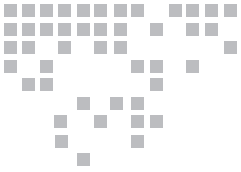

Chapter 2 第 2 章

# Kafka 的架构

从本章起将详细探讨 Kafka 内部的实现原理：其中包括 Kafka 的基本组成、Kafka 的拓扑结构以及 Kafka 内部的通信协议。Kafka 内部的通信协议是建立在 Kafka 的拓扑结构之上的，而 Kafka 的拓扑结构是由 Kafka 的基本模块所组成的，因此本章是其他剩余章节的基础，是理解 Kafka 的关键。

## 2.1 Kafka 的基本组成

在 Kafka 集群中生产者将消息发送给以 Topic 命名的消息队列 Queue 中，消费者订阅发往以某个 Topic 命名的消息队列 Queue 中的消息。其中 Kafka 集群由若干个 Broker 组成，Topic 由若干个 Partition 组成，每个 Partition 里面的消息通过 Offset 来获取。

- Broker：一台 Kafka 服务器就是一个 Broker，一个集群由多个 Broker 组成，一个 Broker 可以容纳多个 Topic，Broker 和 Broker 之间没有 Master 和 Standby 的概念，它们之间的地位基本是平等的。
- Topic：每条发送到 Kafka 集群的消息都属于某个主题，这个主题就称为 Topic。物理上不同 Topic 的消息分开存储，逻辑上一个 Topic 的消息虽然保存在一个或多个 Broker 上，但是用户只需指定消息的主题 Topic 即可生产或消费数据而不需要去关心数据存放在何处。
- Partition：为了实现可扩展性，一个非常大的 Topic 可以被分为多个 Partition，从而分布到多台 Broker 上。Partition 中的每条消息都会被分配一个自增 Id（Offset）。Kafka 只保证按一个 Partition 中的顺序将消息发送给消费者，但是不保证单个 Topic 中的

多个 Partition 之间的顺序。

- Offset：消息在 Topic 的 Partition 中的位置，同一个 Partition 中的消息随着消息的写入，其对应的 Offset 也自增，其内部实现原理如图 2-1 所示。

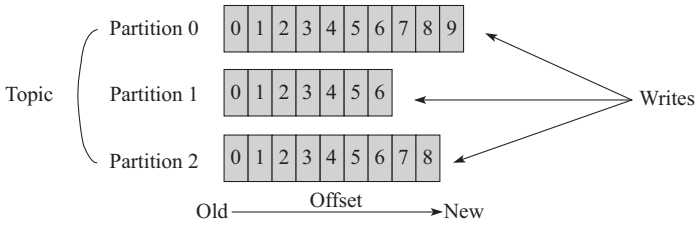


图 2-1　Topic、Partition 和 Offset 解析

- Replica：副本。Topic 的 Partition 含有 N 个 Replica，N 为副本因子。其中一个 Replica 为 Leader，其他都为 Follower，Leader 处理 Partition 的所有读写请求，与此同时，Follower 会定期地去同步 Leader 上的数据。
- Message：消息，是通信的基本单位。每个 Producer 可以向一个 Topic（主题）发布一些消息。
- Producer：消息生产者，即将消息发布到指定的 Topic 中，同时 Producer 也能决定此消息所属的 Partition：比如基于 Round-Robin（轮询）方式或者 Hash（哈希）方式等一些算法。
- Consumer：消息消费者，即向指定的 Topic 获取消息，根据指定 Topic 的分区索引及其对应分区上的消息偏移量来获取消息。
- Consumer Group：消费者组，每个 Consumer 属于一个 Consumer Group；反过来，每个 Consumer Group 中可以包含多个 Consumer。如果所有的 Consumer 都具有相同的 Consumer Group，那么消息将会在 Consumer 之间进行负载均衡。也就是说一个 Partition 中的消息只会被相同 Consumer Group 中的某个 Consumer 消费，每个 Consumer Group 消息消费是相互独立的。如果所有的 Consumer 都具有不同的 Consumer Group，则消息将会被广播给所有的 Consumer。Producer、Consumer 和 Consumer Group 之间的关系如图 2-2 所示。

