获取教材和讲义 PPT 等各种课程资料请访问 http://dblab.xmu.edu.cn/node/422

=课程教材由林子雨老师根据网络资料编著=

厦门大学计算机科学系教师 林子雨 编著

http://www.cs.xmu.edu.cn/linziyu

2013 年 9 月

前言

本教程由厦门大学计算机科学系教师林子雨编著，可以作为计算机专业研究生课程《大数据技术基础》的辅助教材。

本教程的主要内容包括：大数据概述、大数据处理模型、大数据关键技术、大数据时代面临的新挑战、NoSQL 数据库、云数据库、Google Spanner、Hadoop、HDFS、HBase、MapReduce、Zookeeper、流计算、图计算和 Google Dremel 等。

本教程是林子雨通过大量阅读、收集、整理各种资料后精心制作的学习材料，与广大数据库爱好者共享。教程中的内容大部分来自网络资料和书籍，一部分是自己撰写。对于自写内容，林子雨老师拥有著作权。

本教程 PDF 文档及其全套教学 PPT 可以通过网络免费下载和使用（下载地址：http://dblab.xmu.edu.cn/node/422）。教程中可能存在一些问题，欢迎读者提出宝贵意见和建议！

本教程已经应用于厦门大学计算机科学系研究生课程《大数据技术基础》，欢迎访问 2013 班级网站 http://dblab.xmu.edu.cn/node/423。

林子雨的E-mail是：ziyulin@xmu.edu.cn。

林子雨的个人主页是：http://www.cs.xmu.edu.cn/linziyu。

林子雨于厦门大学海韵园

2013 年 9 月

第 6 章 Zookeeper

厦门大学计算机科学系教师 林子雨 编著

个人主页：http://www.cs.xmu.edu.cn/linziyu

课程网址：http://dblab.xmu.edu.cn/node/422

2013 年 9 月

第 6 章 Zookeeper

Zookeeper 是 Hadoop 的一个子项目，是一种分布式的、开源的、应用于分布式应用的协作服务，主要是用来解决分布式应用中经常遇到的一些数据管理问题，如：统一命名服务、状态同步服务、集群管理、分布式应用配置项的管理等。Zookeeper 很容易编程接入，它使用了一个和文件树结构相似的数据模型，可以使用 Java 或者 C 来进行编程接入。

在分布式应用中，由于工程师不能很好地使用锁机制，以及基于消息的协调机制不适合在某些应用中使用，因此，需要有一种可靠的、可扩展的、分布式的、可配置的协调机制来统一系统的状态。Zookeeper 的目的就在于此。

本章介绍 Zookeeper 的相关知识，内容要点如下：

- Zookeeper 简介
- Zookeeper 的工作原理
- Zookeeper 的数据模型
- Zookeeper 的典型应用场景

6.1 Zookeeper 简介

Zookeeper 是一种分布式的、开源的、应用于分布式应用的协作服务，它提供了一些简单的操作，可以实现统一命名服务、状态同步服务、集群管理、分布式应用配置项的管理等。Zookeeper 很容易编程接入，它使用了一个和文件树结构相似的数据模型，可以使用 Java 或者 C 来进行编程接入。

众所周知，分布式的系统协作服务很难有让人满意的产品。这些协作服务产品很容易陷入一些诸如竞争选择条件或者死锁的陷阱中，而 Zookeeper 则可以很好地实现协作服务。

6.1.1 系统架构

Zookeeper 不仅可以单机提供服务，同时也支持多机组成集群来提供服务（如图 6-1 所示）。实际上 Zookeeper 还支持另外一种伪集群的方式，也就是可以在一台物理机上运行多

个 Zookeeper 实例。

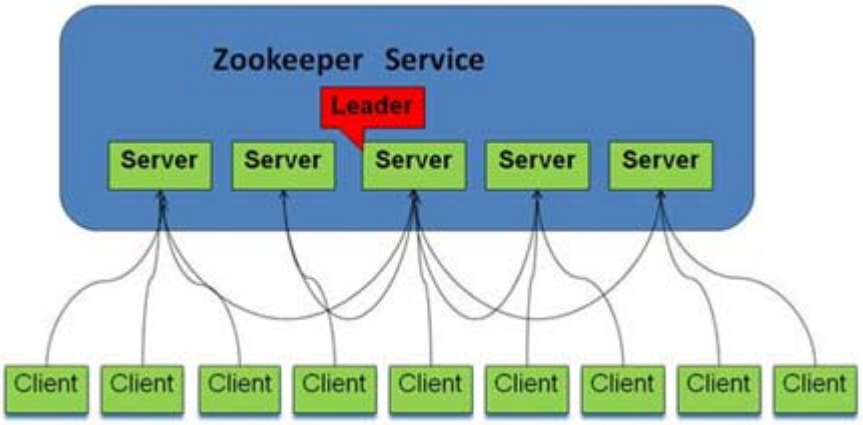

图 6-1 Zookeeper 的系统架构图

Zookeeper 中的角色主要包括领导者、学习者和客户端，如表 6-1 所示。

表 6-1 Zookeeper 中的角色

角色		描述
领导者（Leader）		领导者负责进行投票的发起和决议，更新系统状态
学习者（Learner）	跟随者（Follower）	Follower 用于接收客户请求并向客户端返回结果，在选主过程中参与投票
	观察者（ObServer）	ObServer 可以接收客户端连接，将写请求转发给 leader 节点。但 ObServer 不参加投票过程，只同步 leader 的状态。ObServer 的目的是为了扩展系统，提高读取速度
客户端（Client）		请求发起方

6.1.2 设计目的

Zookeeper 的设计目的包括以下几个方面：

- **最终一致性**：客户端不论连接到哪个服务器，展示给它的都是同一个视图，这是 Zookeeper 最重要的性能；
- **可靠性**：具有简单、健壮、良好的性能，如果消息被其中一台服务器接受，那么它将被所有的服务器接受；
- **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；但由于网络延时等原因，Zookeeper 不能保证两个客户端能同时得到刚更新的数据，如果需要最新数据，应该在读数据之前调用 sync()接口；

