全部开发者教程

1 初始NameNode

2 NameNode进阶

3 HDFS高级

4 Hadoop核心复盘

第5周-Hadoop之初识MR

1 初始MapReduce

2 实战：WordCount

3 深入MapReduce

4 精讲Shuffle执行过程及源码分析输入输出

第6周-拿来就用的企业级解决方案

1 剖析小文件问题与企业级解决方案

徐老师 • 更新于 2020-08-23

上一节 2 NameNode进... 4 Hadoop核心... 下一节

HDFS的回收站

我们windows系统里面有一个回收站，当想恢复删除的文件的话就可以到这里面进行恢复，HDFS也有回收站。

HDFS会为每一个用户创建一个回收站目录：/user/用户名/.Trash/，每一个被用户在Shell命令行删除的文件/目录，会进入到对应的回收站目录中，在回收站中的数据都有一个生存周期，也就是当回收站中的文件/目录在一段时间之内没有被用户恢复的话，HDFS就会自动的把这个文件/目录彻底删除，之后，用户就永远也找不回这个文件/目录了。

默认情况下hdfs的回收站是没有开启的，需要通过一个配置来开启，在core-site.xml中添加如下配置，value的单位是分钟，1440分钟表示是一天的生存周期

代码块

```
<property>
    <name>fs.trash.interval</name>
    <value>1440</value>
</property>
```

在修改配置信息之前先验证一下删除操作，显示的是直接删除掉了。

代码块

```
[root@bigdata01 hadoop-3.2.0]# hdfs dfs -rm -r /NOTICE.txt
Deleted /NOTICE.txt
```

修改回收站配置，先在bigdata01上操作，然后再同步到其它两个节点，先停止集群

代码块

```
[root@bigdata01 hadoop-3.2.0]# sbin/stop-all.sh
[root@bigdata01 hadoop-3.2.0]# vi etc/hadoop/core-site.xml
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://bigdata01:9000</value>
    </property>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/data/hadoop_repo</value>
   </property>
    <property>
        <name>fs.trash.interval</name>
        <value>1440</value>
    </property>
</configuration>
[root@bigdata01 hadoop-3.2.0]# scp -rq etc/hadoop/core-site.xml bigdata02:/da
[root@bigdata01 hadoop-3.2.0]# scp -rq etc/hadoop/core-site.xml bigdata03:/da
```

启动集群，再执行删除操作

代码块

```
[root@bigdata01 hadoop-3.2.0]# sbin/start-all.sh
[root@bigdata01 hadoop-3.2.0]# hdfs dfs -rm -r /README.txt
2020-04-09 11:43:47,664 INFO fs.TrashPolicyDefault: Moved: 'hdfs://bigdata01:
```

此时看到提示信息说把删除的文件移到到了指定目录中，其实就是移动到了当前用户的回收站目录。

回收站的文件也是可以下载到本地的。其实在这回收站只是一个具备了特殊含义的HDFS目录。

> 注意：如果删除的文件过大，超过回收站大小的话会提示删除失败
> 需要指定参数 -skipTrash ，指定这个参数表示删除的文件不会进回收站

代码块

```
[root@bigdata01 hadoop-3.2.0]# hdfs dfs -rm -r -skipTrash /user.txt
Deleted /user.txt
```

HDFS的安全模式

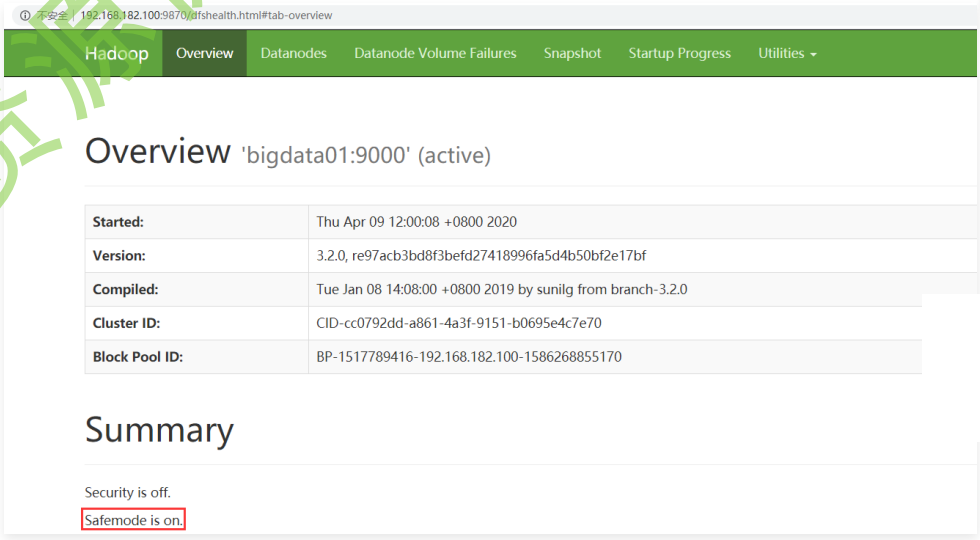
大家在平时操作HDFS的时候，有时候可能会遇到这个问题，特别是刚启动集群的时候去上传或者删除文件，会发现报错，提示NameNode处于safe mode。

这个属于HDFS的安全模式，因为在集群每次重新启动的时候，HDFS都会检查集群中文件信息是否完整，例如副本是否缺少之类的信息，所以这个时间段内是不允许对集群有修改操作的，如果遇到了这个情况，可以稍微等一会，等HDFS自检完毕，就会自动退出安全模式。

