第5章 Kafka分布式发布订阅消息系统

学习目标：

- 了解消息队列，能够说出消息队列的主要应用场景。
- 熟悉 Kafka 的概念，能够叙述 Kafka 的优点。
- 熟悉 Kafka 的基本架构，能够说出 Kafka 基本架构的内容。
- 掌握 Kafka 的工作流程，能够叙述生产者生产消息过程和消费者消费消息过程。
- 掌握 Kafka 集群的搭建方法，能够独立完成部署 Kafka 集群。
- 掌握 Kafka 的基本操作，能够使用 Shell 命令和 Python API 操作 Kafka。
- 掌握实时单词计数，能够从一个 Topic 中消费数据实现实时单词计数，并将结果发送到另一个 Topic 中。

Kafka 是一个基于 ZooKeeper 系统的分布式发布订阅消息系统适用于实时计算系统。通常情况下，使用 Kafka 能够构建系统或应用程序之间的数据管道，用来转换或响应实时数据，使数据能够及时地进行业务计算，得出相应结果。本章针对消息队列简介、Kafka 简介、Kafka 工作原理、Kafka 集群的搭建以及 Kafka 的基本操作进行详细讲解。

5.1 消息队列简介

消息队列(Message Queue，MQ)是分布式系统中的一个关键组件，用于存储消息，它的作用是将待传输的数据存放在队列中，以便生产者和消费者可以并行地处理数据，而无须等待对方的响应。通过消息队列，生产者可以将消息发送到队列，而消费者可以从队列中获取消息进行处理。这种解耦的设计模式使得系统的可伸缩性和可靠性得到提高，同时也减少了系统间的依赖性。

消息队列既然能够用来存储消息，那么消息队列的主要应用场景有哪些呢？接下来，针对消息队列的主要应用场景进行介绍。

(1) 异步处理。

异步处理是指应用程序允许用户将一个消息放入队列中，但是应用程序并不立即处理用户提交的消息，而是在用户需要用到该消息时应用程序再去处理。例如，用户在注册电商网站时，在没有使用异步处理的场景下，注册流程是电商网站把用户提交的注册信息保存到数据库中，同时额外发送注册的邮件通知以及短信注册码给用户。由于发送邮件通知和短信注册码需要连接其对应的服务器，如果发送完邮件通知再发送短信注册码，用户就会等待

较长的时间。针对上述情况,使用消息队列将邮件通知以及短信注册码保存起来,电商网站只需要将用户的注册信息保存到数据库中便可完成注册,这样便能实现快速响应用户注册的操作。下面,通过图 5-1 来了解使用异步处理前后的区别。