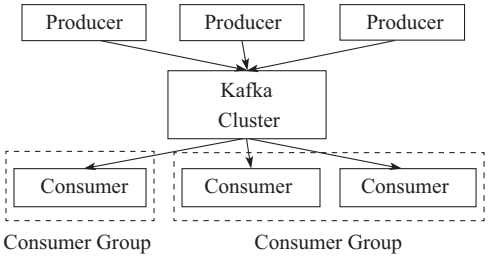


图 2-2　Producer、Consumer 和 Consumer Group 三者的关系

- Zookeeper：存放 Kafka 集群相关元数据的组件。在 Zookeeper 集群中会保存 Topic 的状态信息，例如分区的个数、分区的组成、分区的分布情况等；保存 Broker 的状态信息；保存消费者的消费信息等。通过这些信息，Kafka 很好地将消息生产、消息存储、消息消费的过程结合起来。

## 2.2 Kafka 的拓扑结构

一个典型的 Kafka 集群的拓扑结构如图 2-3 所示。

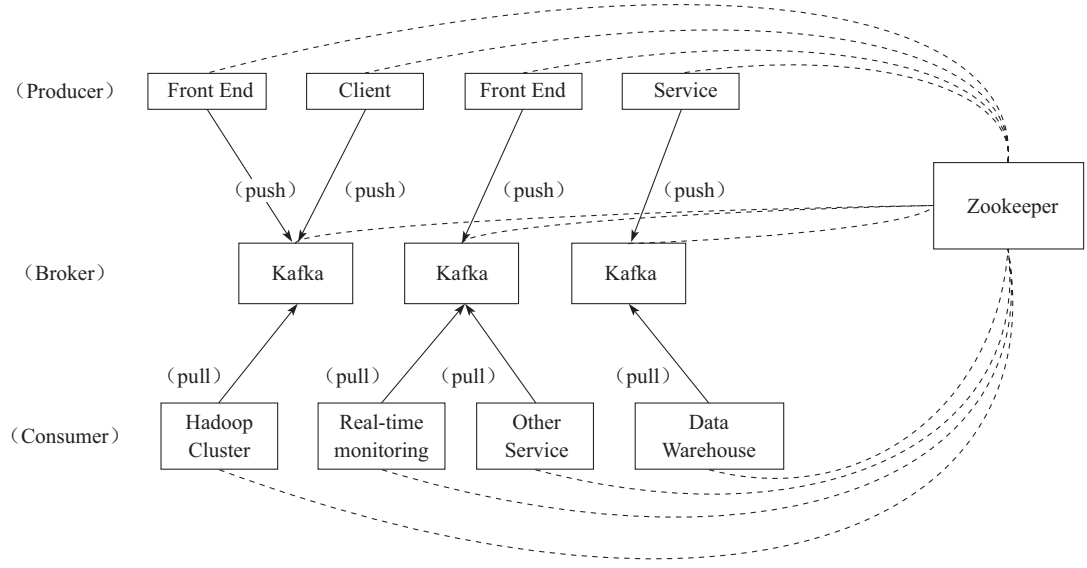


图 2-3 Kafka 拓扑结构

一个典型的 Kafka 集群中包含若干个 Producer（可以是某个模块下发的 Command，或者是 Web 前端产生的 Page View，或者是服务器日志，系统 CPU、Memory 等），若干个 Broker（Kafka 集群支持水平扩展，一般 Broker 数量越多，整个 Kafka 集群的吞吐率也就越高），若干个 Consumer Group，以及一个 Zookeeper 集群。Kafka 通过 Zookeeper 管理集群配置。Producer 使用 Push 模式将消息发布到 Broker 上，Consumer 使用 Pull 模式从 Broker 上订阅并消费消息。

