

21 | Kafka Consumer源码分析：消息消费的实现过程

2019-09-12 李玥

消息队列高手课

[进入课程 >](#)



讲述：李玥

时长 14:20 大小 13.13M



你好，我是李玥。

我们在上节课中提到过，用于解决消息队列一些常见问题的知识和原理，最终落地到代码上，都包含在收、发消息这两个流程中。对于消息队列的生产和消费这两个核心流程，在大部分消息队列中，它实现的主要流程都是一样的，所以，通过这两节课的学习之后，掌握了这两个流程的实现过程。无论你使用的是哪种消息队列，遇到收发消息的问题，你都可以用同样的思路去分析和解决问题。

上一节课我和你一起通过分析源代码学习了 RocketMQ 消息生产的实现过程，本节课我们来看一下 Kafka 消费者的源代码，理清 Kafka 消费的实现过程，并且能从中学习到一些 Kafka 的优秀设计思路和编码技巧。

在开始分析源码之前，我们一起来回顾一下 Kafka 消费模型的几个要点：

Kafka 的每个 Consumer（消费者）实例属于一个 ConsumerGroup（消费组）；
在消费时，ConsumerGroup 中的每个 Consumer 独占一个或多个 Partition（分区）；


对于每个 ConsumerGroup，在任意时刻，每个 Partition 至多有 1 个 Consumer 在消费；

每个 ConsumerGroup 都有一个 Coordinator(协调者) 负责分配 Consumer 和 Partition 的对应关系，当 Partition 或是 Consumer 发生变更是，会触发 rebalance（重新分配）过程，重新分配 Consumer 与 Partition 的对应关系；

Consumer 维护与 Coordinator 之间的心跳，这样 Coordinator 就能感知到 Consumer 的状态，在 Consumer 故障的时候及时触发 rebalance。

掌握并理解 Kafka 的消费模型，对于接下来理解其消费的实现过程是至关重要的，如果你对上面的这些要点还有不清楚的地方，建议回顾一下之前的课程或者看一下 Kafka 相关的文档，然后再继续接下来的内容。


我们使用当前最新的版本 2.2 进行分析，使用 Git 在 GitHub 上直接下载源码到本地：

 复制代码

```
1 git clone git@github.com:apache/kafka.git
2 cd kafka
3 git checkout 2.2
```

在《[09 | 学习开源代码该如何入手？](#)》这节课中，我讲过，分析国外源码最好的方式就是从文档入手，接下来我们就找一下 Kafka 的文档，看看从哪儿来入手开启我们的分析流程。

Kafka 的 Consumer 入口类[KafkaConsumer 的 JavaDoc](#)，给出了关于如何使用 KafkaConsumer 非常详细的说明文档，并且给出了一个使用 Consumer 消费的最简代码示例：

 复制代码

```

1      // 设置必要的配置信息
2      Properties props = new Properties();
3      props.put("bootstrap.servers", "localhost:9092");
4      props.put("group.id", "test");
5      props.put("enable.auto.commit", "true");
6      props.put("auto.commit.interval.ms", "1000");
7      props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeseria
8      props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeser:
9
10     // 创建 Consumer 实例
11     KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
12
13     // 订阅 Topic
14     consumer.subscribe(Arrays.asList("foo", "bar"));
15
16     // 循环拉消息
17     while (true) {
18         ConsumerRecords<String, String> records = consumer.poll(100);
19         for (ConsumerRecord<String, String> record : records)
20             System.out.printf("offset = %d, key = %s, value = %s\n", record.offset(),
21     }

```

这段代码主要的主要流程是：

1. 设置必要的配置信息，包括：起始连接的 Broker 地址，Consumer Group 的 ID，自动提交消费位置的配置和序列化配置；
2. 创建 Consumer 实例；
3. 订阅了 2 个 Topic: foo 和 bar；
4. 循环拉取消息并打印在控制台上。


通过上面的代码实例我们可以看到，消费这个大的流程，在 Kafka 中实际上是被分成了“订阅”和“拉取消息”这两个小的流程。另外，我在之前的课程中反复提到过，Kafka 在消费过程中，每个 Consumer 实例是绑定到一个分区上的，那 Consumer 是如何确定，绑定到哪一个分区上的呢？这个问题也是可以通过分析消费流程来找到答案的。所以，我们分析整个消费流程主要聚焦在三个问题上：