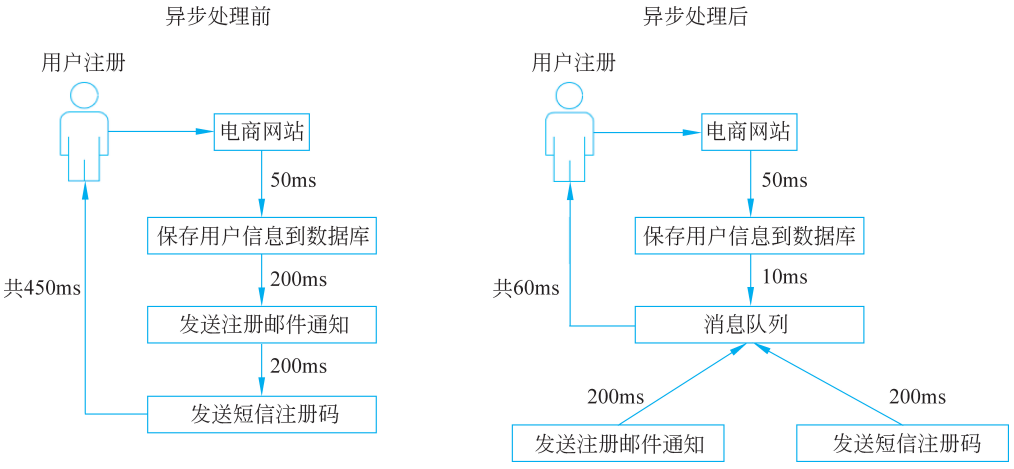

图 5-1 使用异步处理前后的区别

从图 5-1 可以看出,使用异步处理前,用户需要经历用户注册→电商网站→保存用户信息到数据库→发送注册邮件通知→发送短信注册码 5 个步骤,用户注册到发送短信注册码总共需要耗时 450ms;而使用异步处理后,用户只需经历用户注册→电商网站→保存用户信息到数据库→消息队列 4 个步骤,用户注册到将注册信息保存到消息队列总共需要消耗 60ms,通过对比可以发现,异步处理的注册方式要比传统注册方式响应得快。

(2) 系统解耦。

系统解耦是指用户提交的请求需要与应用程序中另一个模块建立联系,两个模块之间不会因为各自功能的问题而影响另一个模块的使用。例如,用户在电商网站购买物品并提交订单时,订单模块会调用库存模块确认商品是否还有库存,在没有系统解耦的场景下,如果库存模块功能出现问题,会导致订单模块下单失败,而且当库存模块的对外接口发生变化,订单模块也依旧无法正常工作。当使用了系统解耦,订单模块便不会直接调用库存模块,而是将订单信息保存到消息队列中,库存模块再从消息队列中获取订单信息,从而实现订单模块与库存模块之间互不影响。下面,通过图 5-2 来了解使用系统解耦前后的区别。

从图 5-2 可以看出,使用系统解耦前,订单模块需要直接调用库存模块,而使用系统解耦后,订单模块先将订单信息保存到消息队列中,然后库存模块在消息队列中获取订单信息,这样订单模块与库存模块之间不会产生直接的影响。

(3) 流量削峰。

流量削峰是在物品秒杀或促销等场景中,避免用户访问次数过多导致应用程序崩溃的一种策略。它通过控制参与活动的人数,以缓解短时间内访问次数过多对应用程序造成的压力。例如,电商网站推出物品秒杀活动,在未使用流量削峰的场景下,电商网站推出物品秒杀活动时,大量用户会访问物品秒杀活动界面,造成该界面的访问次数过多,导致电商网站负载过重而容易崩溃。当使用了流量削峰,物品秒杀界面在接收用户的请求后,会将用户的请求保存到消息队列中,如果请求的数据量超过了设定的消息队列的容量,就会告知当前