- **等待无关（wait-free）**：慢的或者失效的客户端不得干预快速的客户端的请求，使得每个客户端都能有效的等待；
- **原子性**：更新只能成功或者失败，没有中间状态；
- **顺序性**：包括全局有序和偏序两种：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。

6.1.3 特点

Zookeeper 的特点主要包括以下几个方面：

- **Zookeeper 是简易的**

Zookeeper 通过一种和文件系统很像的层级命名空间来让分布式进程互相协同工作。这些命名空间由一系列数据寄存器组成，我们也叫这些数据寄存器为 znodes。这些 znodes 就有点像是文件系统中的文件和文件夹。和文件系统不一样的是，文件系统的文件是存储在存储区上的，而 Zookeeper 的数据是存储在内存上的。同时，这就意味着 Zookeeper 有着高吞吐和低延迟。

Zookeeper 实现了高性能、高可靠性和有序的访问。高性能保证了 Zookeeper 能应用在大型的分布式系统上。高可靠性保证它不会由于单一节点的故障而造成任何问题。有序的访问能保证客户端可以实现较为复杂的同步操作。

- **Zookeeper 是可用的**

组成 Zookeeper 的各个服务器必须要能相互通信。他们在内存中保存了服务器状态，也保存了操作的日志，并且持久化快照。只要大多数的服务器是可用的，那么 Zookeeper 就是可用的。

客户端连接到一个 Zookeeper 服务器，并且维持 TCP 连接，发送请求，获取回复，获取事件，并且发送连接信号。如果这个 TCP 连接断掉了，那么，客户端还可以连接另外一个服务器。

- **Zookeeper 是有序的**

Zookeeper 使用数字来对每一个更新进行标记，这样能保证 Zookeeper 交互的有序。后续的操作可以根据这个顺序实现诸如同步操作这样更高更抽象的服务。

- **Zookeeper 是高效的**

Zookeeper 的高效更表现在以读为主的系统上。Zookeeper 可以在上千台服务器组成的读写比例大约为 10:1 的分布系统上表现优异。

6.2 Zookeeper 的工作原理

Zookeeper 的核心是原子广播，这个机制保证了各个服务器之间的同步。实现这个机制的协议叫做 Zab 协议。Zab 协议有两种模式，即恢复模式（选主）和广播模式（同步）。当服务启动或者在领导者崩溃后，Zab 就进入了恢复模式；当领导者被选举出来，且大多数 server 完成了和 leader 的状态同步以后，恢复模式就结束了。状态同步保证了 leader 和 server 具有相同的系统状态。

为了保证事务的顺序一致性，Zookeeper 采用了递增的事务 id 号（zxid）来标识事务。所有的提议（proposal）都在被提出的时候加上了 zxid。在实现时，zxid 是一个 64 位的数字，它的高 32 位是 epoch，用来表示 leader 关系是否改变，每次一个 leader 被选出来时，它都会有一个新的 epoch，表示当前属于那个 leader 的统治时期。低 32 位用于递增计数。

每个 server 在工作过程中有三种状态：

- LOOKING：当前 server 不知道 leader 是谁，正在搜寻；
- LEADING：当前 server 即为选举出来的 leader；
- FOLLOWING：leader 已经选举出来，当前 server 与之同步。

6.2.1 选主流程

当 leader 崩溃或者 leader 失去大多数的 follower，这时候 Zookeeper 就会进入恢复模式。恢复模式需要重新选举出一个新的 leader，让所有的 server 都恢复到一个正确的状态。Zookeeper 的选举算法有两种：一种是基于 basic paxos 实现的，另外一种是基于 fast paxos 算法实现的。系统默认的选举算法为 fast paxos。这里先介绍 basic paxos 流程：

- 选举线程由当前 Server 发起选举的线程担任，其主要功能是对投票结果进行统计，并选出推荐的 Server；
- 选举线程首先向所有 Server 发起一次询问(包括自己)；

- 选举线程收到回复后，验证是否是自己发起的询问(验证 zxid 是否一致)，然后获取对方的 id(myid)，并存储到当前询问对象列表中，最后获取对方提议的 leader 相关信息(id,zxid)，并将这些信息存储到当次选举的投票记录表中；
- 收到所有 Server 回复以后，就计算出 zxid 最大的那个 Server，并将这个 Server 相关信息设置成下一次要投票的 Server；
- 选举线程将当前 zxid 最大的 Server 设置为当前 Server 要推荐的 Leader 后，如果此 Server 获得 n/2 + 1 的 Server 票数，则设置当前推荐的 leader 为获胜的 Server，产生选举结果 leader，然后，根据获胜的 Server 相关信息设置自己的状态；否则，继续这个过程，直到 leader 被选举出来。

通过流程分析我们可以得出：要使 leader 获得多数 Server 的支持，则 Server 总数必须是奇数 2n+1，且存活的 Server 的数目不得少于 n+1。

每个 Server 启动后都会重复以上流程。在恢复模式下，如果是刚从崩溃状态恢复的或者刚启动的 Server，还会从磁盘快照中恢复数据和会话信息，Zookeeper 会记录事务日志并定期进行快照，方便在恢复时进行状态恢复。选主的具体流程图如图 6-2 所示。

