git://124.207.192.18/nsd1808.git

**Morning：**

1. Zookeeper

**-什么是Zookeeper**

开源的分布式应用程序协调服务，避免资源争抢；

用于保证数据在集群间的事务一致性(从请求->释放的全过程)。

**-应用场景**

集群分布式锁：程序执行过程的锁，执行完释放

集群统一命名服务

分布式协调服务

**-Zookeeper角色与选举**

1）角色

Leader：发起提案的投票 + 写操作

Follower：参与投票 + 读操作

Observer：读操作

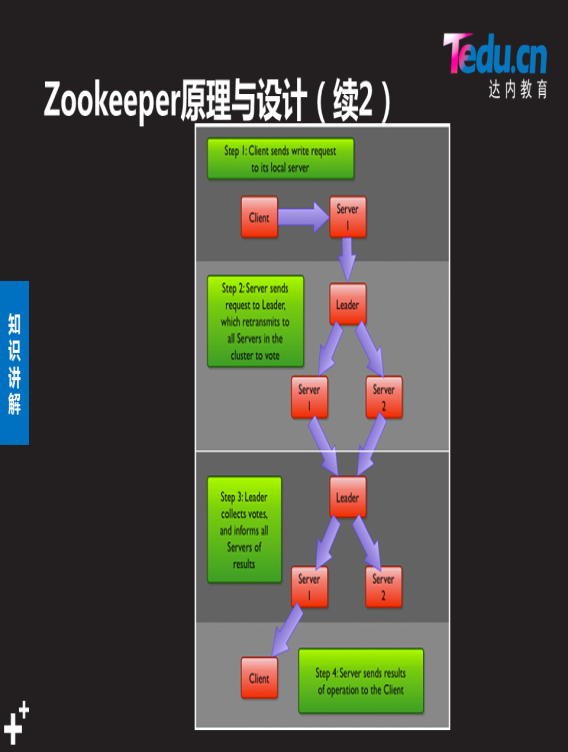
1. 选举

集群中超过半数机器投票选择Leader；

如果Leader死亡，重新选举；Follower死亡超过一半，集群停止工作。

奇数台与偶数台冗余性一致，因此配奇数台

**-Zookeeper原理**



客户端读请求 -> Follower/Observer直接响应

客户端写请求 -> 通过一致性协议(Zab)处理

写请求转发给Leader，Leader发起提案，Follower参数投票；

投票过半时，Leader给所有Server发送通知；

客户连接的Server收到后通知，把写操作更新到内存，执行并回应

1. 搭建Zookeeper集群

**步骤1：准备工作**

1）虚拟机准备

192.168.1.21 zk1 (2CPU、2Gmem、20Gdisk)

192.168.1.22 zk2

192.168.1.23 zk3

192.168.1.15 nfsgw

1. 安装java-1.8.0-openjdk-devel
2. 配置/etc/hosts

**步骤2：安装并配置Zookeeper**

[root@zk1 ~]# tar -xf zookeeper-3.4.10.tar.gz

[root@zk1 ~]# mv zookeeper-3.4.10/ /usr/local/zookeeper

#安装Zookeeper

[root@zk1 zookeeper]# mv conf/zoo\_sample.cfg conf/zoo.cfg

[root@zk1 zookeeper]# vim conf/zoo.cfg

... ...

server.1=zk1:2888:3888

server.2=zk2:2888:3888

server.3=zk3:2888:3888

server.4=nfsgw:2888:3888:observer

#配置Follower、Observer；Leader自动选举无需配置

[root@zk1 ~]# for i in zk{2..3} nfsgw; do

> rsync -avz --delete /usr/local/zookeeper $i:/usr/local/

> done

#同步zookeeper安装目录

**步骤3：创建工作目录、指定myid**

[root@zk1 ~]# mkdir /tmp/zookeeper

[root@zk1 ~]# ssh zk2 ‘mkdir /tmp/zookeeper’

... ...

[root@zk1 ~]# echo 1 > /tmp/zookeeper/myid

[root@zk1 ~]# ssh zk2 ‘echo 2 > /tmp/zookeeper/myid’

... ...

