# Zookeeper -- 介绍、原理、应用

田志鹏 20151105

内容:

1. Zookeeper是什么
2. Zookeeper实现原理
3. Zookeeper与Paxos
4. 应用场景

# ZK是什么？要做什么？

# Zookeeper是什么

- 分布式协调服务
- 解决多台机器上同一个值的多个副本如何一致的问题

---

用他来干什么：

It exposes common services - such as naming, configuration management, synchronization, and group services - in a simple interface so you don't have to write them from scratch

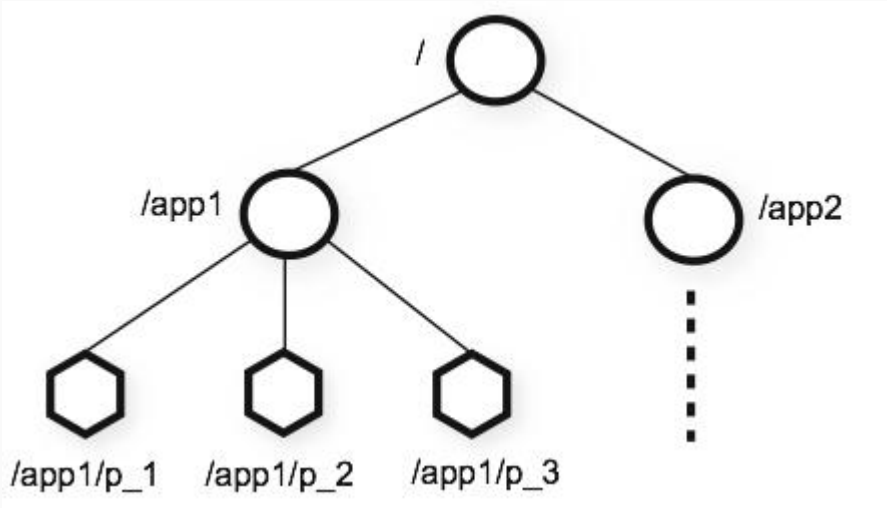
所有需要解决协调问题的分布式系统都可以借助Zookeeper（分布式问题很难搞，竞争、死锁）

# Zookeeper是什么

数据模型:

- 类似文件系统的树形结构
- 类似文件系统的操作

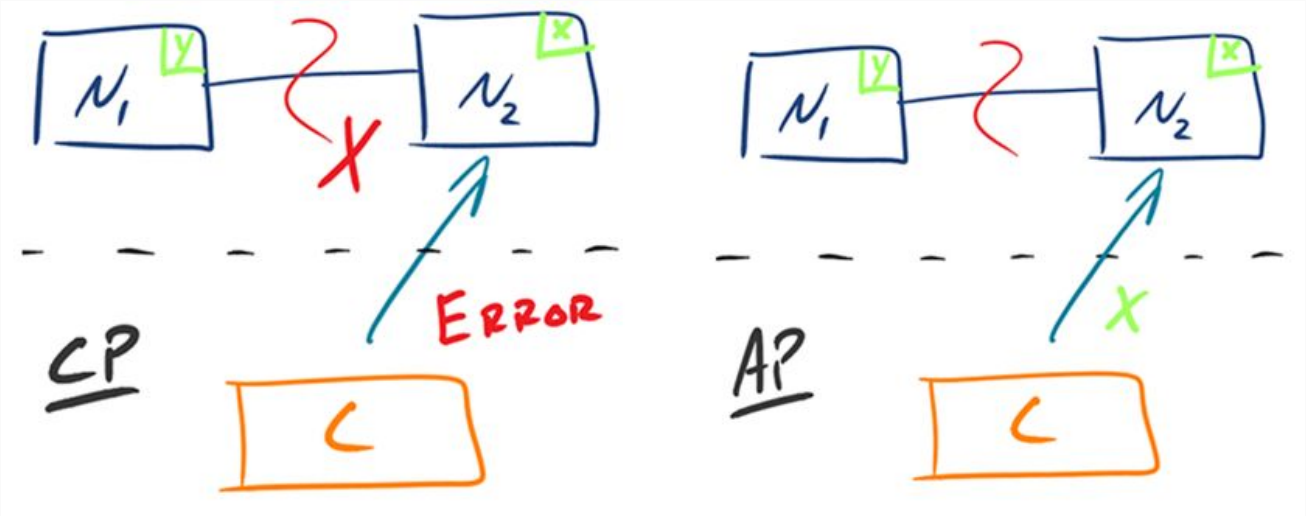
- 每个节点znode 既是目录又是文件
- 节点存储了数据、版本号、时间戳、ACL等信息
- 在内存中(一个HashMap用于快速查询, 一个字典树Trie用于序列化)
- 有持久化:快照和事务日志
- 数据大小限制(默认最大1M)
- 临时节点  序列节点
- Watch