代码块

```
[root@bigdata01 hadoop-3.2.0]# hdfs dfs -rm -r /hadoop-3.2.0.tar.gz
2020-04-09 12:00:36,646 WARN fs.TrashPolicyDefault: Can't create trash direct
org.apache.hadoop.hdfs.server.namenode.SafeModeException: Cannot create direc
```

此时访问HDFS的web ui界面，可以看到下面信息，on表示处于安全模式，off表示安全模式已结束



或者通过hdfs命令也可以查看当前的状态

代码块

```
[root@bigdata01 hadoop-3.2.0]# hdfs dfsadmin -safemode get
Safe mode is ON
```

如果想快速离开安全模式，可以通过命令强制离开，正常情况下建议等HDFS自检完毕，自动退出

代码块

```
[root@bigdata01 hadoop-3.2.0]# hdfs dfsadmin -safemode leave
Safe mode is OFF
```

此时，再操作HDFS中的文件就可以了。

实战：定时上传数据至HDFS

> 需求分析：
> 在实际工作中会有定时上传数据到HDFS的需求，我们有一个web项目每天都会产生日志文件，日志文件的格式为access_2020_01_01.log这种格式的，每天产生一个，我们需要每天凌晨将昨天生成的日志文件上传至HDFS上，按天分目录存储，HDFS上的目录格式为20200101

针对这个需求，我们需要开发一个shell脚本，方便定时调度执行

第一步：我们需要获取到昨天日志文件的名称

第二步：在HDFS上面使用昨天的日期创建目录

第三步：将昨天的日志文件上传到刚创建的HDFS目录中

第四步：要考虑到脚本重跑，补数据的情况

第五步：配置crontab任务

开始开发shell脚本，脚本内容如下：

代码块

```
[root@bigdata01 ~]# mkdir -p  /data/shell
[root@bigdata01 ~]# cd /data/shell
[root@bigdata01 shell]# vi uploadLogData.sh
#!/bin/bash

# 获取昨天日期字符串
yesterday=$1
if [ "$yesterday" = "" ]
then
        yesterday=`date +%Y_%m_%d --date="1 days ago"`
fi

# 拼接日志文件路径信息
logPath=/data/log/access_${yesterday}.log

# 将日期字符串中的_去掉
hdfsPath=/log/${yesterday//_/}
# 在hdfs上创建目录
hdfs dfs -mkdir -p ${hdfsPath}
# 将数据上传到hdfs的指定目录中
hdfs dfs -put  ${logPath} ${hdfsPath}
```

生成测试数据，注意，文件名称中的日期根据昨天的日期命名

代码块

```
[root@bigdata01 shell]# mkdir -p /data/log
[root@bigdata01 shell]# cd /data/log
[root@bigdata01 log]# vi access_2020_04_08.log
log1
```

```
7    log3
```

执行脚本

代码块

```
1    [root@bigdata01 log]# cd /data/shell/
2    [root@bigdata01 shell]# sh -x uploadLogData.sh
3    + yesterday=
4    + '[' '' = '' ']'
5    ++ date +%Y_%m_%d '--date=1 days ago'
6    + yesterday=2020_04_08
7    + logPath=/data/log/access_2020_04_08.log
8    + hdfsPath=/log/20200408
9    + hdfs dfs -mkdir -p /log/20200408
10   + hdfs dfs -put /data/log/access_2020_04_08.log /log/20200408
11   [root@bigdata01 shell]# hdfs dfs -ls /log/20200408
12   Found 1 items
13   -rw-r--r--   2 root supergroup         15 2020-04-09 16:05 /log/20200408/acce
```

> 注意：如果想要指定日期上传数据，可以通过在脚本后面传递参数实现

先创建一个日期的测试数据

代码块

```
1    [root@bigdata01 shell]# cd /data/log/
2    [root@bigdata01 log]# cp access_2020_04_08.log  access_2020_01_01.log
```

执行脚本

代码块

```
1    [root@bigdata01 log]# cd /data/shell/
2    [root@bigdata01 shell]# sh -x uploadLogData.sh 2020_01_01
3    + yesterday=2020_01_01
4    + '[' 2020_01_01 = '' ']'
5    + logPath=/data/log/access_2020_01_01.log
6    + hdfsPath=/log/20200101
7    + hdfs dfs -mkdir -p /log/20200101
8    + hdfs dfs -put /data/log/access_2020_01_01.log /log/20200101
9    [root@bigdata01 shell]# hdfs dfs -ls /log/20200101
10   Found 1 items
11   -rw-r--r--   2 root supergroup         15 2020-04-09 16:17 /log/20200101/acce
```

这样后期如果遇到某天的数据漏传了，或者需要重新上传，就可以通过手工指定日期实现上传操作，
际工作中这种操作是不可避免的，所以我们在开发脚本的时候就直接考虑好补数据的情况，别等需
时候了再去增加这个功能。

最后配置crontab定时任务，每天凌晨1点执行

代码块

```
1    [root@bigdata01 shell]# vi /etc/crontab
2    0 1 * * * root sh /data/shell/uploadLogData.sh >> /data/shell/uploadLogData.l
```

HDFS的高可用和高扩展

我们前面分析了NameNode负责接收用户的操作请求，所有的读写请求都会经过它，如果它挂了怎么办?

