# 内容介绍

极客时间

扫码试看/订阅

《ZooKeeper实战与源码剖析》视频课程

极客时间

# 目录

# 1. ZooKeeper 基础

这一章介绍 ZooKeeper 的安装配置，ZooKeeper 的基本概念和 zkCli.sh 的使用，为后面的学习打好基础。

# 2. ZooKeeper 开发

介绍 ZooKeeper API 的使用。首先会对 ZooKeeper 核心 API 做一个讲解，然后结合具体事例讲解如何使用 ZooKeeper API 进行协同服务的开发。

# 3. ZooKeeper 运维

介绍 ZooKeeper 生产环境的安装配置和 ZooKeeper 的监控。监控部分除了介绍 ZooKeeper 自带的监控工具，还会介绍和其他监控系统的集成。

# 4. ZooKeeper 进阶开发

这一章分成两部分。第一部分讲解一个使用 ZooKeeper 来实现服务发现的实战项目，会使用 Apache Curator 的代码来讲。第二部分讲解 Apache Kafka 是如何使用 ZooKeeper 的。

# 5. 比较 ZooKeeper、etcd 和 Chubby

etcd 和 Chubby 是和 ZooKeeper 类似的系统。这一章会对 etcd 和 Chubby 做一个简单的介绍，并把他们和 ZooKeeper 做一个比较，帮助大家在一个更广的视角来理解 ZooKeeper。

# 6. ZooKeeper 实现原理和源码解读

结合 ZooKeeper 的 Paper 讲一下 ZooKeeper 的设计实现原理，并带着大家把 ZooKeeper 的核心模块的代码阅读一下。在讲解 ZooKeeper 原理的时候会把相关的计算机理论知识点讲一下。

# ZooKeeper 概述

# 大纲

- 什么是 ZooKeeper?
- ZooKeeper 发展历史
- ZooKeeper 应用场景

# 什么是 ZooKeeper

ZooKeeper 是一个分布式的，开放源码的分布式应用程序协同服务。ZooKeeper 的设计目标是将那些复杂且容易出错的分布式一致性服务封装起来，构成一个高效可靠的原语集，并以一系列简单易用的接口提供给用户使用。

# ZooKeeper 发展历史

ZooKeeper 最早起源于雅虎研究院的一个研究小组。在当时，研究人员发现，在雅虎内部很多大型系统基本都需要依赖一个类似的系统来进行分布式协同，但是这些系统往往都存在分布式单点问题。

所以，雅虎的开发人员就开发了一个通用的无单点问题的分布式协调框架，这就是 ZooKeeper。ZooKeeper 之后在开源界被大量使用，下面列出了 3 个著名开源项目是如何使用 ZooKeeper：

- Hadoop：使用 ZooKeeper 做 Namenode 的高可用。
- HBase：保证集群中只有一个 master，保存 hbase:meta 表的位置，保存集群中的 RegionServer 列表。
- Kafka：集群成员管理，controller 节点选举。

