第 5 章 ZooKeeper分布式协调服务

学习目标

- 了解 ZooKeeper 的特性，能够简述 ZooKeeper 的 5 个特性。
- 掌握 ZooKeeper 的集群架构，能够描述 ZooKeeper 集群中 Leader、Follower 和 Observer 的作用。
- 熟悉 ZooKeeper 的数据模型，能够描述 ZooKeeper 中 3 种 ZNode 类型的作用。
- 熟悉 ZooKeeper 的 Watcher 机制，能够描述事件监听机制的特性。
- 熟悉 ZooKeeper 的选举机制，能够描述 ZooKeeper 集群选举 Leader 的两种情况及其过程。
- 掌握完全分布式模式部署 ZooKeeper 集群，能够独立完成基于完全分布式模式部署 ZooKeeper 集群。
- 了解伪分布式模式部署 ZooKeeper 集群，能够完成基于伪分布式模式部署 ZooKeeper 集群。
- 掌握 ZooKeeper 的 Shell 操作，能够熟练使用客户端工具 zkCli 对 ZooKeeper 集群进行操作。
- 掌握 ZooKeeper 的 Java API 操作，能够熟练使用 ZooKeeper 提供的 Java API 对 ZooKeeper 进行操作。

ZooKeeper 是一个开源的分布式协调服务，它是 Google Chubby 的开源实现，设计目标是将那些复杂且容易出错的分布式应用封装起来，以构成一个高效可靠的原语集，并以一系列简单易用的接口提供给用户使用。本章将针对 ZooKeeper 的基础知识和常见操作进行详细讲解。

5.1 ZooKeeper 简介

ZooKeeper 最早起源于某研究院的一个研究小组。当时，研究人员发现很多分布式应用基本都需要依赖一个类似的应用进行分布式协调，但是这些分布式应用往往都存在分布式单点问题，即在整个分布式应用中，如果某个独立功能的程序或角色只运行在某一台服务器上，那么这个结点就称为单点，一旦这台服务器宕机，那么整个分布式应用都将无法正常运行，这种现象就称为单点问题。所以，研究人员试图开发一个通用的无单点问题的分布式协调服务，以便让研究人员将精力集中在业务逻辑上，此时，ZooKeeper 便应运而生。

5.1.1 ZooKeeper 特性

ZooKeeper 是一个开源的分布式协调服务,它为分布式应用提供了高效且可靠的分布式协调服务,并且是分布式应用保证数据一致性的解决方案。分布式应用可以基于 ZooKeeper 实现诸如数据发布/订阅、负载均衡、命名服务、分布式协调/通知、集群管理、Master 选举、分布式锁、分布式队列等功能。ZooKeeper 之所以能够确保分布式应用的数据一致,离不开它的 5 个特性,具体介绍如下。

1. 顺序一致性

顺序一致性(Sequential Consistency)是指对于客户端发起的每个事务请求,ZooKeeper 都会为其分配一个全局唯一的递增编号,这个编号反映了所有事务请求的先后顺序,ZooKeeper 会严格按照事务请求的顺序进行处理。

2. 原子性

原子性(Atomicity)是指事务请求的处理结果在整个集群中的应用情况是一致的,也就是说,要么集群中的所有服务器成功地将事务请求应用到 ZooKeeper,要么整个集群中的所有服务器都没有将事务请求应用到 ZooKeeper。

3. 可靠性

可靠性(Reliability)是指一旦 ZooKeeper 应用了客户端的某个事务请求,并且将该事务请求的处理结果响应给客户端,那么该事务请求引起的 ZooKeeper 数据的变更就会被一直保留,除非其他事务请求对其进行了变更。

4. 单一系统映像

单一系统映像(Single System Image)是指无论客户端连接集群中的哪个 ZooKeeper,看到的数据都是一致的。

5. 时效性

时效性(Timeliness)是指当客户端发送的事务请求成功应用到 ZooKeeper 之后,ZooKeeper 可以确保在一定时间内,其他客户端能够读取该事务请求对数据进行变更的结果,而不是立即读取该事务请求对数据进行变更的结果,因此时效性较低。

