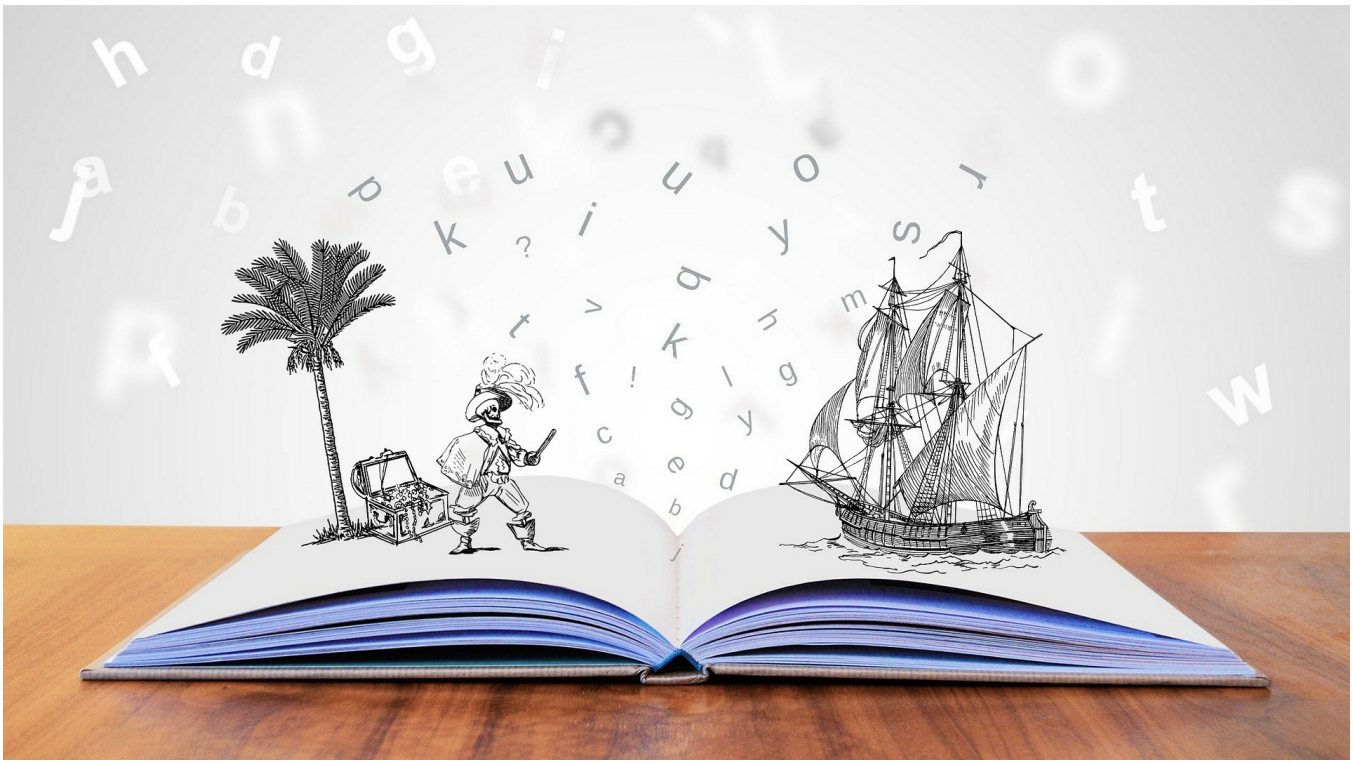
16 | 揭开神秘的“位移主题”面纱

2019-07-09 胡夕



你好，我是胡夕。今天我要和你分享的内容是：Kafka中神秘的内部主题（Internal Topic）__consumer_offsets。

__consumer_offsets在Kafka源码中有个更为正式的名字，叫**位移主题**，即Offsets Topic。为了方便今天的讨论，我将统一使用位移主题来指代__consumer_offsets。需要注意的是，它有两个下划线哦。

好了，我们开始今天的内容吧。首先，我们有必要探究一下位移主题被引入的背景及原因，即位移主题的前世今生。

在上一期中，我说过老版本Consumer的位移管理是依托于Apache ZooKeeper的，它会自动或手动地将位移数据提交到ZooKeeper中保存。当Consumer重启后，它能自动从ZooKeeper中读取位移数据，从而在上次消费截止的地方继续消费。这种设计使得Kafka Broker不需要保存位移数据，减少了Broker端需要持有的状态空间，因而有利于实现高伸缩性。