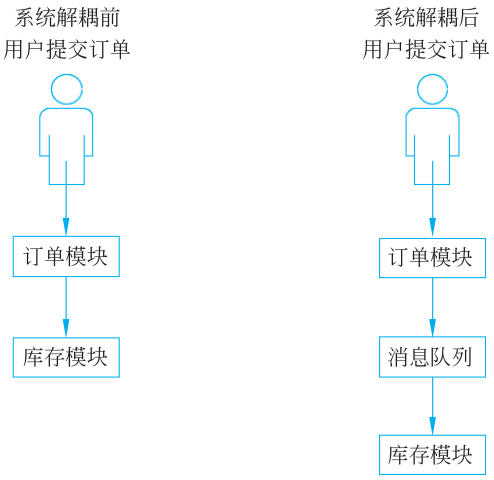

图 5-2　使用系统解耦前后的区别

活动参与人数过多，这样可以避免整个电商网站崩溃的现象。下面，通过图 5-3 来了解使用流量削峰前后的区别。

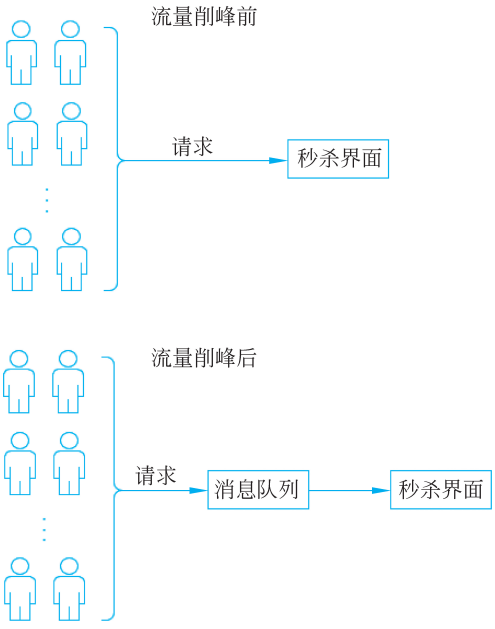

图 5-3　使用流量削峰前后的区别

从图 5-3 可以看出，使用流量削峰前，用户直接请求秒杀界面，短时间内该界面被用户请求的次数过多，而使用流量削峰后，用户请求被保存到消息队列中，秒杀界面在消息队列中获取用户请求，这样能避免秒杀界面请求次数过多导致整个电商网站崩溃的现象。

了解了消息队列的应用场景，那么消息队列中的消息是如何进行传递的呢？消息传递一共有两种模式，分别是点对点消息传递和发布/订阅消息传递模式，关于这两种消息传递模式的介绍如下。

(1) 点对点消息传递模式。

在点对点(Point to Point，P2P)消息传递模式下，消息生产者将消息发送到特定队列，

消息消费者从队列中拉取或轮询以获取消息。

点对点消息传递模式结构如图 5-4 所示。

图 5-4 点对点消息传递模式结构

从图 5-4 可以看出,生产者将消息发送到消息队列中,此时将有一个或者多个消费者会消费消息队列中的消息,但是消息队列中的每条消息只能被消费一次,并且消费后的消息会从消息队列中删除。

(2) 发布/订阅消息传递模式。

在发布/订阅(Publish/Subscribe)消息传递模式下,消息生产者将消息发送到消息队列中,所有消费者会即时收到并消费消息队列中的消息。

发布/订阅消息传递模式结构如图 5-5 所示。

图 5-5 发布/订阅消息传递模式结构

从图 5-5 可以看出,在发布/订阅消息传递模式结构中,生产者将消息发送到消息队列中,此时将有多个不同的消费者消费消息队列中的消息。与点对点模式不同的是,发布/订阅消息传递模式中消息队列的每条消息可以被多次消费,并且消费完的消息不会立即删除。

【小提示】

点对点消息传递模式和发布/订阅消息传递模式都会采用基于拉取或推送方式传递消息。基于拉取方式传递消息时消费者会定期查询消息队列是否有新消息,基于推送方式传递消息时消息队列会将消息推送给已订阅该消息队列的消费者。不过在发布/订阅消息传递模式中,常用的消息传递方式为拉取方式。

5.2 Kafka 简介

Kafka 是一个基于 ZooKeeper 系统的分布式发布订阅消息系统,它使用 Scala 和 Java 语言编写,该系统的设计初衷是为实时数据提供一个统一、高吞吐、低延迟的消息传递平台。在 0.10 版本之前,Kafka 只是一个消息系统,主要用来解决异步处理、系统解耦等问题,在 0.10 版本之后,Kafka 推出了流处理的功能,使其逐渐成为了一个流式数据平台。

Kafka 作为分布式发布订阅消息系统,可以处理大量的数据,并能够将消息从一个端点传递到另外一个端点。Kafka 在大数据领域中的应用非常普遍,它能够在离线和实时两种大数据计算场景中处理数据,这得益于 Kafka 的优点,其优点具体如下。