多学一招:事务请求和非事务请求

在 ZooKeeper 中,客户端向 ZooKeeper 发起的请求分为事务请求和非事务请求,其中,事务请求会使 ZooKeeper 的数据发生变更,例如写入、修改、删除等;非事务操作仅仅读取 ZooKeeper 的数据,并不会使 ZooKeeper 的数据发生变更。

5.1.2 ZooKeeper 集群架构

ZooKeeper 集群由多个服务器组成,每个服务器在 ZooKeeper 集群中扮演着不同的角色,包括 Leader、Follower 和 Observer,如图 5-1 所示。

接下来对图 5-1 展示的 ZooKeeper 集群中的 3 个角色进行详细介绍。

1. Leader

Leader 是 ZooKeeper 集群的核心,每个 ZooKeeper 集群中只能有一个 Leader,主要用于处理客户端发送的事务请求,并将事务请求广播到所有 Follower,以保证处理事务请求的顺序

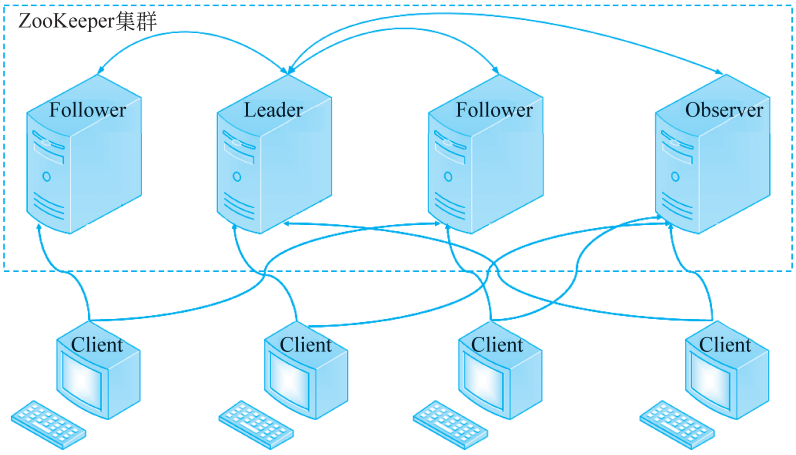

图 5-1　ZooKeeper 集群架构

一致性。同时，为了确保 Follower 可以检测到 Leader 的存在，Leader 会与 Follower 同步状态信息。

2. Follower

Follower 可以看作 Leader 的跟随者，每个 ZooKeeper 集群中可以包含多个 Follower，它的作用主要有以下 4 点。

- 处理客户端发送的非事务请求。
- 将客户端发送的事务请求转发给 Leader。
- 参与事务请求投票，即每个 Follower 成功应用 Leader 广播的事务请求后，将向 Leader 发送反馈信息，如果超过半数的 Follower 向 Leader 发送了反馈信息，则认为该事务请求应用到了整个 ZooKeeper 集群。
- 参与 Leader 选举投票，如果当前 Leader 宕机，则所有 Follower 会重新投票选举出一个新的 Leader。

3. Observer

Observer 可以看作 Leader 的观察者，每个 ZooKeeper 集群中可以包含多个 Observer，主要负责同步 Leader 的数据，处理客户端发送的非事务请求，并将事务请求转发给 Leader，通常用于在不影响集群事务请求处理能力的前提下，提升集群的非事务请求处理能力。需要注意的是，Observer 不参与任何形式的投票，如 Leader 选举投票。

在 ZooKeeper 集群中，每个服务器扮演着不同的角色，就像一个团队中的每个成员也扮演着不同的角色。只有 ZooKeeper 集群中的每个服务器都履行好自己的职责，才能实现高可用和高可靠。类似地，在一个团队中，每个成员只有履行好自己的职责，才能为团队的成功做出贡献。

5.2　ZooKeeper 数据模型

ZooKeeper 的视图结构和标准的 UNIX 文件系统非常类似，但没有引入 UNIX 文件系统中目录和文件的相关概念，而是使用了其特有的“数据结点”概念，称为 ZNode。ZNode 是

