厦门大学计算机科学系研究生课程

《大数据技术基础》

第6章 Zookeeper

（2013年新版）

林子雨

厦门大学计算机科学系

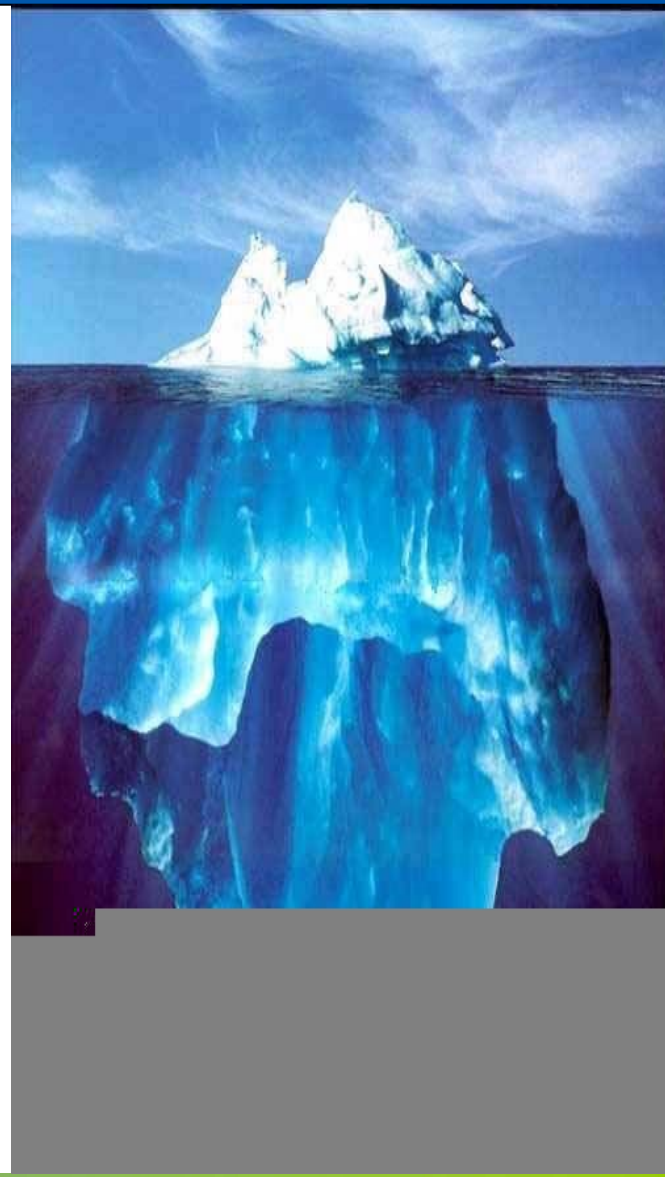
E-mail: ziyulin@xmu.edu.cn

主页：http://www.cs.xmu.edu.cn/linziyu

提纲

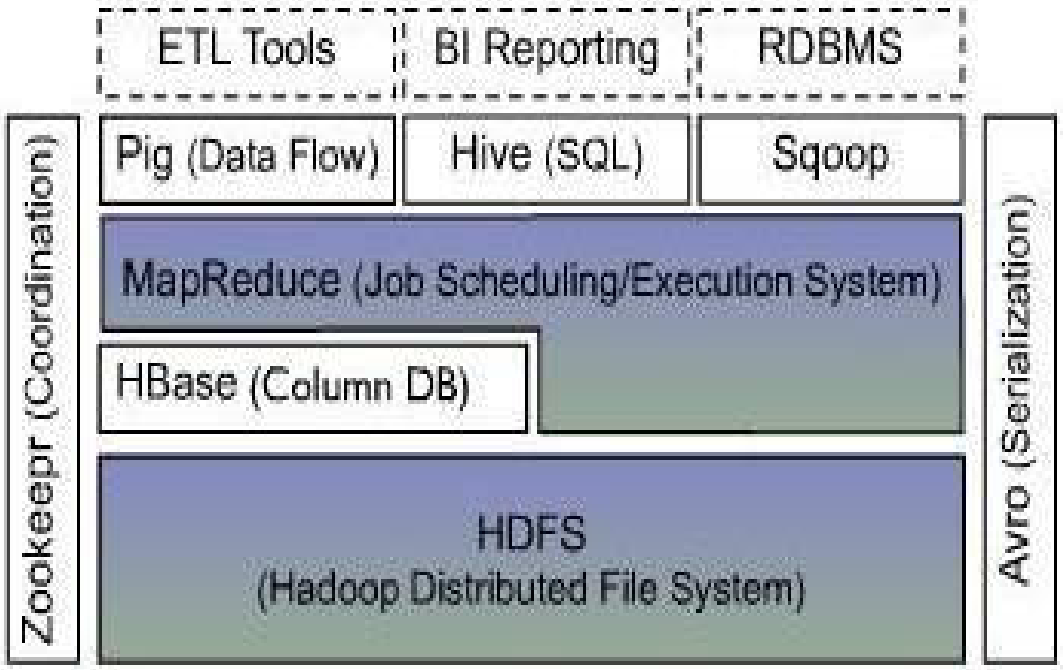
- **Zookeeper简介**
- **Zookeeper的工作原理**
- **Zookeeper的数据模型**
- **Zookeeper的典型应用场景**
- 小结

本讲义PPT存在配套教材，由林子雨通过大量阅读、收集、整理各种资料后编写而成
下载配套教材请访问《大数据技术基础》2013班级网站：http://dblab.xmu.edu.cn/node/423



