第四十九章：zookeeper分布式协调工具

**一、分布式协调技术概述：**

**二、zookeeper概述：**

**三、zookeeper提供的功能：**

**四、zookeeper工作角色：**

**五、ZooKeeper搭建Hadoop的HA集群：**

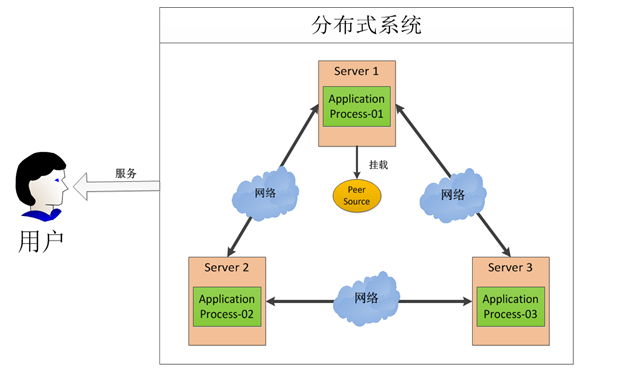
**六、案例：搭建hadoop2.7.6结合zookeeper-3.4.10完全分布式存储集群：**

**一、分布式协调技术概述：**

**概述：**在给大家介绍ZooKeeper之前先来给大家介绍一种技术——分布式协调技术。那么什么是分布式协调技术？那么我来告诉大家，其实分布式协调技术主要用来解决分布式环境当中多个进程之间的同步控制，让他们有序的去访问某种临界资源；

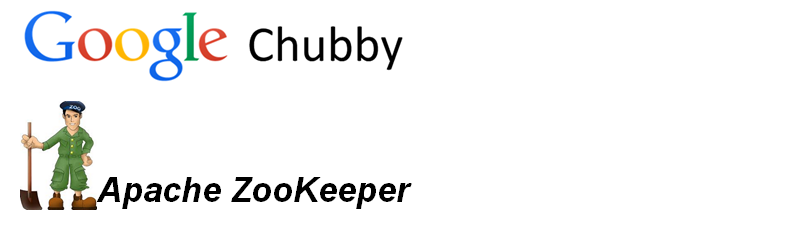
图中有三台机器，每台机器各跑一个应用程序。然后我们将这三台机器通过网络将其连接起来，构成一个系统来为用户提供服务，对用户来说这个系统的架构是非透明的，他感觉不到我这个系统是一个什么样的架构。那么我们就可以把这种系统称作一个分布式系统。

　　那我们接下来再分析一下，在这个分布式系统中如何对进程进行调度，我假设在第一台机器上挂载了一个资源，然后这三个物理分布的进程都要竞争这个资源，但我们又不希望他们同时进行访问，这时候我们就需要一个协调器，来让他们有序的来访问这个资源；



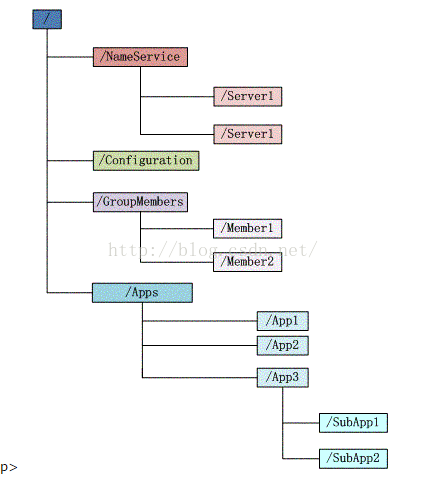
**二、zookeeper概述：**

**概述：**ZooKeeper 是一个分布式的，开放源码的分布式应用程序协调服务，是 Google 的 Chubby 一个开源的实现。它提供了简单原始的功能，分布式应用可以基于它实现更高级的服务，比如分布式同步，配置管理，集群管理，命名管理，队列管理。它被设计为易于编程，使用文件系统目录树作为数据模型。我们设计 ZooKeeper 的目的是为了减轻分布式应用程序所承担的协调任务 ZooKeeper 是集群的管理者，监视着集群中各节点的状态，根据节点提交的反馈进行下 一步合理的操作。最终，将简单易用的接口和功能稳定，性能高效的系统提供给用户；

****

**三、zookeeper提供的功能：**  
**1.文件系统**

每个子目录项如 NameService 都被称作为znode，和文件系统一样，我们能够自由的增加、删除znode，在一个znode下增加、删除子znode，有四种类型的znode：   
PERSISTENT-持久化目录节点 ：客户端与zookeeper断开连接后，该节点依旧存在；  
PERSISTENT\_SEQUENTIAL-持久化顺序编号目录节点 ：客户端与zookeeper断开连接后，该节点依旧存在，只是Zookeeper给该节点名称进行顺序编号 ；  
EPHEMERAL-临时目录节点 ：客户端与zookeeper断开连接后，该节点被删除 ；  
EPHEMERAL\_SEQUENTIAL-临时顺序编号目录节点 ：客户端与zookeeper断开连接后，该节点被删除，只是Zookeeper给该节点名称进行顺序编号；

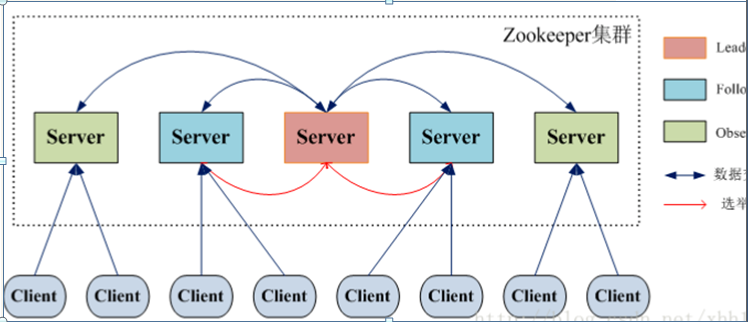
  
**2.通知机制**：客户端注册监听它关心的目录节点，当目录节点发生变化（数据改变、被删除、子目录节点增加删除）时，zookeeper会通知客户端；

**四、zookeeper工作角色：**

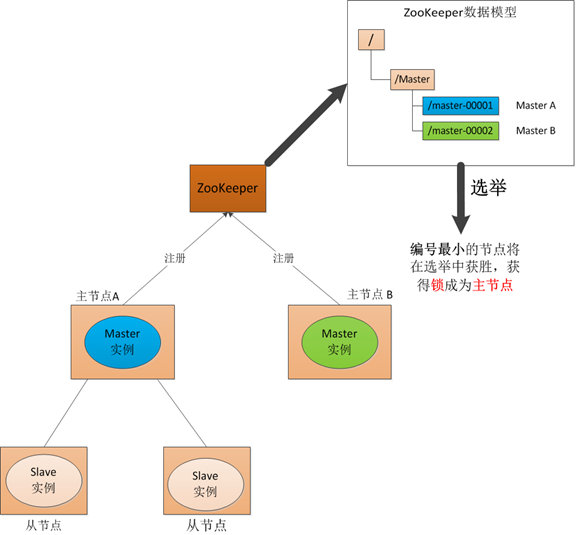
**角色：**

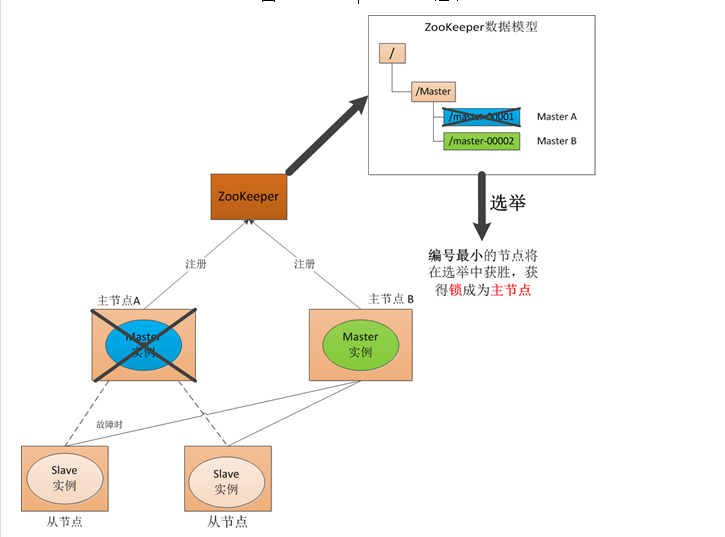


**工作流程：**



**选举机制：**



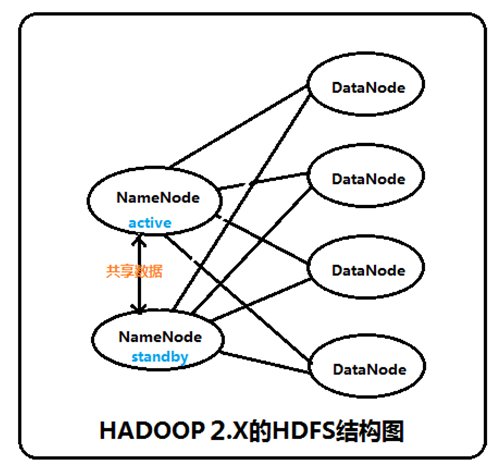


**五、ZooKeeper搭建Hadoop的HA集群：**

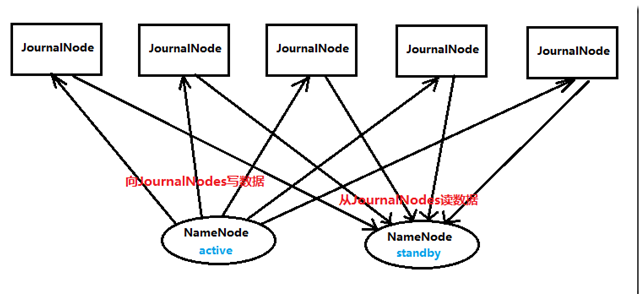
**HA--High Available高可用概述：**在Hadoop 2.0之前,在HDFS 集群中NameNode 存在单点故障 (SPOF：A Single Point of Failure)。 对于只有一个 NameNode 的集群，如果 NameNode 机器出现故障（比如宕机或是软件、硬件升级），那么整个集群将无法使用；

**实现原理：**HDFS 的 HA 功能通过配置 Active/Standby 两个 NameNodes 实现在集群中对 NameNode 的 热备来解决上述问题。如果出现故障，如机器崩溃或机器需要升级维护，这时可通过此种方式将 NameNode 很快的切换到另外一台机器；

