Kafka客户端是如何找到 leader 分区的

在正常情况下，Kafka中的每个Topic都会有很多个分区，每个分区又会存在多个副本。在这些副本中，存在一个leader分区，而剩下的分区叫做 follower，所有对分区的读写操作都是对leader分区进行的。所以当我们向Kafka写消息或者从Kafka读取消息的时候，必须先找到对应分区的Leader及其所在的Broker地址，这样才可以进行后续的操作。本文将要介绍的就是 Kafka 是如何找到 leader 分区的。

我们知道， Kafka 是使用 Scala 语言 编写的， 但是其支持很多语言的客户端，包括：C/C++、PHP、Go以及Ruby等等（参见https://cwiki.apache.org/confluence/display/KAFKA/Clients ）。这是为什么呢？这是因为 Kafka 内部实现了一套基于TCP层的协议，只要使用这种协议与Kafka进行通信，就可以使用很多语言来操作Kafka。

目前 Kafka 内部支持多达30多种协议，本文介绍的 Kafka 客户端是如何找到 leader 分区就涉及到 Kafka 内部的 Metadata 协议。Metadata 协议主要解决以下四种问题：

- Kafka中存在哪些主题？
- 每个主题有几个分区？
- Leader分区所在的broker地址及端口？
- 每个broker的地址及端口是多少？

客户端只需要构造相应的请求，并发送到Broker端，即可获取到上面四个问题的答案。整个过程如下：

- 客户端构造相应的请求
- 客户端将请求发送到Broker端
- Broker端接收到请求处理，并将结果发送到客户端。

Metadata 请求协议（v0-v3版本）如下：

```
TopicMetadataRequest => [TopicNames]
  TopicNames => string
```

客户端只需要构造一个 TopicMetadataRequest ，里面包括我们需要查询主题的名字（TopicNames）；当然，我们可以一次查询多个主题，只需要将这些主题放进List里面即可。同时，我们还可以不传入任何主题的名字，这时候 Kafka 将会把内部所有的主题相关的信息发送给客户端。

目前 Metadata 请求协议存在五个版本，v0-v3版本格式一致。但是这些协议存在一个问题：当Kafka 服务器端将 auto.create.topics.enable 参数设置为 ture
时，如果我们查询的主题不存在，Kafka
将会自动创建这个主题，这很可能不是我们想要的结果。所以，基于这个问题，到了 Metadata请求协议第五版，格式已经变化了，如下：

```
Metadata Request (Version: 4) => [TopicNames] allow_auto_topic_creation
  TopicNames => STRING
  allow_auto_topic_creation => BOOLEAN
```

我们可以指定 allow_auto_topic_creation 参数来告诉 Kafka
是否需要在主题不存在的时候创建，这时候控制权就在我们了。

Kafka 的 Broker 收到客户端的请求处理完之后，会构造一个
TopicMetadataResponse，并发送给客户端。TopicMetadataResponse 协议的格式如下：

```
MetadataResponse => [Broker][TopicMetadata]
  Broker => NodeId Host Port  (any number of brokers may be returned)
    NodeId => int32
    Host => string
    Port => int32
  TopicMetadata => TopicErrorCode TopicName [PartitionMetadata]
    TopicErrorCode => int16
  PartitionMetadata => PartitionErrorCode PartitionId Leader Replicas Isr
    PartitionErrorCode => int16
    PartitionId => int32
    Leader => int32
    Replicas => [int32]
    Isr => [int32]
```