#myid必须与配置项对应

**步骤4：启动集群，查看验证**

[root@zk1 ~]# /usr/local/zookeeper/bin/zkServer.sh start

[root@zk2 ~]# /usr/local/zookeeper/bin/zkServer.sh start

... ...

#启动至少半数，才可以选举Leader，启动集群

[root@zk1 ~]# /usr/local/zookeeper/bin/zkServer.sh status

Mode: follower

#查看状态，验证角色

**步骤5：利用API查看状态**

1）socat TCP:zk1:2181 - #向zk发送指令，-号表示标准输入/输出

> stat #查看状态/角色

> ruok #查看健康性

> conf #查看配置

2）编写SHELL脚本，获取节点角色

[root@zk1 ~]# vim zk\_stat.sh

#!/bin/bash

function zk\_stat(){

exec 2>/dev/null

exec 8<>/dev/tcp/$1/2181

echo 'stat' >&8

MSG=`cat >&1 <&8 | grep -E "^Mode:"`

exec 8<&-

echo -e "${1}\t${MSG:-NULL}"

}

if [ $# -eq 0 ];then

echo 'Usage: ./zk\_stat.sh host1 host2 host3 ...'

exit 1

fi

for i in $@

do

zk\_stat ${i}

done

3、Kafka(了解)

**-什么是Kafka**

消息中间件（类似中介平台）

作用：解耦、冗余、提高扩展性、缓冲、保证顺序、异步通信

**-Kafka角色与集群机构**

生产者：发布消息

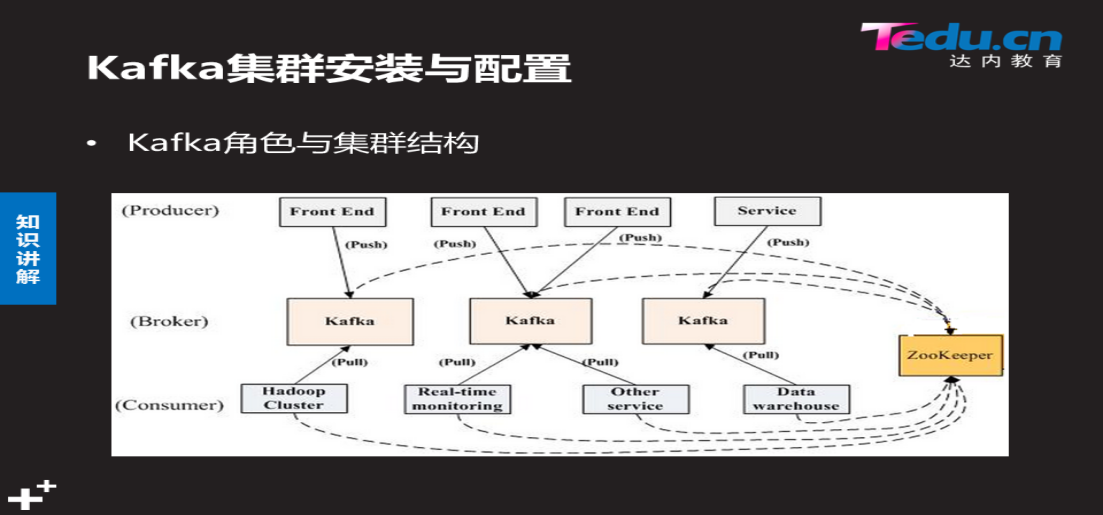
消费者：读取处理消息

topic：消息的类别(包含一个或多个Partition)

