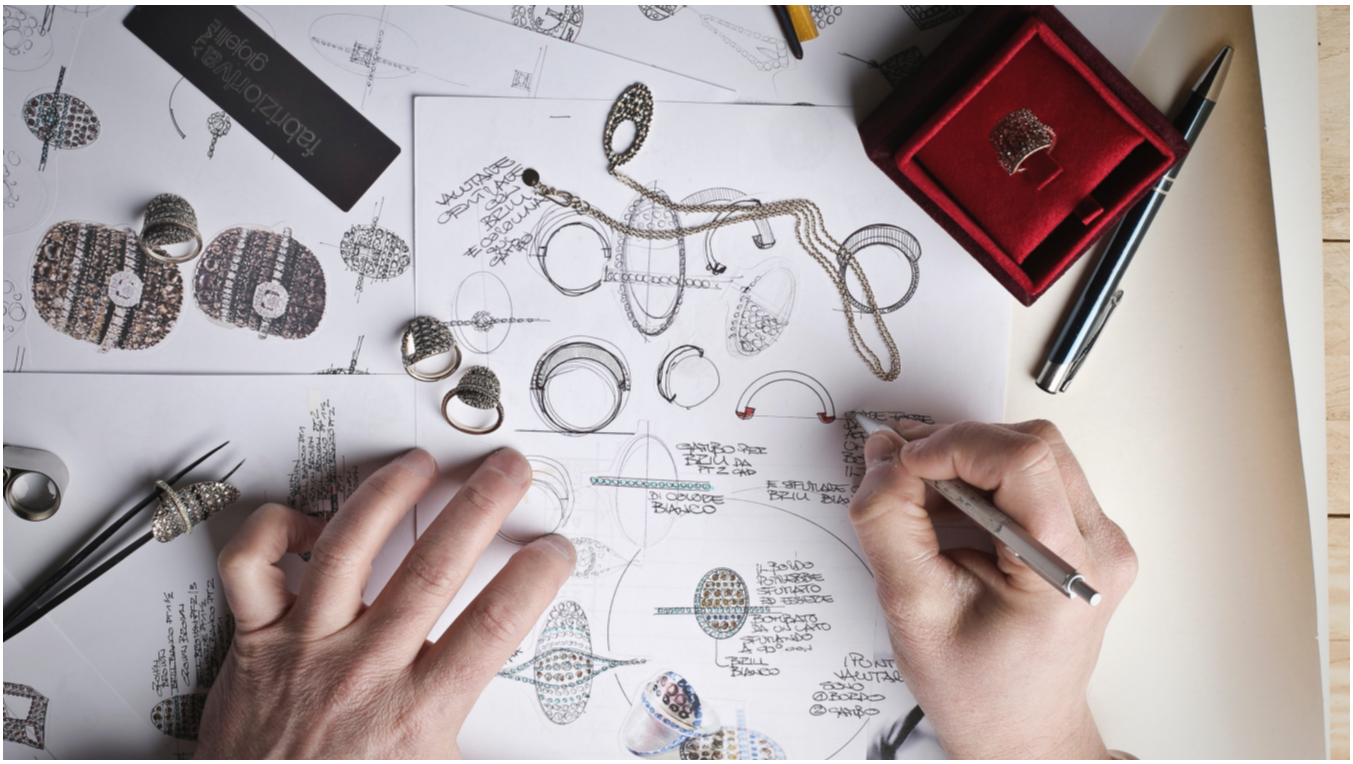
13 | Java生产者是如何管理TCP连接的？

2019-07-02 胡夕



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

为何采用TCP？

Apache Kafka的所有通信都是基于TCP的，而不是基于HTTP或其他协议。无论是生产者、消费者，还是Broker之间的通信都是如此。你可能会问，为什么Kafka不使用HTTP作为底层的通信协议呢？其实这里面的原因有很多，但最主要的原因在于TCP和HTTP之间的区别。

从社区的角度来看，在开发客户端时，人们能够利用TCP本身提供的一些高级功能，比如多路复用请求以及同时轮询多个连接的能力。

所谓的多路复用请求，即multiplexing request，是指将两个或多个数据流合并到底层单一物理连接中的过程。TCP的多路复用请求会在一条物理连接上创建若干个虚拟连接，每个虚拟连接负责流转各自对应的数据流。其实严格来说，TCP并不能多路复用，它只是提供可靠的消息交付语义保证，比如自动重传丢失的报文。