一个简单的消息发送流程如图 2-4 所示。

1）Producer 根据指定的路由方法（Round-Robin、Hash 等），将消息 Push 到 Topic 的某个 Partition 里面。

2）Kafka 集群接收到 Producer 发过来的消息后，将其持久化到硬盘，并保留消息指定时长（可配置），而不关注消息是否被消费。

3）Consumer 从 Kafka 集群 Pull 数据，并控制获取消息的 Offset。

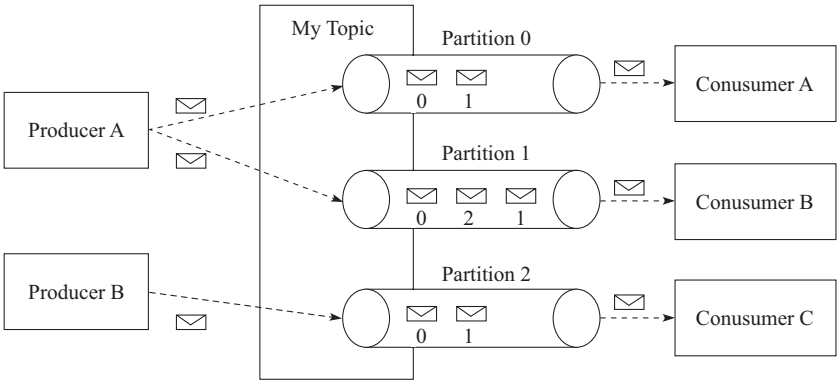


图 2-4　消息发送的简易流程

## 2.3　Kafka 内部的通信协议

Kafka 内部各个 Broker 之间的角色并不是完全相等的，Broker 内部负责管理分区和副本状态以及异常情况下分区的重新分配等这些功能的模块称为 KafkaController。每个 Kafka 集群中有且只有 1 个 Leader 状态的 KafkaController，当 Leader 状态的 KafkaController 出现异常时，其余的 Standby 状态下的 KafkaController 会通过 Zookeeper 选举出又一个 Leader 状态的 KafkaController，因此 Broker 又可根据 KafkaController 模块的状态进行进一步的细分。

在一个正常运行的 Kafka 集群中，生产者和 Broker 之间，消费者和 Broker 之间，Broker 和 Broker 之间不断地在进行不同网络协议的交互，正是由于各个组件不断地进行网络交互才维持了整个集群稳定的运行。其主要的网络通信协议如图 2-5 所示。

利用命令 bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 2 --partitions 3 --topic my_topic 在图 2-5 中创建了以 my_topic 命名的消息主题，其分区个数为 3 个，每个分区的副本因子为 2，其中分区 0 的第一个副本和分区 2 的第二个副本位于 Broker 1 上，分别以 /my_topic/partition-0/replica0 和 /my_topic/partition-2/replica1 表示；分区 1 的第一个副本和分区 0 的第二个副本位于 Broker 2 上，分别以 /my_topic/partition-1/replica0 和 /my_topic/partition-0/replica1 表示；分区 2 的第一个副本和分区 1 的第二个副本位于 Broker 3 上，分别以 /my_topic/partition-2/replica0 和 /my_topic/partition-1/replica1 表示。假设每个分区的 replica0 为对应分区的副本 Leader，其余为 Follower，并且此时 Broker1 的 KafkaController 模块状态为 Leader，其余为 Standy。在此基础上将分两个维度来阐述图 2-5 所表达的意思：