这个时候集群是不是就无法正常提供服务了？是的，那现在我们这个集群就太不稳定了，因为NameNode只有一个，是存在单点故障的，咱们在现实生活中，例如，县长，是有正的和副的，这样就是为了解决当正县长遇到出差的时候，副县长可以顶上去。

所以在HDFS的设计中，NameNode也是可以支持多个的，一个主的 多个备用的，，当主的挂掉了，备用的可以顶上去，这样就可以解决NameNode节点宕机导致的单点故障问题了，也就实现了HDFS的高可用

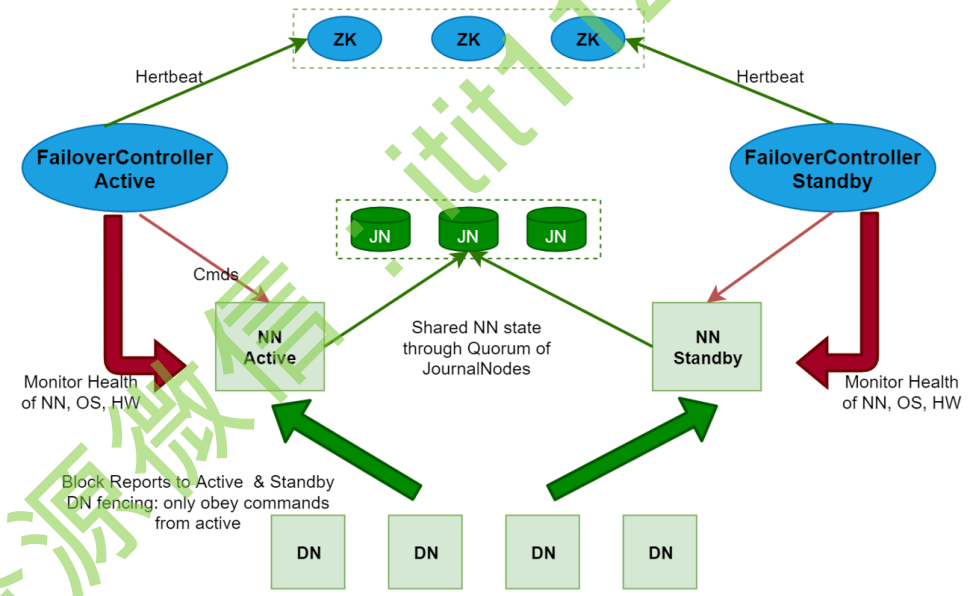
还有一个问题是，前面我们说了NameNode节点的内存是有限的，只能存储有限的文件个数，那使用一个主NameNode，多个备用的NameNode能解决这个问题吗?

不能!

一个主NameNode，多个备用的NameNode的方案只能解决NameNode的单点故障问题，无法解决单个NameNode内存不够用的问题，那怎么办呢？不用担心，官方提供了Federation机制，可以翻译为联邦，它可以解决单节点内存不够用的情况，具体实现思路我们稍后分析，这个就是HDFS的高扩展

HDFS的高可用(HA)

下面我们首先来看一下HDFS的高可用，也可以称之为HA(High Available)



HDFS的HA，指的是在一个集群中存在多个NameNode，分别运行在独立的物理节点上。在任何时间点，只有一个NameNode是处于Active状态，其它的是处于Standby状态。 Active NameNode（简写为Active NN）负责所有的客户端的操作，而Standby NameNode（简写为Standby NN）用来同步Active NameNode的状态信息，以提供快速的故障恢复能力。

为了保证Active NN与Standby NN节点状态同步，即元数据保持一致。除了DataNode需要向NameNode发送block位置信息外，还构建了一组独立的守护进程"JournalNodes"（简写为JN），用来同步Edits信息。当Active NN执行任何有关命名空间的修改，它需要持久化到一半以上的JNs上。Standby NN负责观察JNs的变化，读取从Active NN发送过来的Edits信息，并更新自己内部的命名空间。一旦Active NN遇到错误，Standby NN需要保证从JNs中读出了全部的Edits，然后切换成Active状态，如果有多个Standby NN，还会涉及到选主的操作，选择一个切换为Active 状态。

> 需要注意一点，为了保证Active NN与Standby NN节点状态同步，即元数据保持一致

这里的元数据包含两块，一个是静态的，一个是动态的

静态的是fsimage和edits，其实fsimage是由edits文件合并生成的，所以只需要保证edits文件内容的一致性。这个就是需要保证多个NameNode中edits文件内容的事务性同步。这块的工作是由JournalNodes集群进行同步的

动态数据是指block和DataNode节点的信息，这个如何保证呢?
当DataNode启动的时候，上报数据信息的时候需要向每个NameNode都上报一份。
这样就可以保证多个NameNode的元数据信息都一样了，当一个NameNode down掉以后，立刻从Standby NN中选择一个进行接管，没有影响，因为每个NameNode 的元数据时刻都是同步的。

> 注意:使用HA的时候，不能启动SecondaryNameNode，会出错。
> 之前是SecondaryNameNode负责合并edits到fsimage文件 那么现在这个工作被standby NN负责了。

NameNode 切换可以自动切换，也可以手工切换，如果想要实现自动切换，需要使用到zookeeper集群。

使用zookeeper集群自动切换的原理是这样的

当多个NameNode 启动的时候会向zookeeper中注册一个临时节点，当NameNode挂掉的时候，这个临时节点也就消失了，这属于zookeeper的特性，这个时候，zookeeper就会有一个watcher监视器监视到，就知道这个节点down掉了，然后会选择一个节点转为Active，把down掉的节点转为Standby。

注意：下面的配置步骤建议大家有空闲时间了再来操作就行，作为一个课外扩展，因为在工作中这个配置是不需要我们做的，我们只需要知道这种特性即可。

下面开始配置HDFS 的HA
HA集群规划

代码块

```
                namenode    datanode    journalnode     zkfc    zookeeper
   bigdata01    yes                     yes             yes     yes
   bigdata02    yes         yes         yes             yes     yes
   bigdata03                yes         yes             yes     yes
```

