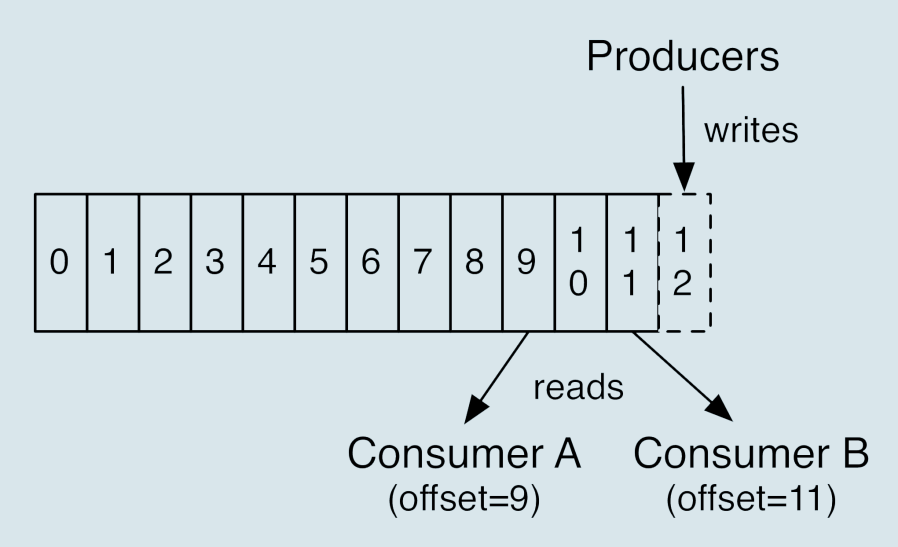
# Kafka位移提交那些事

2021.04.20

1. 理解Consumer位移提交

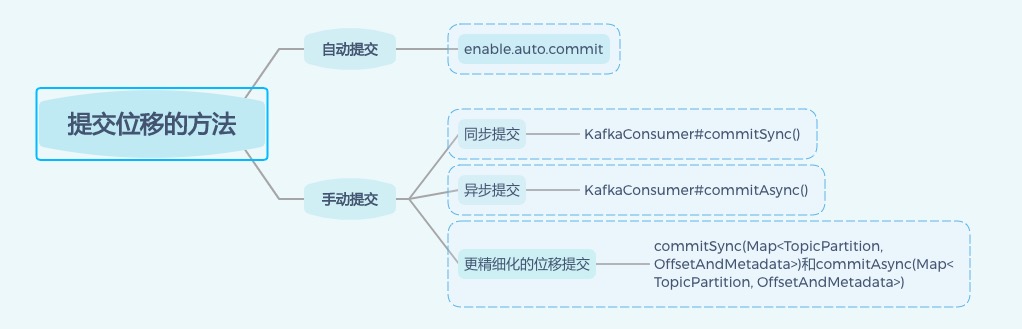
Consumer 需要向 Kafka 汇报自己的位移数据，这个汇报过程被称为提交位移（Committing Offsets）。因为 Consumer 能够同时消费多个分区的数据，所以**位移的提交实际上是在分区粒度上进行的**，即 Consumer 需要为分配给它的每个分区提交各自的位移数据。

提交位移主要是为了**表征 Consumer 的消费进度**，这样当 Consumer 发生故障重启之后，就能够从 Kafka 中读取之前提交的位移值，然后从相应的位移处继续消费，从而避免整个消费过程重来一遍。



1. 位移提交的分类

从用户的角度来说，位移提交分为**自动提交和手动提交**；从 Consumer 端的角度来说，位移提交分为**同步提交和异步提交：**



1. 自动提交位移：

一旦设置了 enable.auto.commit 为 true，Kafka 会保证在开始调用poll方法时，提交上次 poll 返回的所有消息。从顺序上来说，**poll方法的逻辑是先提交上一批消息的位移，再处理下一批消息**，因此它能保证不出现消费丢失的情况。但自动提交位移的一个问题在于，**它可能会出现重复消费**。