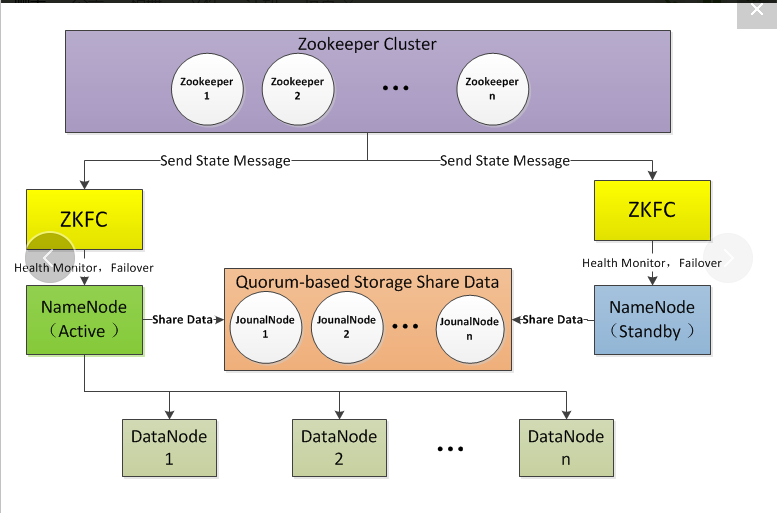
　　在一个典型的 HDFS(HA) 集群中，使用两台单独的机器配置为 NameNodes 。在任何时间点， 确保 NameNodes 中只有一个处于 Active 状态，其他的处在 Standby 状态。其中 ActiveNameNode 负责集群中的所有客户端操作，StandbyNameNode 仅仅充当备机，保证一 旦 ActiveNameNode 出现问题能够快速切换；

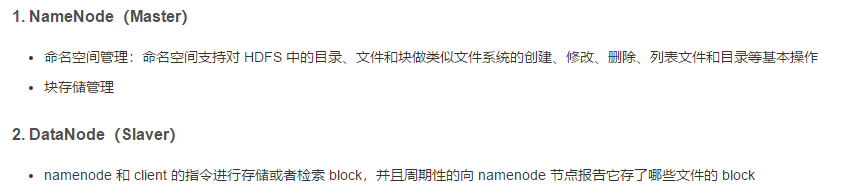


为了能够实时同步 Active 和 Standby 两个 NameNode 的元数据信息（实际上 editlog），需提 供一个共享存储系统，可以是 NFS、QJM（Quorum Journal Manager）或者 Zookeeper，Active Namenode 将数据写入共享存储系统，而 Standby 监听该系统，一旦发现有新数据写入，则 读取这些数据，并加载到自己内存中，以保证自己内存状态与 Active NameNode 保持基本一致，如此这般，在紧急情况下 standby 便可快速切为 active namenode；

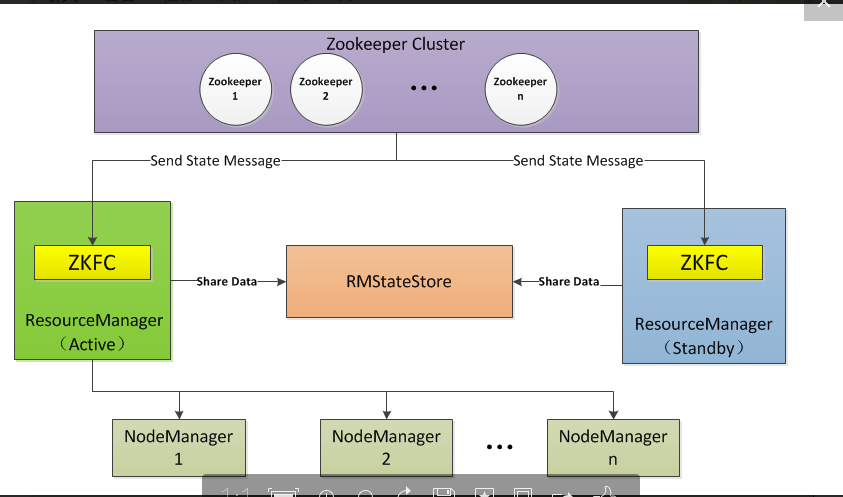


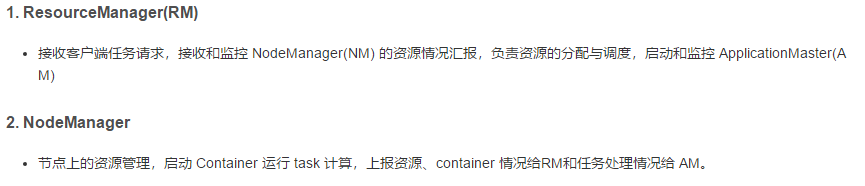
**Zookeeper+Namenode**



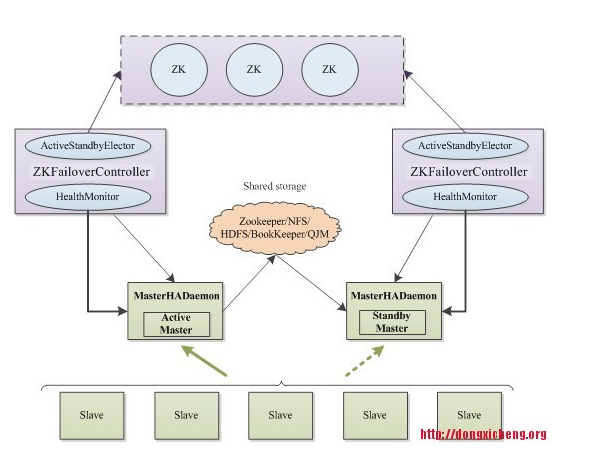


Zookeeper+MapReduce（Yarn）：





Zookeeper+Hadoop ha总结：



**六、案例：搭建hadoop2.7.6结合zookeeper-3.4.10完全分布式存储集群：**

**案例环境：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 系统类型 | IP地址 | 主机名 | 所需软件 | 角色 |
| Centos 7.4 1708 64bit | 192.168.100.101 | master1 | jdk-8u171-linux-x64.tar.gz  zookeeper-3.4.10.tar.gz  hadoop-2.7.6.tar.gz | hdfs：namenode  zookeeper：投票 |
| Centos 7.4 1708 64bit | 192.168.100.102 | master2 | jdk-8u171-linux-x64.tar.gz  zookeeper-3.4.10.tar.gz  hadoop-2.7.6.tar.gz | hdfs：namenode  zookeeper：投票 |
| Centos 7.4 1708 64bit | 192.168.100.103 | slave1 | jdk-8u171-linux-x64.tar.gz  zookeeper-3.4.10.tar.gz  hadoop-2.7.6.tar.gz | hdfs：datanode  zookeeper：投票 |
| Centos 7.4 1708 64bit | 192.168.100.104 | slave2 | jdk-8u171-linux-x64.tar.gz  zookeeper-3.4.10.tar.gz  hadoop-2.7.6.tar.gz | hdfs：datanode  zookeeper：observer |

**角色进程分布：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 主机名 | hdfs角色 | Map-reduce（yarn）角色 | Zookeeper角色 | JNode角色 |
| master1 | namenode | NodeManager | QuorumPeerMain  DFSZKFailoverController  JobHistoryServer | JournalNode |
| master2 | namnenode | NodeManager | QuorumPeerMain  DFSZKFailoverController  JobHistoryServer | JournalNode |
| slave1 | datanode | NodeManager  ResourceManager | QuorumPeerMain | JournalNode |
| slave2 | datanode | NodeManager  ResourceManager | QuorumPeerMain | JournalNode |

**案例步骤（保证节点间时间一致）：**

* 配置所有节点间的域名解析及创建用户（所有节点配置相同，在此列举master1节点配置）：
* 配置master1节点远程管理：
* 配置master2节点远程管理：
* 在所有节点安装JDK环境（所有节点配置相同，在此列举master1节点配置）：
* 在master1节点安装zookeeper服务：
* 配置master1节点的zookerper服务：
* 在master2、slave1、slave2准备zookeeper目录并设置环境变量，同步master1的配置文件：
* 配置所有节点的zookeeper的id号码并且设置slave2为observer：
* 启动所有节点的zookeeper服务并查看服务角色状态：
* 在master1节点访问zookeeper管理客户端控制台测试znode管理：
* 在master1节点安装hadoop服务；
* 配置master1节点的Hadoop服务：
* 在master2、slave1、slave2节点准备hadoop目录，同步master1的配置：
* 在所有节点初始化journalnade服务，自动创建目录：
* 在master1节点格式化namenode节点：
* 启动所有节点的对应namenode或datanode服务：
* 访问web节点，查看hadoop集群状态；
* 手动切换状态，在各个NameNode节点上启动DFSZK Failover Controller，并且强制设置master1作为active节点；
* 测试Hadoop分布式集群数据存储：
* 在master1节点初始化ha集群在zookeeper中的状态：
* 在master1节点启动mapreduce-yarn进程：
* 在master1节点启动historyserver进程：
* 在master1节点开启集群的DFSZKFailoverController进程（按照以下方法）；
* 在slave1、slave2节点启动resourcemanager客户端任务进程，并且验证最终集群进程分布状态：
* 集群Namenode节点高可用测试：
* 集群resourceManager节点高可用测试：
* 错误总结：
* **配置所有节点间的域名解析及创建用户（所有节点配置相同，在此列举master1节点配置）：**

[root@master1 ~]# cat <<END >>/etc/hosts

192.168.100.101 master1

192.168.100.102 master2

192.168.100.103 slave1

192.168.100.104 slave2

END

[root@master1 ~]# useradd hadoop

[root@master1 ~]# echo "hadoop" |passwd --stdin hadoop

[root@master1 ~]# yum -y install psmisc ##切换节点用命令fuser

* **配置master1节点远程管理：**

[root@master1 ~]# su - hadoop

上一次登录：四 5月 31 01:54:26 CST 2018pts/0 上

[hadoop@master1 ~]$ ssh-keygen -t rsa

[hadoop@master1 ~]$ ssh-copy-id hadoop@192.168.100.101

[hadoop@master1 ~]$ ssh-copy-id hadoop@192.168.100.102

[hadoop@master1 ~]$ ssh-copy-id hadoop@192.168.100.103

[hadoop@master1 ~]$ ssh-copy-id hadoop@192.168.100.104

[hadoop@master1 ~]$ ssh hadoop@master1 ##远程连接slave节点，进行确认key值文件，不然在启动hadoop时，会出现key的问题导致无法启动