1、Zookeeper简介

- 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。Zookper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。



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。

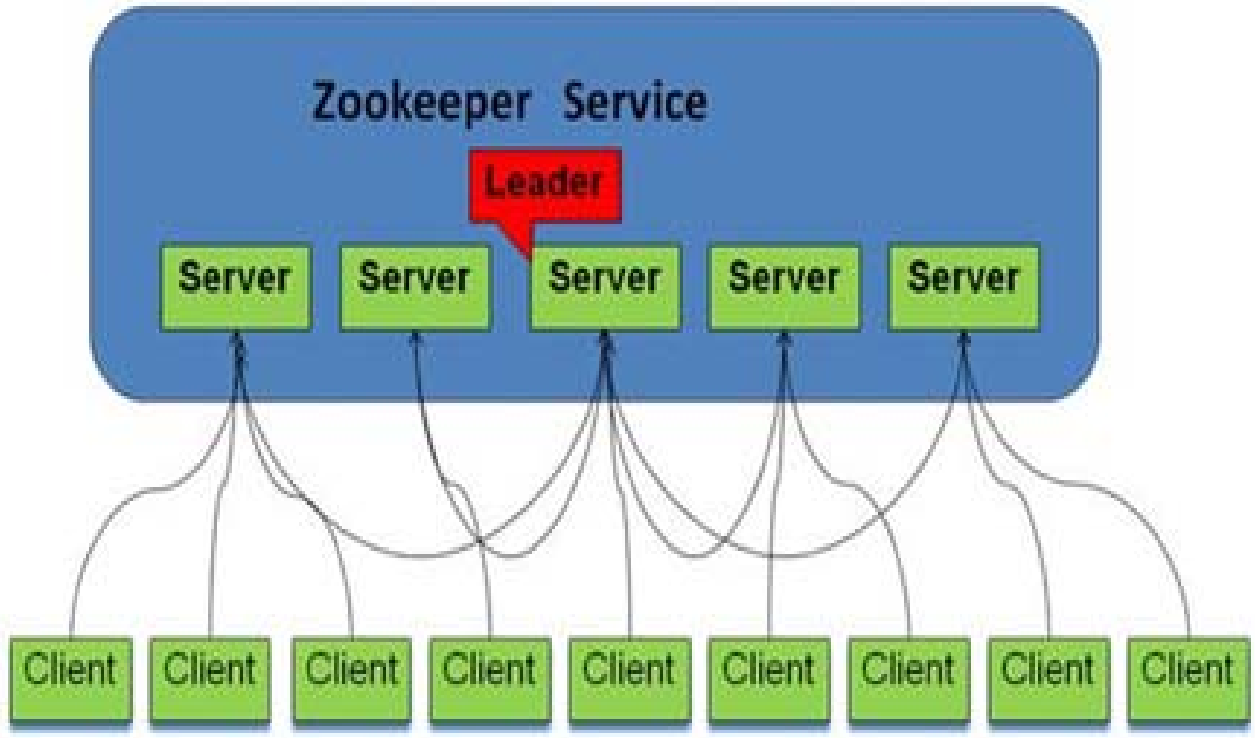


图1 Zookeeper的系统架构图

Zookeeper中的角色

Zookeeper中的角色主要包括领导者、学习者和客户端，如下表所示。

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

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**：client不论连接到哪个Server，展示给它都是同一个视图，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client不得干预快速的client的请求，使得每个client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Server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。

1.3特点

Zookeeper的特点主要包括以下几个方面：

- 简易性：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，实现了高性能、高可靠性和有序的访问。
- 可用性：组成Zookeeper的各个服务器之间能够相互通信，在内存中保存服务器状态，也保存了操作日志，并且持久化快照，只要大多数服务器是可用的，那么，Zookeeper就是可用的。
- 有序性：使用数字来对每个更新进行标记，保证Zookeeper交互的有序。
- 高效性：表现在以读为主的系统上。

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Server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，它们分别是恢复模式（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。实现中zxid是一个64位的数字，它高32位是epoch用来标识leader关系是否改变，每次一个leader被选出来，它都会有一个新的epoch，标识当前属于那个leader的统治时期。低32位用于递增计数。

