# 参考

<https://zhuanlan.zhihu.com/p/44731983>

阿里云云栖号：可能是全网把 ZooKeeper 概念讲的最清楚的一篇文章

<https://github.com/Snailclimb/JavaGuide/blob/main/docs/distributed-system/distributed-process-coordination/zookeeper/zookeeper-intro.md>

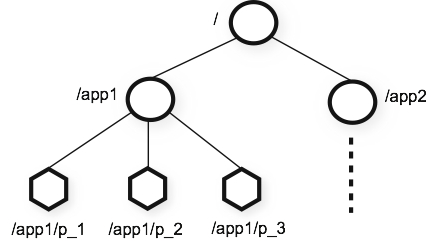
https://github.com/Snailclimb/JavaGuide/blob/main/docs/distributed-system/distributed-process-coordination/zookeeper/zookeeper-plus.md

# ZooKeeper设计目标

zookeeper是一个分布式的一致性协调服务。

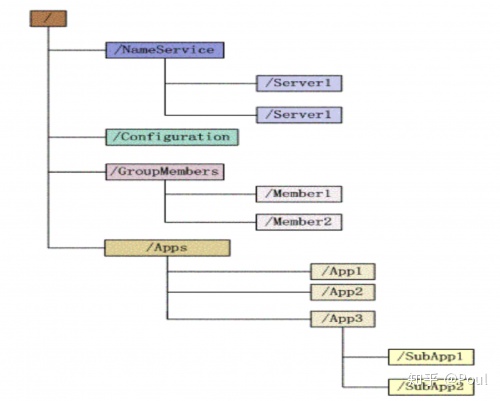
## 数据模型znode

ZooKeeper 允许分布式进程通过共享的层次结构命名空间进行相互协调，这与标准文件系统类似。 名称空间由 ZooKeeper 中的数据寄存器组成 - 称为znode，这些类似于文件和目录。 与为存储设计的典型文件系统不同，ZooKeeper数据保存在内存中，这意味着ZooKeeper可以实现高吞吐量和低延迟。



ZooKeeper 数据模型采用层次化的多叉树形结构，每个节点上都可以存储数据，这些数据可以是数字、字符串或者是二级制序列。并且。每个节点还可以拥有 N 个子节点，最上层是根节点以“/”来代表。每个数据节点在 ZooKeeper 中被称为 znode，它是 ZooKeeper 中数据的最小单元。并且，每个 znode 都一个唯一的路径标识。

**强调一句：ZooKeeper 主要是用来协调服务的，而不是用来存储业务数据的，所以不要放比较大的数据在 znode 上，ZooKeeper 给出的上限是每个结点的数据大小最大是 1M。**



Zookeeper中的节点,没有文件夹和文件之分,所有节点都可以进行数据存储,同时也可以拥有子节点。每个节点称之为znode

### znode的类型分类

* **持久（PERSISTENT）节点** ：一旦创建就一直存在即使 ZooKeeper 集群宕机，直到将其删除。
* **临时（EPHEMERAL）节点** ：临时节点的生命周期是与 客户端会话（session） 绑定的，会话消失则节点消失 。并且，临时节点只能做叶子节点 ，不能创建子节点。
* **持久顺序（PERSISTENT\_SEQUENTIAL）节点** ：除了具有持久（PERSISTENT）节点的特性之外， 子节点的名称还具有顺序性。比如 /node1/app0000000001 、/node1/app0000000002 。
* **临时顺序（EPHEMERAL\_SEQUENTIAL）节点** ：除了具备临时（EPHEMERAL）节点的特性之外，子节点的名称还具有顺序性。

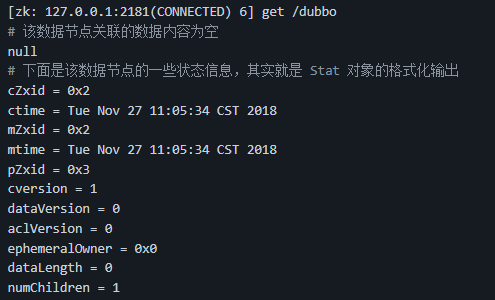
对于来自客户端的每个更新请求，ZooKeeper 都会分配一个全局唯一的递增编号，这个编号反应了所有事务操作的先后顺序，应用程序可以使用 ZooKeeper 这个特性来实现更高层次的同步原语。 这个编号也叫做时间戳——zxid（Zookeeper Transaction Id）