1. 订阅过程是如何实现的？
2. Consumer 是如何与 Coordinator 协商，确定消费哪些 Partition 的？
3. 拉取消息的过程是如何实现的？

了解前两个问题，有助于你充分理解 Kafka 的元数据模型，以及 Kafka 是如何在客户端和服务端之间来交换元数据的。最后一个问题，拉取消息的实现过程，实际上就是消费的主要流程，我们上节课讲过，这是消息队列最核心的两个流程之一，也是必须重点掌握的。我们就带着这三个问题，来分析 Kafka 的订阅和拉取消息的过程如何实现。

订阅过程如何实现？

我们先来看看订阅的实现流程。从上面的例子跟踪到订阅的主流程方法：

 复制代码

```
1 public void subscribe(Collection<String> topics, ConsumerRebalanceListener listener) {
2     acquireAndEnsureOpen();
3     try {
4         // 省略部分代码
5
6         // 重置订阅状态
7         this.subscriptions.subscribe(new HashSet<>(topics), listener);
8
9         // 更新元数据
10        metadata.setTopics(subscriptions.groupSubscription());
11    } finally {
12        release();
13    }
14 }
```

在这个代码中，我们先忽略掉各种参数和状态检查的分支代码，订阅的主流程主要更新了两个属性：一个是订阅状态 `subscriptions`，另一个是更新元数据中的 `topic` 信息。订阅状态 `subscriptions` 主要维护了订阅的 `topic` 和 `partition` 的消费位置等状态信息。属性 `metadata` 中维护了 Kafka 集群元数据的一个子集，包括集群的 `Broker` 节点、`Topic` 和 `Partition` 在节点上分布，以及我们聚焦的第二个问题：Coordinator 给 Consumer 分配的 `Partition` 信息。

请注意一下，这个 `subscribe()` 方法的实现有一个非常值得大家学习的地方：就是开始的 `acquireAndEnsureOpen()` 和 `try-finally release()`，作用就是保护这个方法只能单线程调用。


Kafka 在文档中明确地注明了 Consumer 不是线程安全的，意味着 Consumer 被并发调用时会出现不可预期的结果。为了避免这种情况发生，Kafka 做了主动的检测并抛出异常，而

不是放任系统产生不可预期的情况。

Kafka “**主动检测不支持的情况并抛出异常，避免系统产生不可预期的行为**” 这种模式，对于增强的系统的健壮性是一种非常有效的做法。如果你的系统不支持用户的某种操作，正确的做法是，检测不支持的操作，直接拒绝用户操作，并给出明确的错误提示，而不应该只是在文档中写上 “不要这样做”，却放任用户错误的操作，产生一些不可预期的、奇怪的错误结果。

具体 Kafka 是如何实现的并发检测，大家可以看一下方法 `acquireAndEnsureOpen()` 的实现，很简单也很经典，我们就不再展开讲解了。

继续跟进到更新元数据的方法 `metadata.setTopics()` 里面，这个方法的实现除了更新元数据类 `Metadata` 中的 `topic` 相关的一些属性以外，还调用了 `Metadata.requestUpdate()` 方法请求更新元数据。

 复制代码

```
1 public synchronized int requestUpdate() {  
2     this.needUpdate = true;  
3     return this.updateVersion;  
4 }
```