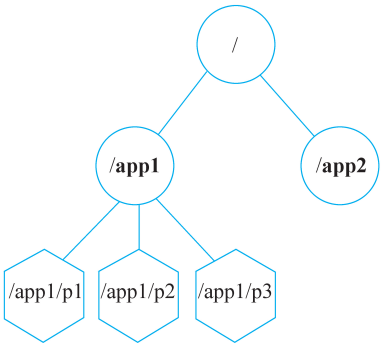

图 5-2 ZooKeeper 数据模型

ZooKeeper 中数据的最小单元,每个 ZNode 默认能够保存 1MB 的数据,同时可以挂载子结点,挂载的子结点也可以单独看作 ZNode,从而构成一个层次化的命名空间,称为树。ZooKeeper 数据模型如图 5-2 所示。

从图 5-2 可以看出,ZooKeeper 的数据模型由多个 ZNode 组成,如“/”“/app1”“/app2”,每个 ZNode 都可以挂载子结点,如“/”的子结点为“/app1”和“/app2”,这些 ZNode 按层次化结构组织形成树状结构。ZNode 的路径标识方式和 UNIX 文件系统相似,都使用一系列斜杠(/)进行分割的路径表示。

在 ZooKeeper 中,每个 ZNode 都是有生命周期的,其生命周期的长短取决于 ZNode 的类型。ZNode 的类型主要分为持久结点(PERSISTENT)、临时结点(EPHEMERAL)和顺序结点(SEQUENTIAL)。

1. 持久结点

持久结点是 ZooKeeper 中最常见的一种 ZNode 类型,它的生命周期取决于用户何时进行删除操作,持久结点被创建后,便会一直存在于 ZooKeeper 中,除非主动删除持久结点。

2. 临时结点

与持久结点有所不同,临时结点的生命周期取决于客户端会话。客户端会话是指客户端与 ZooKeeper 成功建立连接后创建的会话,若此时在 ZooKeeper 中创建临时结点,则在客户端与 ZooKeeper 断开连接时,临时结点便会被自动清理。需要注意的是,临时结点不能挂载子结点,只能存储数据。

3. 顺序结点

顺序结点基于持久结点和临时结点创建,因此可以将顺序结点分为持久顺序结点和临时顺序结点。在创建顺序结点时,默认会在顺序结点的基础上设置一个不断增加的序号,该序号对于当前顺序结点的父结点来说是唯一的,这样便于记录父结点中每个子结点创建的先后顺序。

序号的格式由 10 位数字组成,起始序号为 0000000000,例如在名称为/node 的 ZNode 下先后创建了 3 个顺序结点,那么这 3 个顺序结点的序号分别是 0000000000、000000001 和 0000000002。需要注意的是,持久结点和临时结点也会使序号增加,例如再次在/node 下先创建一个持久结点,再创建一个顺序结点,那么此时该顺序结点的序号为 0000000004,而不是 0000000003。值得一提的是,只有当在 ZNode 下创建了顺序结点之后,序号才会增加,如在同一 ZNode 下先创建一个持久结点,再创建一个顺序结点,那么该顺序结点的序号为 0000000000,而不是 0000000001。

5.3 ZooKeeper 典型应用场景

随着近年来互联网系统规模的不断扩大,大数据时代飞速到来,越来越多的分布式应用将 ZooKeeper 作为核心组件,如 Hadoop、HBase 和 Kafka 等,因此,正确理解 ZooKeeper 的应用场景,对于 ZooKeeper 的使用者来说显得尤为重要。本节将重点围绕数据发布/订阅、命名服务和分布式锁讲解 ZooKeeper 的典型应用场景。

1. 数据发布/订阅

数据发布/订阅是指发布者将数据发布到 ZooKeeper 的一个或一系列 ZNode 上，供订阅者进行数据订阅，进而达到动态获取数据的目的，实现配置信息的集中式管理和数据的动态更新。

在数据发布/订阅中应用 ZooKeeper 时，订阅者会向 ZooKeeper 指定的 ZNode 注册一个 Watcher 进行监听，一旦该 ZNode 的数据发生变化，那么 ZooKeeper 就会向相应的订阅者发送事件通知，当订阅者接收到这个通知后，便会主动到 ZooKeeper 指定的 ZNode 中获取最新的数据。