[hadoop@master1 ~]$ ssh hadoop@master2

[hadoop@master1 ~]$ ssh hadoop@slave1

[hadoop@master1 ~]$ ssh hadoop@slave2

* **配置master2节点远程管理：**

[root@master2 ~]# su - hadoop

上一次登录：四 5月 31 01:54:26 CST 2018pts/0 上

[hadoop@master2 ~]$ ssh-keygen -t rsa

[hadoop@master2 ~]$ ssh-copy-id hadoop@192.168.100.101

[hadoop@master2 ~]$ ssh-copy-id hadoop@192.168.100.102

[hadoop@master2 ~]$ ssh-copy-id hadoop@192.168.100.103

[hadoop@master2 ~]$ ssh-copy-id hadoop@192.168.100.104

[hadoop@master2 ~]$ ssh hadoop@master1

[hadoop@master2 ~]$ ssh hadoop@master2

[hadoop@master2 ~]$ ssh hadoop@slave1

[hadoop@master2 ~]$ ssh hadoop@slave2

* **在所有节点安装JDK环境（所有节点配置相同，在此列举master1节点配置）：**

[hadoop@master1 ~]$ exit

[root@master1 ~]# tar zxvf jdk-8u171-linux-x64.tar.gz

[root@master1 ~]# mv /root/jdk1.8.0\_171/ /usr/local/java/

[root@master1 ~]# ls /usr/local/java/

[root@master1 ~]# cat <<END >>/etc/profile

JAVA\_HOME=/usr/local/java/

JRE\_HOME=\$JAVA\_HOME/jre

CLASS\_PATH=.:\$JAVA\_HOME/lib/dt.jar:\$JAVA\_HOME/lib/tools.jar:\$JAVA\_HOME/lib

PATH=\$PATH:\$JAVA\_HOME/bin:\$JRE\_HOME/bin

export JAVA\_HOME JRE\_HOME CLASS\_PATH PATH

END

[root@master1 ~]# source /etc/profile

[root@master1 ~]# java -version

java version "1.8.0\_171"

* **在master1节点安装zookeeper服务：**

[root@master1 ~]# wget http://mirror.bit.edu.cn/apache/zookeeper/zookeeper-3.4.10/zookeeper-3.4.10.tar.gz

[root@master1 ~]# tar zxvf zookeeper-3.4.10.tar.gz

[root@master1 ~]# mv zookeeper-3.4.10 /usr/local/zookeeper

[root@master1 ~]# ls /usr/local/zookeeper

bin docs NOTICE.txt zookeeper-3.4.10.jar

build.xml ivysettings.xml README\_packaging.txt zookeeper-3.4.10.jar.asc

conf ivy.xml README.txt zookeeper-3.4.10.jar.md5

contrib lib recipes zookeeper-3.4.10.jar.sha1

dist-maven LICENSE.txt src

[root@master1 ~]# chown hadoop:hadoop /usr/local/zookeeper -R

[root@ master ~]# vi /etc/profile

export ZOOKEEPER\_HOME=/usr/local/zookeeper

export PATH=$PATH:$ZOOKEEPER\_HOME/bin

[root@master ~]# source /etc/profile

* **配置master1节点的zookerper服务：**

[root@master1 ~]# su - hadoop

[hadoop@master1 ~]$ mkdir /usr/local/zookeeper/data

[hadoop@master1 ~]$ mv /usr/local/zookeeper/conf/zoo\_sample.cfg /usr/local/zookeeper/conf/zoo.cfg

[hadoop@master1 ~]$ vi /usr/local/zookeeper/conf/zoo.cfg

12 dataDir=/usr/local/zookeeper/data

29 server.1=master1:2888:3888

30 server.2=master2:2888:3888

31 server.3=slave1:2888:3888

32 server.4=slave2:2888:3888:observer

注解：zoo.cfg配置文件详解

* Server.A=B:C:D。

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是这个服务器与集群中的Leader服务器交换信息的端口；

D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。

集群模式下配置一个文件myid，这个文件在dataDir目录下，这个文件里面有一个数据就是A的值，Zookeeper启动时读取此文件，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哪个server；

* tickTime=2000：通信心跳数，Zookeeper服务器心跳时间，单位毫秒；
* initLimit=10：LF初始通信时限，集群中的follower跟随者服务器与leader领导者服务器之间初始连接时能容忍的最多心跳数（tickTime的数量），用它来限定集群中的Zookeeper服务器连接到Leader的时限；
* syncLimit=5：跟随者与领导者同步通信时限，集群中Leader与Follower之间的最大响应时间单位，假如响应超过syncLimit \* tickTime，Leader认为Follwer死掉，从服务器列表中删除Follwer。
* dataDir：数据文件目录+数据持久化路径，保存内存数据库快照信息的位置，如果没有其他说明，更新的事务日志也保存到数据库；
* clientPort=2181：客户端连接端口；

[hadoop@master1 ~]$ touch /usr/local/zookeeper/data/myid

[hadoop@master1 ~]$ echo 1 >/usr/local/zookeeper/data/myid

* **在master2、slave1、slave2准备zookeeper目录并设置环境变量，同步master1的配置文件：**

[root@master2 ~]# mkdir /usr/local/zookeeper

[root@master2 ~]# chown hadoop:hadoop /usr/local/zookeeper

[root@master2 ~]# vi /etc/profile

export ZOOKEEPER\_HOME=/usr/local/zookeeper

export PATH=$PATH:$ZOOKEEPER\_HOME/bin

[root@master2 ~]# source /etc/profile

[root@slave1 ~]# mkdir /usr/local/zookeeper

[root@slave1 ~]# chown hadoop:hadoop /usr/local/zookeeper

[root@ slave1 ~]# vi /etc/profile

export ZOOKEEPER\_HOME=/usr/local/zookeeper

export PATH=$PATH:$ZOOKEEPER\_HOME/bin

[root@ slave1 ~]# source /etc/profile

[root@slave2 ~]# mkdir /usr/local/zookeeper

[root@slave2 ~]# chown hadoop:hadoop /usr/local/zookeeper

[root@ slave2 ~]# vi /etc/profile

export ZOOKEEPER\_HOME=/usr/local/zookeeper

export PATH=$PATH:$ZOOKEEPER\_HOME/bin

[root@ slave2 ~]# source /etc/profile

[hadoop@master1 ~]$ scp -r /usr/local/zookeeper/\* hadoop@master2:/usr/local/zookeeper/

[hadoop@master1 ~]$ scp -r /usr/local/zookeeper/\* hadoop@slave1:/usr/local/zookeeper/

[hadoop@master1 ~]$ scp -r /usr/local/zookeeper/\* hadoop@slave2:/usr/local/zookeeper/

* **配置所有节点的zookeeper的id号码并且设置slave2为observer：**

[root@master2 ~]# su - hadoop

[hadoop@master2 ~]$ echo 2 >/usr/local/zookeeper/data/myid

[root@slave1 ~]# su - hadoop

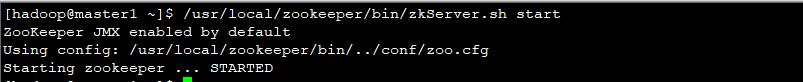
[hadoop@slave1 ~]$ echo 3 >/usr/local/zookeeper/data/myid

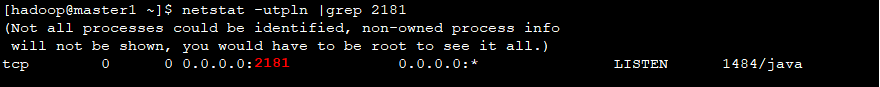
[root@slave2 ~]# su - hadoop

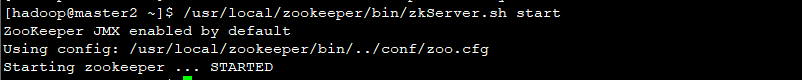
[hadoop@slave2 ~]$ echo 4 >/usr/local/zookeeper/data/myid

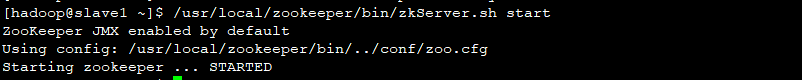
[hadoop@slave2 ~]$ echo peerType=observer >>/usr/local/zookeeper/conf/zoo.cfg