但是，ZooKeeper其实并不适用于这种高频的写操作，因此，Kafka社区自0.8.2.x版本开始，就在酝酿修改这种设计，并最终在新版本Consumer中正式推出了全新的位移管理机制，自然也包括这个新的位移主题。

新版本Consumer的位移管理机制其实也很简单，就是**将Consumer的位移数据作为一条条普通的Kafka消息，提交到__consumer_offsets中。可以这么说，__consumer_offsets的**

主要作用是保存Kafka消费者的位移信息。它要求这个提交过程不仅要实现高持久性，还要支持高频的写操作。显然，Kafka的主题设计天然就满足这两个条件，因此，使用Kafka主题来保存位移这件事情，实际上就是一个水到渠成的想法了。

这里我想再次强调一下，和你创建的其他主题一样，位移主题就是普通的Kafka主题。你可以手动地创建它、修改它，甚至是删除它。只不过，它同时也是一个内部主题，大部分情况下，你其实并不需要“搭理”它，也不用花心思去管理它，把它丢给Kafka就完事了。

虽说位移主题是一个普通的Kafka主题，但**它的消息格式却是Kafka自己定义的**，用户不能修改，也就是说你不能随意地向这个主题写消息，因为一旦你写入的消息不满足Kafka规定的格式，那么Kafka内部无法成功解析，就会造成Broker的崩溃。事实上，Kafka Consumer有API帮你提交位移，也就是向位移主题写消息。你千万不要自己写个Producer随意向该主题发送消息。

你可能会好奇，这个主题存的到底是什么格式的消息呢？所谓的消息格式，你可以简单地理解为是一个KV对。Key和Value分别表示消息的键值和消息体，在Kafka中它们就是字节数组而已。想象一下，如果让你来设计这个主题，你觉得消息格式应该长什么样子呢？我先不说社区的设计方案，我们自己先来设计一下。

首先从Key说起。一个Kafka集群中的Consumer数量会有很多，既然这个主题保存的是Consumer的位移数据，那么消息格式中必须要有字段来标识这个位移数据是哪个Consumer的。这种数据放在哪个字段比较合适呢？显然放在Key中比较合适。

现在我们知道该主题消息的Key中应该保存标识Consumer的字段，那么，当前Kafka中什么字段能够标识Consumer呢？还记得之前我们说Consumer Group时提到的Group ID吗？没错，就是这个字段，它能够标识唯一的Consumer Group。

说到这里，我再多说几句。除了Consumer Group，Kafka还支持独立Consumer，也称Standalone Consumer。它的运行机制与Consumer Group完全不同，但是位移管理的机制却是相同的。因此，即使是Standalone Consumer，也有自己的Group ID来标识它自己，所以也适用于这套消息格式。

Okay，我们现在知道Key中保存了Group ID，但是只保存Group ID就可以了吗？别忘了，Consumer提交位移是在分区层面上进行的，即它提交的是某个或某些分区的位移，那么很显然，Key中还应该保存Consumer要提交位移的分区。

好了，我们来总结一下我们的结论。**位移主题的Key中应该保存3部分内容：<Group ID，主题名，分区号>**。如果你认同这样的结论，那么恭喜你，社区就是这么设计的！

接下来，我们再来看看消息体的设计。也许你会觉得消息体应该很简单，保存一个位移值就可以了。实际上，社区的方案要复杂得多，比如消息体还保存了位移提交的一些其他元数据，诸如时间戳和用户自定义的数据等。保存这些元数据是为了帮助Kafka执行各种各样后续的操作，比如

删除过期位移消息等。但总体来说，我们还是可以简单地认为消息体就是保存了位移值。

当然了，位移主题的消息格式可不是只有这一种。事实上，它有3种消息格式。除了刚刚我们说的这种格式，还有2种格式：

1. 用于保存Consumer Group信息的消息。
2. 用于删除Group过期位移甚至是删除Group的消息。

第1种格式非常神秘，以至于你几乎无法在搜索引擎中搜到它的身影。不过，你只需要记住它是用来注册Consumer Group的就可以了。

第2种格式相对更加有名一些。它有个专属的名字：tombstone消息，即墓碑消息，也称delete mark。下次你在Google或百度中见到这些词，不用感到惊讶，它们指的是一个东西。这些消息只出现在源码中而不暴露给你。它的主要特点是它的消息体是null，即空消息体。