# Zookeeper是什么

CAP定理: 在分布式系统(web service)中至多只能满足下面两项:

1. 一致性(Consistency等同于所有节点访问同一份最新的数据副本)
2. 可用性(Availability)(对数据更新具备高可用性)
3. 容忍网络分区(Partition tolerance)(以实际效果而言，分区相当于对通信的时限要求)



## Zookeeper的设计目标/保证:

- Sequential Consistency - Updates from a client will be applied in the order that they were sent.
- Atomicity - Updates either succeed or fail. No partial results.
- Single System Image - A client will see the same view of the service regardless of the server that it connects to.
- Reliability - Once an update has been applied, it will persist from that time forward until a client overwrites the update.
- Timeliness - The clients view of the system is guaranteed to be up-to-date within a certain time bound.

# Zookeeper是什么 -- 宏观

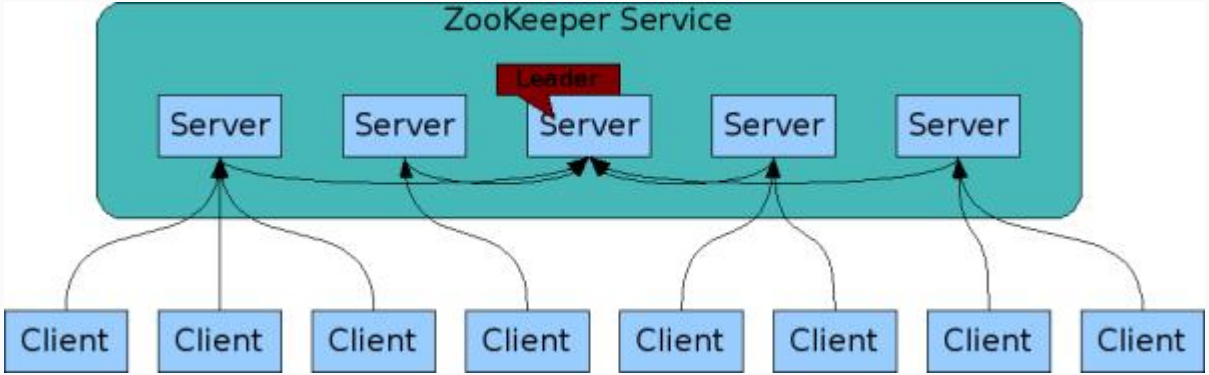
## Zookeeper的设计目标/保证：

- **顺序一致性**：来自一个客户端的请求会按照他的发送顺序来生效（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都在消息b之前发布）
- **原子性**：更新只能成功 或者失败，没有中间结果
- **单一系统镜像**：client无论连接到哪个server，都会看到同样的视图。（不是时间上的 同时）
- **可靠性**：一旦更新生效，他会一直有效直到被覆盖。（fail-over特性）
- **时效性**：客户端对于系统的视图，在一定时间范围内被保证up-to-date（不是实时最新）

# ZK结构和原理

# Zookeeper中的结构

整体结构：




Serving状态/角色：

| | | |
|---|---|---|
| Looking | 存在无主的状态，进行选主 | 都为client提供服务。<br>读请求返回副本。<br>写请求发给leader。 |
| Leading | 只有一个leader，只有他发起提议。心跳 | |
| Following | 同步状态，参与对提议表决 和 选举 | |
| Observing | 同步状态，不参与其他 | |