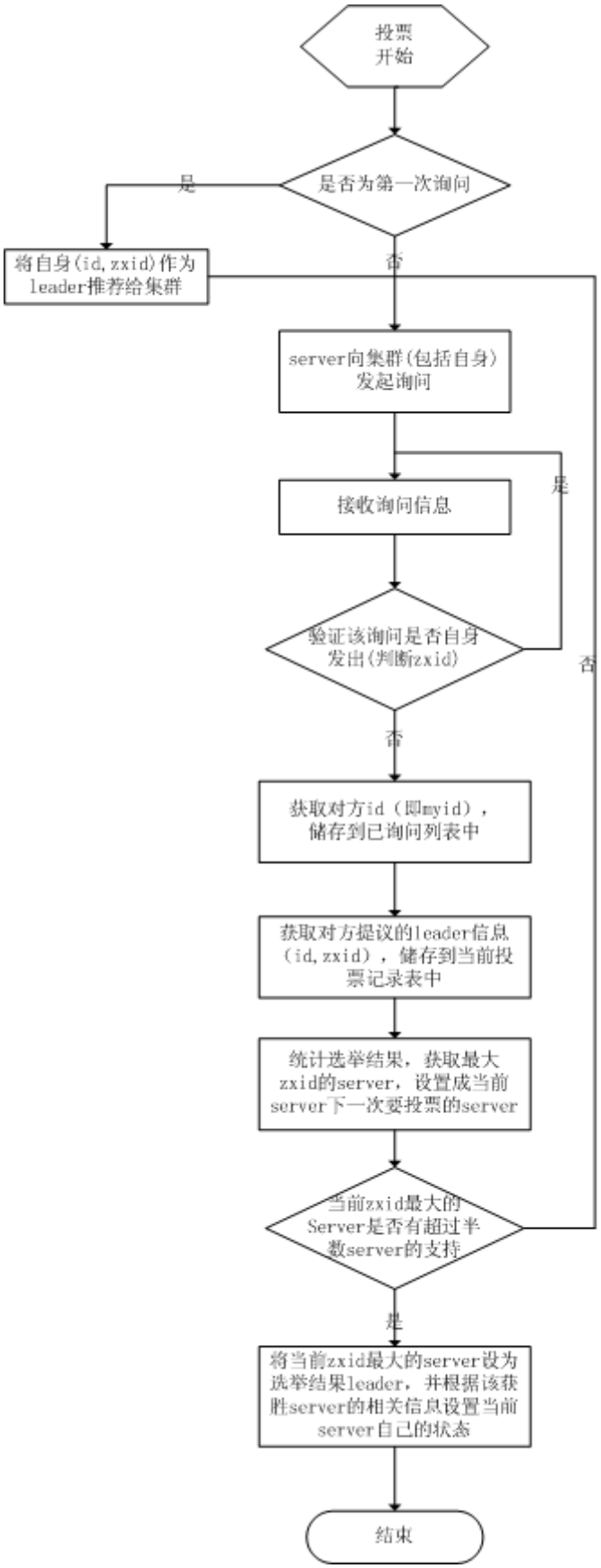

图 6-2 采用 basic paxos 算法实现选主流程

fast paxos 流程是在选举过程中，某 Server 首先向所有 Server 提议自己要成为 leader，当其它 Server 收到提议以后，解决 epoch 和 zxid 的冲突，并接受对方的提议，然后向对方发送接受提议完成的消息。重复这个流程，最后一定能选举出 leader。其流程图如图 6-3 所示。

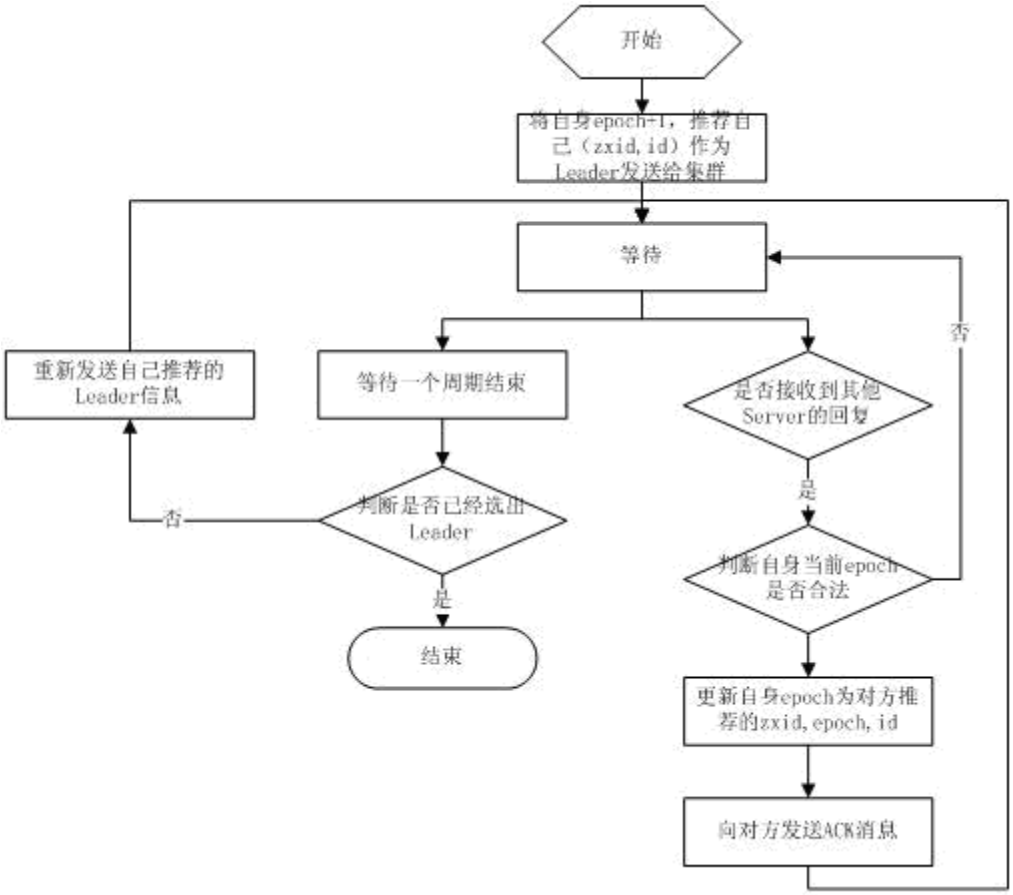

图 6-3 采用 fast paxos 算法实现选主流程

6.2.2 同步流程

选完 leader 以后，Zookeeper 就进入状态同步过程，具体如下：

- leader 等待 server 连接；
- Follower 连接 leader，将最大的 zxid 发送给 leader；
- Leader 根据 follower 的 zxid 确定同步点；
- 完成同步后通知 follower 已经成为 uptodate 状态；
- Follower 收到 uptodate 消息后，又可以重新接受 client 的请求进行服务了。

