第1章 HDFS总览

1.1 HDFS产生背景

随着数据量越来越大，单机存不下了，那么就分配到更多的操作系统管理的磁盘中，但是不方便管理和维护，迫切需要一种系统来管理多台机器上的文件，这就是分布式文件管理系统。

1.2 HDFS架构

HDFS组成架构如图所示

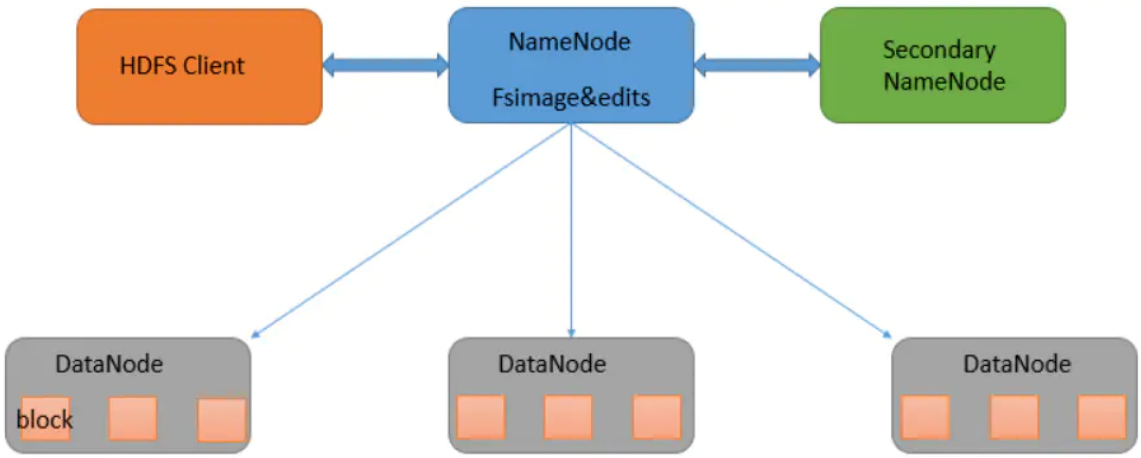


图 HDFS的架构图

HDFS，它是一个文件系统，用于存储文件，通过目录树来定位文件；其次，它是分布式的，由很多服务器联合起来实现其功能，集群中的服务器有各自的角色。

架构主要由四个部分组成，分别为HDFS Client、NameNode、DataNode和Secondary NameNode。下面我们分别介绍这四个组成部分。

1）Client：就是客户端。

2）NameNode：就是Master，它是一个主管、管理者。

3） DataNode：就是Slave。NameNode下达命令，DataNode执行实际的操作

4） SecondaryNameNode：并非NameNode的热备。当NameNode挂掉的时候，它并不能马上替换NameNode并提供服务。

1.3 核心概念block

HDFS中的文件在物理上是分块存储（block），块的大小可以通过配置参数(dfs.blocksize)来规定，默认大小在hadoop2.x版本中是128M，老版本中是64M。

假设我们现在要存储一个300M的文件，这个300M就会被如何切分呢?

datanode1, 128M，

datanode2 ： 128M，

datanode3：44M

为什么block设置在128M左右?

1、寻址

2、数据读写传输。

寻址时间/传输时间 =1%，

10ms，

数据读写时间：1s，磁盘传输的100M/s, 一个块在100M左右是合适的。

问题：为什么无法高效的对大量小文件进行存储。

特点：仅支持数据的append，不支持文件随机修改。

1.4 block副本

为什么hadoop直至今天会这么流行，就是因为它的初始设计就是可以部署在商用小型服务器,pc上，而我们知道，普通服务器是非常廉价的，而这种廉价的服务器就很容易出现故障，比如CPU,IO内存等等都可能会产生问题。

副本数可以自己进行手动设置，一般是3个副本

hdfs-site.xml

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>3</value>
    </property>
</configuration>
```

可以清晰看到value的值为3，我们的副本数就为3

这里我们就可以看到了，dfs.replication的默认值就是为3，后面的参数说明写着default block replication，而下面的参数dfs.replication.max就是副本最大可设置为512的意思了

1.5 机架的存储策略

实际机房中，会有机架，每个机架上会有若干台服务器。

一般来说我们会把一个block的3个副本分别按照下述方法进行存储：