解释：针对HDFS的HA集群，在这里我们只需要启动HDFS相关的进程即可，YARN相关的进程可以不启动，它们两个的进程本来就是相互独立的。
在HDFS的HA集群中，就不需要启动SecondaryNameNode进程了

- namenode：hdfs的主节点
- datanode：hdfs的从节点
- journalnode：JournalNode进程，用来同步Edits信息的
- zkfc(DFSZKFailoverController)：监视namenode的状态，负责切换namenode节点的状态
- zookeeper(QuorumPeerMain)：保存ha集群的节点状态信息

环境准备：三个节点

代码块

```
   bigdata01 192.168.182.100
   bigdata02 192.168.182.101
   bigdata03 192.168.182.102
```

每个节点的基础环境都要先配置好，先把ip、hostname、firewalld、ssh免密码登录、host、免密码登录，JDK这些基础环境配置好
我们目前使用的这几台机器之前已经搭建过分布式集群，所以这些基础环境都是没有问题的

> 但是注意：有一点还需要完善一下，由于namenode在进行故障切换的时候，需要在两个namenode节点之间互相使用ssh进行连接，所以需要实现这两个namenode之间的互相免密码登录，目前我们只实现了bigdata01免密码登录到bigdata02，所以还需要实现bigdata02免密码登录到bigdata01，这一步如果不做，后期无法实现namenode故障自动转移。

代码块

```
[root@bigdata02 hadoop]# scp ~/.ssh/authorized_keys bigdata01:~/
root@bigdata01's password: 输入密码
authorized_keys                     100%  792   456.2KB/s   00:00
[root@bigdata01 hadoop]# cat ~/authorized_keys  >> ~/.ssh/authorized_keys
```

然后验证一下bigdata02是否可以免密码登录bigdata01，只要可以不输入密码就能连接进去就说明免密码登录搞定了。

代码块

```
[root@bigdata02 hadoop]# ssh bigdata01
Last login: Fri Feb  6 23:54:41 2026 from 192.168.182.1
```

接着把bigdata01、bigdata02、bigdata03中之前安装的hadoop删掉，删除解压的目录，以及hadoop_repo目录。

> 注意：我们需要把bigdata01、bigdata02、bigdata03节点中/data目录下的hadoop_repo目录和/data/soft下的hadoop-3.2.0目录删掉，恢复这些节点的环境，这里面记录的有之前集群的一些信息。

代码块

```
[root@bigdata01 ~]# rm -rf  /data/soft/hadoop-3.2.0
[root@bigdata01 ~]# rm -rf  /data/hadoop_repo
[root@bigdata02~]# rm -rf  /data/soft/hadoop-3.2.0
[root@bigdata02 ~]# rm -rf  /data/hadoop_repo
[root@bigdata03 ~]# rm -rf  /data/soft/hadoop-3.2.0
[root@bigdata04 ~]# rm -rf  /data/hadoop_repo
```

我们在这里需要使用到zookeeper这个组件，所以先把它安装起来。

1. 集群节点规划，使用三个节点搭建一个zookeeper集群

代码块

```
bigdata01
bigdata02
bigdata03
```

2. 首先在bigdata01节点上配置zookeeper

   解压

代码块

```
[root@bigdata01 soft]# tar -zxvf apache-zookeeper-3.5.8-bin.tar.gz
```

修改配置

代码块

```
dataDir=/data/soft/apache-zookeeper-3.5.8-bin/data
server.0=bigdata01:2888:3888
server.1=bigdata02:2888:3888
server.2=bigdata03:2888:3888
```

创建目录保存myid文件，并且向myid文件中写入内容
myid中的值其实是和zoo.cfg中server后面指定的编号是一一对应的
编号0对应的是bigdata01这台机器，所以在这里指定0
在这里使用echo 和 重定向 实现数据写入

代码块

```
[root@bigdata01 conf]#cd /data/soft/apache-zookeeper-3.5.8-bin
[root@bigdata01 apache-zookeeper-3.5.8-bin]# mkdir data
[root@bigdata01 apache-zookeeper-3.5.8-bin]# cd data
[root@bigdata01 data]# echo 0 > myid
```

3. 把修改好配置的zookeeper拷贝到其它两个节点

代码块

```
[root@bigdata01 soft]# scp -rq apache-zookeeper-3.5.8-bin bigdata02:/data/soft
[root@bigdata01 soft]# scp -rq apache-zookeeper-3.5.8-bin bigdata03:/data/soft
```

4. 修改bigdata02和bigdata03上zookeeper中myid文件的内容

首先修改bigdata02节点上的myid文件

代码块

```
[root@bigdata02 ~]# cd /data/soft/apache-zookeeper-3.5.8-bin/data/
[root@bigdata02 data]# echo 1 > myid
```

然后修改bigdata03节点上的myid文件

代码块

```
[root@bigdata03 ~]# cd /data/soft/apache-zookeeper-3.5.8-bin/data/
[root@bigdata03 data]# echo 2 > myid
```

5. 启动zookeeper集群

分别在bigdata01、bigdata02、bigdata03上启动zookeeper进程

在bigdata01上启动

代码块

```
[root@bigdata01 apache-zookeeper-3.5.8-bin]# bin/zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /data/soft/apache-zookeeper-3.5.8-bin/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

在bigdata02上启动

代码块

```
[root@bigdata02 apache-zookeeper-3.5.8-bin]# bin/zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /data/soft/apache-zookeeper-3.5.8-bin/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