流程图如图 6-4 所示。

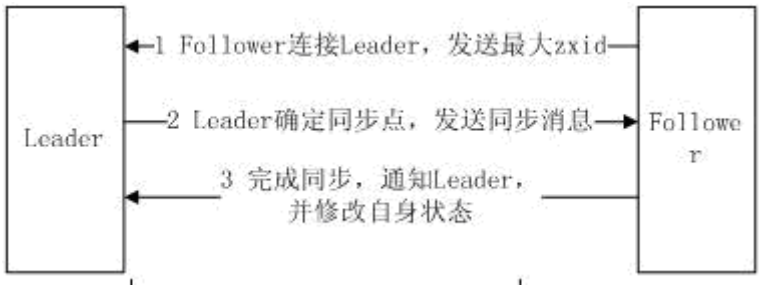

图 6-4 同步过程

6.2.3 工作流程

6.2.3.1 Leader 工作流程

Leader 主要有三个功能：

- 恢复数据；
- 维持与 Learner 的心跳，接收 Learner 请求并判断 Learner 的请求消息类型；
- Learner 的消息类型主要有 PING 消息、REQUEST 消息、ACK 消息、REVALIDATE 消息，根据不同的消息类型，进行不同的处理。

PING 消息是指 Learner 的心跳信息；REQUEST 消息是 Follower 发送的提议信息，包括写请求及同步请求；ACK 消息是 Follower 的对提议的回复，超过半数的 Follower 通过，则 commit 该提议；REVALIDATE 消息是用来延长 SESSION 有效时间。

Leader 的工作流程简图如图 6-5 所示，在实际实现中，流程要比该图复杂得多，启动了三个线程来实现功能。

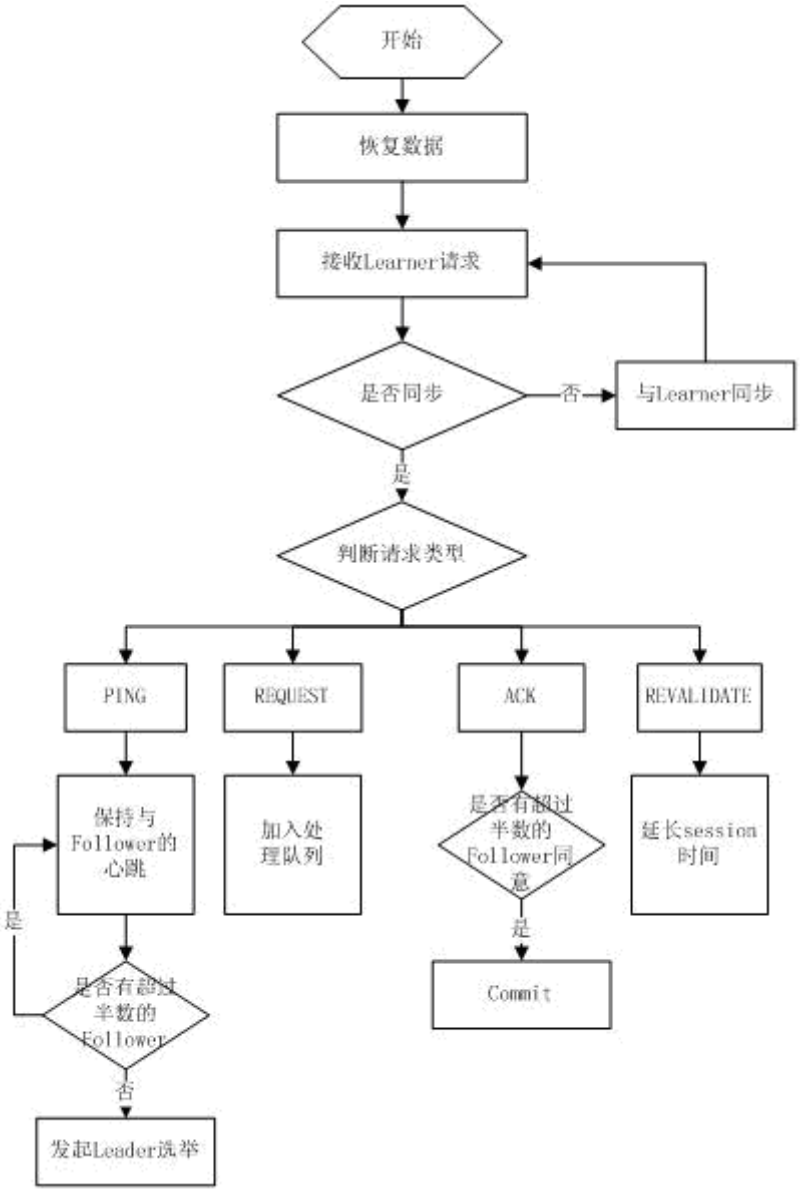

图 6-5 Leader 的工作流程

6.2.3.2 Follower 工作流程

Follower 主要有四个功能：

- 向 Leader 发送请求（PING 消息、REQUEST 消息、ACK 消息、REVALIDATE 消息）；
- 接收 Leader 消息并进行处理；
- 接收 Client 的请求，如果为写请求，发送给 Leader 进行投票；
- 返回 Client 结果。

Follower 循环处理如下几种来自 Leader 的消息：

- **PING** 消息： 心跳消息；
- **PROPOSAL** 消息：Leader 发起的提案，要求 Follower 投票；
- **COMMIT** 消息：服务器端最新一次提案的信息；
- **UPTODATE** 消息：表明同步完成；
- **REVALIDATE** 消息：根据 Leader 的 REVALIDATE 结果，来决定关闭待 revalidate 的 session 还是允许其接受消息；
- **SYNC** 消息：返回 SYNC 结果到客户端，这个消息最初由客户端发起，用来强制得到最新的更新。