Broker：Kafka集群

Kafka通过Zookeeper管理集群配置，选举Leader

**-Kafka工作原理**



**步骤1：Kafka安装与配置**

#先创建一个可用的Zookeeper集群

[root@zk1 ~]# yum install -y java-1.8.0-openjdk-devel

[root@zk1 ~]# tar -xf kafka\_2.10-0.10.2.1.tgz

[root@zk1 ~]# mv kafka\_2.10-0.10.2.1/ /usr/local/kafka

#安装Kafka

[root@zk1 kafka]# vim config/server.properties

broker.id=22

zookeeper.connect=zk1:2181,zk2:2181,zk3:2181

#配置Kafka的broker.id、zookeeper的集群地址

[root@zk1 ~]# for i in zk{2..3};do

> rsync -avz --delete /usr/local/kafka $i:/usr/local/

> done

#同步安装目录

[root@zk2 ~]# vim /usr/local/kafka/config/server.properties

broker.id=23

[root@zk3 ~]# vim /usr/local/kafka/config/server.properties

broker.id=24

#修改zk2、zk3的broker.id唯一

**步骤2：启动集群、验证**

[root@zk1 kafka]# ./bin/kafka-server-start.sh -daemon \

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

#每一台都启动

[root@zk1 ~]# jps

**步骤3：测试配置，创建一个topic**

[root@zk1 kafka]# ./bin/kafka-topics.sh --create --partitions 1 \

> --replication-factor 1 \

> --zookeeper node3:2181 --topic aa

Created topic "aa".

**步骤4：模拟生产者，发布消息**

[root@zk2 kafka]# ./bin/kafka-console-producer.sh \

> --broker-list node2:9092 --topic aa //写一个数据

ccc

ddd

**步骤5：模拟消费者，接收消息**

[root@zk3 kafka]# ./bin/kafka-console-consumer.sh \

> --bootstrap-server node1:9092 --topic aa //这边会直接同步

ccc

ddd

**Afternoon：**

1. Hadoop高可用概述

**-各节点的高可用**

hdfs： NameNode(重要)

SecondaryNameNode

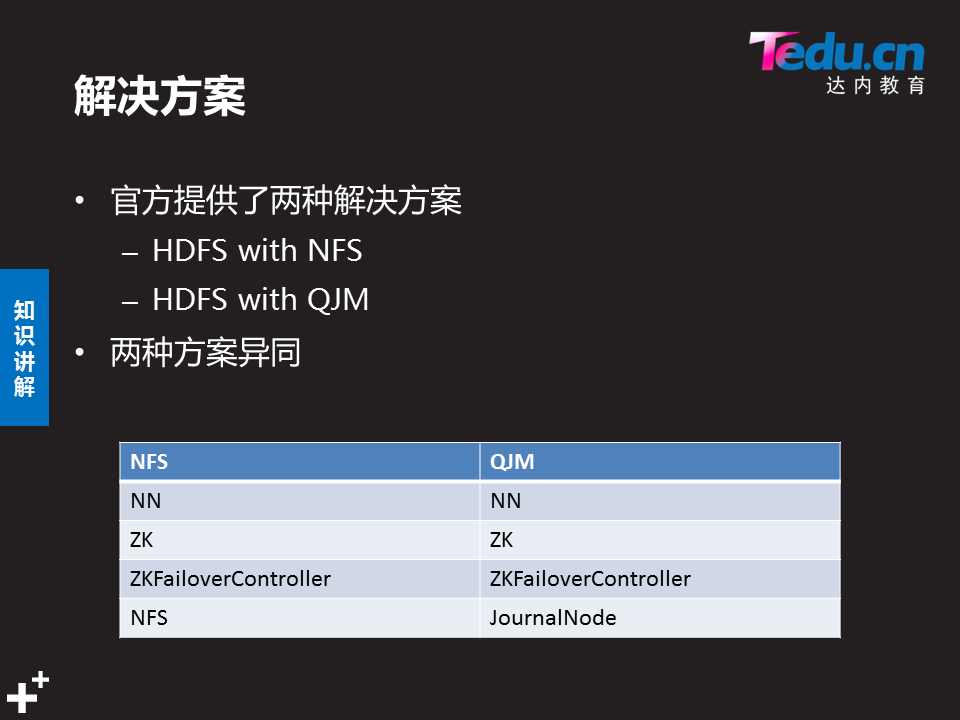
DataNode(配置参数dfs.replication=2备份已解决)

Yarn： Resourcemanager

Nodemanager(无需高可用，计算速度慢一点)