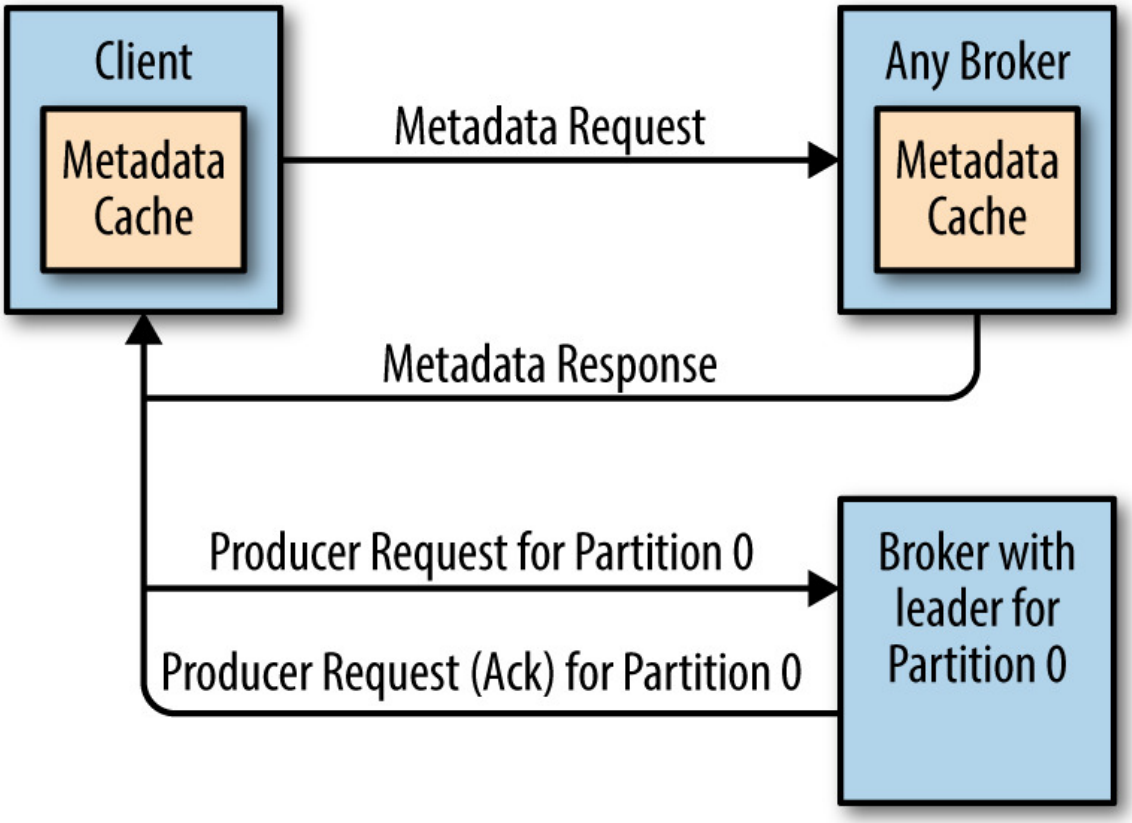
可以看到，相应协议里面包含了每个分区的 Leader、Replicas 以及 Isr 信息，同时还包括了Kafka集群所有Broker的信息。如果处理出现了问题，会出现相应的错误信息码，主要包括下面几个：

```
UnknownTopic (3)
LeaderNotAvailable (5)
InvalidTopic (17)
TopicAuthorizationFailed (29)
```

而且，Metadata 协议是目前唯一一个可以向任何 Broker 发送的协议。因为任何一个 Broker 在启动之后会存储这些Metadata信息的。而且，Kafka 提供的客户端在获取到 Metadata 信息之后也会将它存储到内存中的。并且在以下几种情况会更新已经缓存下来的 Metadata 信息：

- 在 meta-data.max.age.ms 参数配置的时间过期之后；
- 在往Kafka发送请求是收到 Not a Leader 异常。

以上两种情况 Kafka提供的客户端会自动再发送一次 Metadata 请求，这样就可以获取到更新的信息。整个过程如下：



如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

好了，说了半天的，我们来看看程序里面如何构造 TopicMetadataRequest 以及处理 TopicMetadataResponse。

```
package com.iteblog.kafka

import kafka.api.TopicMetadataRequest._
import kafka.api.{TopicMetadataRequest, TopicMetadataResponse}
import kafka.consumer.SimpleConsumer
```

```
//////////////////////////////////////////////////////////////////////
 User: 过往记忆
 Date: 2017年07月28日
 Time: 22:12:43
 bolg: https://www.iteblog.com
 本文地址：https://www.iteblog.com/archives/2215
 过往记忆博客，专注于hadoop、hive、spark、shark、flume的技术博客，大量的干货
 过往记忆博客微信公共帐号：iteblog_hadoop
//////////////////////////////////////////////////////////////////////

object MetaDataDemo {
  def main(args: Array[String]): Unit = {
    val consumer = new SimpleConsumer("1.iteblog.com", 9092, 50, 1024 * 4, DefaultClientId)

    val req: TopicMetadataRequest = new TopicMetadataRequest(CurrentVersion, 0, DefaultClie
ntId, List("iteblog_hadoop"))
    val resp: TopicMetadataResponse = consumer.send(req)

    println("Broker Infos:")
    println(resp.brokers.mkString("₩n₩t"))
    val metadata = resp.topicsMetadata
    metadata.foreach { topicMetadata =>
      val partitionsMetadata = topicMetadata.partitionsMetadata
      partitionsMetadata.foreach { partitionMetadata =>
        println(s"partitionId=${partitionMetadata.partitionId}₩n₩tleader=${partitionMetadata.l
eader}" +
          s"₩n₩tisr=${partitionMetadata.isr}₩n₩treplicas=${partitionMetadata.replicas}")
      }
    }
  }
}
```

