AskHarries
2018年05月22日 阅读 2006

咱们一起聊聊Zookeeper

Google的三篇论文影响了很多很多人，也影响了很多很多系统。这三篇论文一直是分布式领域传阅的经典。根据MapReduce，于是我们有了Hadoop；根据GFS，于是我们有了HDFS；根据BigTable，于是我们有了HBase。而在这三篇论文里都提及Google的一个Lock Service ——Chubby，哦，于是我们有了Zookeeper。

随着大数据的火热，Hxx们已经变得耳熟能详，现在作为一个开发人员如果都不知道这几个名词出门都好像不好意思跟人打招呼。但实际上对我们这些非大数据开发人员而言，Zookeeper是比Hxx们可能接触到更多的一个基础服务。但是，无奈的是它一直默默的位于二线，从来没有Hxx们那么耀眼。那么到底什么是Zookeeper呢？Zookeeper可以用来干什么？我们将如何使用Zookeeper？Zookeeper又是怎么实现的？

什么是Zookeeper

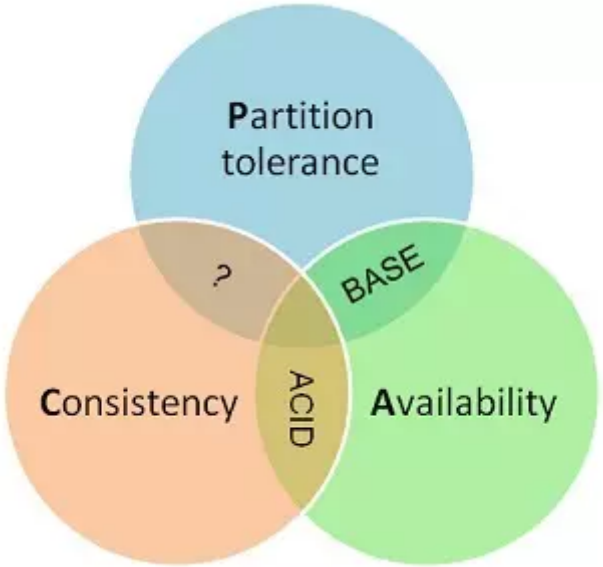
在Zookeeper的官网上有这么一句话：ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services。

这大概描述了Zookeeper主要是一个分布式服务协调框架，实现同步服务，配置维护和命名服务等分布式应用。是一个高性能的分布式数据一致性解决方案。

通俗地讲，ZooKeeper是动物园管理员，它是拿来管大象 Hadoop、鲸鱼 HBase、Kafka等的管理员。

Zookeeper和CAP的关系

作为一个分布式系统，分区容错性是一个必须要考虑的关键点。一个分布式系统一旦丧失了分区容错性，也就表示放弃了扩展性。因为在分布式系统中，网络故障是经常出现的，一旦出现在这种问题就会导致整个系统不可用是绝对不能容忍的。所以，大部分分布式系统都会在保证分区容错性的前提下在一致性和可用性之间做权衡。



CAP关系

ZooKeeper是个CP（一致性+分区容错性）的，即任何时刻对ZooKeeper的访问请求能得到一致的数据结果，同时系统对网络分割具备容错性；但是它不能保证每次服务请求的可用性。也就是在极端环境下，ZooKeeper可能会丢弃一些请求，消费者程序需要重新请求才能获得结果。