2. 命名服务

命名服务也是分布式系统中比较常见的应用场景，它是分布式应用基础的公共服务之一。在分布式应用中，被命名的实体通常是集群中的服务器、提供服务的地址或远程对象等，这些实体统称为命名空间(NameSpace)，其中，较为常见的是一些分布式服务框架(如 RPC、RMI)中的服务地址列表，通过命名服务，客户端能够根据指定名字从 ZooKeeper 获取资源的实体、服务地址和提供者等信息。

3. 分布式锁

分布式锁是控制分布式应用之间同步访问共享资源的一种方式。如果不同的应用或同一应用的不同主机共享资源，那么访问这些资源时，往往需要通过一些互斥手段防止彼此之间的干扰，以保证一致性，在这种情况下，就需要使用分布式锁。

分布式锁主要分为排他锁和共享锁，具体内容如下。

1) 排他锁

排他锁(Exclusive Locks，简称 X 锁)又称独占锁，ZooKeeper 实现排他锁时，通常的做法是把 ZNode 看作一把锁，所有客户端都会试图在该 ZNode 下创建临时结点以获取这个锁，不过最终只有一个客户端可以获取该锁，这是因为同一 ZNode 下的临时结点不能存在相同名称，其他没有获取锁的客户端会在 ZNode 下注册一个 Watcher 实时监听临时结点的变化，当获取锁的客户端正常执行完业务逻辑后，便会断开连接，此时该客户端创建的临时结点便会自动删除，没有获取锁的客户端会重新试图在 ZNode 中创建临时目录以获取锁。

2) 共享锁

共享锁(Shared Locks，简称 S 锁)又称读锁，ZooKeeper 在实现共享锁时，通常的做法是把 ZNode 看作一把锁，所有客户端都会试图在该 ZNode 下创建类似于“/shared_lock/[hostname]-请求类型-序号”的临时顺序结点，所有客户端都会按照临时顺序结点创建的先后顺序依次执行。例如/shared_lock/192.168.121.161-R-0000000001 和/shared_lock/192.168.121.161-W-0000000002 分别表示读请求和写请求的临时顺序结点，其中，读请求的临时顺序结点先执行，写请求的临时顺序结点后执行。

5.4　ZooKeeper 的 Watcher 机制

ZooKeeper 在实现数据发布/订阅时，订阅者会在 ZooKeeper 指定的 ZNode 上注册 Watcher 进行监听，一旦该 ZNode 的数据发生变化，ZooKeeper 就会向相应的订阅者发送事件通知，这一过程是通过 ZooKeeper 的 Watcher 机制实现的。ZooKeeper 的 Watcher 机制允许客户端向 ZooKeeper 注册 Watcher 以监听指定事件，一旦 Watcher 监听的事件被触发，

ZooKeeper 便会向客户端发送一个事件通知。ZooKeeper 的 Watcher 机制依赖于以下几点特性实现数据发布、订阅的功能。

1. 一次性

一次性指的是 Watcher 监听的事件一旦被触发,那么该 Watcher 就会被移除,所以 Watcher 需要反复注册才能使用。这样做的好处是可以有效减轻 ZooKeeper 的压力。试想,如果注册的 Watcher 一直有效,那么针对那些更新比较频繁的 ZNode,ZooKeeper 会不断向客户端发送事件通知,这对于网络 I/O 以及 ZooKeeper 性能的影响都非常大。

2. 客户端串行执行

客户端 Watcher 回调的过程是一个串行同步的过程,客户端接收到 ZooKeeper 发送的事件通知时,会通过回调实现对事件的处理,并且所有 Watcher 的回调都不会并发执行,而是按照回调顺序一个一个地执行。如果某个客户端 Watcher 回调的处理逻辑过于复杂,那么将影响其他客户端的 Watcher 回调。

3. 轻量级

WatchedEvent 是 Watcher 机制的最小事件通知单元,它的数据结构中只包含三部分内容,分别是通知状态、事件类型和结点路径。也就是说,ZooKeeper 发送的事件通知内容非常简单,只会告诉客户端发生了事件,而不会说明事件的具体内容。例如,注册在 ZooKeeper 的 Watcher 监听某个 ZNode 时,如果 ZNode 的数据发生变化,那么 ZooKeeper 发送的事件通知只会告知客户端 ZNode 的数据发生了变化,而不是告知数据发生了什么变化。