代码块

```
[root@bigdata03 apache-zookeeper-3.5.8-bin]# bin/zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /data/soft/apache-zookeeper-3.5.8-bin/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

6. 验证

分别在bigdata01、bigdata02、bigdata03上执行jps命令验证是否有QuorumPeerMain进程

如果都有就说明zookeeper集群启动正常了

如果没有就到对应的节点的logs目录下查看zookeeper*-*.out日志文件

执行bin/zkServer.sh status 命令会发现有一个节点显示为leader，其他两个节点为follower

代码块

```
[root@bigdata01 apache-zookeeper-3.5.8-bin]# bin/zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /data/soft/apache-zookeeper-3.5.8-bin/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost.
Mode: follower
```

代码块

```
[root@bigdata02 apache-zookeeper-3.5.8-bin]# bin/zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /data/soft/apache-zookeeper-3.5.8-bin/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost.
Mode: leader
```

代码块

```
[root@bigdata03 apache-zookeeper-3.5.8-bin]# bin/zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /data/soft/apache-zookeeper-3.5.8-bin/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost.
Mode: follower
```

7. 停止zookeeper集群

最后想要停止zookeeper集群的时候可以在bigdata01、bigdata02、bigdata03三台机器上分别执行 `bin/zkServer.sh stop` 命令即可。

接下来我们来配置Hadoop集群

先在bigdata01节点上进行配置

1. 解压hadoop安装包

代码块

```
[root@bigdata01 soft]# tar -zxvf hadoop-3.2.0.tar.gz
```

2. 修改hadoop相关配置文件

进入配置文件所在目录

代码块

```
[root@bigdata01 soft]# cd hadoop-3.2.0/etc/hadoop/
[root@bigdata01 hadoop]#
```

代码块

```
[root@bigdata01 hadoop]# vi hadoop-env.sh
export JAVA_HOME=/data/soft/jdk1.8
export HADOOP_LOG_DIR=/data/hadoop_repo/logs/hadoop
```

修改core-site.xml文件

代码块

```
[root@bigdata01 hadoop]# vi core-site.xml
<configuration>
    # mycluster是集群的逻辑名称，需要和hdfs-site.xml中dfs.nameservices值一致
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://mycluster</value>
    </property>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/data/hadoop_repo</value>
   </property>
    # 用户角色配置，不配置此项会导致web页面报错
    <property>
        <name>hadoop.http.staticuser.user</name>
        <value>root</value>
    </property>
    # zookeeper集群地址
    <property>
        <name>ha.zookeeper.quorum</name>
        <value>bigdata01:2181,bigdata02:2181,bigdata03:2181</value>
    </property>
</configuration>
```

修改hdfs-site.xml文件

代码块

```
[root@bigdata01 hadoop]# vi hdfs-site.xml
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>2</value>
    </property>
    # 自定义的集群名称
    <property>
        <name>dfs.nameservices</name>
        <value>mycluster</value>
    </property>
    # 所有的namenode列表，逻辑名称，不是namenode所在的主机名
    <property>
        <name>dfs.ha.namenodes.mycluster</name>
        <value>nn1,nn2</value>
    </property>
    # namenode之间用于RPC通信的地址，value填写namenode所在的主机地址
    # 默认端口8020，注意mycluster与nn1要和前面的配置一致
    <property>
        <name>dfs.namenode.rpc-address.mycluster.nn1</name>
        <value>bigdata01:8020</value>
    </property>
    <property>
        <name>dfs.namenode.rpc-address.mycluster.nn2</name>
        <value>bigdata02:8020</value>
    </property>
    # namenode的web访问地址，默认端口9870
```

```
        <name>dfs.namenode.http-address.mycluster.nn1</name>
        <value>bigdata01:9870</value>
    </property>
    <property>
        <name>dfs.namenode.http-address.mycluster.nn2</name>
        <value>bigdata02:9870</value>
    </property>
    # journalnode主机地址，最少三台，默认端口8485
    <property>
        <name>dfs.namenode.shared.edits.dir</name>
        <value>qjournal://bigdata01:8485;bigdata02:8485;bigdata03:8485/myclus
    </property>
    # 故障时自动切换的实现类
    <property>
        <name>dfs.client.failover.proxy.provider.mycluster</name>
        <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverPr
    </property>
    # 故障时相互操作方式(namenode要切换active和standby)，使用ssh方式
    <property>
      <name>dfs.ha.fencing.methods</name>
      <value>sshfence</value>
    </property>
    # 修改为自己用户的ssh key存放地址
    <property>
      <name>dfs.ha.fencing.ssh.private-key-files</name>
      <value>/root/.ssh/id_rsa</value>
    </property>
    # namenode日志文件输出路径，即journalnode读取变更的位置
    <property>
        <name>dfs.journalnode.edits.dir</name>
        <value>/data/hadoop_repo/journalnode</value>
    </property>
    # 启用自动故障转移
    <property>
        <name>dfs.ha.automatic-failover.enabled</name>
        <value>true</value>
    </property>
</configuration>
```

mapred-site.xml和yarn-site.xml在这暂时就不修改了，因为我们只需要启动hdfs相关的服务。

修改workers文件，增加所有从节点的主机名，一个一行

代码块

```
[root@bigdata01 hadoop]# vi workers
bigdata02
bigdata03
```

修改启动脚本

修改 `start-dfs.sh` ， `stop-dfs.sh` 这两个脚本文件，在文件前面增加如下内容

代码块

```
[root@bigdata01 hadoop]# cd /data/soft/hadoop-3.2.0/sbin
[root@bigdata01 sbin]# vi start-dfs.sh
HDFS_DATANODE_USER=root
HDFS_DATANODE_SECURE_USER=hdfs
HDFS_NAMENODE_USER=root
HDFS_ZKFC_USER=root
HDFS_JOURNALNODE_USER=root

