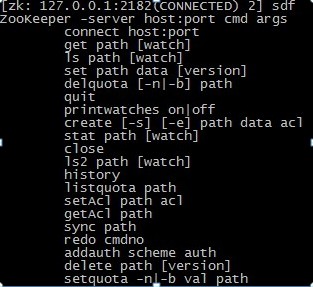
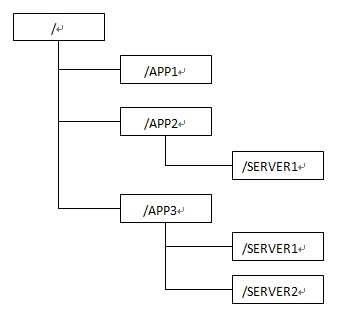
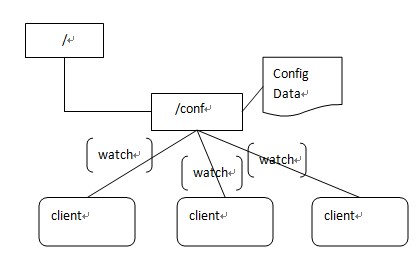
**zookeeper使用和原理探究（一）**

**zookeeper介绍**zookeeper是一个为分布式应用提供一致性服务的软件，它是开源的Hadoop项目中的一个子项目，并且根据google发表的<The Chubby lock service for loosely-coupled distributed systems>论文来实现的，接下来我们首先来安装使用下这个软件，然后再来探索下其中比较重要一致性算法。    
  
**zookeeper安装和使用**zookeeper的安装基本上可以按照 http://hadoop.apache.org/zookeeper/docs/current/ zookeeperStarted.html 这个页面上的步骤完成安装，这里主要介绍下部署一个集群的步骤，因为这个官方页面似乎讲得并不是非常详细(Running Replicated Zookeeper)。  
  
由于手头机器不足，所以在一台机器上部署了3个server,如果你手头也比较紧，也可以这么做。那么我建了3个文件夹，如下  
**server1   server2   server3**  
然后每个文件夹里面解压一个zookeeper的下载包，并且还建了几个文件夹，总体结构如下,最后那个是下载过来压缩包的解压文件  
**data dataLog logs zookeeper-3.3.2**那么首先进入data目录，创建一个myid的文件，里面写入一个数字，比如我这个是server1,那么就写一个1，server2对应myid文件就写入2，server3对应myid文件就写个3  
  
然后进入**zookeeper-3.3.2/conf**目录，那么如果是刚下过来，会有3个文件，**configuration.xml, log4j.properties,zoo\_sample.cfg**,这3个文件我们首先要做的就是在这个目录创建一个zoo.cfg的配置文件，当然你可以把zoo\_sample.cfg文件改成zoo.cfg，配置的内容如下所示：   
tickTime=2000  
initLimit=5  
syncLimit=2  
dataDir=xxxx/zookeeper/server1/data  
dataLogDir=xxx/zookeeper/server1/dataLog  
clientPort=2181  
server.1=127.0.0.1:2888:3888  
server.2=127.0.0.1:2889:3889  
server.3=127.0.0.1:2890:3890  
  
标红的几个配置应该官网讲得很清楚了，只是需要注意的是clientPort这个端口如果你是在1台机器上部署多个server,那么每台机器都要不同的clientPort，比如我server1是2181,server2是2182，server3是2183，dataDir和dataLogDir也需要区分下。   
  
最后几行唯一需要注意的地方就是 server.X 这个数字就是对应 data/myid中的数字。你在3个server的myid文件中分别写入了1，2，3，那么每个server中的zoo.cfg都配server.1,server.2,server.3就OK了。因为在同一台机器上，后面连着的2个端口3个server都不要一样，否则端口冲突，其中第一个端口用来集群成员的信息交换，第二个端口是在leader挂掉时专门用来进行选举leader所用。  
  
进入zookeeper-3.3.2/bin 目录中，**./zkServer.sh start**启动一个server,这时会报大量错误？其实没什么关系，因为现在集群只起了1台server，zookeeper服务器端起来会根据zoo.cfg的服务器列表发起选举leader的请求，因为连不上其他机器而报错，那么当我们起第二个zookeeper实例后，leader将会被选出，从而一致性服务开始可以使用，这是因为3台机器只要有2台可用就可以选出leader并且对外提供服务(2n+1台机器，可以容n台机器挂掉)。  
  