**-NameNode高可用方案对比**

1）方案对比



共同点：

2个NameNode（Active和Standby）

使用Zookeeper协调NameNode的主备关系

ZKFailoverController实现故障切换(使用ssh执行脚本的方法)

不同点：

数据存储的位置不一样。

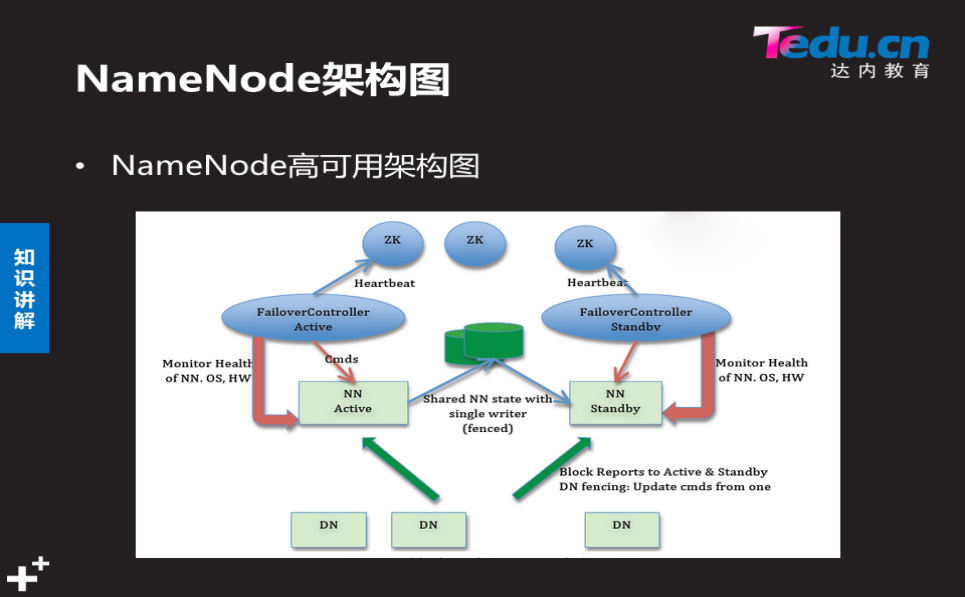
NFS：通过挂载实现数据一致（但还需要考虑NFS的高可用）；

QJM：DataNode需要知道2台NN的位置、并把块信息和心跳包发送给2台NN（增加配置的复杂度），通过JournalNode把数据变更日志进行主从同步（类似Mysql）。

为了保证ActiveNN与StandbyNN保持同步，2台NN都需要与JNS(JournalNodes)保持通信；当ActiveNN宕机，StandbyNN从JNS读取所有日志。

为了防止“脑裂”(出现双Active)，Active只负责向JNS写日志，Standby负责从JNS读取。

2）架构图



2、搭建NameNode、ResourceManager高可用

**步骤1：系统规划**

|  |  |
| --- | --- |
| 192.168.1.10 nn01 | Namenode，Resourcemanager |
| 192.168.1.20 nn02 | Namenode，Resourcemanager |
| 192.168.1.11 node1 | Datanode，Nodemanager |
| 192.168.1.12 node2 | Datanode，Nodemanager |
| 192.168.1.13 node3 | Datanode，Nodemanager |
| 192.168.1.21 zk1 | Zookeeper，Journalnode |
| 192.168.1.22 zk2 | Zookeeper，Journalnode |
| 192.168.1.23 zk3 | Zookeeper，Journalnode |

Tips：Secondary Namenode被Journalnode替代

**步骤2：环境准备**

1）准备机器，安装java环境

2）停止所有服务

[root@\*\* ~]# reboot

[root@\*\* ~]# jps

1234.. Jps

3）清空hadoop数据目录（所有节点）

[root@node\* ~]# rm -rf /tmp/hadoop/\*

4）启动zookeeper集群

[root@zk\* ~]# /usr/local/zookeeper/bin/zkServer.sh start

1. 配置/etc/hosts（所有）、nn01~02可以免密登陆所有(包括自己)

[root@nn01 ~]# vim /etc/ssh/ssh\_config

Host \*

StrictHostKeyChecking no #取消密钥检查

...

