第5章 Zookeeper分布式协调服务

学习目标

- 了解 Zookeeper 的概念和特性。
- 理解 Zookeeper 数据模型。
- 掌握 Zookeeper 的 Watch 机制和选举机制。
- 掌握 Zookeeper 的集群部署。
- 掌握 Zookeeper 的 Shell 操作和 Java API 操作。
- 熟悉 Zookeeper 的应用场景。

构建分布式系统并不容易，然而，用户日常所使用的应用程序，如淘宝、支付宝等大多是基于分布式系统，分布式系统是建立在网络之上的软件系统，具有高度的内聚性和透明性，在短时间内依赖于分布式系统的现状依旧不会改变。

Apache Zookeeper 旨在减轻构建健壮的分布式系统的服务，它是基于分布式计算的核心概念而设计的，主要目的是给开发人员提供一套容易理解和开发的接口，从而简化分布式系统构建的服务。本章对 Zookeeper 的简介、Zookeeper 的数据模型、Zookeeper 的机制、Zookeeper 集群的部署、Zookeeper 的操作以及 Zookeeper 的典型应用场景进行详细讲解。

5.1 初识 Zookeeper

5.1.1 Zookeeper 简介

Zookeeper 起源于雅虎研究院的一个研究小组。当时研究人员发现，在雅虎内部很多大型系统基本都需要依赖一个类似的系统来进行分布式协调，但是这些系统往往都存在分布式单点问题。所谓单点问题，即在整个分布式系统中，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时，这个节点就被称为单点。一旦这台服务器宕机，整个分布式系统将无法正常运行，这种现象被称为单点故障。所以，雅虎的开发人员试图开发一个通用的无单点问题的分布式协调框架，以便让开发人员将精力集中在处理业务逻辑上。而 Zookeeper 正好是要用来进行分布式服务的协调。

Zookeeper 是一个分布式协调服务的开源框架，它是由 Google 的 Chubby 开源实现。Zookeeper 主要用来解决分布式集群中应用系统的一致性问题，如如何避免同时操作同一数据造成脏读的问题等。

Zookeeper 本质上是一个分布式的小文件存储系统，提供基于类似于文件系统的目录

树方式的数据存储，并且可以对树中的节点进行有效管理。从而用来维护和监控存储的数据的状态变化。通过监控这些数据状态的变化，从而达到基于数据的集群管理。如统一命名服务、分布式配置管理、分布式消息队列、分布式锁、分布式协调等功能。

5.1.2　Zookeeper 的特性

Zookeeper 具有全局数据一致性、可靠性、顺序性、原子性以及实时性，可以说 Zookeeper 的其他特性都是为满足 Zookeeper 全局数据一致性这一特性。具体介绍如下：

1. 全局数据一致性

每个服务器都保存一份相同的数据副本，客户端连接到集群的任意节点上，看到的目录树都是一致的（也就是数据都是一致的），这也是 Zookeeper 最重要的特征。

2. 可靠性

如果消息（对目录结构的增删改查）被其中一台服务器接收，那么将被所有的服务器接收。

3. 顺序性

Zookeeper 顺序性主要分为全局有序和偏序两种，其中全局有序是指如果在一台服务器上消息 A 在消息 B 前发布，则在所有服务器上消息 A 都将在消息 B 前被发布；偏序是指如果一个消息 B 在消息 A 后被同一个发送者发布，A 必将排在 B 前面。无论全局有序还是偏序，其目的都是为了保证 Zookeeper 全局数据一致。

4. 数据更新原子性

一次数据更新操作要么成功（半数以上节点成功），要么失败，不存在中间状态。

5. 实时性

Zookeeper 保证客户端将在一个时间间隔范围内获得服务器的更新信息，或者服务器失效的信息。

5.1.3　Zookeeper 集群角色

Zookeeper 对外提供一个类似于文件系统的层次化的数据存储服务，为了保证整个 Zookeeper 集群的容错性和高性能，每一个 Zookeeper 集群都是由多台服务器节点（Server）组成，这些节点通过复制保证各个服务器节点之间的数据一致。只要这些服务器节点过半数可用，那么整个 Zookeeper 集群就可用。下面我们来学习 Zookeeper 的集群架构，如图 5-1 所示。