[root@bigdata01 sbin]# vi stop-dfs.sh
```

```
HDFS_DATANODE_SECURE_USER=hdfs
HDFS_NAMENODE_USER=root
HDFS_ZKFC_USER=root
HDFS_JOURNALNODE_USER=root
```

start-yarn.sh， stop-yarn.sh 这两个脚本暂时也不需要修改了，因为不启动YARN相关的进程用不到。

3. 把bigdata01节点上将修改好配置的安装包拷贝到其他两个从节点

代码块

```
[root@bigdata01 sbin]# cd /data/soft/
[root@bigdata01 soft]# scp -rq hadoop-3.2.0 bigdata02:/data/soft/
[root@bigdata01 soft]# scp -rq hadoop-3.2.0 bigdata03:/data/soft/
```

4. 格式化HDFS【此步骤只需要在第一次配置HA集群的时候操作一次即可】

注意：此时在格式化HDFS之前需要先启动所有的journalnode

代码块

```
[root@bigdata01 hadoop-3.2.0]# bin/hdfs --daemon start journalnode
[root@bigdata02 hadoop-3.2.0]# bin/hdfs --daemon start journalnode
[root@bigdata03 hadoop-3.2.0]# bin/hdfs --daemon start journalnode
```

接下来就可以对HDFS进行格式化了，此时在哪个namenode节点上操作都可以(bigdata01或者bigdata02)，在这我们使用bigdata01
能看到has been successfully formatted就说明hdfs格式化成功了

代码块

```
[root@bigdata01 hadoop-3.2.0]# bin/hdfs namenode -format
....
....
2026-02-07 00:35:06,212 INFO common.Storage: Storage directory /data/hadoop_r
2026-02-07 00:35:06,311 INFO namenode.FSImageFormatProtobuf: Saving image fil
2026-02-07 00:35:06,399 INFO namenode.FSImageFormatProtobuf: Image file /data
2026-02-07 00:35:06,405 INFO namenode.NNStorageRetentionManager: Going to ret
2026-02-07 00:35:06,432 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at bigdata01/192.168.182.100
************************************************************/
```

然后启动此namenode进程

代码块

```
[root@bigdata01 hadoop-3.2.0]# bin/hdfs --daemon start namenode
```

接下来在另一个namenode节点(bigdata02)上同步信息，看到下面的信息，则说明同步成功

代码块

```
[root@bigdata02 hadoop-3.2.0]# bin/hdfs namenode -bootstrapStandby
....
....
=====================================================
About to bootstrap Standby ID nn2 from:
           Nameservice ID: mycluster
        Other Namenode ID: nn1
  Other NN's HTTP address: http://bigdata01:9870
  Other NN's IPC  address: bigdata01/192.168.182.100:8020
```

```
            Block pool ID: BP-1332041116-192.168.182.100-1770395706205
               Cluster ID: CID-c12130ca-3a7d-4722-93b0-a79b0df3ed84
           Layout version: -65
       isUpgradeFinalized: true
================================================
2026-02-07 00:39:38,594 INFO common.Storage: Storage directory /data/hadoop_r
2026-02-07 00:39:38,654 INFO namenode.FSEditLog: Edit logging is async:true
2026-02-07 00:39:38,767 INFO namenode.TransferFsImage: Opening connection to
2026-02-07 00:39:38,854 INFO common.Util: Combined time for file download and
2026-02-07 00:39:38,855 INFO namenode.TransferFsImage: Downloaded file fsimag
2026-02-07 00:39:38,894 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at bigdata02/192.168.182.101
************************************************************/
```

5. 格式化zookeeper节点【此步骤只需要在第一次配置HA集群的时候操作一次即可】

   在任意一个节点上操作都可以，在这里我使用bigdata01节点

   能看到日志中输出Successfully created /hadoop-ha/mycluster in ZK.则说明操作成功

代码块

