**15Architecture\_07常用组件+kafka集群**

**一 常用组件-zookeeper**

**1.1 zookeeper是什么**

1.1.1 zookeeper是一个开源的分布式应用程序协调服务

1.1.2 作用: zookeeper用来保证数据在集群间的事务一致性

1.1.3 zookeeper应用场景

集群分布式锁 集群统一命名服务 分布式协调服务

**1.2 角色与特性**

**1.2.1 Leader**: 接受所有Follower的天请求并统一协调发起的提案的投票,负责所有的Follower进行内部数据交换

**1.2.2 Follower**: 直接为客户端服务,参与提案的投票,同时与Leader进行数据交换

1**.2.3 observer**: 直接为客户端服务,不参与提案的投票,同时也与Leader进行数据交换

**1.3 角色与选举**

**1.3.1 zookeeper角色与选举**

服务在启动的时候是没有角色的(LOOKING)

角色是通过选举产生的

选举产生一个Leader,剩下的是follower

**1.3.2 选举leader原则**

集群中超过半数机器投票选择leader

假如集群中拥有n太服务器,那么leader必须得到n/2+1台服务器的投票

**1.3.3 其他规则**

如果leader死亡,重新选举leader

如果死亡的机器数量达到一半,则集群挂掉

如果无法得到足够的投票数量,则重新发起投票,如果参与投票的机器不足n/2+1,则集群停止工作

如果follower死亡过多,剩余机器不足n/2+1,则集群也会停止工作

observer不计算在投票总设备数量里面

**1.4 zookeeper原理与设计**

**1.4.1 zookeeper可伸缩扩展性原理与设计**

客户端提交一个请求,若是读请求,则由每台server的本地副本数据库直接相应;若是写请求,需要通过一致性协议(zab)来处理.

zab协议规定:来自client的所有写请求都要转发给ZK服务中唯一的leader,由leader根据该请求发起一个proposal,然后其他的server对该proposal进行vote.之后leader对vote进行收集,但vote数量过半时leader会向所有的server发送一个通知消息.最后,当client所连接的server收到该消息时,会把该操作更新到内存中,并对client的写请求做出回应.

zookeeper在上述协议中实际扮演了两个职能:一方面从客户端接受连接与操作请求;另一方面对操作结果进行投票.这两个职能在zookeeper集群扩展的时候彼此制约.

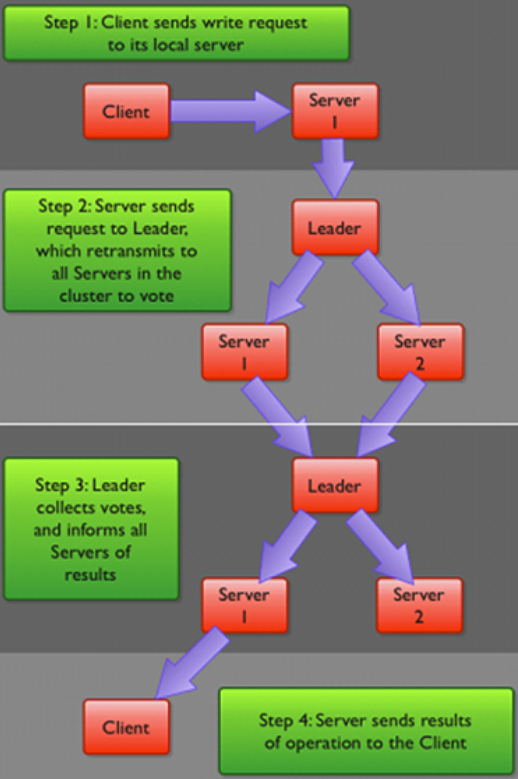
从zab协议对写请求的处理过程可以发现,增加follower的数量,则增加了协议投票过程的压力.因为leader节点必须等待集群中过半server响应投票,是节点的增加使得部分计算机运行较慢,从而拖慢整个投票过程的可能性也随之提高,随着集群变大,写操作也会随之下降.

