Zookeeper 学习笔记

概述

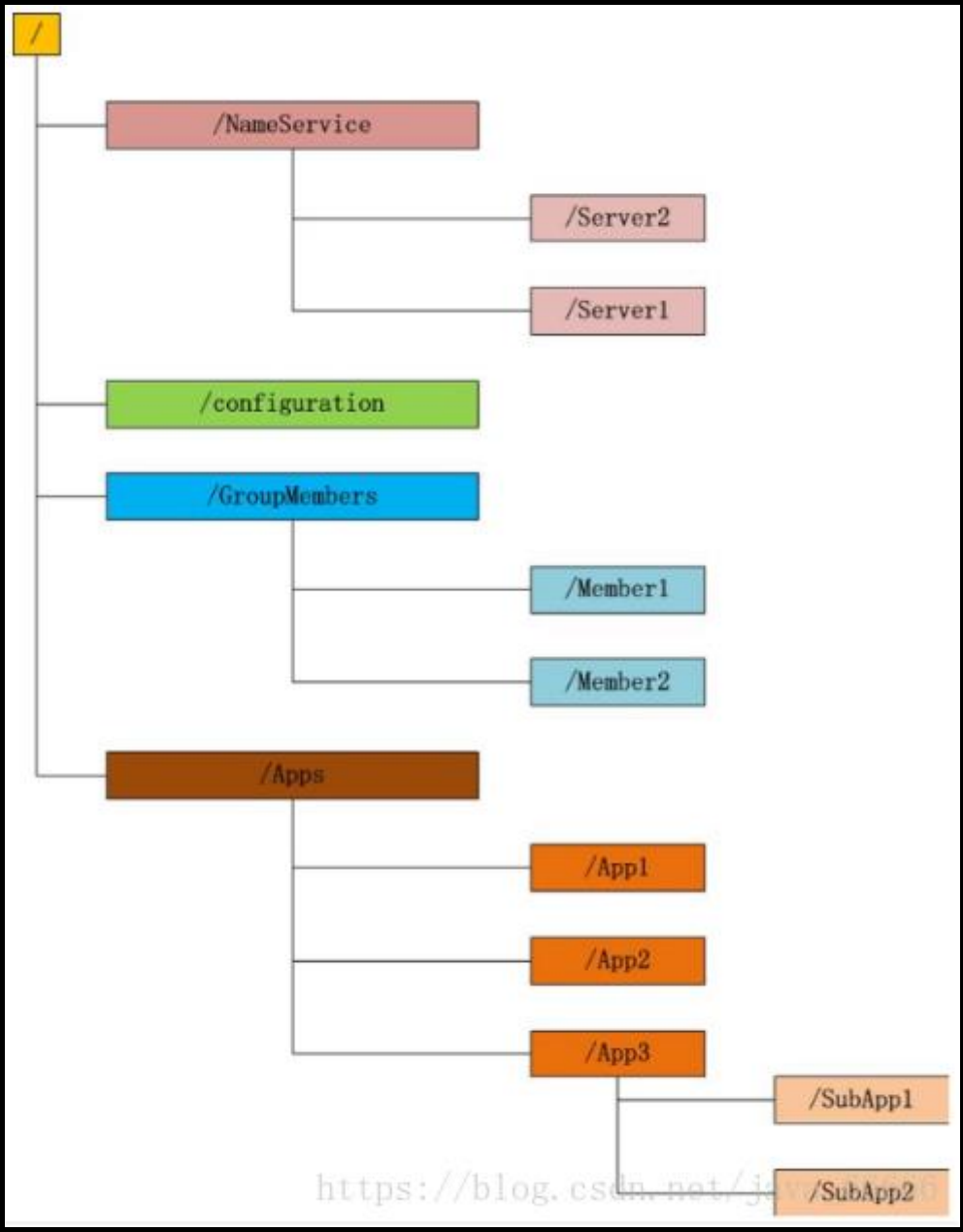
Zookeeper 是什么

zookeeper，它是一个分布式服务框架，是 Apache Hadoop 的一个子项目，它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。