### znode数据结构

每个 znode 由 2 部分组成:

* stat ：状态信息
* data ： 节点存放的数据的具体内容

通过 get 命令来获取 根目录下的 dubbo 节点的内容。（get 命令在下面会介绍到）



Stat 类中包含了一个数据节点的所有状态信息的字段，包括事务 ID-cZxid、节点创建时间-ctime 和子节点个数-numChildren 等等。



### 版本（version）

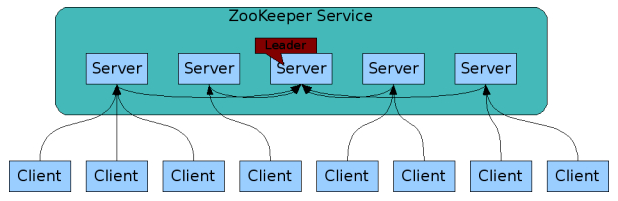
在前面我们已经提到，对应于每个 znode，ZooKeeper 都会为其维护一个叫作 Stat 的数据结构，Stat 中记录了这个 znode 的三个相关的版本：

* + dataVersion ：当前 znode 节点的版本号
  + cversion ： 当前 znode 子节点的版本
  + aclVersion ： 当前 znode 的 ACL 的版本。

## 集群

为了保证高可用，最好是以集群形态来部署 ZooKeeper，这样只要集群中大部分机器是可用的（能够容忍一定的机器故障），那么zookeeper本身仍然是可用的。

客户端在使用 ZooKeeper 时，需要知道集群机器列表，通过与集群中的某一台机器建立 TCP 连接来使用服务，客户端使用这个TCP链接来发送请求、获取结果、获取监听事件以及发送心跳包。如果这个连接异常断开了，客户端可以连接到另外的机器上。



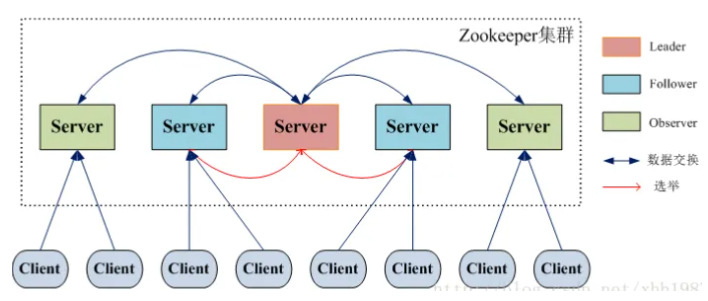
上图中每一个Server代表一个安装Zookeeper服务的服务器。组成 ZooKeeper 服务的服务器都会在内存中维护当前的服务器状态，并且每台服务器之间都互相保持着通信。**集群间通过 Zab 协议（Zookeeper Atomic Broadcast）来保持数据的一致性。**

zookeeper集群可能有N台机器,这些机器中一定会存在一个leader,其他的机器就是follower。对于客户端来说,可以随意连接任何一个集群中服务器。如果某个客户端需要对zk进行更改的操作,这些操作命令最终需要提交给leader。leader将命令分发给所有的集群服务器。当一半以上的集群服务器执行该命令成功,则leader就会通知所有节点进行事务提交,达到数据同步更新的目的。

### 集群角色介绍

最典型集群模式： Master/Slave 模式（主备模式）。在这种模式中，通常 Master服务器作为主服务器提供写服务，其他的 Slave 服务器从服务器通过异步复制的方式获取 Master 服务器最新的数据提供读服务。