TopicMetadataRequest 是通过 SimpleConsumer 的 send 方法发送的，其返回的是 TopicMetadataResponse ，其中就包含了我们需要的信息。 运行上面的程序输出如下：

```
Broker Infos:
 id:5,host:5.iteblog.com,port:9092
 id:1,host:1.iteblog.com,port:9092
 id:6,host:6.iteblog.com,port:9092
 id:2,host:2.iteblog.com,port:9092
 id:7,host:7.iteblog.com,port:9092
 id:3,host:3.iteblog.com,port:9092
 id:8,host:8.iteblog.com,port:9092
```

id:4,host:4.iteblog.com,port:9092
partitionId=0
leader=Some(id:1,host:1.iteblog.com,port:9092)
isr=Vector(id:1,host:1.iteblog.com,port:9092)
replicas=Vector(id:1,host:1.iteblog.com,port:9092, id:8,host:8.iteblog.com,port:9092)
partitionId=1
leader=Some(id:2,host:2.iteblog.com,port:9092)
isr=Vector(id:2,host:2.iteblog.com,port:9092, id:1,host:1.iteblog.com,port:9092)
replicas=Vector(id:2,host:2.iteblog.com,port:9092, id:1,host:1.iteblog.com,port:9092)
partitionId=2
leader=Some(id:3,host:3.iteblog.com,port:9092)
isr=Vector(id:3,host:3.iteblog.com,port:9092, id:2,host:2.iteblog.com,port:9092)
replicas=Vector(id:3,host:3.iteblog.com,port:9092, id:2,host:2.iteblog.com,port:9092)
partitionId=3
leader=Some(id:4,host:4.iteblog.com,port:9092)
isr=Vector(id:4,host:4.iteblog.com,port:9092, id:3,host:3.iteblog.com,port:9092)
replicas=Vector(id:4,host:4.iteblog.com,port:9092, id:3,host:3.iteblog.com,port:9092)
partitionId=4
leader=Some(id:5,host:5.iteblog.com,port:9092)
isr=Vector(id:5,host:5.iteblog.com,port:9092, id:4,host:4.iteblog.com,port:9092)
replicas=Vector(id:5,host:5.iteblog.com,port:9092, id:4,host:4.iteblog.com,port:9092)
partitionId=5
leader=Some(id:6,host:6.iteblog.com,port:9092)
isr=Vector(id:6,host:6.iteblog.com,port:9092, id:5,host:5.iteblog.com,port:9092)
replicas=Vector(id:6,host:6.iteblog.com,port:9092, id:5,host:5.iteblog.com,port:9092)
partitionId=6
leader=Some(id:7,host:7.iteblog.com,port:9092)
isr=Vector(id:6,host:6.iteblog.com,port:9092, id:7,host:7.iteblog.com,port:9092)
replicas=Vector(id:7,host:7.iteblog.com,port:9092, id:6,host:6.iteblog.com,port:9092)
partitionId=7
leader=Some(id:8,host:8.iteblog.com,port:9092)
isr=Vector(id:8,host:8.iteblog.com,port:9092)
replicas=Vector(id:8,host:8.iteblog.com,port:9092, id:7,host:7.iteblog.com,port:9092)
partitionId=8
leader=Some(id:1,host:1.iteblog.com,port:9092)
isr=Vector(id:2,host:2.iteblog.com,port:9092, id:1,host:1.iteblog.com,port:9092)
replicas=Vector(id:1,host:1.iteblog.com,port:9092, id:2,host:2.iteblog.com,port:9092)
partitionId=9
leader=Some(id:2,host:2.iteblog.com,port:9092)
isr=Vector(id:3,host:3.iteblog.com,port:9092, id:2,host:2.iteblog.com,port:9092)
replicas=Vector(id:2,host:2.iteblog.com,port:9092, id:3,host:3.iteblog.com,port:9092)
partitionId=10
leader=Some(id:3,host:3.iteblog.com,port:9092)
isr=Vector(id:4,host:4.iteblog.com,port:9092, id:3,host:3.iteblog.com,port:9092)
replicas=Vector(id:3,host:3.iteblog.com,port:9092, id:4,host:4.iteblog.com,port:9092)
partitionId=11

```
 leader=Some(id:6,host:6.iteblog.com,port:9092)
