21-Java消费者是如何管理TCP连接的-

你好，我是胡夕。今天我要和你分享的主题是：Kafka的Java消费者是如何管理TCP连接的。

在专栏第13讲中，我们专门聊过“Java**生产者**是如何管理TCP连接资源的”这个话题，你应该还有印象吧？今天算是它的姊妹篇，我们一起来研究下Kafka的Java**消费者**管理TCP或Socket资源的机制。只有完成了今天的讨论，我们才算是对Kafka客户端的TCP连接管理机制有了全面的了解。

和之前一样，我今天会无差别地混用TCP和Socket两个术语。毕竟，在Kafka的世界中，无论是ServerSocket，还是SocketChannel，它们实现的都是TCP协议。或者这么说，Kafka的网络传输是基于TCP协议的，而不是基于UDP协议，因此，当我今天说到TCP连接或Socket资源时，我指的是同一个东西。

何时创建TCP连接？

我们先从消费者创建TCP连接开始讨论。消费者端主要的程序入口是KafkaConsumer类。**和生产者不同的是，构建KafkaConsumer实例时是不会创建任何TCP连接的**，也就是说，当你执行完new KafkaConsumer(properties)语句后，你会发现，没有Socket连接被创建出来。这一点和Java生产者是有区别的，主要原因就是生产者入口类KafkaProducer在构建实例的时候，会在后台默默地启动一个Sender线程，这个Sender线程负责Socket连接的创建。

从这一点上来看，我个人认为KafkaConsumer的设计比KafkaProducer要好。就像我在第13讲中所说的，在Java构造函数中启动线程，会造成this指针的逃逸，这始终是一个隐患。

如果Socket不是在构造函数中创建的，那么是在KafkaConsumer.subscribe或KafkaConsumer.assign方法中创建的吗？严格来说也不是。我还是直接给出答案吧：**TCP连接是在调用KafkaConsumer.poll方法时被创建的**。再细粒度地说，在poll方法内部有3个时机可以创建TCP连接。

1.**发起FindCoordinator请求时**。

还记得消费者端有个组件叫协调者（Coordinator）吗？它驻留在Broker端的内存中，负责消费者组的组成员管理和各个消费者的位移提交管理。当消费者程序首次启动调用poll方法时，它需要向Kafka集群发送一个名为FindCoordinator的请求，希望Kafka集群告诉它哪个Broker是管理它的协调者。

不过，消费者应该向哪个Broker发送这类请求呢？理论上任何一个Broker都能回答这个问题，也就是说消费者可以发送FindCoordinator请求给集群中的任意服务器。在这个问题上，社区做了一点点优化：消费者程序会向集群中当前负载最小的那台Broker发送请求。负载是如何评估的呢？其实很简单，就是看消费者连接的所有Broker中，谁的待发送请求最少。当然了，这种评估显然是消费者端的单向评估，并非是站在全局角度，因此有的时候也不一定是最优解。不过这不并影响我们的讨论。总之，在这一步，消费者会创建一个Socket连接。

2.**连接协调者时。**

Broker处理完上一步发送的FindCoordinator请求之后，会返还对应的响应结果（Response），显式地告诉消费者哪个Broker是真正的协调者，因此在这一步，消费者知晓了真正的协调者后，会创建连向该Broker的Socket连接。只有成功连入协调者，协调者才能开启正常的组协调操作，比如加入组、等待组分配方案、心跳请求处理、位移获取、位移提交等。

3.**消费数据时。**

消费者会为每个要消费的分区创建与该分区领导者副本所在Broker连接的TCP。举个例子，假设消费者要消费5个分区的数据，这5个分区各自的领导者副本分布在4台Broker上，那么该消费者在消费时会创建与这4台Broker的Socket连接。

创建多少个TCP连接？

下面我们来说说消费者创建TCP连接的数量。你可以先思考一下大致需要的连接数量，然后我们结合具体的Kafka日志，来验证下结果是否和你想的一致。

我们来看看这段日志。

> [2019-05-27 10:00:54,142] DEBUG [Consumer clientId=consumer-1, groupId=test] Initiating connection to node localhost:9092 (id: -1 rack: null) using address localhost/127.0.0.1 (org.apache.kafka.clients.NetworkClient:944)

> ...

> [2019-05-27 10:00:54,188] DEBUG [Consumer clientId=consumer-1, groupId=test] Sending metadata request MetadataRequestData(topics=[MetadataRequestTopic(name=‘t4’)], allowAutoTopicCreation=true, includeClusterAuthorizedOperations=false, includeTopicAuthorizedOperations=false) to node localhost:9092 (id: -1 rack: null) (org.apache.kafka.clients.NetworkClient:1097)

> ...