Server工作过程中的三种状态

- 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。

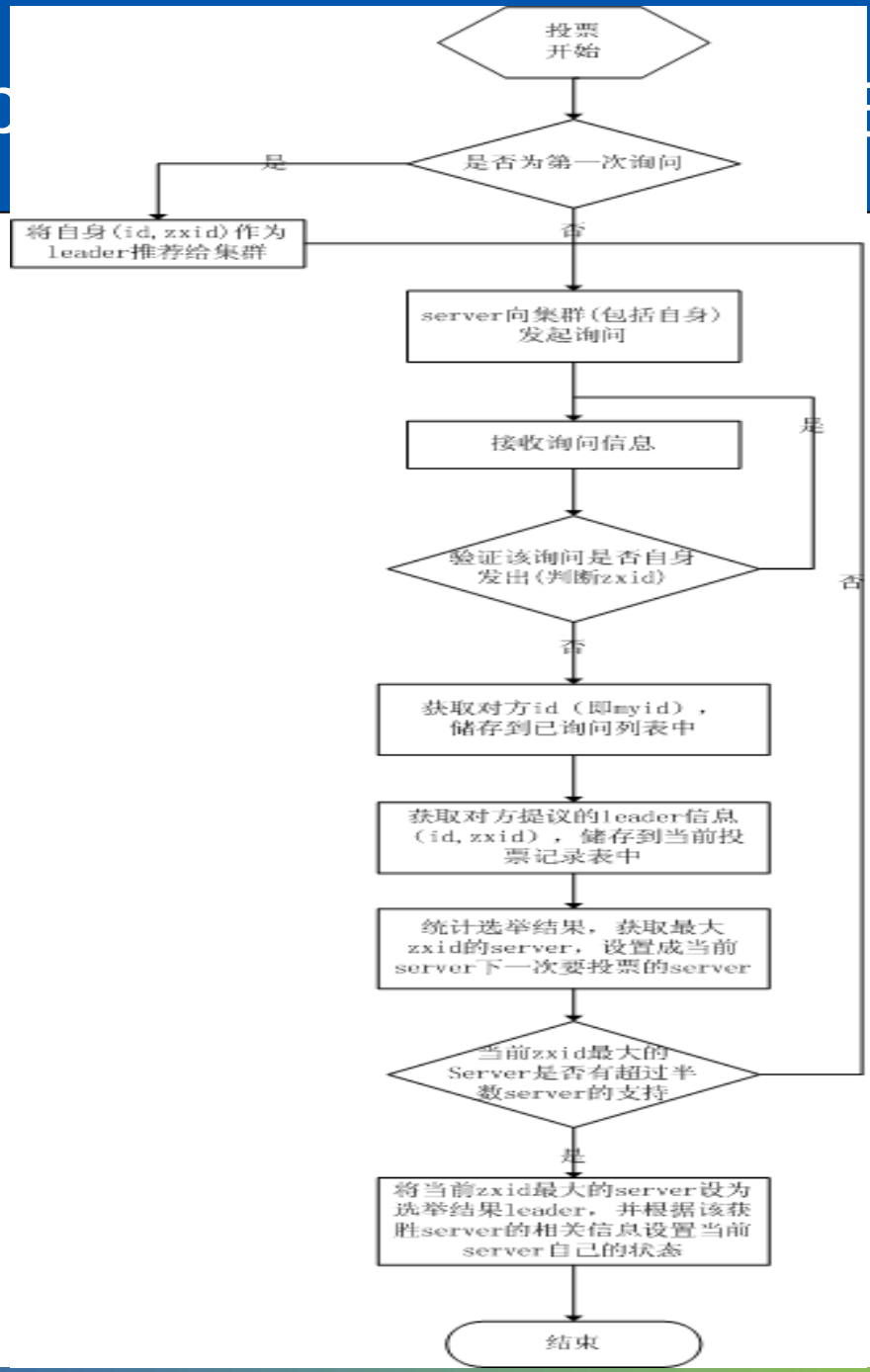
2.1 选主流程

- 当leader崩溃或者leader失去大多数follower，这时候zk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。Zk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。

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；
- 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，当前server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。