简单来说 **zookeeper=文件系统+监听通知机制。**

文件系统：
Zookeeper 维护一个类似文件系统的数据结构：

Edwin Xu



每个子目录项如 NameService 都被称作为 **znode**(目录节点)，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。

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 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

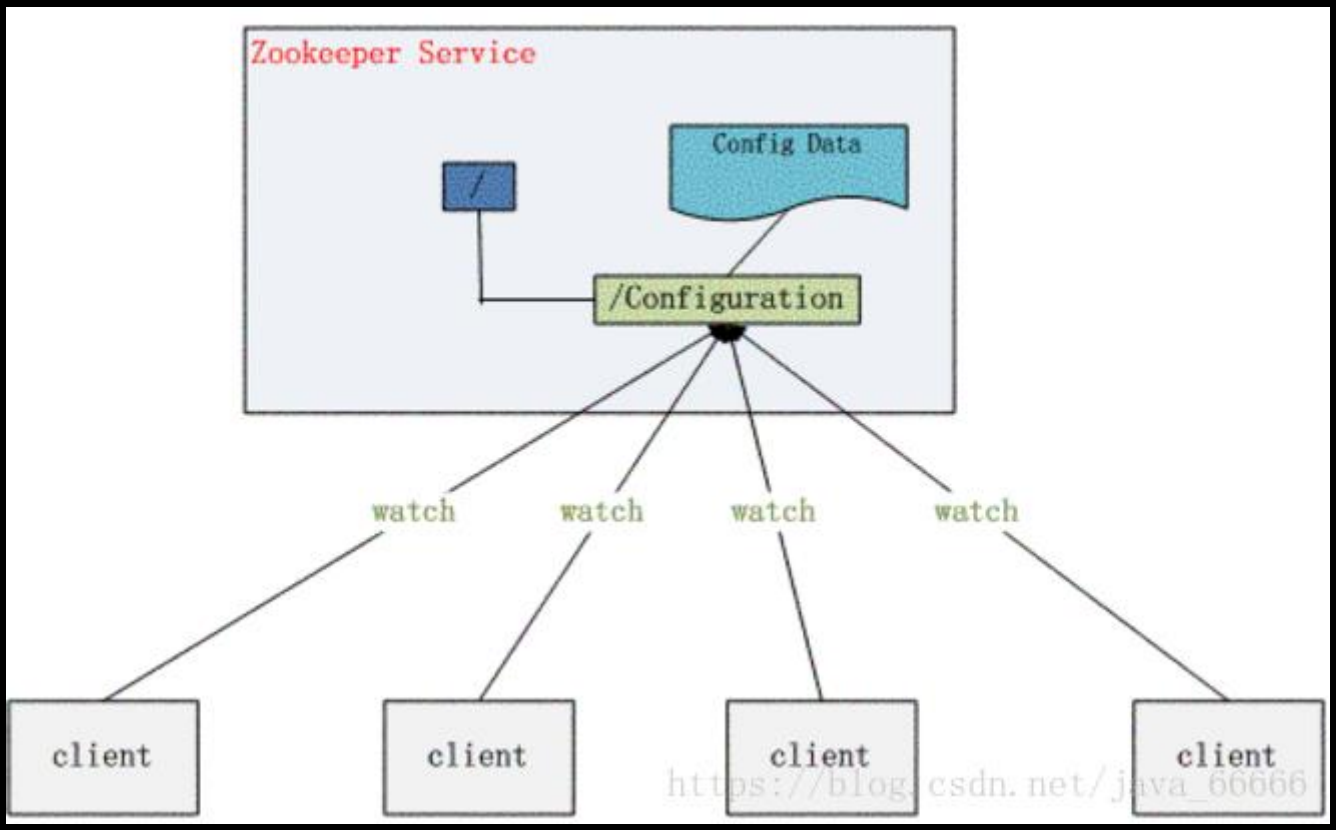
监听通知机制：
客户端注册监听它关心的目录节点，当目录节点发生变化(数据改变、被删除、子目录节点增加删除)

时，zookeeper 会通知客户端。

Zookeeper 能做什么

zookeeper 功能非常强大，可以实现诸如**分布式应用配置管理、统一命名服务、状态同步服务、集群管理等**功能，我们这里拿比较简单的分布式应用配置管理为例来说明。