```
第一个副本就存储在一个机架A上
第二个副本存储在和这个block块不同机架（比如机架B）的一个服务器上
```

我 们存储第2个副本时会优先把副本存储在不同的机架上，这是为了防止出现一个机架断电的情况，如果副本也存储在同机架上的不同服务器上，这时候数据就可能丢失了。

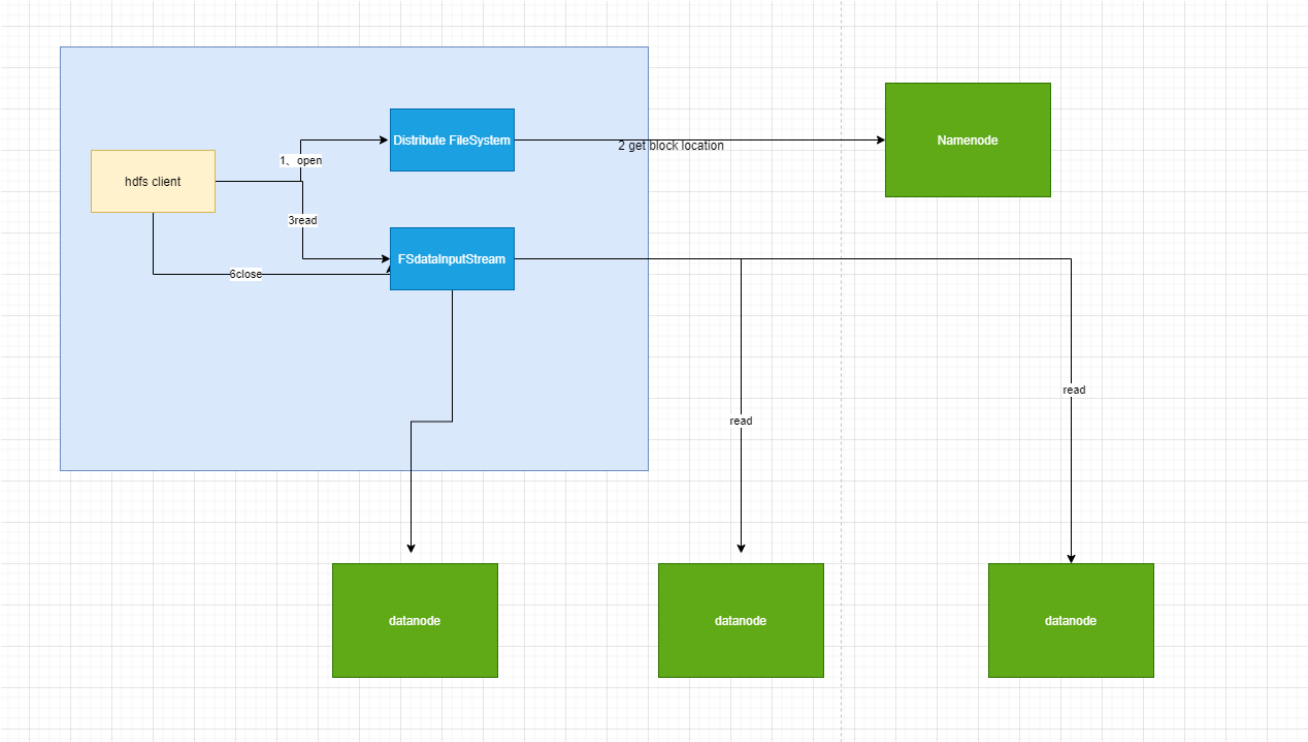
```
第三个副本存储在机架B的另外一个服务器上（注意副本2,3都存储在了机架B）
```

当然我们的副本数也是可以手动通过命令增加的，在客户端访问量多的时候，可以适当分配一下压力

```
$ hadoop fs -setrep -R 4 path
```

第2章 HDFS的数据流

2.1 HDFS的读流程



1.认识角色

简单过一下图里面的角色，最大的块是一个client node