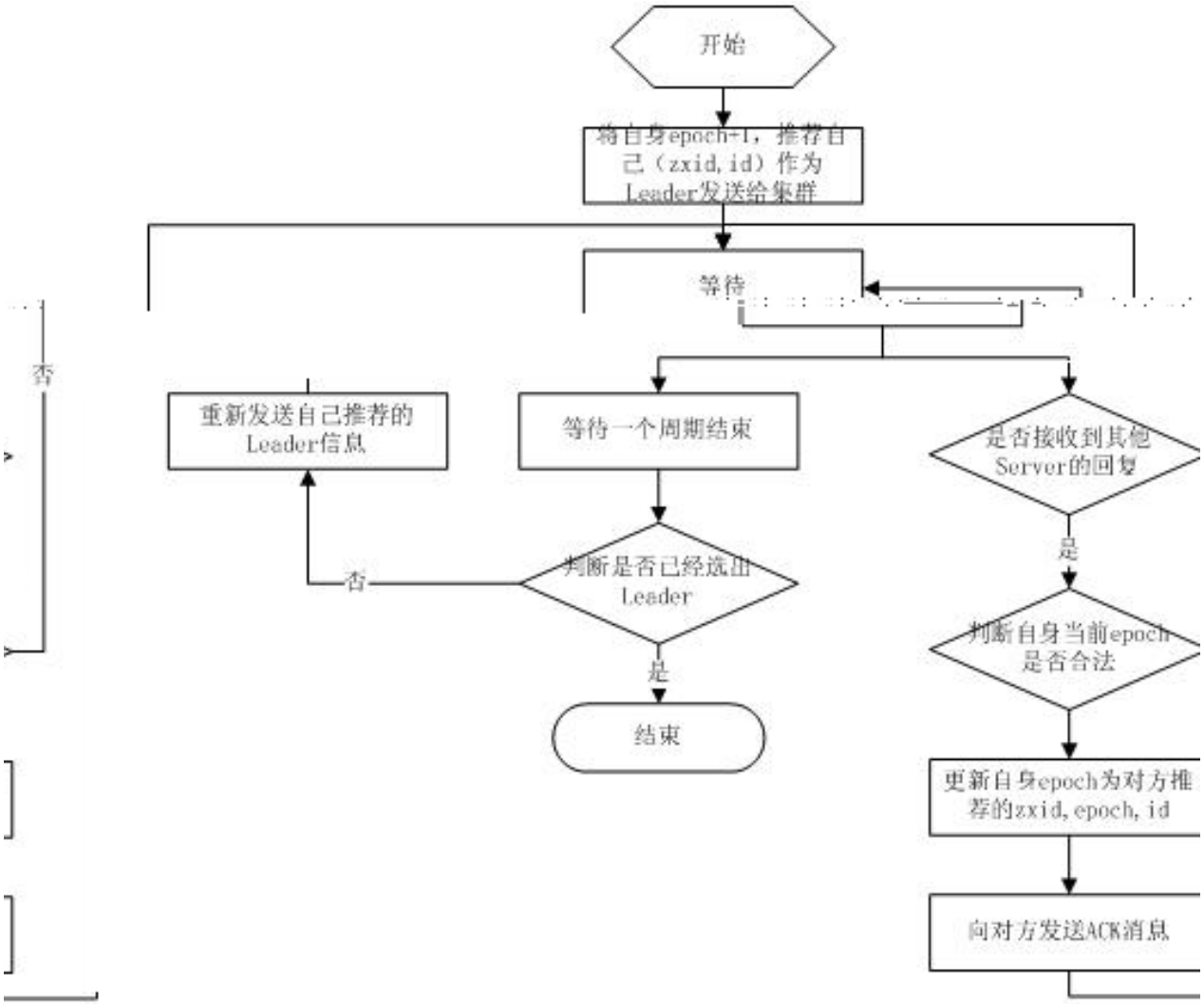
采用b 主流程


投票开始
是否为第一次询问
是
否
将自身(id, zxid)作为leader推荐给集群
server向集群(包括自身)发起询问
接收询问信息
是
验证该询问是否自身发出(判断zxid)
否
否
获取对方id（即myid），储存到已询问列表中
获取对方提议的leader信息(id, zxid)，储存到当前投票记录表中
统计选举结果，获取最大zxid的server，设置成当前server下一次要投票的server
当前zxid最大的Server是否有超过半数server的支持
是
将当前zxid最大的server设为选举结果leader，并根据该获胜server的相关信息设置当前server自己的状态
结束