# ZooKeeper 应用场景

典型应用场景：

- 配置管理（configuration management）
- DNS 服务
- 组成员管理（group membership）
- 各种分布式锁

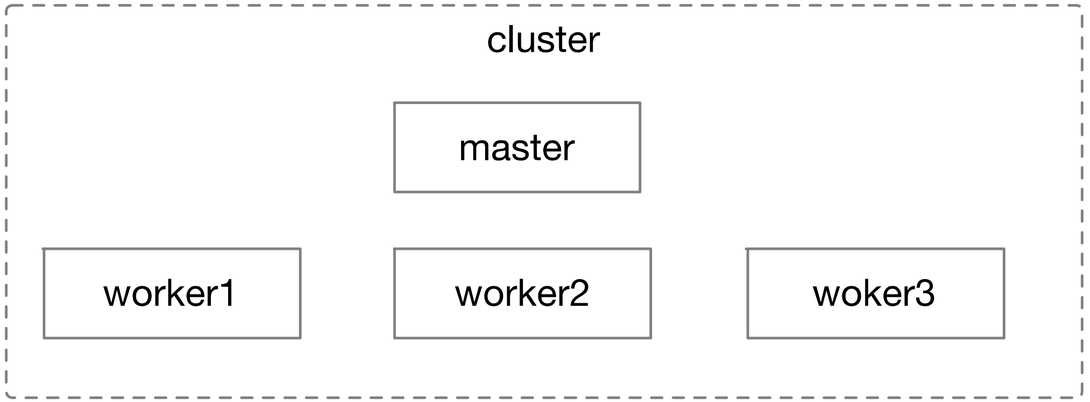
ZooKeeper 适用于存储和协同相关的关键数据，不适合用于大数据量存储。

# ZooKeeper 实现 Master-Worker 协同

# master-worker架构

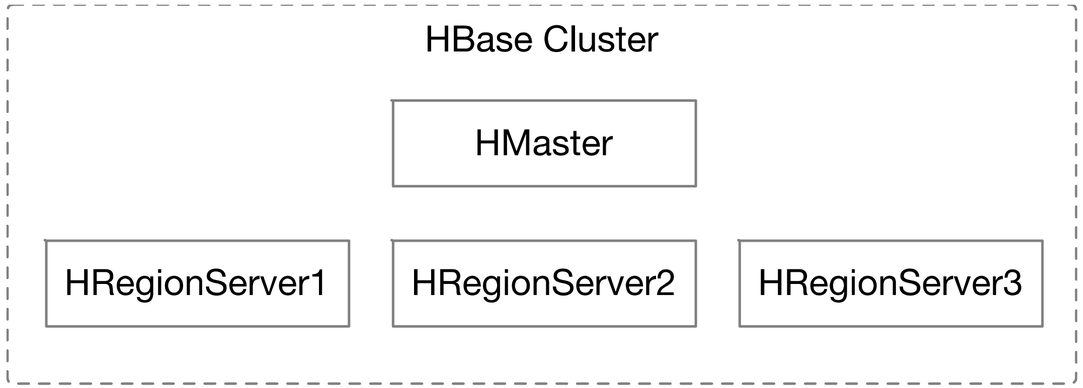
master-work是一个广泛使用的分布式架构。 master-work架构中有一个master负责监控worker的状态，并为worker分配任务。

1. 在任何时刻，系统中最多只能有一个master，不可以出现两个master的情况，多个master共存会导致脑裂。
2. 系统中除了处于active状态的master还有一个bakcup master，如果active master失败了，backup master可以很快的进入active状态。
3. master实时监控worker的状态，能够及时收到worker成员变化的通知。master在收到worker成员变化的时候，通常重新进行任务的重新分配。



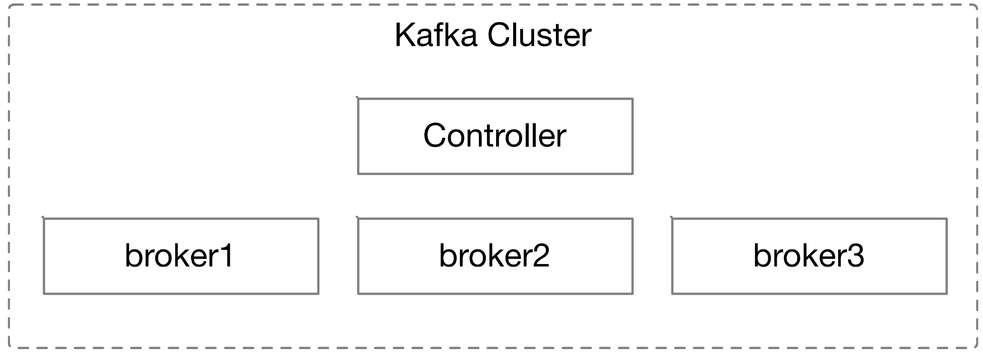
# master-worker架构示例1-HBase

HBase采用的是master-worker的架构。HMBase是系统中的master，HRegionServer是系统中的worker。HMBase监控HBase Cluster中worker的成员变化，把region分配给各个HRegionServer 。系统中有一个HMaster处于active状态，其他HMaster处于备用状态。