```
hadoop fs -ls /
```

这个命令的机器，那就是客户端了，其他就是NameNode和DataNode，在client node上运行着一个JVM虚拟机，让HDFS client跑起来

2.步骤分析

① HDFS client调用文件系统的open方法

Distributed FileSystem顾名思义是一个分布式文件系统，它会通过RPC的方式远程过程调用NameNode里的open方法，这个open方法有什么作用呢，就是获取要读的文件的**file block locations**，也就是文件的block的位置，之前我们也已经提到了，一个文件是会分割成128M一块的大小分别存储在各个数据节点的。

同时在执行open方法时，客户端会产生一个FSData InputStream的一个输入流对象（客户端读数据是从外部读回来的）

② FSData InputStream读数据

HDFS client调用FSData InputStream的read方法，同上也是远程过程**调用DataNode的read方法**，此时的读取顺序是由近到远，就是DataNode和client node的距离，这里所指的距离是一种物理距离。

在联系上DataNode并成功读取后，关闭流就走完了一个正常的流程。

上面Distributed FileSystem所调用的get block locations的方法只会返回部分数据块，get block locations会分批次地返回block块的位置信息。读block块理论上来说是依次读，当然也可以通过多线程的方式实现同步读。

③ 容错机制

1.如果client从DataNode上读取block时网络中断了如何解决?

2.如果一个DataNode挂掉了怎么办?

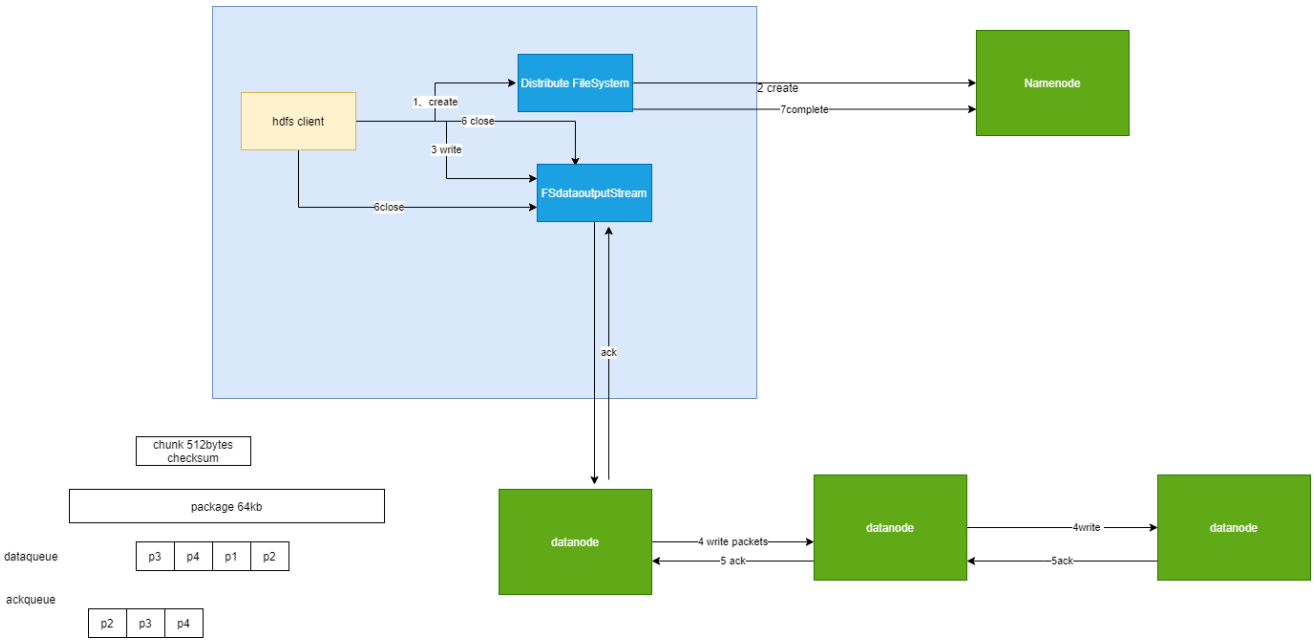
3.client如何保证读取数据的完整性

checksum (crc32),

4.上一个问题完成后工作

FSData InputStream会告诉NameNode，这个DataNode上的这个block有问题了，NameNode收到消息后就会再通过心跳机制通知这个DataNode删除它的block，让正常的DataNode去copy一份正常的block数据给其它节点，保证副本数为3

2.2 HDFS的写流程



客户端向HDFS写数据的时候是把文件分块存储在HDFS的各个节点上，而规定了存储位置的是NameNode，所以Client node在存储文件时需要先和NameNode进行联系让它进行分配。

步骤分析

① 客户端调用分布式文件系统的create方法

和上面读的方法类似，不过这次调用的是Distributed FileSystem的create方法，此时也是通过远程调用NameNode的create方法

此时NameNode会进行的举措

- 1.**检测自己是否正常运行**
- 2.**判断要创建的文件是否存在**
- 3.**client是否有创建文件的权限**
- 4.**对HDFS做状态的更改需要在edits log写日志记录**

② 客户端调用输出流的write方法

create方法的返回值是一个OutputStream对象。

但是有个问题，此时我们还不知道我们的这些block块要分别存放于哪些节点上，所以此时FSData OutputStream就要再和NameNode交互一下，远程过程调用**NameNode的addBlock**方法，这个方法**返回的是各个block块分别需要写在哪3个DataNode**上面。

此时OutputStream就完整得知了数据和数据该往哪里去写了

③ 具体的写流程分析

这时候开始真正的写数据过程

data queue中的package往数据节点DataNode上传输，传输的顺序按照NameNode的addBlock()方法返回的列表依次传输

1. **往DataNode上传输的同时也往确认队列ack queue上传输**
2. **针对DataNode中传输完成的数据做一个checkSum，并与原本打包前的checkSum做一个比较**
3. **校验成功，就从确认队列ack queue中删除该package，否则该package重新置入data queue重传**

补充：

1.以上逻辑归属于FSData OutputStream的逻辑

2.虽然本身一个block为128M，而package为64Kb，128M对于网络传输过程来说算是比较大，拆分为小包是为了可靠传输。

3.如果是某个DataNode不可用，在1中我们也提到过了，通过心跳机制会通知其余的可用DataNode的其中一个进行copy到一个可用节点上

流程总结

1.client端调用Distributed FileSystem的create，此时是远程调用了NameNode的create，此时NameNode进行4个操作，

2.create的返回值为一个FSData OutputStream对象，此时client调用流的write方法，和NameNode进行连接，NameNode的addBlock方法返回块分配的DataNode列表

3.开始写数据，先写在chunk，后package，置入data queue，此时两个操作，向DataNode传输，和放入ack queue，DataNode传输结束会检测checkSum，成功就删除ack queue的package，否则放回data queue重传

4.结束后关闭流，告诉NameNode，调用complete方法结束

第3章 NameNode详解

3.1 metaData的介绍

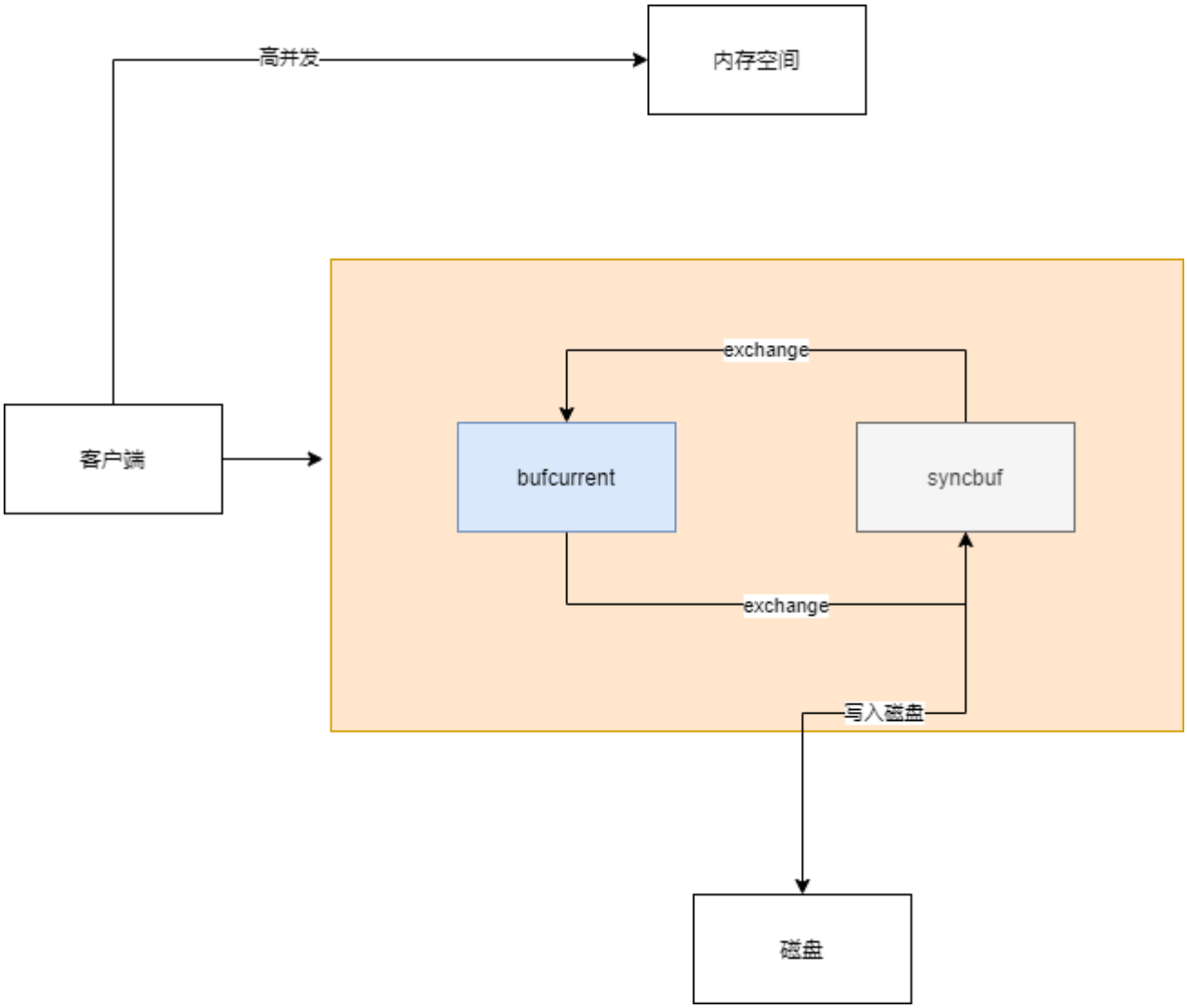
metaData的大小：文件，block，目录占用大概150byte字节的元数据。

元数据信息以命名空间镜像文件（以下称为fsimage）和编辑日志（以下称为edits log）的方式保存，两者的作用分别是