```
[root@bigdata01 hadoop-3.2.0]# bin/hdfs zkfc -formatZK
....
....
2026-02-07 00:42:17,212 INFO zookeeper.ClientCnxn: Socket connection establis
2026-02-07 00:42:17,220 INFO zookeeper.ClientCnxn: Session establishment comp
2026-02-07 00:42:17,244 INFO ha.ActiveStandbyElector: Successfully created /h
2026-02-07 00:42:17,249 INFO zookeeper.ZooKeeper: Session: 0x100001104b00098
2026-02-07 00:42:17,251 WARN ha.ActiveStandbyElector: Ignoring stale result f
2026-02-07 00:42:17,251 INFO zookeeper.ClientCnxn: EventThread shut down for
2026-02-07 00:42:17,254 INFO tools.DFSZKFailoverController: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down DFSZKFailoverController at bigdata01/192.168.182.
************************************************************/
```

6. 启动HDFS的HA集群

   注意：以后启动HDFS的HA集群直接使用这里面的命令即可，不需要再执行4、5步中的操作了

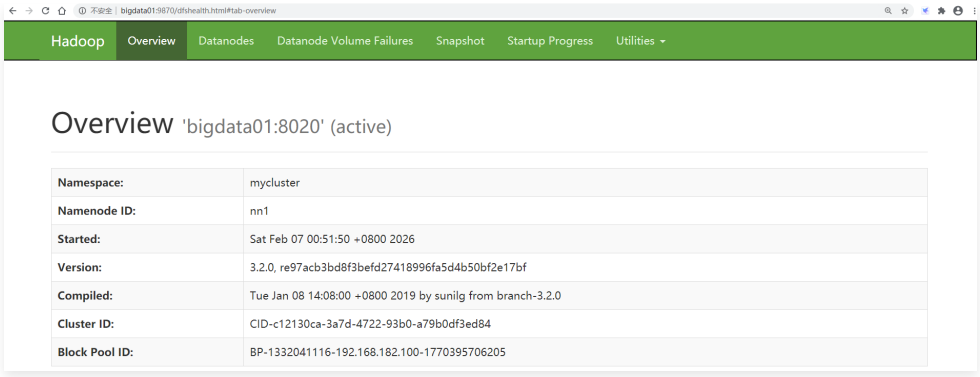
   在bigdata01上执行下面命令

代码块

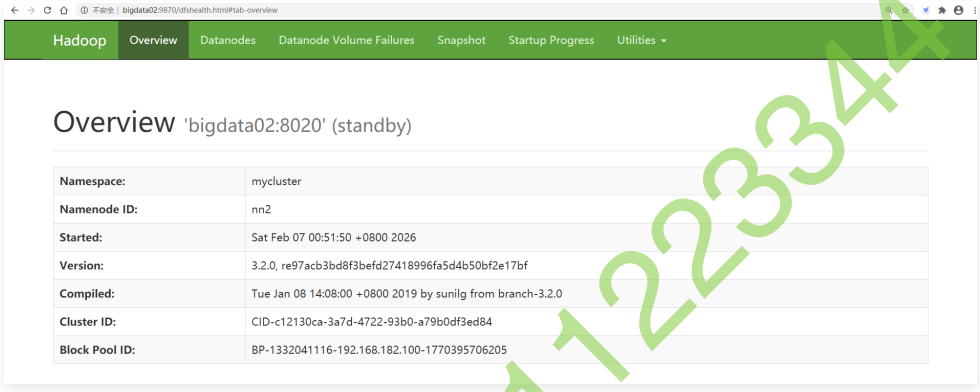
```
[root@bigdata01 hadoop-3.2.0]# sbin/start-dfs.sh
Starting namenodes on [bigdata01 bigdata02]
Last login: Sat Feb  7 00:02:27 CST 2026 on pts/0
bigdata01: namenode is running as process 6424.  Stop it first.
Starting datanodes
Last login: Sat Feb  7 00:47:13 CST 2026 on pts/0
Starting journal nodes [bigdata01 bigdata03 bigdata02]
Last login: Sat Feb  7 00:47:13 CST 2026 on pts/0
bigdata02: journalnode is running as process 4864.  Stop it first.
bigdata01: journalnode is running as process 6276.  Stop it first.
bigdata03: journalnode is running as process 2479.  Stop it first.
Starting ZK Failover Controllers on NN hosts [bigdata01 bigdata02]
Last login: Sat Feb  7 00:47:18 CST 2026 on pts/0
```

7. 验证HA集群

   此时访问两个namenode节点的9870端口，其中一个显示为Active，另一个显示为Standby



`http://bigdata02:9870/dfshealth.html`

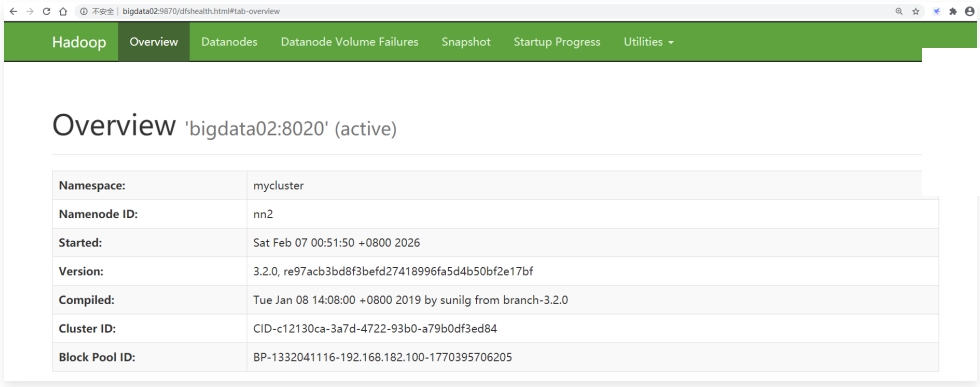


此时我们来手工停掉active状态的namenode，模拟namenode宕机的情况，验证一下另一个standby的namenode是否可以自动切换为active

代码块

```
[root@bigdata01 hadoop-3.2.0]# jps
8758 DFSZKFailoverController
8267 NameNode
1581 QuorumPeerMain
8541 JournalNode
8814 Jps
[root@bigdata01 hadoop-3.2.0]# kill 8267
[root@bigdata01 hadoop-3.2.0]# jps
8758 DFSZKFailoverController
1581 QuorumPeerMain
8541 JournalNode
8845 Jps
```

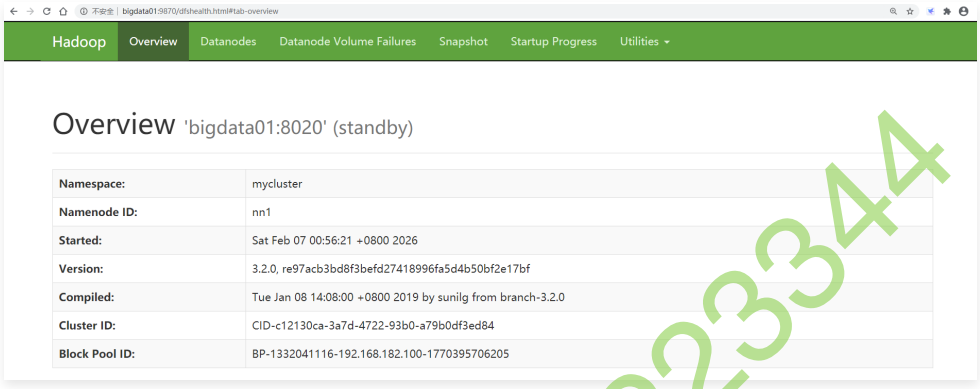
此时再刷新查看bigdata02的信息，会发现它的状态变为了active



接着我们再把bigdata01中的namenode启动起来，会发现它的状态变为了standby

代码块