所以,我们不得不在增加client数量的期望和我们希望保持较好吞吐性能的期望之间进行权衡.要打破这一耦合关系,我们引入了不参与投票的服务器observer.observer可以接受客户端的连接,并将写请求转发给leader节点.但leader节点不会要求observer参与投票,仅仅在上述第3步那样,和其他服务节点一起得到投票结果.

observer的扩展,给zookeeper的可伸缩性带来了全新的景象.加入很多observer节点,无须担心严重影响写吞吐量.但并非是无懈可击,因为协议中的通知阶段,仍然与服务器的数量呈线性关系.但是这里的串行开销非常低.因此,我们可以认为在通知服务器阶段的开销不会成为瓶颈.

observer提升读性能的可伸缩性

observer提供了广域网能力



**1.5 zookeeper集群**

**1.5.1 zookeeper集群的安装配置**

**配置文件改名zoo.cfg**

mv zoo\_sample.cfg zoo.cfg

**zoo.cfg文件最后添加如下内容**

server.1=node1:2888:3888

server.2=node2:2888:3888

server.3=node3:2888:3888

server.4=nn011:2888:3888:observer

**创建datadir指定的目录**

mkdir /tmp/zookeeper

**在目录下创建id对应主机的myid文件**

关于myid文件

myid文件中只有一个数字

注意:确保每个server的myid文件中id数字不同

server.id中的id与myid中的id必须一致

id的范围是1-255

**启动集群,查看验证(在所有集群节点执行)**

/usr/local/zookeeper/bin/zkServer.sh start

**查看角色**

/usr/local/zookeeper/bin/zkServer.sh status

**zookeeper管理文档**

<http://zookeeper.apache.org/doc/r3.4.10/zookeeperAdmin.html>

**案例一:zookeeper安装**

**1. 关机调整node61 node62 node63的内存为3000000kib,并开机**

room9pc01 ~]# virsh edit node61\62\63

<memory unit='KiB'>3000000</memory>

<currentMemory unit='KiB'>3000000</currentMemory>

room9pc01 ~]# virsh start node61\62\63

**2. 编辑hosts文件,所有主机能相互ping通**

room9pc01 ~]# scp /linux-soft/04/hadoop/zookeeper-3.4.13.tar.gz [root@192.168.1.60:/root](mailto:root@192.168.1.60:/root)

nn01 ~]# tar -xf zookeeper-3.4.13.tar.gz

nn01 ~]# mv zookeeper-3.4.13 /usr/local/zookeeper

**3. 安装依赖java-1.8.0-openjdk java-1.8.0-openjdk-devel(所有主机)**

**4.修改配置文件,并修改权限归属**

nn01 ~]# cd /usr/local/zookeeper/conf

nn01 conf]# cp zoo\_sample.cfg zoo.cfg

nn01 conf]# vim zoo.cfg

server.61=node61:2888:3888 #最下面添加此4行

server.62=node62:2888:3888

server.63=node63:2888:3888

server.60=nn01:2888:3888:observer

nn01 conf]# chown root.root zoo.cfg

**5. 拷贝/usr/local/zookeeper到其他主机**

n01 ~]# for i in {61..63}

> do

> scp -r /usr/local/zookeeper 192.168.1.$i:/usr/local/zookeeper

> done

**6. 手动创建/tmp/zookeeper,每台机器**

nn01\61\62\63 ~]# mkdir /tmp/zookeeper

**7. 创建myid文件,id必须与配置文件里主机名对应的server.id一致**

nn01 ~]# echo 60 > /tmp/zookeeper/myid

node61 ~]# echo 61 > /tmp/zookeeper/myid

node62 ~]# echo 62 > /tmp/zookeeper/myid

node63 ~]# echo 63 > /tmp/zookeeper/myid

**8. 启动服务(每台机器)**

nn01\61\62\63 ~]# /usr/local/zookeeper/bin/zkServer.sh start

**9.验证**

nn01 ~]# /usr/local/zookeeper/bin/zkServer.sh status

ZooKeeper JMX enabled by default

Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg

Mode: observer

node61\62\63 ~]# /usr/local/zookeeper/bin/zkServer.sh status

ZooKeeper JMX enabled by default

Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg

Mode: follower(62为leader)

