**zookeeper集群安装**

2010-07-19 15:43:22|  分类： [分布式](http://eshilin.blog.163.com/blog/#m=0&t=1&c=fks_084067083081084065080080082095085085083075092086085064084)

|字号

 订阅

Linux/OS X下:

**1 install**

1.1) 建立安装目录

mkdir -p /local/software/zookeeper/

cd /local/software/zookeeper/

1.2) 下载压缩包

 curl -O  http://apache.oss.eznetsols.org/hadoop/zookeeper/zookeeper-3.3.1/zookeeper-3.3.1.tar.gz

 1.3) 解压缩文件

 tar xzf zookeeper-3.3.1.tar.gz

 cd /local/software/zookeeper/

**2. Config**

我们把提供相同应用的服务器组称之为一个quorum，quorum中的所有机器都有相同的配置文件。

在这里我们以3个机器组成的quorum为例，具体配置如下。

2.1) 建立数据目录

 mkdir -p /local/software/zookeeper/zookeeper-3.3.1/data

2.2) 编辑配置文件***/local/software/zookeeper/zookeeper-3.3.1/conf/zoo.cfg***

dataDir=/local/software/zookeeper/zookeeper-3.3.1/data

clientPort=2181

initLimit=10

syncLimit=5

tickTime=2000

server.1=175.41.131.28:2888:3888

server.2=175.41.131.114:2888:3888

server.3=175.41.132.171:2888:3888

其中server.X代表组成整个服务的机器，当服务启动时，会在数据目录下查找这个文件myid,这个文件中存有服务器的号码。下面会讲myid文件的配置。

2.3) 创建dataDir参数指定的目录(这里指的是"/local/software/zookeeper/zookeeper-3.3.1/data")，并在目录下创建文件，命名为“myid”, 编辑该“myid”文件，并在对应的IP的机器上输入对应的编号。如在175.41.131.114上，“myid”文件内容就是2，在175.41.132.171上，内容就是 3

2.4）注意打开防火墙的端口

（1）如果是用iptable的话，在/etc/sysconfig/iptables中加入:

 -A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 2181 –j ACCEPT

 -A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 2888 –j ACCEPT

 -A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 3888 –j ACCEPT

执行命令：

       service iptables stop

       service iptables start

（2）EC2上的请打开相应的防火墙端口

**3. 启动服务**

 三台電腦先後下zkServer start 指令，無限定誰先誰後，但三台電腦間執行此指令的間隔不宜過久（因為有設定timeout時間）

 bin/zkServer.sh start

 没有出错信息，则成功启动

**4. 执行测试**

4.1）在一台机器如server.2上进行一个写操作：

＄*bin/zkCli.sh -server 127.0.0.1:2181*

[zk: 127.0.0.1:2181(CONNECTED) 1] *create /mytest test*

*[zk: 127.0.0.1:2181(CONNECTED) 3] ls /*

*[mytest, zookeeper]*

*[zk: 127.0.0.1:2181(CONNECTED) 5] get /mytest*

*test*

*cZxid = 0x100000002*

*ctime = Mon Jul 19 03:30:20 EDT 2010*

*mZxid = 0x100000002*

*mtime = Mon Jul 19 03:30:20 EDT 2010*

*pZxid = 0x100000002*

*cversion = 0*

*dataVersion = 0*

*aclVersion = 0*

*ephemeralOwner = 0x0*

*dataLength = 4*

*numChildren = 0*

4.2）在其他机器上查询：（如ssh://root@175.41.131.28:22）

*[zk: 127.0.0.1:2181(CONNECTED) 1] ls /*

*[mytest, zookeeper]*

*[zk: 127.0.0.1:2181(CONNECTED) 2] ls /mytest*

*[]*

*[zk: 127.0.0.1:2181(CONNECTED) 3] get /mytest*

*test*

*cZxid = 0x100000002*

*ctime = Mon Jul 19 03:30:20 EDT 2010*

*mZxid = 0x100000002*

*mtime = Mon Jul 19 03:30:20 EDT 2010*

*pZxid = 0x100000002*

*cversion = 0*

*dataVersion = 0*

*aclVersion = 0*

*ephemeralOwner = 0x0*

*dataLength = 4*

*numChildren = 0*

*$bin/zkServer.sh status*

*JMX enabled by default*

*Using config: /local/software/zookeeper/zookeeper-3.3.1/bin/../conf/zoo.cfg*

*2010-07-19 04:09:49,862 - INFO  [NIOServerCxn.Factory:0.0.0.0/0.0.0.0:2181:NIOServerCnxn$Factory@250] - Accepted socket connection from /127.0.0.1:42234*

*2010-07-19 04:09:49,864 - INFO  [NIOServerCxn.Factory:0.0.0.0/0.0.0.0:2181:NIOServerCnxn@1236] - Processing stat command from /127.0.0.1:42234*

*2010-07-19 04:09:49,866 - INFO  [Thread-20:NIOServerCnxn$StatCommand@1152] - Stat command output*

*2010-07-19 04:09:49,866 - INFO  [Thread-20:NIOServerCnxn@1434] - Closed socket connection for client /127.0.0.1:42234 (no session established for client)*

*Mode: follower*

5.注意事项

     通过shell脚本在每个机器上启动zookeeper的时候，可能会显示错误信息“Cannot open channel to X at election address”。这是由于zoo.cfg文件中指定的其他zookeeper服务找不到所导致。所有机器的zookeeper服务启动之后该错误提示将会消失。

**Zookeeper 簡介**

2010-07-19 15:53:23|  分类： [分布式](http://eshilin.blog.163.com/blog/#m=0&t=1&c=fks_084067083081084065080080082095085085083075092086085064084)

|字号

 订阅

Zookeeper是Google的Chubby一個開源的實現.是高有效和可靠的協同工作系統.Zookeeper能夠用來leader選舉,配置信息維護等.在一個分佈式的環境中,我們需要一個Master實例或存儲一些配置信息,確保文件寫入的一致性等.

在Zookeeper中,znode是一個跟Unix文件系統路徑相似的節點,可以往這個節點存儲或獲取數據.如果在創建znode時Flag設置為 EPHEMERAL,那麼當這個創建這個znode的節點和Zookeeper失去連接後,這個znode將不再存在在Zookeeper 裡.Zookeeper使用Watcher察覺事件信息,當客戶端接收到事件信息,比如連接超時,節點數據改變,子節點改變,可以調用相應的行為來處理數 據.

那麼Zookeeper能幫我們作什麼事情呢?簡單的例子:

假設我們我們有個20個搜索引擎的服務器(每個負責總索引中的一部分的搜索任務)和一個 總服務器(負責向這20個搜索引擎的服務器發出搜索請求併合並結果集),一個備用的總服務器(負責當總服務器宕機時替換總服務器),一個web的 cgi(向總服務器發出搜索請求).搜索引擎的服務器中的15個服務器現在提供搜索服務,5個服務器正在生成索引.這20個搜索引擎的服務器經常要讓正在 提供搜索服務的服務器停止提供服務開始生成索引,或生成索引的服務器已經把索引生成完成可以搜索提供服務了.

使用Zookeeper可以保證總服務器自動感知有多少提供搜索引擎的服務器並向這些服務器發出搜索請求,備用的總服務器宕機時自動啟用備用的總服務器,web的cgi能夠自動地獲知總服務器的網絡地址變化.這些又如何做到呢?

1. 提供搜索引擎的服務器都在Zookeeper中創建znode,zk.create("/search/nodes/node1", "hostname".getBytes(), Ids.OPEN\_ACL\_UNSAFE, CreateFlags.EPHEMERAL);

2. 總服務器可以從Zookeeper中獲取一個znode的子節點的列表,zk.getChildren("/search/nodes", true);

3. 總服務器遍歷這些子節點,並獲取子節點的數據生成提供搜索引擎的服務器列表.

4. 當總服務器接收到子節點改變的事件信息,重新返回第二步.

5. 總服務器在Zookeeper中創建節點,zk.create("/search/master", "hostname".getBytes(), Ids.OPEN\_ACL\_UNSAFE, CreateFlags.EPHEMERAL);

6. 備用的總服務器監控Zookeeper中的"/search/master"節點.當這個znode的節點數據改變時,把自己啟動變成總服務器,並把自己的網絡地址數據放進這個節點.