假设我们的程序是分布式部署在多台机器上，如果我们要改变程序的配置文件，需要逐台机器去修改，非常麻烦，现在把这些配置全部放到 zookeeper 上去，保存在 zookeeper 的某个目录节点中，然后所有相关应用程序对这个目录节点进行监听，一旦配置信息发生变化，每个应用程序就会收到 zookeeper 的通知，然后从 zookeeper 获取新的配置信息应用到系统中。



为什么需要 Zookeeper

https://zhuanlan.zhihu.com/p/69114539

正经点来回答，就是我们需要一个**用起来像单机但是又比单机更可靠的东西。**

分布式系统例子：
比如我们搭建了一个数据库集群，里面有一个 Master，多个 Slave，Master 负责写，Slave 只读，我们需要一个系统，来告诉客户端，哪个是 Master。

Edwin Xu

有人说，很简单，我们把这个信息写到一个 Java 服务器的内存就好了，用一个 map，key:master，value:master 机器对应的 ip
但是别忘了，这是个单机，一旦这个机器挂了，就完蛋了，客户端将无法知道到底哪个是 Master。
于是开始进行拓展，拓展成三台服务器的集群。
这下问题来了，如果我在其中一台机器修改了 Master 的 ip，数据还没同步到其他两台，这时候客户端过来查询，如果查询走的是另外两台还没有同步到的机器，就会拿到旧的数据，往已经不是 master 的机器写数据。
所以我们需要**这个存储 master 信息的服务器集群，做到当信息还没同步完成时，不对外提供服务**，阻塞住查询请求，等待信息同步完成，再给查询请求返回信息。

这样一来，请求就会变慢，变慢的时间取决于什么时候这个集群认为数据同步完成了。

假设这个数据同步时间无限短，比如是 1 微妙，可以忽略不计，那么其实这个分布式系统，就和我们之前单机的系统一样，既可以保证数据的一致，又让外界感知不到请求阻塞，同时，又不会有 SPOF（Single Point of Failure）的风险，即不会因为一台机器的宕机，导致整个系统不可用。

这样的系统，就叫**分布式协调系统**。谁能把这个**数据同步的时间压缩的更短**，谁的请求响应就更快，谁就更出色，Zookeeper 就是其中的佼佼者。

它用起来像单机一样，能够提供**数据强一致性**，但是其实背后是多台机器构成的集群，不会有 SPOF。

其实就是 CAP 理论中，**满足 CP，不满足 A 的那类分布式系统**

官网：ZooKeeper: A Distributed Coordination Service for Distributed Applications

安装

Java API 操作 zookeeper：

```
<dependency>
    <groupId>org.apache.zookeeper</groupId>
    <artifactId>zookeeper</artifactId>
    <version>3.4.12</version>
</dependency>
```

优缺点

zookeeper 的 CP 模型不适合注册中心
zookeeper 是一个非常优秀的项目，非常成熟，被大量的团队使用，但对于服务发现来讲，zookeeper 真的是一个错误的方案。

在 CAP 模型中，zookeeper 是 CP，意味着面对网络分区时，为了保持一致性，他是不可用的
因为 zookeeper 是一个分布式协调系统，如果使用最终一致性（AP）的话，将是一个糟糕的设计，他的核心算法是 Zab，所有设计都是为了一致性。

应用

很多中间件，比如 Kafka、Hadoop、HBase，都用到了 Zookeeper

<从 Paxos 到 Zookeeper>

官网

Overview

https://zookeeper.apache.org/doc/r3.5.5/zookeeperStarted.html

ZooKeeper: A Distributed Coordination Service for Distributed Applications
分布式协调系统
服务于分布式应用的分布式、开源协调系统

