Hadoop深入浅出

Zookeeper

* 概述
* 安装
* 结构与原理
* 应用

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，配置维护和命名服务等



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？

* 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**

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  
  dataDir=/Users/zdandljb/zookeeper/data  
  dataLogDir=/Users/zdandljb/zookeeper/dataLog clientPort=2181
* 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）

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

dataLogDir=/Users/zdandljb/zookeeper/dataLog

clientPort=2181 #端口号

initLimit=5

syncLimit=2

server.1=server1:2888:3888 server.2=server2:2888:3888 server.3=server3:2888:3888

Zookeeper的安装和配置（伪集群模式）

* 建了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

Zookeeper的安装和配置（伪集群模式）

* 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 n
* Znode有两种类型，临时的（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
* 临时znode的客户端会话结束时，zookeeper会将该临时znode删除，临时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
* 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 （写全部是由leader来执行，然后其他的节点从leader同步）

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，在一定时间范围内，client能读到最新数据

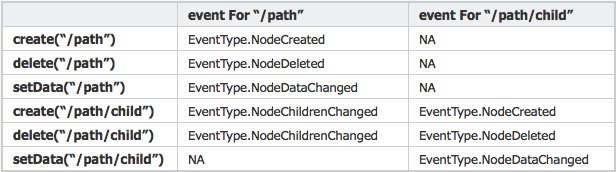
Zookeeper的API接口

* [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)

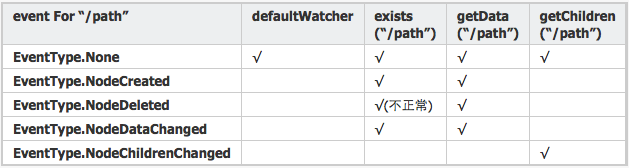
观察（watcher）

* 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：exists,getChildren,getData
* 可以触发观察的操作：create,delete,setData

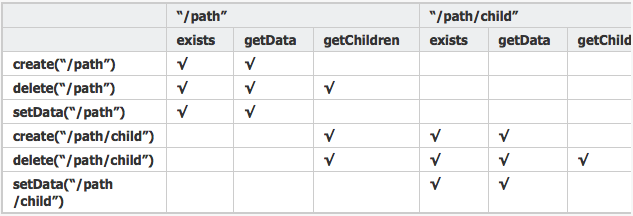
写操作与zookeeper内部事件的对应关系



zookeeper内部事件与watcher的对应关系



写操作与watcher的对应关系



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

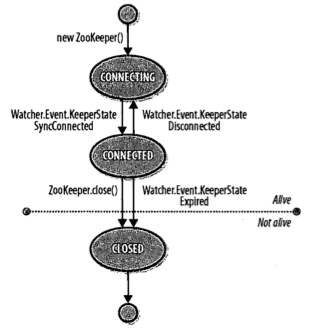


ACL

* 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。

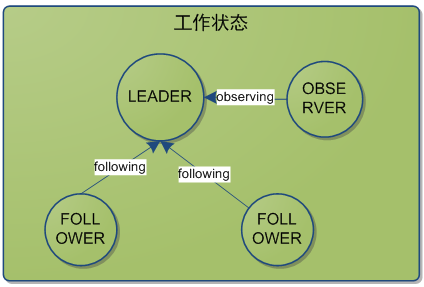
Zookeeper工作原理

* 一旦leader已经和多数的follower进行了状态同步后，他就可以开始广播消息了，即进入广播状态。这时候当一个server加入zookeeper服务中，它会在恢复模式下启动，发现leader，并和leader进行状态同步。待到同步结束，它也参与消息广播。Zookeeper服务一直维持在Broadcast状态，直到leader崩溃了或者leader失去了大部分的followers支持。
* 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。

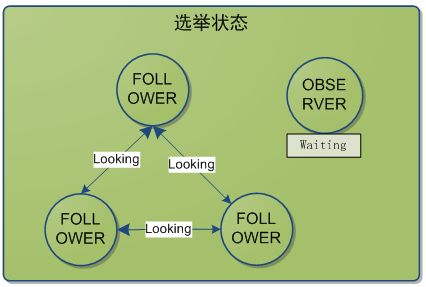
Leader选举

* 每个Server启动以后都询问其它的Server它要投票给谁。
* 对于其他server的询问，server每次根据自己的状态都回复自己推荐的leader的id和上一次处理事务的zxid（系统启动时每个server都会推荐自己）
* 收到所有Server回复以后，就计算出zxid最大的哪个Server，并将这个Server相关信息设置成下一次要投票的Server。
* 计算这过程中获得票数最多的的sever为获胜者，如果获胜者的票数超过半数，则该server被选为leader。否则，继续这个过程，直到leader被选举出来。
* leader就会开始等待server连接
* Follower连接leader，将最大的zxid发送给leader
* Leader根据follower的zxid确定同步点
* 完成同步后通知follower 已经成为uptodate状态
* Follower收到uptodate消息后，又可以重新接受client的请求进行服务了