从图 5-1 可以看出，Zookeeper 集群是一个主从集群，它一般是由一个 Leader（领导者）和多个 Follower（跟随者）组成。此外，针对访问量比较大的 Zookeeper 集群，还可新增 Observer（观察者）。Zookeeper 集群中的三种角色各司其职，共同完成分布式协调服务。下面我们针对 Zookeeper 集群中的三种角色进行简单介绍。

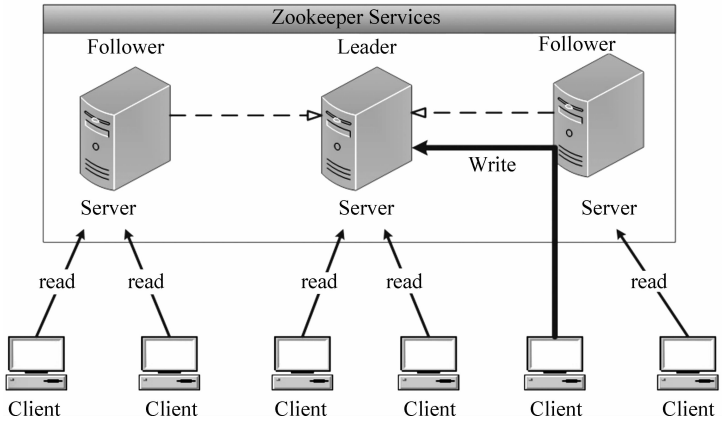

图 5-1 Zookeeper 集群架构图

1. Leader

它是 Zookeeper 集群工作的核心，也是事务性请求(写操作)的唯一调度和处理者，它保证集群事务处理的顺序性，同时负责进行投票的发起和决议，以及更新系统状态。

2. Follower

它负责处理客户端的非事务(读操作)请求，如果接收到客户端发来的事务性请求，则会转发给 Leader，让 Leader 进行处理，同时还负责在 Leader 选举过程中参与投票。

3. Observer

它负责观察 Zookeeper 集群的最新状态的变化，并且将这些状态进行同步。对于非事务性请求可以进行独立处理；对于事务性请求，则会转发给 Leader 服务器进行处理。它不会参与任何形式的投票，只提供非事务性的服务，通常用于在不影响集群事务处理能力的前提下，提升集群的非事务处理能力(提高集群读的能力，也降低了集群选主的复杂程度)。

5.2 数据模型

5.2.1 数据存储结构

Zookeeper 中数据存储的结构和标准文件系统非常类似，拥有一个层次的命名空间，也是使用斜杠(/)进行分隔，两者都是采用树状层次结构。不同的是，标准文件系统是由文件夹和文件来组成的树，而 Zookeeper 是由什么来组成的树呢？下面我们来看一下 Zookeeper 数据存储结构，如图 5-2 所示。

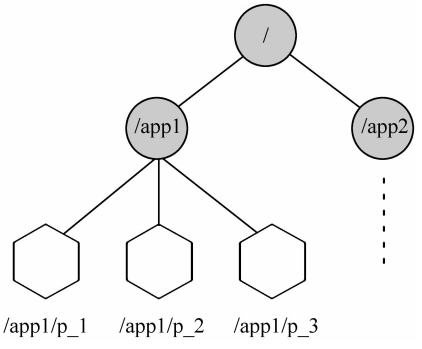

图 5-2 Zookeeper 数据模型

从图 5-2 可知，Zookeeper 是由节点组成的树，树中的每个节点被称为 Znode。每个节点都可以拥

有子节点。每一个 Znode 默认能够存储 1MB 的数据，每个 Znode 都可以通过其路径唯一标识，如图 5-2 中第三层的第一个 Znode，它的路径是/app1/p_1。Zookeeper 数据模型中的每个 Znode 都是由 3 部分组成，分别是 stat(状态信息，描述该 Znode 的版本，权限信息等组成)、data(与该 Znode 关联的数据)和 children(该 Znode 下的子节点)。