采用类型文件系统的数据结构

协调系统是很难保证高可用性的，很容易出错误

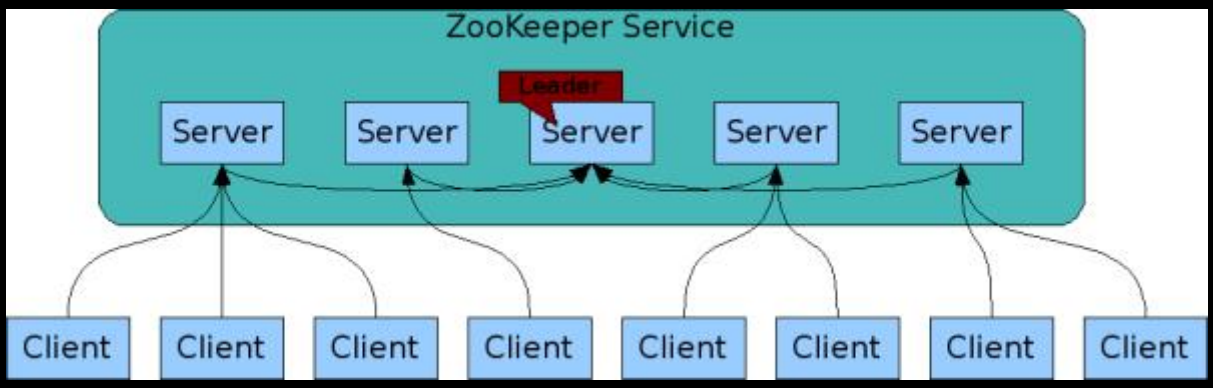
设计目标：
Zookeeper 是简单的，让分布式应用之间分享类似 fS 的分层 namespace，
这些 namespace 包含：

1. Znode: data registers，他们类型文件、文件夹，不想传统的 FS 是用来存储的，Zookeeper 的数据是保存在内存中的——实现高吞吐量和低延迟数

Zookeeper 是高性能、高可用的、strictly ordered access

高性能意味着 zookeeper 可用被用在大的分布式系统
可靠性 reliability 意味着避免单点故障
严格的顺序意味着复杂的同步原语可以在客户机上实现

Zookeeper 是 replicated 的，即可复制的，多节点



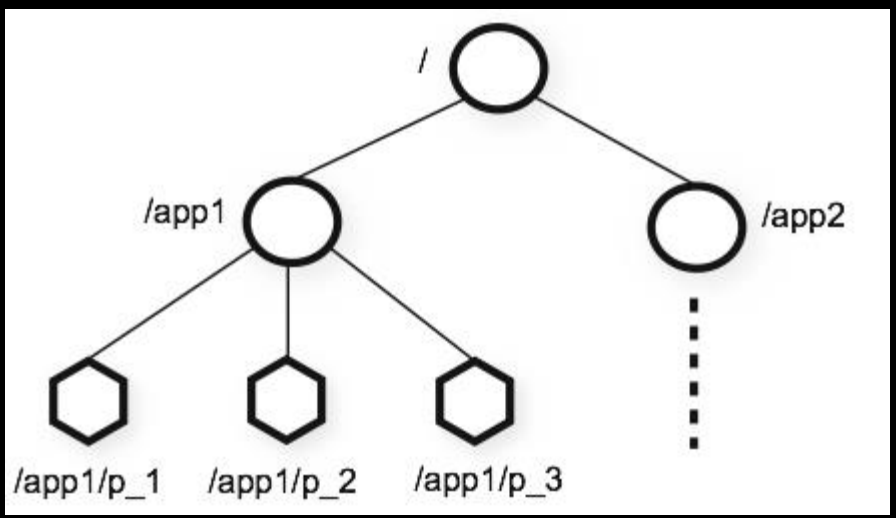
Server 内存中存储状态和数据，当然有持久化快照，server 之间相互通信。

Client 和 server 保持着 TCP 连接，获取监听、发送心跳，一旦连接断开，client 将会转向另一个 server

Zookeeper 是有序的 ordered：用反映所有 ZooKeeper 事务顺序的数字标记每个更新。

Zookeeper 是 fast 的：尤其在读时，非常的快。 Zookeeper 应该用于读多于写的场景，一般读：写>10:1

Data model and the hierarchical namespace:
类型标准的 FS，通过路径标识每一个 znode，如/app1



Nodes and ephemeral(短暂的) nodes
不像 FS，zookeeper 的节点 namespace 都可以有相关的数据和子节点。就像允许文件也是目录的 FS。