# Zookeeper 提议流程

过程:

1. cient的请求被转发给Leader。
2. Leader将提案发给所有Follower
3. Follower收到之后ACK(不能反对)
4. Leader收到超过半数的ACK就通过发送commit

要点:

1. 只有Leader能处理提案
2. 提案有全局唯一的编号zxid

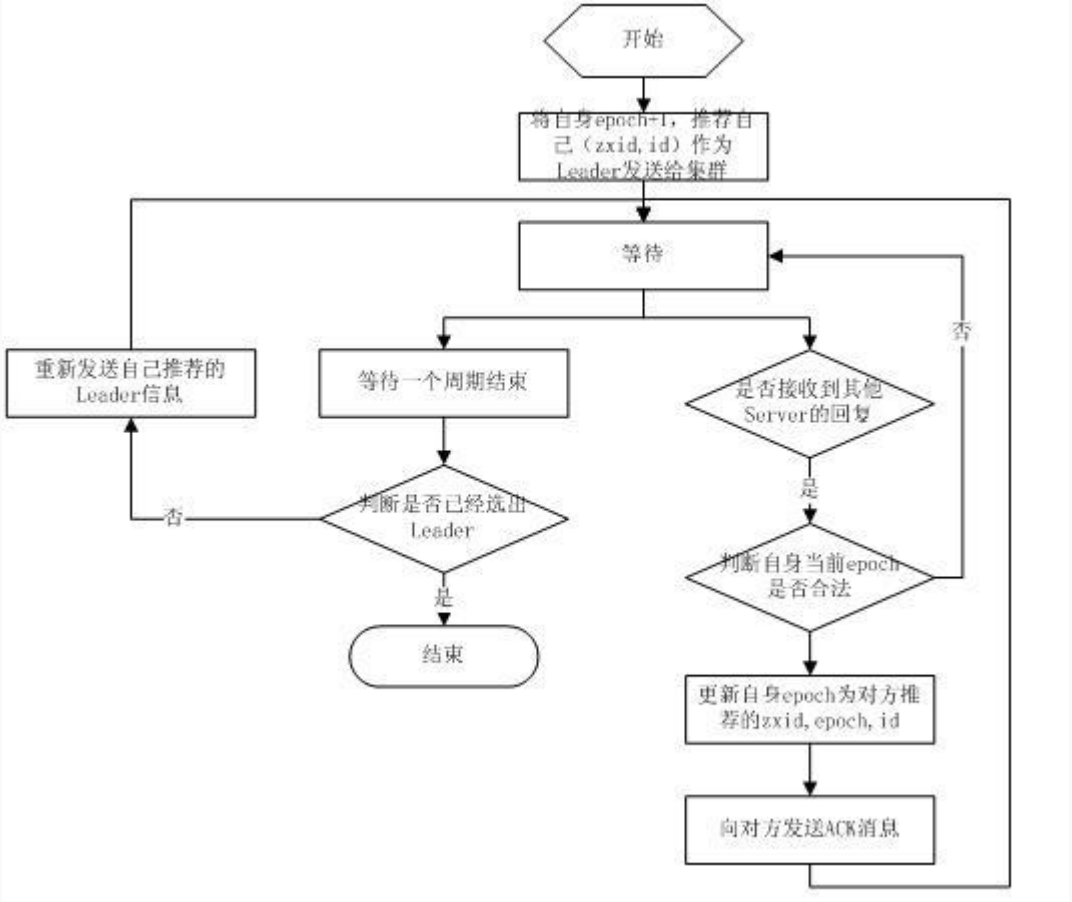
# Zookeeper 学习过程

1 .Follower连接leader，将最大的zxid发送给leader；
2 .Leader根据follower的zxid确定同步点；
3 .完成同步后通知follower 已经成为uptodate状态；