接下来就可以使用了，我们可以先通过 zookeeper自带的客户端交互程序来简单感受下zookeeper到底做一些什么事情。进入zookeeper-3.3.2/bin（3个server中任意一个）下，**./zkCli.sh –server 127.0.0.1:2182**,我连的是开着2182端口的机器。  
  
那么，首先我们随便打个命令，因为zookeeper不认识，他会给出命令的help,如下图    
     
ls(查看当前节点数据),  
ls2(查看当前节点数据并能看到更新次数等数据) ,  
create(创建一个节点) ,  
get(得到一个节点，包含数据和更新次数等数据),  
set(修改节点)  
delete(删除一个节点)  
  
通过上述命令实践，我们可以发现，zookeeper使用了一个类似文件系统的树结构，数据可以挂在某个节点上，可以对这个节点进行删改。另外我们还发现，当改动一个节点的时候，集群中活着的机器都会更新到一致的数据。   
  
**zookeeper的数据模型**在简单使用了zookeeper之后，我们发现其数据模型有些像操作系统的文件结构，结构如下图所示  
  
  
  
(1)     每个节点在zookeeper中叫做znode,并且其有一个唯一的路径标识，如/SERVER2节点的标识就为/APP3/SERVER2  
(2)     Znode可以有子znode，并且znode里可以存数据，但是EPHEMERAL类型的节点不能有子节点  
(3)     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。  
(4)     znode 可以是临时节点，一旦创建这个 znode 的客户端与服务器失去联系，这个 znode 也将自动删除，Zookeeper 的客户端和服务器通信采用长连接方式，每个客户端和  服务器通过心跳来保持连接，这个连接状态称为 session，如果 znode 是临时节点，这个 session 失效，znode 也就删除了  
(5)     znode 的目录名可以自动编号，如 App1 已经存在，再创建的话，将会自动命名为 App2   
(6)     znode 可以被监控，包括这个目录节点中存储的数据的修改，子节点目录的变化等，一旦变化可以通知设置监控的客户端，这个功能是zookeeper对于应用最重要的特性，通过这个特性可以实现的功能包括配置的集中管理，集群管理，分布式锁等等。    
  
**通过java代码使用zookeeper**Zookeeper的使用主要是通过创建其jar包下的Zookeeper实例，并且调用其接口方法进行的，主要的操作就是对znode的增删改操作，监听znode的变化以及处理。   
  
以下为主要的API使用和解释

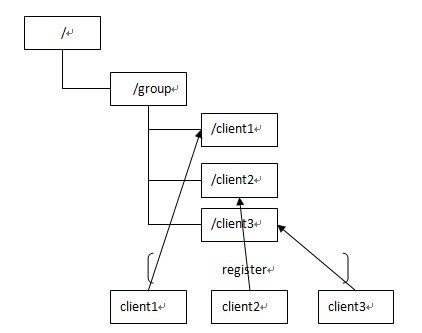
http://www.blogjava.net/Images/OutliningIndicators/None.gif//创建一个Zookeeper实例，第一个参数为目标服务器地址和端口，第二个参数为Session超时时间，第三个为节点变化时的回调方法  
http://www.blogjava.net/Images/OutliningIndicators/ExpandedBlockStart.gifZooKeeper zk = new ZooKeeper("127.0.0.1:2181", 500000,new Watcher() {  
http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif           // 监控所有被触发的事件  
http://www.blogjava.net/Images/OutliningIndicators/ExpandedSubBlockStart.gif             public void process(WatchedEvent event) {  
http://www.blogjava.net/Images/OutliningIndicators/InBlock.gif           //dosomething  
http://www.blogjava.net/Images/OutliningIndicators/ExpandedSubBlockEnd.gif           }  
http://www.blogjava.net/Images/OutliningIndicators/ExpandedBlockEnd.gif      });  
http://www.blogjava.net/Images/OutliningIndicators/None.gif//创建一个节点root，数据是mydata,不进行ACL权限控制，节点为永久性的(即客户端shutdown了也不会消失)  
http://www.blogjava.net/Images/OutliningIndicators/None.gifzk.create("/root", "mydata".getBytes(),Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT);  
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif//在root下面创建一个childone znode,数据为childone,不进行ACL权限控制，节点为永久性的  
http://www.blogjava.net/Images/OutliningIndicators/None.gifzk.create("/root/childone","childone".getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.PERSISTENT);  
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif//取得/root节点下的子节点名称,返回List<String>  
http://www.blogjava.net/Images/OutliningIndicators/None.gifzk.getChildren("/root",true);  
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif//取得/root/childone节点下的数据,返回byte[]  
http://www.blogjava.net/Images/OutliningIndicators/None.gifzk.getData("/root/childone", true, null);  
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif//修改节点/root/childone下的数据，第三个参数为版本，如果是-1，那会无视被修改的数据版本，直接改掉  
http://www.blogjava.net/Images/OutliningIndicators/None.gifzk.setData("/root/childone","childonemodify".getBytes(), -1);  
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif//删除/root/childone这个节点，第二个参数为版本，－1的话直接删除，无视版本  
http://www.blogjava.net/Images/OutliningIndicators/None.gifzk.delete("/root/childone", -1);  
http://www.blogjava.net/Images/OutliningIndicators/None.gif        
http://www.blogjava.net/Images/OutliningIndicators/None.gif//关闭session  
http://www.blogjava.net/Images/OutliningIndicators/None.gifzk.close();  
http://www.blogjava.net/Images/OutliningIndicators/None.gif