Follower 的工作流程简图如图 6-6 所示，在实际实现中，Follower 是通过 5 个线程来实现功能的。

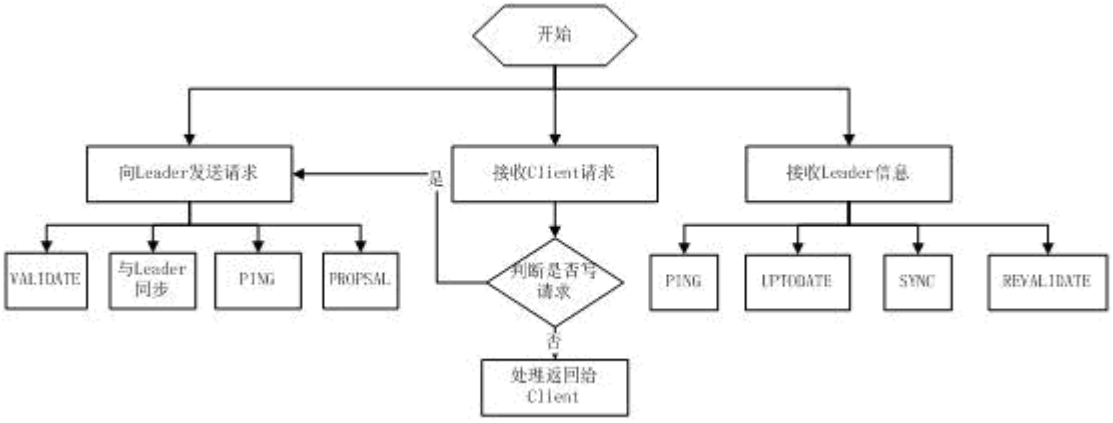

图 6-6 Follower 的工作流程

对于 observer 的流程不再叙述，observer 流程和 Follower 的唯一不同的地方就是 observer 不会参加 leader 发起的投票。

6.3 Zookeeper 的数据模型

Zookeeper 维护一个类似文件系统的数据结构，如图 6-7 所示。

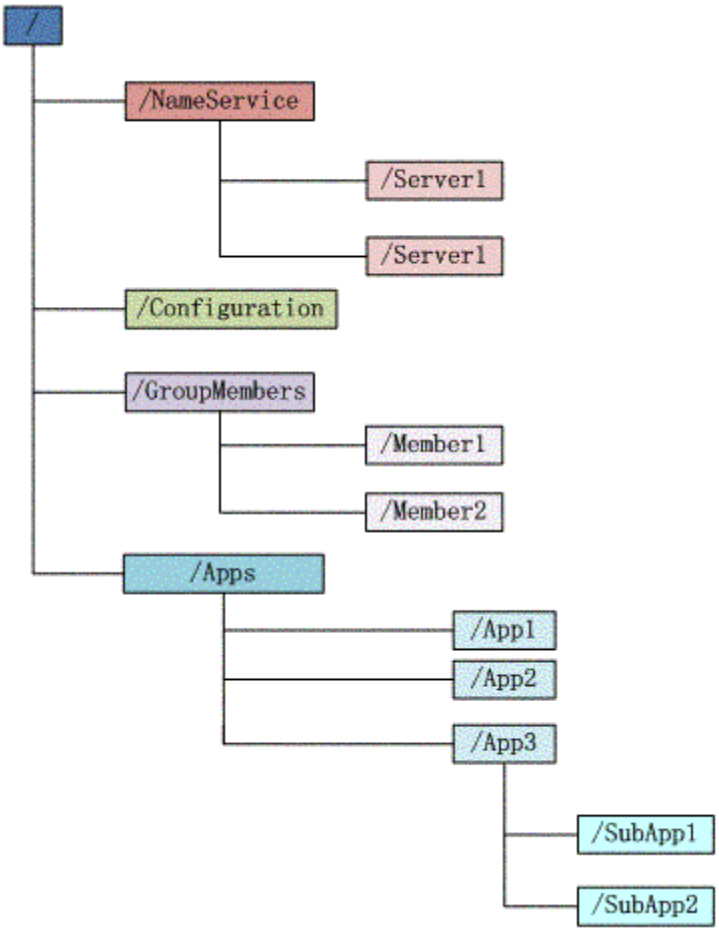

图 6-7 Zookeeper 的数据模型

每个子目录项如 NameService 都被称作为 znode，和文件系统一样，我们能够自由地增加、删除 znode，在一个 znode 下增加、删除子 znode，唯一的不同在于 znode 是可以存储数据的。Zookeeper 有四种类型的 znode：

- PERSISTENT-持久化目录节点：客户端与 Zookeeper 断开连接后，该节点依旧存在；
- PERSISTENT_SEQUENTIAL-持久化顺序编号目录节点：客户端与 Zookeeper 断开连接后，该节点依旧存在，只是 Zookeeper 会给该节点名称进行顺序编号；
- EPHEMERAL-临时目录节点：客户端与 Zookeeper 断开连接后，该节点被删除；
- EPHEMERAL_SEQUENTIAL-临时顺序编号目录节点：客户端与 Zookeeper 断开连接后，该节点被删除，只是 Zookeeper 会给该节点名称进行顺序编号。

