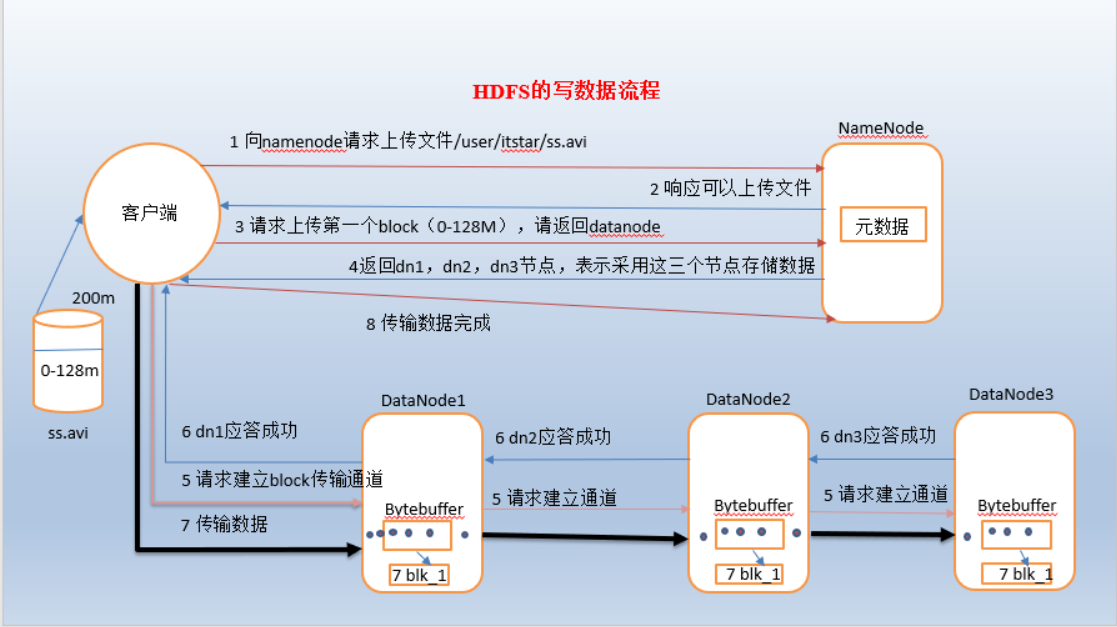
# **四 HDFS的数据流**

## **4.1 HDFS写数据流程**

### **4.1.1 剖析文件写入**



1）客户端向namenode请求上传文件，namenode检查目标文件是否已存在，父目录是否存在。

2）namenode返回是否可以上传。

3）客户端请求第一个 block上传到哪几个datanode服务器上。

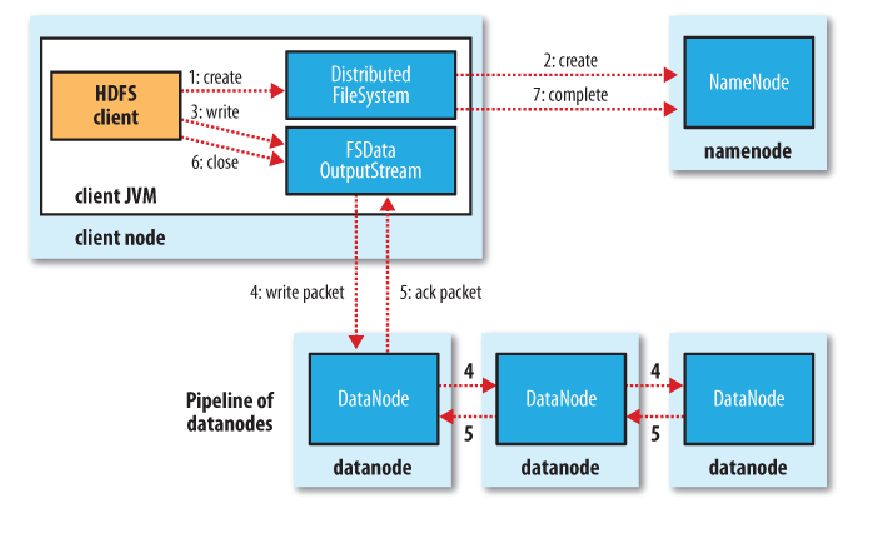
4）namenode返回3个datanode节点，分别为dn1、dn2、dn3。