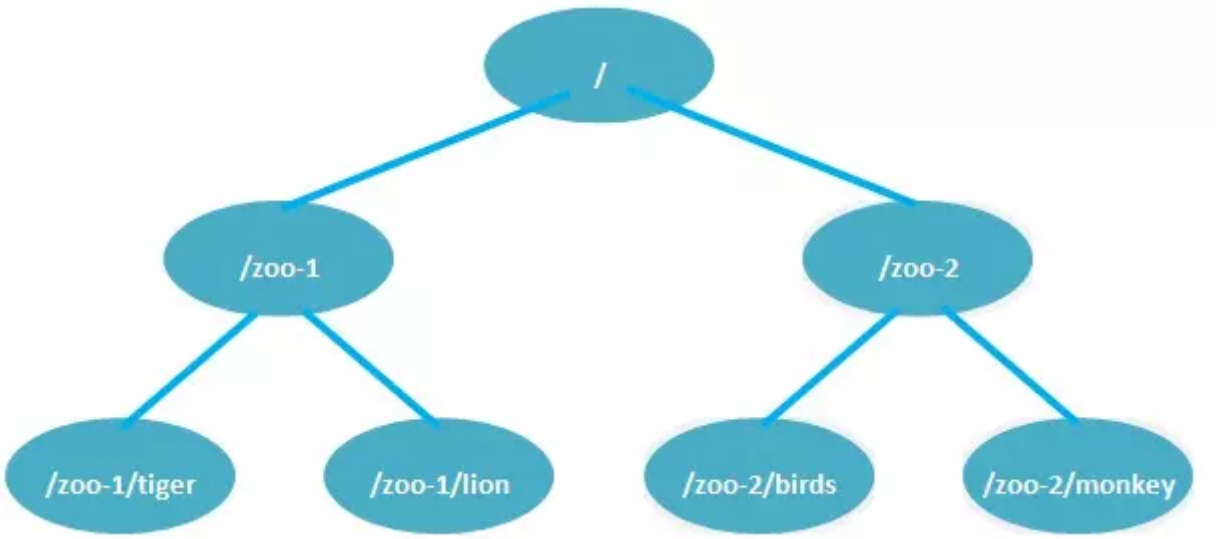
ZooKeeper是分布式协调服务，它的职责是保证数据在其管辖下的所有服务之间保持同步、一致；所以就不难理解为什么ZooKeeper被设计成CP而不是AP特性的了。而且， 作为ZooKeeper的核心实现算法Zab，就是解决了分布式系统下数据如何在多个服务之间保持同步问题的。

Zookeeper节点特性及节点属性分析

Zookeeper提供基于类似于文件系统的目录节点树方式的数据存储，但是Zookeeper并不是用来专门存储数据的，它的作用主要是用来维护和监控你存储的数据的状态变化。通过监控这些数据状态的变化，从而可以达到基于数据的集群管理。

数据模型

与Linux文件系统不同的是，Linux文件系统有目录和文件的区别，而Zookeeper的数据节点称为ZNode，ZNode是Zookeeper中数据的最小单元，每个ZNode都可以保存数据，同时还可以挂载子节点，因此构成了一个层次化的命名空间，称为树。



znode树形结构图

Zookeeper中ZNode的节点创建时候是可以指定类型的，主要有下面几种类型。

PERSISTENT：持久化ZNode节点，一旦创建这个ZNode点存储的数据不会主动消失，除非是客户端主动的delete。

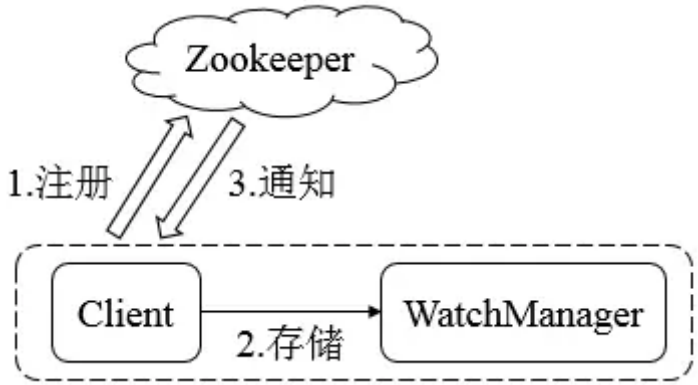
EPHEMERAL：临时ZNode节点，Client连接到Zookeeper Service的时候会建立一个Session，之后用这个Zookeeper连接实例创建该类型的znode，一旦Client关闭了Zookeeper的连接，服务器就会清除Session，然后这个Session建立的ZNode节点都会从命名空间消失。总结就是，这个类型的znode的生命周期是和Client建立的连接一样的。