 isr=Vector(id:6,host:6.iteblog.com,port:9092, id:1,host:1.iteblog.com,port:9092)
 replicas=Vector(id:6,host:6.iteblog.com,port:9092, id:1,host:1.iteblog.com,port:9092)
partitionId=12
 leader=Some(id:7,host:7.iteblog.com,port:9092)
 isr=Vector(id:7,host:7.iteblog.com,port:9092, id:2,host:2.iteblog.com,port:9092)
 replicas=Vector(id:7,host:7.iteblog.com,port:9092, id:2,host:2.iteblog.com,port:9092)
partitionId=13
 leader=Some(id:8,host:8.iteblog.com,port:9092)
 isr=Vector(id:8,host:8.iteblog.com,port:9092, id:3,host:3.iteblog.com,port:9092)
 replicas=Vector(id:8,host:8.iteblog.com,port:9092, id:3,host:3.iteblog.com,port:9092)
partitionId=14
 leader=Some(id:1,host:1.iteblog.com,port:9092)
 isr=Vector(id:1,host:1.iteblog.com,port:9092, id:4,host:4.iteblog.com,port:9092)
 replicas=Vector(id:1,host:1.iteblog.com,port:9092, id:4,host:4.iteblog.com,port:9092)
partitionId=15
 leader=Some(id:2,host:2.iteblog.com,port:9092)
 isr=Vector(id:2,host:2.iteblog.com,port:9092, id:5,host:5.iteblog.com,port:9092)
 replicas=Vector(id:2,host:2.iteblog.com,port:9092, id:5,host:5.iteblog.com,port:9092)
partitionId=16
 leader=Some(id:3,host:3.iteblog.com,port:9092)
 isr=Vector(id:3,host:3.iteblog.com,port:9092, id:7,host:7.iteblog.com,port:9092)
 replicas=Vector(id:3,host:3.iteblog.com,port:9092, id:7,host:7.iteblog.com,port:9092)
partitionId=17
 leader=Some(id:4,host:4.iteblog.com,port:9092)
 isr=Vector(id:4,host:4.iteblog.com,port:9092, id:8,host:8.iteblog.com,port:9092)
 replicas=Vector(id:4,host:4.iteblog.com,port:9092, id:8,host:8.iteblog.com,port:9092)
partitionId=18
 leader=Some(id:5,host:5.iteblog.com,port:9092)
 isr=Vector(id:5,host:5.iteblog.com,port:9092, id:1,host:1.iteblog.com,port:9092)
 replicas=Vector(id:5,host:5.iteblog.com,port:9092, id:1,host:1.iteblog.com,port:9092)
partitionId=19
 leader=Some(id:6,host:6.iteblog.com,port:9092)
 isr=Vector(id:6,host:6.iteblog.com,port:9092, id:2,host:2.iteblog.com,port:9092)
 replicas=Vector(id:6,host:6.iteblog.com,port:9092, id:2,host:2.iteblog.com,port:9092)
partitionId=20
 leader=Some(id:7,host:7.iteblog.com,port:9092)
 isr=Vector(id:7,host:7.iteblog.com,port:9092, id:3,host:3.iteblog.com,port:9092)
 replicas=Vector(id:7,host:7.iteblog.com,port:9092, id:3,host:3.iteblog.com,port:9092)
partitionId=21
 leader=Some(id:8,host:8.iteblog.com,port:9092)
 isr=Vector(id:8,host:8.iteblog.com,port:9092, id:4,host:4.iteblog.com,port:9092)
 replicas=Vector(id:8,host:8.iteblog.com,port:9092, id:4,host:4.iteblog.com,port:9092)
partitionId=22
 leader=Some(id:1,host:1.iteblog.com,port:9092)
 isr=Vector(id:1,host:1.iteblog.com,port:9092, id:5,host:5.iteblog.com,port:9092)
```

replicas=Vector(id:1,host:1.iteblog.com,port:9092, id:5,host:5.iteblog.com,port:9092)

上面的输出就可以看到各个分区的leader所在机器、isr以及所有replicas等信息。有一点我们需要注意，因为目前存在多个版本的 Metadata
请求协议，我们可以使用低版本的协议与高版本的Kafka集群进行通信，因为高版本的 Kafka 能够支持低版本的 Metadata 请求协议；但是我们不能使用高版本的 Metadata
请求协议与低版本的 Kafka 通信。