ZooKeeper 中没有选择传统的 Master/Slave 概念，而是引入了Leader、Follower 和 Observer 三种角色。如下图所示



三个主要的角色，Leader 领导者、Follower跟随者、Observer观察者 。

* **Leader** ：集群中 唯一的写请求处理者 ，能够发起投票（投票也是为了进行写请求）。
* **Follower**：能够接收客户端的请求，如果是读请求则可以自己处理，如果是写请求则要转发给 Leader 。在选举过程中会参与投票，有选举权和被选举权 。
* **Observer** ：就是没有选举权和被选举权的 Follower 。

因此 Observer 机器可以在不影响写性能的情况下提升集群的读性能。



当 Leader 服务器出现网络中断、崩溃退出与重启等异常情况时，就会进入 Leader 选举过程，这个过程会选举产生新的 Leader 服务器。

这个过程大致是这样的：

* Leader election（选举阶段）：节点在一开始都处于选举阶段，只要有一个节点得到超半数节点的票数，它就可以当选准 leader。
* Discovery（发现阶段） ：在这个阶段，followers 跟准 leader 进行通信，同步 followers 最近接收的事务提议。
* Synchronization（同步阶段） :同步阶段主要是利用 leader 前一阶段获得的最新提议历史，同步集群中所有的副本。同步完成之后 准 leader 才会成为真正的 leader。
* Broadcast（广播阶段） :到了这个阶段，ZooKeeper 集群才能正式对外提供事务服务，并且 leader 可以进行消息广播。同时如果有新的节点加入，还需要对新节点进行同步。

### 服务器状态

* LOOKING ：寻找 Leader。
* LEADING ：Leader 状态，对应的节点为 Leader。
* FOLLOWING ：Follower 状态，对应的节点为 Follower。
* OBSERVING ：Observer 状态，对应节点为 Observer，该节点不参与 Leader 选举。

### 过半机制是如何防止脑裂现象产生的？

对于一个集群，通常多台机器会部署在不同机房，来提高这个集群的可用性。保证可用性的同时，会发生一种机房间网络线路故障，导致机房间网络不通，而集群被割裂成几个小集群。这时候子集群各自选主导致“脑裂”的情况。

举例说明：比如现在有一个由 6 台服务器所组成的一个集群，部署在了 2 个机房，每个机房 3 台。正常情况下只有 1 个 leader，但是当两个机房中间网络断开的时候，每个机房的 3 台服务器都会认为另一个机房的 3 台服务器下线，而选出自己的 leader 并对外提供服务。若没有过半机制，当网络恢复的时候会发现有 2 个 leader。仿佛是 1 个大脑（leader）分散成了 2 个大脑，这就发生了脑裂现象。脑裂期间 2 个大脑都可能对外提供了服务，这将会带来数据一致性等问题。

zookeeper集群的“过半数存活原则”: 在zookeeper集群中,当存活的机器数量超过总集群一半的时候,整个集群才能正常工作。基于过半数存活原则,zookeeper的集群数量一定是奇数台。

## 顺序访问

对于来自客户端的每个更新请求，ZooKeeper 都会分配一个全局唯一的递增编号，这个编号反应了所有事务操作的先后顺序，应用程序可以使用 ZooKeeper 这个特性来实现更高层次的同步原语。 这个编号也叫做时间戳——zxid（Zookeeper Transaction Id）

