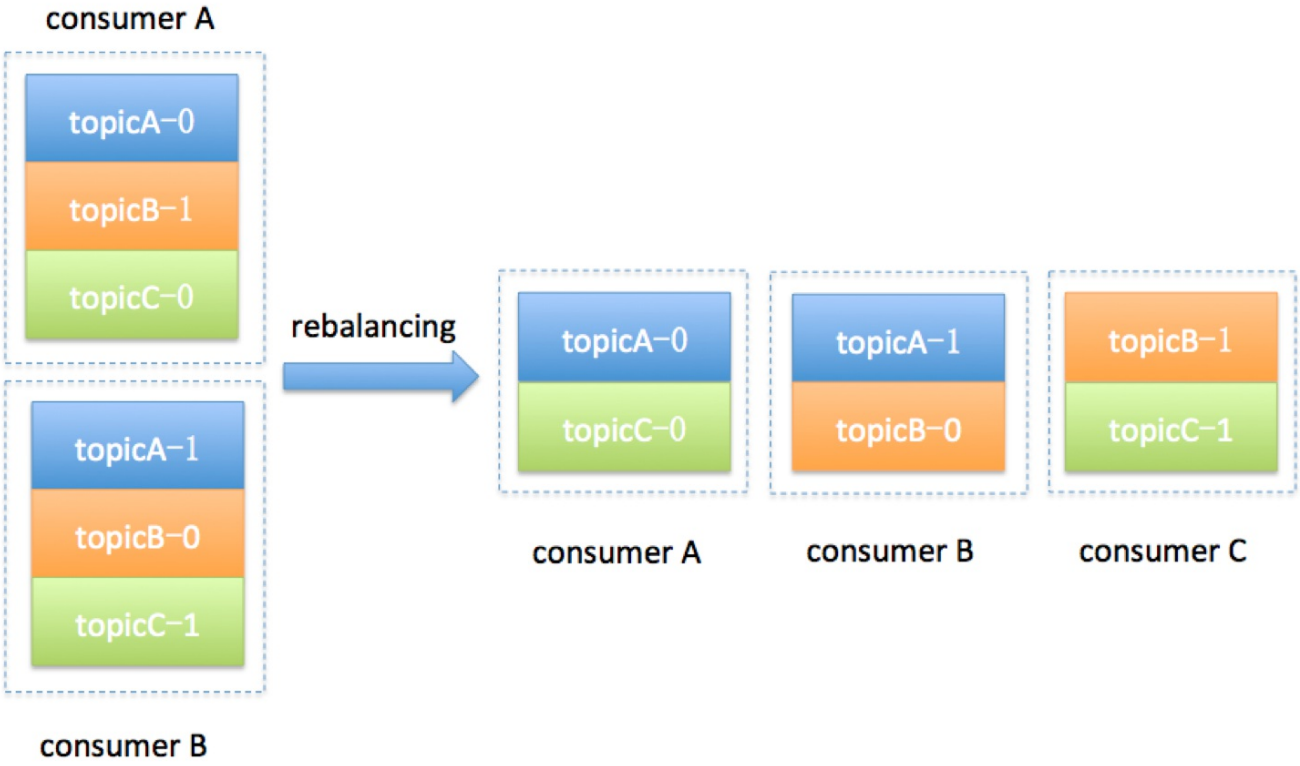
那么Consumer Group何时进行Rebalance呢？Rebalance的触发条件有3个。

1. 组成员数发生变更。比如有新的Consumer实例加入组或者离开组，抑或是有Consumer实例崩溃被“踢出”组。
2. 订阅主题数发生变更。Consumer Group可以使用正则表达式的方式订阅主题，比如consumer.subscribe(Pattern.compile(“t.*c”))就表明该Group订阅所有以字母t开头、字母c结尾的主题。在Consumer Group的运行过程中，你新创建了一个满足这样条件的主题，那么该Group就会发生Rebalance。
3. 订阅主题的分区数发生变更。Kafka当前只能允许增加一个主题的分区数。当分区数增加时，就会触发订阅该主题的所有Group开启Rebalance。

Rebalance发生时，Group下所有的Consumer实例都会协调在一起共同参与。你可能会问，每个Consumer实例怎么知道应该消费订阅主题的哪些分区呢？这就需要分配策略的协助了。