5.2.2　Znode 的类型

在 5.2.1 节中，我们初步了解了什么是 Znode，下面，我们来介绍一下 Znode 的类型。节点的类型在创建时被指定，一旦创建就无法改变。Znode 有两种类型，分别是临时节点和永久节点。

临时节点，该生命周期依赖于创建它们的会话，一旦会话结束，临时节点将会被自动删除，当然也可以手动删除。虽然每个临时的 Znode 都会绑定到一个客户端，但它们对所有的客户端还是可见的。另外，需要注意的是临时节点不允许拥有子节点。

永久节点，该生命周期不依赖于会话，并且只有在客户端显示执行删除操作的时候，它们才能被删除。

由于 Znode 的序列化特性，在创建节点时，用户可以请求在该 Znode 的路径结尾添加一个不断增加的序列号，序列号对于此节点的父节点来说是唯一的，这样便会记录每个子节点创建的先后顺序。它的格式为“%010d”(10 位数字，没有数值的数位用 0 补充，如 0000000001)。当计数值大于 $2^{32}-1$ 时，计数器将会溢出。这样便会存在 4 种类型的目录节点，分别对应如下。

- PERSISTENT：永久节点；
- EPHEMERAL：临时节点；
- PERSISTENT_SEQUENTIAL：序列化永久节点；
- EPHEMERAL_SEQUENTIAL：序列化临时节点。

5.2.3　Znode 的属性

每个 Znode 都包含了一系列的属性，接下来详细讲解 Znode 的属性，如表 5-1 所示。

表 5-1　Zookeeper 节点属性

属 性 名 称	属 性 描 述
czxid	节点被创建的 Zxid 值
ctime	节点被创建的时间
mzxid	节点最后一次修改的 Zxid 值
mtime	节点最后一次的修改时间
pZxid	与该节点的子节点最后一次修改的 Zxid 值
cversion	子节点被修改的版本号
dataVersion	数据版本号
aclVersion	ACL 版本号

续表

属性名称	属性描述
ephemeralOwner	如果此节点为临时节点，那么该值代表这个节点拥有者的会话 ID；否则值为 0
dataLength	节点数据域长度
numChildren	节点拥有的子节点个数

表 5-1 介绍了 Znode 的属性，对于 Zookeeper 来说，Znode 状态改变的每一个操作都将使节点接收到唯一的 zxid(Zookeeper Transaction ID)格式的时间戳，并且这个时间戳是全局有序的，通常被称为事物 ID，通过 zxid，可以确定更新操作的先后顺序，例如，如果 zxid1 小于 zxid2，说明 zxid1 操作先于 zxid2 发生。

5.3 Zookeeper 的 Watch 机制

5.3.1 Watch 机制的简介

ZooKeeper 提供了分布式数据发布/订阅功能，一个典型的发布/订阅模型系统定义了一种一对多的订阅关系，能让多个订阅者同时监听某一个主题对象，当这个主题对象自身状态变化时，会通知所有订阅者，使他们能够做出相应的处理。

在 ZooKeeper 中，引入了 Watch 机制来实现这种分布式的通知功能。ZooKeeper 允许客户端向服务端注册一个 Watch 监听，当服务端的一些事件触发了这个 Watch，那么就会向指定客户端发送一个事件通知，来实现分布式的通知功能。

5.3.2 Watch 机制的特点

1. 一次性触发

当 Watch 的对象发生改变时，将会触发此对象上 Watch 所对应的事件，这种监听是一次性的，后续再次发生同样的事件，也不会再次触发。

2. 事件封装

Zookeeper 使用 WatchedEvent 对象来封装服务端事件并传递。该对象包含了每个事件的 3 个基本属性，即通知状态(keeperState)、事件类型(EventType)和节点路径(path)。