补充: 使用官方管理文档中4字命令的方法(示例)

socat TCP:nn01\node61\node62\node63:2181 -

ruok

imok

**二 kafka集群**

**2.1 什么是kafka**

kafka是由linkedIn开发的一个分布式的消息系统

kafka是使用Scala编写的,需要java-x.x.x-openjdk[-devel]

kafka是一种消息中间件

**为什么要使用kafka**

解耦,冗余,提高扩展性,缓冲;保证顺序,灵活,削峰填谷;异步通信

**2.2 kafka角色**

producer: 生产者,负责发布消息

consumer: 消费者,负责读取处理消息

topic: 消息的类别

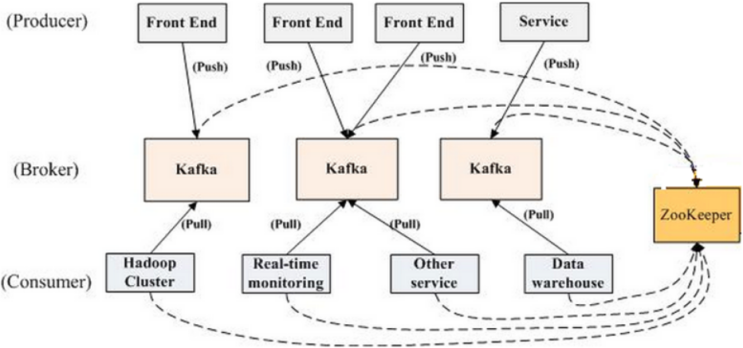
partition: 每个topic包含一个或多个partition

broker: kafka集群包含一个或多个服务器

kafka通过zookeeper管理集群配置,选举leader

**2.3 kafka集群安装与配置**

**2.3.1 kafka角色与集群结构**



**2.3.2 kafka集群的安装配置依赖zookeeper,搭建kafka集群之前,需要创建好zookeeper集群(已完成)**

**2.3.3 安装openJDK运行环境(已完成)**

**2.3.4 安装kafka**

room9pc01 ~]# scp /linux-soft/04/hadoop/kafka\_2.12-2.1.0.tgz [root@192.168.1.61\62\63:/root](mailto:root@192.168.1.60/61/62/63:/root)

node61 ~]# tar -xf kafka\_2.12-2.1.0.tgz

node61 ~]# mv kafka\_2.12-2.1.0 /usr/local/kafka

**2.3.5 修改配置文件/usr/local/kafka/config/server.properties**

/usr/local/kafka/config/server.properties

broker.id: 每台服务器的broker.id都不能相同

zookeeper.connect: zookeeper集群地址

node61\62\63 ~]# vim /usr/local/kafka/config/server.properties

21 broker.id=61 #62\63主机上要修改为62\63

123 zookeeper.connect=node61:2181,node62:2181,node63:2181

**2.3.6 同步kafka拷贝到所有集群主机**

node61 ~]# for i in 62 63

> do scp -r /usr/local/kafka root@192.168.1.$i:/usr/local/kafka;

> done

修改62\63主机/usr/local/kafka/config/server.properties中broker.id的值为62\63

**2.3.7 启动与验证**

node61\62\63 ~]# /usr/local/kafka/bin/kafka-server-start.sh \

-daemon /usr/local/kafka/config/server.properties

node61 ~]# jps #验证

834 QuorumPeerMain

1304 Kafka

1370 Jps

node61\62\63 ~]# ss -antulp | grep 9092

**2.3.8 集群验证与消息发布**(验证kafka安装是否完成)

**创建一个topic**

node61 ~]# cd /usr/local/kafka/

[root@node61 kafka]# ./bin/kafka-topics.sh --create \

--partitions 2 --replication-factor 2 --zookeeper \

localhost:2181 --topic nsd1906

Created topic "nsd1906".

**生产者**

node62 ~]# cd /usr/local/kafka/

node62 kafka]# ./bin/kafka-console-producer.sh \

--broker-list localhost:9092 --topic nsd1906

> #表示进入消息发布界面

**消费者**

node63 ~]# cd /usr/local/kafka/

node63 kafka]# ./bin/kafka-console-consumer.sh \

--bootstrap-server localhost:9092 --topic nsd1906

#进入消息接收界面

**在生产者(192.168.1.62)发布消息**

>abc

**在消费者(192.168.1.63)查看消息**

abc

**三 hadoop高可用**

**3.1 为什么需要namenode**

namenode是HDFS的核心配置,HDFS又是Hadoop核心组件,namenode在Hadoop集群中至关重要

namenode宕机,将导致集群不可用,如果namenode的书记丢失将导致整个集群的数据丢失,而namenode的数据更新又比较频繁,诗仙namenode的高可用势在必行