当前Kafka默认提供了3种分配策略，每种策略都有一定的优势和劣势，我们今天就不展开讨论了，你只需要记住社区会不断地完善这些策略，保证提供最公平的分配策略，即每个Consumer实例都能够得到较为平均的分区数。比如一个Group内有10个Consumer实例，要消费100个分区，理想的分配策略自然是每个实例平均得到10个分区。这就叫公平的分配策略。如果出现了严重的分配倾斜，势必会出现这种情况：有的实例会“闲死”，而有的实例则会“忙死”。

我们举个简单的例子来说明一下Consumer Group发生Rebalance的过程。假设目前某个Consumer Group下有两个Consumer，比如A和B，当第三个成员C加入时，Kafka会触发Rebalance，并根据默认的分配策略重新为A、B和C分配分区，如下图所示：



显然，Rebalance之后的分配依然是公平的，即每个Consumer实例都获得了3个分区的消费权。这是我们希望出现的情形。

讲完了Rebalance，现在我来说说它“遭人恨”的地方。

首先，Rebalance过程对Consumer Group消费过程有极大的影响。如果你了解JVM的垃圾回收机制，你一定听过万物静止的收集方式，即著名的stop the world，简称STW。在STW期间，所有应用线程都会停止工作，表现为整个应用程序僵在那边一动不动。Rebalance过程也和这个类似，在Rebalance过程中，所有Consumer实例都会停止消费，等待Rebalance完成。这是Rebalance为人诟病的一个方面。

其次，目前Rebalance的设计是所有Consumer实例共同参与，全部重新分配所有分区。其实更

高效的做法是尽量减少分配方案的变动。例如实例A之前负责消费分区1、2、3，那么Rebalance之后，如果可能的话，最好还是让实例A继续消费分区1、2、3，而不是被重新分配其他的分区。这样的话，实例A连接这些分区所在Broker的TCP连接就可以继续用，不用重新创建连接其他Broker的Socket资源。

最后，Rebalance实在是太慢了。曾经，有个国外用户的Group内有几百个Consumer实例，成功Rebalance一次要几个小时！这完全是不能忍受的。最悲剧的是，目前社区对此无能为力，至少现在还没有特别好的解决方案。所谓“本事大不如不摊上”，也许最好的解决方案就是避免Rebalance的发生吧。

小结

总结一下，今天我跟你分享了Kafka Consumer Group的方方面面，包括它是怎么定义的，它解决了哪些问题，有哪些特性。同时，我们也聊到了Consumer Group的位移管理以及著名的Rebalance过程。希望在你开发Consumer应用时，它们能够助你一臂之力。

开放讨论

今天我貌似说了很多Consumer Group的好话（除了Rebalance），你觉得这种消费者组设计的弊端有哪些呢？

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

QQ怪 5

老师最后那个Rebalance例子有问题吧，最后每个consumer只有两个而不是三个吧

2019-07-06

作者回复

嗯嗯，不就是每个consumer分配两个吗？

2019-07-08

nightmare 5

什么时候来个consumer端手动管理offset的方案

2019-07-06

东方奇骥 2

难得一个双休，今天终于学习到这篇了，好想实战上手玩一玩Kafka。老师，最后一个章节才会有实战Demo吗？

2019-07-06

October 2

消费组中的消费者个数如果超过topic的分区数，就会有消费者消费不到数据。但如果是同一个消费组里的两个消费者通过assign方法订阅了同一个TopicPartition，是不是会有一个消费者不能消费到消息？

2019-07-06

作者回复

如果使用assign，则表明该consumer是独立consumer（standalone consumer），它不属于任何消费者组。独立consumer可以订阅任何分区，彼此之间也没有关系，即两个独立consumer可以订阅并消费相同的分区

2019-07-08

Tony Du 2

请老师讲讲手动管理consumer offset的工程实践

2019-07-06

作者回复

专栏后面有专门的内容：）

2019-07-08

骨汤鸡蛋面 1

我们现在服务是3个topic，每个topic有64个分区，6个消费实例，每次服务重新部署（6个实例依次关闭启动）都会导致长时间的rebalance，是否减少topic的分区数可以减少服务部署时rebalance的时间呢？

2019-07-06

作者回复

嗯，会的。减少consumer个数也有缩短rebalance。

2019-07-08

Xiao 0

老师，rebalance的时候可以不可以这样做，如果是consumer挂掉导致，那不做group的rebalance，仅仅是将挂掉节点上的partition重新分配给别的consume让，只有在consumer消费特别不均匀的情况下才做group的rebalance；
如果是添加节点导致rebalance，那也不用一次性就做，可以分阶段，比如说先把消费压力大的consumer上的partition分一部分给新进来的consumer！