ZooKeeper 发送的事件通知由通知状态、事件类型和结点路径组成,其中,结点路径就是 Watcher 监听 ZNode 的路径,但不包含 ZNode 的数据;通知状态和事件类型用于标识 Watcher 监听的事件,常见的通知状态和事件类型如表 5-1 所示。

表 5-1 常见的通知状态和事件类型

通知状态	事件类型	事件
SyncConnected	None	客户端与 ZooKeeper 成功建立会话
	NodeCreated	Watcher 监听的 ZNode 被创建
	NodeDeleted	Watcher 监听的 ZNode 被删除
	NodeDataChanged	Watcher 监听的 ZNode 的数据发生变化
	NodeChildrenChanged	Watcher 监听的 ZNode 的子结点发生变化
Disconnected	None	客户端与 ZooKeeper 断开连接
Expired	None	客户端与 ZooKeeper 建立的会话连接超时

从表 5-1 可以看出,Watcher 监听的每种事件都由不同通知状态和事件类型的组合进行标识,例如通知状态 SyncConnected 和事件类型 None 的组合标识的事件为客户端与 ZooKeeper 成功建立会话。

5.5 ZooKeeper 的选举机制

ZooKeeper 的选举机制是指从 ZooKeeper 集群的多个服务器中选举出一个服务器作为 Leader。为了确保 ZooKeeper 能够成功选举出 Leader,ZooKeeper 集群的服务器数量要满足 2N+1(N 为正整数),即 ZooKeeper 集群最少需要 3 台服务器。ZooKeeper 触发选举机制的

情况有两种，一种是启动 ZooKeeper 集群时，通过触发选举机制选举 Leader；另一种是 ZooKeeper 集群运行期间，如果 Leader 宕机，那么可以通过触发选举机制选举出新的 Leader。

触发选举机制进行选举 Leader 的过程中，ZooKeeper 的 myid 和 zxid 会直接影响最终的选举结果，关于 myid 和 zxid 的介绍如下。

- myid。myid 可以称为服务器 ID，它是用户在部署 ZooKeeper 集群时，为每台服务器运行的 ZooKeeper 服务指定的编号，编号的值越大，ZooKeeper 服务被选举为 Leader 的权重越高。需要注意的是，用户为每个 ZooKeeper 服务指定的编号不能重复。
- zxid。zxid 可以称为事务 ID，它是一个 Long 型的 64 位整数，其中，低 32 位可以看作一个简单的单调递增的计数器，用于记录 ZooKeeper 服务处理事务请求的次数，每处理一次事务请求，计数器便会加 1；高 32 位表示时间戳，用于记录 Leader 周期的编号，每当选举产生一个新的 Leader 时，便会将时间戳加 1，并且重置计数器为 0。例如，启动 ZooKeeper 集群时，在未选举出 Leader 之前，所有 ZooKeeper 服务中 zxid 的时间戳都是 0，当选举出 Leader 之后，时间戳变为 1，若在后续 ZooKeeper 运行过程中 Leader 宕机了，重新选举出新的 Leader，那么此时除宕机的 Leader 之外，所有 ZooKeeper 服务中 zxid 的时间戳都会变为 2。

了解 myid 和 zxid 的概念之后，下面分别介绍 ZooKeeper 集群启动时和 ZooKeeper 集群运行期间触发选举机制选举 Leader 的过程。

1. ZooKeeper 集群启动时

假设某个 ZooKeeper 集群由 5 台服务器组成，这 5 台服务器的 myid 按照 1～5 编号，从服务器 1 开始，依此启动每台服务器的 ZooKeeper 服务，从而启动 ZooKeeper 集群，此时，选举 Leader 的过程如下。

（1）服务器 1 启动 ZooKeeper 服务，当 ZooKeeper 服务启动后，先投票给自己，并将投票信息发送给其他服务器启动的 ZooKeeper 服务，投票信息中包含自身的 myid 和 zxid，不过此时服务器 1 只有 1 票，不符合得票数大于服务器总数一半的规则，所以 ZooKeeper 集群还没有选举出 Leader。