**Zookeeper的主流应用场景实现思路（除去官方示例）   
  
(1)配置管理**集中式的配置管理在应用集群中是非常常见的，一般商业公司内部都会实现一套集中的配置管理中心，应对不同的应用集群对于共享各自配置的需求，并且在配置变更时能够通知到集群中的每一个机器。  
  
Zookeeper很容易实现这种集中式的配置管理，比如将APP1的所有配置配置到/APP1 znode下，APP1所有机器一启动就对/APP1这个节点进行监控(zk.exist("/APP1",true)),并且实现回调方法Watcher，那么在zookeeper上/APP1 znode节点下数据发生变化的时候，每个机器都会收到通知，Watcher方法将会被执行，那么应用再取下数据即可(zk.getData("/APP1",false,null));  
  
以上这个例子只是简单的粗颗粒度配置监控，细颗粒度的数据可以进行分层级监控，这一切都是可以设计和控制的。       
**(2)集群管理**应用集群中，我们常常需要让每一个机器知道集群中（或依赖的其他某一个集群）哪些机器是活着的，并且在集群机器因为宕机，网络断链等原因能够不在人工介入的情况下迅速通知到每一个机器。  
  
Zookeeper同样很容易实现这个功能，比如我在zookeeper服务器端有一个znode叫/APP1SERVERS,那么集群中每一个机器启动的时候都去这个节点下创建一个EPHEMERAL类型的节点，比如server1创建/APP1SERVERS/SERVER1(可以使用ip,保证不重复)，server2创建/APP1SERVERS/SERVER2，然后SERVER1和SERVER2都watch /APP1SERVERS这个父节点，那么也就是这个父节点下数据或者子节点变化都会通知对该节点进行watch的客户端。因为EPHEMERAL类型节点有一个很重要的特性，就是客户端和服务器端连接断掉或者session过期就会使节点消失，那么在某一个机器挂掉或者断链的时候，其对应的节点就会消失，然后集群中所有对/APP1SERVERS进行watch的客户端都会收到通知，然后取得最新列表即可。  
  
另外有一个应用场景就是集群选master,一旦master挂掉能够马上能从slave中选出一个master,实现步骤和前者一样，只是机器在启动的时候在APP1SERVERS创建的节点类型变为EPHEMERAL\_SEQUENTIAL类型，这样每个节点会自动被编号，例如

http://www.blogjava.net/Images/OutliningIndicators/None.gifzk.create("/testRootPath/testChildPath1","1".getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL\_SEQUENTIAL);  
http://www.blogjava.net/Images/OutliningIndicators/None.gif          
http://www.blogjava.net/Images/OutliningIndicators/None.gifzk.create("/testRootPath/testChildPath2","2".getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL\_SEQUENTIAL);  
http://www.blogjava.net/Images/OutliningIndicators/None.gif          
http://www.blogjava.net/Images/OutliningIndicators/None.gifzk.create("/testRootPath/testChildPath3","3".getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL\_SEQUENTIAL);  
http://www.blogjava.net/Images/OutliningIndicators/None.gif          
http://www.blogjava.net/Images/OutliningIndicators/None.gif// 创建一个子目录节点  
http://www.blogjava.net/Images/OutliningIndicators/None.gifzk.create("/testRootPath/testChildPath4","4".getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL\_SEQUENTIAL);  
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gifSystem.out.println(zk.getChildren("/testRootPath", false));  
http://www.blogjava.net/Images/OutliningIndicators/None.gif