[root@nn01 ~]# for i in nn01 node{1..3} zk{1..3}; do

> ssh-copy-id -i /root/.ssh/

> ssh-copy-id -i /root/.ssh/id\_rsa.pub $i

> done

[root@nn02 ~]# rsync -avz --delete 192.168.1.10:/root/.ssh /root/

#nn02直接将nn01的.ssh/同步过来

6）创建/tmp/hadoop（所有机器）

[root@\*\* ~]# mkdir /tmp/hadoop

7）需配置以下配置文件：

hadoop-env.sh (配置JAVA与hadoop环境目录，不需改)

core-site.xml

hdfs-site.xml

mapred-site.xml (不需改)

yarn-site.xml

slaves (不需改)

**步骤3：NameNode高可用配置**

[root@nn01 hadoop]# vim etc/hadoop/core-site.xml

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://nsd1808ha</value>

</property>

#指定文件系统为nsd1808ha组(nn01+nn02)的hdfs

<property>

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

<value>/tmp/hadoop</value>

</property>

#定义hadoop根目录

<property>

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

<value>zk1:2181,zk2:2181,zk3:2181</value>

</property>

#声明zookeeper

<property>

<name>hadoop.proxyuser.nsd1808.groups</name>

<value>\*</value>

</property>

<property>

<name>hadoop.proxyuser.nsd1808.hosts</name>

<value>\*</value>

</property>

#NFS网关的代理用户授权，可不配置

</configuration>

[root@nn01 hadoop]# vim etc/hadoop/hdfs-site.xml

<configuration>

<property>

<name>dfs.nameservices</name>

<value>nsd1808ha</value>

</property>

#声明组的名称

<property>

<name>dfs.ha.namenodes.nsd1808ha</name>

<value>nn1,nn2</value>

</property>

#指定组的角色名(并不是主机名)

<property>

<name>dfs.namenode.rpc-address.nsd1808ha.nn1</name>

<value>nn01:8020</value>

</property>

<property>

<name>dfs.namenode.rpc-address.nsd1808ha.nn2</name>

<value>nn02:8020</value>

</property>

#声明角色nn1-2的rpc通信端口

<property>

<name>dfs.namenode.http-address.nsd1808ha.nn1</name>

<value>nn01:50070</value>

</property>

<property>

<name>dfs.namenode.http-address.nsd1808ha.nn2</name>

<value>nn02:50070</value>

</property>

#声明角色nn1-2的http通信端口(接收DataNode的数据块与心跳包)

<property>

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

<value>qjournal://zk1:8485;zk2:8485;zk3:8485/nsd1808ha</value>

</property>

#定义Journalnode的集群地址

<property>

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

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

</property>

#定义数据变更日志的数据存储路径

<property>

<name>dfs.client.failover.proxy.provider.nsd1808ha</name>

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

</property>

#定义FailoverController健康检测(引用一个java类实现检测)

<property>

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

<value>sshfence</value>

</property>

#配置健康检测机制为SSH(引用一个java调用ssh的类)

<property>

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

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

</property>

#指定SSH私钥的位置

<property>

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

<value>true</value>

</property>

#开启故障自动切换

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

#定义备份策略

<property>

<name>dfs.hosts.exclude</name>

<value>/usr/local/hadoop/etc/hadoop/exclude</value>

</property>

#定义节点黑名单

</configuration>

**步骤4：ResourceManager高可用配置**

#ResourceManager高可用的思路与NN一样(定义组+zookeeper)

[root@nn01 hadoop]# vim etc/hadoop/yarn-site.xml

<configuration>

<property>

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

<value>mapreduce\_shuffle</value>

</property>

<property>

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

<value>true</value>

</property>

#开启ResourceManager的高可用

<property>

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

<value>rm1,rm2</value>

</property>

#定义两个角色(并不是主机名)