（1）高吞吐，低延迟。Kafka 可以每秒处理数量庞大的消息，并且具有较低的延迟。

（2）可扩展性。Kafka 是一个分布式系统，用户可以根据实际应用场景自由、动态地扩展 Kafka 服务器。

（3）持久性。Kafka 可以将消息存储在磁盘上，以确保数据的持久性。

（4）容错性。Kafka 会将数据备份到多台服务器中，即使 Kafka 集群中的某台服务器宕机，也不会影响整个系统的功能。

（5）支持多种语言。Kafka 支持 Java、Scala、PHP、Python 等多种语言，这使得开发人员在不同语言环境下使用 Kafka 更加便捷。

在实际的大数据计算场景中，若需要对接外部数据源时，就可以使用 Kafka，如日志收集系统和消息系统，Kafka 读取日志系统中的数据，每得到一条数据，就可以及时地处理一条数据，这就是常见的流式计算框架应用场景之一。在流式计算框架中，Kafka 一般用来缓存数据，它与 Apache 旗下的 Spark、Storm 等框架紧密集成，这些框架可以接收 Kafka 中的缓存数据并进行计算，实时得出相应的计算结果。

5.3　Kafka 工作原理

5.3.1　Kafka 的基本架构

学习 Kafka 的基本架构对于有效地使用和管理 Kafka 是至关重要的。Kafka 的基本架构由 Producer、Broker、Consumer 和 ZooKeeper 构成，它们之间共同协作，构建了高效、可靠的消息处理系统。接下来，通过图 5-6 学习 Kafka 的基本架构。

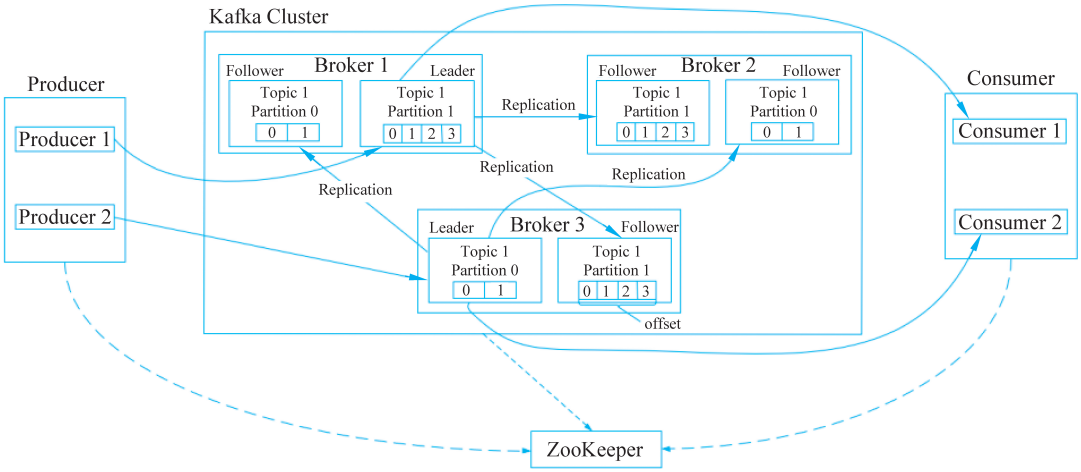

图 5-6　Kafka 的基本架构

1. Producer

Producer 作为 Kafka 中的生产者，主要负责将消息发送到 Broker（消息代理）内部的 Topic（主题）中，在发送消息时，消息的内容主要包括键和值两部分，其中键默认为 null，值是指发送消息的内容。除此之外，用户还可以根据需求添加属性信息。为消息指定键可以将相同键的消息发送到相同的分区，从而保证相关消息的顺序性。

2. Broker

Broker 作为 Kafka 中的消息代理,是存储和管理消息的载体,每一个 Broker 都可以看作 Kafka 服务。存储在 Broker 中的消息基于 Topic 进行分类和组织。在 Kafka 中,Topic 是消息的逻辑概念,类似于一个消息类别或话题,每个 Topic 可以有一个或多个 Partition(分区)。例如,具有 3 个 Partition 的 Topic,如图 5-7 所示。