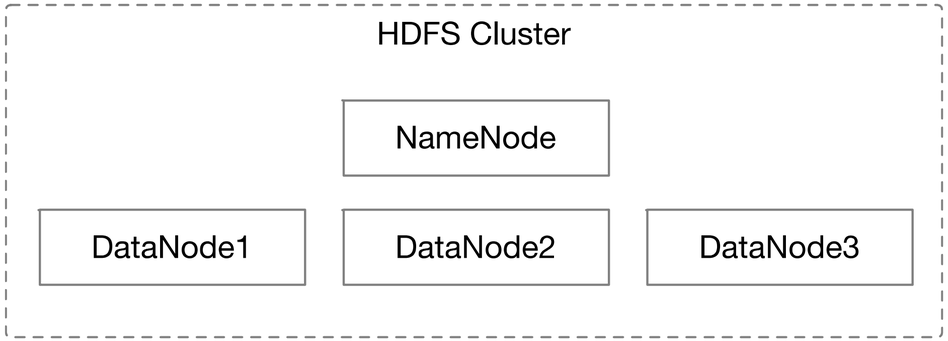
# master-worker架构示例2-Kafka

一个Kafka集群由多个broker组成，这些borker是系统中的worker。Kafkai会从这些worker选举出一个controller，这个controlle是系统中的master，负责把topic partition分配给各个broker。



# master-worker架构示例2-HDFS

HDFS采用的也是一个master-worker的架构，NameNode是系统中的master，DataNode是系统中的worker。NameNode用来保存整个分布式文件系统的metadata，并把数据块分配给cluster中的DataNode进行保存。

