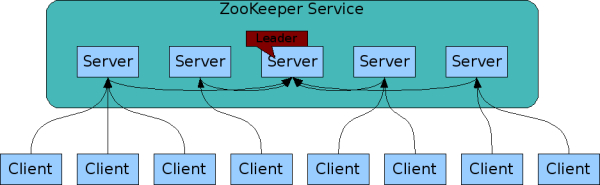
什么是Zookeeper？

* Zookeeper 是 Google 的 Chubby一个开源的实现，是 Hadoop 的分布式协调服务
* 它包含一个简单的原语集，分布式应用程序可以基于它实现同步服务-synchronized，配置维护-configuration和命名服务-naming等



Follower

Follower

Follower

Follower

为什么使用zookeeper?

* 大部分分布式应用需要一个主控、协调器或控制器来管理物理分布的子进程（如资源、任务分配等）
* 目前，大部分应用需要开发私有的协调程序，缺乏一个通用(me:也注定了zookeeper大而全)的机制
* 协调程序的反复编写浪费，且难以形成通用、伸缩性好的协调器
* ZooKeeper：提供通用的分布式锁服务，用以协调分布式应用

Zookeeper能帮我们做什么?（配置管理 集群管理 分布式锁）

* Hadoop2.0,使用Zookeeper的事件处理确保整个集群只有一个活跃的NameNode,存储配置信息等.
* HBase,使用Zookeeper的事件处理确保整个集群只有一个HMaster,察觉HRegionServer联机和宕机,存储访问控制列表等.

Zookeeper的特性

* **Zookeeper是简单的**
* **Zookeeper是富有表现力的**
* **Zookeeper具有高可用性**
* **Zookeeper采用松耦合交互方式**
* **Zookeeper是一个资源库(me:类似数据库，存储的是程序或集群运行阶段的资源)**

Zookeeper的安装和配置（单机模式）

* 下载ZooKeeper：<http://labs.renren.com/apache-mirror/zookeeper/zookeeper-3.4.3/zookeeper-3.4.3.tar.gz>
* 解压：tar xzf [zookeeper-3.4.3.tar.gz](http://labs.renren.com/apache-mirror/zookeeper/zookeeper-3.4.3/zookeeper-3.4.3.tar.gz)
* 在conf目录下创建一个配置文件zoo.cfg，

tickTime=2000 #zookeeper的时间单位，毫秒，之后的时间配置以这里的配置为单位  
dataDir=/Users/zdandljb/zookeeper/data #数据目录，zookeeper保存资源的目录  
dataLogDir=/Users/zdandljb/zookeeper/dataLog #配置日志路径，以便zookeeper作为后台服务运行时出现故障的排错

clientPort=2181 #客户端连接端口

* 启动ZooKeeper的Server：sh bin/zkServer.sh start, 如果想要关闭，输入：zkServer.sh stop

Zookeeper的安装和配置（集群模式）

* 在data目录创建myid文件：

server1机器的内容为：1，server2机器的内容为：2，server3机器的内容为：3，不一定为1，2,3，可以为任意的数字，主要不同而且与配置文件中配置的一样即可

* 在conf目录下创建一个配置文件zoo.cfg，
* tickTime=2000  
  dataDir=/Users/zdandljb/zookeeper/data  
  dataLogDir=/Users/zdandljb/zookeeper/dataLog

clientPort=2181

initLimit=5 #初始化时间

syncLimit=2 #同步时间

server.1=server1:2888:3888 #1即为上面在data目录下myid中的数值 server.2=server2:2888:3888 server.3=server3:2888:3888