> [2019-05-27 10:00:54,188] TRACE [Consumer clientId=consumer-1, groupId=test] Sending FIND_COORDINATOR {key=test,key_type=0} with correlation id 0 to node -1 (org.apache.kafka.clients.NetworkClient:496)

> [2019-05-27 10:00:54,203] TRACE [Consumer clientId=consumer-1, groupId=test] Completed receive from node -1 for FIND_COORDINATOR with correlation id 0, received {throttle_time_ms=0,error_code=0,error_message=null, node_id=2,host=localhost,port=9094} (org.apache.kafka.clients.NetworkClient:837)

> ...

> [2019-05-27 10:00:54,204] DEBUG [Consumer clientId=consumer-1, groupId=test] Initiating connection to node localhost:9094 (id: 2147483645 rack: null) using address localhost/127.0.0.1 (org.apache.kafka.clients.NetworkClient:944)

> ...

> [2019-05-27 10:00:54,237] DEBUG [Consumer clientId=consumer-1, groupId=test] Initiating

> connection to node localhost:9094 (id: 2 rack: null) using address localhost/127.0.0.1 (org.apache.kafka.clients.NetworkClient:944)

> [2019-05-27 10:00:54,237] DEBUG [Consumer clientId=consumer-1, groupId=test] Initiating connection to node localhost:9092 (id: 0 rack: null) using address localhost/127.0.0.1 (org.apache.kafka.clients.NetworkClient:944)

> [2019-05-27 10:00:54,238] DEBUG [Consumer clientId=consumer-1, groupId=test] Initiating connection to node localhost:9093 (id: 1 rack: null) using address localhost/127.0.0.1 (org.apache.kafka.clients.NetworkClient:944)

这里我稍微解释一下，日志的第一行是消费者程序创建的第一个TCP连接，就像我们前面说的，这个Socket用于发送FindCoordinator请求。由于这是消费者程序创建的第一个连接，此时消费者对于要连接的Kafka集群一无所知，因此它连接的Broker节点的ID是-1，表示消费者根本不知道要连接的Kafka Broker的任何信息。

值得注意的是日志的第二行，消费者复用了刚才创建的那个Socket连接，向Kafka集群发送元数据请求以获取整个集群的信息。

日志的第三行表明，消费者程序开始发送FindCoordinator请求给第一步中连接的Broker，即localhost:9092，也就是nodeId等于-1的那个。在十几毫秒之后，消费者程序成功地获悉协调者所在的Broker信息，也就是第四行标为橙色的“node_id = 2”。

完成这些之后，消费者就已经知道协调者Broker的连接信息了，因此在日志的第五行发起了第二个Socket连接，创建了连向localhost:9094的TCP。只有连接了协调者，消费者进程才能正常地开启消费者组的各种功能以及后续的消息消费。

在日志的最后三行中，消费者又分别创建了新的TCP连接，主要用于实际的消息获取。还记得我刚才说的吗？要消费的分区的领导者副本在哪台Broker上，消费者就要创建连向哪台Broker的TCP。在我举的这个例子中，localhost:9092，localhost:9093和localhost:9094这3台Broker上都有要消费的分区，因此消费者创建了3个TCP连接。

看完这段日志，你应该会发现日志中的这些Broker节点的ID在不断变化。有时候是-1，有时候是2147483645，只有在最后的时候才回归正常值0、1和2。这又是怎么回事呢？

前面我们说过了-1的来由，即消费者程序（其实也不光是消费者，生产者也是这样的机制）首次启动时，对Kafka集群一无所知，因此用-1来表示尚未获取到Broker数据。

那么2147483645是怎么来的呢？它是**由Integer.MAX_VALUE减去协调者所在Broker的真实ID计算得来的**。看第四行标为橙色的内容，我们可以知道协调者ID是2，因此这个Socket连接的节点ID就是Integer.MAX_VALUE减去2，即2147483647减去2，也就是2147483645。这种节点ID的标记方式是Kafka社区特意为之的结果，目的就是要让组协调请求和真正的数据获取请求使用不同的Socket连接。

至于后面的0、1、2，那就很好解释了。它们表征了真实的Broker ID，也就是我们在server.properties中配置的broker.id值。

我们来简单总结一下上面的内容。通常来说，消费者程序会创建3类TCP连接：

1. 确定协调者和获取集群元数据。
2. 连接协调者，令其执行组成员管理操作。
3. 执行实际的消息获取。

那么，这三类TCP请求的生命周期都是相同的吗？换句话说，这些TCP连接是何时被关闭的呢？

何时关闭TCP连接？

和生产者类似，消费者关闭Socket也分为主动关闭和Kafka自动关闭。主动关闭是指你显式地调用消费者API的方法去关闭消费者，具体方式就是**手动调用KafkaConsumer.close()方法，或者是执行Kill命令**，不论是Kill -2还是Kill -9；而Kafka自动关闭是由**消费者端参数connection.max.idle.ms**控制的，该参数现在的默认值是9分钟，即如果某个Socket连接上连续9分钟都没有任何请求“过境”的话，那么消费者会强行“杀掉”这个Socket连接。