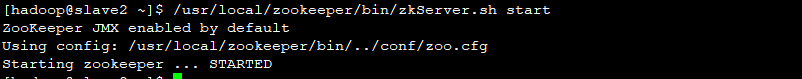
* **启动所有节点的zookeeper服务并查看服务角色状态：**

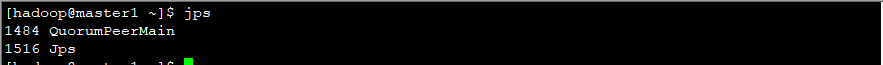






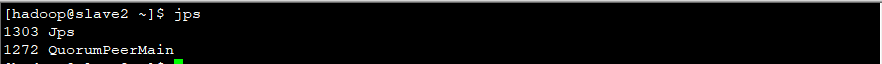


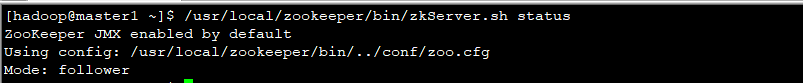


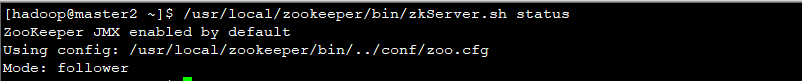


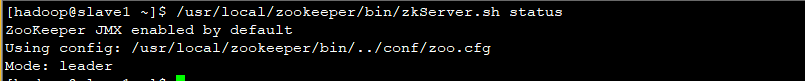


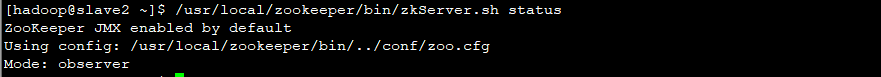




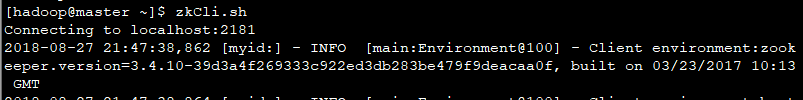




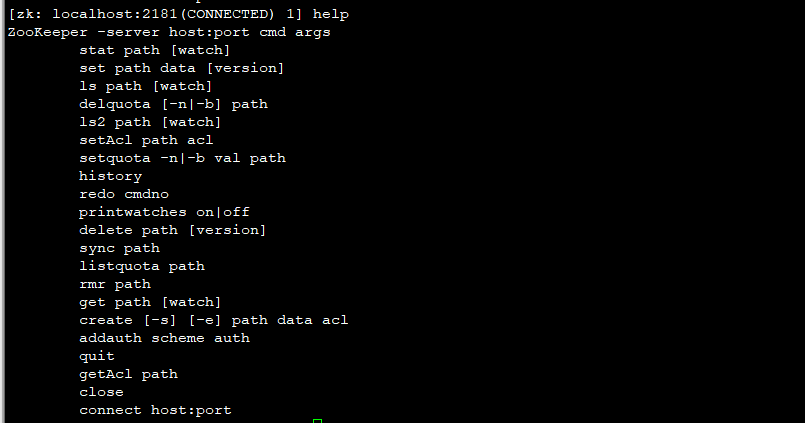




* **在master1节点访问zookeeper管理客户端控制台测试znode管理：**



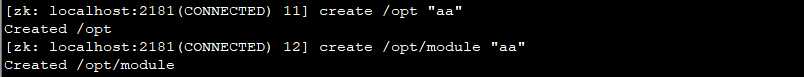
查看帮助：

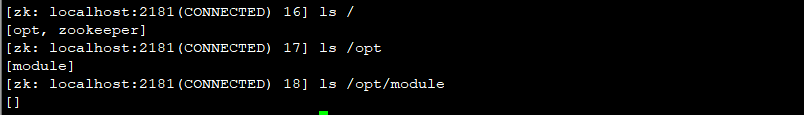


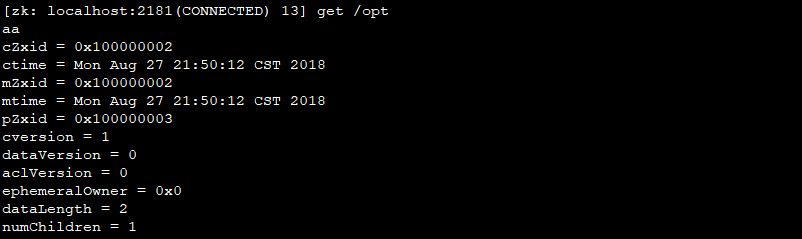
查看根目录节点：

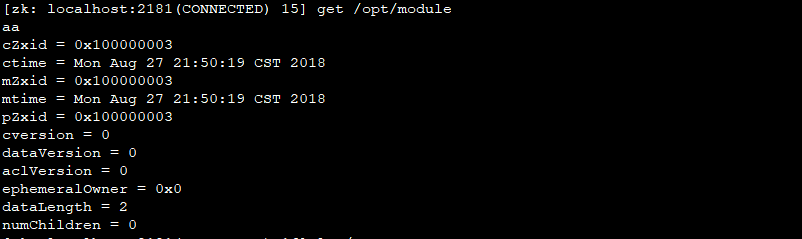


创建节点：

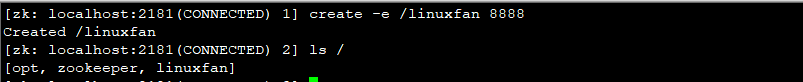


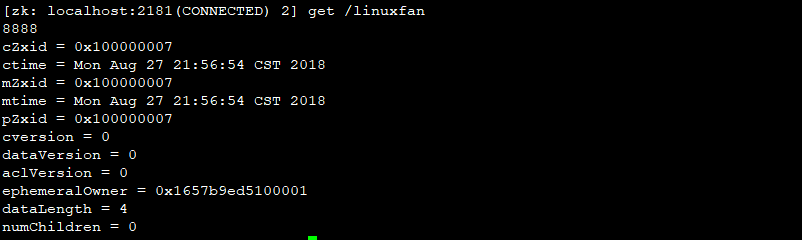




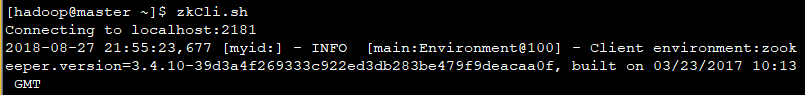


创建临时节点：



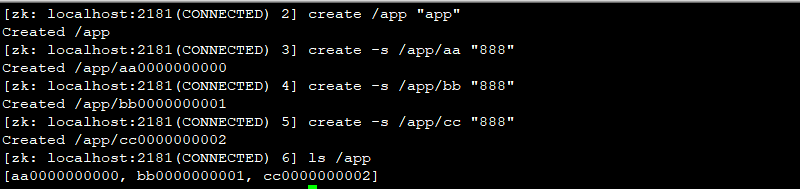


重新进入控制台后，发现临时节点消失：

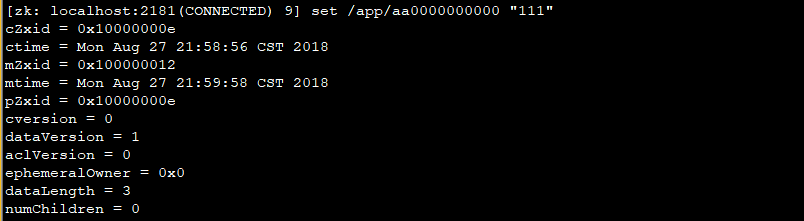


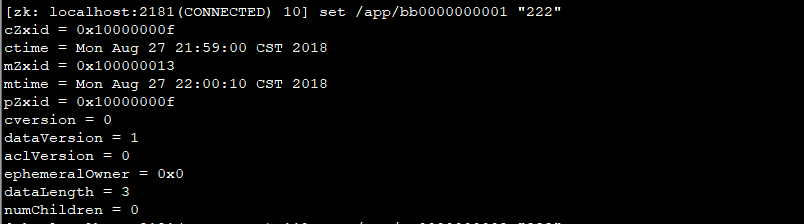


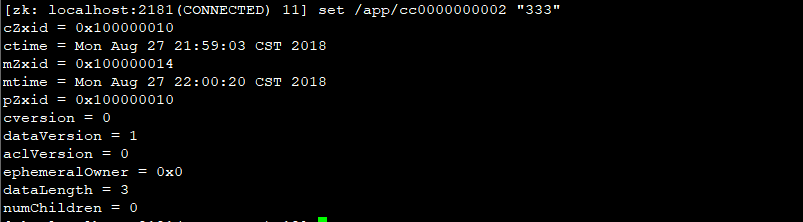
创建带有序号的节点：



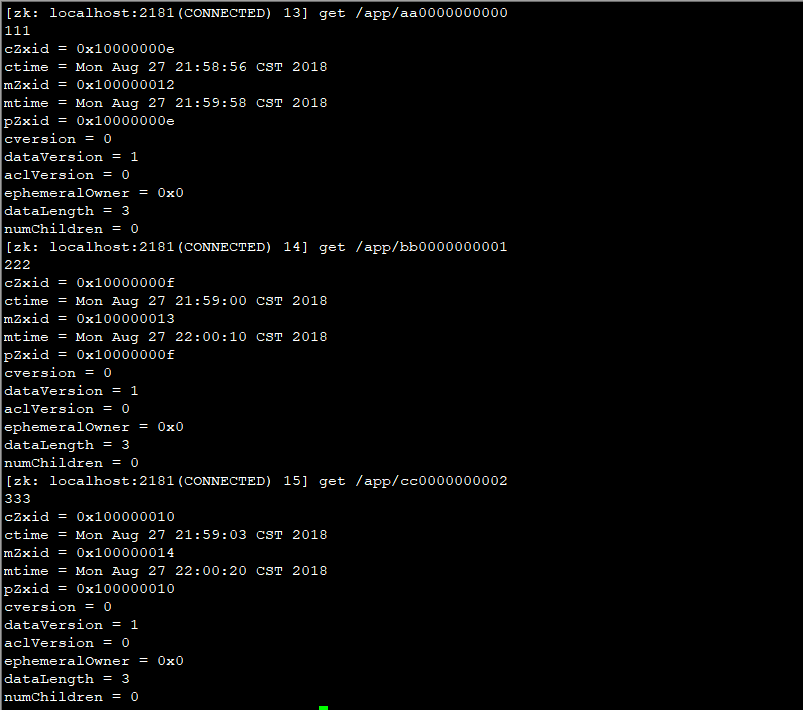
修改节点的值：



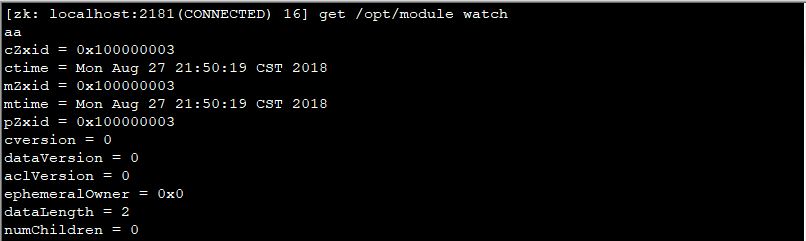




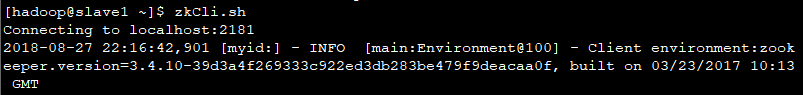


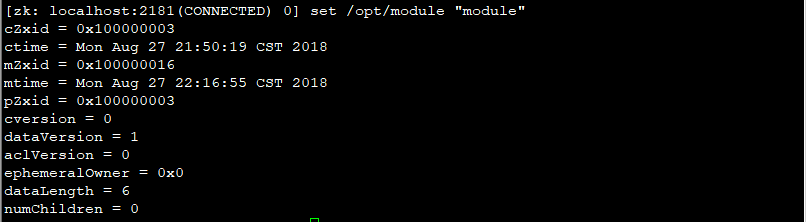


监控节点的变化：

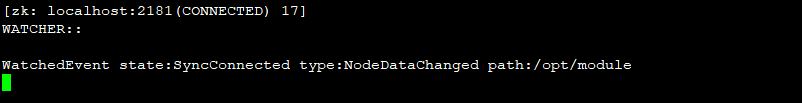


在slave1节点登录，更改节点的值：





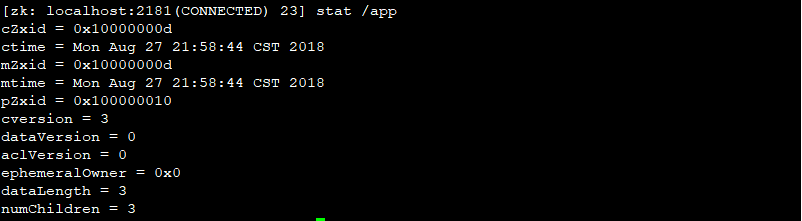
在master1节点验证监控情况：

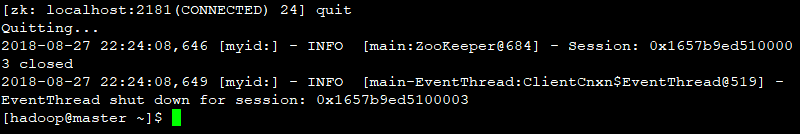


删除单个节点与递归删除整个节点目录：



查看节点状态：





* **在master1节点安装hadoop服务；**

[hadoop@master1 ~]$ exit

[root@master1 ~]# tar zxvf hadoop-2.7.6.tar.gz

[root@master1 ~]# mv /root/hadoop-2.7.6/ /usr/local/hadoop/

[root@master1 ~]# ls /usr/local/hadoop/

bin etc include lib libexec LICENSE.txt NOTICE.txt README.txt sbin share

[root@master1 ~]# cat <<END >>/etc/profile

export HADOOP\_HOME=/usr/local/hadoop/

export PATH=\$PATH:\$HADOOP\_HOME/bin

END

[root@master1 ~]# source /etc/profile

[root@master1 ~]# echo "export JAVA\_HOME=/usr/local/java/" >>/usr/local/hadoop/etc/hadoop/hadoop-env.sh