PERSISTENT_SEQUENTIAL：顺序自动编号的ZNode节点，这种znoe节点会根据当前已近存在的ZNode节点编号自动加 1，而且不会随Session断开而消失。

EPEMERAL_SEQUENTIAL：临时自动编号节点，ZNode节点编号会自动增加，但是会随Session消失而消失

Watcher数据变更通知

Zookeeper使用Watcher机制实现分布式数据的发布/订阅功能。



Watcher机制

Zookeeper的Watcher机制主要包括客户端线程、客户端WatcherManager、Zookeeper服务器三部分。客户端在向Zookeeper服务器注册的同时，会将Watcher对象存储在客户端的WatcherManager当中。当Zookeeper服务器触发Watcher事件后，会向客户端发送通知，客户端线程从WatcherManager中取出对应的Watcher对象来执行回调逻辑。

ACL保障数据的安全

Zookeeper内部存储了分布式系统运行时状态的元数据，这些元数据会直接影响基于Zookeeper进行构造的分布式系统的运行状态，如何保障系统中数据的安全，从而避免因误操作而带来的数据随意变更而导致的数据库异常十分重要，Zookeeper提供了一套完善的ACL权限控制机制来保障数据的安全。

我们可以从三个方面来理解ACL机制：权限模式 Scheme、授权对象 ID、权限 Permission，通常使用"scheme:id:permission"来标识一个有效的ACL信息。

内存数据

Zookeeper的数据模型是树结构，在内存数据库中，存储了整棵树的内容，包括所有的节点路径、节点数据、ACL信息，Zookeeper会定时将这个数据存储到磁盘上。

DataTree：DataTree是内存数据存储的核心，是一个树结构，代表了内存中一份完整的数据。DataTree不包含任何与网络、客户端连接及请求处理相关的业务逻辑，是一个独立的组件。

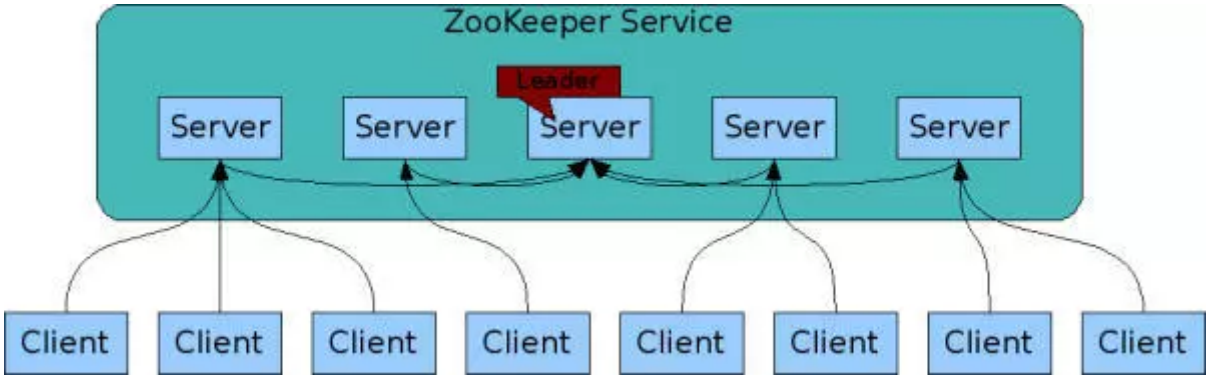
DataNode：DataNode是数据存储的最小单元，其内部除了保存了结点的数据内容、ACL列表、节点状态之外，还记录了父节点的引用和子节点列表两个属性，其也提供了对子节点列表进行操作的接口。

ZKDatabase：Zookeeper的内存数据库，管理Zookeeper的所有会话、DataTree存储和事务日志。ZKDatabase会定时向磁盘dump快照数据，同时在Zookeeper启动时，会通过磁盘的事务日志和快照文件恢复成一个完整的内存数据库。