**3.2 解决方案**

官方提供了两种解决方案

HDFS with NFS HDFS with QJM

两种方案异同

NFS QJM

NN NN

ZK ZK

ZKFailoverController ZKFailoverController

NFS JournalNode

HA方案对比

都能实现热备

都是一个active NN和一个standby NN

都使用zookeeper和ZKFC来诗仙自动失效恢复

失效切换都使用fencin配置的方法来active NN

NFS数据共享变更方案把数据存储在共享存储里,我们还需要考虑NFS的高可用设计

QJM不需要共享存储,但需要让每一个DN都知道两个NN的位置,并把块信息和心跳包发送给active和standby这两个NN

**3.3 使用方案**

**3.3.1 使用QJM原因**

解决namenode单点故障问题

Hadoop给出了HDFS的高可用HA方案: HDFS通常由两个namenode组成,一个处于active状态,另一个处理standby状态.active namenode对外提供服务,比如 处理来自客户的RPC请求,而standby namenode则不对外提供服务,仅同步active namenode的状态,以便能够在active namenode失败时进行就切换

**3.3.2 典型的HA集群**

namenode会配置在两台独立的机器上,在任何时候,一个namenode处理活动状态,而另一个namenode则处理备份状态

活动状态的namenode会响应集群中所有的客户端,备份状态的namenode只是作为一个副本,保证在必要的时候提供一个快速的转移

3.4 namenode高可用

为了让standby node与active node保持同步,这两个node都与一组称为JNS(journal nodes)的互相独立的进程保持通信.当active更新了namespace,它将记录修改日志发送给JNS的多数派.standby node将会从JNS中读取这些edits,并持续关注它们对日志的变更.

standby node将日志变更应用在自己的namespace中,当failvoer发生时,standby将会提升自己为active之前,确保能够从JNS中读取所有的edits,即在failover发生之前standby持有的namespace与active保持完全同步

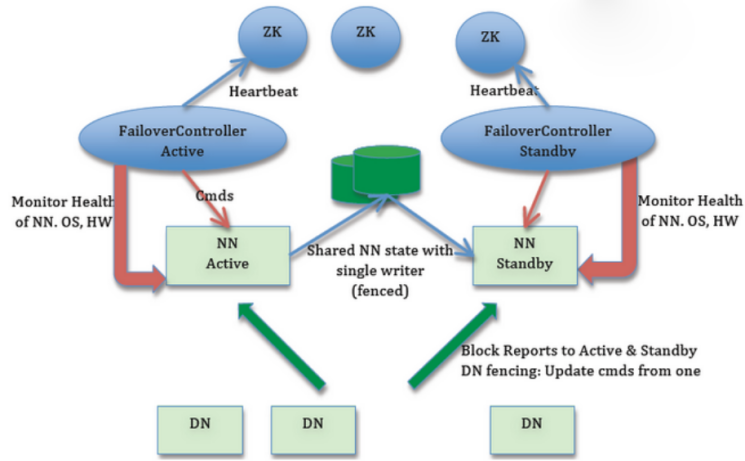
namenode更新很频繁,为了保持主备数据的一致性,为了支持快速failvoer,standbynode持有集群中blocks的最新位置是非常必要的.为了达到这一目的,datanode上需要同时配置这两个namenode的地址,同时和它们都建立心跳连接,并把block位置发送给它们

任何时刻.只能有一个active namenode,否则会导致集群操作混乱,两个namenode将会有两种不同的数据状态,可能会导致数据丢失或状态异常,这种情况通常称为split-brain(脑裂,三节点通讯阻断,即集群中不同的datanode看到了不同的active namenodes)

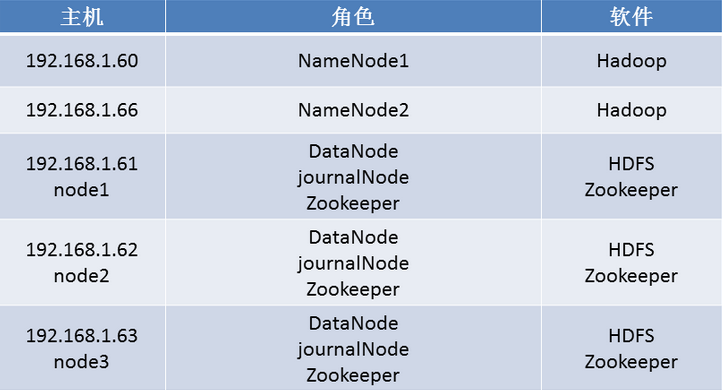
对于JNS而言,任何时候只允许一个namenode作为writer;在failover期间,原来的standby node将会接管active的所有职能,并负责向JNS写入日志记录,这种机制阻止了其他namenode处于active状态的问题