##设置HDFS存储加载jdk的环境变量

[root@master1 ~]# echo "export JAVA\_HOME=/usr/local/java/" >>/usr/local/hadoop/etc/hadoop/yarn-env.sh

##设置mapreduce的V2版本--YARN加载jdk的环境变量

[root@master1 ~]# mkdir /usr/local/hadoop/name/ ##存放namenode中元数据的位置

[root@master1 ~]# mkdir /usr/local/hadoop/data/ ##存放datanode中的数据目录

[root@master1 ~]# mkdir /usr/local/hadoop/tmp/ ##存放用户临时文件

[root@master1 ~]# mkdir /usr/local/hadoop/var/ ##存放服务动态变化文件

[root@master1 ~]# chown hadoop /usr/local/hadoop/ -R

* **配置master1节点的Hadoop服务：**

[root@master1 ~]# su - hadoop

[hadoop@master1 ~]$ vi /usr/local/hadoop/etc/hadoop/core-site.xml

<configuration>

<!-- 指定运行hadoop后生成临时文件存放位置 -->

<property>

<name>hadoop.tmp.dir</name>

<value>/usr/local/hadoop/tmp</value>

</property>

<!-- 指定默认hadoop集群名称 -->

<property>

<name>fs.defaultFS</name>

<value>hdfs://mycluster</value>

</property>

<!-- 指定ZKFC故障自动切换转移,2181是zookeeper的端口 -->

<property>

<name>ha.zookeeper.quorum</name>

<value>master1:2181,master2:2181,slave1:2181,slave2:2181</value>

</property>

</configuration>

[hadoop@master1 ~]$ vi /usr/local/hadoop/etc/hadoop/hdfs-site.xml

<configuration>

<!-- 指定name元数据存储位置 -->

<property>

<name>dfs.name.dir</name>

<value>/usr/local/hadoop/name</value>

</property>

<!-- 指定data数据存放位置 -->

<property>

<name>dfs.data.dir</name>

<value>/usr/local/hadoop/data</value>

</property>

<!-- 指定hdfs文件副本数量 -->

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

<!-- 开启hdfsweb界面 -->

<property>

<name>dfs.webhdfs.enabled</name>

<value>true</value>

</property>

<!-- 完全分布式集群名称 -->

<property>

<name>dfs.nameservices</name>

<value>mycluster</value>

</property>

<!-- 集群中NameNode节点都有哪些 -->

<property>

<name>dfs.ha.namenodes.mycluster</name>

<value>nn1,nn2</value>

</property>

<!-- nn1的RPC通信地址 -->

<property>

<name>dfs.namenode.rpc-address.mycluster.nn1</name>

<value>master1:8020</value>

</property>

<!-- nn2的RPC通信地址 -->

<property>

<name>dfs.namenode.rpc-address.mycluster.nn2</name>

<value>master2:8020</value>

</property>

<!-- nn1的http通信地址 -->

<property>

<name>dfs.namenode.http-address.mycluster.nn1</name>

<value>master1:50070</value>

</property>

<!-- nn2的http通信地址 -->

<property>

<name>dfs.namenode.http-address.mycluster.nn2</name>

<value>master2:50070</value>

</property>

<!-- 指定NameNode元数据在JournalNode上的存放位置 -->

<property>

<name>dfs.namenode.shared.edits.dir</name>

<value>qjournal://master1:8485;master2:8485;slave1:8485;slave2:8485/mycluster</value>

</property>

<!-- 配置隔离机制，即同一时刻只能有一台服务器对外响应 -->

<property>

<name>dfs.ha.fencing.methods</name>

<value>sshfence</value>

</property>

<!-- 使用隔离机制时需要ssh无秘钥登录-->

<property>

<name>dfs.ha.fencing.ssh.private-key-files</name>

<value>/home/hadoop/.ssh/id\_rsa</value>

</property>

<!-- 声明journalnode服务器存储目录-->

<property>

<name>dfs.journalnode.edits.dir</name>

<value>/usr/local/hadoop/data/ha/jn</value>

</property>

<!-- 关闭权限检查-->

<property>

<name>dfs.permissions.enable</name>

<value>false</value>

</property>

<!-- 访问代理类：client，mycluster，active配置失败自动切换实现方式-->

<property>

<name>dfs.client.failover.proxy.provider.mycluster</name>

<value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>

</property>

<!-- 配置自动故障转移-->

<property>

<name>dfs.ha.automatic-failover.enabled</name>

<value>true</value>

</property>

</configuration>

[hadoop@master1 ~]$ mv /usr/local/hadoop/etc/hadoop/mapred-site.xml.template /usr/local/hadoop/etc/hadoop/mapred-site.xml

[hadoop@master1 ~]$ vi /usr/local/hadoop/etc/hadoop/mapred-site.xml

<configuration>

<!-- 指定mapreduce生成文件的目录 -->

<property>

<name>mapred.local.dir</name>

<value>/usr/local/hadoop/var</value>

</property>

<!-- 指定mapreduce的v2版本yarn -->

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

<!-- 指定mr历史服务器主机,端口 -->

<property>

<name>mapreduce.jobhistory.address</name>

<value>master1:10020</value>

</property>

<!-- 指定mr历史服务器WebUI主机、端口 -->

<property>

<name>mapreduce.jobhistory.webapp.address</name>

<value>master1:19888</value>

</property>

<!-- 历史服务器的WEB UI上最多显示20000个历史的作业记录信息 -->

<property>

<name>mapreduce.jobhistory.joblist.cache.size</name>

<value>20000</value>

</property>

</configuration>

[hadoop@master1 ~]$ vi /usr/local/hadoop/etc/hadoop/yarn-site.xml

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<!--启用resourcemanager ha-->

<property>

<name>yarn.resourcemanager.ha.enabled</name>

<value>true</value>

</property>

<!--声明两台resourcemanager的地址-->

<property>

<name>yarn.resourcemanager.cluster-id</name>

<value>rmCluster</value>

</property>

<property>

<name>yarn.resourcemanager.ha.rm-ids</name>

<value>rm1,rm2</value>

</property>

<property>

<name>yarn.resourcemanager.hostname.rm1</name>

<value>slave1</value>

</property>

<property>

<name>yarn.resourcemanager.hostname.rm2</name>

<value>slave2</value>

</property>

<!--指定zookeeper集群的地址-->

<property>

<name>yarn.resourcemanager.zk-address</name>

<value>master1:2181,master2:2181,slave1:2181,slave2:2181</value>

</property>

<!--启用自动恢复-->

<property>

<name>yarn.resourcemanager.recovery.enabled</name>

<value>true</value>

</property>

<!--指定resourcemanager的状态信息存储在zookeeper集群-->

<property>

<name>yarn.resourcemanager.store.class</name>

<value>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</value>

</property>

</configuration>

[hadoop@master1 ~]$ cat <<END >/usr/local/hadoop/etc/hadoop/slaves

master1

master2

slave1

slave2

END

* **在master2、slave1、slave2节点准备hadoop目录，同步master1的配置：**

[hadoop@master2 ~]$ exit

[root@master2 ~]# mkdir /usr/local/hadoop

[root@master2 ~]# chown hadoop:hadoop /usr/local/hadoop

[root@master2 ~]# cat <<END >>/etc/profile

export HADOOP\_HOME=/usr/local/hadoop/

export PATH=\$PATH:\$HADOOP\_HOME/bin

END

[root@master2 ~]# source /etc/profile

[hadoop@slave1 ~]$ exit

[root@slave1 ~]# mkdir /usr/local/hadoop

[root@slave1 ~]# chown hadoop:hadoop /usr/local/hadoop

[root@slave1 ~]# cat <<END >>/etc/profile

export HADOOP\_HOME=/usr/local/hadoop/

export PATH=\$PATH:\$HADOOP\_HOME/bin

END

[root@slave1 ~]# source /etc/profile

[hadoop@slave2 ~]$ exit

[root@slave2 ~]# mkdir /usr/local/hadoop

[root@slave2~]# chown hadoop:hadoop /usr/local/hadoop

[root@slave2 ~]# cat <<END >>/etc/profile

export HADOOP\_HOME=/usr/local/hadoop/

export PATH=\$PATH:\$HADOOP\_HOME/bin

END

[root@slave2 ~]# source /etc/profile

[hadoop@master1 ~]$ scp -r /usr/local/hadoop/\* hadoop@master2:/usr/local/hadoop/

[hadoop@master1 ~]$ scp -r /usr/local/hadoop/\* hadoop@slave1:/usr/local/hadoop/

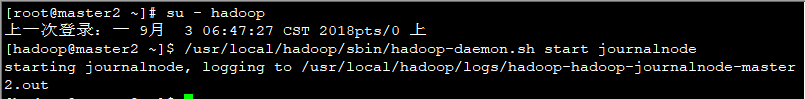
[hadoop@master1 ~]$ scp -r /usr/local/hadoop/\* hadoop@slave2:/usr/local/hadoop/