5）客户端请求dn1上传数据，dn1收到请求会继续调用dn2，然后dn2调用dn3，将这个通信管道建立完成

6）dn1、dn2、dn3逐级应答客户端

7）客户端开始往dn1上传第一个block（先从磁盘读取数据放到一个本地内存缓存），以packet为单位，dn1收到一个packet就会传给dn2，dn2传给dn3；dn1每传一个packet会放入一个应答队列等待应答

8）当一个block传输完成之后，客户端再次请求namenode上传第二个block的服务器。（重复执行3-7步）



1. 客户端通过调用DistributedFileSystem的create方法创建新文件。
2. DistributedFileSystem通过RPC调用namenode去创建一个没有blocks关联的新文件，创建前， namenode会做各种校验，比如文件是否存在，客户端有无权限去创建等。如果校验通过， namenode就会记录下新文件，否则就会抛出IO异常。
3. 前两步结束后，会返回FSDataOutputStream的对象，与读文件的时候相似， FSDataOutputStream被封装成DFSOutputStream。DFSOutputStream可以协调namenode和 datanode。客户端开始写数据到DFSOutputStream，DFSOutputStream会把数据切成一个个小的packet，然后排成队 列data quene（数据队列）。
4. DataStreamer会去处理接受data quene，它先询问namenode这个新的block最适合存储的在哪几个datanode里（比如重复数是3，那么就找到3个最适合的 datanode），把他们排成一个pipeline。DataStreamer把packet按队列输出到管道的第一个datanode中，第一个 datanode又把packet输出到第二个datanode中，以此类推。
5. DFSOutputStream还有一个对列叫ack quene，也是由packet组成，等待datanode的收到响应，当pipeline中的所有datanode都表示已经收到的时候，这时ack quene才会把对应的packet包移除掉。

* 如果在写的过程中某个datanode发生错误，会采取以下几步：
  1. pipeline被关闭掉；
  2. 为了防止防止丢包ack quene里的packet会同步到data quene里；
  3. 把产生错误的datanode上当前在写但未完成的block删掉；
  4. block剩下的部分被写到剩下的两个正常的datanode中；
  5. namenode找到另外的datanode去创建这个块的复制。当然，这些操作对客户端来说是无感知的。

1. 客户端完成写数据后调用close方法关闭写入流。
2. DataStreamer把剩余得包都刷到pipeline里，然后等待ack信息，收到最后一个ack后，通知datanode把文件标视为已完成。

* 注意：客户端执行write操作后，写完的block才是可见的(注:和下面的一致性所对应)，正在写的block对客户端是不可见的，只有 调用sync方法，客户端才确保该文件的写操作已经全部完成，当客户端调用close方法时，会默认调用sync方法。是否需要手动调用取决你根据程序需 要在数据健壮性和吞吐率之间的权衡

### **4.1.2 网络拓扑概念**

在本地网络中，两个节点被称为“彼此近邻”是什么意思？在海量数据处理中，其主要限制因素是节点之间数据的传输速率——带宽很稀缺。这里的想法是将两个节点间的带宽作为距离的衡量标准。

节点距离：两个节点到达最近的共同祖先的距离总和。

例如，假设有数据中心d1机架r1中的节点n1。该节点可以表示为/d1/r1/n1。利用这种标记，这里给出四种距离描述。

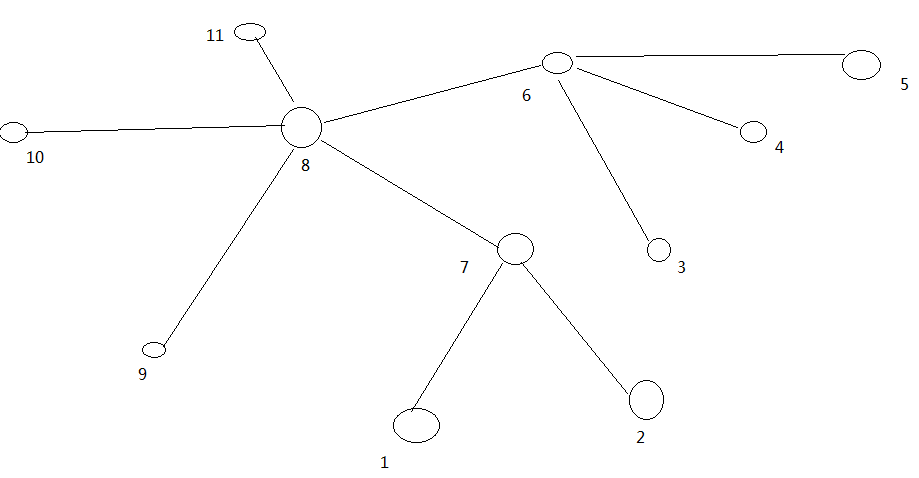
Distance(/d1/r1/n1, /d1/r1/n1)=0（同一节点上的进程）