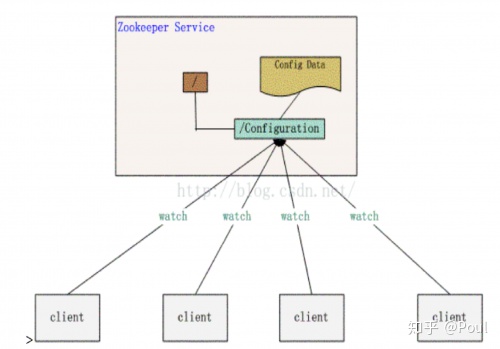
## 高性能

ZooKeeper 是高性能的。 在“读”多于“写”的应用程序中尤其地高性能，因为“写”会导致所有的服务器间同步状态。（“读”多于“写”是协调服务的典型场景。）

# 使用场景

## 配置文件统一管理

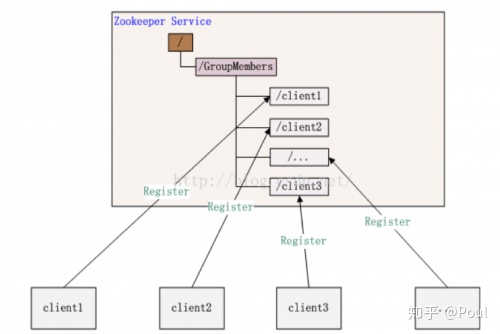
在分布式集群的工程中,通常由很多服务部署在不同的服务上,每个服务都有自己的配置信息,如果需要修改某个配置,则可能需要对多态服务器进行配置的修改,是非常不方便的。那么就可以使用zookeeper帮助我们进行统一的配置文件管理。



在zookeeper上创建一个持久化节点,将所有的配置信息放入到这个节点中,然后每台服务器都去监听这个节点的变化(Watch机制)。如果有新的配置信息,开发者只需要上传到zookeeper的这个节点上(更新节点的配置数据)。每个服务就能收到zookeeper节点的更新通知,然后从节点中读取新的配置,应用到系统中,完成配置的更新。

## 集群管理

在某些集群中,可能需要知道其他集群服务器的状态,比如有新的机器加入集群,或者有老的机器退出集群等。这个时候就可以通过zookeeper进行集群的统一管理。



## 分布式锁

通过创建唯一节点获得分布式锁，当获得锁的一方执行完相关代码或者是挂掉之后就释放锁。

所有客户端同时在一个节点的下面创建临时顺序节点。然后只需要让编号最小的节点的机器获得锁就可以了。

## 命名服务

可以通过 ZooKeeper 的顺序节点生成全局唯一 ID

## 数据发布/订阅

通过 Watcher 机制 可以很方便地实现数据发布/订阅。当你将数据发布到 ZooKeeper 被监听的节点上，其他机器可通过监听 ZooKeeper 上节点的变化来实现配置的动态更新。

# 有哪些著名的开源项目用到了 ZooKeeper?

## Kafka

ZooKeeper 主要为 Kafka 提供 Broker 和 Topic 的注册以及多个 Partition 的负载均衡等功能。

## Hbase

ZooKeeper 为 Hbase 提供确保整个集群只有一个 Master 以及保存和提供 regionserver 状态信息（是否在线）等功能。

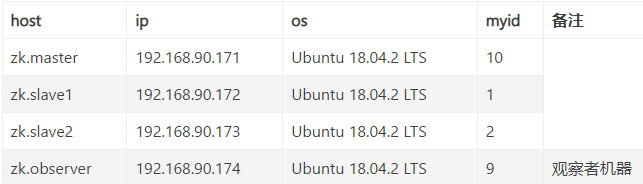
## Hadoop

ZooKeeper 为 Namenode 提供高可用支持。

# 其他

## zookeeper集群部署方式（leader+follower+observer）

<https://www.jianshu.com/p/e9becafcbaa7>



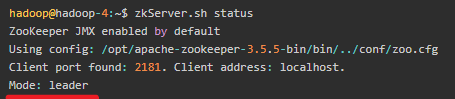
非observer节点zoo.cfg配置



针对observer角色配置，在zk.observer机器的zoo.cfg上单独添加：

peerType=observer

通过zkServer.sh status命令查看zookeeper状态,



通过jps命令可以查看zookeeper的java进程为QuorumPeerMain：



**QuorumPeerMain**：zookeeper集群的启动入口类，是用来加载配置启动QuorumPeer线程的。

**QuorumPeer：**谷歌翻译quorum是法定人数，定额的意思，peer是对等的意思，quorum代表的意思就是每个zookeeper集群启动的时候集群中zookeeper服务数量就已经确定。

**QuorumPeerConfig**会构建一个QuorumServer对象，每个zookeeper都会编辑自己的sid在dataDir目下的myid文件中，sid标记每个服务，在快速选举中起作用。

zookeeper是基于paxos算法实现的，在每个zookeeper的配置文件zoo.cfg中配置集群中的所有机器

## ACL（权限控制）

ZooKeeper 采用 ACL（AccessControlLists）策略来进行权限控制，类似于 UNIX 文件系统的权限控制。

对于 znode 操作的权限，ZooKeeper 提供了以下 5 种：

* + CREATE : 能创建子节点
  + READ ：能获取节点数据和列出其子节点
  + WRITE : 能设置/更新节点数据
  + DELETE : 能删除子节点
  + ADMIN : 能设置节点 ACL 的权限