7. web的cgi從Zookeeper中"/search/master"節點獲取總服務器的網絡地址數據並向其發送搜索請求.

8. web的cgi監控Zookeeper中的"/search/master"節點,當這個znode的節點數據改變時,從這個節點獲取總服務器的網絡地址數據,並改變當前的總服務器的網絡地址.

一個Zookeeper的集群中,3個Zookeeper節點.一個leader,兩個follower的情況下,停掉leader,然後兩個follower選舉出一個leader.獲取的數據不變

Zookeeper能夠幫助Hadoop做到:

Hadoop,使用Zookeeper的事件處理確保整個集群只有一個NameNode,存儲配置信息等.

HBase,使用Zookeeper的事件處理確保整個集群只有一個HMaster,察覺HRegionServer聯機和宕機,存儲訪問控制列表等.

[转]sites.google.com/site/waue0920/Home/zookeeper/

Zookeeper原理介绍

Zookeeper可以运行在两种模式下，一种是单机模式，单机模式就是只有一个zookeeper service ，这对与测试来讲非常方便，但不能保证服务的高性能和高可靠性；另一种就是复制模式（集群模式），在这种模式下，只要集群中的大多数正常工作，就可以提供稳定的高性能服务，如五个节点的ensemble，任意两个节点失败，服务器仍然可以正常工作，实现原理很简单：确保znode树的每一次修改都被复制到ensemble的大多数机器中。

Zookeeper使用zab协议，运行在两个阶段（状态）[phase]：

Phase1：leader elction

选举一个杰出的组员（一个zookeeper service），称之为leader，其他的机器称之为followers.

Phase2:Atomic broadcast

所有的写请求传递到leader，leader通过广播更新followers。当大多数更改后，leader提交更新，同时client得到响应：更新成功。

如果leader失败，剩下的机器进行另一个leader的选举，如果以前的leader恢复，它将变成一个follower，leader elction非常快，大约200ms，因此，在选举过程中不会是行你呢个显著降低。

除了request processor, 组成zookeeper服务的每个server都会在本地备份其它组件的拷贝. replicated database是一个包含整个数据树的内存数据库.更新被logged到磁盘以提供可恢复性,写操作先持久化到磁盘，然后再对内存数据库作变更.

      每个zookeeper server都对client提供服务.client连接到具体的某一个server以提交请求.读操作依赖与每个server的本地数据库.改变服务状态的请求,写操作,由一致性协议来处理.

作为一致性协议的一部分，所有的client写请求被提交到专门的一个leader server. 其余的server,被称为followers,从leader接收消息,并对消息的传递达成一致.

      消息层负责替换失效leader并同步followers.zookeeper使用自定义的原子消息传递协议.因为消息传递层是原子性的,zookeeper能够确保本地备份不会出现分歧.When the leader receives a write request, it calculates what the state of the system is when the write is to be applied and transforms this into a transaction that captures this new state.

        在使用上，zookeeper的编程接口特意做的非常简单.在此之上，你可以实现更高层次的操作, 如同步原语, 组管理, 所有权等等.

++++++++++++++++++++++++++

2.1 Katta和zookeeper的关系

2.4.1 什么是zookeeper?

zookeeper是针对分布式应用的分布式协作服务，它的目的就是为了减轻分布式应用从头开发协作服务的负担。

它的基本功能是命名服务(naming)，配置管理(configuration management)，同步(synchronization)和组服务 (group services)。在此基础上可以实现分布式系统的一致性，组管理，Leader选举等功能。

一个zookeeper机群包含多个zookeeper服务器，这些Server彼此都知道对方的存在。 Zookeeper系统结构图如图2所示：

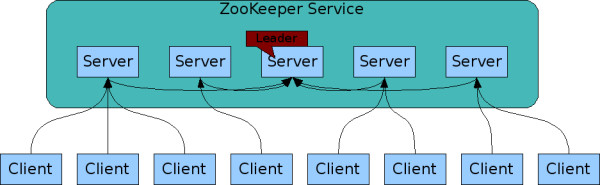


                图2    zookeeper系统结构图

l.所有的Server都保存有一份目前zookeeper系统状态的备份；

2.在zookeeper启动的时候，会自动选取一个Server作为Leader，其余的Server都是Follower；

3.作为Follower的Server服务于Client,接受Client的请求，并把Client的请求转交给Leader，由Leader提交请求。

4.Client只与单个的zookeeper服务器连接。Client维护一个持久TCP连接，通过其发送请求, 获取响应和事件,并发送心跳信息。如果Client到Server的TCP连接中断, Client将会连接到另外一个Server。

5.zookeeper机群的鲁棒性是我们使用它的原因，只要不超过半数的服务器当机（如果正常服务的服务器数目不足一半，那么原有的机群将可能被划分成两个信息无法一致的zookeeper服务），该服务就能正常运行。

2.4.2 zookeeper的虚拟文件系统

Zookeeper允许多个分布在不同服务器上的进程基于一个共享的、类似标准文件系统的树状虚拟文件系统来进行协作。虚拟文件系统中的每个数据节点都称作一个znode。每个znode都可以把数据关联到它本身或者它的子节点.如图3所示：

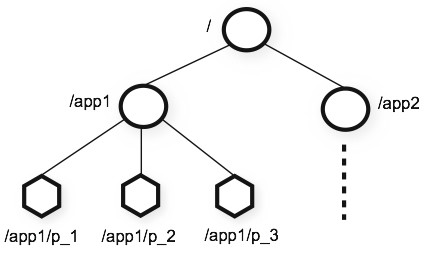


         图3 zookeeper的虚拟文件系统目录结构

1.每个znode的名称都是绝对路径名来组成的，如“/katta/index/index\_name”等。

2.读取或写入znode中的数据都是原子操作，read会获取znode中的所有字节, write会整个替换znode中的信息.每个znode都包含一个访问控制列表(ACL)以约束该节点的访问者和权限.

3.有些znode是临时节点.临时节点在创建它的session的生命周期内存活, 当其session终止时，此类节点将会被删除.

4.zookeeper提供znode监听器的概念. Client可以在某个znode上设置监听器以监听该znode的变更. 当znode有变更时, 这些Client将会收到通知,并执行预先敲定好的函数。

**2.4.3 Katta与zookeeper的关系**

l Katta使用[zookeeper](http://www.hadoop.org.cn/hadoop/what-is-hadoop-zookeeper/)保证Master主节点和Node子节点的有效性，在Master主节点和Node子节点之间传递消息，保存一些配置信息，保证文件读取的一致性等。

2 Master管理服务器, Node检索服务器和Client服务器之间的通信就是通过zookeeper来实现的。

3 Client服务器可以直接从zookeeper服务中读取Node检索服务器列表，并向Node检索服务器发送检索请求，最后从Node检索服务器得到结果，不必经过Master管理服务器。

+++++++++++++++++++++++++++++

**Hadoop中的子项目Zookeeper能做什么**

很高兴得看到Yahoo捐献的Zookeeper已经从sourceforge迁移到Apache,并成为Hadoop的子项目.那么ZooKeeper是什么呢?Zookeeper是Google的[Chubby](http://labs.google.com/papers/chubby.html)一个开源的实现.是高有效和可靠的协同工作系统.Zookeeper能够用来leader选举,配置信息维护等.在一个分布式的环境中,我们需要一个Master实例或存储一些配置信息,确保文件写入的一致性等.Zookeeper能够保证如下3点:

* Watches are ordered with respect to other events, other watches, and asynchronous replies. The ZooKeeper client libraries ensures that everything is dispatched in order.
* A client will see a watch event for a znode it is watching before seeing the new data that corresponds to that znode.
* The order of watch events from ZooKeeper corresponds to the order of the updates as seen by the ZooKeeper service.

在Zookeeper中,znode是一个跟Unix文件系统路径相似的节点,可以往这个节点存储或获取数据.如果在创建znode时Flag设置为EPHEMERAL,那么当这个创建这个znode的节点和Zookeeper失去连接后,这个znode将不再存在在Zookeeper里.Zookeeper使用Watcher察觉事件信息,当客户端接收到事件信息,比如连接超时,节点数据改变,子节点改变,可以调用相应的行为来处理数据.Zookeeper的[Wiki页面](http://zookeeper.wiki.sourceforge.net/ZooKeeperRecipes)展示了如何使用Zookeeper来处理事件通知,队列,优先队列,锁,共享锁,可撤销的共享锁,两阶段提交.