Zookeeper的实现原理分析

1. Zookeeper Service网络结构

Zookeeper的工作集群可以简单分成两类，一个是Leader，唯一一个，其余的都是follower，如何确定Leader是通过内部选举确定的。



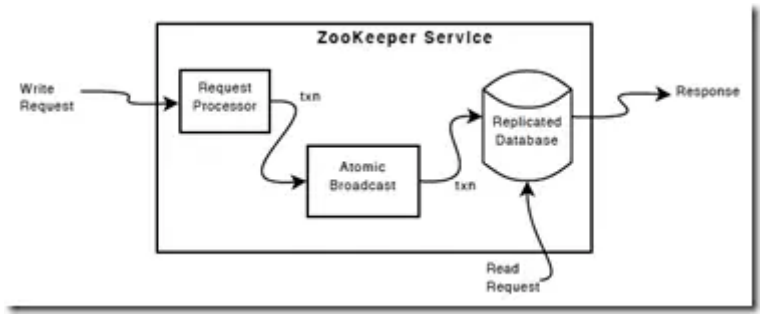
Zookeeper架构图

Leader和各个follower是互相通信的，对于Zookeeper系统的数据都是保存在内存里面的，同样也会备份一份在磁盘上。

如果Leader挂了，Zookeeper集群会重新选举，在毫秒级别就会重新选举出一个Leader。

集群中除非有一半以上的Zookeeper节点挂了，Zookeeper Service才不可用。

2. Zookeeper读写数据



zk读取数据流程

写数据，一个客户端进行写数据请求时，如果是follower接收到写请求，就会把请求转发给Leader，Leader通过内部的Zab协议进行原子广播，直到所有Zookeeper节点都成功写了数据后（内存同步以及磁盘更新），这次写请求算是完成，然后Zookeeper Service就会给Client发回响应。

读数据，因为集群中所有的Zookeeper节点都呈现一个同样的命名空间视图（就是结构数据），上面的写请求已经保证了写一次数据必须保证集群所有的Zookeeper节点都是同步命名空间的，所以读的时候可以在任意一台Zookeeper节点上。

3. Zookeeper工作原理

Zab协议

Zookeeper的核心是广播，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（ZooKeeper Atomic Broadcast）原子消息广播协议作为数据一致性的核心算法，Zab协议是专为Zookeeper设计的支持崩溃恢复原子消息广播算法。

Zab协议核心如下：

所有的事务请求必须一个全局唯一的服务器（Leader）来协调处理，集群其余的服务器称为follower服务器。Leader服务器负责将一个客户端请求转化为事务提议（Proposal），并将该proposal分发给集群所有的follower服务器。之后Leader服务器需要等待所有的follower服务器的反馈，一旦超过了半数的follower服务器进行了正确反馈后，那么Leader服务器就会再次向所有的follower服务器分发commit消息，要求其将前一个proposal进行提交。

Zab模式

Zab协议包括两种基本的模式：崩溃恢复和消息广播。

当整个服务框架启动过程中或Leader服务器出现网络中断、崩溃退出与重启等异常情况时，Zab协议就会进入恢复模式并选举产生新的Leader服务器。

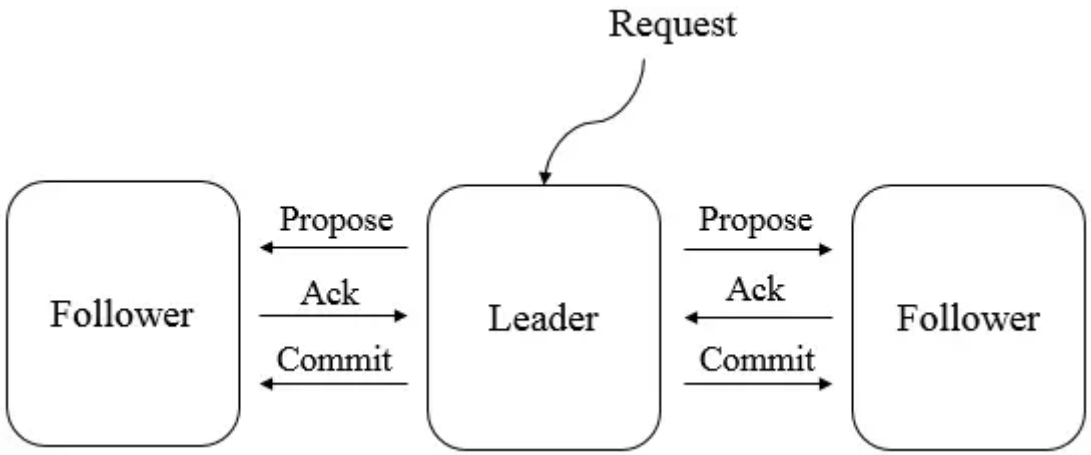
当选举产生了新的Leader服务器，同时集群中已经有过半的机器与该Leader服务器完成了状态同步之后，Zab协议就会退出恢复模式，状态同步是指数据同步，用来保证集群在过半的机器能够和Leader服务器的数据状态保持一致。