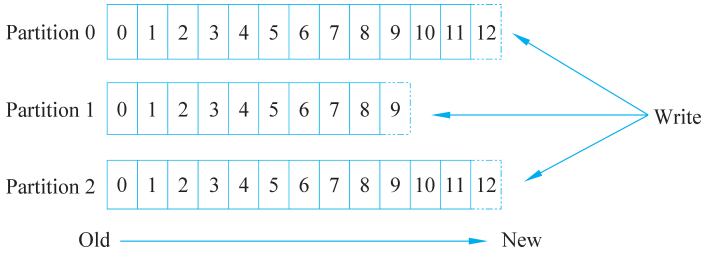

图 5-7 具有 3 个 Partition 的 Topic

在图 5-7 中,Partition 的标识从 0 开始,Producer 生产的消息会被分配到不同的 Partition 中,每条消息都会被分配一个从 0 开始具有递增顺序的 offset(偏移量),不同 Partition 之间的 offset 相互独立,互不影响。

Topic 中的每个 Partition 可以存在多个副本,这些副本分布在不同的 Broker 上,实现消息的备份和容错。在 Kafka 中,Partition 分为 Leader 和 Follower 两个角色。Leader 负责接收和发送消息,而 Follower 作为 Leader 的副本则负责复制 Leader 的消息。这种设计保证了在某个 Broker 失效时,系统依然能够确保消息的可用性和一致性。

此外,Broker 还负责响应 Consumer(消费者)消费消息的请求,Broker 根据 Consumer 提供的 offset 检索 Topic 中相应 Partition 的消息,并将这些消息传递给 Consumer。

3. Consumer

Consumer 作为 Kafka 中的消费者,负责消费 Topic 中的消息,一旦 Consumer 成功消费了消息,Consumer 记录自身已消费消息的 offset,并且根据配置策略手动或定期自动地将已消费消息的 offset 保存在 Broker 内部名为__consumer_offsets 的 Topic,这确保了 Consumer 即使重新消费或崩溃时,Broker 能够准确地确定消息的 offset,实现从正确的位置继续消费消息。

在 Kafka 中,多个 Consumer 可以组成特定的消费者组,消费者组之间相互独立,互不影响,这种设计可以让多个 Consumer 协同处理同一个 Topic 中的消息,实现负载均衡。

4. ZooKeeper

ZooKeeper 在 Kafka 中负责管理和协调 Broker,并且 ZooKeeper 存储了 Kafka 的元数据信息,包括 Topic 名称、Partition 副本等。

多学一招:Kafka 分区策略

生产者将消息发送到 Broker 内部的 Topic 时,如果需要确保每个 Topic 中的 Partition 负载均衡,可以在生产者发送消息时为生产者指定相应的分区策略。Kafka 中常见的分区策略有 DefaultPartitioner、RoundRobinPartitioner、StickyPartitioner 和 UniformStickyPartitioner,关于这 4 种分区策略的介绍如下。

1. DefaultPartitioner

该分区策略是 Kafka 默认的分区策略，针对消息保存到 Partition 时会存在 3 种情况，具体如下。

（1）生产者发送消息的时候指定了 Partition，则消息将保存到指定的 Partition 中。

（2）生产者发送消息的时候没有指定 Partition，但消息的键不为空，则基于键的哈希值来选择一个 Partition 进行保存。

（3）生产者发送消息的时候不但没有指定 Partition，而且消息的键为空，则通过轮询的方式将消息均匀地保存到所有 Partition。这种情况下，DefaultPartitioner 分区策略会基于 Partition 的数量和可用性以确保消息的平均保存。

2. RoundRobinPartitioner

该分区策略是一种轮询分区策略，在保存消息时并不考虑消息中键的影响，而是通过轮询的方式将每条消息依次发送到每个 Partition，确保消息在所有 Partition 间按照严格的轮询顺序分布，适用于希望均匀地保存消息以实现负载平衡，但不考虑消息的相关性或顺序性。

