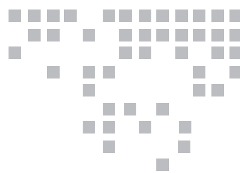
互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



第 3 章 Chapter 3

Broker 概述

回顾第 2 章 Kafka 的架构，Kafka 集群是由若干个 Broker 组成的，Broker 和 Broker 之间，Broker 和生产者之间，Broker 和消费者之间都存在不同的交互。因此本章将从 Broker 的启动脚本开始描述 Broker 的启动过程，以及基本阐述启动之后 Broker 内部存在的各个功能模块，包括 SocketServer、KafkaRequestHandlerPool、LogManager、ReplicaManager、OffsetManager、KafkaScheduler、KafkaApis、KafkaHealthcheck 和 TopicConfigManager 九大基本模块以及 KafkaController 集群控制管理模块。

3.1 Broker 的启动

Kafka 的安装包目录结构如图 3-1 所示。

```
drwxr-xr-x 3 hadoop hadoop  4096 Jan 29  2015 bin
drwxr-xr-x 2 hadoop hadoop  4096 Nov 16 21:38 config
drwxr-xr-x 2 hadoop hadoop  4096 Jan 29  2015 libs
-rw-r--r-- 1 hadoop hadoop 11358 Jan 29  2015 LICENSE
drwxrwxr-x 2 hadoop hadoop  4096 Mar 21 12:27 logs
-rw-r--r-- 1 hadoop hadoop   162 Jan 29  2015 NOTICE
```

图 3-1　Kafka 安装包的目录结构

bin 目录存放的是 Kafka 提供的管理工具，其中包括 Broker 的启动脚本；config 目录存放的是 Broker 的配置文件；libs 目录存放的是相关的 jar 包。

进入 bin 目录，执行以下命令后台启动 Broker：

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章


```
nohup ./bin/kafka-server-start.sh config/server.properties &
```

可见 Broker 是通过脚本 kafka-server-start.sh 调用起来的，继续查看这个脚本的内容，如下所示：

```
if [ $# -lt 1 ];
then
  echo "USAGE: $0 [-daemon] server.properties"
  exit 1
fi
base_dir=$(dirname $0)
//省略中间步骤
exec $base_dir/kafka-run-class.sh $EXTRA_ARGS kafka.Kafka $@
```

最终执行的是 kafka.Kafka 这个类，即内部 package kafka 里面的 Kafka 类。查看源码中关于这个类的详情，如下所示：

```
object Kafka extends Logging {
  def main(args: Array[String]): Unit = {
    if (args.length != 1) {
      println("USAGE: java [options] %s
        server.properties".format(classOf[KafkaServer].getSimpleName()))
      System.exit(1)
    }
    try {
      val props = Utils.loadProps(args(0))
      val serverConfig = new KafkaConfig(props)
      KafkaMetricsReporter.startReporters(serverConfig.props)
      val kafkaServerStartable = new KafkaServerStartable(serverConfig)
      Runtime.getRuntime().addShutdownHook(new Thread() {
        override def run() = {
          kafkaServerStartable.shutdown
        }
      })
      kafkaServerStartable.startup
      kafkaServerStartable.awaitShutdown
    }
    catch {
      case e: Throwable => fatal(e)
    }
    System.exit(0)
  }
}
```

程序在 kafkaServerStartable.awaitShutdown 停住，如果继续走下去，那么 Broker 就退出了。

注意 在 Scala 语言中，class 类对象中不可有静态变量和静态方法，但是提供了“伴生对象”的功能：在和类的同一个文件中定义同名的 object 对象，所有的 main 方法都必须在 object 中被调用，来提供程序的主入口，十分简单。

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

其中上面的 kafkaServerStartable 封装了 KafkaServer，最终执行 startup 的是 KafkaServer，如下代码所示：

```
class KafkaServerStartable(val serverConfig: KafkaConfig) extends Logging {
  private val server = new KafkaServer(serverConfig)
  def startup() {
    try {
      server.startup()
      AppInfo.registerInfo()
    }
    catch {
      case e: Throwable =>
        fatal("Fatal error during KafkaServerStartable startup. Prepare to shutdown", e)
        System.exit(1)
    }
  }
  ......
}
```

在这里终于见到了 Broker 启动过程中最关键的类 KafkaServer，接下来将围绕 KafkaServer 讲解里面的模块组成。

3.2　Broker 内部的模块组成

首先，我们来看 KafkaServer 这个类包含的模块：

```
class KafkaServer(val config: KafkaConfig, time: Time = SystemTime)
extends Logging with KafkaMetricsGroup {
  this.logIdent = "[Kafka Server " + config.brokerId + "], "
  private var isShuttingDown = new AtomicBoolean(false)
  private var shutdownLatch = new CountDownLatch(1)
  private var startupComplete = new AtomicBoolean(false)
  val brokerState: BrokerState = new BrokerState
  val correlationId: AtomicInteger = new AtomicInteger(0)
  var socketServer: SocketServer = null
  var requestHandlerPool: KafkaRequestHandlerPool = null
  var logManager: LogManager = null
  var offsetManager: OffsetManager = null
  var kafkaHealthcheck: KafkaHealthcheck = null
  var topicConfigManager: TopicConfigManager = null
  var replicaManager: ReplicaManager = null
  var apis: KafkaApis = null
  var kafkaController: KafkaController = null
  val kafkaScheduler = new KafkaScheduler(config.backgroundThreads)
  var zkClient: ZkClient = null
  ......
}
```