第一个端口为zookeeper之间通信的端口，第二个端口为选举leader端口

Zookeeper的安装和配置（伪集群模式，在一台机器上通过端口的配置来搭建）

* 建立3个文件夹，**server1 server2 server3，**然后每个文件夹里面解压一个zookeeper的下载包
* 进入data目录，创建一个myid的文件，里面写入一个数字，server1,就写一个1，server2对应myid文件就写入2，server3对应myid文件就写个3
* 在conf目录下创建一个配置文件zoo.cfg，

tickTime=2000  
dataDir=/Users/zdandljb/zookeeper/data  
dataLogDir=xxx/zookeeper/server1/

clientPort=2181

initLimit=5

syncLimit=2

server.1=server1:2888:3888 server.2=server1:2889:3889 server.3=server1:2890:3890

Zookeeper的数据模型

* 层次化的目录结构，命名符合常规文件系统规范
* 每个数据节点在zookeeper中叫做znode,并且其有一个唯一的路径标识
* 节点Znode可以包含数据和子节点，但是EPHEMERAL类型的节点不能有子节点
* Znode中的数据可以有多个版本，比如某一个路径下存有多个数据版本，那么查询这个路径下的数据就需要带上版本
* 客户端应用可以在节点上设置监视器
* 节点不支持部分读写，而是一次性完整读写

Zookeeper的节点

* Znode有两种类型，短暂的（ephemeral）和持久的（persistent）
* Znode的类型在创建时确定并且之后不能再修改
* 短暂znode的客户端会话结束时，zookeeper会将该短暂znode删除，短暂znode不可以有子节点
* 持久znode不依赖于客户端会话，只有当客户端明确要删除该持久znode时才会被删除
* Znode有四种形式的目录节点，PERSISTENT、PERSISTENT\_SEQUENTIAL、EPHEMERAL、EPHEMERAL\_SEQUENTIAL

Zookeeper的角色

* 领导者（leader），负责进行投票的发起和决议，更新系统状态
* 学习者（learner），包括跟随者（follower）和观察者（observer），follower用于接受客户端请求并向客户端返回结果，在选主过程中参与投票
* Observer可以接受客户端连接，将写请求转发给leader，但observer不参加投票过程，只同步leader的状态，observer的目的是为了扩展系统，提高读取速度
* 客户端（client），请求发起方



Zookeeper的顺序号

* 创建znode时设置顺序标识，znode名称后会附加一个值
* 顺序号是一个单调递增的计数器，由父节点维护
* 在分布式系统中，顺序号可以被用于为所有的事件进行全局排序，这样客户端可以通过顺序号推断事件的顺序

Zookeeper的读写机制

* Zookeeper是一个由多个server组成的集群
* 一个leader，多个follower和observer
* 每个server保存一份数据副本
* 全局数据一致
* 分布式读写
* 更新请求转发，由leader实施

Zookeeper的保证

* 更新请求顺序进行，来自同一个client的更新请求按其发送顺序依次执行
* 数据更新原子性，一次数据更新要么成功，要么失败
* 全局唯一数据视图，client无论连接到哪个server，数据视图都是一致的
* 实时性，在一定事件范围内，client能读到最新数据

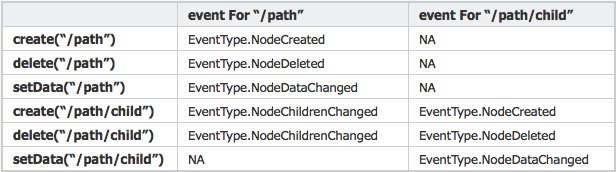
Zookeeper的API接口

* [String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) [create](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html%23create(java.lang.String,%20byte%5B%5D,%20java.util.List,%20org.apache.zookeeper.CreateMode))([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, byte[] data, [List](http://java.sun.com/javase/6/docs/api/java/util/List.html?is-external=true)<[ACL](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/ACL.html)> acl, [CreateMode](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/CreateMode.html) createMode)
* [Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html) [exists](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html%23exists(java.lang.String,%20boolean))([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, boolean watch)
* void [delete](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html%23delete(java.lang.String,%20int))([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, int version)
* [List](http://java.sun.com/javase/6/docs/api/java/util/List.html?is-external=true)<[String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true)> [getChildren](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html%23getChildren(java.lang.String,%20boolean))([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, boolean watch)
* [List](http://java.sun.com/javase/6/docs/api/java/util/List.html?is-external=true)<[String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true)> [getChildren](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html%23getChildren(java.lang.String,%20boolean))([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, boolean watch)
* [Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html) [setData](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html%23setData(java.lang.String,%20byte%5B%5D,%20int))([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, byte[] data, int version)
* byte[] [getData](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html%23getData(java.lang.String,%20boolean,%20org.apache.zookeeper.data.Stat))([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, boolean watch, [Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html) stat)
* void [addAuthInfo](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html%23addAuthInfo(java.lang.String,%20byte%5B%5D))([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) scheme, byte[] auth)
* [Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html) [setACL](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html%23setACL(java.lang.String,%20java.util.List,%20int))([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, [List](http://java.sun.com/javase/6/docs/api/java/util/List.html?is-external=true)<[ACL](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/ACL.html)> acl, int version)
* [List](http://java.sun.com/javase/6/docs/api/java/util/List.html?is-external=true)<[ACL](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/ACL.html)> [getACL](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html%23getACL(java.lang.String,%20org.apache.zookeeper.data.Stat))([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, [Stat](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/data/Stat.html) stat)