不过，和生产者有些不同的是，如果在编写消费者程序时，你使用了循环的方式来调用poll方法消费消息，那么上面提到的所有请求都会被定期发送到Broker，因此这些Socket连接上总是能保证有请求在发送，从而也就实现了“长连接”的效果。

针对上面提到的三类TCP连接，你需要注意的是，**当第三类TCP连接成功创建后，消费者程序就会废弃第一类TCP连接**，之后在定期请求元数据时，它会改为使用第三类TCP连接。也就是说，最终你会发现，第一类TCP连接会在后台被默默地关闭掉。对一个运行了一段时间的消费者程序来说，只会有后面两类TCP连接存在。

可能的问题

从理论上说，Kafka Java消费者管理TCP资源的机制我已经说清楚了，但如果仔细推敲这里面的设计原理，还是会发现一些问题。

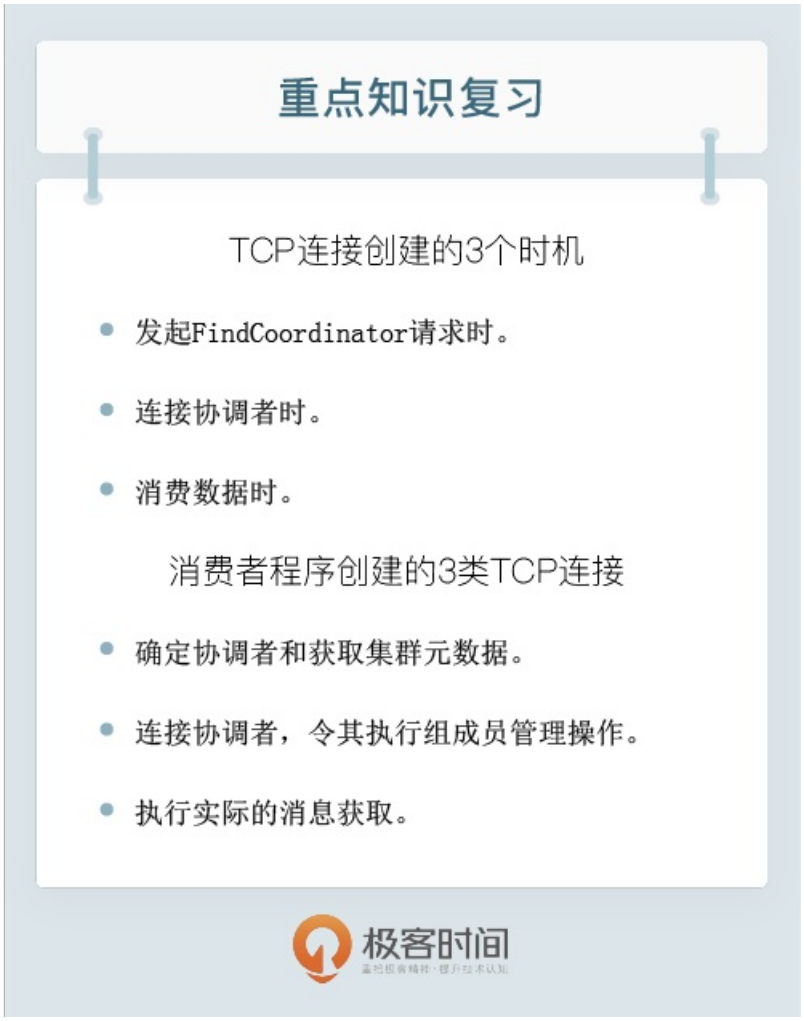
我们刚刚讲过，第一类TCP连接仅仅是为了首次获取元数据而创建的，后面就会被废弃掉。最根本的原因是，消费者在启动时还不知道Kafka集群的信息，只能使用一个“假”的ID去注册，即使消费者获取了真实的Broker ID，它依旧无法区分这个“假”ID对应的是哪台Broker，因此也就无法重用这个Socket连接，只能再重新创建一个新的连接。

为什么会出现这种情况呢？主要是因为目前Kafka仅仅使用ID这一个维度的数据来表征Socket连接信息。这点信息明显不足以确定连接的是哪台Broker，也许在未来，社区应该考虑使用**<主机名、端口、ID>**三元组的方式来定位Socket资源，这样或许能够让消费者程序少创建一些TCP连接。