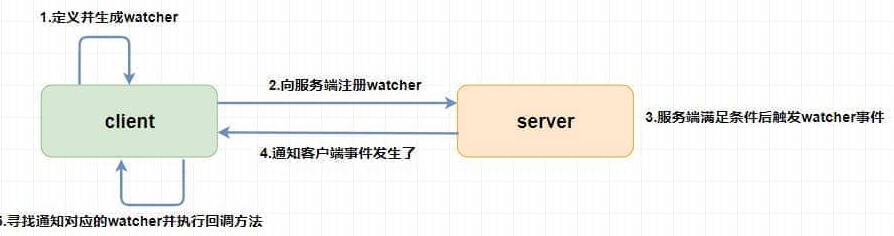
其中尤其需要注意的是，CREATE 和 DELETE 这两种权限都是针对 子节点 的权限控制。

对于身份认证，提供了以下几种方式：

* + world ： 默认方式，所有用户都可无条件访问。
  + auth :不使用任何 id，代表任何已认证的用户。
  + digest :用户名:密码认证方式： username:password 。
  + ip : 对指定 ip 进行限制。

## Watcher（事件监听器）

Watcher（事件监听器），是 ZooKeeper 中的一个很重要的特性。ZooKeeper 允许用户在指定节点上注册一些 Watcher，并且在一些特定事件触发的时候，ZooKeeper 服务端会将事件通知到感兴趣的客户端上去，该机制是 ZooKeeper 实现分布式协调服务的重要特性。



## 会话（Session）

Session 可以看作是 ZooKeeper 服务器与客户端的之间的一个 TCP 长连接，通过这个连接，客户端能够通过心跳检测与服务器保持有效的会话，也能够向 ZooKeeper 服务器发送请求并接受响应，同时还能够通过该连接接收来自服务器的 Watcher 事件通知。

Session 有一个属性叫做：sessionTimeout ，sessionTimeout 代表会话的超时时间。当由于服务器压力太大、网络故障或是客户端主动断开连接等各种原因导致客户端连接断开时，只要在sessionTimeout规定的时间内能够重新连接上集群中任意一台服务器，那么之前创建的会话仍然有效。

另外，在为客户端创建会话之前，服务端首先会为每个客户端都分配一个 sessionID。由于 sessionID是 ZooKeeper 会话的一个重要标识，许多与会话相关的运行机制都是基于这个 sessionID 的，因此，无论是哪台服务器为客户端分配的 sessionID，都务必保证全局唯一。

## ZAB 协议和 Paxos 算法

Paxos 算法应该可以说是 ZooKeeper 的灵魂了。但是，ZooKeeper 并没有完全采用 Paxos 算法 ，而是使用 ZAB 协议作为其保证数据一致性的核心算法。另外，在 ZooKeeper 的官方文档中也指出，ZAB 协议并不像 Paxos 算法那样，是一种通用的分布式一致性算法，它是一种特别为 Zookeeper 设计的崩溃可恢复的原子消息广播算法。

ZAB（ZooKeeper Atomic Broadcast 原子广播） 协议是为分布式协调服务 ZooKeeper 专门设计的一种支持崩溃恢复的原子广播协议。 在 ZooKeeper 中，主要依赖 ZAB 协议来实现分布式数据一致性，基于该协议，ZooKeeper 实现了一种主备模式的系统架构来保持集群中各个副本之间的数据一致性。

ZAB 协议包括两种基本的模式，分别是

* **崩溃恢复**：当整个服务框架在启动过程中，或是当 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。其中，所谓的状态同步是指数据同步，用来保证集群中存在过半的机器能够和 Leader 服务器的数据状态保持一致。
* **消息广播：**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了。 当一台同样遵守 ZAB 协议的服务器启动后加入到集群中时，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，那么新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。