Leader选举



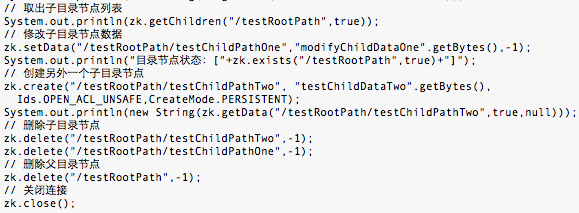
Leader选举



Zookeeper示例代码



Zookeeper示例代码

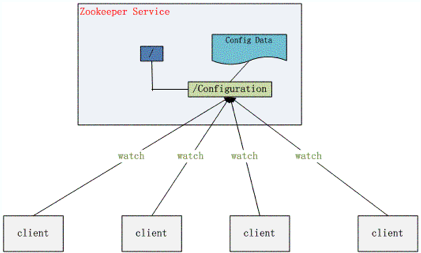


应用场景1－统一命名服务

* 分布式应用中，通常需要有一套完整的命名规则，既能够产生唯一的名称又便于人识别和记住，通常情况下用树形的名称结构是一个理想的选择，树形的名称结构是一个有层次的目录结构，既对人友好又不会重复。
* Name Service 是 Zookeeper 内置的功能，只要调用 Zookeeper 的 API 就能实现

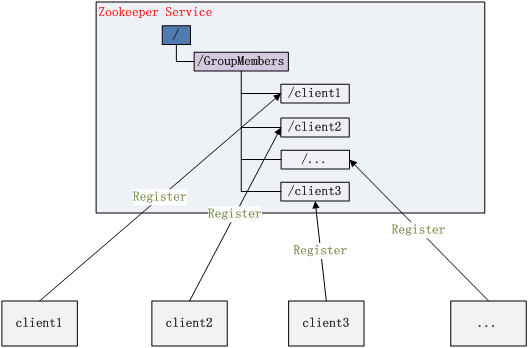
应用场景2－配置管理

* 配置的管理在分布式应用环境中很常见，例如同一个应用系统需要多台 PC Server 运行，但是它们运行的应用系统的某些配置项是相同的，如果要修改这些相同的配置项，那么就必须同时修改每台运行这个应用系统的 PC Server，这样非常麻烦而且容易出错。
* 将配置信息保存在 Zookeeper 的某个目录节点中，然后将所有需要修改的应用机器监控配置信息的状态，一旦配置信息发生变化，每台应用机器就会收到 Zookeeper 的通知，然后从 Zookeeper 获取新的配置信息应用到系统中。



应用场景3－集群管理

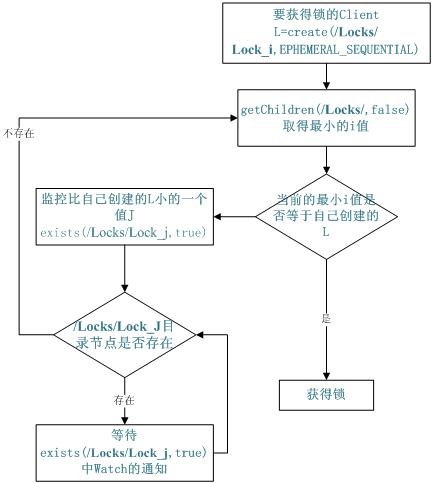
* Zookeeper 能够很容易的实现集群管理的功能，如有多台 Server 组成一个服务集群，那么必须要一个“总管”知道当前集群中每台机器的服务状态，一旦有机器不能提供服务，集群中其它集群必须知道，从而做出调整重新分配服务策略。同样当增加集群的服务能力时，就会增加一台或多台 Server，同样也必须让“总管”知道。
* Zookeeper 不仅能够维护当前的集群中机器的服务状态，而且能够选出一个“总管”，让这个总管来管理集群，这就是 Zookeeper 的另一个功能 Leader Election。