3. 异步发送

Watch 的通知事件是从服务端异步发送到客户端的。

4. 先注册再触发

Zookeeper 中的 Watch 机制，必须由客户端先去服务端注册监听，这样才会触发事件的监听，并通知给客户端。

5.3.3　Watch 机制的通知状态和事件类型

同一个事件类型在不同的连接状态中代表的含义有所不同，表 5-2 列举了常见的连接状态和事件类型。

表 5-2　Zookeeper 连接状态和事件类型

连接状态	状态含义	事件类型	事件含义
Disconnected	连接失败	NodeCreated	节点被创建
SyncConnected	连接成功	NodeDataChanged	节点数据变更
AuthFailed	认证失败	NodeChildrenChanged	子节点数据变更
Expired	会话过期	NodeDeleted	节点被删除

从表 5-2 可知，Zookeeper 常见的连接状态和事件类型分别有 4 种，具体含义如下。

当客户端断开连接，这时客户端和服务器的连接就是 Disconnected 状态，说明连接失败；当客户端和服务器的某一个节点建立连接，并完成一次 version、zxid 的同步，这时客户端和服务器的连接状态就是 SyncConnected，说明连接成功；当 Zookeeper 客户端连接认证失败，这时客户端和服务器的连接状态就是 AuthFailed，说明认证失败；当客户端发送 Request 请求，通知服务器其上一个发送心跳的时间，服务器收到这个请求后，通知客户端下一个发送心跳的时间是哪个时间点。当客户端时间戳达到最后一个发送心跳的时间，而没有收到服务器发来的新发送心跳的时间，即认为自己下线，这时客户端和服务器的连接状态就是 Expired 状态，说明会话过期。

当节点被创建时，NodeCreated 事件被触发；当节点的数据发生变更时，NodeDataChanged 事件被触发；当节点的直接子节点被创建、被删除、子节点数据发生变更时，NodeChildrentChanged 事件被触发；当节点被删除时，NodeDeleted 事件被触发。

5.4　Zookeeper 的选举机制

5.4.1　选举机制的简介

Zookeeper 为了保证各节点的协同工作，在工作时需要一个 Leader 角色，而 Zookeeper 默认采用 FastLeaderElection 算法，且投票数大于半数则胜出的机制，再介绍选举机制前，首先了解选举涉及的相关概念。

1. 服务器 ID

这是在配置集群时设置的 myid 参数文件，且参数分别表示为服务器 1、服务器 2 和服务器 3，编号越大在 FastLeaderElection 算法中的权重越大。

2. 选举状态

在选举过程中，Zookeeper 服务器有 4 种状态，它们分别为竞选状态（LOOKING）、随从状态（FOLLOWING，同步 leader 状态，参与投票）、观察状态（OBSERVING，同步 leader 状

态,不参与投票)和领导者状态(LEADING)。

3. 数据 ID

这是服务器中存放的最新数据版本号,该值越大说明数据越新,在选举过程中数据越新权重越大。

4. 逻辑时钟

通俗地讲,逻辑时钟被称为投票次数,同一轮投票过程中的逻辑时钟值是相同的,逻辑时钟起始值为 0,每投完一次票,这个数据就会增加。然后,与接收到其他服务器返回的投票信息中的数值相比较,根据不同的值做出不同的判断。如果某台机器宕机,那么这台机器不会参与投票,因此逻辑时钟也会比其他的低。

5.4.2 选举机制的类型

Zookeeper 选举机制有两种类型,分别为全新集群选举和非全新集群选举,下面分别对两种类型进行详细讲解。

1. 全新集群选举

全新集群选举是新搭建起来的,没有数据 ID 和逻辑时钟来影响集群的选举。假设,目前有 5 台服务器,它们的编号分别是 1～5,按编号依次启动 Zookeeper 服务。下面来讲解全新集群选举的过程。