更严谨地说，作为一个基于报文的协议，TCP能够被用于多路复用连接场景的前提是，上层的应用协议（比如HTTP）允许发送多条消息。不过，我们今天并不是要详细讨论TCP原理，因此你只需要知道这是社区采用TCP的理由之一就行了。

除了TCP提供的这些高级功能有可能被Kafka客户端的开发人员使用之外，社区还发现，目前已知的HTTP库在很多编程语言中都略显简陋。

基于这两个原因，Kafka社区决定采用TCP协议作为所有请求通信的底层协议。

Kafka生产者程序概览

Kafka的Java生产者API主要的对象就是KafkaProducer。通常我们开发一个生产者的步骤有4步。

第1步：构造生产者对象所需的参数对象。

第2步：利用第1步的参数对象，创建KafkaProducer对象实例。

第3步：使用KafkaProducer的send方法发送消息。

第4步：调用KafkaProducer的close方法关闭生产者并释放各种系统资源。

上面这4步写成Java代码的话大概是这个样子：

```
Properties props = new Properties ();
props.put(“参数1”, “参数1的值”);
props.put(“参数2”, “参数2的值”);
……
try (Producer<String, String> producer = new KafkaProducer<>(props)) {
            producer.send(new ProducerRecord<String, String>(……), callback);
 ……
}
```

这段代码使用了Java 7 提供的try-with-resource特性，所以并没有显式调用producer.close()方法。无论是否显式调用close方法，所有生产者程序大致都是这个路数。

现在问题来了，当我们开发一个Producer应用时，生产者会向Kafka集群中指定的主题（Topic）发送消息，这必然涉及与Kafka Broker创建TCP连接。那么，Kafka的Producer客户端是如何管理这些TCP连接的呢？

何时创建TCP连接？

要回答上面这个问题，我们首先要弄明白生产者代码是什么时候创建TCP连接的。就上面的那段代码而言，可能创建TCP连接的地方有两处：Producer producer = new KafkaProducer(props)和producer.send(msg, callback)。你觉得连向Broker端的TCP连接会是哪里创建的呢？前者还是后者，抑或是两者都有？请先思考5秒钟，然后我给出我的答案。

首先，生产者应用在创建KafkaProducer实例时是会建立与Broker的TCP连接的。其实这种表述也不是很准确，应该这样说：**在创建KafkaProducer实例时，生产者应用会在后台创建并启**

动一个名为Sender的线程，该Sender线程开始运行时首先会创建与Broker的连接。我截取了一段测试环境中的日志来说明这一点：

> [2018-12-09 09:35:45,620] DEBUG [Producer clientId=producer-1] Initialize connection to node localhost:9093 (id: -2 rack: null) for sending metadata request (org.apache.kafka.clients.NetworkClient:1084)

> [2018-12-09 09:35:45,622] DEBUG [Producer clientId=producer-1] Initiating connection to node localhost:9093 (id: -2 rack: null) using address localhost/127.0.0.1 (org.apache.kafka.clients.NetworkClient:914)

> [2018-12-09 09:35:45,814] DEBUG [Producer clientId=producer-1] Initialize connection to node localhost:9092 (id: -1 rack: null) for sending metadata request (org.apache.kafka.clients.NetworkClient:1084)

> [2018-12-09 09:35:45,815] DEBUG [Producer clientId=producer-1] Initiating connection to node localhost:9092 (id: -1 rack: null) using address localhost/127.0.0.1 (org.apache.kafka.clients.NetworkClient:914)

> [2018-12-09 09:35:45,828] DEBUG [Producer clientId=producer-1] Sending metadata request (type=MetadataRequest, topics=) to node localhost:9093 (id: -2 rack: null) (org.apache.kafka.clients.NetworkClient:1068)

你也许会问：怎么可能是这样？如果不调用send方法，这个Producer都不知道给哪个主题发消息，它又怎么能知道连接哪个Broker呢？难不成它会连接bootstrap.servers参数指定的所有Broker吗？嗯，是的，Java Producer目前还真是这样设计的。