Zookeeper 这种数据结构有如下这些特点：

- 每个子目录项如 NameService 都被称作为 znode，这个 znode 是被它所在的路径唯一标识的，如 Server1 这个 znode 的标识为/NameService/Server1；
- znode 可以有子节点目录，并且每个 znode 可以存储数据，注意 EPHEMERAL 类型的目录节点不能有子节点目录；
- znode 是有版本的，每个 znode 中存储的数据可以有多个版本，也就是一个访问路径中可以存储多份数据；
- znode 可以是临时节点，一旦创建这个 znode 的客户端与服务器失去联系，这个 znode 也将自动删除，Zookeeper 的客户端和服务器通信采用长连接方式，每个客户端和服务器通过心跳来保持连接，这个连接状态称为 session，如果 znode 是临时节点，这个 session 失效，znode 也就删除了；
- znode 的目录名可以自动编号，如 App1 已经存在，再创建的话，将会自动命名为 App2；
- znode 可以被监控，包括这个目录节点中存储的数据的修改，子节点目录的变化等，一旦发生变化就可以通知那些设置了监控该 znode 的客户端，这个是 Zookeeper 的核心特性，Zookeeper 的很多功能都是基于这个特性实现的，后面在典型的应用场景中会有实例介绍。

6.4 Zookeeper 的典型应用场景

6.4.1 统一命名服务

分布式应用中，通常需要有一套完整的命名规则，既能够产生唯一的名称又便于人识别和记住，通常情况下用树形的名称结构是一个理想的选择，树形的名称结构是一个有层次的目录结构，既对人友好又不会重复。说到这里你可能想到了 JNDI，没错，Zookeeper 的命名服务（Name Service）与 JNDI 能够完成的功能是差不多的，它们都是将有层次的目录结构关联到一定资源上，但是，Zookeeper 的 Name Service 更加是广泛意义上的关联，也许你并不需要将名称关联到特定资源上，你可能只需要一个不会重复名称，就像数据库中产生一个唯一的数字主键一样。

Name Service 已经是 Zookeeper 内置的功能，你只要调用 Zookeeper 的 API 就能实现。如调用 create 接口就可以很容易创建一个目录节点。

6.4.2 配置管理

配置的管理在分布式应用环境中很常见，例如，同一个应用系统需要多台 PC 服务器运行，但是，它们运行的应用系统的某些配置项是相同的，如果要修改这些相同的配置项，那么，就必须同时修改每台运行这个应用系统的 PC 服务器，这样非常麻烦，而且容易出错。

像这样的配置信息完全可以交给 Zookeeper 来管理，将配置信息保存在 Zookeeper 的某个目录节点中，然后，将所有需要修改的应用机器监控配置信息的状态，一旦配置信息发生变化，每台应用机器就会收到 Zookeeper 的通知，然后从 Zookeeper 获取新的配置信息应用到系统中。

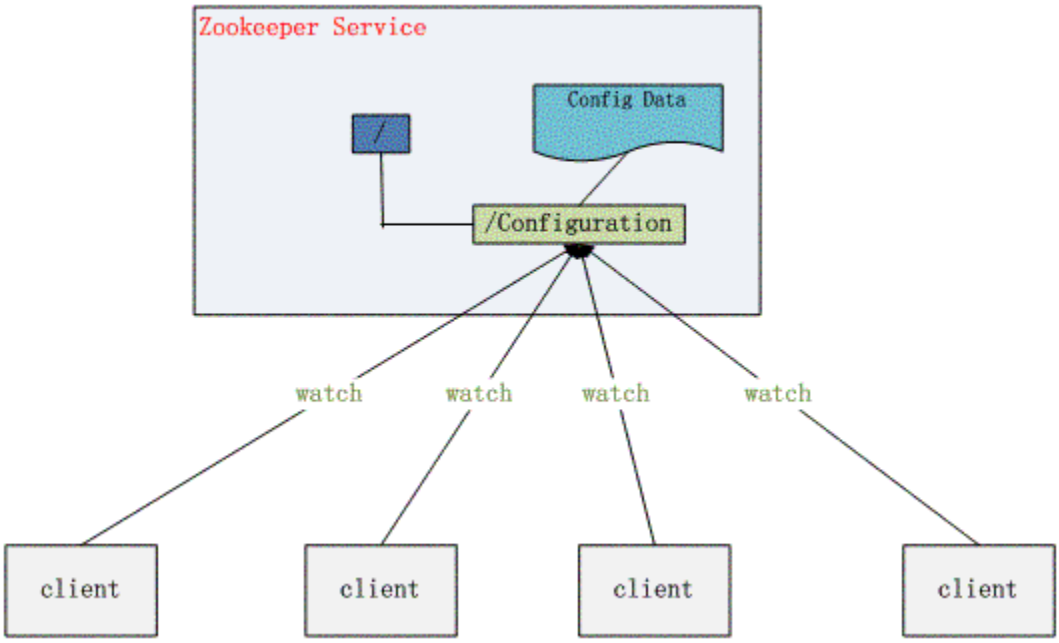

图 6-8 配置管理结构图

6.4.3 集群管理

Zookeeper 能够很容易地实现集群管理的功能，如果有多台 Server 组成一个服务集群，那么必须要一个“总管”知道当前集群中每台机器的服务状态，一旦有机器不能提供服务，集群中其它机器必须知道，从而做出调整重新分配服务策略。同样，当增加集群的服务能力时，就会增加一台或多台 Server，同样也必须让“总管”知道。

Zookeeper 不仅能够帮你维护当前的集群中机器的服务状态，而且能够帮你选出一个“总管”，让这个总管来管理集群，这就是 Zookeeper 的另一个功能——Leader Election。

它们的实现方式都是在 Zookeeper 上创建一个 EPHEMERAL 类型的目录节点，然后每个 Server 在它们创建目录节点的父目录节点上调用 getChildren(String path, boolean watch)方法并设置 watch 为 true，由于是 EPHEMERAL 目录节点，当创建它的 Server 死去，这个目录节点也随之被删除，所以 Children 将会变化，这时 getChildren 上的 Watch 将会被调用，所以其它 Server 就知道已经有某台 Server 死去了。新增 Server 也是同样的原理。