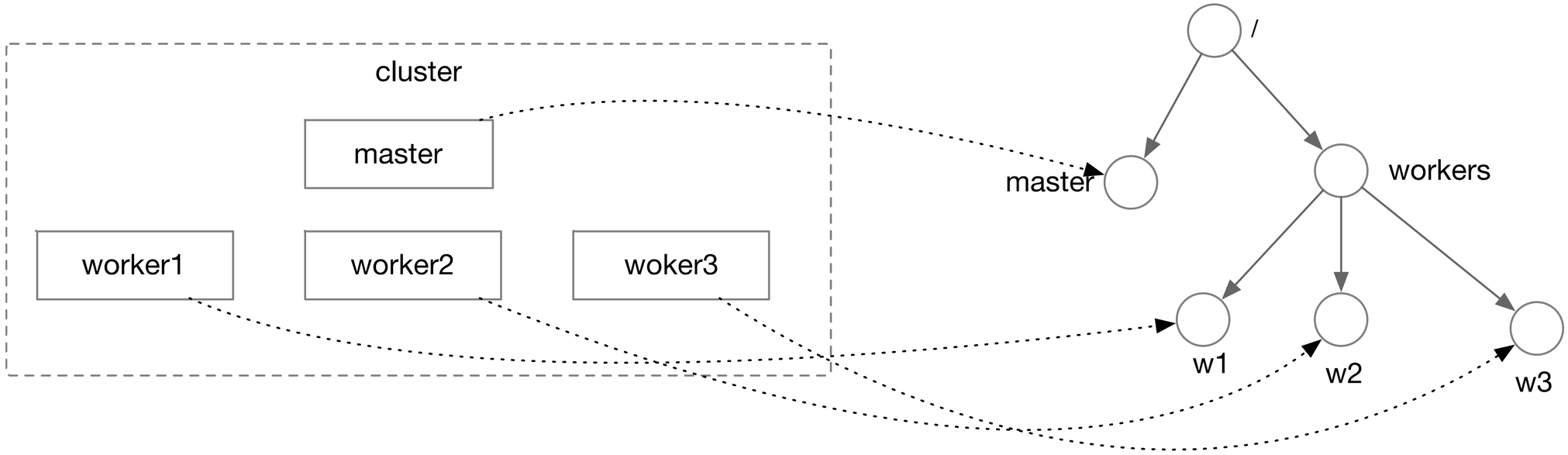


# 如何使用ZooKeeper实现master-worker

1. 使用一个临时节点/master表示master。master在行使master的职能之前，首先要创建这个znode。如果能创建成功，进入active状态，开始行使master职能。否则的话，进入backup状态，使用watch机制监控/master。假设系统中有一个active master和一个backup master。如果active master失败，它创建的/master就会被ZooKeeper自动删除。这时backup master就会收到通知，通过再次创建/master节点成为新的active master。

2. worker通过在/workers下面创建临时节点来加入集群。

3. 处于active状态的master会通过watch机制监控/workers下面znode列表来实时获取worker成员的变化。



# 使用zkCli.sh演示如何实现master-worker

需要首先使用create /workers创建/workers znode。

# ZooKeeper 服务说明

# ZooKeeper 服务的使用

应用使用 ZooKeeper 客户端库使用 ZooKeeper 服务。

ZooKeeper 客户端负责和 ZooKeeper 集群的交互。

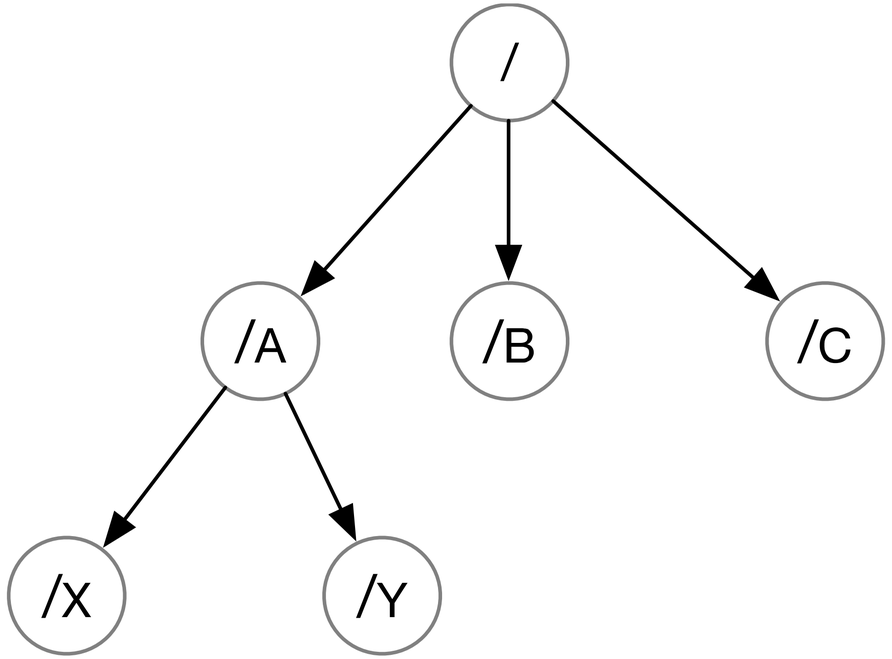
# ZooKeeper 数据模型

ZooKeeper 的数据模型是层次模型（Google Chubby 也是这么做的）。层次模型常见于文件系统。层次模型和 key-value 模型是两种主流的数据模型。ZooKeeper 使用文件系统模型主要基于以下两点考虑：

1. 文件系统的树形结构便于表达数据之间的层次关系。

2. 文件系统的树形结构便于为不同的应用分配独立的命名空间（namespace）。

ZooKeeper 的层次模型称作 data tree。Data tree 的每个节点叫作 znode。不同于文件系统，每个节点都可以保存数据。每个节点都有一个版本 (version)。版本从 0 开始计数。