那么，何时会写入这类消息呢？一旦某个Consumer Group下的所有Consumer实例都停止了，而且它们的位移数据都已被删除时，Kafka会向位移主题的对应分区写入tombstone消息，表明要彻底删除这个Group的信息。

好了，消息格式就说这么多，下面我们来说说位移主题是怎么被创建的。通常来说，**当Kafka集群中的第一个Consumer程序启动时，Kafka会自动创建位移主题**。我们说过，位移主题就是普通的Kafka主题，那么它自然也有对应的分区数。但如果是Kafka自动创建的，分区数是怎么设置的呢？这就要看Broker端参数offsets.topic.num.partitions的取值了。它的默认值是50，因此Kafka会自动创建一个50分区的位移主题。如果你曾经惊讶于Kafka日志路径下冒出很多__consumer_offsets-xxx这样的目录，那么现在应该明白了吧，这就是Kafka自动帮你创建的位移主题啊。

你可能会问，除了分区数，副本数或备份因子是怎么控制的呢？答案也很简单，这就是Broker端另一个参数offsets.topic.replication.factor要做的事情了。它的默认值是3。

总结一下，**如果位移主题是Kafka自动创建的，那么该主题的分区数是50，副本数是3**。

当然，你也可以选择手动创建位移主题，具体方法就是，在Kafka集群尚未启动任何Consumer之前，使用Kafka API创建它。手动创建的好处在于，你可以创建满足你实际场景需要的位移主题。比如很多人说50个分区对我来讲太多了，我不想要这么多分区，那么你可以自己创建它，不用理会offsets.topic.num.partitions的值。

不过我给你的建议是，还是让Kafka自动创建比较好。目前Kafka源码中有一些地方硬编码了50分区数，因此如果你自行创建了一个不同于默认分区数的位移主题，可能会碰到各种各种奇怪的问题。这是社区的一个bug，目前代码已经修复了，但依然在审核中。

创建位移主题当然是为了用的，那么什么地方会用到位移主题呢？我们前面一直在说Kafka

Consumer提交位移时会写入该主题，那Consumer是怎么提交位移的呢？目前Kafka Consumer提交位移的方式有两种：**自动提交位移和手动提交位移。**

Consumer端有个参数叫enable.auto.commit，如果值是true，则Consumer在后台默默地为你定期提交位移，提交间隔由一个专属的参数auto.commit.interval.ms来控制。自动提交位移有一个显著的优点，就是省事，你不用操心位移提交的事情，就能保证消息消费不会丢失。但这一点同时也是缺点。因为它太省事了，以至于丧失了很大的灵活性和可控性，你完全没法把控Consumer端的位移管理。

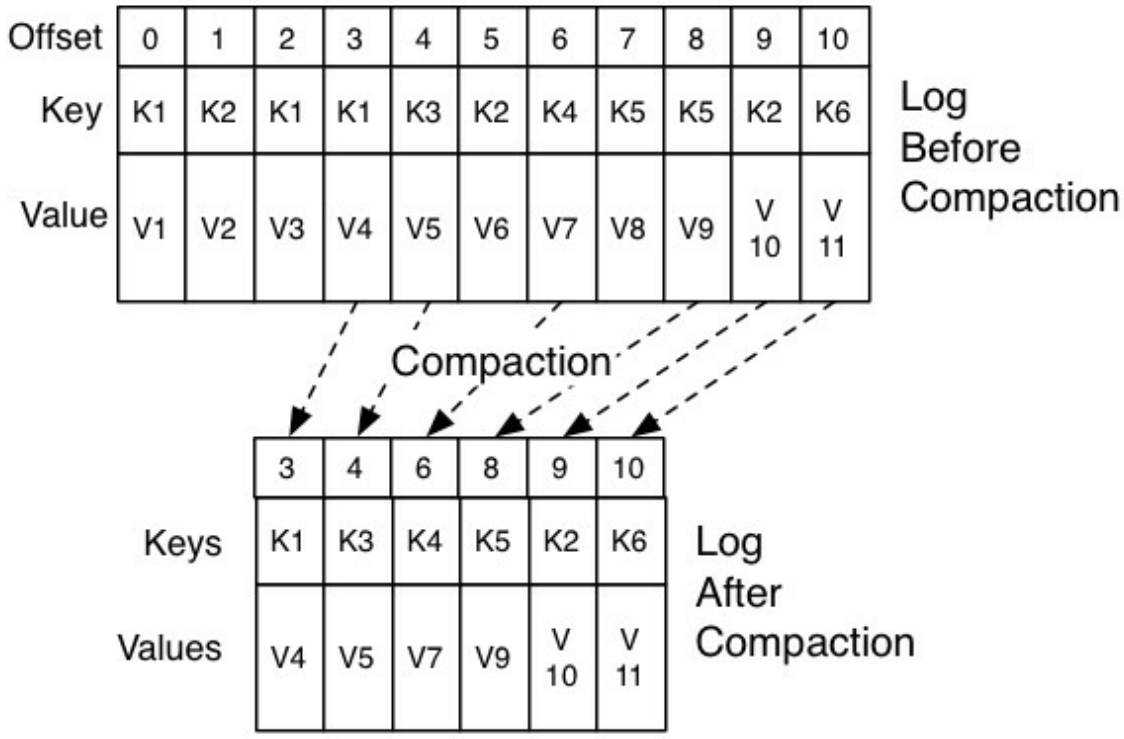
事实上，很多与Kafka集成的大数据框架都是禁用自动提交位移的，如Spark、Flink等。这就引出了另一种位移提交方式：**手动提交位移**，即设置enable.auto.commit = false。一旦设置了false，作为Consumer应用开发的你就要承担起位移提交的责任。Kafka Consumer API为你提供了位移提交的方法，如consumer.commitSync等。当调用这些方法时，Kafka会向位移主题写入相应的消息。