Zookeeper 如何实现 Leader Election，也就是选出一个 Master Server 呢？和前面的一样，每台 Server 创建一个 EPHEMERAL 目录节点，不同的是，它还是一个 SEQUENTIAL 目录节点，所以它是个 EPHEMERAL_SEQUENTIAL 目录节点。之所以它是 EPHEMERAL_SEQUENTIAL 目录节点，是因为我们可以给每台 Server 编号，我们可以选择当前是最小编号的 Server 为 Master，假如这个最小编号的 Server 死去，由于是 EPHEMERAL 节点，死去的 Server 对应的节点也被删除，所以当前的节点列表中又出现一个最小编号的节点，我们就选择这个节点为当前 Master。这样就实现了动态选择 Master，避免了传统意义上单 Master 容易出现单点故障的问题。

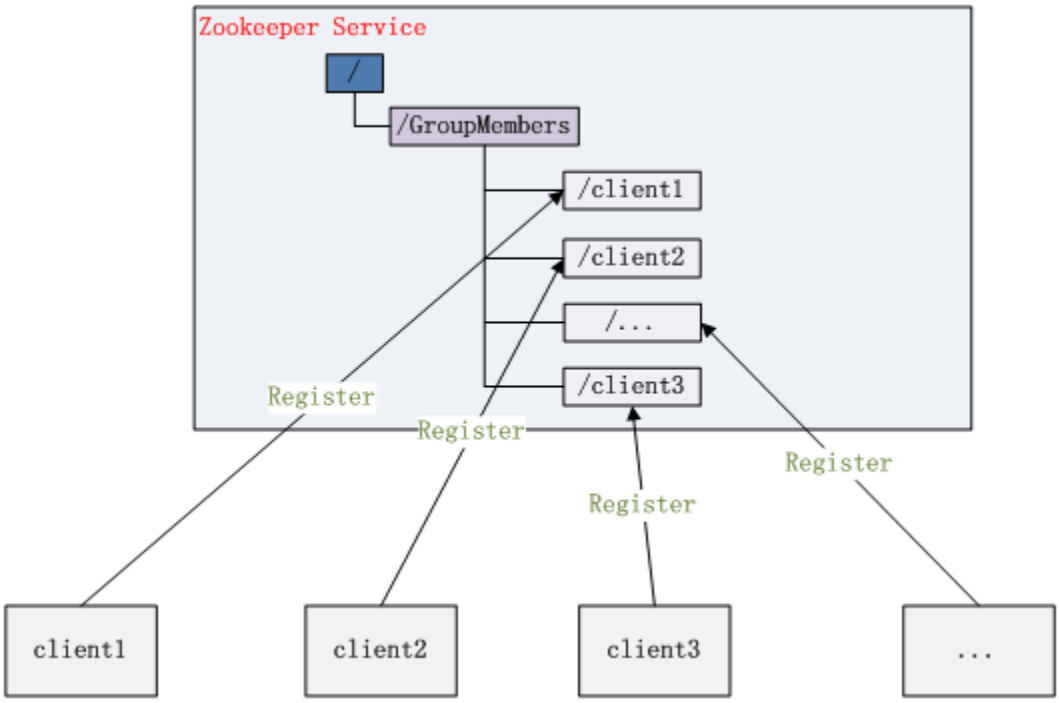

图 6-9 集群管理结构图

6.4.4 共享锁

共享锁在同一个进程中很容易实现，但是，在跨进程或者在不同 Server 之间就不好实现了。Zookeeper 却很容易实现这个功能，实现方式也是需要获得锁的 Server 创建一个 EPHEMERAL_SEQUENTIAL 目录节点，然后调用 getChildren 方法获取当前的目录节点列表中最小的目录节点是不是就是自己创建的目录节点，如果正是自己创建的，那么它就获得了这个锁，如果不是，那么它就调用 exists(String path, boolean watch)方法并监控 Zookeeper 上目录节点列表的变化，一直到自己创建的节点是列表中最小编号的目录节点，从而获得锁，释放锁很简单，只要删除前面它自己所创建的目录节点就行了。