 打印结果：[testChildPath10000000000, testChildPath20000000001, testChildPath40000000003, testChildPath30000000002]

http://www.blogjava.net/Images/OutliningIndicators/None.gifzk.create("/testRootPath", "testRootData".getBytes(),Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT);  
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif// 创建一个子目录节点  
http://www.blogjava.net/Images/OutliningIndicators/None.gifzk.create("/testRootPath/testChildPath1","1".getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL);  
http://www.blogjava.net/Images/OutliningIndicators/None.gif          
http://www.blogjava.net/Images/OutliningIndicators/None.gifzk.create("/testRootPath/testChildPath2","2".getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL);  
http://www.blogjava.net/Images/OutliningIndicators/None.gif          
http://www.blogjava.net/Images/OutliningIndicators/None.gifzk.create("/testRootPath/testChildPath3","3".getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL);  
http://www.blogjava.net/Images/OutliningIndicators/None.gif          
http://www.blogjava.net/Images/OutliningIndicators/None.gif// 创建一个子目录节点  
http://www.blogjava.net/Images/OutliningIndicators/None.gifzk.create("/testRootPath/testChildPath4","4".getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL);  
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gifSystem.out.println(zk.getChildren("/testRootPath", false));  
http://www.blogjava.net/Images/OutliningIndicators/None.gif

打印结果:[testChildPath2, testChildPath1, testChildPath4, testChildPath3]  
  
我们默认规定编号最小的为master,所以当我们对/APP1SERVERS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。  


zookeeper 里面有个jetty控制台：所以每个集群最好改下 他的节点

zookeeper最近的版本中有个内嵌的管理控制台是通过jetty启动，也会占用8080 端口。  
通过查看zookeeper的官方文档，发现有3种解决途径：  
（1）.删除jetty。  
（2）修改端口。  
修改方法的方法有两种，一种是在启动脚本中增加 -Dzookeeper.admin.serverPort=你的端口号.一种是在zoo.cfg中增加admin.serverPort=没有被占用的端口号  
（3）停用这个服务，在启动脚本中增加"-Dzookeeper.admin.enableServer=false"  
推荐第二种。以上答案来自林氏120技术中心。

数据模型

Zookeeper 会维护一个具有层次关系的数据结构，它非常类似于一个标准的文件系统，如图 1 所示：

图 1 Zookeeper 数据结构



Zookeeper 这种数据结构有如下这些特点：

1. 每个子目录项如 NameService 都被称作为 znode，这个 znode 是被它所在的路径唯一标识，如 Server1 这个 znode 的标识为 /NameService/Server1
2. znode 可以有子节点目录，并且每个 znode 可以存储数据，注意 EPHEMERAL 类型的目录节点不能有子节点目录
3. znode 是有版本的，每个 znode 中存储的数据可以有多个版本，也就是一个访问路径中可以存储多份数据
4. znode 可以是临时节点，一旦创建这个 znode 的客户端与服务器失去联系，这个 znode 也将自动删除，Zookeeper 的客户端和服务器通信采用长连接方式，每个客户端和服务器通过心跳来保持连接，这个连接状态称为 session，如果 znode 是临时节点，这个 session 失效，znode 也就删除了
5. znode 的目录名可以自动编号，如 App1 已经存在，再创建的话，将会自动命名为 App2
6. znode 可以被监控，包括这个目录节点中存储的数据的修改，子节点目录的变化等，一旦变化可以通知设置监控的客户端，这个是 Zookeeper 的核心特性，Zookeeper 的很多功能都是基于这个特性实现的，后面在典型的应用场景中会有实例介绍

如何使用

Zookeeper 作为一个分布式的服务框架，主要用来解决分布式集群中应用系统的一致性问题，它能提供基于类似于文件系统的目录节点树方式的数据存储，但是 Zookeeper 并不是用来专门存储数据的，它的作用主要是用来维护和监控你存储的数据的状态变化。通过监控这些数据状态的变化，从而可以达到基于数据的集群管理，后面将会详细介绍 Zookeeper 能够解决的一些典型问题，这里先介绍一下，Zookeeper 的操作接口和简单使用示例。

常用接口列表

客户端要连接 Zookeeper 服务器可以通过创建 org.apache.zookeeper. ZooKeeper 的一个实例对象，然后调用这个类提供的接口来和服务器交互。