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

分别是 SocketServer（监听 Socket 请求）、KafkaRequestHandlerPool（请求处理资源池）、LogManager（日志管理）、ReplicaManager（分区副本管理）、OffsetManager（偏移量管理）、KafkaScheduler（后台任务调度资源池）、KafkaApis（业务逻辑实现层）、KafkaHealthcheck（提供 Broker 健康状态）、TopicConfigManager（Topic 配置信息管理）和 KafkaController（Kafka 集群控制管理）。其相互之间的关系如图 3-2 所示。

详细说明如下：

- ❑ SocketServer：首先开启 1 个 Acceptor 线程用于监听默认端口号为 9092 上的 Socket 链接，然后当有新的 Socket 链接成功建立时会将对应的 SocketChannel 以轮询的方式转发给 N 个 Processor 线程中的某一个，并由其处理接下来该 SocketChannel 上的读写请求，其中 N=num.network.threads，默认为 3。当 Processor 线程监听来自 SocketChannel 的请求时，会将请求放置在 RequestChannel 中的请求队列；当 Processor 线程监听到 SocketChannel 请求的响应时，会将响应从 RequestChannel 中的响应队列中取出来并发送给客户端。
- ❑ KafkaRequestHandlerPool：真正处理 Socket 请求的线程池，其个数默认为 8 个，由参数 num.io.threads 决定。该线程池里面的线程 KafkaRequestHandler 从 RequestChannel 的请求队列中获取 Socket 的请求，然后调用 KafkaApis 完成真正的业务逻辑，最后将响应写回至 RequestChannel 中的响应队列，并交由 SocketServer 中对应的 Processor 线程发送给客户端。
- ❑ LogManager：Kafka 的日志管理模块。主要提供删除任何过期数据和冗余数据，刷新脏数据，对日志文件进行 Checkpoint 以及日志合并的功能。
- ❑ ReplicaManager：Kafka 的副本管理模块。主要提供针对 Topic 分区副本数据的管理功能，包括有关副本的 Leader 和 ISR 的状态变化、副本的删除、副本的监测等。其中 ISR 全称为 In-Sync Replicas，即处于同步状态的副本，在后面的章节还会详细介绍。
- ❑ OffsetManager：Kafka 的偏移量管理模块。主要提供针对偏移量的保存和读取的功能，Kafka 管理 Topic 的偏移量存在两种方式：一种为 Zookeeper，就是把偏移量提交至 Zookeeper 上；另一种为 Kafka，就是把偏移量提交至 Kafka 内部 Topic 为“__consumer_offsets”的日志里面，主要由 offsets.storage 参数决定，默认为 zookeeper。
- ❑ KafkaScheduler：Kafka 的后台任务调度资源池。提供后台定期任务的调度，主要为 LogManager、ReplicaManager 和 OffsetManager 提供调度服务。
- ❑ KafkaApis：Kafka 的业务逻辑实现层，根据不同的 Request 执行不同的操作，其中利用 LogManager、OffsetManager 和 ReplicaManager 来完成内部的处理。KafkaApis 处理的请求包括 ProducerRequest、TopicMetadataRequest、FetchRequest、OffsetRequest、OffsetCommitRequest、OffsetFetchRequest、LeaderAndIsrRequest、StopReplicaRequest、UpdateMetadataRequest、BrokerControlledShutdownRequest 和 ConsumerMetadataRequest。