fast paxos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。

采用fast paxos算法实现选主流程



2.2 同步流程

选完leader以后，zk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。

2.3 工作流程

- **Leader**工作流程
- **Follower**工作流程
- **Observer**工作流程

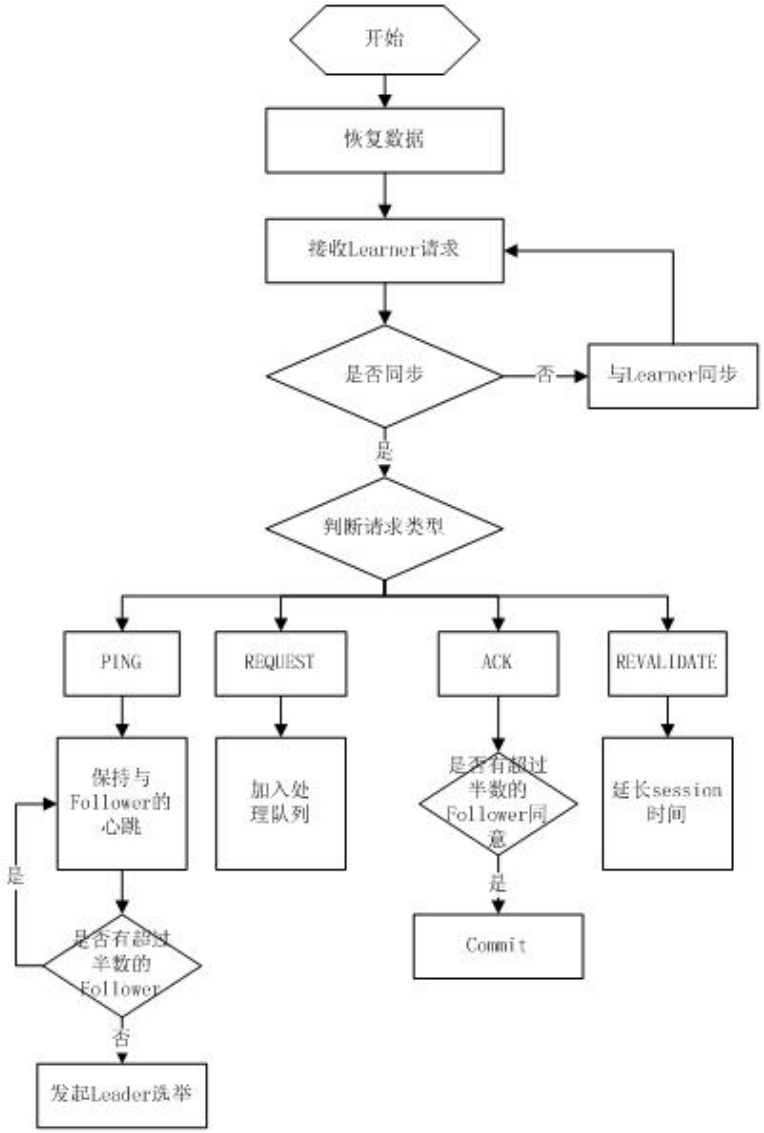
Leader的三个主要功能

- 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

- 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



Follower的四个主要功能：

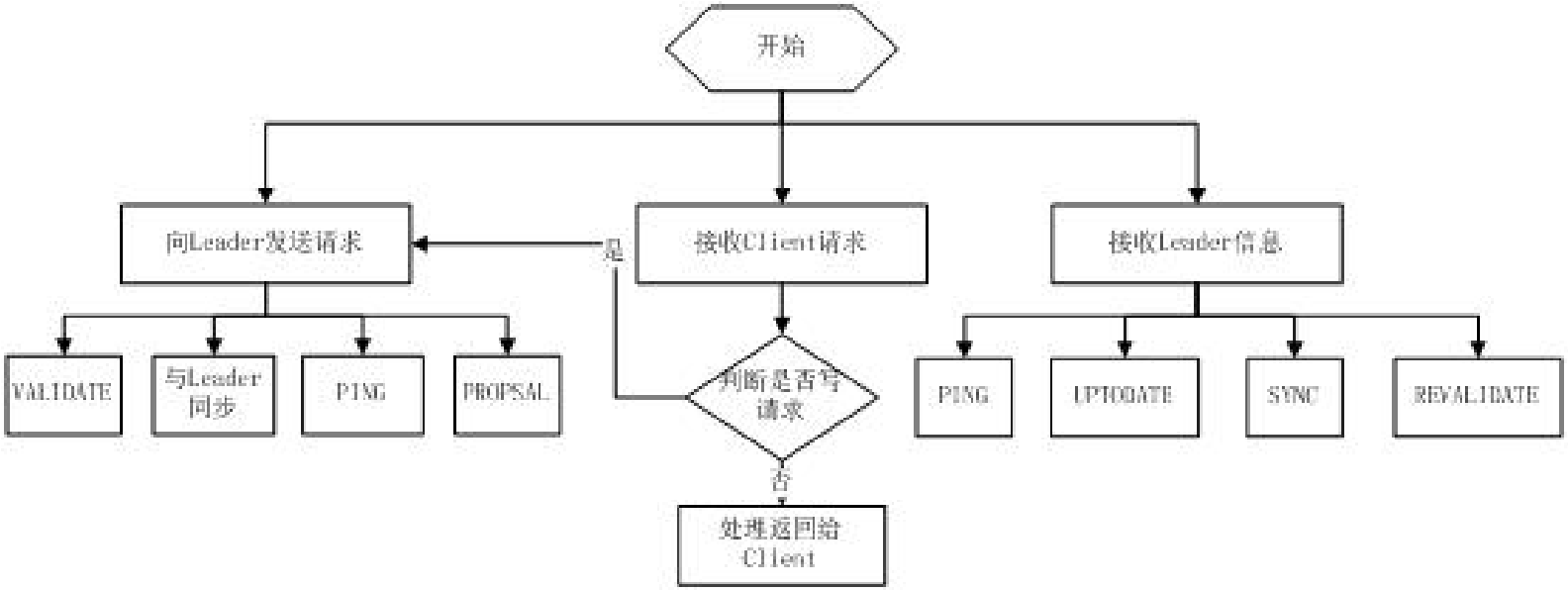
- 向Leader发送请求（PING消息、REQUEST消息、ACK消息、REVALIDATE消息）；
- 接收Leader消息并进行处理；
- 接收Client的请求，如果为写请求，发送给Leader进行投票；
- 返回Client结果。

几种Leader消息

Follower的消息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，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。