前面说了 ZooKeeper 主要是用来维护和监控一个目录节点树中存储的数据的状态，所有我们能够操作 ZooKeeper 的也和操作目录节点树大体一样，如创建一个目录节点，给某个目录节点设置数据，获取某个目录节点的所有子目录节点，给某个目录节点设置权限和监控这个目录节点的状态变化。

这些接口如下表所示：

表 1 org.apache.zookeeper. ZooKeeper 方法列表

| **方法名** | **方法功能描述** |
| --- | --- |
| [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#create%28java.lang.String,%20byte[],%20java.util.List,%20org.apache.zookeeper.CreateMode%29)([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) | 创建一个给定的目录节点 path, 并给它设置数据，[CreateMode](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/CreateMode.html) 标识有四种形式的目录节点，分别是 PERSISTENT：持久化目录节点，这个目录节点存储的数据不会丢失；PERSISTENT\_SEQUENTIAL：顺序自动编号的目录节点，这种目录节点会根据当前已近存在的节点数自动加 1，然后返回给客户端已经成功创建的目录节点名；EPHEMERAL：临时目录节点，一旦创建这个节点的客户端与服务器端口也就是 session 超时，这种节点会被自动删除；EPHEMERAL\_SEQUENTIAL：临时自动编号节点 |
| [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#exists%28java.lang.String,%20boolean%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, boolean watch) | 判断某个 path 是否存在，并设置是否监控这个目录节点，这里的 watcher 是在创建 ZooKeeper 实例时指定的 watcher，[exists](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#exists%28java.lang.String,%20boolean%29)方法还有一个重载方法，可以指定特定的 watcher |
| [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#exists%28java.lang.String,%20org.apache.zookeeper.Watcher%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, [Watcher](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/Watcher.html) watcher) | 重载方法，这里给某个目录节点设置特定的 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 |
| void [delete](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#delete%28java.lang.String,%20int%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, int version) | 删除 path 对应的目录节点，version 为 -1 可以匹配任何版本，也就删除了这个目录节点所有数据 |
| [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#getChildren%28java.lang.String,%20boolean%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, boolean watch) | 获取指定 path 下的所有子目录节点，同样 [getChildren](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#getChildren%28java.lang.String,%20boolean%29)方法也有一个重载方法可以设置特定的 watcher 监控子节点的状态 |
| [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#setData%28java.lang.String,%20byte[],%20int%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, byte[] data, int version) | 给 path 设置数据，可以指定这个数据的版本号，如果 version 为 -1 怎可以匹配任何版本 |
| byte[] [getData](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#getData%28java.lang.String,%20boolean,%20org.apache.zookeeper.data.Stat%29)([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) | 获取这个 path 对应的目录节点存储的数据，数据的版本等信息可以通过 stat 来指定，同时还可以设置是否监控这个目录节点数据的状态 |
| void [addAuthInfo](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#addAuthInfo%28java.lang.String,%20byte[]%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) scheme, byte[] auth) | 客户端将自己的授权信息提交给服务器，服务器将根据这个授权信息验证客户端的访问权限。 |
| [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#setACL%28java.lang.String,%20java.util.List,%20int%29)([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) | 给某个目录节点重新设置访问权限，需要注意的是 Zookeeper 中的目录节点权限不具有传递性，父目录节点的权限不能传递给子目录节点。目录节点 ACL 由两部分组成：perms 和 id。 Perms 有 ALL、READ、WRITE、CREATE、DELETE、ADMIN 几种  而 id 标识了访问目录节点的身份列表，默认情况下有以下两种： ANYONE\_ID\_UNSAFE = new Id("world", "anyone") 和 AUTH\_IDS = new Id("auth", "") 分别表示任何人都可以访问和创建者拥有访问权限。 |
| [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#getACL%28java.lang.String,%20org.apache.zookeeper.data.Stat%29)([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) | 获取某个目录节点的访问权限列表 |

除了以上这些上表中列出的方法之外还有一些重载方法，如都提供了一个回调类的重载方法以及可以设置特定 Watcher 的重载方法，具体的方法可以参考 org.apache.zookeeper. ZooKeeper 类的 API 说明。

基本操作

下面给出基本的操作 ZooKeeper 的示例代码，这样你就能对 ZooKeeper 有直观的认识了。下面的清单包括了创建与 ZooKeeper 服务器的连接以及最基本的数据操作：