```
fsimage：元数据镜像文件，保存了文件系统目录树信息以及文件和块的对应关系
edits log：日志文件，保存了文件的更改记录
```

日志的双缓冲机制

双缓冲机制就是指我们将会开辟两份一模一样的内存空间，一个为bufCurrent，产生的数据会直接写入到这个bufCurrent，而另一个叫syncBuffer，在bufCurrent数据写入（其实这里可能不止一条数据，等下会说明）后，两片内存就会exchange（交换）。然后之前的bufCurrent就负责往磁盘上写入数据，之前的syncBuffer就继续接收客户端写入的数据。其实就是将向磁盘写数据的任务交给了后台去做。



逻辑：

① 定义了两个缓冲区currentBuffer（有序队列）和syncBuffer

② 一个write方法负责写入元数据

那我现在要保证这个写操作（**这里的写操作是客户端写**）的顺序，所以我们在这里会使用synchronized来加锁，然后通过taxid++顺序处理。

③ 一个flush方法把元数据写入到磁盘上，写入完成后清空syncBuffer的数据（包含exchange）

④ 一个exchange方法来交换currentBuffer和syncBuffer

⑤ 还有一个getMaxTaxid方法获取到正在同步数据的内存里面事务ID的最大ID。

NN和2NN工作机制

图 NN和2NN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，直接加载编辑日志和镜像文件到内存。

（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。

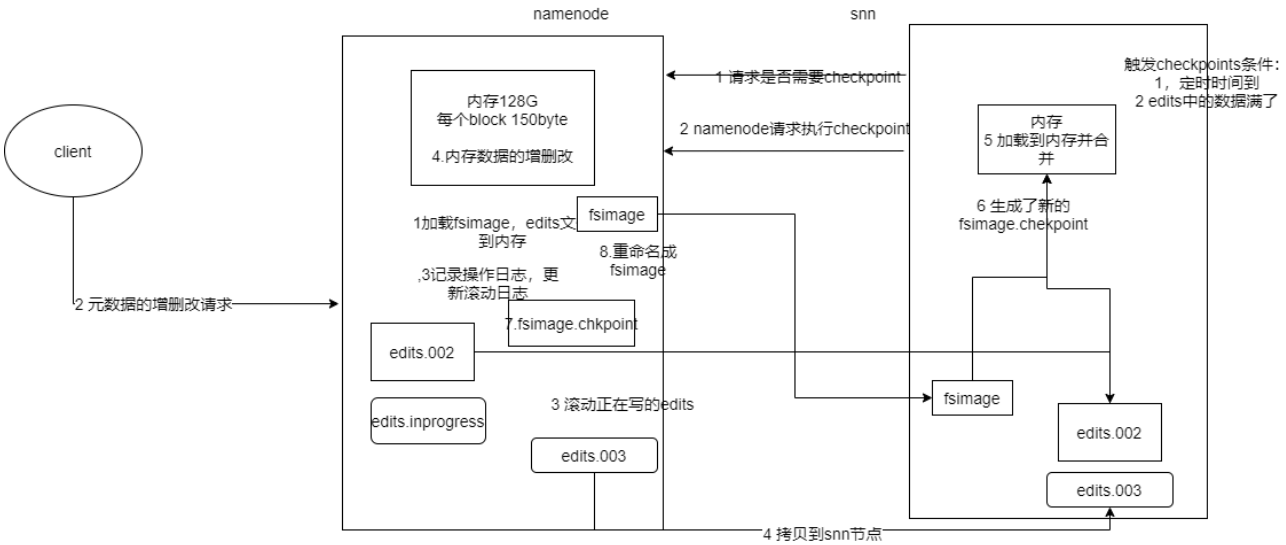
（4）将滚动前的编辑日志和镜像文件拷贝到Secondary NameNode。

（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。



3.2 secondary namenode的作用

1.备份NameNode中的元数据信息 2.提高NameNode的重启速度 3.必要的时候可作为新的NameNode

为什么说SecondaryNameNode可以提高NameNode的恢复速度？！

当集群启动的时候，会记录下启动的时间，而随着一段时间过去后或者NameNode中的edits log文件存满后就会触发checkPoint操作，在Spark中也会有这个知识点，主要作用就是对重要的数据进行备份的一个操作。

secondaryNameNode中合并完成之后，再回传给NameNode里面

为什么说SecondaryNameNode可以提高NameNode的重启速度

首先搞清楚NameNode节点挂掉后它是如何进行恢复的

它会把内存中的镜像文件fsimage读到内存当中，然后通过edits log所记录的所有操作重新执行一遍，把所有的元数据都恢复之后，才能回到关机之前的状态，这个过程十分缓慢

但是有了SecondaryNameNode之后，通过它提供的fsimage.ckpt可以恢复很大一部分的元数据信息，再直接通过执行edits log中所记录下来的，从edits new中合并过来的新操作，就可以进行恢复。

必要的时候可作为新的NameNode

而在NameNode确定无法重启之后，SecondaryNameNode就可以通过以下命令作为新的NameNode对外提供服务