那么Zookeeper能帮我们作什么事情呢?

简单的例子:

假设我们我们有 1)20个搜索引擎的服务器(每个负责索引或搜索任务) 2)一个总服务器(负责向这20个搜索引擎的服务器发出搜索请求并合并结果集), 3)一个备用的总服务器(负责当总服务器宕机时替换总服务器), 4)一个web的cgi(向总服务器发出搜索请求).

现在搜索引擎的20个服务器中的15个服务器现在提供搜索服务,5个服务器正在生成索引.这20个搜索引擎的服务器经常要让正在提供搜索服务的服务器停止提供搜索服务开始生成索引,或生成索引的服务器已经把索引生成完成可以搜索提供服务了.

使用Zookeeper 1）可以保证总服务器自动感知有多少提供搜索引擎的服务器并向这些服务器发出搜索请求, 2）备用的总服务器宕机时自动启用备用的总服务器, 3）web的cgi能够自动地获知总服务器的网络地址变化.

这些又如何做到呢?

1. 提供搜索引擎的服务器都在Zookeeper中创建znode：  zk.create("/search/nodes/node1","hostname".getBytes(), Ids.OPEN\_ACL\_UNSAFE,                                         CreateFlags.EPHEMERAL);
2. 总服务器可以从Zookeeper中获取一个znode的子节点的列表： zk.getChildren("/search/nodes", true);
3. 总服务器遍历这些子节点,并获取子节点的数据生成提供搜索引擎的服务器列表.
4. 当总服务器接收到子节点改变的事件信息,重新返回第二步.
5. 总服务器在Zookeeper中创建节点, zk.create("/search/master", "hostname".getBytes(), Ids.OPEN\_ACL\_UNSAFE,                                         CreateFlags.EPHEMERAL);
6. 备用的总服务器监控Zookeeper中的"/search/master"节点.当这个znode的节点数据改变时,把自己启动变成总服务器,并把自己的网络地址数据放进这个节点.
7. web的cgi从Zookeeper中"/search/master"节点获取总服务器的网络地址数据并向其发送搜索请求.
8. web的cgi监控Zookeeper中的"/search/master"节点,当这个znode的节点数据改变时,从这个节点获取总服务器的网络地址数据,并改变当前的总服务器的网络地址.

在我的测试中:一个Zookeeper的集群中,3个Zookeeper节点.一个leader,两个follower的情况下,停掉leader,然后两个follower选举出一个leader.获取的数据不变.我想Zookeeper能够帮助Hadoop做到:

Hadoop,使用Zookeeper的事件处理确保整个集群只有一个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 3.2 单机安装**

2010-07-19 15:53:59|  分类： [分布式](http://eshilin.blog.163.com/blog/#m=0&t=1&c=fks_084067083081084065080080082095085085083075092086085064084)

|字号

 订阅

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 本篇為Zookeeper 的獨立測試 （不包含搭配 Hadoop, Hbase 的協同合作）   * 測試於 Ubuntu 9.10 ， Zookeeper 為 3.2.1 版，java 版本為 1.6 ，並且於環境變數已經加入 JAVA\_HOME=/usr/lib/jvm/java-6-sun * 設定好 ssh localhost 免密碼  |  |  | | --- | --- | | 安裝目錄 | /opt/zookeeper/ | | 工作目錄 | /var/zookeeper/ |   安装步骤   1. 下載 zookeeper 最新版，並且解壓縮到 /opt/zookeeper 2. 新建 /var/zookeeper 當作 zookeeper的工作目錄，並確實設定可讀寫權限   新建配置文档 /opt/zookeeper/conf/zoo.cfg  tickTime=2000  dataDir=/var/zookeeper  clientPort=2181  $ cd /local/software/zookeeper/;ln -s zookeeper-3.3.1/ zookeeper  $ export ZOOKEEPER\_INSTALL=/local/software/zookeeper/zookeeper/  $ export PATH=$PATH:$ZOOKEEPER\_INSTALL/bin  接着执行  $ cd /opt/zookeeper  $ bin/zkServer.sh start  完成启动  运行 netstat -tnl  可以看到 2181 端口已打开  关闭  ＄bin/zkServer.sh stop  执行测试  接着执行  ＄bin/zkCli.sh -server 127.0.0.1:2181  基本测试  [zkshell: 0] help  [zkshell: 8] ls /  [zookeeper]  [zkshell: 9] create /zk\_test my\_data  Created /zk\_test  [zkshell: 11] ls /  [zookeeper, zk\_test]  [zkshell: 12] get /zk\_test  my\_data  [zkshell: 14] set /zk\_test junk  [zkshell: 15] get /zk\_test  junk  [zkshell: 16] quit |

**ZooKeeper配置文件参数**

2010-07-19 16:31:01|  分类： [分布式](http://eshilin.blog.163.com/blog/#m=0&t=1&c=fks_084067083081084065080080082095085085083075092086085064084)

|字号

 订阅

参考：http://hadoop.apache.org/zookeeper/docs/r3.3.1/zookeeperAdmin.html#sc\_configuration

ZooKeeper Server的行为受配置文件zoo.cfg的控制，zoo.cfg的设计目标是让所有服务器都可以使用相同的配置文件，如果需要使用不同的配置文件，需要保证关于cluster部分的参数相同。下面是具体的参数：

最小必要配置的参数

clientPort

服务的监听端口

dataDir

用于存放内存数据库快照的文件夹，同时用于集群的myid文件也存在这个文件夹里（注意：一个配置文件只能包含一个dataDir字样，即使它被注释掉了。）

tickTime

心跳时间，为了确保连接存在的，以毫秒为单位，最小超时时间为两个心跳时间

高级配置参数

dataLogDir

用于单独设置transaction log的目录，transaction log分离可以避免和普通log还有快照的竞争

globalOutstandingLimit

client请求队列的最大长度，防止内存溢出，默认值为1000

preAllocSize

预分配的Transaction log空间为block个proAllocSize KB，默认block为64M，一般不需要更改，除非snapshot过于频繁

snapCount

在snapCount个snapshot后写一次transaction log，默认值是100,000

traceFile

用于记录请求的log，打开会影响性能，用于debug的，最好不要定义

maxClientCnxns

最大并发客户端数，用于防止DOS的，默认值是10，设置为0是不加限制

clientPortBindAddress

3.3.0后新增参数，可是设置指定的client ip以及端口，不设置的话等于ANY:clientPort

minSessionTimeout

3.3.0后新增参数，最小的客户端session超时时间，默认值为2个tickTime，单位是毫秒

maxSessionTimeout

3.3.0后新增参数，最大的客户端session超时时间，默认值为20个tickTime，单位是毫秒

集群参数

electionAlg

用于选举的实现的参数，0为以原始的基于UDP的方式协作，1为不进行用户验证的基于UDP的快速选举，2为进行用户验证的基于UDP的快速选举，3为基于TCP的快速选举，默认值为3

initLimit

多少个心跳时间内，允许其他server连接并初始化数据，如果ZooKeeper管理的数据较大，则应相应增大这个值

leaderServes

leader是否接受客户端连接。默认值为yes。 leader负责协调更新。当更新吞吐量远高于读取吞吐量时，可以设置为不接受客户端连接，以便leader可以专注于同步协调工作。默认值是yes，说明leader可以接受客户端连接。（注意：当集群中有超过3台ZooKeeper Server时，强烈建议打开leader选举）

server.x=[hostname]:nnnnn[:nnnnn], etc

配置集群里面的主机信息，其中server.x的x要写在myid文件中，决定当前机器的id，第一个port用于连接leader，第二个用于leader选举。如果electionAlg为0，则不需要第二个port。hostname也可以填ip。

syncLimit

多少个tickTime内，允许follower同步，如果follower落后太多，则会被丢弃。

group.x=nnnnn[:nnnnn]