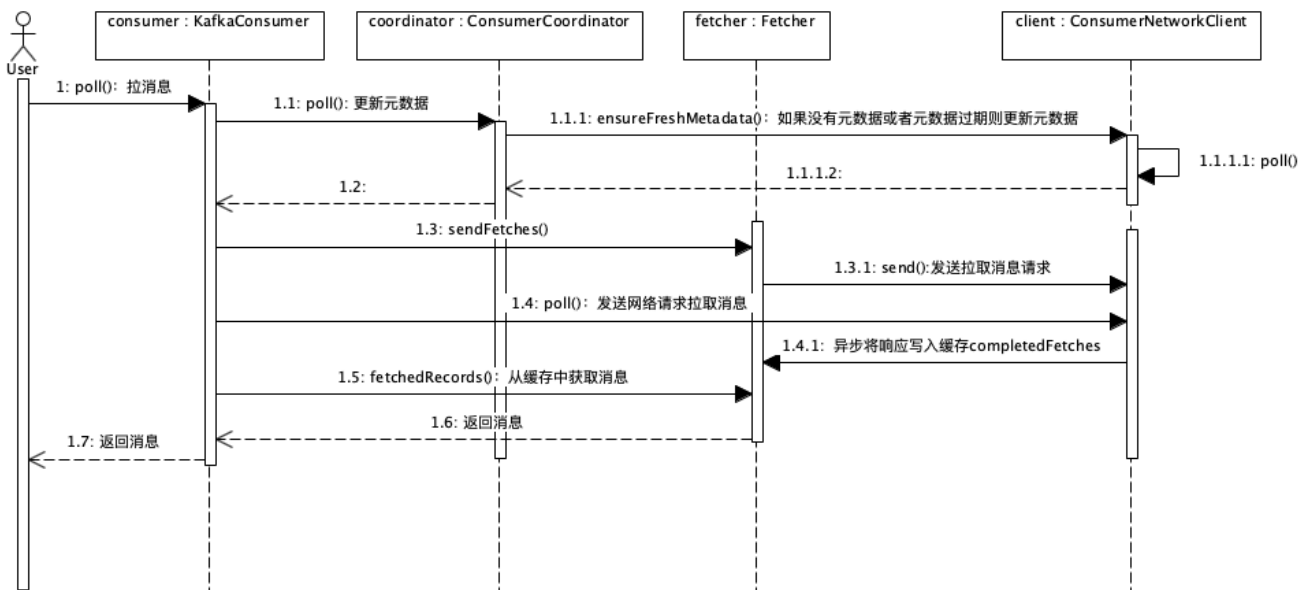
跟进到 `requestUpdate()` 的方法里面我们会发现，这里面并没有真正发送更新元数据的请求，只是将需要更新元数据的标志位 `needUpdate` 设置为 `true` 就结束了。Kafka 必须确保在第一次拉消息之前元数据是可用的，也就是说在第一次拉消息之前必须更新一次元数据，否则 Consumer 就不知道它应该去哪个 Broker 上去拉哪个 Partition 的消息。

分析完订阅相关的代码，我们来总结一下：在订阅的实现过程中，Kafka 更新了订阅状态 `subscriptions` 和元数据 `metadata` 中的相关 `topic` 的一些属性，将元数据状态置为 “需要立即更新”，但是并没有真正发送更新元数据的请求，整个过程没有和集群有任何网络数据交换。

那这个元数据会在什么时候真正做一次更新呢？我们可以先带着这个问题接着看代码。

拉取消息的过程如何实现？

接下来，我们分析拉取消息的流程。这个流程的时序图如下（点击图片可放大查看）：



我们对着时序图来分析它的实现流程。在 `KafkaConsumer.poll()` 方法（对应源码 1179 行）的实现里面，可以看到主要是先后调用了 2 个私有方法：

1. `updateAssignmentMetadataIfNeeded()`: 更新元数据。
2. `pollForFetches()`: 拉取消息。


方法 `updateAssignmentMetadataIfNeeded()` 中，调用了 `coordinator.poll()` 方法，`poll()` 方法里面又调用了 `client.ensureFreshMetadata()` 方法，在 `client.ensureFreshMetadata()` 方法中又调用了 `client.poll()` 方法，实现了与 Cluster 通信，在 Coordinator 上注册 Consumer 并拉取和更新元数据。至此，“元数据会在什么时候真正做一次更新”这个问题也有了答案。

类 `ConsumerNetworkClient` 封装了 Consumer 和 Cluster 之间所有的网络通信的实现，这个类是一个非常彻底的异步实现。它没有维护任何的线程，所有待发送的 Request 都存放在属性 `unsent` 中，返回的 Response 存放在属性 `pendingCompletion` 中。每次调用 `poll()` 方法的时候，在当前线程中发送所有待发送的 Request，处理所有收到的 Response。