也许你会问，反正Kafka有定时关闭机制，这算多大点事呢？其实，在实际场景中，我见过很多将connection.max.idle.ms设置成-1，即禁用定时关闭的案例，如果是这样的话，这些TCP连接将不会被定期清除，只会成为永久的“僵尸”连接。基于这个原因，社区应该考虑更好的解决方案。

小结

好了，今天我们补齐了Kafka Java客户端管理TCP连接的“拼图”。我们不仅详细描述了Java消费者是怎么创建和关闭TCP连接的，还对目前的设计方案提出了一些自己的思考。希望今后你能将这些知识应用到自己的业务场景中，并对实际生产环境中的Socket管理做到心中有数。



开放讨论

假设有个Kafka集群由2台Broker组成，有个主题有5个分区，当一个消费该主题的消费者程序启动时，你认为该程序会创建多少个Socket连接？为什么？

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：

- 常超 2019-07-20 06:42:11
  整个生命周期里会建立4个连接，进入稳定的消费过程后，同时保持3个连接，以下是详细。
  第一类连接：确定协调者和获取集群元数据。
  一个，初期的时候建立，当第三类连接建立起来之后，这个连接会被关闭。

  第二类连接：连接协调者，令其执行组成员管理操作。
  一个

  第三类连接：执行实际的消息获取。
  两个分别会跟两台broker机器建立一个连接，总共两个TCP连接，同一个broker机器的不同分区可以复用一个socket。 [4赞]

- rm -rf 😊 2019-07-21 17:08:19
  我认为也是3个连接，第一个是查找Coordinator的，这个会在后面断开。然后5个partition会分布在2个broker上，那么客户端最多也就连接2次就能消费所有partition了，因此是连接3个，最后保持2个。 [1赞]

- nightmare 2019-07-20 00:53:38
  3个tcp连接 一个查询协调着和获取元数据的tcp连接 一个连接协调写 管理组成员的tcp连接 主题5个分区只有连接leader副本的broker需要创建连接 [1赞]

- taj3991 2019-07-22 21:13:58
  老师同一个消费组的客户端都只会连接到一个协调者吗？

- 吴宇晨 2019-07-22 10:23:07
  一个获取元数据的连接（之后会断开）+两个连接分区leader的连接+一个连接协调者的连接

- Williamzhang 2019-07-22 09:05:43
  我觉得作者可以跟学员的留言互动，然后每期课后思考可以在下期中贴出答案及分析，其实留言讨论也是一个非常让人有收获的地方

- 30斤的大番薯 2019-07-22 08:34:12
  老师您好。我之前搭建了一个单机kafka，能正常收发消息。最近我按网上的介绍，把kafka改成集群，出现一个问题：使用kafka自带的命令行生产者工具可以成功发送消息。但是用命令行消费者工具总是接收不到数据（启动消费者一直不输出数据，仿佛没收到消息一样）。我用strace跟踪发现消费者进程一在循环进行epoll（超时）调用，kafka服务器日志无异常。请问这种情况要怎么检查问题。

- 小木匠 2019-07-22 07:43:51
  “负载是如何评估的呢？其实很简单，就是看消费者连接的所有 Broker 中，谁的待发送请求最少。”
  老师这个没太明白，这时候消费者不是还没连接么？那这部分信息是从哪获取到的呢？消费者本地吗？

- October 2019-07-21 10:57:28
  总共创建4个连接，最终保持3个连接：
  确定消费者所属的消费组对应的GroupCoordinator和获取集群的metadata时创建一个TCP连接，由于此时的node id = -1，所以该连接无法重用。
  连接GroupCoordinator时，创建第二个TCP连接，node id值为Integer.MAX_VALUE-id
  消费者会与每个分区的leader创建一个TCP连接来消费数据，node id为broker.id，由于kafka只是用id这一维度来表征Socket连接信息，因此如果多个分区的leader在同一个broker上时，会共用一个TCP连接，由于分区数大于broker的数量，所以会创建两个TCP连接消费数据。

- 杨陆伟 2019-07-20 17:07:30
  我觉得是3个TCP连接，查找协调者1个连接，连接两个Broker2个连接，且查找协调者的连接慢慢会关闭

- 明翼 2019-07-20 09:29:04
  连接有三个阶段：首先获取协调者连接同时也获取元数据信息，这个连接后面会关闭；连接协调者执行，等待分配分区，组协调等，这需要一个连接；后面真正消费五个分区两个broker最多就两个连接，分区大于broker所以一定是两个，因为第一类连接没有id，所以无法重用，会在第三类开启连接后关闭，所以开始四个连接最终保持三个连接

- 空知 2019-07-19 22:56:01
  同一个broker 一个topic下的分区可以复用一个连接 但是topic的不同分区的 leader 不一定都在一个broker上 也就是消费时候 只有一个broker上的tcp连接起作用