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

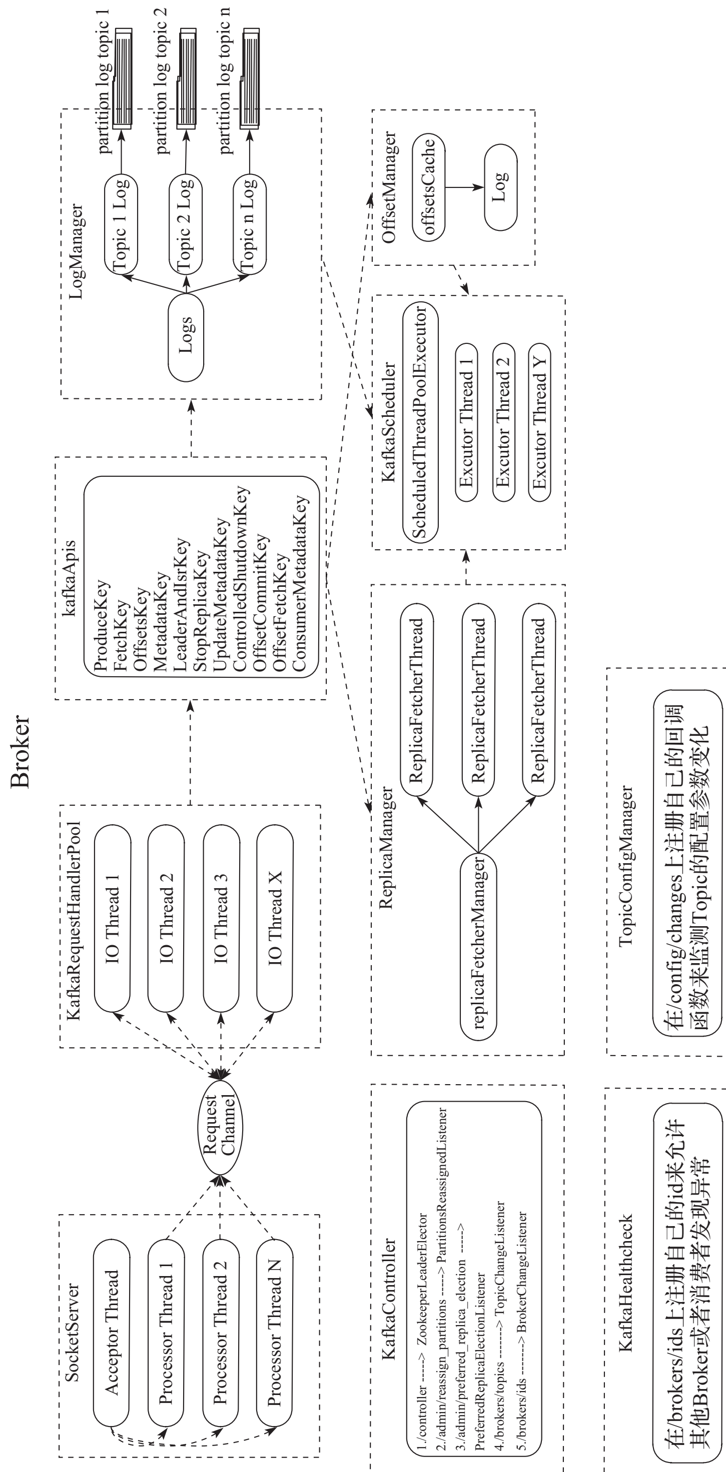


图 3-2　Broker 内部模块之间的关系

互动出版网 www.china-pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

- ❑ KafkaHealthcheck：Broker Server 在 /brokers/ids 上注册自己的 ID，当 Broker 在线的时候，则对应的 ID 存在；当 Broker 离线的时候，则对应的 ID 不存在，以此来达到集群状态监测的目的。
- ❑ TopicConfigManager：在 /config/changes 上注册自己的回调函数来监测 Topic 配置信息的变化。
- ❑ KafkaController：Kafka 的集群控制管理模块。由于 Zookeeper 上保存了 Kafka 集群的元数据信息，因此 KafkaController 通过在不同目录注册不同的回调函数来达到监测集群状态的目的，及时响应集群状态的变化。比如说：

  1）/controller 目录保存了 Kafka 集群中状态为 Leader 的 KafkaController 标识，通过监测这个目录的变化可以及时响应 KafkaController 状态的切换；

  2）/admin/reassign_partitions 目录保存了 Topic 重分区的信息，通过监测这个目录的变化可以及时响应 Topic 分区变化的请求；

  3）/admin/preferred_replica_election 目录保存了 Topic 分区副本的信息，通过监测这个目录的变化可以及时响应 Topic 分区副本变化的请求；

  4）/brokers/topics 目录保存了 Topic 的信息，通过监测这个目录的变化可以及时响应 Topic 创建和删除的请求；

  5）/brokers/ids 目录保存了 Broker 的状态，通过监测这个目录的变化可以及时响应 Broker 的上下线情况等。

希望读者着重理解 Broker 内部各个模块之间的相互关系，这对于从整体上把握 Kafka 架构会起到比较大的作用。在接下来的章节中将会针对以上每个模块进行进一步的讲解，如果在这一章节对某些概念不是很理解的话，也不要紧，可以继续往后看，然后再回过头看整体图，以便加深记忆。

3.3 本章小结

本章大致讲述了 Broker 的脚本启动过程以及 Broker 内部的模块组成。Broker 的模块组成主要区分为基本模块和控制管理模块，其中基本模块在每个 Broker 内部无论对内还是对外都提供服务，但是控制管理模块在整个 Kafka 集群中有且只有一个对外提供服务。在接下去的第 4 章和第 5 章将分别描述基本模块和控制管理模块的内部实现原理。