我们在之前的课程中讲到过，这种异步设计的优势就是用很少的线程实现高吞吐量，劣势也非常明显，极大增加了代码的复杂度。对比上节课我们分析的 RocketMQ 的代码，

Producer 和 Consumer 在主要收发消息流程上功能的复杂度是差不多的，但是你可以很明显地感受到 Kafka 的代码实现要比 RocketMQ 的代码实现更加的复杂难于理解。

我们继续分析方法 pollForFetches() 的实现。

 复制代码

```
1 private Map<TopicPartition, List<ConsumerRecord<K, V>>> pollForFetches(Timer timer)
2     // 省略部分代码
3     // 如果缓存里面有未读取的消息，直接返回这些消息
4     final Map<TopicPartition, List<ConsumerRecord<K, V>>> records = fetcher.fetchedl
5     if (!records.isEmpty()) {
6         return records;
7     }
8     // 构造拉取消息请求，并发送
9     fetcher.sendFetches();
10    // 省略部分代码
11    // 发送网络请求拉取消息，等待直到有消息返回或者超时
12    client.poll(pollTimer, () -> {
13        return !fetcher.hasCompletedFetches();
14    });
15    // 省略部分代码
16    // 返回拉到的消息
17    return fetcher.fetchedRecords();
18 }
```

这段代码的主要实现逻辑是：

1. 如果缓存里面有未读取的消息，直接返回这些消息；
2. 构造拉取消息请求，并发送；
3. 发送网络请求并拉取消息，等待直到有消息返回或者超时；
4. 返回拉到的消息。

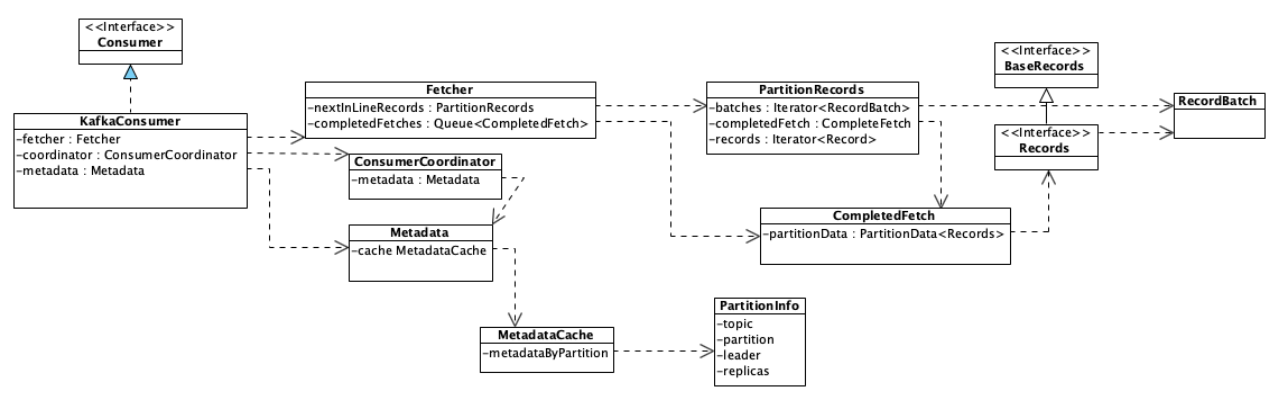
在方法 fetcher.sendFetches() 的实现里面，Kafka 根据元数据的信息，构造到所有需要的 Broker 的拉消息的 Request，然后调用 client.Send() 方法将这些请求异步发送出去。并且，注册了一个回调类来处理返回的 Response，所有返回的 Response 被暂时存放在 Fetcher.completedFetches 中。需要注意的是，这时的 Request 并没有被真正发给各个 Broker，而是被暂存在了 client.unsend 中等待被发送。

然后，在调用 `client.poll()` 方法时，会真正将之前构造的所有 `Request` 发送出去，并处理收到的 `Response`。

最后，`fetcher.fetchedRecords()` 方法中，将返回的 `Response` 反序列化后转换为消息列表，返回给调用者。

综合上面的实现分析，我在这里给出整个拉取消息的流程涉及到的相关类的类图，在这个类图中，为了便于你理解，我并没有把所有类都绘制上去，只是把本节课两个流程相关的主要类和这些类里的关键属性画在了图中。你可以配合这个类图和上面的时序图进行代码阅读。