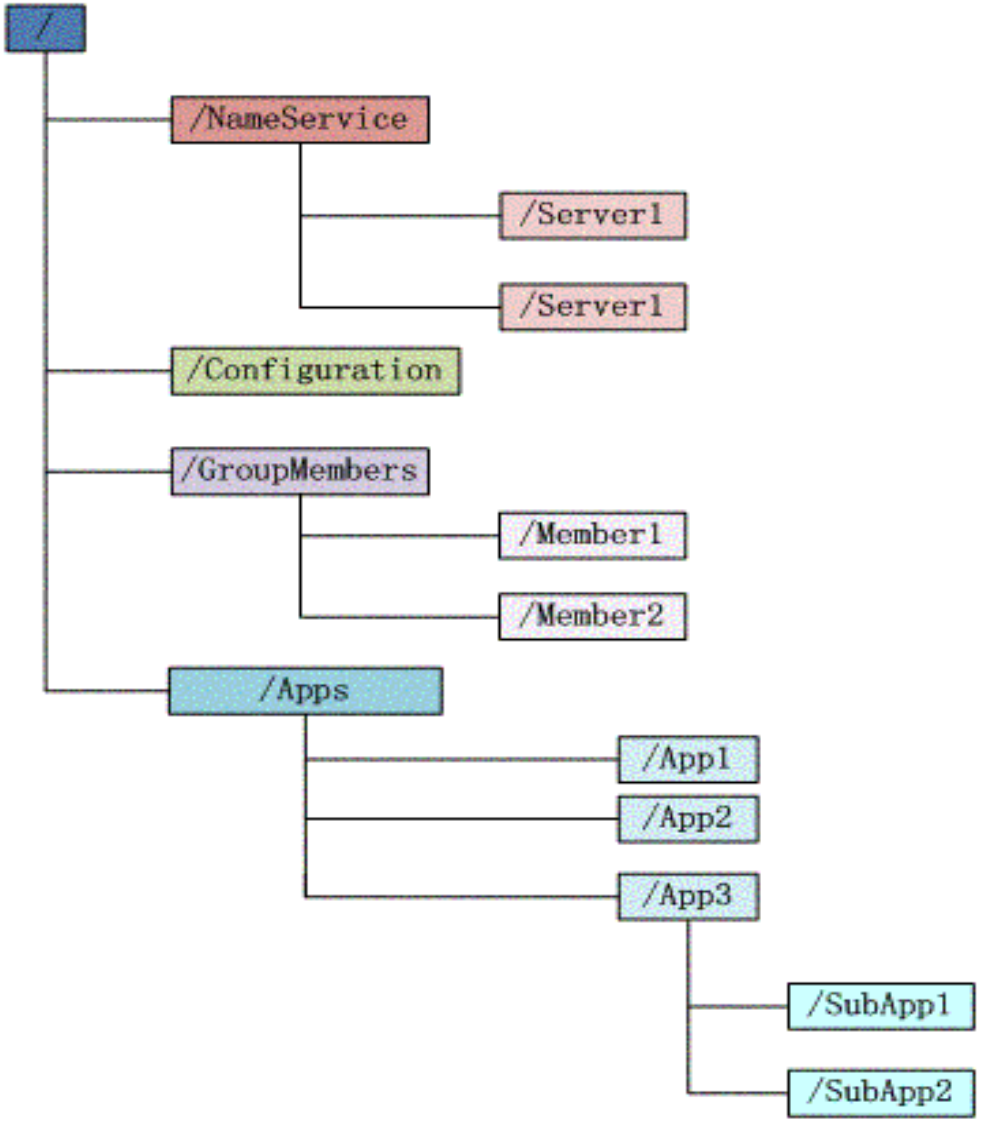
Follower的工作流程简图



observer的流程

- 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。

3、Zookeeper数据模型



- Zookeeper维护一个类似文件系统的数据结构，如图所示。
- 每个子目录项如 NameService 都被称作为 znode，和文件系统一样，我们能够自由的增加、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。

Znode的四种类型

- 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给该节点名称进行顺序编号。

Zookeeper 数据结构的特点

- 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，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，一旦变化可以通知设置监控的客户端，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。

4、Zookeeper的典型应用场景

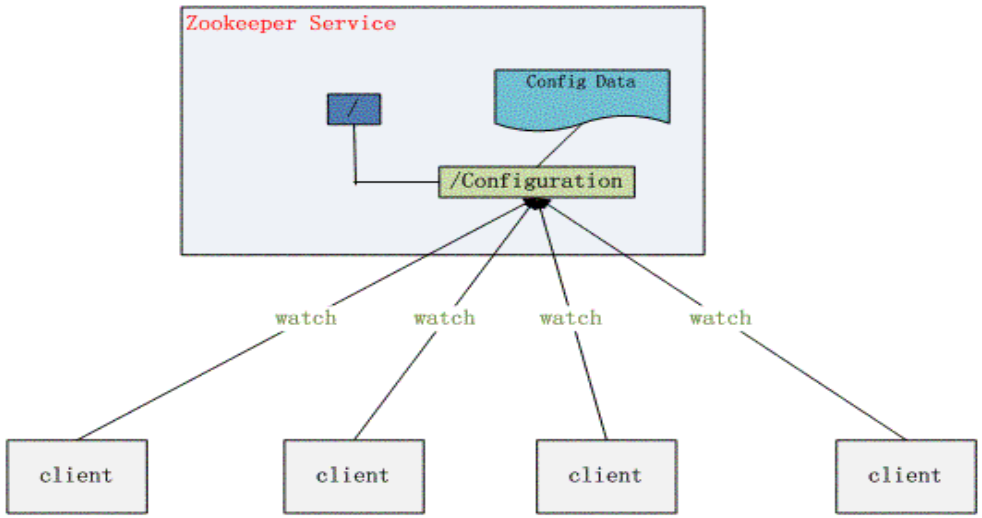
- 统一命名服务（**Name Service**）
- 配置管理（**Configuration Management**）
- 集群管理（**Group Membership**）
- 共享锁（**Locks**）
- 队列管理

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。
- Zookeeper的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。
- 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。

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 Server 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 Server，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。