# Zookeeper 选主过程

改进的Paxos流程：

1. Server首先向所有Server提议自己要成为leader，
2. 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
3. 重复这个流程，最后一定能选举出Leader



# Paxos

Paxos决议分为两个阶段：

- prepare阶段：
  - proposer选择一个提案编号n并将prepare请求发送给acceptors中的一个多数派
  - acceptor收到prepare消息后，如果提案的编号大于它已经回复的所有prepare消息，则acceptor将自己上次接受的提案回复 给proposer，并承诺不再回复小于n的提案；
- 批准阶段：
  - 当一个proposer收到了多数acceptors对prepare的回复后，就进入批准阶段。它要向回复prepare请求的acceptors发送accept请求，包括编号n和根据P2c决定的value（如果根据P2c没有已经接受的value，那么它可以自由决定value）。
  - 在不违背自己向其他proposer的承诺的前提下，acceptor收到accept请求后即接受这个请求。

# 比较

Fast Paxos：有Leader的Paxos

Zookeeper的工作流程：类似fast paxos，就是不可以反对意见的二段提交

Zookeeper的选主流程：只有一段的paxos

# 应用场景

# 应用场景

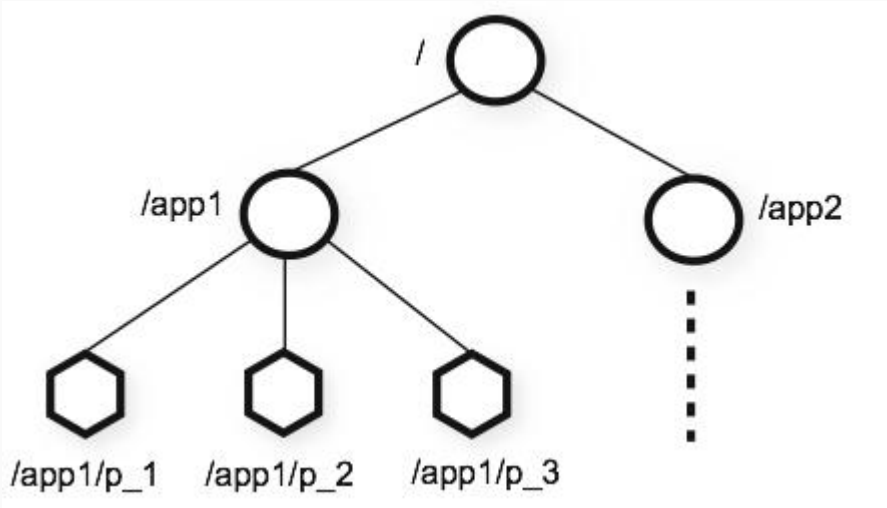
1. 配置服务:统一管理分布式系统的配置
2. 命名服务:全局唯一不重复的名字
3. 集群管理/Leader 选举:hadoop
4. 共享锁: 临时的序列节点、最小的
5. 队列管理/生产者消费者队列

# Zookeeper是什么

数据模型:

- 类似文件系统的树形结构
- 类似文件系统的操作

- 每个节点znode 既是目录又是文件
- 节点存储了数据、版本号、时间戳、ACL等信息
- 在内存中(一个HashMap用于快速查询, 一个字典树Trie用于序列化)
- 有持久化:快照和事务日志
- 数据大小限制(默认最大1M)
- 临时节点  序列节点
- Watch



# 参考资料

CAP

- http://www.tuicool.com/articles/BVBbayV
- https://zh.wikipedia.org/wiki/CAP%E5%AE%9A%E7%90%86 wiki
- http://www.infoq.com/cn/articles/cap-twelve-years-later-how-the-rules-have-changed cap十二年后

Paxos

https://zh.wikipedia.org/wiki/Paxos%E7%AE%97%E6%B3%95#.E5.AE.9E.E4.BE.8B wiki

Zookeeper

- http://zookeeper.apache.org/doc/r3.4.6/index.html 官方介绍
- http://cailin.iteye.com/blog/2014486 原理分析（第一张图有问题）
-