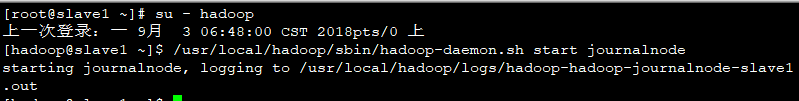
* **在所有节点初始化journalnode服务，自动创建目录：**

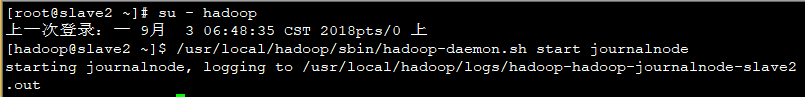






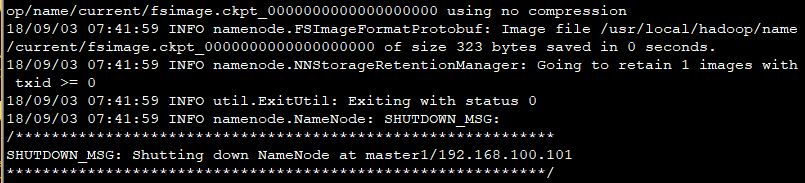


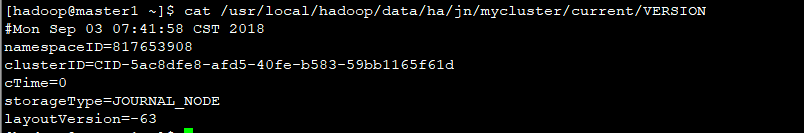




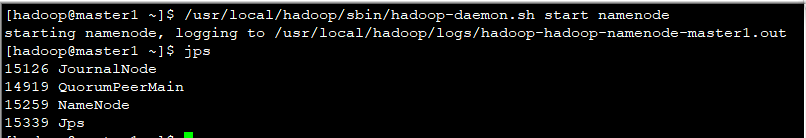
* **在master1节点格式化namenode节点：**

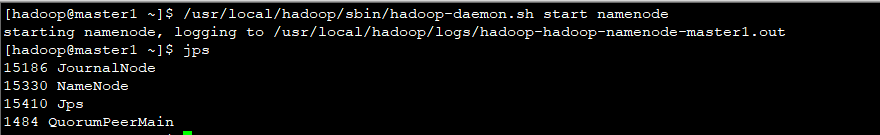




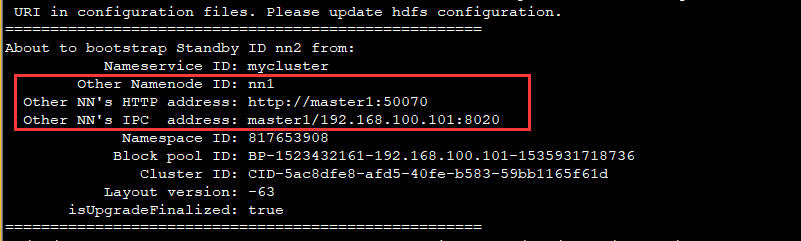


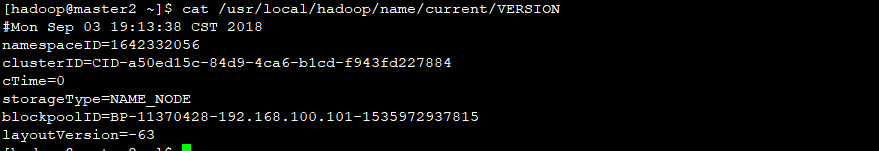
* **启动所有节点的对应namenode或datanode服务：**

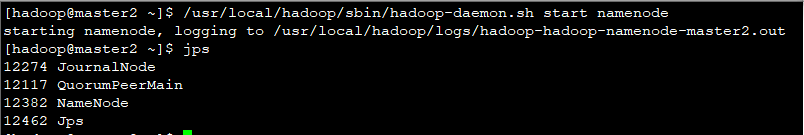


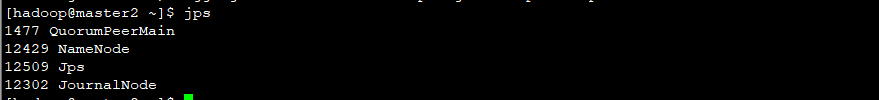


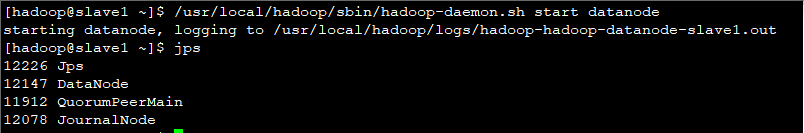


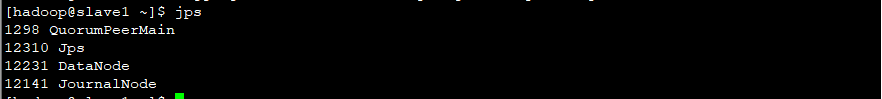


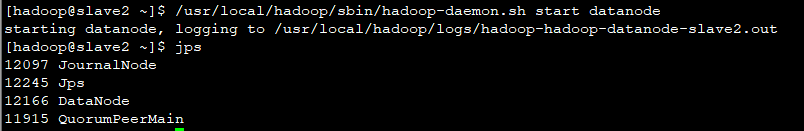


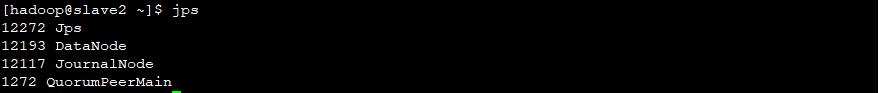




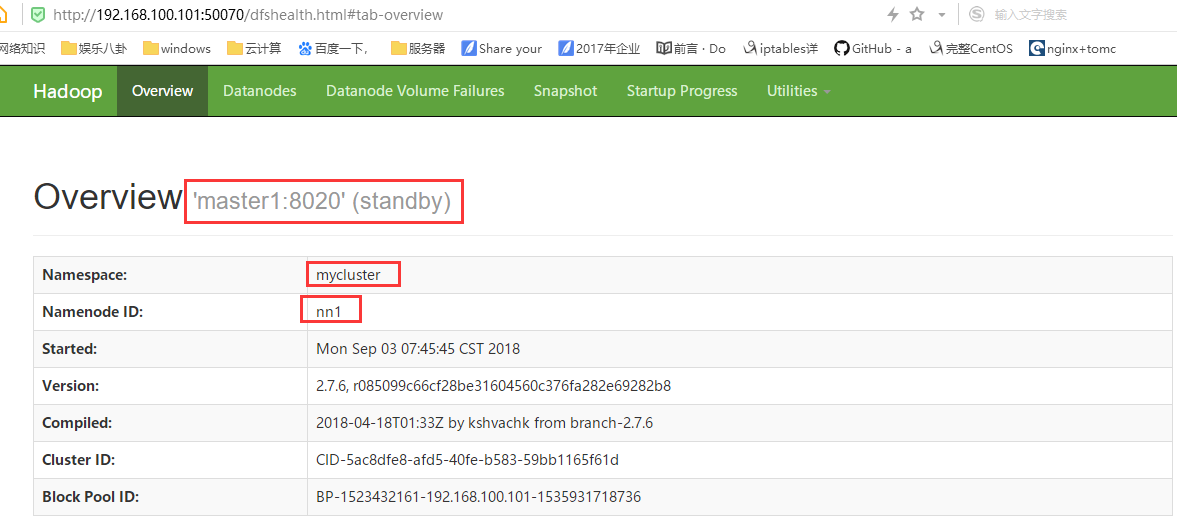


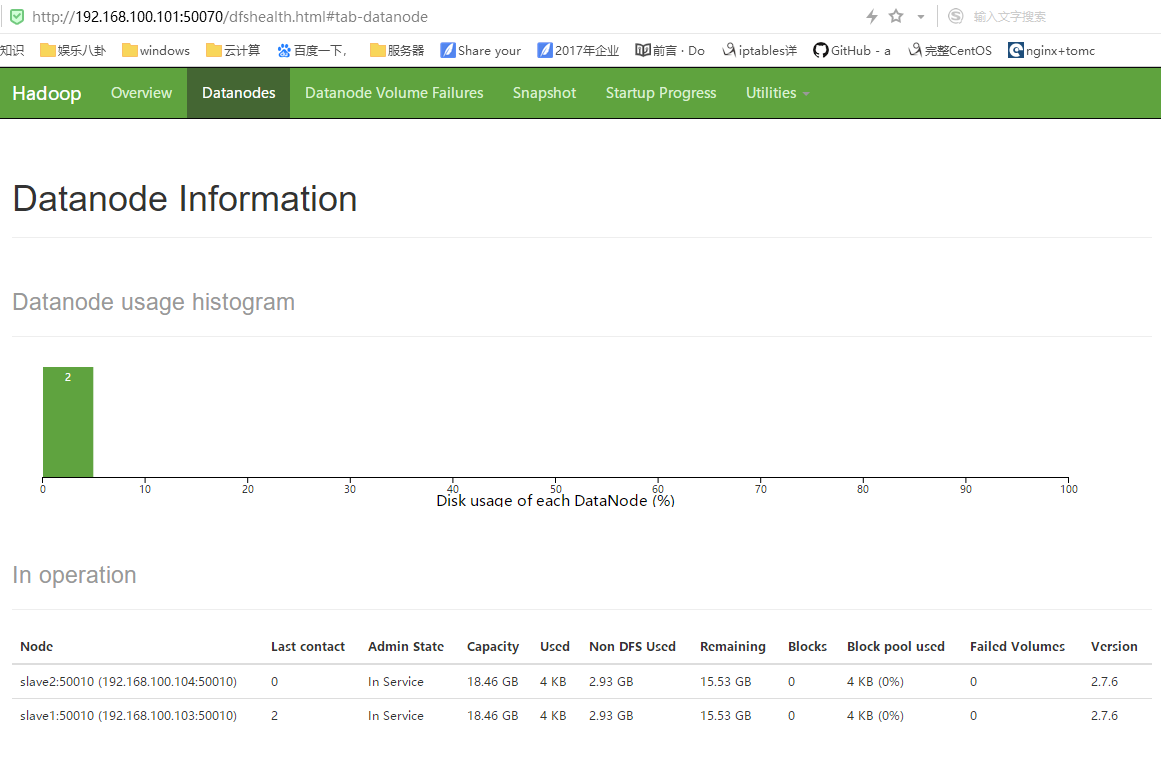


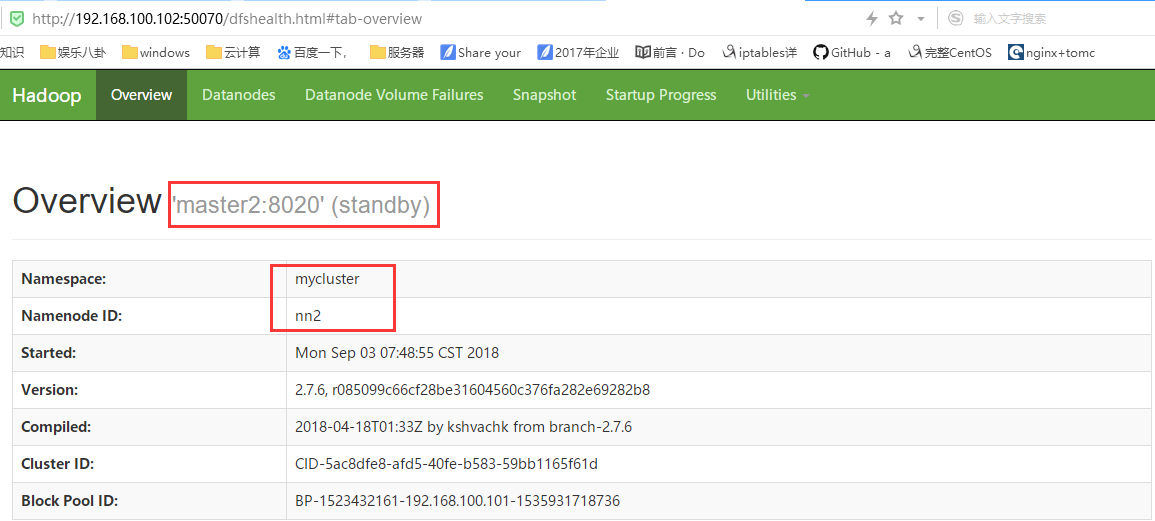




* **访问web节点，查看hadoop集群状态；**







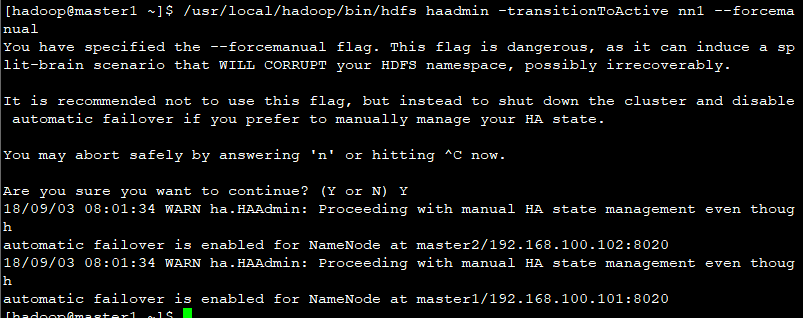
* **手动切换状态，在各个NameNode节点上启动DFSZK Failover Controller，并且强制设置master1作为active节点；**