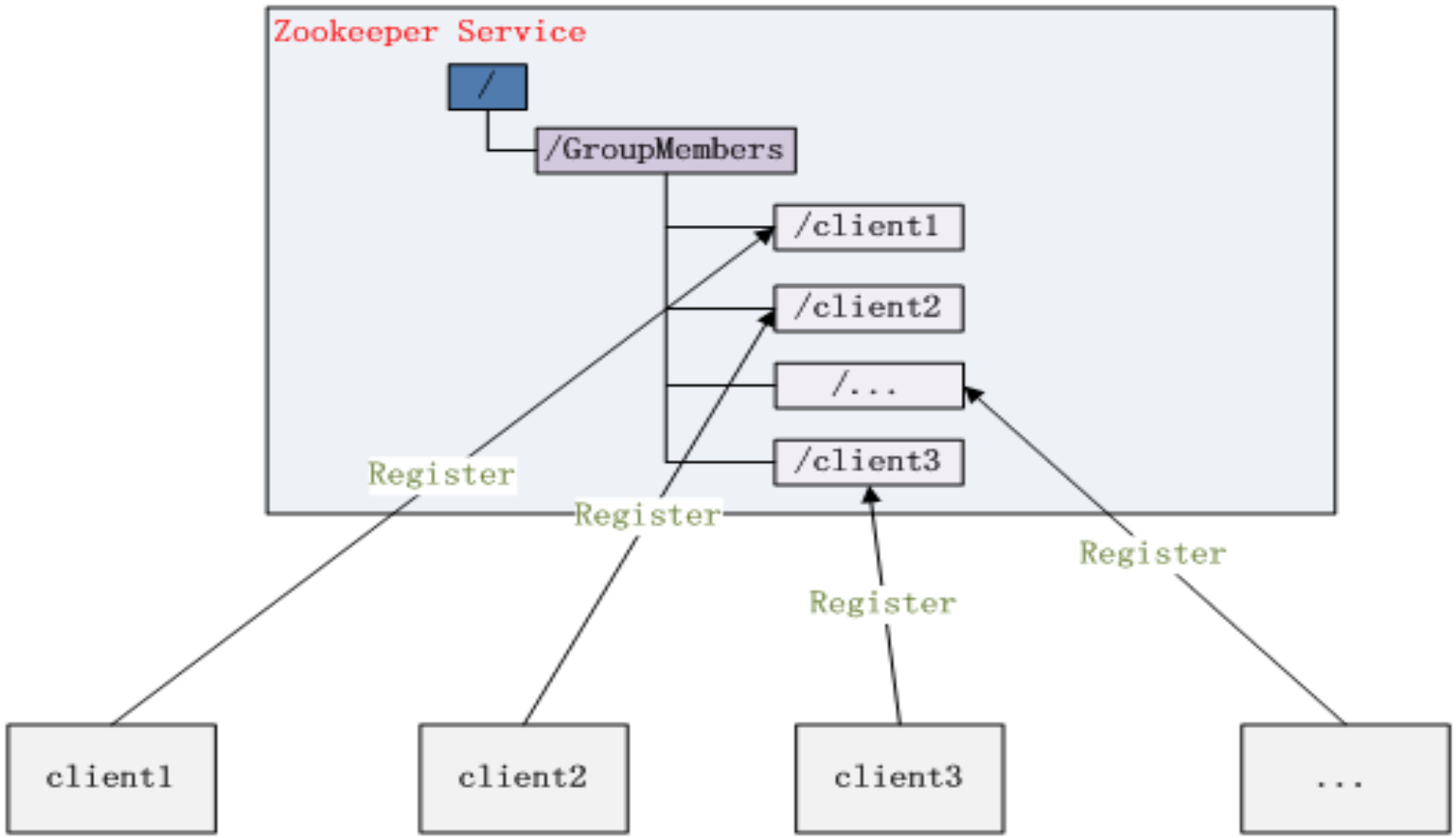
集群管理

- Zookeeper 能够很容易地实现集群管理的功能
- 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。

集群管理

- 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。

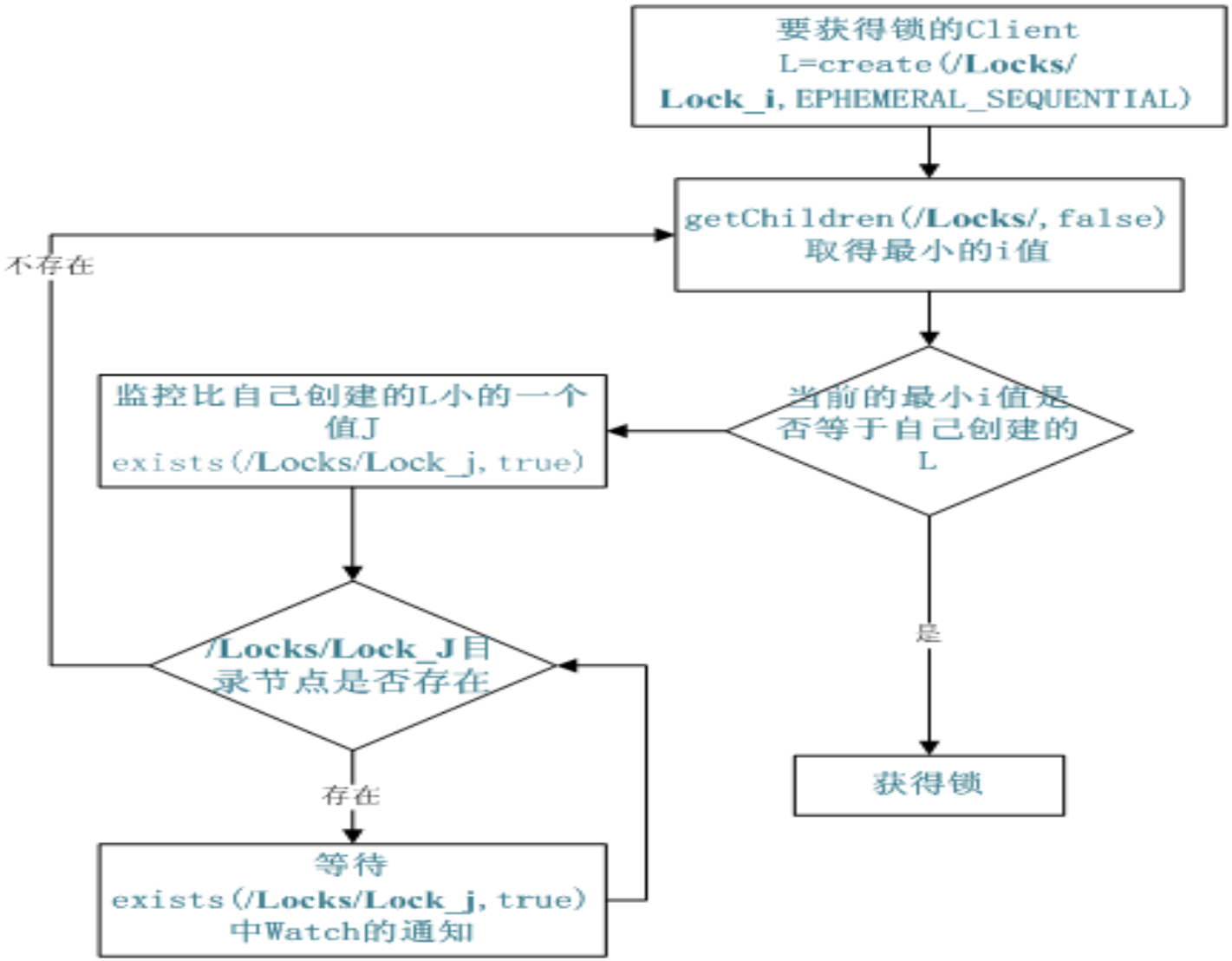
集群管理结构图


Zookeeper Service
/
/GroupMembers
/client1
/client2
/...
/client3
Register
Register
Register
Register
client1
client2
client3
...