```
hadoop-daemon.sh start namenode
```

当然我们不难发现，这种方式非常地不优雅，因为在NameNode进行重启或者SecondaryNameNode进行上位的时间段中我们的集群肯定都会有一段空白期，所以之后讲到的hadoop HA的方式就会帮助我们解决这个问题

3.3 Fsimage和Edits解析

1. **概念**

   namenode被格式化之后，将在/home/teacher/opt/module/hadoop-2.7.2/data/tmp/dfs/name/current目录中产生如下文件

fsimage_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

[root@teacher1 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

[root@teacher1 current]$ pwd

/home/teacher/opt/module/hadoop-2.7.2/data/tmp/dfs/name/current

[root@teacher1 current]$ hdfs oiv -p XML -i fsimage_0000000000000003655 -o ~/tmp/fsimage.xml

[root@teacher1 current]$ cat /opt/module/hadoop-2.7.2/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。部分显示结果如下。Xml参数必须大写

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

（2）案例实操

[root@teacher1 current]$ hdfs oev -p XML -i edits_0000000000000000012-0000000000000000013 -o ~/tmp/edits.xml

[root@teacher1 current]$ cat /opt/module/hadoop-2.7.2/edits.xml

3.4 checkpoint时间设置

（1）通常情况下，SecondaryNameNode每隔一小时执行一次。如果修改

在hdfs-site中

默认值在[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 >
```