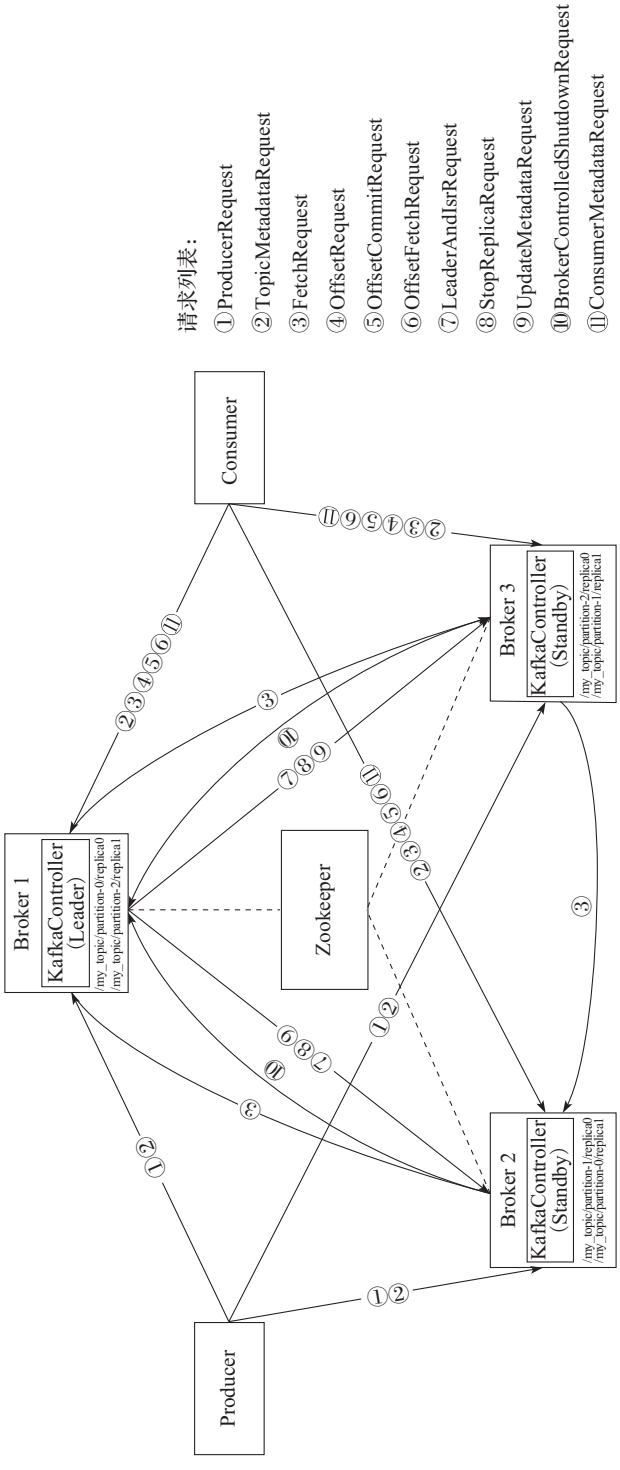


图 2-5 Kafka 内部的通信协议

**维度一：通信协议详情**

- ProducerRequest：生产者发送消息的请求，生产者将消息发送至 Kafka 集群中的某个 Broker，Broker 接收到此请求后持久化此消息并更新相关元数据信息。
- TopicMetadataRequest：获取 Topic 元数据信息的请求，无论是生产者还是消费者都需要通过此请求来获取感兴趣的 Topic 的元数据。
- FetchRequest：消费者获取感兴趣 Topic 的某个分区的消息的请求，除此之外，分区状态为 Follower 的副本也需要利用此请求去同步分区状态为 Leader 的对应副本数据。
- OffsetRequest：消费者发送至 Kafka 集群来获取感兴趣 Topic 的分区偏移量的请求，通过此请求可以获知当前 Topic 所有分区在不同时间段的偏移量详情。
- OffsetCommitRequest：消费者提交 Topic 被消费的分区偏移量信息至 Broker，Broker 接收到此请求后持久化相关偏移量信息。
- OffsetFetchRequest：消费者发送获取提交至 Kafka 集群的相关 Topic 被消费的详细信息，和 OffsetCommitRequest 相互对应。
- LeaderAndIsrRequest：当 Topic 的某个分区状态发生变化时，处于 Leader 状态的 KafkaController 发送此请求至相关的 Broker，通知其做出相应的处理。
- StopReplicaRequest：当 Topic 的某个分区被删除或者下线的时候，处于 Leader 状态的 KafkaController 发送此请求至相关的 Broker，通知其做出相应的处理。
- UpdateMetadataRequest：当 Topic 的元数据信息发生变化时，处于 Leader 状态的 KafkaController 发送此请求至相关的 Broker，通知其做出相应的处理。
- BrokerControlledShutdownRequest：当 Broker 正常下线时，发生此请求至处于 Leader 状态的 KafkaController。
- ConsumerMetadataRequest：获取保存特定 Consumer Group 消费详情的分区信息。

**维度二：通信协议交互**

- Producer 和 Kafka 集群：Producer 需要利用 ProducerRequest 和 TopicMetadataRequest 来完成 Topic 元数据的查询、消息的发送。
- Consumer 和 Kafka 集群：Consumer 需要利用 TopicMetadataRequest 请求、FetchRequest 请求、OffsetRequest 请求、OffsetCommitRequest 请求、OffsetFetchRequest 请求和 ConsumerMetadataRequest 请求来完成 Topic 元数据的查询、消息的订阅、历史偏移量的查询、偏移量的提交、当前偏移量的查询。
- KafkaController 状态为 Leader 的 Broker 和 KafkaController 状态为 Standby 的 Broker：KafkaController 状态为 Leader 的 Broker 需要利用 LeaderAndIsrRequest 请求、StopReplicaRequest 请求、UpdateMetadataRequest 请求来完成对 Topic 的管理；KafkaController 状态为 Standby 的 Broker 需要利用 BrokerControlledShutdownRequest 请求来通知 KafkaController 状态为 Leader 的 Broker 自己的下线动作。

- Broker 和 Broker 之间：Broker 相互之间需要利用 FetchRequest 请求来同步 Topic 分区的副本数据，这样才能使 Topic 分区各副本数据实时保持一致。

## 2.4 本章小结

本章主要讲解了有关 Kafka 消息系统的基本组成、拓扑结构和通信协议。Kafka 消息系统主要由若干个 Broker Server 组成，其中生产者向指定的 Topic 发送消息，消费者订阅发往某个 Topic 的消息。因此 Kafka 消息系统内部的通信协议主要区分为三种：生产者和 Broker 之间，消费者和 Broker 之间以及 Broker 和 Broker 之间。掌握 Kafka 消息系统内部的通信协议是理解 Kafka 内部实现原理的关键，Kafka 消息系统内部的各个模块都是通过其内部通信协议相互联系起来的。