我在这里稍微解释一下bootstrap.servers参数。它是Producer的核心参数之一，指定了这个Producer启动时要连接的Broker地址。请注意，这里的“启动时”，代表的是Producer启动时会发起与这些Broker的连接。因此，如果你为这个参数指定了1000个Broker连接信息，那么很遗憾，你的Producer启动时会首先创建与这1000个Broker的TCP连接。

在实际使用过程中，我并不建议把集群中所有的Broker信息都配置到bootstrap.servers中，通常你指定3～4台就足以了。因为Producer一旦连接到集群中的任一台Broker，就能拿到整个集群的Broker信息，故没必要为bootstrap.servers指定所有的Broker。

让我们回顾一下上面的日志输出，请注意我标为橙色的内容。从这段日志中，我们可以发现，在KafkaProducer实例被创建后以及消息被发送前，Producer应用就开始创建与两台Broker的TCP连接了。当然了，在我的测试环境中，我为bootstrap.servers配置了localhost:9092、localhost:9093来模拟不同的Broker，但是这并不影响后面的讨论。另外，日志输出中的最后一行也很关键：它表明Producer向某一台Broker发送了METADATA请求，尝试获取集群的元数据

信息——这就是前面提到的Producer能够获取集群所有信息的方法。

讲到这里，我有一些个人的看法想跟你分享一下。通常情况下，我都不认为社区写的代码或做的设计就一定是对的，因此，很多类似的这种“质疑”会时不时地在我脑子里冒出来。

拿今天的这个KafkaProducer创建实例来说，社区的官方文档中提及KafkaProducer类是线程安全的。我本人并没有详尽地去验证过它是否真的就是thread-safe的，但是大致浏览一下源码可以得出这样的结论：KafkaProducer实例创建的线程和前面提到的Sender线程共享的可变数据结构只有RecordAccumulator类，故维护了RecordAccumulator类的线程安全，也就实现了KafkaProducer类的线程安全。

你不需要了解RecordAccumulator类是做什么的，你只要知道它主要的数据结构是一个ConcurrentMap<TopicPartition, Deque>。TopicPartition是Kafka用来表示主题分区的Java对象，本身是不可变对象。而RecordAccumulator代码中用到Deque的地方都有锁的保护，所以基本上可以认定RecordAccumulator类是线程安全的。

说了这么多，我其实是想说，纵然KafkaProducer是线程安全的，我也不赞同创建KafkaProducer实例时启动Sender线程的做法。写了《Java并发编程实践》的那位布赖恩·格茨（Brian Goetz）大神，明确指出了这样做的风险：在对象构造器中启动线程会造成this指针的逃逸。理论上，Sender线程完全能够观测到一个尚未构造完成的KafkaProducer实例。当然，在构造对象时创建线程没有任何问题，但最好是不要同时启动它。

好了，我们言归正传。针对TCP连接何时创建的问题，目前我们的结论是这样的：**TCP连接是在创建KafkaProducer实例时建立的**。那么，我们想问的是，它只会在这个时候被创建吗？

当然不是！**TCP连接还可能在两个地方被创建：一个是在更新元数据后，另一个是在消息发送时**。为什么说是可能？因为这两个地方并非总是创建TCP连接。当Producer更新了集群的元数据信息之后，如果发现与某些Broker当前没有连接，那么它就会创建一个TCP连接。同样地，当要发送消息时，Producer发现尚不存在与目标Broker的连接，也会创建一个。

接下来，我们来看看Producer更新集群元数据信息的两个场景。

场景一：当Producer尝试给一个不存在的主题发送消息时，Broker会告诉Producer说这个主题不存在。此时Producer会发送METADATA请求给Kafka集群，去尝试获取最新的元数据信息。

场景二：Producer通过metadata.max.age.ms参数定期地去更新元数据信息。该参数的默认值是300000，即5分钟，也就是说不管集群那边是否有变化，Producer每5分钟都会强制刷新一次元数据以保证它是最及时的数据。

讲到这里，我们可以“挑战”一下社区对Producer的这种设计的合理性。目前来看，一个Producer默认会向集群的所有Broker都创建TCP连接，不管是否真的需要传输请求。这显然是没有必要

的。再加上Kafka还支持强制将空闲的TCP连接资源关闭，这就更显得多此一举了。

试想一下，在一个有着1000台Broker的集群中，你的Producer可能只会与其中的3～5台Broker长期通信，但是Producer启动后依次创建与这1000台Broker的TCP连接。一段时间之后，大约有995个TCP连接又被强制关闭。这难道不是一种资源浪费吗？很显然，这里是有改善和优化的空间的。