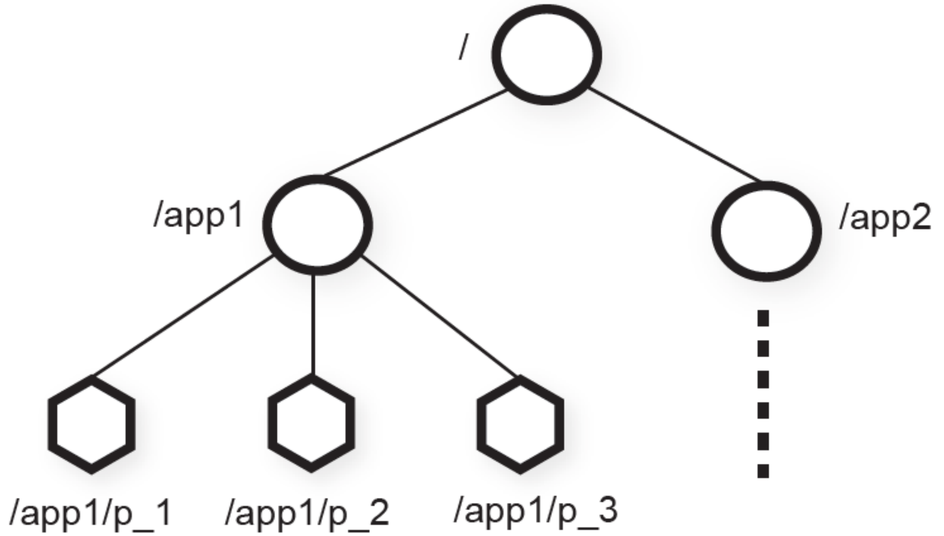



ZooKeeper层次数据模型data tree

# data tree 示例

在右图所示的 data tree 中有两个子树，一个用于应用1（/app1）和另一个用于应用2（/app2）。

应用1的子树实现了一个简单的组成员协议：每个客户端进程 $p_i$ 创建一个 znode p_i 在 / app1 下，只要 /app1/p_i 存在就代表进程 $p_i$ 在正常运行。



# data tree 接口

ZooKeeper 对外提供一个用来访问 data tree 的简化文件系统 API：

- 使用 UNIX 风格的路径名来定位 znode,例如 /A/X 表示 znode A 的子节点 X。
- znode 的数据只支持全量写入和读取，没有像通用文件系统那样支持部分写入和读取。
- data tree 的所有 API 都是 wait-free 的，正在执行中的 API 调用不会影响其他 API 的完成。
- data tree 的 API 都是对文件系统的 wait-free 操作，不直接提供锁这样的分布式协同机制。但是 data tree 的 API 非常强大，可以用来实现多种分布式协同机制。

# znode 分类

一个 znode 可以使持久性的，也可以是临时性的：

1. 持久性的 znode (PERSISTENT): ZooKeeper 宕机，或者 client 宕机，这个 znode 一旦创建就不会丢失。
2. 临时性的 znode (EPHEMERAL): ZooKeeper 宕机了，或者 client 在指定的 timeout 时间内没有连接 server，都会被认为丢失。

znode 节点也可以是顺序性的。每一个顺序性的 znode 关联一个唯一的单调递增整数。这个单调递增整数是 znode 名字的后缀。如果上面两种 znode 如果具备顺序性，又有以下两种 znode：

3. 持久顺序性的 znode(PERSISTENT_SEQUENTIAL): znode 除了具备持久性 znode 的特点之外，znode 的名字具备顺序性。
4. 临时顺序性的 znode(EPHEMERAL_SEQUENTIAL): znode 除了具备临时性 znode 的特点之外，znode 的名字具备顺序性。