2019-07-08

Geek_jacky 0

老师好，多进程同一个消费组去消费同一个topic，会出现有的进程消费不到的问题。同时一个进程多个线程消费同一个topic是可以的，那这个多线程与多进程同消费组消费同一个topic这两个之间有什么不一样？

2019-07-08

外星人 0

那也就是说__consumer_offset这个topic和其他的topic做reassign的时候可以一样对待，没有特殊需要考虑或注意的地方是吗？consumer的groupcoordinator不是根据partition leader所在的broker来确定的吗？做reassign会不会影响消费端啊？

2019-07-08

曹操 0

请问一下老师，kafka有基于时间戳的消费方式吗？比如我想从某个时间点之后投入kafka的数据开始消费？

2019-07-08

老醋 0

不行吧，Kafka为了消除多个consumer消费同一分区进行的同步操作，组内不同consumer不能消费同一分区

2019-07-08

电光火石 0

如何避免rebalance发生？我发现线上在没有这三种情况也会发生，我猜是网络瞬断导致的，但不知道kafka是否会发生定时的rebalance？谢谢了

2019-07-08

作者回复

网络断了，心跳中断，consumer被踢出组，也属于第一种情况

2019-07-08

☆appleう 0

举个简单的例子，假设一个 Consumer Group 订阅了 3 个主题，分别是 A、B、C，它们的分区数依次是 1、2、3，那么通常情况下，为该 Group 设置 6 个 Consumer 实例是比较理想的情形，因为它能最大限度地实现高伸缩性。

你可能会问，我能设置小于或大于 6 的实例吗？当然可以！如果你有 3 个实例，那么平均下来每个实例大约消费 2 个分区（6 / 3 = 2）；如果你设置了 8 个实例，那么很遗憾，有 2 个实例（8 - 6 = 2）将不会被分配任何分区，它们永远处于空闲状态。因此，在实际使用过程中一般不推荐设置大于总分区数的 Consumer 实例。设置多余的实例只会浪费资源，而没有任何好处。

老师，针对上面这段例子有一个问题: 如果8个实例，6个分区，每个实例负责分配一个分区，多出来的2个实例不能与其他消费实例共同负责一个分区吗？

2019-07-07

作者回复

如果它们都属于同一个消费者组，那么不能。消费者组订阅的分区只能被组内一个consumer实例消费。

2019-07-08

蒙开强 0

老师，你好，怎么设置消费组里的消费实例个数呢

2019-07-07

作者回复

没有一个参数来设置此事。主要看启动了多少个具有相同group.id的consumer实例。

2019-07-08

外星人 0

老师，你好，我们最近集群扩容，需要对__consumer_offset这个topic进行reassign，请问有没有坑啊？需要注意哪些事项呢？期待您的回答，谢谢

2019-07-07

作者回复

reassign操作很容易出错，不只是对__consumer_offsets。我个人的建议哈：1. 业务低峰时段做；2. 不要topic级别整体迁移，最好按照分区级别来做。比如一次迁移几个分区这样

2019-07-08

October 0

一个TopicPatition的消息只能被消费组的某个消费者所消费，当出现数据倾斜时（比如某个key的消息的特别多就会导致数据倾斜），可能一个消费者的的消费速度不足以应付业务对消息处理延迟的要求，不知道这算不算一个缺点。请老师指正。

2019-07-06

作者回复

嗯。如果单个分区依然数据很多，可以考虑多加一些分区，将数据分散到更多的分区上，降低单个分区的负载。

2019-07-08

风中花 0

看的我都stop the word！一口气看完！看觉一下子吸收了不少功法套路！

2019-07-06

wykkx

0

“在新版本的 Consumer Group 中，Kafka 社区重新设计了 Consumer Group 的位移管理方式”
老师 请问这是在哪个版本以后算是你 说的新版本，这里要如何进行配置是使用内部的还是使用外部的zk集群，在新版本里。

2019-07-06

作者回复

0.9版本引入了新版本consumer。新版本将位移保存在__consumer_offsets中，不需要额外配置。如果你想保存在外部的Zk集群中，那么设置consumer端参数enable.auto.commit=false，然后自己调用ZooKeeper API提交位移到Zk即可。

2019-07-08