Distance(/d1/r1/n1, /d1/r1/n2)=2（同一机架上的不同节点）

Distance(/d1/r1/n1, /d1/r3/n2)=4（同一数据中心不同机架上的节点）

Distance(/d1/r1/n1, /d2/r4/n2)=6（不同数据中心的节点）

大家算一算每两个节点之间的距离。



### **4.1.3 机架感知（副本节点选择）**

1）官方ip地址：

<http://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/RackAwareness.html>

http://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html#Data\_Replication

2）低版本Hadoop副本节点选择

第一个副本在client所处的节点上。如果客户端在集群外，随机选一个。

第二个副本和第一个副本位于不相同机架的随机节点上。

第三个副本和第二个副本位于相同机架，节点随机。

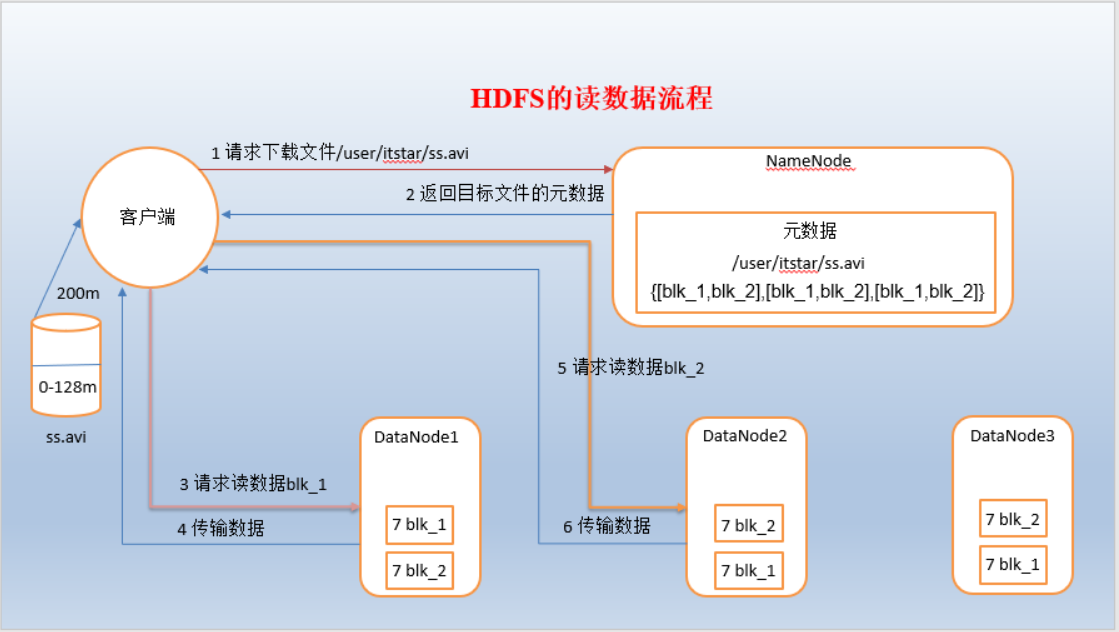
3）高副本节点选择

第一个副本在client所处的节点上。如果客户端在集群外，随机选一个。

第二个副本和第一个副本位于相同机架，随机节点。

第三个副本位于不同机架，随机节点。

## **4.2 HDFS读数据流程**

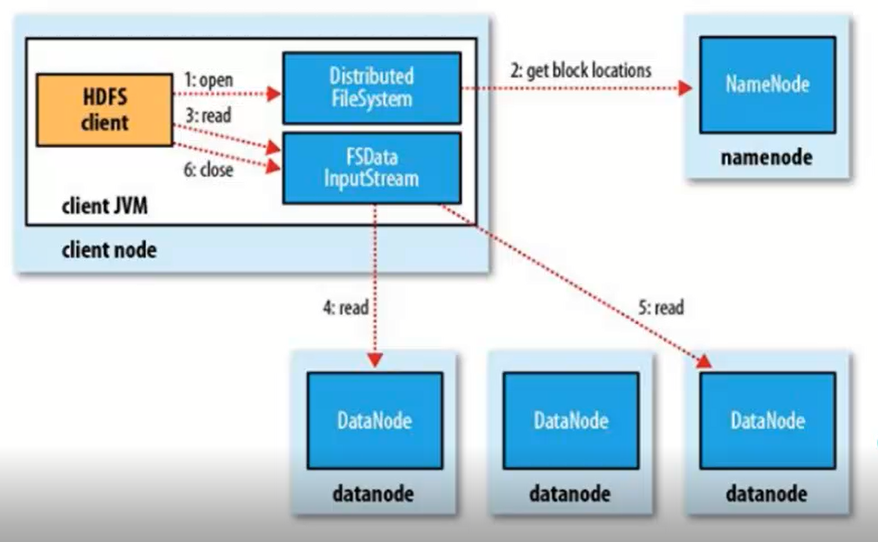


1）客户端向namenode请求下载文件，namenode通过查询元数据，找到文件块所在的datanode地址。

2）挑选一台datanode（就近原则，然后随机）服务器，请求读取数据。

3）datanode开始传输数据给客户端（从磁盘里面读取数据放入流，以packet为单位来做校验）。

4）客户端以packet为单位接收，先在本地缓存，然后写入目标文件。



1. 首先调用FileSystem对象的open方法，其实是一个DistributedFileSystem的实例。
2. DistributedFileSystem通过rpc获得文件的第一批block的locations，同一个block按照重复数会返回多个locations，这些locations按照hadoop拓扑结构排序，距离客户端近的排在前面。