步骤 1:服务器 1 启动,首先,会给自己投票;其次,发投票信息,由于其他机器还没有启动所以它无法接收到投票的反馈信息,因此服务器 1 的状态一直属于 LOOKING 状态。

步骤 2:服务器 2 启动,首先,会给自己投票;其次,在集群中启动 Zookeeper 服务的机器发起投票对比,这时它会与服务器 1 交换结果,由于服务器 2 的编号大,所以服务器 2 胜出,此时服务器 1 会将票投给服务器 2,但此时服务器 2 的投票数并没有大于集群半数(2＜5/2),所以两个服务器的状态依然是 LOOKING 状态。

步骤 3:服务器 3 启动,首先,会给自己投票;其次,与之前启动的服务器 1 和服务器 2 交换信息,由于服务器 3 的编号最大所以服务器 3 胜出,那么服务器 1 和 2 会将票投给服务器 3,此时投票数正好大于半数(3＞5/2),所以服务器 3 成为领导者状态,服务器 1 和 2 成为追随者状态。

步骤 4:服务器 4 启动,首先,给自己投票;其次,与之前启动的服务器 1、2 和 3 交换信息,尽管服务器 4 的编号大,但是服务器 3 已经胜出。所以服务器 4 只能成为追随者状态。

步骤 5:服务器 5 启动,同服务器 4 一样,均成为追随者状态。

2. 非全新集群选举

对于正常运行的 Zookeeper 集群,一旦中途有服务器宕机,则需要重新选举时,选举的过程中就需要引入服务器 ID、数据 ID 和逻辑时钟。这是由于 Zookeeper 集群已经运行过一段时间,那么服务器中就会存在运行的数据。下面来讲解非全新集群选举的过程。

步骤 1:首先,统计逻辑时钟是否相同,逻辑时钟小,则说明途中可能存在宕机问题,因

此数据不完整，那么该选举结果被忽略，重新投票选举；

步骤 2：其次，统一逻辑时钟后，对比数据 ID 值，数据 ID 反应数据的新旧程度，因此数据 ID 大的胜出；

步骤 3：如果逻辑时钟和数据 ID 都相同的情况下，那么比较服务器 ID(编号)，值大则胜出；

简单地讲，非全新集群选举时是优中选优，保证 Leader 是 Zookeeper 集群中数据最完整、最可靠的一台服务器。

5.5　Zookeeper 分布式集群部署

Zookeeper 分布式集群部署指的是 Zookeeper 分布式模式安装。Zookeeper 集群搭建通常是由 $2n+1$ 台服务器组成，这是为了保证 Leader 选举(基于 Paxos 算法的实现)能够通过半数以上服务器选举支持，因此，Zookeeper 集群的数量一般为奇数。

5.5.1　Zookeeper 安装包的下载安装

由于 Zookeeper 集群的运行需要 Java 环境支持，所以需要提前安装 JDK。本章讲解的是 Leader+Follower 模式的 Zookeeper 集群。这里我们选择 Zookeeper 的版本是 3.4.10。具体下载安装步骤如下。

1. 下载 Zookeeper 安装包

Zookeeper 的下载地址为 http://mirror.bit.edu.cn/apache/zookeeper/zookeeper-3.4.10/。

2. 上传 Zookeeper 安装包

将下载完毕的 Zookeeper 安装包上传至 Linux 系统的/export/software/目录下。

3. 解压 Zookeeper 安装包

首先，我们进入安装包目录，具体命令如下：

```
$ cd /export/software/
```

其次，解压安装包 zookeeper-3.4.10.tar.gz 至/export/servers/目录，具体命令如下：

```
$ tar -zxvf zookeeper-3.4.10.tar.gz -C /export/servers/
```

安装包解压完毕，也就是 Zookeeper 的安装结束。但是，并不意味着 Zookeeper 集群的部署已经结束，还需要对其进行配置和启动，若是启动成功，才算是 Zookeeper 集群部署成功。