（2）服务器 2 启动 ZooKeeper 服务，当 ZooKeeper 服务启动后，先投票给自己，然后将投票信息发送给其他服务器启动的 ZooKeeper 服务，并接收其他服务器发送的投票信息，当服务器 2 的 ZooKeeper 服务接收到服务器 1 的 ZooKeeper 服务发送的投票信息之后，会比较两者投票信息中的 zxid 和 myid，如果 zxid 相等，则比较 myid。由于启动 ZooKeeper 集群时 zxid 中的值为 0，而服务器 2 中 ZooKeeper 服务的 myid 大于服务器 1 中 ZooKeeper 服务的 myid，所以服务器 2 中 ZooKeeper 服务的投票不会改变。

（3）服务器 1 中的 ZooKeeper 服务接收到服务器 2 中 ZooKeeper 服务的投票信息之后，同样会比较两者投票信息中的 zxid 和 myid，由于两者 zxid 的值都等于 0，而服务器 2 中 ZooKeeper 服务的 myid 大于服务器 1 中 ZooKeeper 服务的 myid，所以服务器 1 中的 ZooKeeper 服务会将投给自身的票转投给服务器 2 的 ZooKeeper 服务，不过此时服务器 2 的投票数仍然不符合得票数大于服务器总数一半的规则。

（4）服务器 3 启动 ZooKeeper 服务，当 ZooKeeper 服务启动后，先投票给自己，然后将投票信息发送给其他服务器启动的 ZooKeeper 服务，并接收其他服务器发送的投票信息，当服务器 3 的 ZooKeeper 服务接收到服务器 1 和服务器 2 的 ZooKeeper 服务发送的投票信息之后，会依次比较三者投票信息中的 zxid 和 myid。由于启动 ZooKeeper 集群时 zxid 中的值为

0,而服务器 3 中 ZooKeeper 服务的 myid 大于服务器 1 和服务器 2 中 ZooKeeper 服务的 myid,所以服务器 3 中 ZooKeeper 服务的投票不会改变。

(5) 服务器 1 和服务器 2 中的 ZooKeeper 服务接收到服务器 3 中 ZooKeeper 服务的投票信息之后,同样会比较两者投票信息中的 zxid 和 myid。由于两者 zxid 的值都等于 0,而服务器 3 中 ZooKeeper 服务的 myid 大于服务器 1 和服务器 2 中的 ZooKeeper 服务的 myid,所以服务器 1 中的 ZooKeeper 服务会将投给服务器 2 中 ZooKeeper 服务的票转投给服务器 3 的 ZooKeeper 服务,并且服务器 2 的 ZooKeeper 服务会将投给自身的票转投给服务器 3 的 ZooKeeper 服务,此时服务器 3 的 ZooKeeper 服务获得了 3 张投票,符合得票数大于服务器总数一半的规则,因此服务器 3 的 ZooKeeper 服务被选举为 Leader,选举结束。

(6) 服务器 1 和服务器 2 已经启动的 ZooKeeper 服务,以及后续服务器 4 和服务器 5 启动的 ZooKeeper 服务都会成为 Follower。

2. ZooKeeper 集群运行期间

对于正常运行的 ZooKeeper 集群,一旦 Leader 角色的 ZooKeeper 服务中途宕机,则需要从其他角色为 Follower 的 ZooKeeper 服务中重新选举出一个新的 Leader。假设运行的 ZooKeeper 集群由 5 台服务器构成,服务器 3 中 ZooKeeper 服务的角色为 Leader,当服务器 3 中的 ZooKeeper 服务宕机时,选举 Leader 的过程如下。

(1) ZooKeeper 服务的角色为 Follower 的服务器 1、服务器 2、服务器 4 和服务器 5 会将自己的状态变为 Looking(竞选状态)。

(2) 服务器 1、服务器 2、服务器 4 和服务器 5 的 ZooKeeper 服务会先给自己投票,然后分别将自身的投票信息发送给其他服务器的 ZooKeeper 服务,由于 ZooKeeper 集群在运行过程中,每个 ZooKeeper 服务的 zxid 都可能不同,因此假定服务器 1、服务器 2、服务器 4 和服务器 5 的 ZooKeeper 服务中的 zxid 分别为 99、101、100 和 101。