3. 前两步会返回一个FSDataInputStream对象，该对象会被封装DFSInputStream对象，DFSInputStream可 以方便的管理datanode和namenode数据流。客户端调用read方法，DFSInputStream最会找出离客户端最近的datanode 并连接。
4. 数据从datanode源源不断的流向客户端。
5. 如果第一块的数据读完了，就会关闭指向第一块的datanode连接，接着读取下一块。这些操作对客户端来说是透明的，客户端的角度看来只是读一个持续不断的流。
6. 如果第一批block都读完了， DFSInputStream就会去namenode拿下一批block的locations，然后继续读，如果所有的块都读完，这时就会关闭掉所有的流。
7. 如果在读数据的时候， DFSInputStream和datanode的通讯发生异常，就会尝试正在读的block的排序第二近的datanode,并且会记录哪个 datanode发生错误，剩余的blocks读的时候就会直接跳过该datanode。 DFSInputStream也会检查block数据校验和，如果发现一个坏的block,就会先报告到namenode节点，然后 DFSInputStream在其他的datanode上读该block的镜像。
8. 该设计就是客户端直接连接datanode来检索数据并且namenode来负责为每一个block提供最优的datanode， namenode仅仅处理block location的请求，这些信息都加载在namenode的内存中，hdfs通过datanode集群可以承受大量客户端的并发访问。

## **4.3 一致性模型**

1）debug调试如下代码

|  |
| --- |
| @Test  public void writeFile() throws Exception{  // 1 创建配置信息对象  Configuration configuration = new Configuration();  fs = FileSystem.get(configuration);    // 2 创建文件输出流  Path path = new Path("F:\\date\\H.txt");  FSDataOutputStream fos = fs.create(path);    // 3 写数据  fos.write("hello Andy".getBytes());  // 4 一致性刷新  fos.hflush();    fos.close();  } |