5.5.2　Zookeeper 相关配置

在上一节中，已经把 Zookeeper 的安装包成功解压至/export/servers 目录下。接下来，开始配置 Zookeeper 集群。

1. 修改 Zookeeper 的配置文件

首先，进入 Zookeeper 解压目录下的 conf 目录，复制配置文件 zoo_sample.cfg 并重命名为 zoo.cfg，具体命令如下：

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

其次，修改配置文件 zoo.cfg，分别设置 dataDir 目录，配置服务器编号与主机名映射关系，设置与主机连接的心跳端口和选举端口，具体配置内容如下：

```
# The number of milliseconds of each tick
# 设置通信心跳数
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
# 设置初始通信时限
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
# 设置同步通信时限
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
# 设置数据文件目录+数据持久化路径/
dataDir=/export/data/zookeeper/zkdata
# the port at which the clients will connect
# 设置客户端连接的端口号
clientPort=2181
# the maximum number of client connections.
# increase this if you need to handle more clients
# maxClientCnxns=60
# Be sure to read the maintenance section of the
# administrator guide before turning on autopurge.
# http://zookeeper.apache.org/doc/current/zookeeperAdmin.html# sc_maintenance
# The number of snapshots to retain in dataDir
# autopurge.snapRetainCount=3
# Purge task interval in hours
# Set to "0" to disable auto purge feature
# autopurge.purgeInterval=1
# 配置 Zookeeper 集群的服务器编号以及对应的主机名、选举端口号和通信端口号(心跳端口号)
server.1=hadoop01:2888:3888
server.2=hadoop02:2888:3888
server.3=hadoop03:2888:3888
```

小提示：配置文件 zoo.cfg 中的参数 server.1＝hadoop01:2888:3888，其中，1 表示服务器的编号；hadoop01 表示这个服务器的 IP 地址；2888 表示 Leader 选举的端口号；3888 表示 Zookeeper 服务器之间的通信端口号(心跳端口号)。

2. 创建 myid 文件

首先，根据配置文件 zoo.cfg 中设置的 dataDir 目录，创建 zkdata 文件夹，具体命令如下：

```
$ mkdir -p /export/data/zookeeper/zkdata
```

其次，在 zkdata 文件夹下创建 myid 文件，该文件里面的内容就是服务器编号（hadoop01 服务器对应编号 1，hadoop02 服务器对应编号 2，hadoop03 服务器对应编号 3），具体命令如下：

```
$ cd /export/data/zookeeper/zkdata
$ echo 1>myid
```

3. 配置环境变量

Linux 系统目录/etc 下的文件 profile 里面的内容都是与 Linux 环境变量相关的。所以，一般配置环境变量都是在 profile 文件里面。执行命令 vi /etc/profile 对 profile 文件进行修改，添加 Zookeeper 的环境变量，具体命令如下：

```
export ZK_HOME=/export/servers/zookeeper-3.4.10
export PATH=$PATH:$JAVA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$ZK_HOME/bin
```

4. 分发 Zookeeper 相关文件至其他服务器

首先，将 Zookeeper 安装目录分发至 hadoop02 和 hadoop03 服务器上。具体命令如下：

```
$ scp -r /export/servers/zookeeper-3.4.10/hadoop02:/export/servers/
$ scp -r /export/servers/zookeeper-3.4.10/hadoop03:/export/servers/
```

其次，将 myid 文件分发至 hadoop02 和 hadoop03 服务器上，并且修改 myid 的文件内容，依次对应服务器号进行设置，分别为 2 和 3。具体命令如下：

```
$ scp -r /export/data/zookeeper/hadoop02:/export/data/
$ scp -r /export/data/zookeeper/hadoop03:/export/data/
```

最后，将 profile 文件也分发至 hadoop02 和 hadoop03 服务器上。具体命令如下：

```
$ scp /etc/profile hadoop02:/etc/profile
$ scp /etc/profile hadoop03:/etc/profile
```

5. 环境变量生效

分别在 hadoop01、hadoop02 和 hadoop03 服务器上刷新 profile 配置文件，使环境变量