ZooKeeper 主要有以上 4 种 znode。

# ZooKeeper 入门

# 安装 ZooKeeper

- 唯一的依赖是 JDK 7+。
- 到 https://zookeeper.apache.org/ 下载ZooKeeper，目前的最新版是 3.5.5。
- 把 apache-zookeeper-3.5.5-bin.tar.gz 解压到一个本地目录 (目录名最好不要包含空格和中文)。我使用 /Users/jing/tools 目录。
- 创建 conf/zoo.cfg。
- 配置以下环境变量。

```
export ZOOKEEPER_HOME="$HOME/tools/apache-zookeeper-3.5.5-bin"
export PATH="$ZOOKEEPER_HOME/bin:$PATH"
```

# 启动 ZooKeeper

- 使用 zkServer.sh start 启动 ZooKeeper 服务。
- 检查 ZooKeeper 日志是否有出错信息。
- 检查 ZooKeeper 数据文件。
- 检查 ZooKeeper 是否在 2181 端口上监听。

# zkCli 演示

打开 zkCli.sh，输入以下命令：

- `help`
- `ls -R /`
- `create /app1`
- `create /app2`
- `create /app1/p_1 1`
- `create /app1/p_2 2`
- `create /app1/p_3 3`
- `ls -R /`

# 实现一个锁

分布式锁要求如果锁的持有者宕了，锁可以被释放。ZooKeeper 的 ephemeral 节点恰好具备这样的特性。

终端1:

```
zkCli.sh

create -e /lock



quit
```

终端2:

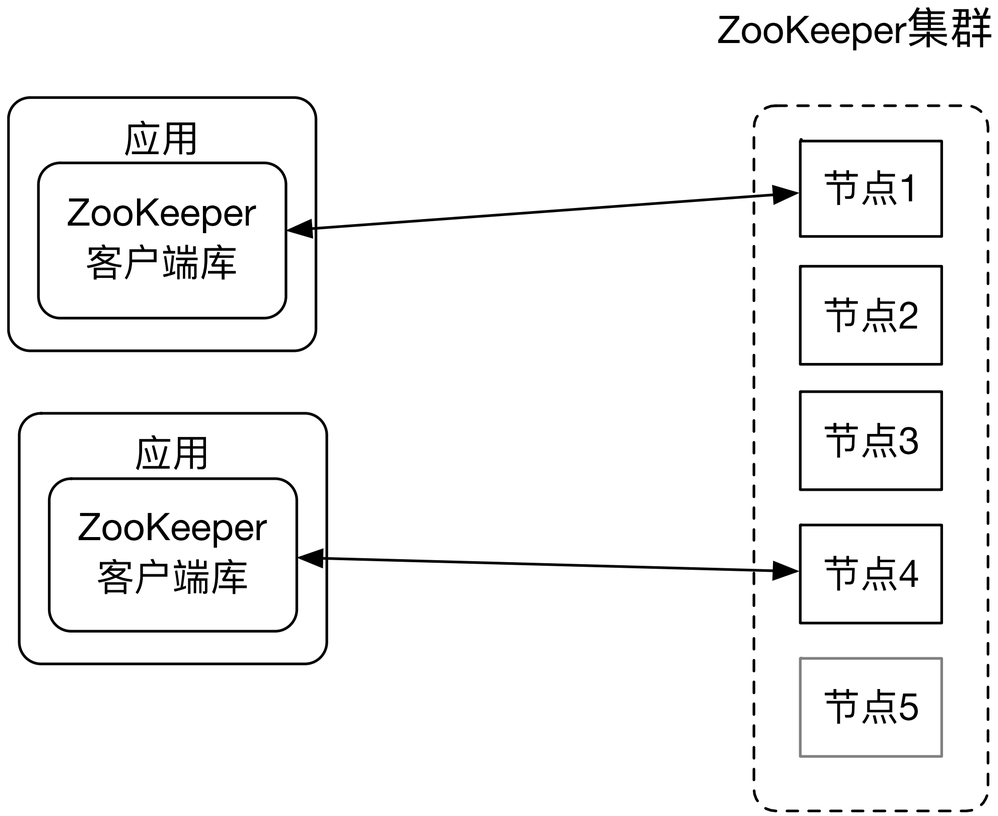
```
zkCli.sh


create -e /lock

stat -w /lock


create -e /lock
```