如果你选择的是自动提交位移，那么就可能存在一个问题：只要Consumer一直启动着，它就会无限期地向位移主题写入消息。

我们来举个极端一点的例子。假设Consumer当前消费到了某个主题的最新一条消息，位移是100，之后该主题没有任何新消息产生，故Consumer无消息可消费了，所以位移永远保持在100。由于是自动提交位移，位移主题中会不停地写入位移=100的消息。显然Kafka只需要保留这类消息中的最新一条就可以了，之前的消息都是可以删除的。这就要求Kafka必须要有针对位移主题消息特点的消息删除策略，否则这种消息会越来越多，最终撑爆整个磁盘。

Kafka是怎么删除位移主题中的过期消息的呢？答案就是Compaction。国内很多文献都将其翻译成压缩，我个人是有一点保留意见的。在英语中，压缩的专有术语是Compression，它的原理和Compaction很不相同，我更倾向于翻译成压实，或干脆采用JVM垃圾回收中的术语：整理。

不管怎么翻译，Kafka使用**Compact策略**来删除位移主题中的过期消息，避免该主题无限期膨胀。那么应该如何定义Compact策略中的过期呢？对于同一个Key的两条消息M1和M2，如果M1的发送时间早于M2，那么M1就是过期消息。Compact的过程就是扫描日志的所有消息，剔除那些过期的消息，然后把剩下的消息整理在一起。我在这里贴一张来自官网的图片，来说明Compact过程。



图中位移为0、2和3的消息的Key都是K1。Compact之后，分区只需要保存位移为3的消息，因为它是最新发送的。

Kafka提供了专门的后台线程定期地巡检待Compact的主题，看看是否存在满足条件的可删除数据。这个后台线程叫Log Cleaner。很多实际生产环境中都出现过位移主题无限膨胀占用过多磁盘空间的问题，如果你的环境中也有这个问题，我建议你去检查一下Log Cleaner线程的状态，通常都是这个线程挂掉了导致的。

小结

总结一下，今天我跟你分享了Kafka神秘的位移主题__consumer_offsets，包括引入它的契机与原因、它的作用、消息格式、写入的时机以及管理策略等，这对我们了解Kafka特别是Kafka Consumer的位移管理是大有帮助的。实际上，将很多元数据以消息的方式存入Kafka内部主题的做法越来越流行。除了Consumer位移管理，Kafka事务也是利用了这个方法，当然那是另外的一个内部主题了。

社区的想法很简单：既然Kafka天然实现了高持久性和高吞吐量，那么任何有这两个需求的子服务自然也就不必求助于外部系统，用Kafka自己实现就好了。

开放讨论

今天我们说了位移主题的很多好处，请思考一下，与ZooKeeper方案相比，它可能的劣势是什么？

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

海贼王 2

相对于zookeeper方案可能的劣势是，kafka得保证offset topic的读写都是线性一致的。

但是有个疑惑，如果kafka自己实现了类似zab协议的话，那写性能会比zk高吗

2019-07-09

作者回复

hmm.... 哈哈哈，不知道。。。。

2019-07-10

mellow 2

老师能讲一下，同一个group下的consumer启动之后是怎么去offset topic 拿到该group上次消费topic每个partition的最新offset呢？是根据key来定位offset topic的partition吗，然后拿到所有消息得到最新的offset吗

