前言

你可能听说过高水位，但不一定听说过Leader Epoch。前者是Kafka中非常重要的概念。而后者是0.11版本中新推出的。主要是为了弥补前者水位机制的一些缺陷。

1.高水位

1.1 什么是高水位

Kafka的水位不是时间戳，更与时间无关。它是和位置信息绑定的，具体来说，它是用消息位移来表征的。

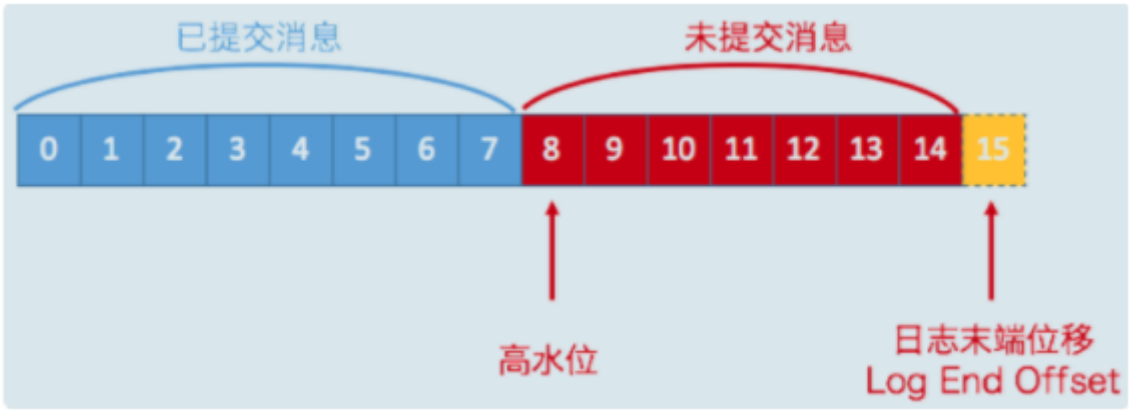
这个offset是所有ISR的LEO的最小位置（minimum LEO across all the ISR of this partition），consumer不能读取超过HW的消息，因为这意味着读取到未完全同步（因此没有完全备份）的消息。换句话说就是：HW是所有ISR中的节点都已经复制完的消息.也是消费者所能获取到的消息的最大offset（注意，并不是所有replica都一定有这些消息，而只是ISR里的那些才肯定会有）。

1.2 高水位的作用

- 定义消息可见性，用来标识分区下的哪些消息是可以被消费者消费的

- 帮助kafka完成副本同步

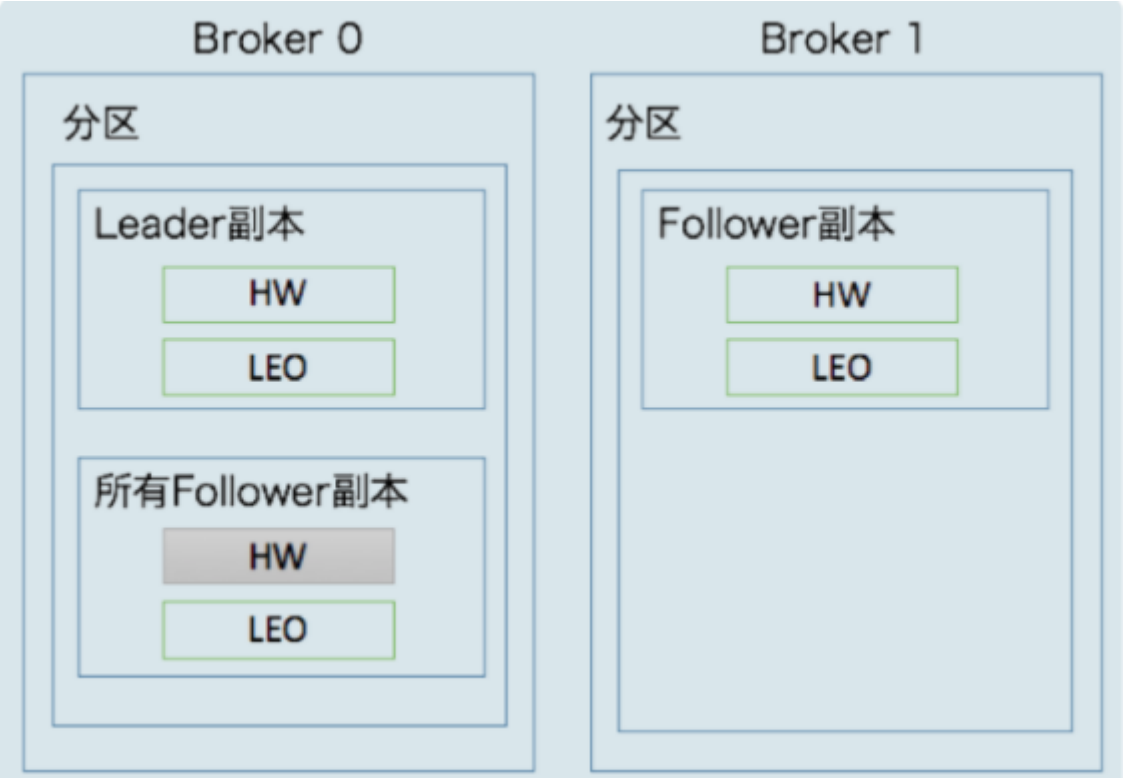
1.3 已提交消息和未提交消息



- 在分区高水位以下的消息就被认为是已提交消息，反之就是未提交消息
- 消费者只能消费已提交消息，即位移值小于8的消息。
- 这里不存在kafka的事务，因为事务机制会影响消息者所能看到的消息的范围，他不只是简单依赖高水位来判断，是依赖于一个名为LSO的位移值来判断事务性消费者的可见性
- 位移值等于高水位的消息也属于为未提交消息。即高水位的消息也是不能被消费者消费的
- LEO表示副本写入下一条消息的位移值。同一个副本对象，起高水位值不会超过LEO