极客时间

# ZooKeeper 架构

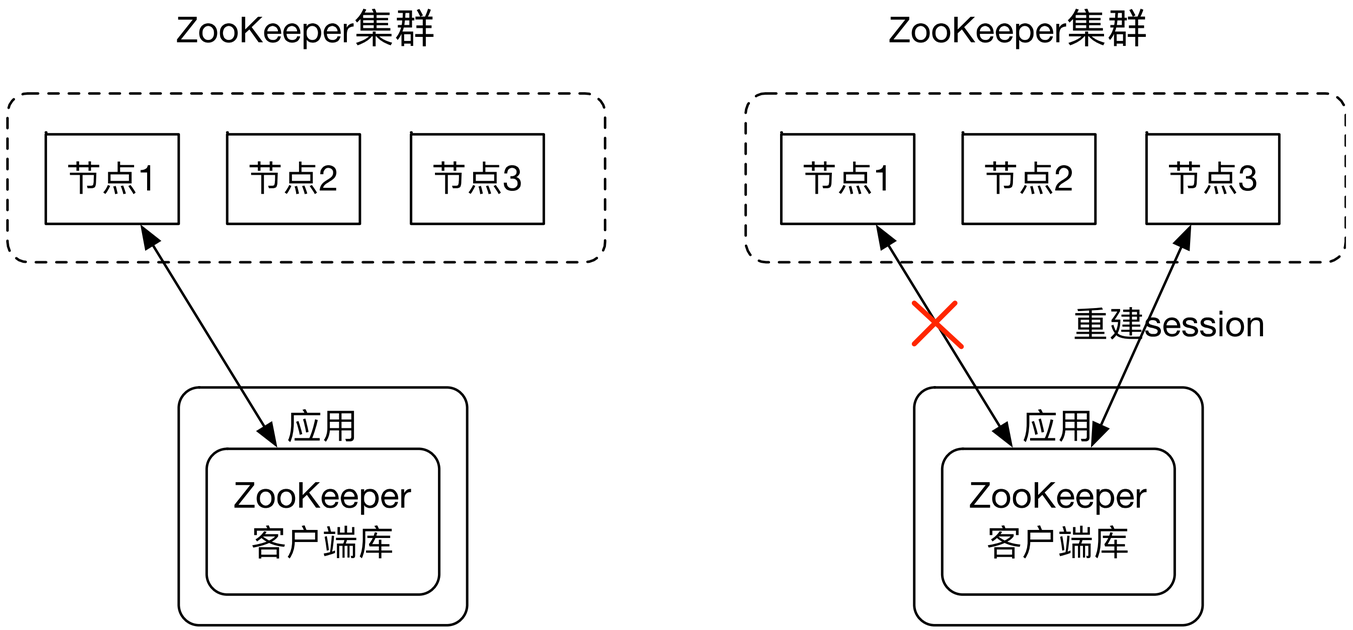
# ZooKeeper 总体架构

应用使用 ZooKeeper 客户端库使用 ZooKeeper 服务。 ZooKeeper 客户端负责和 ZooKeeper集群的交互。 ZooKeeper 集群可以有两种模式：standalone 模式和 quorum 模式。处于standalone 模式的 ZooKeeper 集群还有一个独立运行的 ZooKeeper 节点。处于 quorum 模式的 ZooKeeper 集群包换多个 ZooKeeper 节点。