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。

Zookeeper 实现 Locks 的流程图



队列管理

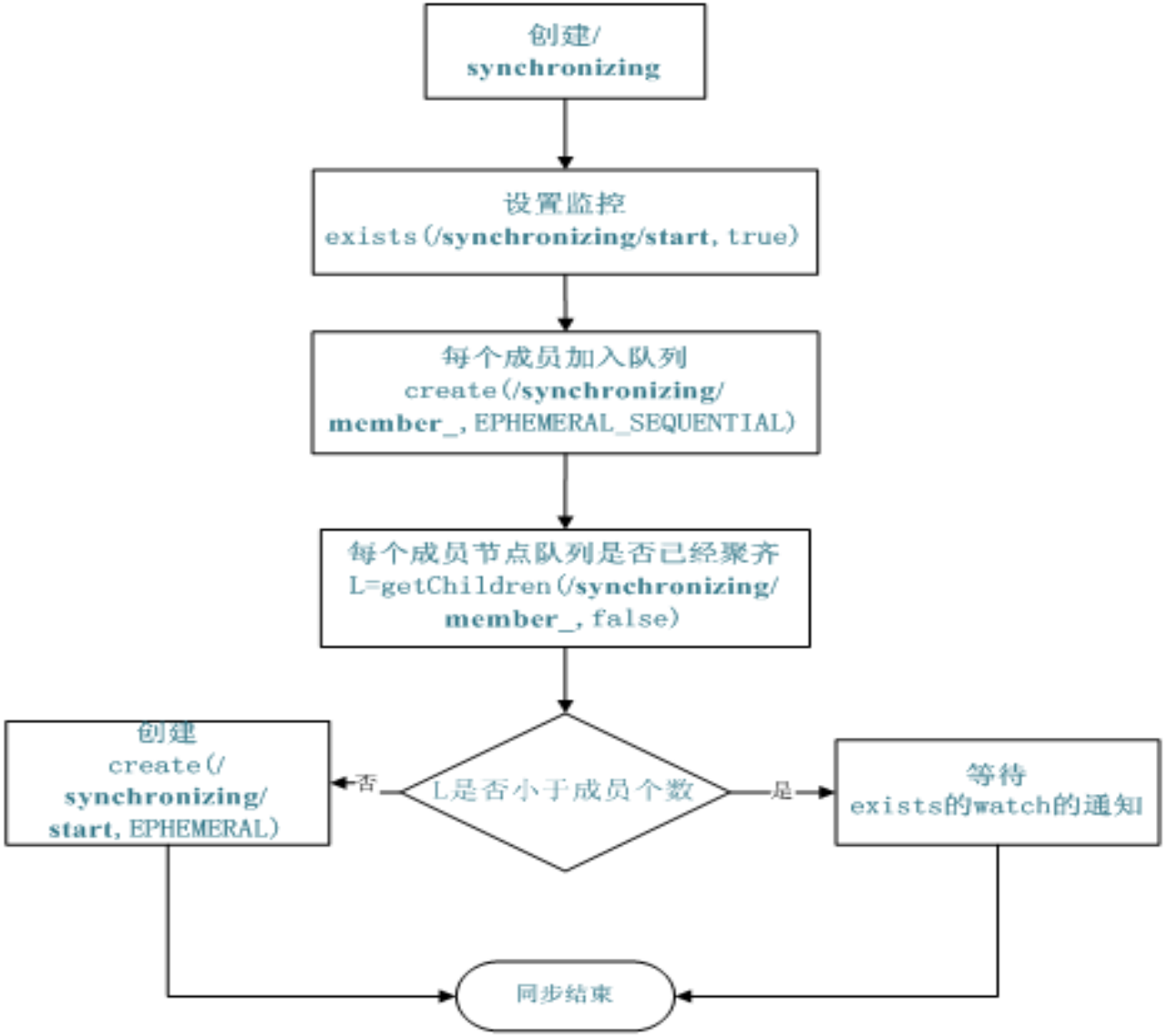
Zookeeper可以处理两种类型的队列：

- 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，如果小于成员个数等待 /synchronizing/start 的出现，如果已经相等就创建 /synchronizing/start。

同步队列流程图


创建/
synchronizing
设置监控
exists(/synchronizing/start, true)
每个成员加入队列
create(/synchronizing/
member_, EPHEMERAL_SEQUENTIAL)
每个成员节点队列是否已经聚齐
L=getChildren(/synchronizing/
member_, false)
L是否小于成员个数
否
是
创建
create(/
synchronizing/
start, EPHEMERAL)
等待
exists的watch的通知
同步结束

5、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Zoopkeeper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。

主讲教师和助教

主讲教师：林子雨

单位：厦门大学计算机科学系
E-mail: ziyulin@xmu.edu.cn
个人网页：http://www.cs.xmu.edu.cn/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn

助教：赖明星

单位：厦门大学计算机科学系数据库实验室2011级硕士研究生（导师：林子雨）
E-mail: mingxinglai@gmail.com
个人主页：http://mingxinglai.com

欢迎访问《大数据技术基础》2013班级网站：http://dblab.xmu.edu.cn/node/423
本讲义PPT存在配套教材《大数据技术基础》，请到上面网站下载。

Department of Computer Science, Xiamen University, Sep, 2013