weight.x=nnnnn

这两个是用于集群分组的参数，暂时只有3台机器，没仔细研究，先给个例子看看吧

group.1=1:2:3

group.2=4:5:6

group.3=7:8:9

weight.1=1

weight.2=1

weight.3=1

weight.4=1

weight.5=1

weight.6=1

weight.7=1

weight.8=1

weight.9=1

用于用户认证的选项

略

不安全选项

略

**ZooKeeper集群部署管理详细版**

2010-07-19 16:33:11|  分类： [分布式](http://eshilin.blog.163.com/blog/#m=0&t=1&c=fks_084067083081084065080080082095085085083075092086085064084)

|字号

 订阅

1. 约定：

a. ZooKeeper Server，[server1]的ip为192.168.1.201，[server2]的ip为192.168.1.202，[server3]的ip为192.168.1.203。

b. [zk\_dir]表示ZooKeeper的根目录，假设为/home/user/zookeeper

c. [zk\_data]表示ZooKeeper数据文件存放地址，假设为/home/user/zk\_data

d. [zk\_trlog]表示ZooKeeper的Transaction Log目录，假设为/home/user/zk\_trlog

e. [zk\_logs]表示ZooKeeper的一半log目录，假设为/home/user/zk\_logs

2. 服务端环境要求：

a. Linux或者其他类UNIX系统（Windows及Mac OS只能作为开发环境，不能做生产环境）

b. JDK 1.6或更高

c. Server管理需要netcat或者telnet

3. 获得ZooKeeper发行版：

访问：http://hadoop.apache.org/zookeeper/releases.html，目前ZooKeeper的最新版本为3.3.1版本（2010年5月17日发行）

4． 修改配置文件：

因为ZooKeeper的配置文件设计目标是可供所有机器使用的通用配置文件，因此可以现在一台机器上写完，然后复制到其他server上。在[zk\_dir]/conf下新建zoo.cfg文件，详细参数说明参见附录，这里给出一个配置文件的例子：

*# BASIC SETTINGS*

*# The number of milliseconds of each tick*

tickTime=2000

*# the port at which the clients will connect*

clientPort=2181

*# the directory where the snapshot is stored.*

dataDir=**/**home**/**user**/**zk\_logs**/**zk\_data

*# the directory where the transaction log is stored*

dataLogDir=**/**home**/**user**/**zk\_logs**/**zk\_trog

*# BASIC SETTINGS END*

*##########################################################*

*# CLUSTER SETTINGS*

*# 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

*# CLUSTER SERVER CONFIG*

server.1=192.168.1.201:2888:3888

server.2=192.168.1.202:2888:3888

server.3=192.168.1.203:2888:3888

*# CLUSTER SETTINGS END*

其中的# CLUSTER SERVER CONFIG部分为集群的三台机器的ip

5． 为每台服务器设置id：

在[server1]的[zk\_data]目录下新建myid文件，纯文本ASCII编码文件即可，内容为1，也就是配置文件中server.x中的x的值。其他两台机器上依此类推，也新建内容为自己id的myid文件。

6． 配置log文件信息：

ZooKeeper采用log4j记录日志，log4j.properties文件在[zk\_dir]/conf目录下，编辑[zk\_dir]/conf/log4j.properties文件，按需要配置，这里给出一个在文件里记录DEBUG, INFO, ERROR等级，并且文件每天重命名一次的例子：

*#*

*# ZooKeeper Logging Configuration*

*#*

log4j.rootLogger=INFO, DEBUG, ERROR

*# DEBUG*

log4j.appender.DEBUG=org.apache.log4j.DailyRollingFileAppender

log4j.appender.DEBUG.Threshold=DEBUG

log4j.appender.DEBUG.layout=org.apache.log4j.PatternLayout

log4j.appender.DEBUG.layout.ConversionPattern=**%**d**{**ISO8601**}** - **%**-5p **[%**t:**%**C**{**1**}@%**L**]** - **%**m**%**n

log4j.appender.DEBUG.datePattern='.'yyyy-MM-dd'.log'

log4j.appender.DEBUG.append=**true**

log4j.appender.DEBUG.File=**/**home**/**user**/**zk\_logs**/**zk\_debug.log

*# INFO*

log4j.appender.INFO=org.apache.log4j.DailyRollingFileAppender

log4j.appender.INFO.Threshold=INFO

log4j.appender.INFO.layout=org.apache.log4j.PatternLayout

log4j.appender.INFO.layout.ConversionPattern=**%**d**{**ISO8601**}** - **%**-5p **[%**t:**%**C**{**1**}@%**L**]** - **%**m**%**n

log4j.appender.INFO.datePattern='.'yyyy-MM-dd'.log'

log4j.appender.INFO.append=**true**

log4j.appender.INFO.File=**/**home**/**user**/**zk\_logs**/**zk\_error.log

*# ERROR*

log4j.appender.ERROR=org.apache.log4j.DailyRollingFileAppender

log4j.appender.ERROR.Threshold=ERROR

log4j.appender.ERROR.layout=org.apache.log4j.PatternLayout

log4j.appender.ERROR.layout.ConversionPattern=**%**d**{**ISO8601**}** - **%**-5p **[%**t:**%**C**{**1**}@%**L**]** - **%**m**%**n

log4j.appender.ERROR.datePattern='.'yyyy-MM-dd'.log'

log4j.appender.ERROR.append=**true**

log4j.appender.ERROR.File=**/**home**/**user**/**zk\_logs**/**zk\_error.log

7. Server简单管理命令：

现在可以启动ZooKeeper服务器了：

# Server启动命令

$ [zk\_dir]/bin/zkServer.sh start

# Server重启

$ [zk\_dir]/bin/zkServer.sh restart

# Server停止

$ [zk\_dir]/bin/zkServer.sh stop

# Server基本状态查看，需要netcat

$ [zk\_dir]/bin/zkServer.sh status

8. ZooKeeper的管理命令：

需要使用telnet或者netcat连接到ZooKeeper Server的clientPort端口，命令均为4个字母，命令及功能如下：

conf

3.3.0新增: 打印server配置文件内容

cons

3.3.0新增:列出连接到本Server的Client的完整连接及Session信息。包括分组发送/接收信息，session id，操作延时，最后一个操作的执行等

crst

3.3.0新增: 重置所有连接/session

dump

Lists the outstanding sessions and ephemeral nodes. 只在leader server上有效。

envi

屏幕打印server的环境变量

ruok

测试server是否处于无错状态。 如果状态健康则返回”imok”，否则无任何结果。

A response of “imok” does not necessarily indicate that the server has joined the quorum, just that the server process is active and bound to the specified client port. Use “stat” for details on state wrt quorum and client connection information.

srst

Reset server statistics.

srvr

3.3.0新增: Lists full details for the server.

stat

Lists brief details for the server and connected clients.

wchs

3.3.0新增: Lists brief information on watches for the server.

wchc

3.3.0新增: Lists detailed information on watches for the server, by session. This outputs a list of sessions(connections) with associated watches (paths). Note, depending on the number of watches this operation may be expensive (ie impact server performance), use it carefully.

wchp

3.3.0新增: Lists detailed information on watches for the server, by path. This outputs a list of paths (znodes) with associated sessions. Note, depending on the number of watches this operation may be expensive (ie impact server performance), use it carefully.

**Apache Zookeeper入门1**

*1*人收藏此文章， [收藏此文章](javascript:pay_attention(11280,true))

发表于6个月前 , 已有**1256**次阅读 共**0**个评论 **1**人收藏此文章

作者: H.E. | 您可以转载, 但必须以超链接形式标明文章原始出处和作者信息及版权声明  网址: <http://www.javabloger.com/article/apache-zookeeper-hadoop.html>