类图（点击图片可放大查看）：



小结

本节课我们一起分析了 Kafka Consumer 消费消息的实现过程。大家来分析代码过程中，不仅仅是要掌握 Kafka 整个消费的流程是如何实现的，更重要的是理解它这种完全异步的设计思想。

发送请求时，构建 `Request` 对象，暂存入发送队列，但不立即发送，而是等待合适的时机批量发送。并且，用回调或者 `RequestFuture` 方式，预先定义好如何处理响应的逻辑。在收到 Broker 返回的响应之后，也不会立即处理，而是暂存在队列中，择机处理。那这个择机策略就比较复杂了，有可能是需要读取响应的时候，也有可能是缓冲区满了或是时间到了，都有可能触发一次真正的网络请求，也就是在 `poll()` 方法中发送所有待发送 `Request` 并处理所有 `Response`。

这种设计的好处是，不需要维护用于异步发送的和处理响应的线程，并且能充分发挥批量处理的优势，这也是 Kafka 的性能非常好的原因之一。这种设计的缺点也非常的明显，就是实现的复杂度太大了，如果没有深厚的代码功力，很难驾驭这么复杂的设计，并且后续维护的成本也很高。

总体来说，不推荐大家把代码设计得这么复杂。代码结构简单、清晰、易维护是我们在设计过程中需要考虑的一个非常重要的因素。很多时候，为了获得较好的代码结构，在可接受的范围内，去牺牲一些性能，也是划算的。

思考题

我们知道，Kafka Consumer 在消费过程中是需要维护消费位置的，Consumer 每次从当前消费位置拉取一批消息，这些消息都被正常消费后，Consumer 会给 Coordinator 发一个提交位置的请求，然后消费位置会向后移动，完成一批消费过程。那 kafka Consumer 是如何维护和提交这个消费位置的呢？请你带着这个问题再回顾一下 Consumer 的代码，尝试独立分析代码并找到答案。欢迎在留言区与我分享讨论。

感谢阅读，如果你觉得这篇文章对你有一些启发，也欢迎把它分享给你的朋友。



消息队列高手课

从源码角度全面解析 MQ 的设计与实现

李玥

京东零售技术架构部资深架构师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

上一篇 20 | RocketMQ Producer源码分析：消息生产的实现过程

下一篇 22 | Kafka和RocketMQ的消息复制实现的差异点在哪？

精选留言 (5)

写留言



业余草

2019-09-12

我们需要一些速成的实战经验，比如消息队列突然延迟2个小时该如何解决？希望带着这些类似的问题，结合设计原理，帮助我们层层解析。。。



5



leslie

2019-09-12

先打卡：代码慢慢研究；老师今天讲述了研究代码的目的：

1.消息队列实现的主要流程都一样，掌握流程的实现过程；遇到收发消息的问题，都可以用同样的思路去分析和解决问题。

2.看一下源代码，理清消费生产的实现过程，从中学习一些优秀的设计思路和编码技巧。

这个算是一个小的总结吧：明天中秋休息刚好可以啃代码，好好研究代码去体会。...

展开



1



米兰铁匠

2019-09-15

用代码跟类图来讲解一个系统的实现还是太过于抽象，难以把握住，最好的方式是用老师理解的东西然后转换成通俗易懂的原理图来讲解

展开



Geek_71a2ee

2019-09-13

消费者如何从服务端拉取消息的，用for循环效率太低吧，能否说说实际的代码

作者回复: 同学，我们这节课通篇就是讲得这个问题啊，给你们讲解使用的就是实际的代码。



微微一笑

2019-09-12

老师好，先祝您节日快乐!!! 您辛苦了~

有几个疑问需要老师解答一下:

①今天在看rocketMq源码过程中，发现DefaultMQProducer有个属性defaultTopicQueueNums，它是用来设置topic的ConsumeQueue的数量的吗？我之前的理解是，consumeQueue的数量是创建topic的时候指定的，跟producer没有关系，那这个参数又有什...

展开 ∨