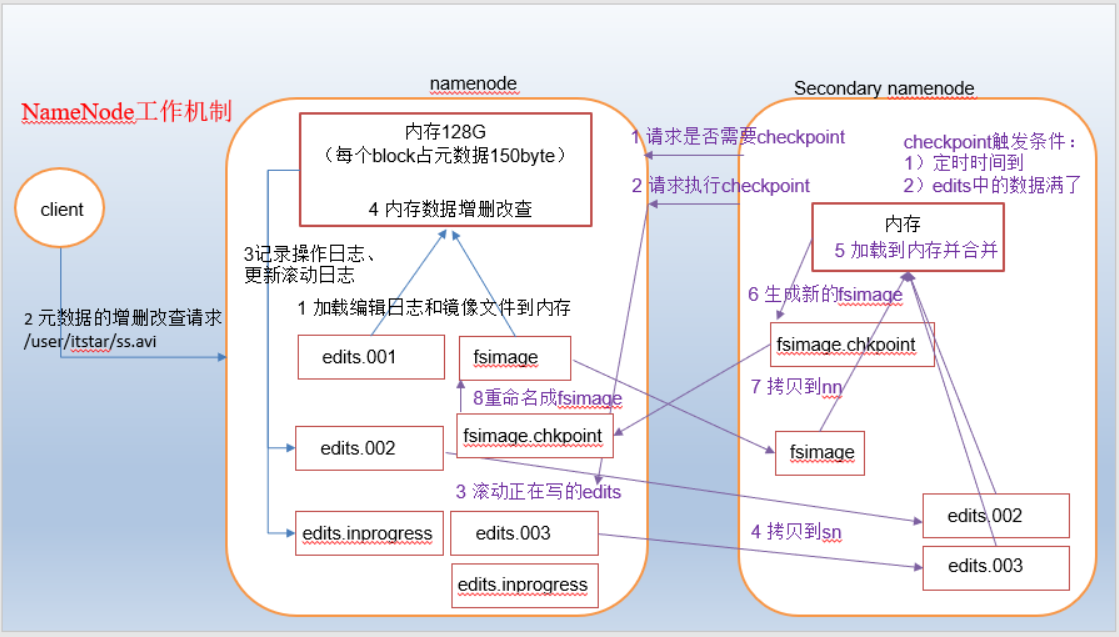
2）总结

写入数据时，如果希望数据被其他client立即可见，调用如下方法

FSDataOutputStream. hflush (); //清理客户端缓冲区数据，被其他client立即可见

# **五 NameNode工作机制**

## **5.1 NameNode&Secondary NameNode工作机制**



1）第一阶段：namenode启动

（1）第一次启动namenode格式化后，创建fsimage和edits文件。如果不是第一次启动，直接加载编辑日志(edits)和镜像文件(fsimage)到内存

（2）客户端对元数据进行增删改的请求

（3）namenode记录操作日志，更新滚动日志

（4）namenode在内存中对数据进行增删改查

2）第二阶段：Secondary NameNode工作

（1）Secondary NameNode询问namenode是否需要checkpoint。直接带回namenode是否检查结果。

（2）Secondary NameNode请求执行checkpoint。

（3）namenode滚动正在写的edits日志

（4）将滚动前的编辑日志和镜像文件拷贝到Secondary NameNode

（5）Secondary NameNode加载编辑日志和镜像文件到内存，并合并。

（6）生成新的镜像文件fsimage.chkpoint

（7）拷贝fsimage.chkpoint到namenode

（8）namenode将fsimage.chkpoint重新命名成fsimage

3）chkpoint检查时间参数设置

（1）通常情况下，SecondaryNameNode每隔一小时执行一次。

[hdfs-default.xml]

|  |
| --- |
| <property>  <name>dfs.namenode.checkpoint.period</name>  <value>3600</value>  </property> |

（2）一分钟检查一次操作次数，当操作次数达到1百万时，SecondaryNameNode执行一次。

|  |
| --- |
| <property>  <name>dfs.namenode.checkpoint.txns</name>  <value>1000000</value>  <description>操作动作次数</description>  </property>  <property>  <name>dfs.namenode.checkpoint.check.period</name>  <value>60</value>  <description> 1分钟检查一次操作次数</description>  </property> |

## **5.2 镜像文件和编辑日志文件**

1）概念

namenode被格式化之后，将在/opt/module/hadoop-2.8.4/data/dfs/name/current目录中产生如下文件,注只能在NameNode所在的节点才能找到此文件