1.4 高水位更新机制

Kafka中所有副本对象都保存一组高水位值和LEO值，但Leader副本中还保留着其他Follower副本的LEO值。



> Kafka副本机制在运行过程中，会更新Broker1上Follower副本的高水位和LEO值，同时也会更新Broker0上Leader副本的高水位和LEO以及Follow副本的LEO，但不会更新其HW。

1.5 副本同步机制解析

当生产者发送一条消息时，Leader和Follower副本对应的高水位是怎么被更新的呢?

> Follower副本也成功地更新LEO为1.此时，Leader和Follower副本的LEO都是1，但各自的高水位依然是0，还没有被更新。**他们需要在下一轮的拉取中被更新**
> 在新一轮的拉去请求中，由于位移值是0的消息已经拉取成功，因此Follower副本这次请求拉去的位移值为1的消息。Leader副本接收此请求后，更新远程副本LEO为1，然后更新Leader高水位为1，然后才会将更新过的高水位值1发送给Follower副本。Follower副本接收到以后，也将自己的高水位值更新为1.至此，一个完整的消息同步周期就结束了。
>
> 总的来说: 第一次拉取只会更新LEO,第二次拉取时才会更新HW

2. Leader Epoch

Follower副本的高水位更新是需要额外一轮的拉取请求才能实现的。若有多个副本的情况下，则需要多轮的拉取请求。也就是说，Leader副本高水位更新和Follower副本高水位更

新在时间上是存在错配的。而这种错配往往是数据丢失，数据不一致问题现象的根源。因此kafka社区在0.11版本中引入了Leader Epoch。

2.1 Leader Epoch的组成

- Epoch。一个单调递增的版本号。每当副本领导权发生变更时，都会增加该版本号。小版本号的Leader被认为是过期的Leader，不能再行使Leader的权力。
- 起始位移（Start Offset）。Leader副本在该Epoch上写入的首条消息的位移。

> Leader Epoch<0,0>和<1,100>。第一个Epoch指的是0版本，位移0开始保存消息，一共保存100条消息。之后Leader发生了变更，版本号增加到1，新版本起始位移为100.

Kafka Broker会在内存中为每个分区都缓存Leader Epoch数据，同时它还会定期的将这信息持久化一个checkpoint文件中。当Leader副本写入消息到磁盘时，Broker会尝试更新这部分缓存，如果该Leader是首次写入消息，那么Broker会向缓存中增加一个Leader Epoch条目，否则就不做更新。

2.2 Leader Epoch使用

Leader Epoch是怎样防止数据丢失的呢?

单纯依赖高水位是怎么造成数据丢失的。开始时，副本A和副本B都处于正常状态，A是Leader副本，B是Follower副本。当生产者使用ack=1（默认）往Leader副本A中发送两条消息。且A全部写入成功，此时Kafka会通知生产者说这两条消息写入成功。

现在假设A,B都写入了这两条消息，而且Leader副本的高水位也已经更新了，但Follower副本高水位还未更新。因为Follower端高水位的更新与Leader端有时间错配。假如现在副本B所在Broker宕机了，那么当它重启回来后，副本B就会执行日志截断操作，将LEO值调整为之前的高水位值，也就是1.所以副本B当中位移值为1的消息就丢失了。副本B中只保留了位移值0的消息。

当执行完截断操作之后，副本B开始从A中拉取消息，执行正常的消息同步。假如此时副本A所在的Broker也宕机了。那么kafka只能让副本B成为新的Leader，然后副本A重启回来之后，也需要执行日志截断操作，即调整高

水位为与B相同的值，也就是1。这样操作之后，位移值为1的那条消息就永远丢失了。

Leader Epoch机制如何规避这种数据丢失现象呢?

延续上文场景，引用了Leader Epoch机制之后，Follower副本B重启回来后，需要向A发送一个特殊的请求去获取Leader的LEO值，该例子中为2。当知道Leader LEO为2时，B发现该LEO值不必自己的LEO值小，而且缓存中也没有保存任何起始位移值>2的Epoch条目，因此B无需执行日志截断操作。这是对高水位机制的一次明显改进，即不是依赖于高水位判断是否进行日志截断操作。

现在，副本A宕机了，B成立新Leader。同样的，在A重启回来后，执行与B逻辑相同的判断，也不需要执行日志截断操作，所以位移值为1的那条消息就全部得以保存。后面当生产者程序向 B 写入新消息时，副本 B 所在的 Broker 缓存中，会生成新的 Leader Epoch 条目：[Epoch=1, Offset=2]。之后，副本 B 会使用这个条目帮助判断后续是否执行日志截断操作。这样，kafka就规避掉了数据丢失的场景。

> https://copyfuture.com/blogs-details/20200603094538911120r9ar5i3bhw0px
> https://www.cnblogs.com/huxi2b/p/7453543.html
> https://mp.weixin.qq.com/s/17b-uA4vxnU_39xXdM9ihQ