清单 2. ZooKeeper 基本的操作示例

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | // 创建一个与服务器的连接  ZooKeeper zk = new ZooKeeper("localhost:" + CLIENT\_PORT,         ClientBase.CONNECTION\_TIMEOUT, new Watcher() {             // 监控所有被触发的事件             public void process(WatchedEvent event) {                 System.out.println("已经触发了" + event.getType() + "事件！");             }         });  // 创建一个目录节点  zk.create("/testRootPath", "testRootData".getBytes(), Ids.OPEN\_ACL\_UNSAFE,    CreateMode.PERSISTENT);  // 创建一个子目录节点  zk.create("/testRootPath/testChildPathOne", "testChildDataOne".getBytes(),    Ids.OPEN\_ACL\_UNSAFE,CreateMode.PERSISTENT);  System.out.println(new String(zk.getData("/testRootPath",false,null)));  // 取出子目录节点列表  System.out.println(zk.getChildren("/testRootPath",true));  // 修改子目录节点数据  zk.setData("/testRootPath/testChildPathOne","modifyChildDataOne".getBytes(),-1);  System.out.println("目录节点状态：["+zk.exists("/testRootPath",true)+"]");  // 创建另外一个子目录节点  zk.create("/testRootPath/testChildPathTwo", "testChildDataTwo".getBytes(),    Ids.OPEN\_ACL\_UNSAFE,CreateMode.PERSISTENT);  System.out.println(new String(zk.getData("/testRootPath/testChildPathTwo",true,null)));  // 删除子目录节点  zk.delete("/testRootPath/testChildPathTwo",-1);  zk.delete("/testRootPath/testChildPathOne",-1);  // 删除父目录节点  zk.delete("/testRootPath",-1);  // 关闭连接  zk.close(); |

输出的结果如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | 已经触发了 None 事件！   testRootData   [testChildPathOne]  目录节点状态：[5,5,1281804532336,1281804532336,0,1,0,0,12,1,6]  已经触发了 NodeChildrenChanged 事件！   testChildDataTwo  已经触发了 NodeDeleted 事件！  已经触发了 NodeDeleted 事件！ |

当对目录节点监控状态打开时，一旦目录节点的状态发生变化，Watcher 对象的 process 方法就会被调用。

ZooKeeper 典型的应用场景

Zookeeper 从设计模式角度来看，是一个基于观察者模式设计的分布式服务管理框架，它负责存储和管理大家都关心的数据，然后接受观察者的注册，一旦这些数据的状态发生变化，Zookeeper 就将负责通知已经在 Zookeeper 上注册的那些观察者做出相应的反应，从而实现集群中类似 Master/Slave 管理模式，关于 Zookeeper 的详细架构等内部细节可以阅读 Zookeeper 的源码

下面详细介绍这些典型的应用场景，也就是 Zookeeper 到底能帮我们解决那些问题？下面将给出答案。

统一命名服务（Name Service）

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。说到这里你可能想到了 JNDI，没错 Zookeeper 的 Name Service 与 JNDI 能够完成的功能是差不多的，它们都是将有层次的目录结构关联到一定资源上，但是 Zookeeper 的 Name Service 更加是广泛意义上的关联，也许你并不需要将名称关联到特定资源上，你可能只需要一个不会重复名称，就像数据库中产生一个唯一的数字主键一样。

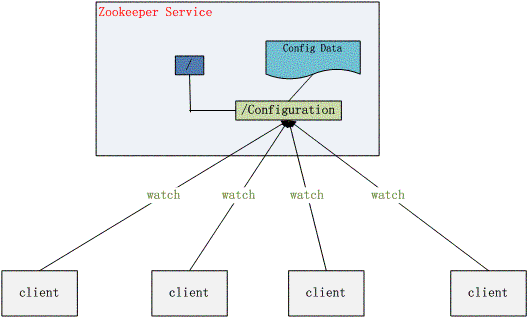
Name Service 已经是 Zookeeper 内置的功能，你只要调用 Zookeeper 的 API 就能实现。如调用 create 接口就可以很容易创建一个目录节点。

配置管理（Configuration Management）

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。

像这样的配置信息完全可以交给 Zookeeper 来管理，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。