(3) 服务器 1、服务器 2、服务器 4 和服务器 5 的 ZooKeeper 服务接收来自各个服务器的投票信息,并且将自身的投票信息与其他服务器中 ZooKeeper 服务发送的投票信息进行比较。

(4) 由于服务器 2 和服务器 5 中 ZooKeeper 服务的 zxid 大于服务器 1 和服务器 4 中 ZooKeeper 服务的 zxid,所以服务器 1 和服务器 4 的 ZooKeeper 服务会退出竞选,将角色变更为 Follower。

(5) 当服务器 1 和服务器 4 的 ZooKeeper 服务接收到服务器 2 和服务器 5 的 ZooKeeper 服务发送的投票信息后,最终会投票给服务器 5 的 ZooKeeper 服务,这是因为服务器 5 中 ZooKeeper 服务的 myid 大于服务器 2 中 ZooKeeper 服务的 myid,并且服务器 2 的 ZooKeeper 服务也会将投给自身的票转投给服务器 5 的 ZooKeeper 服务,此时服务器 5 的 ZooKeeper 服务获得了 3 张投票,符合得票数大于服务器总数一半的规则,因此服务器 5 的 ZooKeeper 服务被选举为新的 Leader,选举结束。

(6) 服务器 2 的 ZooKeeper 服务退出竞选,将角色变更为 Follower。

5.6 部署 ZooKeeper 集群

ZooKeeper 集群的部署模式分为伪分布式模式和完全分布式模式,其中,伪分布式模式是指在单台服务器中部署 ZooKeeper 集群,ZooKeeper 集群中的多个 ZooKeeper 服务运行在单

台服务器的不同进程中；完全分布式模式是指在多台服务器中部署 ZooKeeper 集群，ZooKeeper 集群中的每个 ZooKeeper 服务都运行在单独的服务器中。本节将分别演示如何在伪分布式模式和完全分布式模式下部署 ZooKeeper 集群。

5.6.1　基于伪分布式模式部署 ZooKeeper 集群

这里以虚拟机 Hadoop3 为例，演示如何基于伪分布式模式部署 ZooKeeper 集群，该集群由 3 个 ZooKeeper 服务组成，这 3 个 ZooKeeper 服务的角色分别是 Leader、Follower 和 Observer。

1. 下载 ZooKeeper 安装包

本书使用的 ZooKeeper 版本为 3.7.0，读者可自行到 ZooKeeper 官网下载 ZooKeeper 的安装包 apache-zookeeper-3.7.0-bin.tar.gz。

2. 创建目录

由于后续还会使用虚拟机 Hadoop3 部署完全分布式模式的 ZooKeeper 集群，所以为了避免安装目录产生冲突，这里在虚拟机 Hadoop3 的/export/servers 目录执行“mkdir zookeeper”命令，创建 zookeeper 目录，在该目录中存放基于伪分布式模式部署的 ZooKeeper 集群的安装目录。

3. 安装 ZooKeeper

将下载的 ZooKeeper 安装包 apache-zookeeper-3.7.0-bin.tar.gz 上传至虚拟机 Hadoop3 的/export/software/zookeeper 目录。上传完成后，将 ZooKeeper 安装包以解压缩的方式安装至/export/servers/zookeeper 目录，在/export/software 目录执行如下命令。

```
$ tar -zxvf apache-zookeeper-3.7.0-bin.tar.gz -C /export/servers/zookeeper
```

上述命令执行完成后，会在/export/servers/zookeeper 目录中生成 apache-zookeeper-3.7.0-bin 目录，该目录为 ZooKeeper 的安装目录，不过此时 ZooKeeper 集群还无法使用，还需要对其进行配置。