在默认情况下，Consumer 每 5 秒自动提交一次位移。现在，我们假设提交位移之后的 3 秒发生了 Rebalance 操作。在 Rebalance 之后，所有 Consumer 从上一次提交的位移处继续消费，但该位移已经是 3 秒前的位移数据了，故在 Rebalance 发生前 3 秒消费的所有数据都要重新再消费一次。虽然你能够通过减少 auto.commit.interval.ms 的值来提高提交频率，但这么做只能缩小重复消费的时间窗口，不可能完全消除它。这是自动提交机制的一个缺陷。

Properties props = new Properties();

props.put("bootstrap.servers", "localhost:9092");

props.put("group.id", "test");

props.put("enable.auto.commit", "true");

props.put("auto.commit.interval.ms", "2000");

props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);

consumer.subscribe(Arrays.asList("foo", "bar"));

while (true) {

ConsumerRecords<String, String> records = consumer.poll(100);

for (ConsumerRecord<String, String> record : records)

System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(), record.key(), record.value());

}

1. 手动提交

要配置 props.put("enable.auto.commit", "false");

手动提交位移，它的好处就在于更加灵活，**你完全能够把控位移提交的时机和频率**。但是，它也有一个缺陷，就是在调用 commitSync() 时，Consumer 程序会处于阻塞状态，直到远端的 Broker 返回提交结果，这个状态才会结束。鉴于这个问题，Kafka 社区为手动提交位移提供了另一个 API 方法：KafkaConsumer#commitAsync()。从名字上来看它就不是同步的，而是一个异步操作。调用 commitAsync() 之后，它会立即返回，不会阻塞，因此不会影响 Consumer 应用的 TPS。由于它是异步的，Kafka 提供了回调函数（callback），供你实现提交之后的逻辑。

1）位移同步提交

while (true) {

ConsumerRecords<String, String> records =

consumer.poll(Duration.ofSeconds(1));

process(records); // 处理消息

try {

consumer.commitSync();

} catch (CommitFailedException e) {

handle(e); // 处理提交失败异常

}

}

2）异步提交

while (true) {

ConsumerRecords<String, String> records = consumer.poll(Duration.ofSeconds(1));

process(records); // 处理消息

consumer.commitAsync((offsets, exception) -> {

if (exception != null)

handle(exception);

});

}

3）同步提交效率低，会影响TPS,但是可以进行重试，异步提交则不能进行重试，所以一般会结合两者来手动提交位移：

try {

while(true) {

ConsumerRecords<String, String> records = consumer.poll(Duration.ofSeconds(1));

process(records); // 处理消息

commitAysnc(); // 使用异步提交规避阻塞

}

} catch(Exception e) {

handle(e); // 处理异常

} finally {

try {

consumer.commitSync(); // 最后一次提交使用同步阻塞式提交

} finally {

consumer.close();

}

}

4) 批量提交位移

private Map<TopicPartition, OffsetAndMetadata> offsets = new HashMap<>();

int count = 0;

……

while (true) {

ConsumerRecords<String, String> records = consumer.poll(Duration.ofSeconds(1));

for (ConsumerRecord<String, String> record: records) {

process(record); // 处理消息

offsets.put(new TopicPartition(record.topic(), record.partition()),

new OffsetAndMetadata(record.offset() + 1)；

if（count % 100 == 0）{

consumer.commitAsync(offsets, null); // 回调处理逻辑是 null

}

count++;

}

}

1. 一些问题
2. 消费者提了异步 commit 实际还没更新完offset，消费者再不断地poll，其实会有重复消费的情况吧？

只要consumer没有重启，不会发生重复消费。因为在运行过程中consumer会记录已获取的消息位移。

1. auto.commit.interval.ms为5秒，且为自动提交如果业务5秒内还没处理完，这个客户端怎么处理offset？

这个参数其实有点误导。它其实的意思是至少5秒。可能多于5秒

1. 可以通过如下三种方法监控消费进度：