**口水:**Zookeeper是我目前接触过Apache开源系统中比较复杂的一个产品，要搞清楚这个东东里面的运作关系还真不是一时半会可以搞定的事，本人目前只略知皮毛之术。

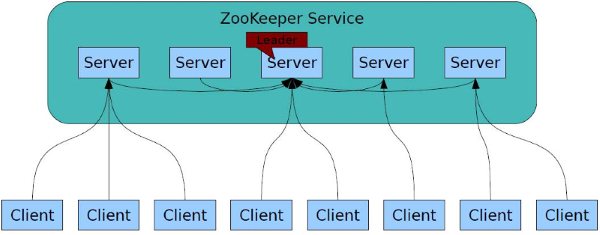
**ZooKeeper 是什么？**

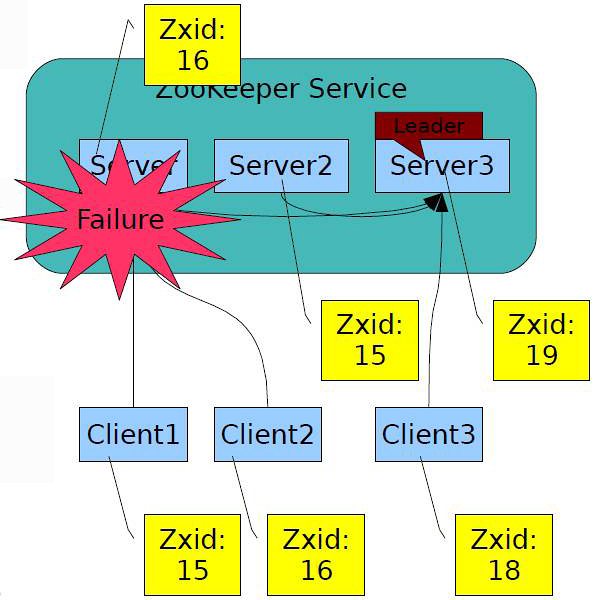
  ZooKeeper 顾名思义 动物园管理员，他是拿来管[大象(Hadoop)](http://hadoop.apache.org/) 、 [蜜蜂(Hive)](http://hive.apache.org/) 、[小猪(Pig)](http://pig.apache.org/)  的管理员， Apache Hbase和 Apache Solr 以及[LinkedIn sensei](http://sna-projects.com/sensei) 等项目中都采用到了 Zookeeper。ZooKeeper是一个分布式的，开放源码的分布式应用程序协调服务，ZooKeeper是以[Fast Paxos](http://rdc.taobao.com/blog/cs/?p=261)算法为基础，实现同步服务，配置维护和命名服务等分布式应用。

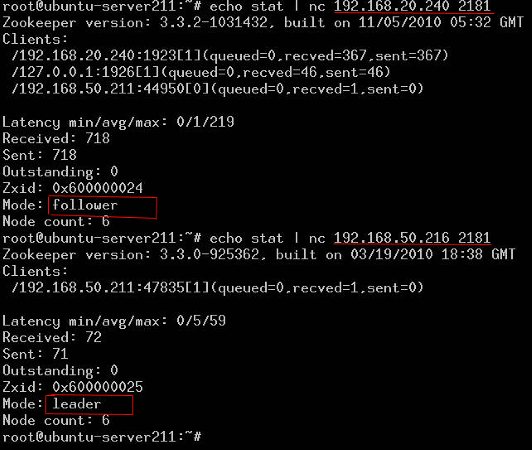
**ZooKeeper 如何工作？**

  ZooKeeper是作为分布式应用建立更高层次的同步(synchronization)、配置管理 (configuration maintenance)、群组(groups)以及名称服务(naming)。在编程上，ZooKeeper设计很简单，所使用的数据模型风格很像文件系统的目录树结构，简单来说，有点类似windows中注册表的结构，有名称，有树节点，有Key(键)/Value(值)对的关系，可以看做一个树形结构的数据库，分布在不同的机器上做名称管理。

   Zookeeper分为2个部分：服务器端和客户端，客户端只连接到整个ZooKeeper服务的某个服务器上。客户端使用并维护一个TCP连接，通过这个连接发送请求、接受响应、获取观察的事件以及发送心跳。如果这个TCP连接中断，客户端将尝试连接到另外的ZooKeeper服务器。客户端第一次连接到ZooKeeper服务时，接受这个连接的 ZooKeeper服务器会为这个客户端建立一个会话。当这个客户端连接到另外的服务器时，这个会话会被新的服务器重新建立。

   启动Zookeeper服务器集群环境后，多个Zookeeper服务器在工作前会选举出一个Leader，在接下来的工作中这个被选举出来的Leader死了，而剩下的Zookeeper服务器会知道这个Leader死掉了，在活着的Zookeeper集群中会继续选出一个Leader，选举出leader的目的是为了可以在分布式的环境中保证数据的一致性。如图所示： [](http://niaklq.bay.livefilestore.com/y1pbARJq1ijQRd1v-pIceq9GqFvcw6FjLE7bGM59B8FAidzR_goQ5z23rLkXdoGFVyES43s3c_lYp8DHVna89l6wx_e4cxii8aP/zookeeper-blog-1.jpg?psid=1)

  另外，ZooKeeper 支持watch(观察)的概念。客户端可以在每个znode结点上设置一个观察。如果被观察服务端的znode结点有变更，那么watch就会被触发，这个watch所属的客户端将接收到一个通知包被告知结点已经发生变化。若客户端和所连接的ZooKeeper服务器断开连接时，其他客户端也会收到一个通知，也就说一个Zookeeper服务器端可以对于多个客户端，当然也可以多个Zookeeper服务器端可以对于多个客户端，如图所示： [](http://niaklq.bay.livefilestore.com/y1pIeYJau1gfAywPWxuBGKiLDdRjr9O2gRWMJyCZf5mP4RyugNPZGhEzJRQg-GV2j2zSZao92O_YXAeVESTKqM4sjiq9FdCaUUr/zookeeper-blog-2.jpg?psid=1)

你还可以通过命令查看出，当前那个Zookeeper服务端的节点是Leader，哪个是Follower，如图所示： [](http://niaklq.bay.livefilestore.com/y1p2oHPXOPJ-U760tUrelzMf-yqtDd46Jl2c-yjOfnFsjQ0y8wH6gTt0JHx80y5Aha3UyAqhNOtECfJyrDYNGiR194G7bH52Gl-/zookeeper-node-list.jpg?psid=1)

我通过试验观察到 Zookeeper的集群环境最好有3台以上的节点，如果只有2台，那么2台当中不管那台机器down掉，将只会剩下一个leader，那么如果有再有客户端连接上来，将无法工作，并且剩下的leader服务器会不断的抛出异常。内容如下： java.net.ConnectException: Connection refused         at sun.nio.ch.Net.connect(Native Method)         at sun.nio.ch.SocketChannelImpl.connect(SocketChannelImpl.java:507)         at java.nio.channels.SocketChannel.open(SocketChannel.java:146)         at org.apache.zookeeper.server.quorum.QuorumCnxManager.connectOne(QuorumCnxManager.java:347)         at org.apache.zookeeper.server.quorum.QuorumCnxManager.connectAll(QuorumCnxManager.java:381)         at org.apache.zookeeper.server.quorum.FastLeaderElection.lookForLeader(FastLeaderElection.java:674)         at org.apache.zookeeper.server.quorum.QuorumPeer.run(QuorumPeer.java:611) 2010-11-15 00:31:52,031 – INFO  [QuorumPeer:/0:0:0:0:0:0:0:0:2181:FastLeaderElection@683] – Notification time out: 12800

并且客户端连接时还会抛出这样的异常，说明连接被拒绝，并且等待一个socket连接新的连接，这里socket新的连接指的是zookeeper中的一个Follower。 org.apache.zookeeper.ClientCnxn$SendThread.startConnect(ClientCnxn.java:1000) Opening socket connection to server 192.168.50.211/192.168.50.211:2181   org.apache.zookeeper.ClientCnxn$SendThread.primeConnection(ClientCnxn.java:908) Socket connection established to 192.168.50.211/192.168.50.211:2181, initiating session   org.apache.zookeeper.ClientCnxn$SendThread.run(ClientCnxn.java:1118) Unable to read additional data from server sessionid 0×0, likely server has closed socket, closing socket connection and attempting reconnect   org.apache.zookeeper.ClientCnxn$SendThread.startConnect(ClientCnxn.java:1000) Opening socket connection to server localhost/127.0.0.1:2181    2010-11-15 13:31:56,626 WARN   org.apache.zookeeper.ClientCnxn$SendThread.run(ClientCnxn.java:1120) Session 0×0 for server null, unexpected error, closing socket connection and attempting reconnect

 先写到这里，全面了解Zookeeper不太容易，光看Apache Zookeeper官方的wiki和文档还不能对其有深入的了解，想阅读Zookeeper中的部分源代码。另外，本人目前正在学习和了解ing，欢迎大家与我交流，谢谢。