**3.5 namenode架构图**

namenode高可用架构图

****

**系统规划**



**创建主机,设置主机名和IP地址**

room9pc01 ~]# base-vm nn02

**编辑hosts文件,保证主机之间能相互ping通**

nn01 ~]# vim /etc/hosts

192.168.1.60 nn01

192.168.1.61 node61

192.168.1.62 node62

192.168.1.63 node63

192.168.1.64 node64

192.168.1.65 nfsgw65

192.168.1.66 nn02

nn01 ~]# for i in {61..66}

> do

> scp /etc/hosts root@192.168.1.$i:/etc/hosts

> done

**保证nn01和nn02能免输入密码ssh到其他主机上(包括本机)**

nn01 ~]# ssh-copy-id 192.168.1.66

nn02 ~]# ssh-keygen -f /root/.ssh/id\_rsa -N ''

nn02 ~]# for i in {60..66}; do ssh-copy-id 192.168.1.$i; done

或 nn01上执行rsync -aXHS ????????14:50

**安装Java-1.8.0-openjdk Java-1.8.0-openjdk-devel依赖**

nn02 ~]# yum -y install java-1.8.0-openjdk java-1.8.0-openjdk-devel

**清理主机的状态**

所有主机删除/var/hadoop/\*,并重启

**配置hadoop-env配置文件**

nn01 ~]# cd /usr/local/hadoop/etc/hadoop/

nn01 hadoop]# vim hadoop-env.sh

25 export JAVA\_HOME="/usr/lib/jvm/java-1.8.0-openjdk \

-1.8.0.161-2.b14.el7.x86\_64/jre" #java安装目录

33 export HADOOP\_CONF\_DIR="/usr/local/hadoop/etc/hadoop"