图 2. 配置管理结构图



集群管理（Group Membership）

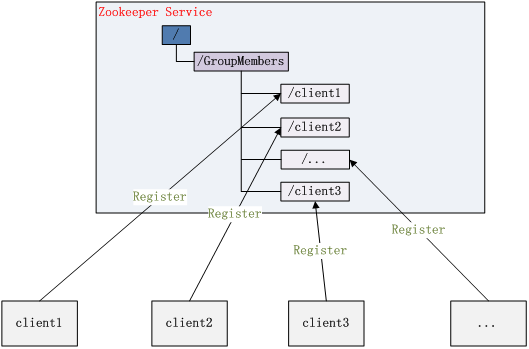
Zookeeper 能够很容易的实现集群管理的功能，如有多台 Server 组成一个服务集群，那么必须要一个“总管”知道当前集群中每台机器的服务状态，一旦有机器不能提供服务，集群中其它集群必须知道，从而做出调整重新分配服务策略。同样当增加集群的服务能力时，就会增加一台或多台 Server，同样也必须让“总管”知道。

Zookeeper 不仅能够帮你维护当前的集群中机器的服务状态，而且能够帮你选出一个“总管”，让这个总管来管理集群，这就是 Zookeeper 的另一个功能 Leader Election。

它们的实现方式都是在 Zookeeper 上创建一个 EPHEMERAL 类型的目录节点，然后每个 Server 在它们创建目录节点的父目录节点上调用 [getChildren](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#getChildren%28java.lang.String,%20boolean%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) path, boolean watch) 方法并设置 watch 为 true，由于是 EPHEMERAL 目录节点，当创建它的 Server 死去，这个目录节点也随之被删除，所以 Children 将会变化，这时 [getChildren](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#getChildren%28java.lang.String,%20boolean%29)上的 Watch 将会被调用，所以其它 Server 就知道已经有某台 Server 死去了。新增 Server 也是同样的原理。

Zookeeper 如何实现 Leader Election，也就是选出一个 Master Server。和前面的一样每台 Server 创建一个 EPHEMERAL 目录节点，不同的是它还是一个 SEQUENTIAL 目录节点，所以它是个 EPHEMERAL\_SEQUENTIAL 目录节点。之所以它是 EPHEMERAL\_SEQUENTIAL 目录节点，是因为我们可以给每台 Server 编号，我们可以选择当前是最小编号的 Server 为 Master，假如这个最小编号的 Server 死去，由于是 EPHEMERAL 节点，死去的 Server 对应的节点也被删除，所以当前的节点列表中又出现一个最小编号的节点，我们就选择这个节点为当前 Master。这样就实现了动态选择 Master，避免了传统意义上单 Master 容易出现单点故障的问题。

图 3. 集群管理结构图



这部分的示例代码如下，完整的代码请看附件：

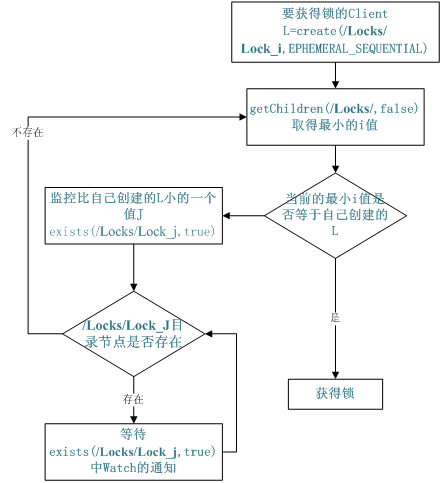
清单 3. Leader Election 关键代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | void findLeader() throws InterruptedException {         byte[] leader = null;         try {             leader = zk.getData(root + "/leader", true, null);         } catch (Exception e) {             logger.error(e);         }         if (leader != null) {             following();         } else {             String newLeader = null;             try {                 byte[] localhost = InetAddress.getLocalHost().getAddress();                 newLeader = zk.create(root + "/leader", localhost,                 ZooDefs.Ids.OPEN\_ACL\_UNSAFE, CreateMode.EPHEMERAL);             } catch (Exception e) {                 logger.error(e);             }             if (newLeader != null) {                 leading();             } else {                 mutex.wait();             }         }     } |

共享锁（Locks）

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](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#getChildren%28java.lang.String,%20boolean%29)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](http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/org/apache/zookeeper/ZooKeeper.html#exists%28java.lang.String,%20boolean%29)([String](http://java.sun.com/javase/6/docs/api/java/lang/String.html?is-external=true) 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。

图 4. Zookeeper 实现 Locks 的流程图



同步锁的实现代码如下，完整的代码请看附件：

清单 4. 同步锁的关键代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | void getLock() throws KeeperException, InterruptedException{         List<String> list = zk.getChildren(root, false);         String[] nodes = list.toArray(new String[list.size()]);         Arrays.sort(nodes);         if(myZnode.equals(root+"/"+nodes[0])){             doAction();         }         else{             waitForLock(nodes[0]);         }     }     void waitForLock(String lower) throws InterruptedException, KeeperException {         Stat stat = zk.exists(root + "/" + lower,true);         if(stat != null){             mutex.wait();         }         else{             getLock();         }     } |