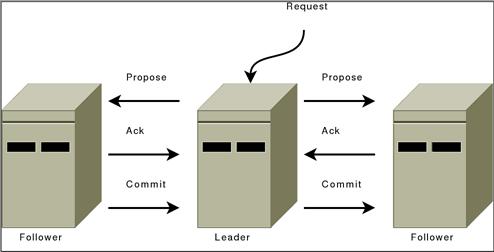
**Apache ZooKeeper入门2**

*1*人收藏此文章， [收藏此文章](javascript:pay_attention(11281,true))

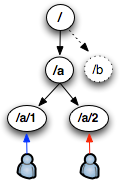
发表于6个月前 , 已有**964**次阅读 共**0**个评论 **1**人收藏此文章

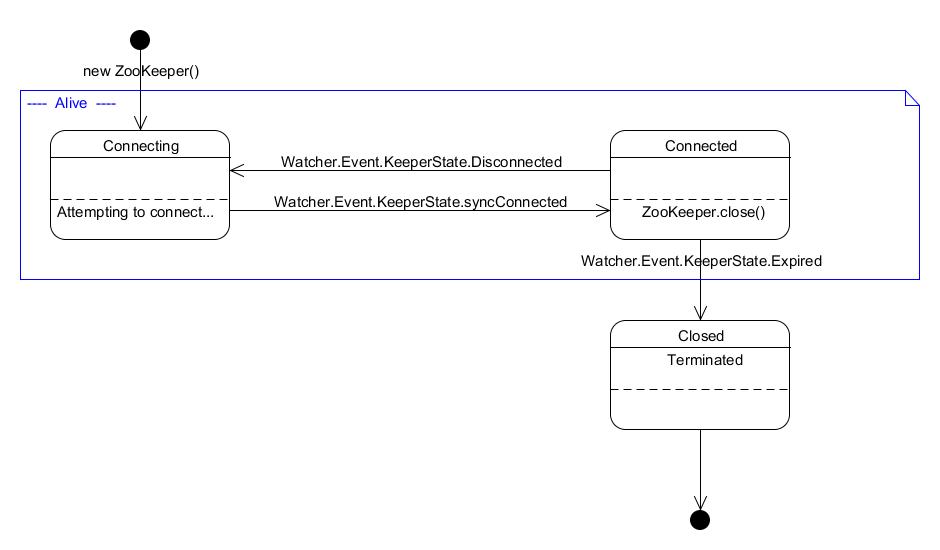
作者: H.E. | 您可以转载, 但必须以超链接形式标明文章原始出处和作者信息及版权声明  网址: <http://www.javabloger.com/article/zookeeper-hapood-apache.html>

   记得在大约在2006年的时候Google出了Chubby来解决分布一致性的问题(distributed consensus problem)，所有集群中的服务器通过Chubby最终选出一个Master Server ，最后这个Master Server来协调工作。简单来说其原理就是：在一个分布式系统中，有一组服务器在运行同样的程序，它们需要确定一个Value，以那个服务器提供的信息为主/为准，当这个服务器经过n/2+1的方式被选出来后，所有的机器上的Process都会被通知到这个服务器就是主服务器 Master服务器，大家以他提供的信息为准。很想知道Google Chubby中的奥妙，可惜人家Google不开源，自家用。

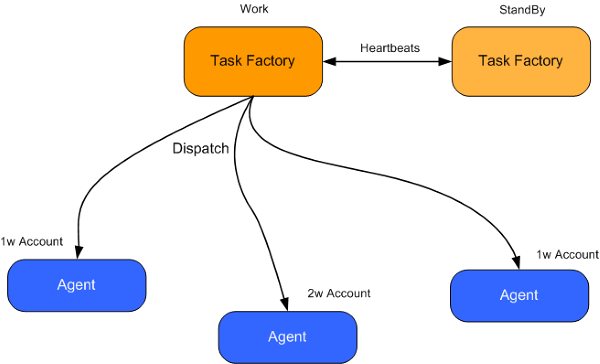
  但是在2009年3年以后沉默已久的Yahoo在Apache上推出了类似的产品ZooKeeper，并且在Google原有Chubby的设计思想上做了一些改进，因为ZooKeeper并不是完全遵循Paxos协议，而是基于自身设计并优化的一个2 phase commit的协议，如图所示： 

  ZooKeeper跟Chubby一样用来存放一些相互协作的信息(Coordination)，这些信息比较小一般不会超过1M，在zookeeper中是以一种hierarchical tree的形式来存放，这些具体的Key/Value信息就store在tree node中，如图所示：



当有事件导致node数据，例如：变更，增加，删除时，Zookeeper就会调用 triggerWatch方法，判断当前的path来是否有对应的监听者(watcher),如果有watcher，会触发其process方法，执行process方法中的业务逻辑，如图所示：[](http://niaklq.bay.livefilestore.com/y1piGSH3Kl8DpmN4UCok8LsEv6Vvv09vpcDySSxklC8p6N6WHvCjAbl22xLYtI9s7wKzH_ar62U106hmIk7ugx20yu8Jta796HM/zookeeper-state.jpg?psid=1)

**应用实例**    ZooKeeper有了上述的这些用途，让我们设想一下，在一个分布式系统中有这这样的一个应用：      2个任务工厂(Task Factory)一主一从，如果从的发现主的死了以后，从的就开始工作，他的工作就是向下面很多台代理(Agent)发送指令，让每台代理(Agent)获得不同的账户进行分布式并行计算，而每台代理(Agent)中将分配很多帐号，如果其中一台代理(Agent)死掉了，那么这台死掉的代理上的账户就不会继续工作了。 上述，**出现了3个最主要的问题**：     1.Task Factory 主/从一致性的问题     2.Task Factory 主/从心跳如何用简单+稳定 或者2者折中的方式实现。     3.一台代理(Agent)死掉了以后，一部分的账户就无法继续工作，需要通知所有在线的代理(Agent)重新分配一次帐号。

怕文字阐述的不够清楚，画了系统中的Task Factory和Agent的大概系统关系，如图所示：  [](http://public.bay.livefilestore.com/y1ppti4VmhXt7tDOrU4fJQ477YPpB3Aezw8KKfo9mPs1meXL6TH-kIJyqNwajoRn5XXUQ56J7AFeXeNOEKaJ3uwOA/zookeeper-TaskFactory-Agent.png?psid=1)

OK，让我们想想ZooKeeper是不是能帮助我们去解决目前遇到的这3个最主要的问题呢？ **解决思路** 1. 任务工厂Task Factory都连接到ZooKeeper上，创建节点，设置对这个节点进行监控，监控方法例如：     event= new WatchedEvent(EventType.NodeDeleted, KeeperState.SyncConnected, "/TaskFactory");    这个方法的意思就是只要Task Factory与zookeeper断开连接后，这个节点就会被自动删除。

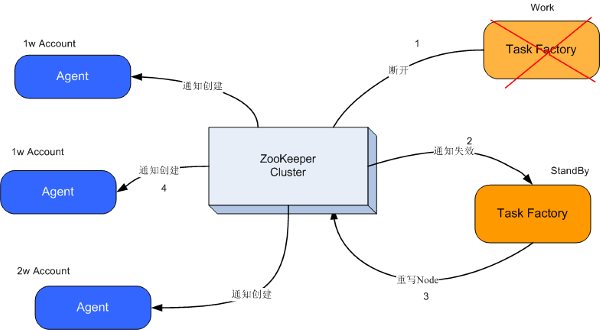
2.原来主的任务工厂断开了TCP连接，这个被创建的/TaskFactory节点就不存在了，而且另外一个连接在上面的Task Factory可以立刻收到这个事件(Event)，知道这个节点不存在了，也就是说主TaskFactory死了。

3.接下来另外一个活着的TaskFactory会再次创建/TaskFactory节点，并且写入自己的ip到znode里面，作为新的标记。

4.此时Agents也会知道主的TaskFactory不工作了，为了防止系统中大量的抛出异常，他们将会先把自己手上的事情做完，然后挂起，等待收到Zookeeper上重新创建一个/TaskFactory节点，收到 EventType.NodeCreated 类型的事件将会继续工作。

5.原来从的TaskFactory 将自己变成一个主TaskFactory，当系统管理员启动原来死掉的主的TaskFactory，世界又恢复平静了。

6.如果一台代理死掉，其他代理他们将会先把自己手上的事情做完，然后挂起，向TaskFactory发送请求，TaskFactory会重新分配(sharding)帐户到每个Agent上了，继续工作。