当集群中已经有过半的Follower服务器完成了和Leader服务器的状态同步，那么整个服务框架就可以进入消息广播模式。

当一台同样遵守Zab协议的服务器启动后加入到集群中，如果此时集群中已经存在一个Leader服务器在负责进行消息广播，那么加入的服务器就会自觉地进入数据恢复模式：找到Leader所在的服务器，并与其进行数据同步，然后一起参与到消息广播流程中去。

Zookeeper只允许唯一的一个Leader服务器来进行事务请求的处理，Leader服务器在接收到客户端的事务请求后，会生成对应的事务提议并发起一轮广播协议，而如果集群中的其他机器收到客户端的事务请求后，那么这些非Leader服务器会首先将这个事务请求转发给Leader服务器。

消息广播



消息广播

Zab协议的消息广播过程使用是一个原子广播协议，类似一个2PC提交过程。具体的：

ZooKeeper使用单一主进程Leader用于处理客户端所有事务请求，并采用Zab的原子广播协议，将服务器数据状态变更以事务Proposal的形式广播Follower上，因此能很好的处理客户端的大量并发请求。

另一方面，由于事务间可能存在着依赖关系，Zab协议保证Leader广播的变更序列被顺序的处理，有些状态的变更必须依赖于比它早生成的那些状态变更。

最后，考虑到主进程Leader在任何时候可能崩溃或者异常退出， 因此Zab协议还要Leader进程崩溃的时候可以重新选出Leader并且保证数据的完整性；Follower收到Proposal后，写到磁盘，返回Ack。Leader收到大多数ACK后，广播Commit消息，自己也提交该消息。Follower收到Commit之后，提交该消息。

Zab协议简化了2PC事务提交：

去除中断逻辑移除，follower要么ack，要么抛弃Leader。

Leader不需要所有的Follower都响应成功，只要一个多数派Ack即可。

崩溃恢复

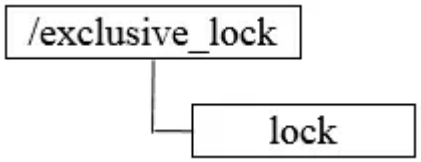
上面我们讲了Zab协议在正常情况下的消息广播过程，那么一旦Leader服务器出现崩溃或者与过半的follower服务器失去联系，就进入崩溃恢复模式。

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实践，共享锁，Leader选举

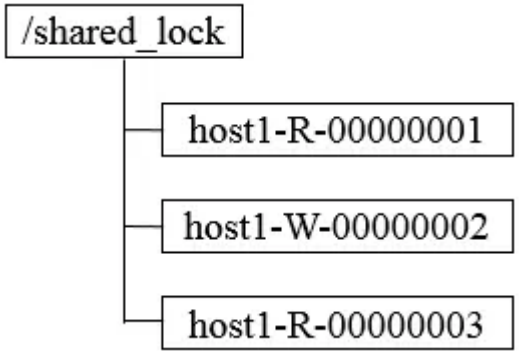
分布式锁用于控制分布式系统之间同步访问共享资源的一种方式，可以保证不同系统访问一个或一组资源时的一致性，主要分为排它锁和共享锁。