3. StickyPartitioner

该分区策略是一种黏性分区策略，在保存消息时需要考虑消息中键的影响，会将具有相同键的消息保存到同一个 Partition 中，以保持消息的顺序性和一致性，适用于需要按照消息的键保存到 Partition 后依然保持顺序，然而这种情况下会出现其中一个 Partition 中具有相同键的消息比较多，而另一个 Partition 中具有相同键的消息比较少。

4. UniformStickyPartitioner

该分区策略是一种统一黏性分区策略，针对消息保存到 Partition 时会存在两种情况，具体如下。

（1）生产者发送消息的时候指定了 Partition，则消息将保存到指定的 Partition 中。

（2）生产者发送消息的时候没有指定 Partition，但消息的键不为空，会将具有相同键的消息保存到不同的 Partition 中，实现 Partition 负载均衡。

5.3.2　Kafka 工作流程

Kafka 的工作流程是 Kafka 实现消息发送和消费的核心过程，了解 Kafka 的工作流程对于理解 Kafka 的基本架构和性能优化有着至关重要的作用。Kafka 的工作流程可以分为生产者生产消息过程和消费者消费消息过程。

接下来，针对生产者生产消息过程和消费者消费消息过程进行详细讲解。

1. 生产者生产消息过程

Kafka 生产者负责生成并发送消息到 Kafka 集群中的指定主题中。下面通过图 5-8 来介绍生产者生产消息过程。

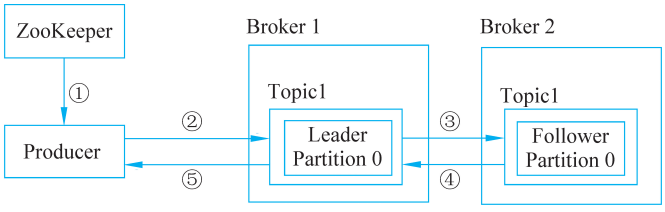

图 5-8　生产者生产消息过程

从图 5-8 可以看出,生产者生产消息过程可以分为 5 个步骤,具体如下。

(1) Producer 通过访问 Broker 间接获取 ZooKeeper 中存储的元数据,包括 Topic 分区分布、Leader 副本位置等。

(2) Producer 将消息发送给角色为 Leader 的 Partition,与此同时,角色为 Leader 的 Partition 会将消息写入自身的日志文件中。

(3) 角色为 Follower 的 Partition 从角色为 Leader 的 Partition 中获取消息,将消息写入自身的日志文件中,完成复制操作。

(4) 角色为 Follower 的 Partition 将消息写入自身的日志文件后,会向角色为 Leader 的 Partition 发送成功复制消息的信号。

(5) 角色为 Leader 的 Partition 收到角色为 Follower 的 Partition 发送的复制消息后,同样向 Producer 发送消息写入成功的信号,此时消息生产完成。

2. 消费者消费消息过程

消息由 Producer 发送到指定 Topic 中角色为 Leader 的 Partition 中后,Consumer 会采用拉取模型的方式消费消息。在拉取模型下,Consumer 主动向 Broker 发送消费消息的请求,请求的内容包括消息的 Partition、offset 等,Broker 根据请求将消息返回给 Consumer,Consumer 消费消息后会将 offset 提交给 Broker,以便下次能够正确消费消息。该模型的优势在于 Consumer 会记录自己的消费状态,后续 Consumer 可以对已消费的消息再次消费,避免出现网络延迟或者宕机等原因造成消息消费延迟或丢失。