上述内容，大致如图所示： [](http://public.bay.livefilestore.com/y1phQrSqs4W52GO5ZlfV8isgESnMoYYytosj7mYfwAeBa5RpOAYKmV6JwuERkSTmE_IPJQu37NEF0_miQp4LlnkaQ/zookeeper-TaskFactory-Agent2.png?psid=1)

**口水：** 1.以上内容说的还不够好，希望能和有经验的同学们交流一下，说说你们在大规模计算中是如何解决分布式一致性的问题，谢谢

2.大量了很多文章，最后的确是看了Hbase部分和zookeeper的源代码 有了点启发。 <http://hbase.apache.org/docs/r0.89.20100726/xref/org/apache/hadoop/hbase/master/ZKMasterAddressWatcher.html>

**ookeeper分布式安装手册**

*1*人收藏此文章， [收藏此文章](javascript:pay_attention(11282,true))

发表于6个月前 , 已有**831**次阅读 共**0**个评论 **1**人收藏此文章

**一、安装准备**

**1、下载zookeeper-3.3.1，**

**地址：**[**http://www.apache.org/dist/hadoop/zookeeper/zookeeper-3.3.1/**](http://www.apache.org/dist/hadoop/zookeeper/zookeeper-3.3.1/)

**2、JDK版本：jdk-6u20-linux-i586.bin**

**3、操作系统：Linux**

**4、默认前提是安装完hadoop 0.20.2版本：**

**192.168.3.131 namenode**

**192.168.3.132 datanode**

**192.168.3.133 datanode**

**二、操作步骤（默认在namenode上进行）**

**1、拷贝以上文件到Linux的“/usr/”目录下。同时新建目录“/zookeeper-3.3.1”。**

**2、安装JDK，此步省略...**

**3、解压zookeeper到/zookeeper-3.3.1目录下。tar -zxvf zookeeper-3.3.1.tar.gz -C /**

**zookeeper-3.3.1**

**4、将“/zookeeper-3.3.1/conf”目录下zoo\_sample.cfg修改名称为“zoo.cfg”**

**5、打开zoo.cfg文件，修改配置如下：**

**dataDir=/usr/zookeeper-3.3.1/data**

**dataLogDir=/usr/zookeeper-3.3.1/log**

**clientPort=2181** **initLimit=10** **syncLimit=5** **tickTime=2000** **server.1=192.168.3.131:2888:3888** **server.2=192.168.3.132:2888:3888** **server.3=192.168.3.133:2888:3888**

**6、创建dataDir参数指定的目录(这里指的是“/zookeeper-3.3.1/data”)，并在目录下创建文件，**

**命名为“myid”。**

**7、编辑“myid”文件，并在对应的IP的机器上输入对应的编号。如在192.168.3.131上，**

**“myid”文件内容就是1，在192.168.3.132上，内容就是2**

**8、在profile.d下面生成zookeeper.sh文件设置如下：**

**# zookeeper path**

**ZOOKEEPER=/usr/zookeeper-3.3.2**

**PATH=$PATH:$ZOOKEEPER/bin**

**export PATH**

**然后应用  . ./zookeeper.sh**

**9、将 “/usr/zookeeper-3.3.1” 目录分别拷贝到192.168.3.132和192.168.3.133下。**

**同时修改对应的“myid”文件内容。**

**10、至此，安装步骤结束，接下来启动zookeeper。**

**三、启动zookeeper**

**1、在三台机器上分别执行shell脚本。“sh /jz/zookeeper-3.3.1/bin/zkServer.sh start”**

**2.启动客户端脚本：“zkCli.sh -server 192.168.1.132:2181”**

**3、执行完成之后输入“jps”命令，查看进程如下：**

**namenode上显示如下：**

**29844 JobTracker** **29583 NameNode** **31477 HMaster** **29762 SecondaryNameNode** **32356 Jps** **31376 HQuorumPeer**

**datanode：**

**16812 DataNode** **17032 HRegionServer** **17752 HQuorumPeer** **16921 TaskTracker** **18461 Jps**

**3、通过输入“sh /jz/zookeeper-3.3.1/bin/zkServer.sh status”检查是否启动，**

**一般返回内容如下：**

**leader：**

**JMX enabled by default** **Using config: /jz/zookeeper-3.3.1/bin/../conf/zoo.cfg** **Mode: leader**

**follower：**

**JMX enabled by default** **Using config: /jz/zookeeper-3.3.1/bin/../conf/zoo.cfg** **Mode: follower**

**4、通过在控制台输入命令检查集群zookeeper启动状态。**

**命令如“echo ruok | nc 192.168.3.131 2181”，**

**控制台输出“imok”**

**四、注意事项**

**通过shell脚本在每个机器上启动zookeeper的时候，**

**可能会显示错误信息“Cannot open channel to X at election address”。**

**这是由于zoo.cfg文件中指定的其他zookeeper服务找不到所导致。**

**所有机器的zookeeper服务启动之后该错误提示将会消失。**

**分布式服务框架 Zookeeper -- 管理分布式环境中的数据**

*1*人收藏此文章， [收藏此文章](javascript:pay_attention(11283,true))发表于6个月前 , 已有**876**次阅读 共[**0**](http://my.oschina.net/cmffire/blog/11283#comments)个评论 **1**人收藏此文章

**安装和配置详解**

本文介绍的 Zookeeper 是以 3.2.2 这个稳定版本为基础，最新的版本可以通过官网 <http://hadoop.apache.org/zookeeper/>来获取，Zookeeper 的安装非常简单，下面将从单机模式和集群模式两个方面介绍 Zookeeper 的安装和配置。

**单机模式**

单机安装非常简单，只要获取到 Zookeeper 的压缩包并解压到某个目录如：/home/zookeeper-3.2.2 下，Zookeeper 的启动脚本在 bin 目录下，Linux 下的启动脚本是 zkServer.sh，在 3.2.2 这个版本 Zookeeper 没有提供 windows 下的启动脚本，所以要想在 windows 下启动 Zookeeper 要自己手工写一个，如清单 1 所示：

**清单 1. Windows 下 Zookeeper 启动脚本**

|  |
| --- |
| setlocal set ZOOCFGDIR=%~dp0%..\conf set ZOO\_LOG\_DIR=%~dp0%.. set ZOO\_LOG4J\_PROP=INFO,CONSOLE set CLASSPATH=%ZOOCFGDIR% set CLASSPATH=%~dp0..\\*;%~dp0..\lib\\*;%CLASSPATH% set CLASSPATH=%~dp0..\build\classes;%~dp0..\build\lib\\*;%CLASSPATH% set ZOOCFG=%ZOOCFGDIR%\zoo.cfg set ZOOMAIN=org.apache.zookeeper.server.ZooKeeperServerMain java "-Dzookeeper.log.dir=%ZOO\_LOG\_DIR%" "-Dzookeeper.root.logger=%ZOO\_LOG4J\_PROP%" -cp "%CLASSPATH%" %ZOOMAIN% "%ZOOCFG%" %\* |-------10--------20--------30--------40--------50--------60--------70--------80--------9| |-------- XML error: The previous line is longer than the max of 90 characters ---------| endlocal |

在你执行启动脚本之前，还有几个基本的配置项需要配置一下，Zookeeper 的配置文件在 conf 目录下，这个目录下有 zoo\_sample.cfg 和 log4j.properties，你需要做的就是将 zoo\_sample.cfg 改名为 zoo.cfg，因为 Zookeeper 在启动时会找这个文件作为默认配置文件。下面详细介绍一下，这个配置文件中各个配置项的意义。

|  |
| --- |
| tickTime=2000 dataDir=D:/devtools/zookeeper-3.2.2/build clientPort=2181 |

* tickTime：这个时间是作为 Zookeeper 服务器之间或客户端与服务器之间维持心跳的时间间隔，也就是每个 tickTime 时间就会发送一个心跳。
* dataDir：顾名思义就是 Zookeeper 保存数据的目录，默认情况下，Zookeeper 将写数据的日志文件也保存在这个目录里。
* clientPort：这个端口就是客户端连接 Zookeeper 服务器的端口，Zookeeper 会监听这个端口，接受客户端的访问请求。