#Hadoop配置文件目录

**配置slaves文件**

nn01 hadoop]# vim slaves

node61

node62

node63

**3.6 core-site配置**

nn01 hadoop]# vim core-site.xml

<configuration>

**<property>**

**<name>fs.defaultFS</name>**

**<value>hdfs://nsd1906</value> #名称随便起,访问这个组**

**</property>**

<property>

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

<value>/var/hadoop</value>

</property>

**<property>**

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

**<value>node61:2181,node62:2181,node63:2181</value>**

**</property> #zookeeper的地址**

<property>

<name>hadoop.proxyuser.nfsuser.groups</name>

<value>\*</value>

</property>

<property>

<name>hadoop.proxyuser.nfsuser.hosts</name>

<value>\*</value>

</property>

</configuration>

**3.7 hdfs-site配置**

nn01 hadoop]# vim hdfs-site.xml

<configuration>

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

<property>

<name>dfs.nameservices</name>

<value>nsd1906</value>

</property>

<property>

<name>dfs.ha.namenodes.nsd1906</name>

<value>nn01,nn02</value>

</property>

<property>

<name>dfs.namenode.rpc-address.nsd1906.nn01</name>

<value>nn01:8020</value>

</property>

<property>

<name>dfs.namenode.rpc-address.nsd1906.nn02</name>

<value>nn02:8020</value>

</property>

<property>

<name>dfs.namenode.http-address.nsd1906.nn01</name>

<value>nn01:50070</value>

</property>

<property>

<name>dfs.namenode.http-address.nsd1906.nn02</name>

<value>nn02:50070</value>

</property>

<property>

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

<value>qjournal://node61:8485;node62:8485;node63:8485/nsd1906</value>

</property>

<property>

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

<value>/var/hadoop/journal</value>

</property>

<property>

<name>dfs.client.failover.proxy.provider.nsd1906</name>

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

</property>

<property>

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

<value>sshfence</value>

</property>

<property>

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

<value>/root/.ssh/id\_rsa</value>

</property>

<property>

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

<value>true</value>

</property>

</configuration>

**3.8 yarn高可用**

**3.9 yarn-site配置**

nn01 hadoop]# vim yarn-site.xml

<configuration>

<!-- Site specific YARN configuration properties -->

<property>

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

<value>mapreduce\_shuffle</value>

</property>

<property>

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

<value>true</value>

</property>

<property>

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

<value>rm1,rm2</value>

</property>

<property>

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

<value>true</value>

</property>

<property>

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

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

</property>

<property>

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

<value>node61:2181,node62:2181,node63:2181</value>

</property>

<property>

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

<value>yarn-ha</value>

</property>

<property>

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

<value>nn01</value>

</property>

<property>

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

<value>nn02</value>

</property>

</configuration>

**3.10 编辑mapred-site.xml**

nn01 hadoop]# vim mapred-site.xml

<configuration>

<property>

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

<value>yarn</value>

</property>

</configuration>

**3.11 同步配置文件到所有主机**

nn01 hadoop]# for i in {61..66}

> do

> scp hadoop-env.sh slaves core-site.xml hdfs-site.xml yarn-site.xml mapred-site.xml root@192.168.1.$i:/usr/local/hadoop/etc/hadoop/

> done

nn01 ~]# scp -r /usr/local/hadoop [root@192.168.1.66:/usr/local/](mailto:root@192.168.1.66:/usr/local/)

3.12 删除所有主机上的/usr/local/hadoop/logs,方便排错

nn01\02 ~]# rm -rf /usr/local/hadoop/logs/\*

node61\62\63 ~]# rm -rf /usr/local/hadoop/logs/\*

**四 高可用验证**

**4.1 集群初始化**

**4.1.1 nn01初始化ZK集群**

nn01 ~]# cd /usr/local/hadoop/

nn01 hadoop]# ./bin/hdfs zkfc -fromatZK #61 62 63上的ZK要开启

**4.1.2 node主机上启动journalnode服务**

node61 ~]# cd /usr/local/hadoop/

node61\62\63 hadoop]# ./sbin/hadoop-daemon.sh start journalnode

**4.1.3 nn01格式化**

nn01 hadoop]# ./bin/hdfs namenode -format

**4.1.4 nn02数据同步到本地/var/hadoop/dfs**

nn02 ~]# rsync -aSH nn01:/var/hadoop/dfs /var/hadoop/

**4.1.5 nn01初始化JNS**

nn01 hadoop]# ./bin/hdfs namenode -initializeShareEdits

**4.1.6 node主机上停止journalnode服务**

node61\62\63 hadoop]# ./sbin/hadoop-daemon.sh stop journalnode

**4.2 启动集群**

**4.2.1 nn01启动hdfs**

nn01 hadoop]# ./sbin/start-dfs.sh

**4.2.2 nn01启动yarn**

nn01 hadoop]# ./sbin/start-yarn.sh

**4.2.3 nn02启动热备resourcemanager**

nn02 hadoop]# ./sbin/yarn-daemon.sh start resourcemanager

**4.3 集群验证**

**4.3.1 查看集群状态**

获取namenod状态

nn01 hadoop]# ./bin/hdfs haadmin -getServiceState nn01

standby

# 执行: nn01 hadoop]# ./bin/hdfs haadmin -transitionToActive nn01 --forcemanual 将nno1变成active

n01 hadoop]# ./bin/hdfs haadmin -getServiceState nn02

standby

**4.3.2 获取resourcemanager状态**

nn01 hadoop]# ./bin/yarn rmadmin -getServiceState rm1

active

nn01 hadoop]# ./bin/yarn rmadmin -getServiceState rm2

standby

**4.3.3 获取节点信息**

nn01 hadoop]# ./bin/hdfs dfsadmin -report

Live datanodes (3):

Name: 192.168.1.61:50010 (node61)

Name: 192.168.1.63:50010 (node63)

Name: 192.168.1.62:50010 (node62)

nn01 hadoop]# ./bin/yarn node -list

Total Nodes:3

Node-Id Node-State Node-Http-Address Number-of-Running-Containers

node62:41529 RUNNING node62:8042 0

node63:40206 RUNNING node63:8042 0

node61:43836 RUNNING node61:8042 0

**4.3.4 访问集群文件**

nn01 hadoop]# ./bin/hadoop fs -mkdir /input

nn01 hadoop]# ./bin/hadoop fs -ls hdfs://nsd1906/

Found 1 items

drwxr-xr-x - root supergroup 0 2019-10-21 17:47 hdfs://nsd1906/input

**4.3.5 主从切换active**

nn01 hadoop]# ./sbin/hadoop-daemon.sh stop namenode