可以执行find . -name edits\* 来查找文件

|  |
| --- |
| edits\_0000000000000000000  fsimage\_0000000000000000000.md5  seen\_txid  VERSION |

（1）Fsimage文件：HDFS文件系统元数据的一个永久性的检查点，其中包含HDFS文件系统的所有目录和文件idnode的序列化信息。

（2）Edits文件：存放HDFS文件系统的所有更新操作的路径，文件系统客户端执行的所有写操作首先会被记录到edits文件中。

（3）seen\_txid文件保存的是一个数字，就是最后一个edits\_的数字

（4）每次Namenode启动的时候都会将fsimage文件读入内存，并从00001开始到seen\_txid中记录的数字依次执行每个edits里面的更新操作，保证内存中的元数据信息是最新的、同步的，可以看成Namenode启动的时候就将fsimage和edits文件进行了合并。

**2）oiv查看fsimage文件**

（1）查看oiv和oev命令

[itstar@bigdata111 current]$ hdfs

oiv apply the offline fsimage viewer to an fsimage

oev apply the offline edits viewer to an edits file

（2）基本语法

hdfs oiv -p 文件类型 -i镜像文件 -o 转换后文件输出路径

（3）案例实操

[itstar@bigdata111 current]$ pwd

/opt/module/hadoop-2.8.4/data/dfs/name/current

[itstar@bigdata111 current]$ hdfs oiv -p XML -i fsimage\_0000000000000000316 -o /opt/fsimage.xml

[itstar@bigdata111 current]$ cat /opt/module/hadop-2.8.4/fsimage.xml

将显示的xml文件内容拷贝到IDEA中创建的xml文件中，并格式化。

**3）oev查看edits文件**

（1）基本语法

hdfs oev -p 文件类型 -i编辑日志 -o 转换后文件输出路径

-p **–processor <arg>**   指定转换类型: binary (二进制格式), xml (默认，XML格式),stats

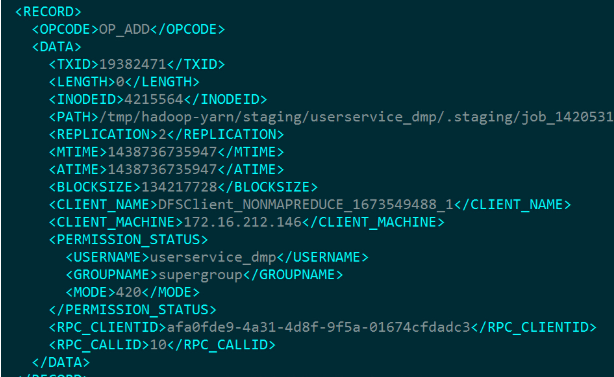
-i **–inputFile <arg>**     输入edits文件，如果是xml后缀，表示XML格式，其他表示二进制

-o **–outputFile <arg>** 输出文件，如果存在，则会覆盖

（2）案例实操

[itstar@bigdata111 current]$ hdfs oev -p XML -i edits\_0000000000000000135-0000000000000000135 -o /opt/module/hadoop-2.8.4/edits.xml -p stats

[itstar@bigdata111 current]$ cat /opt/module/hadoop-2.8.4/edits.xml



每个RECORD记录了一次操作，比如图中的

OP\_ADD代表添加文件操作、OP\_MKDIR代表创建目录操作。里面还记录了

文件路径（PATH）

修改时间（MTIME）

添加时间（ATIME）

客户端名称（CLIENT\_NAME）

客户端地址（CLIENT\_MACHINE）

权限（PERMISSION\_STATUS）等非常有用的信息

**将显示的xml文件内容拷贝到IDEA中创建的xml文件中，并格式化。**

## **5.3 滚动编辑日志**

正常情况HDFS文件系统有更新操作时，就会滚动编辑日志。也可以用命令强制滚动编辑日志。

1）滚动编辑日志（前提必须启动集群）

[itstar@bigdata111 current]$ hdfs dfsadmin -rollEdits

举例：原文件名edits\_inprogress\_0000000000000000321

执行以下命令后

[root@bigdata111 current]# hdfs dfsadmin -rollEdits

Successfully rolled edit logs.