2019-07-09

作者回复

它会去寻找其Coordinator Leader副本对应的broker去拿。根据group.id找到对应Coordinator的分区数

2019-07-10

nightmare 2

比如多个线程同时消费一个分区的话，位移这么处理

2019-07-09

yyyiue

2

请问offset是以最新的为准，还是值最大的为准？

2019-07-09

作者回复

最新的

2019-07-09

ban

1

老师，你说偏移量都保存到了__consumer_offsets，确实看到生成了，但是为什么我在zk还是能看到偏移量信息，记录在
consumers/{group}/offsets/{topic}/{partition}。
这两个有什么区别吗，还是说zk会记录还是因为我们的程序手动改成提交到zk里面了。

2019-07-10

作者回复

如果使用使用老版本consumer，还是会记录在ZooKeeper中的

2019-07-11

玉剑冰锋

1

不好意思老师，地铁上有点匆忙提问的问题没有描述清楚，我想的问的是1.文章提到__consumer_offset这个topic记录的offset信息和zk中记录的offset信息有啥区别？
2.__consumer_offset这个topic我线上环境没有副本，文章提到副本默认是3，是我配置的问题吗？发现这个问题以后我就在线添加副本，过去好几天了仍然有十几个in progress，另外有没有办法强制终止掉这个任务重新添加副本？

2019-07-10

作者回复

1. 没什么区别。你可以简单认为就是换个地方保存：）

2. 这个其实是个bug，不过现在已经修复了。如果reassign hang住了，手动删除Zk下对应的znode就能恢复

2019-07-11

nico

1

大神 Kafka可以发送 定时消息吗，制定某一时刻接受到消息 拜托

2019-07-09

作者回复

需要自己写代码实现，Kafka没有天然提供这个功能

2019-07-10

given

0

consumer端 日常业务发版呢，那每次发版需要重启consumer不是也会导致Rebalance，这个如何规避

2019-07-15

作者回复

可以考虑使用standalone consumer，否则group机制无法避免

2019-07-15

wykkx

0

老师我想请教几个问题，

1，具体是从哪个版本开始，位移数据开始默认的不存在zk而是存在自己内部了？

2，如果验证目前使用的环境，位移是存在内部还是zk上？

3，什么场景下适合使用自动提交位移？

2019-07-13

作者回复

1. 0.9

2. 推荐保存在Kafka内部

3. 不在乎重复消费

2019-07-15

子华

0

你好，请教下。发现线上offsets.topic.replication.factor=1 然后通过官网教程 https://kafka.apache.org/documentation/#basic_ops_increase_replication_factor 在线调整了位移主题的分区数3，也反馈成功了。但是还是不知道 这种调整方式 对于 __consumer_offsets 这种特殊主题是否有效。

因为发现 比如 __consumer_offsets-46 这个主题，之前leader的文件有

-rw-rw-r-- 1 www www 0 7月 10 14:59 00000000000000000000.index

-rw-rw-r-- 1 www www 2620 7月 10 14:59 00000000000000000000.log

-rw-rw-r-- 1 www www 12 7月 10 14:59 00000000000000000000.timeindex

-rw-rw-r-- 1 www www 10 6月 18 18:02 00000000000000000189.snapshot

-rw-rw-r-- 1 www www 10 6月 20 17:59 00000000000000000203.snapshot

-rw-rw-r-- 1 www www 10 7月 3 15:00 00000000000000000224.snapshot

-rw-rw-r-- 1 www www 10485760 7月 12 18:11 00000000000000065983.index

-rw-rw-r-- 1 www www 3091267 7月 12 18:11 00000000000000065983.log

-rw-rw-r-- 1 www www 10 7月 10 15:00 00000000000000065983.snapshot

-rw-rw-r-- 1 www www 10485756 7月 12 18:11 00000000000000065983.timeindex

-rw-rw-r-- 1 www www 28 7月 12 17:14 leader-epoch-checkpoint

但是同步后两个副本文件大概如下

-rw-rw-r-- 1 www www 10485760 7月 12 18:09 00000000000000000000.index

-rw-rw-r-- 1 www www 3093887 7月 12 18:11 00000000000000000000.log

-rw-rw-r-- 1 www www 10485756 7月 12 18:09 00000000000000000000.timeindex

-rw-rw-r-- 1 www www 23 7月 12 17:14 leader-epoch-checkpoint

很明显同步后的副本没有00000000000000065983.index这个文件，但是通过describle 命令看都是lsr状态

Topic: __consumer_offsets Partition: 46 Leader: 0 Replicas: 0,1,2 Isr: 0,2,1

不知道这种状态是否OK?