Zookeeper 被设计为 存储协调数据：状态信息、配置、位置信息等
因此每个节点存储的数据很小：从 byte 到 kilobyte
Znode：zookeeper data node

Znodes 维护一个 stat 结构，其中包括数据更改的版本号、ACL 改变、timestamp、允许缓存验证和协调更新
当有一个节点的数据改变时，版本号增加，这时 client 再次请求时，一定会请求这个版本的数据。

每个节点都有访问控制列表 Access Control List

Zookeeper 存在 临时节点的概念，当一个 session 创建时建立，session 销毁是销毁

Conditional updates and watches:
Client 可以设置一个 watch 在一个 znode 上，当 znode 数据改变时，watch 被触发并移除

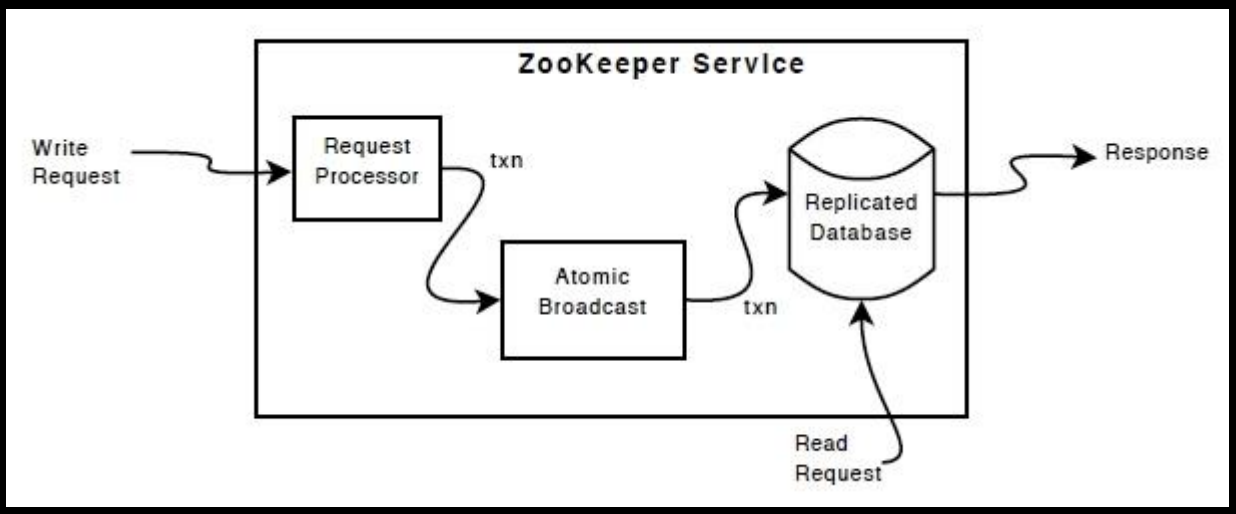
Guarantees:
Zookeeper 是非常简单且快速的，提供如下保障：

1. Sequential Consistency 一致性 - Updates from a client will be applied in the order that they were sent.
2. Atomicity 原子性- Updates either succeed or fail. No partial results.
3. Single System Image 单一系统映像 - A client will see the same view of the service regardless of the server that it connects to.
4. Reliability 可靠性- Once an update has been applied, it will persist from that time forward until a client overwrites the update.
5. Timeliness 及时性 - The clients view of the system is guaranteed to be up-to-date within a certain time bound.

Simple API:

- *create* : creates a node at a location in the tree
- *delete* : deletes a node
- *exists* : tests if a node exists at a location
- *get data* : reads the data from a node
- *set data* : writes data to a node
- *get children* : retrieves a list of children of a node
- *sync* : waits for data to be propagated

Implementation:

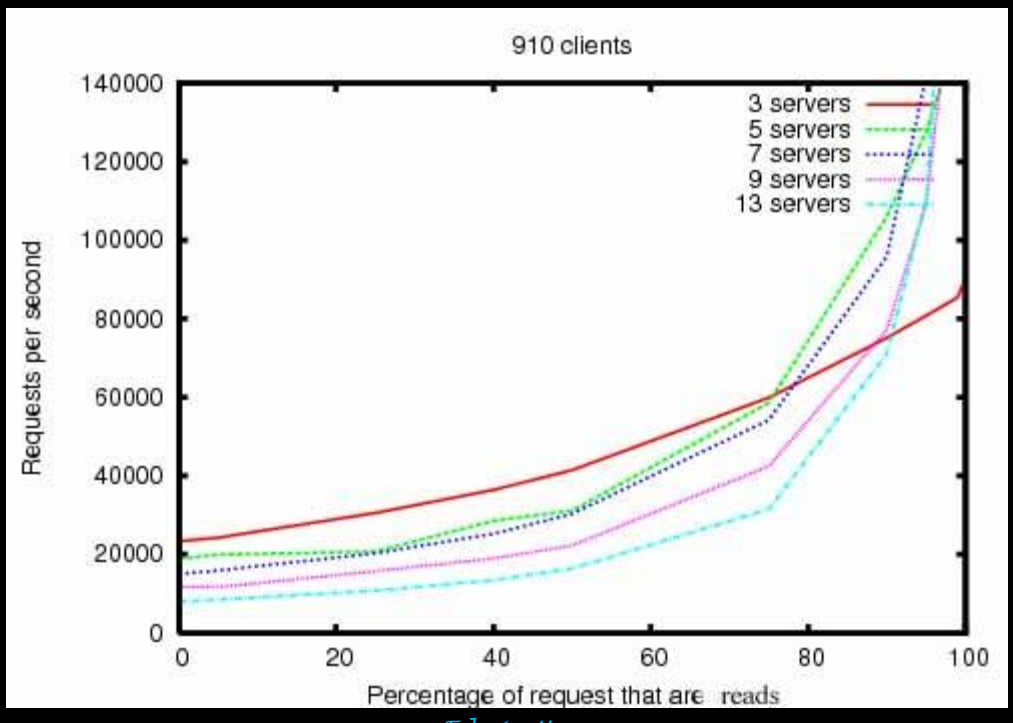


Replicated Database 是一个内存数据库，包含整个数据树， 更新是持久化到磁盘——为了回滚恢复。

读时直接读取，写时需要执行 agreement protocol
agreement protocol 指定，**所有的写请求都被发送到一个单一的 server——Leader,**其他的称为 **Followers**，当 leader 故障时，其中一个 follower 将会成为 Leader

zookeeper 采用一个 **custom atomic messaging protocol(自定义的原子消息协议)**，消息层是原子的，所有副本都一样，不会有不同。
当一个 write 请求发起时，计算系统的状态、应用写需要的时间，然后生成一个事务，在事务中执行写。

性能：



Edwin Xu

ZooKeeper Getting Started Guide

下载一个稳定版：https://zookeeper.apache.org/releases.html

Standalone Operation 单机版：
直截了当，server 就是一个 jar，需要创建一个配置文件
conf/zoo.cfg:

```
tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
```

1. tickTime：基本的时间单元，以毫秒为单位，用来做心跳检测、timeout 等
2. dataDir：用来存储 in-memory database 的快照
3. clientPort：部署的端口

创建配置后，启动：

```
bin/zkServer.sh start
```

zk 使用 log4j 做日志

连接 zookeeper:

```
bin/zkCli.sh -server 127.0.0.1:2181
```

命令：

```
[zkshell: 0] help
ZooKeeper host:port cmd args
    get path [watch]
    ls path [watch]
    set path data [version]
    delquota [-n|-b] path
    quit
    printwatches on|off
    create path data acl
    stat path [watch]
    listquota path
    history
    setAcl path acl
    getAcl path
    sync path
    redo cmdno
    addauth scheme auth
    delete path [version]
    deleteall path
    setquota -n|-b val path
```

eg:

```
[zkshell: 8] ls /
[zkshell: 9] create /zk_test my_data
[zkshell: 12] get /zk_test
```

Running Replicated ZooKeeper 集群:

配置:

```
tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
initLimit=5
syncLimit=2
server.1=zoo1:2888:3888
server.2=zoo2:2888:3888
server.3=zoo3:2888:3888
```

initLimit: 与 leader 的连接过期时间

syncLimit: 限制了服务器与 leader 之间的过期时间

原理

目前单机版可以了，集群、原理还没有学习