排它锁又称为写锁或独占锁，若事务T1对数据对象O1加上了排它锁，那么在整个加锁期间，只允许事务T1对O1进行读取和更新操作，其他任何事务都不能再对这个数据对象进行任何类型的操作，直到T1释放了排它锁。



独占锁

共享锁又称为读锁，若事务T1对数据对象O1加上共享锁，那么当前事务只能对O1进行读取操作，其他事务也只能对这个数据对象加共享锁，直到该数据对象上的所有共享锁都被释放。



共享锁

Leader选举

Leader选举是保证分布式数据一致性的关键所在。当Zookeeper集群中的一台服务器出现以下两种情况之一时，需要进入Leader选举。

服务器初始化启动。

服务器运行期间无法和Leader保持连接。

Zookeeper在3.4.0版本后只保留了TCP版本的 FastLeaderElection 选举算法。当一台机器进入Leader选举时，当前集群可能会处于以下两种状态：

集群中已存在Leader。

集群中不存在Leader。

对于集群中已经存在Leader而言，此种情况一般都是某台机器启动得较晚，在其启动之前，集群已经在正常工作，对这种情况，该机器试图去选举Leader时，会被告知当前服务器的Leader信息，对于该机器而言，仅仅需要和Leader机器建立起连接，并进行状态同步即可。

而在集群中不存在Leader情况下则会相对复杂，其步骤如下：

(1) 第一次投票。无论哪种导致进行Leader选举，集群的所有机器都处于试图选举出一个Leader的状态，即LOOKING状态，LOOKING机器会向所有其他机器发送消息，该消息称为投票。投票中包含了SID（服务器的唯一标识）和ZXID（事务ID），(SID, ZXID)形式来标识一次投票信息。假定Zookeeper由5台机器组成，SID分别为1、2、3、4、5，ZXID分别为9、9、9、8、8，并且此时SID为2的机器是Leader机器，某一时刻，1、2所在机器出现故障，因此集群开始进行Leader选举。在第一次投票时，每台机器都会将自己作为投票对象，于是SID为3、4、5的机器投票情况分别为(3, 9)，(4, 8)， (5, 8)。

(2) 变更投票。每台机器发出投票后，也会收到其他机器的投票，每台机器会根据一定规则来处理收到的其他机器的投票，并以此来决定是否需要变更自己的投票，这个规则也是整个Leader选举算法的核心所在，其中术语描述如下

vote_sid：接收到的投票中所推举Leader服务器的SID。

vote_zxid：接收到的投票中所推举Leader服务器的ZXID。

self_sid：当前服务器自己的SID。

self_zxid：当前服务器自己的ZXID。

每次对收到的投票的处理，都是对(vote_sid, vote_zxid)和(self_sid, self_zxid)对比的过程。

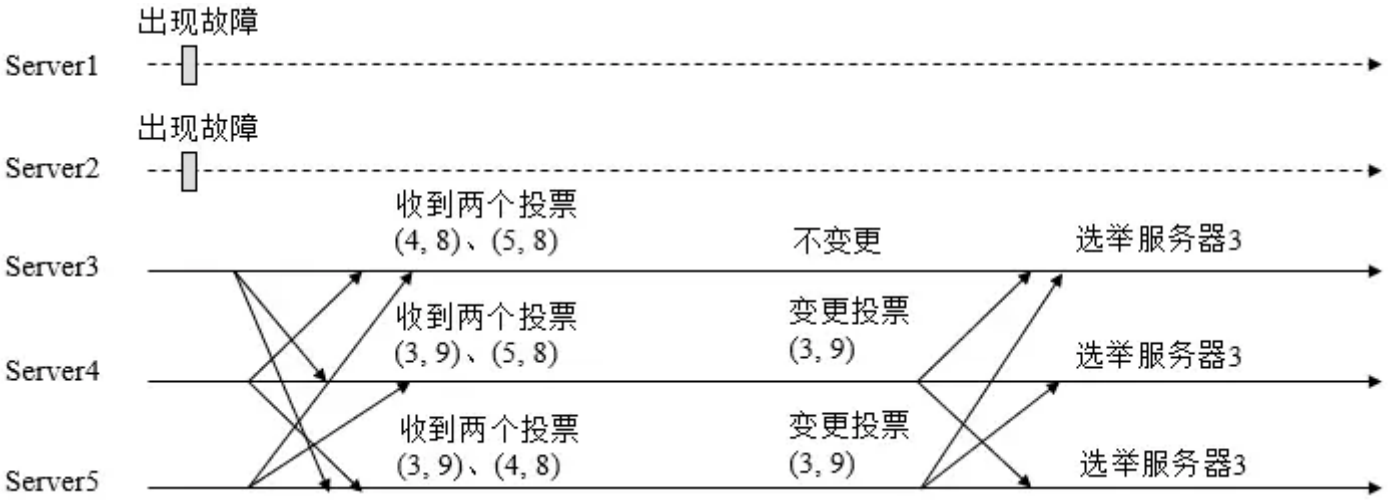
规则一：如果vote_zxid大于self_zxid，就认可当前收到的投票，并再次将该投票发送出去。