New segment starts at txid 323

edits\_inprogress\_0000000000000000321 => edits\_inprogress\_0000000000000000323

2）镜像文件什么时候产生

Namenode启动时加载镜像文件和编辑日志

## **5.4 namenode版本号**

1）查看namenode版本号

在/opt/module/hadoop-2.8.4/data/dfs/name/current这个目录下查看VERSION

**namespaceID**=1778616660

**clusterID**=CID-bc165781-d10a-46b2-9b6f-3beb1d988fe0

**cTime**=1552918200296

**storageType**=NAME\_NODE

**blockpoolID**=BP-274621862-192.168.1.111-1552918200296

**layoutVersion**=-63

2）namenode版本号具体解释

（1） namespaceID在HDFS上，会有多个Namenode，所以不同Namenode的namespaceID是不同的，分别管理一组blockpoolID。

（2）clusterID集群id，全局唯一

（3）cTime属性标记了namenode存储系统的创建时间，对于刚刚格式化的存储系统，这个属性为0；但是在文件系统升级之后，该值会更新到新的时间戳。

（4）storageType属性说明该存储目录包含的是namenode的数据结构。

（5）blockpoolID：一个block pool id标识一个block pool，并且是跨集群的全局唯一。当一个新的Namespace被创建的时候(format过程的一部分)会创建并持久化一个唯一ID。在创建过程构建全局唯一的BlockPoolID比人为的配置更可靠一些。NN将BlockPoolID持久化到磁盘中，在后续的启动过程中，会再次load并使用。

（6）layoutVersion是一个负整数。通常只有HDFS增加新特性时才会更新这个版本号。

（7）storageID (存储ID)：是DataNode的ID,不唯一

## **5.5 SecondaryNameNode目录结构**

Secondary NameNode用来监控HDFS状态的辅助后台程序，每隔一段时间获取HDFS元数据的快照。

在/opt/module/hadoop-2.8.4/data/dfs/namesecondary/current这个目录中查看SecondaryNameNode目录结构。

|  |
| --- |
| edits\_0000000000000000001-0000000000000000002  fsimage\_0000000000000000002  fsimage\_0000000000000000002.md5  VERSION |

SecondaryNameNode的namesecondary/current目录和主namenode的current目录的布局相同。

好处：在主namenode发生故障时（假设没有及时备份数据），可以从SecondaryNameNode恢复数据。

方法一：将SecondaryNameNode中数据拷贝到namenode存储数据的目录；

方法二：使用-importCheckpoint选项启动namenode守护进程，从而将SecondaryNameNode中数据拷贝到namenode目录中。

1）案例实操（一）：

模拟namenode故障，并采用方法一，恢复namenode数据

（1）kill -9 namenode进程

（2）删除namenode存储的数据（/opt/module/hadoop-2.8.4/data/dfs/name）

rm -rf /opt/module/hadoop-2.8.4/data/dfs/name/\*

注：此时hadoop-daemon.sh stop namenode关闭NN,

然后hadoop-daemon.sh start namenode重启NN,发现50070页面启动不了

（3）拷贝SecondaryNameNode中数据到原namenode存储数据目录

cp -r /opt/module/hadoop-2.8.4/data/dfs/namesecondary/\* /opt/module/hadoop-2.8.4/data/dfs/name/

（4）重新启动namenode

sbin/hadoop-daemon.sh start namenode

2）案例实操（二）：

模拟namenode故障，并采用方法二，恢复namenode数据

（0）修改hdfs-site.xml中的配置，value的单位是秒，默认3600，即1小时，仅配置**一台**即可

|  |
| --- |
| <property>  <name>dfs.namenode.checkpoint.period</name>  <value>120</value>  </property>  <property>  <name>dfs.namenode.name.dir</name>  <value>/opt/module/hadoop-2.8.4/data/dfs/name</value>  </property> |

（1）kill -9 namenode进程

（2）删除namenode存储的数据（/opt/module/hadoop-2.8.4/data/dfs/name）

rm -rf /opt/module/hadoop-2.8.4/data/dfs/name/\*

（3）如果SecondaryNameNode不和Namenode在一个主机节点上，需要将SecondaryNameNode存储数据的目录拷贝到Namenode存储数据的平级目录。

|  |
| --- |
| [itstar@bigdata111 dfs]$ pwd  /opt/module/hadoop-2.8.4/data/dfs  [itstar@bigdata111 dfs]$ ls  data name namesecondary |