何时关闭TCP连接？

说完了TCP连接的创建，我们来说说它们何时被关闭。

Producer端关闭TCP连接的方式有两种：**一种是用户主动关闭；一种是Kafka自动关闭**。

我们先说第一种。这里的主动关闭实际上是广义的主动关闭，甚至包括用户调用kill -9主动“杀掉”Producer应用。当然最推荐的方式还是调用producer.close()方法来关闭。

第二种是Kafka帮你关闭，这与Producer端参数connections.max.idle.ms的值有关。默认情况下该参数值是9分钟，即如果在9分钟内没有任何请求“流过”某个TCP连接，那么Kafka会主动帮你把该TCP连接关闭。用户可以在Producer端设置connections.max.idle.ms=-1禁掉这种机制。一旦被设置成-1，TCP连接将成为永久长连接。当然这只是软件层面的“长连接”机制，由于Kafka创建的这些Socket连接都开启了keepalive，因此keepalive探活机制还是会遵守的。

值得注意的是，在第二种方式中，TCP连接是在Broker端被关闭的，但其实这个TCP连接的发起方是客户端，因此在TCP看来，这属于被动关闭的场景，即passive close。被动关闭的后果就是会产生大量的CLOSE_WAIT连接，因此Producer端或Client端没有机会显式地观测到此连接已被中断。

小结

我们来简单总结一下今天的内容。对最新版本的Kafka（2.1.0）而言，Java Producer端管理TCP连接的方式是：

1. KafkaProducer实例创建时启动Sender线程，从而创建与bootstrap.servers中所有Broker的TCP连接。
2. KafkaProducer实例首次更新元数据信息之后，还会再次创建与集群中所有Broker的TCP连接。
3. 如果Producer端发送消息到某台Broker时发现没有与该Broker的TCP连接，那么也会立即创建连接。
4. 如果设置Producer端connections.max.idle.ms参数大于0，则步骤1中创建的TCP连接会被自动关闭；如果设置该参数=-1，那么步骤1中创建的TCP连接将无法被关闭，从而成为“僵尸”连接。

开放讨论

对于今天我们“挑战”的社区设计，你有什么改进的想法吗？

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

柠檬C　3

应该可以用懒加载的方式，实际发送时再进行TCP连接吧，虽然这样第一次发送时因为握手的原因会稍慢一点

2019-07-02

KEEPUP　2

KafkaProducer 实例只是在首次更新元数据信息之后，创建与集群中所有 Broker 的 TCP 连接，还是每次更新之后都要创建？为什么要创建与所有 Broker 的连接呢？

2019-07-02

明翼　2

我的想法这样的，先用客户端去连配置的第一broker server，连不上就连接第二个，一旦连上了，就可以获取元数据信息了，这是创建produce实例时候动作，只发起一个TCP连接。再send时候发现没连接的再连接，至于其他的都还是很合理的。

2019-07-02

JoeyLi666 1

老师，最近使用kakfa，报了个异常：
Caused by: org.apache.kafka.common.KafkaException: Record batch for partition Notify-18 at offset 1803009 is invalid, cause: Record is corrupt (stored crc = 3092077514, computed crc = 2775748463)
kafka的数据还会损坏，不是有校验吗？

2019-07-04

作者回复

也可能是网络传输过程中出现的偶发情况，通常没有什么好的解决办法。。。

2019-07-05

南辕北辙 1

老师您好，我在本地分别用1.x版本和2.x版本的生产者去测试，为什么结果和老师的不一样呢。初始化KafkaProducer时，并没有与参数中设置的所有broker去建立连接，然后我sleep十秒，让sender线程有机会多运行会。但是还是没有看到去连接所有的broker。只有当运行到producer.send时才会有Initialize connection日志输出，以及由于metadata的needUpdate被更新成true，sender线程会开始有机会去更新metadata去连接broker（产生Initialize connection to node...for sending metadata request）。之前学习源码的时候，也只注意到二个地方去连接broker（底层方法initiateConnect，更新metadata时建立连接以及发送数据时判断待发送的node是否建立了连接）。老师是我哪里疏忽了吗，还是理解有问题。翻阅老师的书上TCP管理这块貌似没有过多的讲解，求老师指导。。