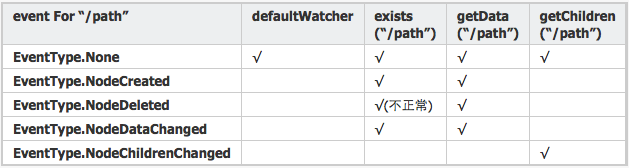
观察（watcher）

* Watcher 在 ZooKeeper 是一个核心功能，Watcher 可以监控目录节点的数据变化以及子目录的变化，一旦这些状态发生变化，服务器就会通知所有设置在这个目录节点上的 Watcher，从而每个客户端都很快知道它所关注的目录节点的状态发生变化，而做出相应的反应
* 可以设置观察的操作（me：类似查询的操作）：exists,getChildren,getData
* 可以触发观察的操作（me:类似更新的操作）：create,delete,setData

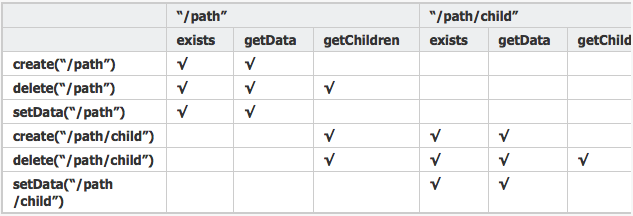
写操作与zookeeper内部事件的对应关系



zookeeper内部事件与watcher的对应关系



写操作与watcher的对应关系（由以上两个表综合而来）



* 每个znode被创建时都会带有一个ACL列表（list），用于决定谁可以对它执行何种操作

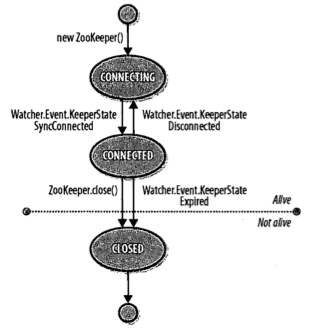


ACL

* 身份验证模式有三种：
* digest:用户名，密码
* host:通过客户端的主机名来识别客户端
* ip： 通过客户端的ip来识别客户端
* new ACL(Perms.READ,new Id("host","example.com"));

这个ACL对应的身份验证模式是host，符合该模式的身份是example.com，权限的组合是：READ

Znode的节点状态



Zookeeper工作原理

* Zookeeper的核心是原子广播，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协议有两种模式，它们分别是恢复模式和广播模式。当服务启动或者在领导者崩溃后，Zab就进入了恢复模式，当领导者被选举出来，且大多数server的完成了和leader的状态同步以后，恢复模式就结束了。状态同步保证了leader和server具有相同的系统状态。
* 广播模式需要保证proposal被按顺序处理，因此zk采用了递增的事务id号(zxid)来保证。所有的提议(proposal)都在被提出的时候加上了zxid。实现中zxid是一个64为的数字，它高32位是epoch用来标识leader关系是否改变，每次一个leader被选出来，它都会有一个新的epoch。低32位是个递增计数。
* 当leader崩溃或者leader失去大多数的follower，这时候zk进入恢复模式，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。

30