（4）导入检查点数据（等待一会ctrl+c结束掉）

bin/hdfs namenode -importCheckpoint

（5）启动namenode

sbin/hadoop-daemon.sh start namenode

（6）如果提示文件锁了，可以删除in\_use.lock

rm -rf /opt/module/hadoop-2.8.4/data/dfs/namesecondary/in\_use.lock

## **5.6 集群安全模式操作**

1）概述

Namenode启动时，首先将映像文件（fsimage）载入内存，并执行编辑日志（edits）中的各项操作。一旦在内存中成功建立文件系统元数据的映像，则创建一个新的fsimage文件和一个空的编辑日志。此时，namenode开始监听datanode请求。但是此刻，namenode运行在安全模式，即namenode的文件系统对于客户端来说是只读的。

系统中的数据块的位置并不是由namenode维护的，而是以块列表的形式存储在datanode中。在系统的正常操作期间，namenode会在内存中保留所有块位置的映射信息。在安全模式下，各个datanode会向namenode发送最新的块列表信息，namenode了解到足够多的块位置信息之后，即可高效运行文件系统。

如果满足“最小副本条件”，namenode会在30秒钟之后就退出安全模式。所谓的最小副本条件指的是在整个文件系统中99.9%的块满足最小副本级别（默认值：dfs.replication.min=1）。在启动一个刚刚格式化的HDFS集群时，因为系统中还没有任何块，所以namenode不会进入安全模式。

2）基本语法

集群处于安全模式，不能执行重要操作（写操作）。集群启动完成后，自动退出安全模式。

（1）bin/hdfs dfsadmin -safemode get （功能描述：查看安全模式状态）

（2）bin/hdfs dfsadmin -safemode enter （功能描述：进入安全模式状态）

（3）bin/hdfs dfsadmin -safemode leave （功能描述：离开安全模式状态）

（4）bin/hdfs dfsadmin -safemode wait （功能描述：等待安全模式状态）

3）案例

模拟等待安全模式

1）先进入安全模式

bin/hdfs dfsadmin -safemode enter

2）执行下面的脚本

编辑一个脚本(注：必须已设置环境变量，要不就写绝对路径)

|  |
| --- |
| #!bin/bash  hdfs dfsadmin -safemode wait  hadoop fs -put /opt/BBB / |

3）再打开一个窗口，执行

bin/hdfs dfsadmin -safemode leave

## **5.7 Namenode多目录配置**

1）namenode的本地目录可以配置成多个，且每个目录存放内容相同，增加了可靠性。

2）具体配置如下：

hdfs-site.xml

|  |
| --- |
| <property>  <name>dfs.namenode.name.dir</name>  <value>file:///${hadoop.tmp.dir}/dfs/name1,file:///${hadoop.tmp.dir}/dfs/name2</value>  </property> |
| 1. 停止集群 删除data 和 logs rm -rf data/\* logs/\* 2. hdfs namenode -format 3. start-dfs.sh 4. 去展示   <https://blog.csdn.net/qq_39657909/article/details/85553525> |
| 实验总结：  **思考1：**如果在非Namenode节点、进行格式化(hdfs namenode -format)  是否和在NN节点上同样会生成name1和name2目录呢？  **答：**只要配置了以上得配置，在该节点下同样会生成name1和name2  **具体解释：**  **格式化做了哪些事情？**  在NameNode节点上，有两个最重要的路径，分别被用来存储元数据信息和操作日志，而这两个路径来自于配置文件，它们对应的属性分别是dfs.name.dir和dfs.name.edits.dir，同时，它们默认的路径均是/tmp/hadoop/dfs/name。格式化时，NameNode会清空两个目录下的所有文件，之后，格式化会在目录dfs.name.dir下创建文件  **hadoop.tmp.dir** 这个配置，会让dfs.name.dir和dfs.name.edits.dir会让两个目录的文件生成在一个目录里  **思考2：**非NN上如果生成了name1和name2，那么他和NN上生成得有没有差别？  **答：**有区别、NN节点上会产生新得edits\_XXX，非NN不会fsimage会更新，而非NN不会，只会产生一个仅初始化得到得fsimage，不会生成edits,更不会发生日志滚动。 |