规则二：如果vote_zxid小于self_zxid，那么坚持自己的投票，不做任何变更。

规则三：如果vote_zxid等于self_zxid，那么就对比两者的SID，如果vote_sid大于self_sid，那么就认可当前收到的投票，并再次将该投票发送出去。

规则四：如果vote_zxid等于self_zxid，并且vote_sid小于self_sid，那么坚持自己的投票，不做任何变更。

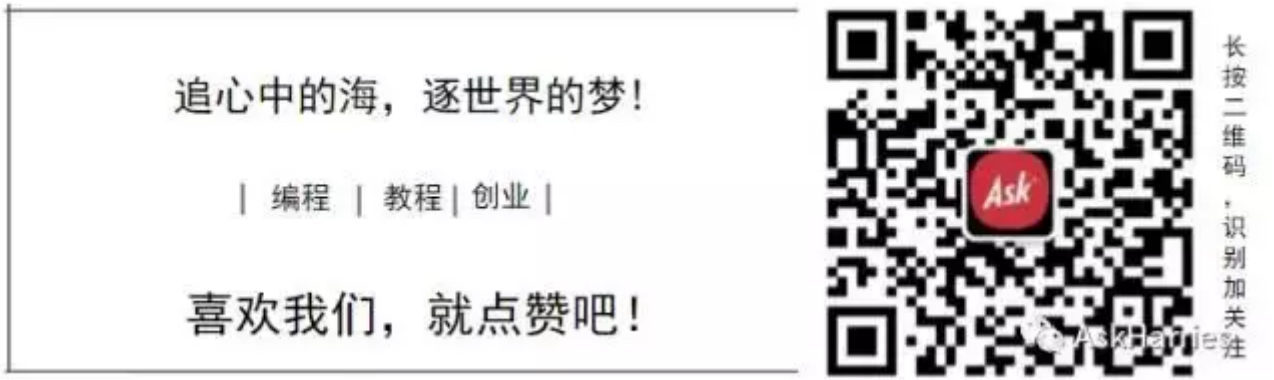
结合上面规则，给出下面的集群变更过程。



Leader选举

(3) 确定Leader。经过第二轮投票后，集群中的每台机器都会再次接收到其他机器的投票，然后开始统计投票，如果一台机器收到了超过半数的相同投票，那么这个投票对应的SID机器即为Leader。此时Server3将成为Leader。

由上面规则可知，通常那台服务器上的数据越新（ZXID会越大），其成为Leader的可能性越大，也就越能够保证数据的恢复。如果ZXID相同，则SID越大机会越大。



关注下面的标签，发现更多相似文章

后端 算法 ZooKeeper 服务器

评论

说说你的看法

Ctrl or ⌘ + Enter 评论

HugoGao Java工程师 @ 网易

讲的很清楚

▲ 0 评论 1月前

飞一样的猪 即兴开发者

牛逼

▲ 0 评论 2月前

相关推荐