```
[root@bigdata01 hadoop-3.2.0]# bin/hdfs --daemon start namenode
[root@bigdata01 hadoop-3.2.0]# jps
8898 NameNode
8758 DFSZKFailoverController
8967 Jps
1581 QuorumPeerMain
8541 JournalNode
```



通过前面的操作可以发现，现在的namenode其实就解决了单点故障的问题，实现了高可用。

现在我们再操作HDFS的时候就应该这样操作了。
这里面的mycluster就是在hdfs-site.xml中配置的dfs.nameservices属性的值。

代码块

```
[root@bigdata02 hadoop-3.2.0]# bin/hdfs dfs -ls hdfs://mycluster/
[root@bigdata02 hadoop-3.2.0]# bin/hdfs dfs -put README.txt hdfs://mycluster/
[root@bigdata02 hadoop-3.2.0]# bin/hdfs dfs -ls hdfs://mycluster/     Found 1
-rw-r--r--   2 root supergroup       1361 2026-02-07 00:58 hdfs://mycluster/R
```

8. 停止HDFS的HA集群

代码块

```
[root@bigdata01 hadoop-3.2.0]# sbin/stop-dfs.sh
Stopping namenodes on [bigdata01 bigdata02]
Last login: Sat Feb  7 00:52:01 CST 2026 on pts/0
Stopping datanodes
Last login: Sat Feb  7 01:03:23 CST 2026 on pts/0
Stopping journal nodes [bigdata01 bigdata03 bigdata02]
Last login: Sat Feb  7 01:03:25 CST 2026 on pts/0
Stopping ZK Failover Controllers on NN hosts [bigdata01 bigdata02]
Last login: Sat Feb  7 01:03:29 CST 2026 on pts/0
```

> 注意：大家在做完HA之后，再根据伪分布或者分布式集群的操作步骤重新操作一遍，因为后其课程学习中，我们不使用HA集群。
> 因为这种方式启动的进程太多了，我们使用的是虚拟机，比较影响性能，在这大家只要能理解HA这种特性就可以了。
> 后面课程中我们使用普通的集群。

HDFS的高扩展(Federation)

HDFS Federation可以解决单一命名空间存在的问题，使用多个NameNode，每个NameNode负责一个

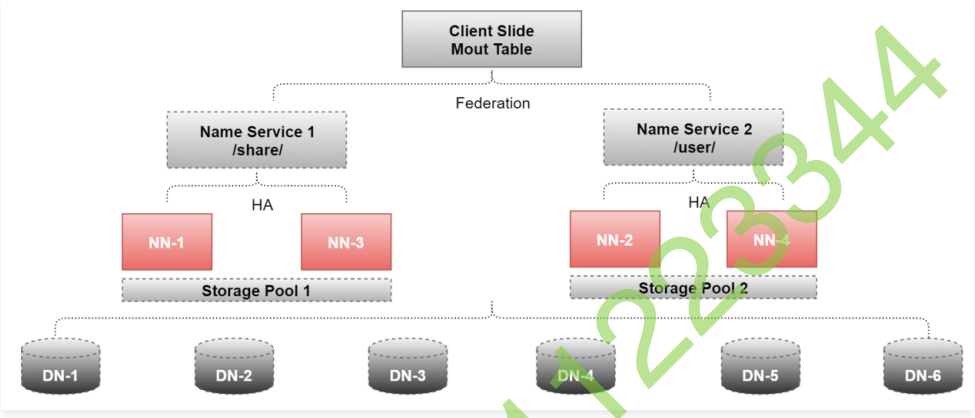
这种设计可提供以下特性：

1：HDFS集群扩展性。多个NameNode分管一部分目录，使得一个集群可以扩展到更多节点，不再因内存的限制制约文件存储数目。

2：性能更高效。多个NameNode管理不同的数据，且同时对外提供服务，将为用户提供更高的读写吞吐率。

3：良好的隔离性。用户可根据需要将不同业务数据交由不同NameNode管理，这样不同业务之间影响很小。

如果真用到了Federation，一般也会和前面我们讲的HA结合起来使用，来看这个图



这里面用到了4个NameNode和6个DataNode

NN-1、NN-2、NN-3、NN-4

DN-1、DN-2、DN-3、DN-4、DN-5、DN-6、

其中NN-1、和NN-3配置了HA，提供了一个命令空间，/share，其实可理解为一个顶级目录

NN-2和NN-4配置了HA，提供了一个命名空间，/user

这样后期我们存储数据的时候，就可以根据数据的业务类型来区分是存储到share目录下还是user目录下，此时HDFS的存储能力就是/share和/user两个命名空间的总和了。

> 注意：由于Federation+HA需要的机器比较多，大家本地的机器开不了那么多虚拟机，所以暂时在这就不再提供对应的安装步骤了，大家主要能理解它的原理就可以了，在工作中也不需要我们去配置。

2 NameNode进阶 上一节 下一节 4 Hadoop核心复盘

我要提出意见反馈

意见反馈 收藏教程 标记书签