3.5 Hadoop HA高可用

NameNode 的高可用架构主要分为下面几个部分：

Active NameNode 和 Standby NameNode：两台 NameNode 形成互备，一台处于 Active 状态，为主 NameNode，另外一台处于 Standby 状态，为备 NameNode，只有主 NameNode 才能对外提供读写服务。

主备切换控制器 ZKFailoverController：ZKFailoverController 作为独立的进程运行，对 NameNode 的主备切换进行总体控制。ZKFailoverController 能及时检测到 NameNode 的健康状况，在主 NameNode 故障时借助 Zookeeper 实现自动的主备选举和切换，当然 NameNode 目前也支持不依赖于 Zookeeper 的手动主备切换。

Zookeeper 集群：为主备切换控制器提供主备选举支持。

共享存储系统：共享存储系统是实现 NameNode 的高可用最为关键的部分，共享存储系统保存了 NameNode 在运行过程中所产生的 HDFS 的元数据。主 NameNode 和NameNode 通过共享存储系统实现元数据同步。在进行主备切换的时候，新的主 NameNode 在确认元数据完全同步之后才能继续对外提供服务。

DataNode 节点：除了通过共享存储系统共享 HDFS 的元数据信息之外，主 NameNode 和备 NameNode 还需要共享 HDFS 的数据块和 DataNode 之间的映射关系。DataNode 会同时向主 NameNode 和备 NameNode 上报数据块的位置信息。

NameNode 的主备切换实现

NameNode 主备切换主要由 ZKFailoverController、HealthMonitor 和 ActiveStandbyElector 这 3 个组件来协同实现。

1. HealthMonitor 初始化完成之后会启动内部的线程来定时调用对应 NameNode 的 HAServiceProtocol RPC 接口的方法，对 NameNode 的健康状态进行检测。
2. HealthMonitor 如果检测到 NameNode 的健康状态发生变化，会回调 ZKFailoverController 注册的相应方法进行处理。
3. 如果 ZKFailoverController 判断需要进行主备切换，会首先使用 ActiveStandbyElector 来进行自动的主备选举。
4. ActiveStandbyElector 与 Zookeeper 进行交互完成自动的主备选举。
5. ActiveStandbyElector 在主备选举完成后，会回调 ZKFailoverController 的相应方法来通知当前的 NameNode 成为主 NameNode 或备 NameNode。
6. ZKFailoverController 调用对应 NameNode 的 HAServiceProtocol RPC 接口的方法将 NameNode 转换为 Active 状态或 Standby 状态。