2019-07-03

作者回复

我不知道您这边是怎么实验的，但是我这边的确会创建新的TCP连接~~

2019-07-04

kursk.ye 1

试想一下，在一个有着 1000 台 Broker 的集群中，你的 Producer 可能只会与其中的 3～5 台 Broker 长期通信，但是 Producer 启动后依次创建与这 1000 台 Broker 的 TCP 连接。一段时间之后，大约有 995 个 TCP 连接又被强制关闭。这难道不是一种资源浪费吗？很显然，这里是有改善和优化的空间的。

这段不敢苟同。作为消息服务器中国，连接应该是种必要资源，所以部署时就该充分给予，而且创建连接会消耗CPU,用到时再创建不合适，我甚至觉得Kafka应该有连接池的设计。

另外最后一部分关于TCP关闭第二种情况，客户端到服务端没有关闭，只是服务端到客户端关闭了，tcp是四次断开，可以单方向关闭，另一方向继续保持连接

2019-07-02

作者回复

嗯嗯，欢迎不同意见。Kafka对于创建连接没有做任何限制。如果一开始就创建所有TCP连接，之后因为超时的缘故又关闭这些连接，当真正使用时再次创建，那么为什么不把创建时机后延到真正需要的时候呢？实际场景中将TCP连接设置为长连接的情形并不多见，因此我说这种设

计是可以改进的。

2019-07-03

诗泽

1

看来无论在bootstrap.servers中是否写全部broker 的地址接下来producer 还是会跟所有的broker 建立一次连接

2019-07-02

电光火石

0

谢谢老师。有几个问题请教一下：

1. producer连接是每个broker一个连接，跟topic没有关系是吗？（consumer也是这样是吗？）

2. 我们运维在所有的broker之前放了一个F5做负载均衡，但其实应该也没用，他会自动去获取kafka所有的broker，绕过这个F5，不知道我的理解是否正确？

3. 在线上我们有个kafka集群，大概200个topic，数据不是很均衡，有的一天才十几m，有的一天500G，我们是想consumer读取所有的topic，然后后面做分发，但是consumer会卡死在哪，也没有报错，也没有日志输出，不知道老师有没有思路可能是什么原因？

谢谢了！

2019-07-07

作者回复

1. 也不能说没有关系。客户端需要和topic下各个分区leader副本所在的broker进行连接的

2. 嗯嗯，目前客户端是直连broker的

3. 光看描述想不出具体的原因。有可能是频繁rebalance、long GC、消息corrupted或干脆就是一个已知的bug

2019-07-08

燃烧的M豆

0

c 包在v1.0.0 已经对这个问题采取了对策。所有使用基础 c 包 librdkafka 都因此受益

Sparse connections
In previous releases librdkafka would maintain open connections to all brokers in the cluster and the bootstrap servers.

With this release librdkafka now connects to a single bootstrap server to retrieve the full broker list, and then connects to the brokers it needs to communicate with: partition leaders, group coordinators, etc.

For large scale deployments this greatly reduces the number of connections between clients and brokers, and avoids the repeated idle connection closes for unused connections.

Sparse connections is on by default (recommended setting), the old behavior of connecting to all brokers in the cluster can be re-enabled by setting enable.sparse.connections=false.

See Sparse connections in the manual for more information.

Original issue librdkafka #825.

为啥 java 目前还是这样或者一已经开始动手修改了？- -有点无法理解。

2019-07-04

流浪的阳光 0

老师、各位专家好，请问kafca适合做两个系统之间的转账处理吗？

另外，请问kafca的使用案例中，最多支持过什么数量级的消费者和生产者？

2019-07-03

ban 0

如果多个线程都使用一个KafkaProducer实例，缓冲器被填满的速度会变快。
老师看评论这句话不太理解，多个线程共用一个实例，没有再new新的实例，为什么缓冲器很快填满，不是利用原有的实例的吗。