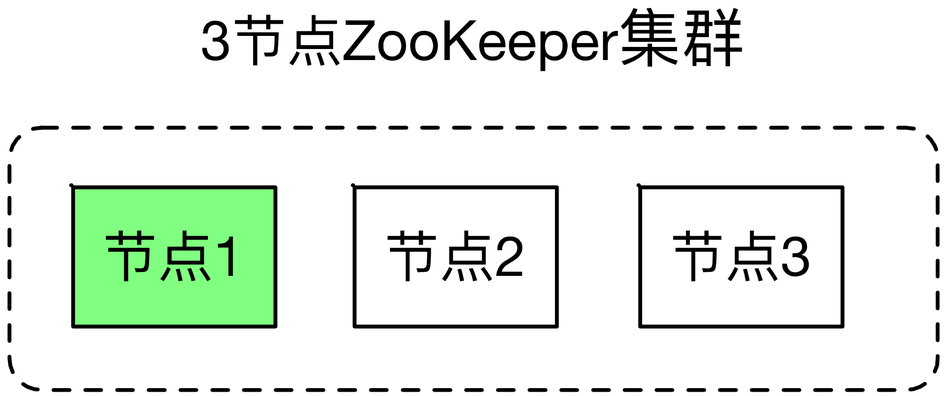
# Session

ZooKeeper 客户端库和 ZooKeeper 集群中的节点创建一个 session。客户端可以主动关闭 session。另外如果ZooKeeper 节点没有在 session 关联的 timeout 时间内收到客户端的数据的话， ZooKeeper 节点也会关闭session。另外 ZooKeeper 客户端库如果发现连接的 ZooKeeper 出错，会自动的和其他 ZooKeeper 节点建立连接。



# Quorum模式

处于 Quorum模式的 ZooKeeper 集群包含多个 ZooKeeper 节点。 下图的 ZooKeeper 集群有 3 个节点，其中节点 1 是 leader 节点，节点 2 和节点 3 是 follower 节点。 leader 节点可以处理读写请求，follower 只可以处理读请求。 follower 在接到写请求时会把写请求转发给leader来处理。



# 数据一致性

- 可线性化（Linearizable）写入：先到达 leader 的写请求会被先处理，leader 决定写请求的执行顺序。
- 客户端 FIFO 顺序：来自给定客户端的请求按照发送顺序执行。

# 3 节点 quorum 模式 ZooKeeper 集群演示

# 配置文件

需要准备 3 个配置文件，dataDir 和 clientPort 配置项要配置不同的值。3 个配置文件的 server.n 部分都是一样的。在 server.1=127.0.0.1:3333:3334，其中3333是 quorum之间的通信，3334 是用于 leader 选举的端口。

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=/data/zk/quorum/node1
# the port at which the clients will connect
clientPort=2181
# the maximum number of client connections.
# increase this if you need to handle more clients
#maxClientCnxns=60
#
# Be sure to read the maintenance section of the
# administrator guide before turning on autopurge.
#
# http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_maintenance
#
# The number of snapshots to retain in dataDir
#autopurge.snapRetainCount=3
# Purge task interval in hours
# Set to "0" to disable auto purge feature
#autopurge.purgeInterval=1

server.1=127.0.0.1:3333:3334
server.2=127.0.0.1:4444:4445
server.3=127.0.0.1:5555:5556
```

# 启动集群

使用以下命令启动集群：

```
zkServer.sh start-foreground src/main/resources/quorum/zoo-quorum-node1.cfg
zkServer.sh start-foreground src/main/resources/quorum/zoo-quorum-node2.cfg
zkServer.sh start-foreground src/main/resources/quorum/zoo-quorum-node3.cfg
```

start-foreground 选项 zkServer.sh 在前台运行，把日志直接打到 console。如果把日志打到文件的话，这三个 zkServer.sh 会把日志打到同一个文件。

# zkCli 演示

使用以下命令启动 zkCli:

```
zkCli.sh -server 127.0.0.1:2181,127.0.0.1:2182,127.0.0.1:2183
```

极客时间

扫码试看/订阅

《ZooKeeper实战与源码剖析》视频课程