* zk.create("/testRootPath/testChildPath1","1".getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL\_SEQUENTIAL);
* zk.create(“/testRootPath/testChildPath2”,“2”.getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL\_SEQUENTIAL);
* zk.create("/testRootPath/testChildPath3","3".getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL\_SEQUENTIAL);
* zk.create("/testRootPath/testChildPath4","4".getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL\_SEQUENTIAL);
* System.out.println(zk.getChildren("/testRootPath", false));
* 规定编号最小的为master,所以当我们对SERVERS节点做监控的时候，得到服务器列表，只要所有集群机器逻辑认为最小编号节点为master，那么master就被选出，而这个master宕机的时候，相应的znode会消失，然后新的服务器列表就被推送到客户端，然后每个节点逻辑认为最小编号节点为master，这样就做到动态master选举。

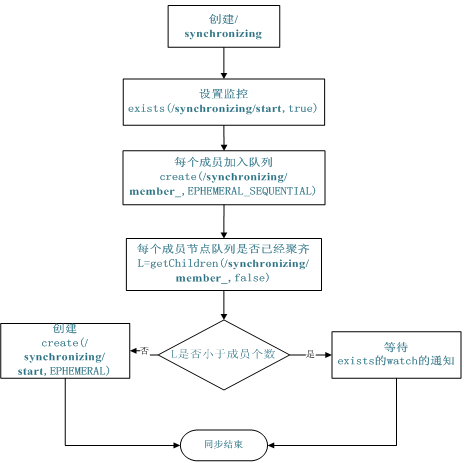
应用场景4－共享锁

* 共享锁在同一个进程中很容易实现，但是在跨进程或者在不同 Server 之间就不好实现了。Zookeeper 却很容易实现这个功能，实现方式也是需要获得锁的 Server 创建一个 EPHEMERAL\_SEQUENTIAL 目录节点，然后调用 getChildren方法获取当前的目录节点列表中最小的目录节点是不是就是自己创建的目录节点，如果正是自己创建的，那么它就获得了这个锁，如果不是那么它就调用 exists(String path, boolean watch) 方法并监控 Zookeeper 上目录节点列表的变化，一直到自己创建的节点是列表中最小编号的目录节点，从而获得锁，释放锁很简单，只要删除前面它自己所创建的目录节点就行了。



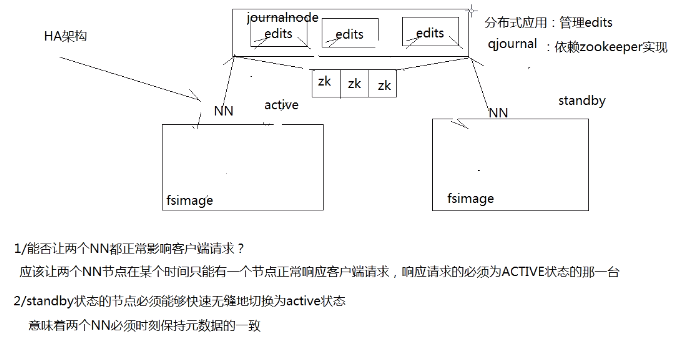
应用场景5－队列管理

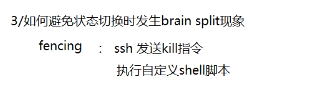
* Zookeeper 可以处理两种类型的队列：
* 1. 同步队列：当一个队列的成员都聚齐时，这个队列才可用，否则一直等待所有成员到达，这种是同步队列；
* 2. 异步队列：队列按照 FIFO 方式进行入队和出队操作，例如实现生产者和消费者模型
* 创建一个父目录 /synchronizing，每个成员都监控目录 /synchronizing/start 是否存在，然后每个成员都加入这个队列（创建 /synchronizing/member\_i 的临时目录节点），然后每个成员获取 / synchronizing 目录的所有目录节点，判断 i 的值是否已经是成员的个数，如果小于成员个数等待 /synchronizing/start 的出现，如果已经相等就创建 /synchronizing/start。



总结

* Zookeeper 作为 Hadoop 项目中的一个子项目，是 Hadoop 集群管理的一个必不可少的模块，它主要用来控制集群中的数据，如它管理 Hadoop 集群中的 NameNode，还有 Hbase 中 Master Election、Server 之间状态同步等。
* Zoopkeeper 提供了一套很好的分布式集群管理的机制，就是它这种基于层次型的目录树的数据结构，并对树中的节点进行有效管理，从而可以设计出多种多样的分布式的数据管理模型





//=================20171027