队列管理

Zookeeper 可以处理两种类型的队列：

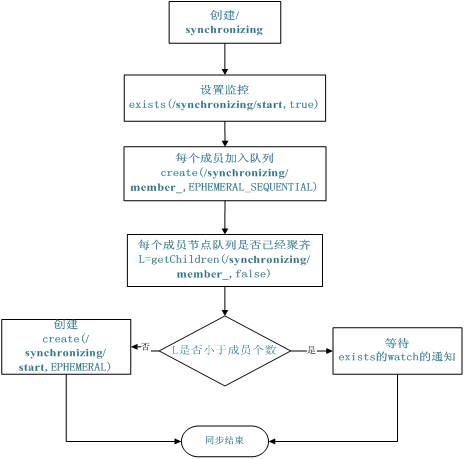
1. 当一个队列的成员都聚齐时，这个队列才可用，否则一直等待所有成员到达，这种是同步队列。
2. 队列按照 FIFO 方式进行入队和出队操作，例如实现生产者和消费者模型。

同步队列用 Zookeeper 实现的实现思路如下：

创建一个父目录 /synchronizing，每个成员都监控标志（Set Watch）位目录 /synchronizing/start 是否存在，然后每个成员都加入这个队列，加入队列的方式就是创建 /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，也就是 member\_i。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。

用下面的流程图更容易理解：

图 5. 同步队列流程图



同步队列的关键代码如下，完整的代码请看附件：

清单 5. 同步队列

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | void addQueue() throws KeeperException, InterruptedException{         zk.exists(root + "/start",true);         zk.create(root + "/" + name, new byte[0], Ids.OPEN\_ACL\_UNSAFE,         CreateMode.EPHEMERAL\_SEQUENTIAL);         synchronized (mutex) {             List<String> list = zk.getChildren(root, false);             if (list.size() < size) {                 mutex.wait();             } else {                 zk.create(root + "/start", new byte[0], Ids.OPEN\_ACL\_UNSAFE,                  CreateMode.PERSISTENT);             }         }  } |

当队列没满是进入 wait()，然后会一直等待 Watch 的通知，Watch 的代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public void process(WatchedEvent event) {         if(event.getPath().equals(root + "/start") &&          event.getType() == Event.EventType.NodeCreated){             System.out.println("得到通知");             super.process(event);             doAction();         }     } |

FIFO 队列用 Zookeeper 实现思路如下：

实现的思路也非常简单，就是在特定的目录下创建 SEQUENTIAL 类型的子目录 /queue\_i，这样就能保证所有成员加入队列时都是有编号的，出队列时通过 getChildren( ) 方法可以返回当前所有的队列中的元素，然后消费其中最小的一个，这样就能保证 FIFO。

下面是生产者和消费者这种队列形式的示例代码，完整的代码请看附件：

清单 6. 生产者代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | boolean produce(int i) throws KeeperException, InterruptedException{         ByteBuffer b = ByteBuffer.allocate(4);         byte[] value;         b.putInt(i);         value = b.array();         zk.create(root + "/element", value, ZooDefs.Ids.OPEN\_ACL\_UNSAFE,                     CreateMode.PERSISTENT\_SEQUENTIAL);         return true;     } |

清单 7. 消费者代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | int consume() throws KeeperException, InterruptedException{         int retvalue = -1;         Stat stat = null;         while (true) {             synchronized (mutex) {                 List<String> list = zk.getChildren(root, true);                 if (list.size() == 0) {                     mutex.wait();                 } else {                     Integer min = new Integer(list.get(0).substring(7));                     for(String s : list){                         Integer tempValue = new Integer(s.substring(7));                         if(tempValue < min) min = tempValue;                     }                     byte[] b = zk.getData(root + "/element" + min,false, stat);                     zk.delete(root + "/element" + min, 0);                     ByteBuffer buffer = ByteBuffer.wrap(b);                     retvalue = buffer.getInt();                     return retvalue;                 }             }         }  } |

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。

本文介绍的 Zookeeper 的基本知识，以及介绍了几个典型的应用场景。这些都是 Zookeeper 的基本功能，最重要的是 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，而不仅仅局限于上面提到的几个常用应用场景。