3.6 集群安全模式

hadoop为了防止数据丢失，启动了“安全模式”的设置，每次启动hadoop后一段时间内集群处于安全模式

1、概述

namenode启动

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。

datanode启动

系统中的数据块的位置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，

退出安全模式条件

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，监控安全模式）

3. 案例

模拟等待安全模式

（1）先进入安全模式

[root@teacher1 hadoop-2.7.2]$ bin/hdfs dfsadmin -safemode enter

（2）执行下面的脚本

编辑一个脚本

#!/bin/bash

bin/hdfs dfsadmin -safemode wait（安全模式关闭）

bin/hdfs dfs -put ~/hello.txt /root/hello.txt

（3）再打开一个窗口，执行

[root@teacher1 hadoop-2.7.2]$ bin/hdfs dfsadmin -safemode leave

第4章 DataNode详解

4.1 心跳机制

心跳机制的流程

1.master namenode启动之后，会开一个ipc server 2.slave DataNode启动，连接NameNode，每隔3s向NameNode发送一个心跳，并携带状态信息 3.NameNode通过对这个心跳的返回值来给DataNode传达任务指令

心跳机制的作用

1.NameNode全权管理数据块的复制，它周期性从集群中的每个DataNode接收心跳信号和块状态报告（blockReport），接收到心跳信号意味着该DataNode节点工作正常，块状态报告包含了该DataNode上所有数据块的列表

2.DataNode启动时向NameNode注册，通过后周期性地向NameNode上报blockReport，每3秒向NameNode发送一次心跳，NameNode返回对该DataNode的指令，如将数据块复制到另一台机器，或删除某个数据块等…而当某一个DataNode超过10min还没向NameNode发送心跳，此时NameNode就会判定该DataNode不可用，此时客户端的读写操作就不会再传达到该DataNode上

3.hadoop集群刚开始启动时会进入安全模式（99.99%），就用到了心跳机制，其实就是在集群刚启动的时候，每一个DataNode都会向NameNode发送blockReport，NameNode会统计它们上报的总block数，除以一开始知道的总个数total，当 block/total < 99.99% 时，会触发安全模式，安全模式下客户端就没法向HDFS写数据，只能进行读数据。

4.2 负载均衡

A datanode级别的reblance。

B Disk级别的reblance

节点的均衡配置

其实就是节点的增加或减少，或者节点的磁盘使用率高低的问题，主要就是通过网络进行数据的迁移工作以达到高可用率

触发命令

```
$ HADOOP_HOME/sbin/start-balancer.sh -t 5%
```

5%其实就是刚刚提到的磁盘的利用率差值，大于5%时会触发负载均衡策略

hadoop2.x的磁盘均衡配置

hdfs-site.xml中设置 每块磁盘的预留空间

```
<property>
<name>dfs.datanode.du.reserved</name>
<value>82212254720</value>
</property>
```

让hdfs 优先往新盘 写入数据 ， 改变 hdfs 的默认写入策略，监控 各个磁盘写入差额 启动自平衡。

两种写入策略：

循环（round-robin）策略将新块均匀分布在可用磁盘上；而可用空间（ available-space ）策略优先将数据写入具有最大可用空间的磁盘（通过百分比计算的）。

```
<property>
<name>dfs.datanode.fsdataset.volume.choosing.policy</name>
<value>org.apache.hadoop.hdfs.server.datanode.fsdataset.AvailableSpaceVolumeChoosingPolicy</valu
e>
</property>
```

第5章 HDFS的Shell客户端操作

5.1 基本语法

bin/hadoop fs 具体命令

5.2 命令行演示

[root@teacher1 hadoop-2.7.2]$ bin/hadoop fs

3．常用命令实操

（0）启动Hadoop集群

[root@teacher1 hadoop-2.7.2]$ sbin/start-dfs.sh

[root@hadoop102 hadoop-2.7.2]$ sbin/start-yarn.sh

（1）-help：输出这个命令参数

[root@teacher1 hadoop-2.7.2]$ hadoop fs -help rm

（2）-ls: 显示目录信息

[root@teacher1 hadoop-2.7.2]$ hadoop fs -ls /

（3）-mkdir：在hdfs上创建目录

[root@teacher1 hadoop-2.7.2]$ hadoop fs -mkdir -p /lagou/

（4）-moveFromLocal从本地**剪切**粘贴到hdfs

[root@teacher1 hadoop-2.7.2]$ touch ai.txt

[root@teacher1 hadoop-2.7.2]$ hadoop fs -moveFromLocal ./ai.txt /lagou/test1

（5）-appendToFile ： 追加一个文件到已经存在的文件末尾

[root@teacher1 hadoop-2.7.2]$ touch bigdata.txt

[root@teacher1 hadoop-2.7.2]$ vim bigdata.txt

输入

abcdefg

[root@hadoop102 hadoop-2.7.2]$ hadoop fs -appendToFile bigdata.txt /lagou/test1/ai.txt