当这些配置项配置好后，你现在就可以启动 Zookeeper 了，启动后要检查 Zookeeper 是否已经在服务，可以通过 netstat – ano 命令查看是否有你配置的 clientPort 端口号在监听服务。

**集群模式**

Zookeeper 不仅可以单机提供服务，同时也支持多机组成集群来提供服务。实际上 Zookeeper 还支持另外一种伪集群的方式，也就是可以在一台物理机上运行多个 Zookeeper 实例，下面将介绍集群模式的安装和配置。

Zookeeper 的集群模式的安装和配置也不是很复杂，所要做的就是增加几个配置项。集群模式除了上面的三个配置项还要增加下面几个配置项：

|  |
| --- |
| initLimit=5 syncLimit=2 server.1=192.168.211.1:2888:3888 server.2=192.168.211.2:2888:3888 |

* initLimit：这个配置项是用来配置 Zookeeper 接受客户端（这里所说的客户端不是用户连接 Zookeeper 服务器的客户端，而是 Zookeeper 服务器集群中连接到 Leader 的 Follower 服务器）初始化连接时最长能忍受多少个心跳时间间隔数。当已经超过 10 个心跳的时间（也就是 tickTime）长度后 Zookeeper 服务器还没有收到客户端的返回信息，那么表明这个客户端连接失败。总的时间长度就是 5\*2000=10 秒
* syncLimit：这个配置项标识 Leader 与 Follower 之间发送消息，请求和应答时间长度，最长不能超过多少个 tickTime 的时间长度，总的时间长度就是 2\*2000=4 秒
* server.A=B：C：D：其中 A 是一个数字，表示这个是第几号服务器；B 是这个服务器的 ip 地址；C 表示的是这个服务器与集群中的 Leader 服务器交换信息的端口；D 表示的是万一集群中的 Leader 服务器挂了，需要一个端口来重新进行选举，选出一个新的 Leader，而这个端口就是用来执行选举时服务器相互通信的端口。如果是伪集群的配置方式，由于 B 都是一样，所以不同的 Zookeeper 实例通信端口号不能一样，所以要给它们分配不同的端口号。

除了修改 zoo.cfg 配置文件，集群模式下还要配置一个文件 myid，这个文件在 dataDir 目录下，这个文件里面就有一个数据就是 A 的值，Zookeeper 启动时会读取这个文件，拿到里面的数据与 zoo.cfg 里面的配置信息比较从而判断到底是那个 server。

**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

[回页首](http://www.ibm.com/developerworks/cn/opensource/os-cn-zookeeper/#ibm-pcon)

**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** | **方法名** |
| **方法功能描述** | **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%5B%5D,%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) | [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%5B%5D,%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 | 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) | [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 | 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) | [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 | 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) | 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 | 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) | [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) |

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**

|  |
| --- |
| // 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); |-------10--------20--------30--------40--------50--------60--------70--------80--------9| |-------- XML error: The previous line is longer than the max of 90 characters ---------| // 创建一个子目录节点 zk.create("/testRootPath/testChildPathOne", "testChildDataOne".getBytes(), Ids.OPEN\_ACL\_UNSAFE,CreateMode.PERSISTENT); |-------10--------20--------30--------40--------50--------60--------70--------80--------9| |-------- XML error: The previous line is longer than the max of 90 characters ---------| 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); |-------10--------20--------30--------40--------50--------60--------70--------80--------9| |-------- XML error: The previous line is longer than the max of 90 characters ---------| 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：

|  |
| --- |
| 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。

[回页首](http://www.ibm.com/developerworks/cn/opensource/os-cn-zookeeper/#ibm-pcon)

**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）**

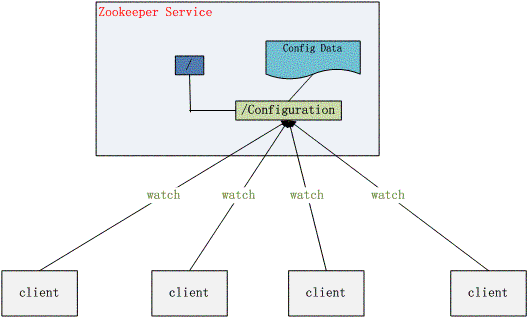
分布式应用中，通常需要有一套完整的命名规则，既能够产生唯一的名称又便于人识别和记住，通常情况下用树形的名称结构是一个理想的选择，树形的名称结构是一个有层次的目录结构，既对人友好又不会重复。说到这里你可能想到了 JNDI，没错 Zookeeper 的 Name Service 与 JNDI 能够完成的功能是差不多的，它们都是将有层次的目录结构关联到一定资源上，但是 Zookeeper 的 Name Service 更加是广泛意义上的关联，也许你并不需要将名称关联到特定资源上，你可能只需要一个不会重复名称，就像数据库中产生一个唯一的数字主键一样。

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**   


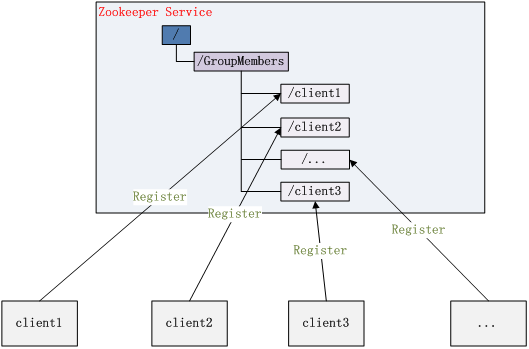
**集群管理（Group Membership）**

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**   


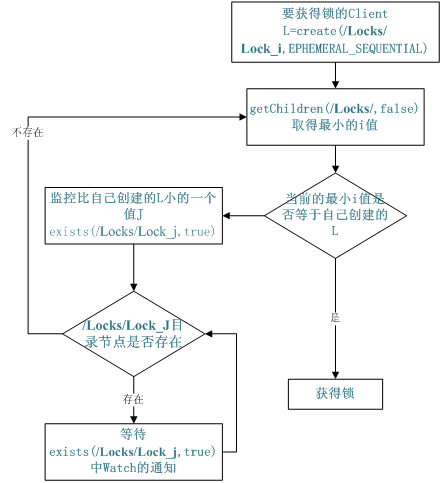
这部分的示例代码如下，完整的代码请看附件：

**清单 3. Leader Election 关键代码**

|  |
| --- |
| 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); |-------10--------20--------30--------40--------50--------60--------70--------80--------9| |-------- XML error: The previous line is longer than the max of 90 characters ---------| } 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**

|  |
| --- |
| 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**

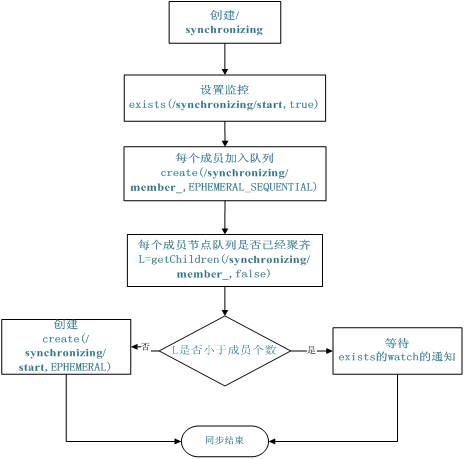
Zookeeper 可以处理两种类型的队列：

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**

|  |
| --- |
| void addQueue() throws KeeperException, InterruptedException{ zk.exists(root + "/start",true); zk.create(root + "/" + name, new byte[0], Ids.OPEN\_ACL\_UNSAFE,CreateMode.EPHEMERAL\_SEQUENTIAL); |-------10--------20--------30--------40--------50--------60--------70--------80--------9| |-------- XML error: The previous line is longer than the max of 90 characters ---------| 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); |-------10--------20--------30--------40--------50--------60--------70--------80--------9| |-------- XML error: The previous line is longer than the max of 90 characters ---------| } } } |

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：

|  |
| --- |
| public void process(WatchedEvent event) { if(event.getPath().equals(root + "/start") && event.getType() == Event.EventType.NodeCreated){ |-------10--------20--------30--------40--------50--------60--------70--------80--------9| |-------- XML error: The previous line is longer than the max of 90 characters ---------| 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**

|  |
| --- |
| 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; } |