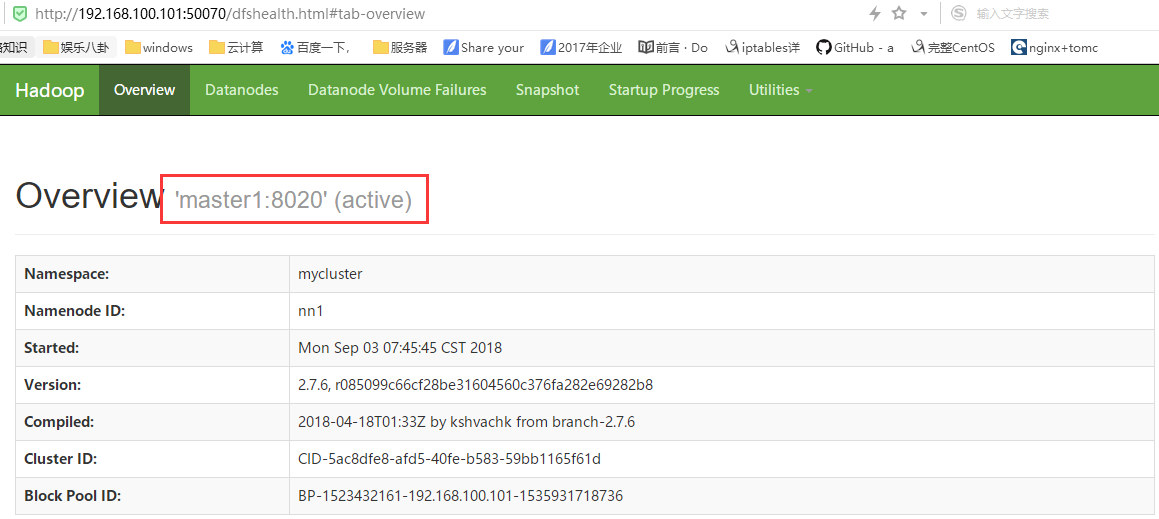
[hadoop@master1 ~]$ /usr/local/hadoop/sbin/hadoop-daemon.sh start zkfc

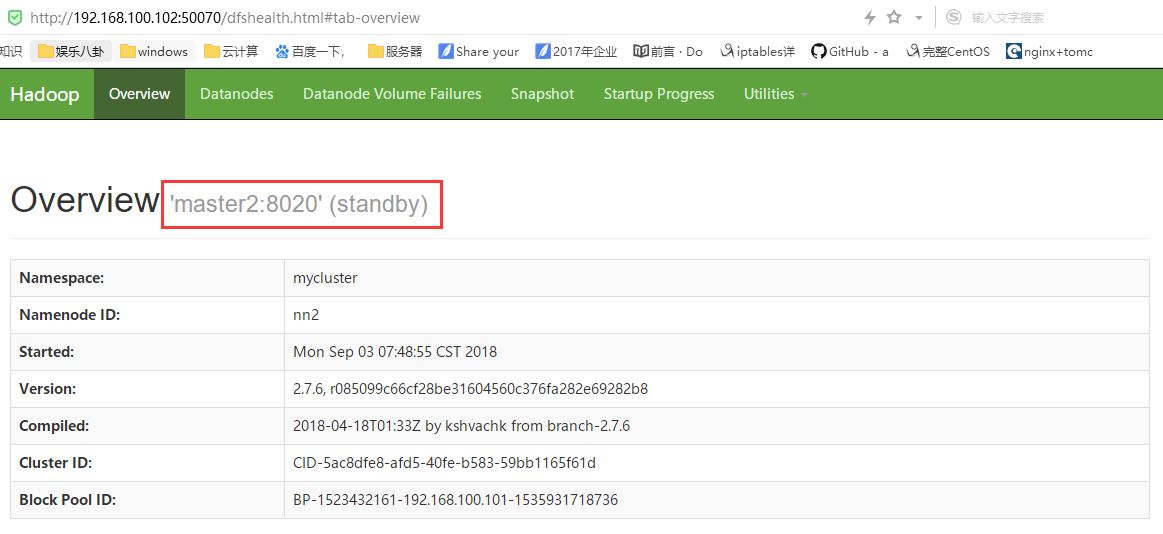
starting zkfc, logging to /usr/local/hadoop/logs/hadoop-hadoop-zkfc-master1.out

[hadoop@master2 ~]$ /usr/local/hadoop/sbin/hadoop-daemon.sh start zkfc

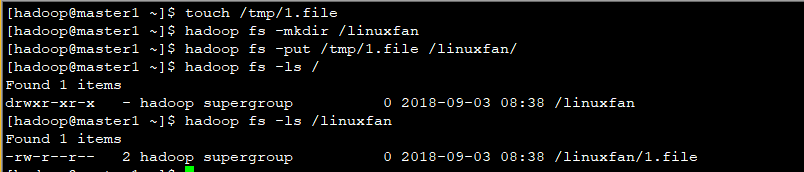
starting zkfc, logging to /usr/local/hadoop/logs/hadoop-hadoop-zkfc-master2.out