2019-07-03

作者回复

一个KafkaProducer实例会开辟一块内存缓冲区，如果多个线程共用这部分buffer，自然buffer被填满的速度就会快很多。我是这个意思。。。。

2019-07-04

Li Shunduo 0

请问Producer会和同一台broker建立多个TCP连接吗？

2019-07-03

作者回复

有可能，可能用于不同的目的，通常最终会收敛成一个

2019-07-04

Geek_Sue 0

胡老师，我想请教一下。
connections.max.idle.ms这个值如果设置为-1，按照您文章里所说，KafkaProducer会创建bootstrap.servers中全部tcp连接，如果是1000个，那么就是说这1000个连接永远不会关闭了？

2019-07-03

作者回复

如果 broker端和clients端的connections.max.idle.ms都是-1，就真是永不关闭了。

2019-07-04

飞翔 0

KafkaProducer 实例首次更新元数据信息之后，还会再次创建与集群中所有 Broker 的 TCP 连接。 这里如果我有1000个broker，就建立1000个链接？ 为啥不是只根据元信息只建立有我要发的主题的broke的链接？

2019-07-03

作者回复

嗯嗯， 目前不是这样设计的。

2019-07-03

rm -rf

0

您好，老师，你说的第二种关闭producer的方法是一个被动关闭，发起方是客户端。

1. 这个客户端指的是producer客户端？所以broker此时会可能出现close wait，是这个意思吗？

2. 还有这句“因此 Producer 端或 Client 端没有机会显式地.”也不是很理解，producer端和client端不是一个意思吗？

2019-07-02

作者回复

1. 是这个意思

2. 我是想说consumer端也是这个原理。因此加上了clients

2019-07-03

曾轼麟

0

老师下面就有一个问题，KafkaProducer是建议创建实例后复用，像连接池那样使用，还是建议每次发送构造一个实例？听完这讲后感觉哪个都不合理，每次new会有很大的开销，但是一次new感觉又有僵尸连接，KafkaProducer适合池化吗？还是建议单例？

2019-07-02

作者回复

KafkaProducer是线程安全的，复用是没有问题的。只是要监控内存缓冲区的使用情况。毕竟如果多个线程都使用一个KafkaProducer实例，缓冲器被填满的速度会变快。

2019-07-03

趙衍

0

老师好，看了今天的文章我有几个问题：

1.Kafka的元数据信息是存储在zookeeper中的，而Producer是通过broker来获取元数据信息的，那么这个过程是否是这样的，Producer向Broker发送一个获取元数据的请求给Broker，之后Broker再向zookeeper请求这个信息返回给Producer?

2.如果Producer在获取完元数据信息之后要和所有的Broker建立连接，那么假设一个Kafka集群中有1000台Broker，对于一个只需要与5台Broker交互的Producer，它连接池中的链接数量是不是从1000->5->1000->5?这样不是显得非常得浪费连接池资源？

2019-07-02

作者回复

1. 集群元数据持久化在ZooKeeper中，同时也缓存在每台Broker的内存中，因此不需要请求ZooKeeper

2. 就我个人认为，的确有一些不高效。所以我说这里有优化的空间的。

2019-07-03

吴宇晨

0

老师你好，kafka更新元数据的方法只有每5分钟的轮训吗，如果有监控zk节点之类的，是不是可以把轮询元数据时间调大甚至取消

2019-07-02

作者回复

Clients端有个参数metadata.max.age.ms强制刷新元数据，默认的确是5分钟。新版本Clients不会与ZooKeeper交互，所以感觉和ZooKeeper没什么关系。。。

2019-07-03

lmtoo 0

Producer应该只跟Controller节点更新元数据，以及相关的topic机器交互数据，而不应该跟所有的机器创建连接；有个疑问，当 Producer 更新了集群的元数据信息之后，如果发现与某些broker没有连接，就去创建，为什么会去创建跟producer无关的连接？

2019-07-02

作者回复

有可能也会连接这些Broker的。Clients获取到集群元数据后知道了集群所有Broker的连接信息。下次再次获取元数据时，它会选择一个负载最少的Broker进行连接。如果发现没有连接也会创建Socket，但其实它并不需要向这个Broker发送任何消息。

2019-07-03

张庆 0

主要的资源浪费应该是在第一次获取元数据的时候创建所有的连接，应该是这个地方可以做一些优化吧，可以做一个最小初始化数量，从元数据中随机获取配置的最少数据，然后进行初始化。然后向Broker发送消息的时候在去判断，如果没有连接就创建连接，这样应该可以折中一下吧。

2019-07-02