下面通过图 5-9 介绍消费者消费消息过程。

从图 5-9 可以看出,消费者消费消息过程可以分为 4 个步骤,具体如下。

(1) Consumer 通过访问 Broker 间接获取 ZooKeeper 中存储的元数据,包括 Topic 分区分布、Leader 副本位置等。

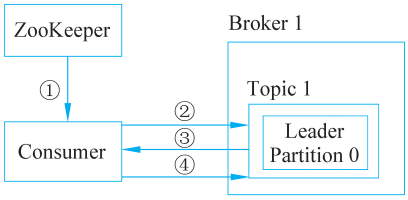

图 5-9 消费者消费消息过程

(2) Consumer 根据消息的 offset,向 Topic 中角色为 Leader 的 Partition 发送请求消费消息。

(3) Topic 中角色为 Leader 的 Partition 根据 offset 将对应的消息返回给 Consumer 进行消费。

(4) Consumer 消费消息后,记录自己的消费状态,将已消费消息的 offset 保存在 Broker 内部特殊的 Topic 中,以便下次消费消息时能够从正确的位置开始消费。

通过本节的学习,了解到 Kafka 中的工作流程是由各个组成部分相互协调实现的。在个人学习成长过程中,也应铭记协调的重要性。协调不仅能够促进团队成员之间的沟通和协商,而且能够协调冲突和不同意见,以实现共同的学习和工作目标。

5.4 搭建 Kafka 集群

学习完 Kafka 理论知识后,接下来讲解如何在虚拟机 Hadoop1、Hadoop2 和 Hadoop3 中搭建 Kafka 集群,具体步骤如下。

1. 下载 Kafka 安装包

本书使用的 Kafka 版本为 3.2.1。通过 Kafka 官网下载 Kafka 安装包 kafka_2.12-3.2.1.tgz。

2. 上传 Kafka 安装包

在虚拟机 Hadoop1 的/export/software 目录执行 rz 命令，将准备好的 Kafka 安装包 kafka_2.12-3.2.1.tgz 上传到虚拟机的/export/software 目录。

3. 安装 Kafka

使用解压操作安装 Kafka，将 Kafka 安装到存放安装程序的目录/export/servers，在/export/software 目录执行如下命令。

```
$ tar -zxvf kafka_2.12-3.2.1.tgz -C /export/servers/
```

4. 配置 Kafka 环境变量

分别在虚拟机 Hadoop1、Hadoop2 和 Hadoop3 执行 vi/etc/profile 命令编辑系统环境变量文件 profile，在该文件的尾部添加如下内容。

```
export KAFKA_HOME=/export/servers/kafka_2.12-3.2.1
export PATH=:$PATH:$KAFKA_HOME/bin
```

成功配置 Kafka 环境变量后，保存并退出系统环境变量文件 profile 即可。不过此时在系统环境变量文件中添加的内容尚未生效，还需要分别在 Hadoop1、Hadoop2 和 Hadoop3 执行 source/etc/profile 命令初始化系统环境变量使配置的 Kafka 环境变量生效。

5. 修改配置文件

为了确保 Kafka 集群能够正常启动，还需要对 Kafka 的配置文件进行相关的配置。执行 cd /export/servers/kafka_2.12-3.2.1/config/命令进入 Kafka 安装目录的 config 目录，在该目录执行 vi server.properties 命令编辑 server.properties 配置文件，将 server.properties 配置文件中对应的参数修改为如下内容。

```
broker.id=0
log.dirs=/export/data/kafka
zookeeper.connect=hadoop1:2181,hadoop2:2181,hadoop3:2181
```

上述内容修改完成后，保存并退出 server.properties 配置文件。针对上述内容中的参数进行如下讲解。

(1) broker.id：Kafka 集群中每个节点的唯一且永久的 ID，该值必须大于或等于 0。在本书中，虚拟机 Hadoop1、Hadoop2 和 Hadoop3 对应的 broker.id 分别为 0，1，2。

(2) log.dirs：指定 Kafka 集群运行日志存放的路径。

(3) zookeeper.connect：指定 ZooKeeper 集群的主机名与端口号。

6. 分发 Kafka 安装目录

执行 scp 命令，将虚拟机 Hadoop1 的 Kafka 安装目录分发至虚拟机 Hadoop2 和 Hadoop3 中存放安装程序的目录，具体命令如下。

```
# 将 Kafka 安装目录分发至虚拟机 Hadoop2 中存放安装程序的目录
$ scp -r /export/servers/kafka_2.12-3.2.1/ hadoop2:/export/servers/
```

```
# 将 Kafka 安装目录分发至虚拟机 Hadoop3 中存放安装程序的目录
$ scp -r /export/servers/kafka_2.12-3.2.1/ hadoop3:/export/servers/
```