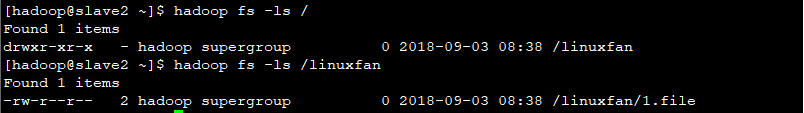




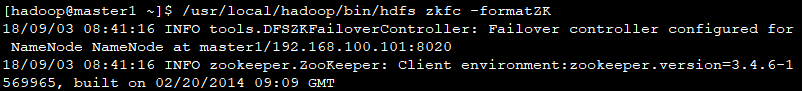


* **测试Hadoop分布式集群数据存储：**

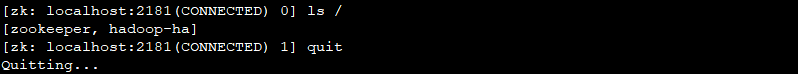




* **在master1节点初始化ha集群在zookeeper中的状态：**



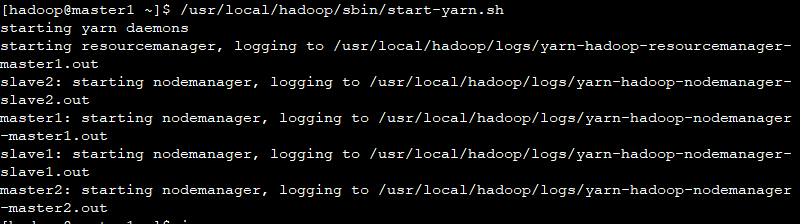


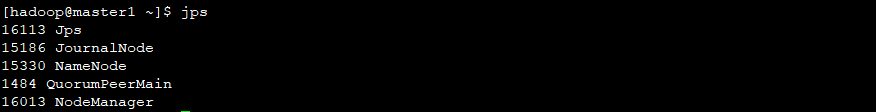


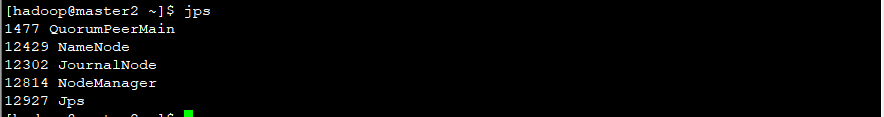


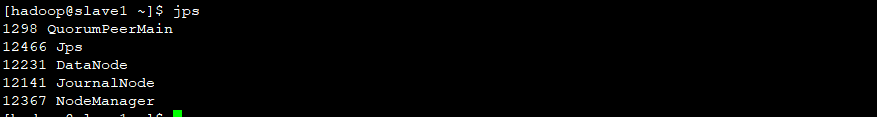


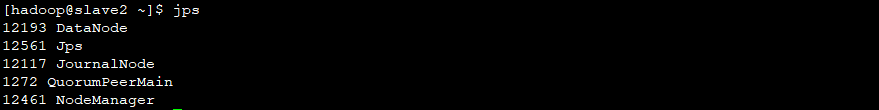
* **在master1节点启动mapreduce-yarn进程：**





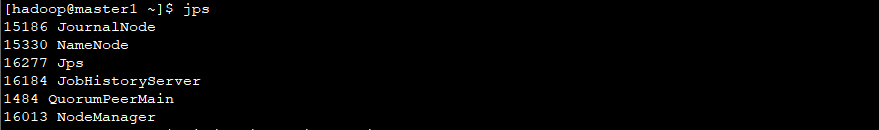


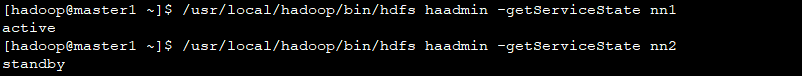




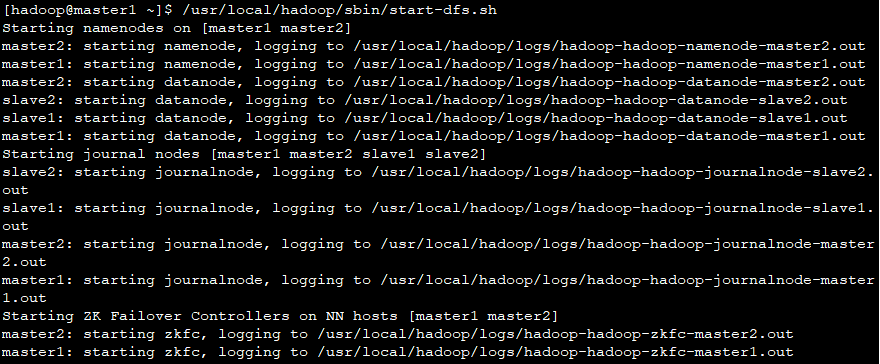
* **在master1节点启动historyserver进程：**

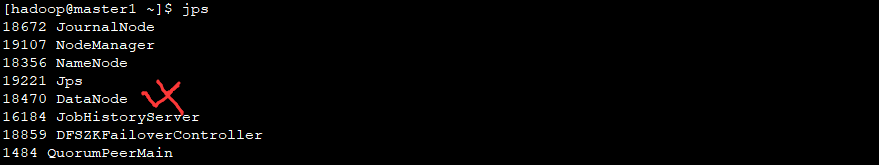


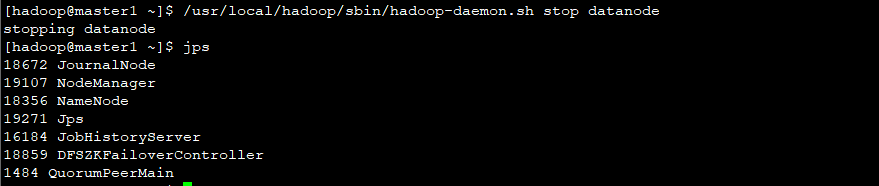


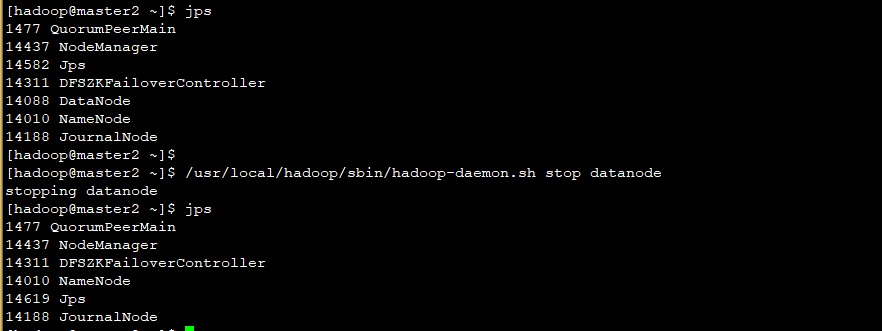


* **在master1节点开启集群的DFSZKFailoverController进程（按照以下方法）；**

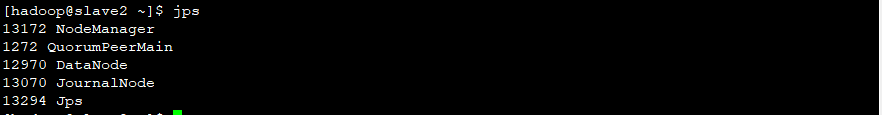




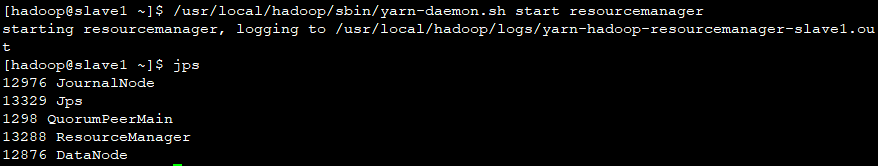


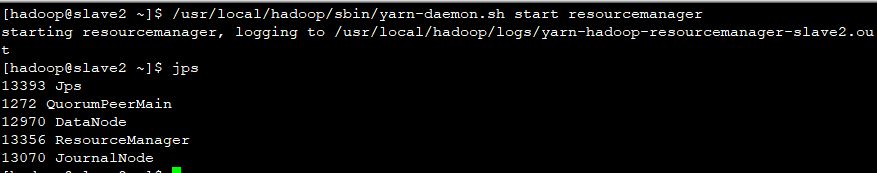


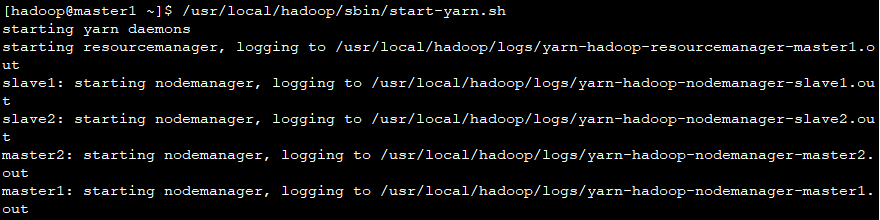


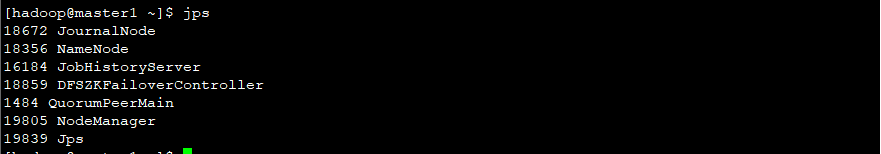


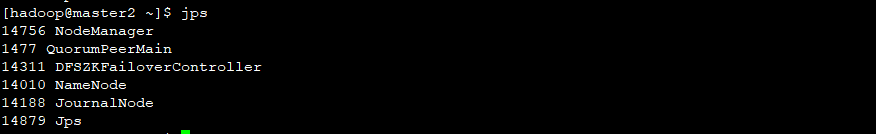
* **在slave1、slave2节点启动resourcemanager客户端任务进程，并且验证最终集群进程分布状态：**

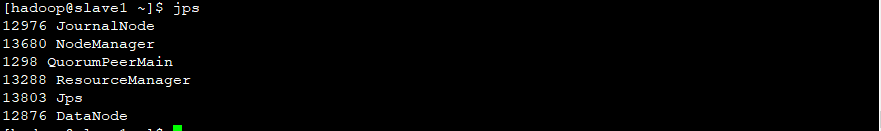


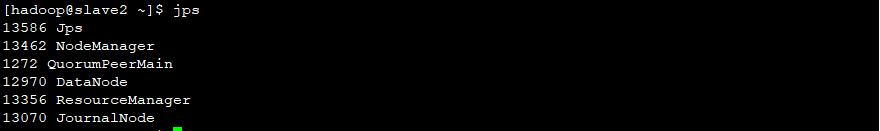


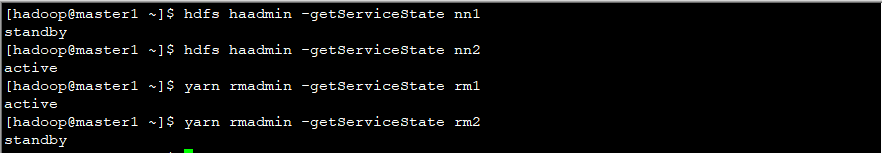




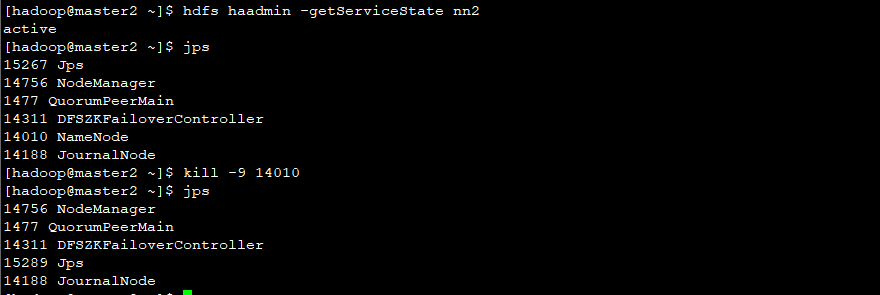


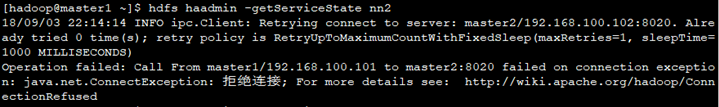




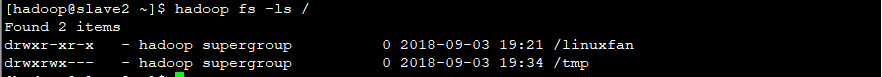


* **集群Namenode节点高可用测试：**

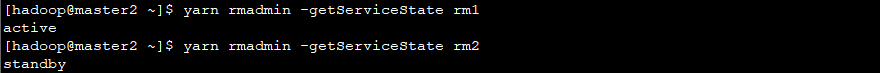


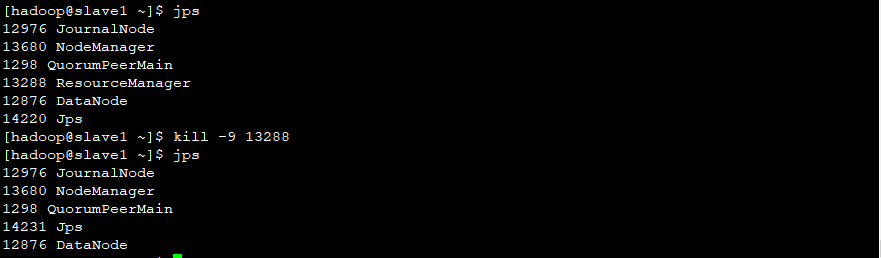


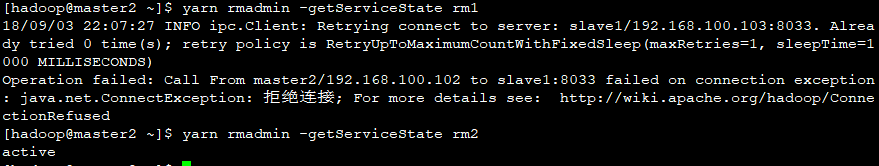


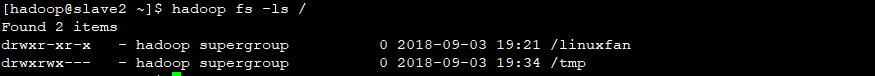


* **集群resourceManager节点高可用测试：**









**注：在以上测试namenode节点和resourcemanager节点高可用时，对于客户端正在读取和写入得过程中得文件，不会发生任何影响；**

* **错误总结：**

故障现象：在两台namenode节点上切换active活跃角色时，无法正常切换；

故障原因：在namenode节点切换的过程中，需要使用到fuser命令；

解决方案：在所有的hadoop节点yum安装psmisc软件，便支持了fuser命令，