（6）-cat：显示文件内容

[root@teacher1 hadoop-2.7.2]$ hadoop fs -cat /lagou/test1/ai.txt

（7）-tail：显示一个文件的末尾

[root@teacher1 hadoop-2.7.2]$ hadoop fs -tail /lagou/test1/ai.txt

（8）-chgrp 、-chmod、-chown：linux文件系统中的用法一样，修改文件所属权限

[root@teacher1 hadoop-2.7.2]$ hadoop fs -chmod 666 /lagou/test1/ai.txt

[root@teacher1 hadoop-2.7.2]$ hadoop fs -chown root:root /lagou/test1/ai.txt

（9）-copyFromLocal：从本地文件系统中拷贝文件到hdfs路径去

[root@teacher1 hadoop-2.7.2]$ hadoop fs -copyFromLocal README.txt /

（10）-copyToLocal：从hdfs拷贝到本地

[root@teacher1 hadoop-2.7.2]$ hadoop fs -copyToLocal /lagou/test1/ai.txt ./

（11）-cp ： 从hdfs的一个路径拷贝到hdfs的另一个路径

[root@teacher1 hadoop-2.7.2]$ hadoop fs -cp /lagou/test1/ai.txt /java.txt

（12）-mv：在hdfs目录中移动文件

[root@teacher1 hadoop-2.7.2]$ hadoop fs -mv /java.txt /lagou/test1/

（13）-get：等同于copyToLocal，就是从hdfs下载文件到本地

[root@teacher1 hadoop-2.7.2]$ hadoop fs -get /lagou/test1/ai.txt ./

（14）-getmerge ： 合并下载多个文件，比如hdfs的目录 /aaa/下有多个文件:log1.txt,log2.txt

[root@teacher1 hadoop-2.7.2]$ hadoop fs -getmerge /lagou/test1/* ./merge.txt

（15）-put：**等同于copyFromLocal**

[root@teacher1 hadoop-2.7.2]$ hadoop fs -put ./zaiyiqi.txt /lagou/test1/

（16）-rm：删除文件或文件夹

[root@teacher1 hadoop-2.7.2]$ hadoop fs -rm /user/root/test/jinlian2.txt

（17）-rmdir：删除目录

[root@teacher1 hadoop-2.7.2]$ hadoop fs -mkdir /test

[root@teacher1 hadoop-2.7.2]$ hadoop fs -rmdir /test

（18）-du统计文件夹的大小信息

[root@teacher1 hadoop-2.7.2]$ hadoop fs -du -s -h /user/root/test

-s：表示汇总值（其实就是查询这个目录总的文件大小，反之查的是表每个文件的大小）

2K /user/root/test

（19）-setrep：设置hdfs中文件的副本数量

[teacher@teacher1 hadoop-2.7.2]$ hadoop fs -setrep 2 /lagou/test1/ai.txt

第6章 HDFS的Java客户端操作

6.1 HDFS客户端环境准备

生产环境下hdfs的客户端一般不会使用hadoop集群的节点，选择独立的一台机器作为client，这样避免影响集群。测试和学习环境可以选择hadoop集群的任何一个节点作为客户端，比如namenode或datanode节点所在的机器。

6.2 HDFS的API操作演示

3.2.1HDFS文件上传

1．编写源代码

2．将hdfs-site.xml拷贝到项目的根目录下

```
    <dependencies>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>

        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.10.0</version>
        </dependency>

        <!--hdfs客户端操作需要的jar-->
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-common</artifactId>
            <version>2.7.2</version>
        </dependency>

        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-client</artifactId>
            <version>2.7.2</version>
```

```
        </dependency>

        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-hdfs</artifactId>
            <version>2.7.2</version>
        </dependency>
    </dependencies>

    <!--指定java编译版本，固定配置即可-->
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <encoding>utf-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

HDFS文件下载

```
    @Test

    public void testCopyToLocalFile() throws IOException, InterruptedException,
URISyntaxException{
        // 1 获取文件系统
        Configuration configuration = new Configuration();
        FileSystem fs = FileSystem.get(new URI("hdfs://teacher1:9000"), configuration,
"teacher");


        // 2 执行下载操作
        fs.copyToLocalFile(false, new Path("/teacher1.txt"), new Path("c:/teacher1.txt"), true);

        // 3 关闭资源

        fs.close();

}
```

hdfs文件夹创建

```
   @Test
    public void testMkdir() throws IOException {
        fileSystem.mkdirs(new Path("/lagoujiaoyu"));
    }
```

hdfs文件夹重命名

```
@Test
public void testRenameFile() throws IOException {
    fileSystem.rename(new Path("/lagoujiaoyu"),new Path("/bigdata"));
}
```

HDFS文件夹删除

```
    @Test
    public void testDelete() throws IOException, InterruptedException, URISyntaxException{
        // 1 获取文件系统
        Configuration configuration = new Configuration();
        FileSystem fs = FileSystem.get(new URI("hdfs://teacher1:9000"), configuration,
"teacher");
        // 2 执行删除
        fs.delete(new Path("/lagoujiaoyu/"), true);
        // 3 关闭资源
        fs.close();

}
```