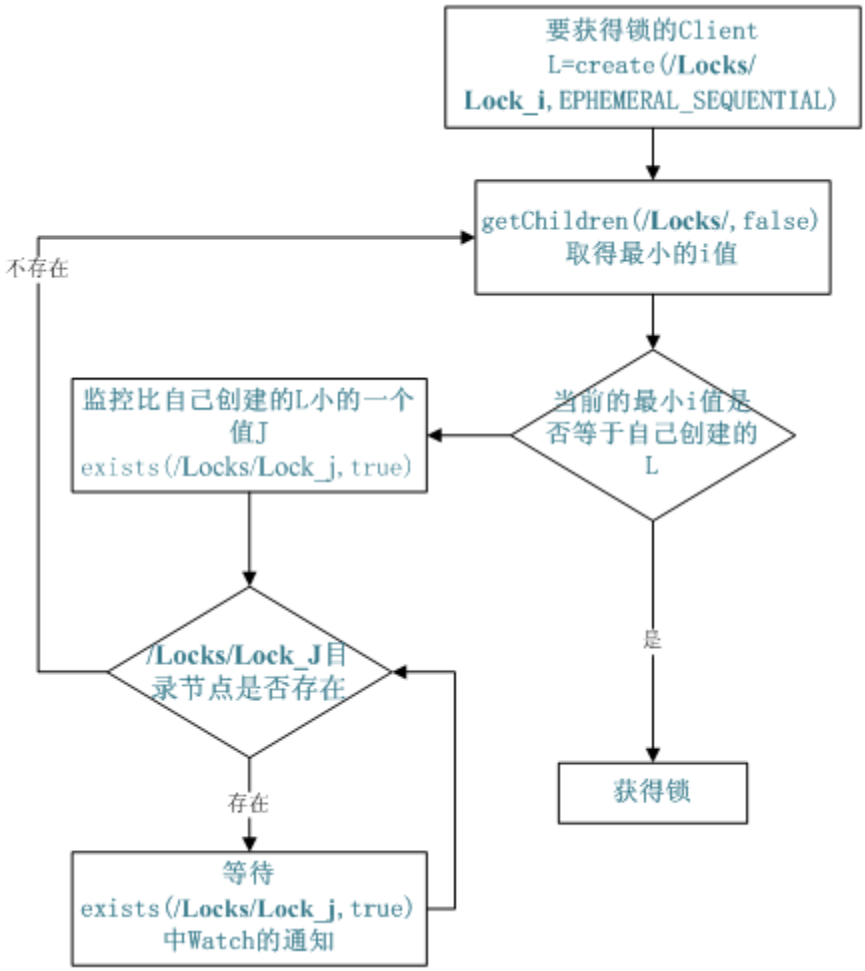

图 6-10 Zookeeper 实现 Locks 的流程图

6.4.5 队列管理

Zookeeper 可以处理两种类型的队列：

- 当一个队列的成员都聚齐时，这个队列才可用，否则一直等待所有成员到达，这种是同步队列。
- 队列按照 FIFO 方式进行入队和出队操作，例如实现生产者和消费者模型。

同步队列用 Zookeeper 实现的实现思路如下：

创建一个父目录/synchronizing，每个成员都监控标志（Set Watch）位目录/synchronizing/start 是否存在，然后每个成员都加入这个队列，加入队列的方式就是创建/synchronizing/member_i 的临时目录节点，然后每个成员获取/ synchronizing 目录的所有目录节点，也就是 member_i。判断 i 的值是否已经是成员的个数，如果小于成员个数，就等待/synchronizing/start 的出现，如果已经相等就创建/synchronizing/start。

用下面的流程图更容易理解：

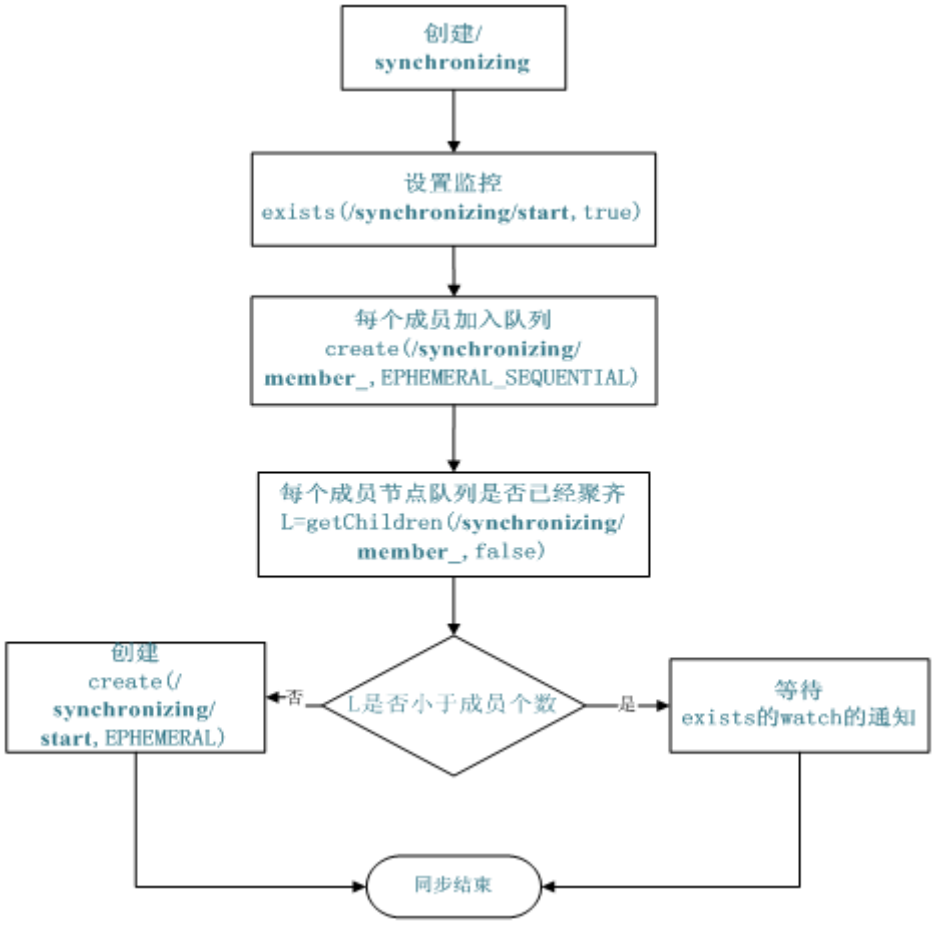

图 6-11 同步队列流程图

本章小结

Zookeeper 作为 Hadoop 项目中的一个子项目，是 Hadoop 集群管理的一个必不可少的模块，它主要用来控制集群中的数据，比如使用它来管理 Hadoop 集群中的 NameNode，还有 HBase 中 Master Election、Server 之间状态同步等。

本章介绍了 Zookeeper 的基本知识，并描述了几个典型的应用场景。这些都是 Zookeeper 的基本功能，最重要的是 Zookeeper 提供了一套很好的分布式集群管理的机制，就是它这种基于层次型的目录树的数据结构，并对树中的节点进行有效管理，从而可以设计出多种多样的分布式的数据管理模型，而不仅仅局限于上面提到的几个常用应用场景。

参考文献

[1] 分布式服务框架 Zookeeper -- 管理分布式环境中的数据. http://www.ibm.com/developerworks/cn/opensource/os-cn-zookeeper/

[2] Zookeeper 与 paxos 算法. http://blog.csdn.net/ronghao100/article/details/7384752

附录 1:任课教师介绍

林子雨(1978－),男,博士,厦门大学计算机科学系助理教授,主要研究领域为数据库,数据仓库,数据挖掘.

主讲课程：《大数据技术基础》

办公地点：厦门大学海韵园科研 2 号楼

E-mail: ziyulin@xmu.edu.cn

个人网页：http://www.cs.xmu.edu.cn/linziyu