4. 复制 ZooKeeper 安装目录

为了在单台虚拟机中启动 3 个 ZooKeeper 服务，这里需要在虚拟机中安装 3 个 ZooKeeper，安装的每个 ZooKeeper 都会启动独立的 ZooKeeper 服务。出于便捷性的考虑，这里通过复制 ZooKeeper 安装目录的方式继续安装另外两个 ZooKeeper。另外，为了便于区分，这里分别将 3 个 ZooKeeper 的安装目录重命名为 zookeeper-3.7.0-001、zookeeper-3.7.0-002 和 zookeeper-3.7.0-003。在/export/servers/zookeeper 目录执行下列命令，复制 ZooKeeper 安装目录并重命名。

```
$ mv apache-zookeeper-3.7.0-bin zookeeper-3.7.0-001
$ cp -r zookeeper-3.7.0-001 zookeeper-3.7.0-002
$ cp -r zookeeper-3.7.0-001 zookeeper-3.7.0-003
```

5. 创建 ZooKeeper 配置文件

ZooKeeper 安装目录的 conf 目录下存放了 ZooKeeper 配置文件的模板文件 zoo_sample.cfg，该文件包含 ZooKeeper 的默认配置信息，如果需要修改 ZooKeeper 的默认配置，则需要将模板文件 zoo_sample.cfg 重命名为 zoo.cfg，并对 zoo.cfg 文件的内容进行修改。在 3 个 ZooKeeper 安装目录的 conf 目录下，依次执行如下命令，将模板文件 zoo_sample.cfg 重命名为 zoo.cfg。

```
$ cp zoo_sample.cfg zoo.cfg
```

6. 修改 ZooKeeper 配置文件

由于 3 个 ZooKeeper 服务运行在同一台虚拟机中,因此为了避免不同 ZooKeeper 服务使用的数据持久化目录和端口号产生冲突,这里分别修改 3 个 ZooKeeper 安装目录的配置文件。

首先,在/export/servers/zookeeper/zookeeper-3.7.0-001/conf 目录下执行“vi zoo.cfg”命令,编辑文件 zoo.cfg 修改其内容,文件 zoo.cfg 修改完成的效果如文件 5-1 所示。

文件 5-1　zoo.cfg

```
1   #The number of milliseconds of each tick
2   tickTime=2000
3   #The number of ticks that the initial
4   #synchronization phase can take
5   initLimit=10
6   #The number of ticks that can pass between
7   #sending a request and getting an acknowledgement
8   syncLimit=5
9   #the directory where the snapshot is stored.
10  #do not use /tmp for storage, /tmp here is just
11  #example sakes.
12  #设置数据持久化目录
13  dataDir=/export/data/zookeeper/zkdata001
14  #the port at which the clients will connect
15  #设置客户端连接当前 ZooKeeper 服务使用的端口号
16  clientPort=2181
17  #the maximum number of client connections.
18  #increase this if you need to handle more clients
19  #maxClientCnxns=60
20  #Be sure to read the maintenance section of the
21  #administrator guide before turning on autopurge.
22  #http://ZooKeeper.apache.org/doc/current/ZooKeeperAdmin.html#
23  #sc_maintenance
24  #The number of snapshots to retain in dataDir
25  #autopurge.snapRetainCount=3
26  #Purge task interval in hours
27  #Set to "0" to disable auto purge feature
28  #autopurge.purgeInterval=1
29  #设置 ZooKeeper 集群中每个 ZooKeeper 服务的地址及端口号
30  server.1=hadoop3:2888:3888
31  server.2=hadoop3:2889:3889
32  server.3=hadoop3:2890:3890:observer
```

文件 5-1 中,第 13 行代码指定 ZooKeeper 的数据持久化目录为/export/data/zookeeper/zkdata001。

第 16 行代码指定 ZooKeeper 服务使用的端口号为 2181。

第 30 行代码指定编号为 1 的 ZooKeeper 服务的地址为 hadoop3:2888:3888,其中,hadoop3 表示 ZooKeeper 服务所在的服务器主机名;2888 表示当前 ZooKeeper 服务与 Leader 进行通信的端口;3888 表示当前 ZooKeeper 选举 Leader 时使用的端口。

第 31 行代码指定编号为 2 的 ZooKeeper 服务的地址为 hadoop3:2889:3889。

第 32 行代码指定编号为 3 的 ZooKeeper 服务的地址为 hadoop3:2890:3890:observer,其中,添加的“:observer”用于指定当前 ZooKeeper 服务的角色为 Observer。