2019-07-12

酱排骨 0

老师，我想问一下工作中的一个问题单个消费实例，单个partion，消费者消费失败，offset可以重新回到前面位置重新消费吗?

2019-07-11

作者回复

消费者重启回来后会从最新一次提交的位移处继续消费

2019-07-12

Coder4 0

老师好，前几年一直有个说法，说kafka不适合创建过多topic，请问现在的新版还有这个问题么?

2019-07-11

作者回复

topic过多其实是指分区数过多。会有两个可能的问题：1. controller无法管理这么多分区；2. 分区数过多导致broker物理随机IO增加，减少吞吐量。

第一个问题社区算是修复了吧，目前单controller能够支持20w的分区数，况且社区也在考虑做多controller方案；第二个问题目前没有太多直接的修复举措，只能说具体问题具体分析吧

2019-07-12

永光 0

位移主题，适用于高频写的操作，为什么ZooKeeper不适用于这种高频的写操作？zookeeper也可以按照<Group ID，主题名，分区号 > 来写入呀?

2019-07-11

作者回复

ZooKeeper本身只是一个分布式协调框架，znode中保存的数据多是那些不怎么频繁修改的元数据，本身不适合频繁更新。

是的，旧版本consumer就是这么使用ZooKeeper来保存位移的

2019-07-12

光辉 0

老师，你好

Kafka有位移主题，是不是Consumer都是从整个位移主题获取数据应该从哪个offset开始读数据，如果Spark Streaming作为一个Consumer，其offset的控制也是在这个位移主题当中？ 这样的话Spark Streaming用Direct方式读取Kafka其实是不需要额外找其他存储作为位移的保存?

2019-07-11

作者回复

Spark Streaming集成Kafka的位移管理有三种：1. 寿司用checkpoint，也就是保存在Spark内部；2. 保存在Kafka内部主题；3. 保存在外部存储中。你可以选择合适的方式。

2019-07-12

MoonGod 0

请教老师一个问题：

在consumer提交位移的时候，通过Coordinator 往所在的broker写消息，那如果当前的broker挂掉了，写入位移主题的消息会丢失吗？还是说位移主题在写入的时候也会把消息同步到其他broker中的副本中，从而保证写入消息不丢失呢

2019-07-11

作者回复

不会。位移主题的failover和高可用管理和普通Kafka topic是一样的。也会执行leader选举。

2019-07-12

MoonGod 0

请教老师一个问题：

在consumer提交位移的时候，通过Coordinator 往所在的broker写消息，那如果写入的broker挂掉了，写入位移主题的消息会丢失吗？还是说位移主题在写入的时候也会把消息同步到其他broker中的副本中，从而保证写入消息不丢失呢

2019-07-11

无菇朋友 0

老师，offset topic是在coordinator对应的broker上创建且只创建一次是么？

2019-07-10

作者回复

offset topic在整个集群上被创建出来，并且只会创建一次

2019-07-11

南辕北辙 0

老师，请教一下consumer 是如何从这个位移主题中拿到曾经属于自己组的offset呢

2019-07-10

作者回复

首先找到对应的Coordinator，Coordinator保存了这些数据，然后consumer向Coordinator发送请求去请求这些数据

2019-07-10

玉剑冰锋 0

现在kafka中同样还在使用zk,每个主题也会记录位移，跟主题位移有啥区别？另外我线上主题位置没有副本，之前从来没动过，这是啥原因导致的？

2019-07-10

作者回复

“每个主题也会记录位移，跟主题位移有啥区别”这是什么意思呢？没看懂。。。。

“我线上主题位置没有副本” 这又是什么意思呢。。。

2019-07-10

dream

0

老师，你最后说“Kafka 提供了专门的后台线程定期地巡检待 Compact 的主题”，那么这个定期是好久呢？这个间隔时间我们可以自己配置吗？

2019-07-09

作者回复

没有相关配置。我的“定期”的意思是有个专门的线程不断地去做这件事。。。。

2019-07-10