将 Kafka 安装目录分发完成后,分别进入虚拟机 Hadoop2 和 Hadoop3 的 Kafka 安装目录的 config 目录,在该目录执行 vi server.properties 命令编辑 server.properties 配置文件,将虚拟机 Hadoop2 的 Kafka 的 server.properties 配置文件中的 broker.id 修改为 1,将虚拟机 Hadoop3 的 Kafka 的 server.properties 配置文件中的 broker.id 修改为 2。

7. 启动 ZooKeeper

在启动 Kafka 之前,需要先启动 ZooKeeper 服务,分别在虚拟机 Hadoop1、Hadoop2 和 Hadoop3 上执行如下命令启动 ZooKeeper 服务。

```
$ zkServer.sh start
```

8. 启动 Kafka 服务

这里以虚拟机 Hadoop1 为例,演示如何启动 Kafka 服务。执行 cd /export/servers/kafka_2.12-3.2.1/命令进入虚拟机 Hadoop1 的 Kafka 安装目录,执行如下命令启动 Kafka 服务。

```
$ bin/kafka-server-start.sh config/server.properties
```

上述命令执行完成后,Kafka 服务启动效果如图 5-10 所示。

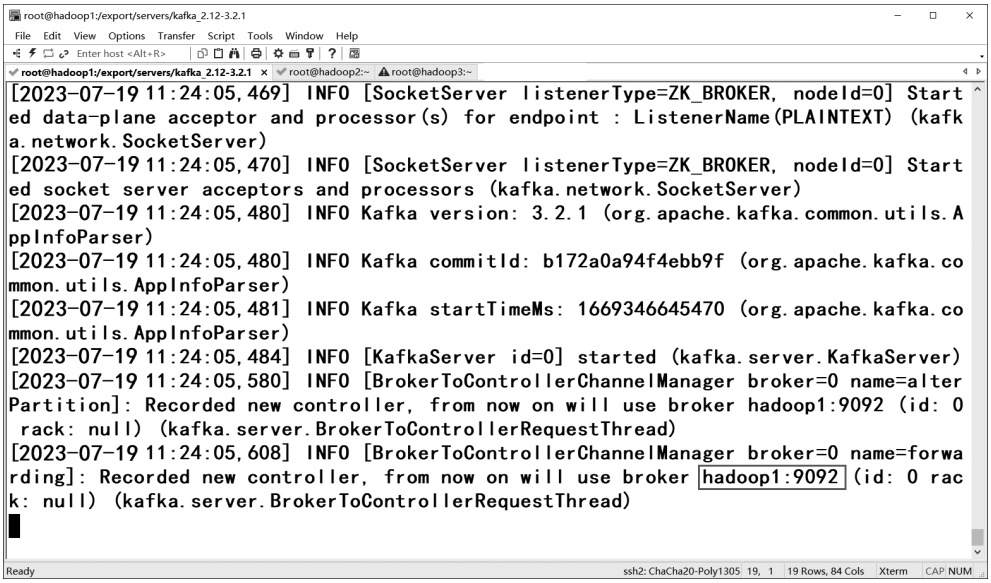

图 5-10 Kafka 服务启动效果

从图 5-10 可以看出,如果 SecureCRT 控制台输出的消息中无异常信息,并且光标始终处于闪烁状态,即表示 Kafka 启动成功。消息代理默认使用的端口号为 9092。

9. 查看 Kafka 启动状态

Kafka 启动完成后,可以克隆虚拟机 Hadoop1 的会话框,执行 jps 命令查看 Kafka 是否正常启动,如图 5-11 所示。