<property>

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

<value>true</value>

</property>

#打开ResourceManager检测

<property>

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

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

</property>

#定义ResourceManager存储的类

<property>

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

<value>zk1:2181,zk2:2181,zk3:2181</value>

</property>

#声明zookeeper集群地址

<property>

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

<value>yarn-ha</value>

</property>

#声明组的名称

<property>

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

<value>nn01</value>

</property>

<property>

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

<value>nn02</value>

</property>

#声明角色rm1-2的主机名

</configuration>

**步骤5：初始化hadoop集群**

ALL：表示所有机器node{1..3} zk{1..3} nn0{1..2}

nodeX：表示node{1..3}

zkX：表示zk{1..3}

NN1：表示nn01

NN2：表示nn02

1）ALL：同步hadoop配置到所有集群机器

[root@nn01 ~]# for i in nn02 node{1..3} zk{1..3}; do

> rsync -avz --delete /usr/local/hadoop $i:/usr/local/ &

> done

[root@nn01 ~]# wait

1. NN1：初始化ZK集群

[root@nn01 ~]# /usr/local/hadoop/bin/hdfs zkfc -formatZK

19/01/04 17:58:39 INFO ha.Act..: Successfully created ... in ZK.

1. zkX：测试启动journalnode服务

[root@zkX hadoop]# ./sbin/hadoop-daemon.sh start journalnode

[root@zkX hadoop]# jps

JournalNode

QuorumPeerMain

1. NN1：hdfs文件系统格式化

[root@nn01 hadoop]# ./bin/hdfs namenode -format

#相当于在/tmp/hadoop/创建初始目录dfs

#dfs/name目录为fs.images的存放位置，初始化后有随机ID文件

1. NN2：同步NN1格式化后的目录

[root@nn02 ~]# rsync -avz --delete nn01:/tmp/hadoop/dfs \

> /tmp/hadoop/

#保证dfs/name目录下的随机ID一致

1. NN1：初始化JournalNode

[root@nn01 hadoop]# ./bin/hdfs namenode -initializeSharedEdits

QJM输入：Y

1. zkX：停止Journalnode

[root@zkX hadoop]# ./sbin/hadoop-daemon.sh stop journalnode

#Journalnode应该由hadoop自动管理启动，因此把它停止

**步骤6：启动hadoop集群**

1. NN1：启动all集群

[root@nn01 hadoop]# ./sbin/start-all.sh

starting namenode logging to /var/log/hadoop/hadoop-root-namenode-nn01.out

starting datanode, logging to /var/log/hadoop/hadoop-root-datanode-node01.out

starting journalnode, logging to /var/log/hadoop/hadoop-root-journalnode-zk01.out

starting zkfc, logging to /var/log/hadoop/hadoop-root-zkfc-nn01.out

starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-root-resou...-nn01.out

nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nodemanager-node01.out

1. NN2：启动yarn-daemon

[root@nn02 hadoop]# ./sbin/yarn-daemon.sh start \

> resourcemanager

**步骤7：验证hadoop集群**

1）使用jps查看

[root@nn0{1..2} ~]# jps

DFSZKFailoverController

ResourceManager

NameNode

[root@node{1..3} ~]# jps

DataNode

NodeManager

[root@zk{1..3} ~]# jps

QurumPeerMain

JournalNode

1. 在nn0\*查看集群状态

[root@nn01 hadoop]# ./bin/hdfs dfsadmin -report

Live datanodes (3) #验证hdfs集群

[root@nn01 hadoop]# ./bin/yarn node -list

Total Nodes：3 #验证yarn集群

[root@nn01 hadoop]# ./bin/hdfs haadmin -getServiceState nn\*

active或standby #验证NameNode高可用

[root@nn01 hadoop]# ./bin/yarn rmadmin -getServiceState rm\*

active或standby #验证ResourceManager高可用

[root@nn01 hadoop]# ./bin/hadoop fs -mkdir hdfs://nsd1808ha/xx

[root@nn01 hadoop]# ./bin/hadoop fs -ls /

>或./bin/hadoop fs -ls hdfs://nsd1808ha/

hdfs://nsd1808ha/ooxx