目录

[课程大纲（HDFS详解） 2](#_Toc439077207)

[1. HDFS前言 3](#_Toc439077208)

[2. HDFS的概念和特性 3](#_Toc439077209)

[3. HDFS的shell(命令行客户端)操作 4](#_Toc439077210)

[3.1 HDFS命令行客户端使用 4](#_Toc439077211)

[3.2命令行客户端支持的命令参数 4](#_Toc439077212)

[3.2 常用命令参数介绍 5](#_Toc439077213)

[4. hdfs的工作机制 8](#_Toc439077214)

[4.1 概述： 8](#_Toc439077215)

[4.2 HDFS写数据流程 9](#_Toc439077216)

[4.2.1 概述 9](#_Toc439077217)

[4.2.2 详细步骤图 9](#_Toc439077218)

[4.2.3 详细步骤解析 9](#_Toc439077219)

[4.3. HDFS读数据流程 10](#_Toc439077220)

[4.3.1 概述 10](#_Toc439077221)

[4.3.2 详细步骤图： 10](#_Toc439077222)

[4.3.3 详细步骤解析 10](#_Toc439077223)

[5. NAMENODE工作机制 11](#_Toc439077224)

[5.1 概述 11](#_Toc439077225)

[5.2元数据管理 11](#_Toc439077226)

[5.2.1 元数据存储机制 11](#_Toc439077227)

[5.2.2 元数据手动查看 11](#_Toc439077228)

[5.2.3 元数据的checkpoint 12](#_Toc439077229)

[6. DATANODE的工作机制 13](#_Toc439077230)

[6.1 概述 13](#_Toc439077231)

[6.2 观察验证DATANODE功能 13](#_Toc439077232)

[7. HDFS的java操作 13](#_Toc439077233)

[7.1 搭建开发环境 13](#_Toc439077234)

[7.2 获取api中的客户端对象 14](#_Toc439077235)

[7.3 DistributedFileSystem实例对象所具备的方法 14](#_Toc439077236)

[7.4 HDFS客户端操作数据代码示例： 15](#_Toc439077237)

[7.4.1 文件的增删改查 15](#_Toc439077238)

[7.4.2 通过流的方式访问hdfs 18](#_Toc439077239)

# 课程大纲（HDFS详解）

|  |  |
| --- | --- |
| Hadoop HDFS | 分布式文件系统DFS简介 |
| HDFS的系统组成介绍 |
| HDFS的组成部分详解 |
| 副本存放策略及路由规则 |
| 命令行接口 |
| Java接口 |
| 客户端与HDFS的数据流讲解 |

学习目标：

掌握hdfs的shell操作

掌握hdfs的java api操作

理解hdfs的工作原理

# \*\*\*\*\*\*HDFS基本概念篇\*\*\*\*\*\*

# 1. HDFS前言

* 设计思想

分而治之：将大文件、大批量文件，分布式存放在大量服务器上，**以便于采取分而治之的方式对海量数据进行运算分析；**

* 在大数据系统中作用：

为各类分布式运算框架（如：mapreduce，spark，tez，……）提供数据存储服务

* 重点概念：文件切块，副本存放，元数据

# 2. HDFS的概念和特性

**首先，它是一个文件系统**，用于存储文件，通过统一的命名空间——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；

**重要特性如下：**

1. 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
2. HDFS文件系统会给客户端提供一个**统一的抽象目录树**，客户端通过路径来访问文件，形如：hdfs://namenode:port/dir-a/dir-b/dir-c/file.data
3. **目录结构及文件分块信息(元数据)**的管理由namenode节点承担

——namenode是HDFS集群主节点，负责维护整个hdfs文件系统的目录树，以及每一个路径（文件）所对应的block块信息（block的id，及所在的datanode服务器）

1. 文件的各个block的存储管理由datanode节点承担

---- datanode是HDFS集群从节点，每一个block都可以在多个datanode上存储多个副本（副本数量也可以通过参数设置dfs.replication）

1. HDFS是设计成适应一次写入，多次读出的场景，且不支持文件的修改

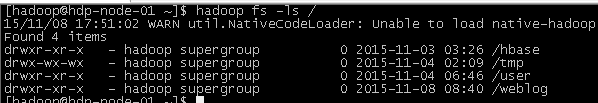
*(注：适合用来做数据分析，并不适合用来做网盘应用，因为，不便修改，延迟大，网络开销大，成本太高)*

# \*\*\*\*\*\*HDFS基本操作篇\*\*\*\*\*\*

# 3. HDFS的shell(命令行客户端)操作

## 3.1 HDFS命令行客户端使用

HDFS提供shell命令行客户端，使用方法如下：



## 3.2 命令行客户端支持的命令参数

|  |
| --- |
| [-appendToFile <localsrc> ... <dst>]  [-cat [-ignoreCrc] <src> ...]  [-checksum <src> ...]  [-chgrp [-R] GROUP PATH...]  [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]  [-chown [-R] [OWNER][:[GROUP]] PATH...]  [-copyFromLocal [-f] [-p] <localsrc> ... <dst>]  [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]  [-count [-q] <path> ...]  [-cp [-f] [-p] <src> ... <dst>]  [-createSnapshot <snapshotDir> [<snapshotName>]]  [-deleteSnapshot <snapshotDir> <snapshotName>]  [-df [-h] [<path> ...]]  [-du [-s] [-h] <path> ...]  [-expunge]  [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]  [-getfacl [-R] <path>]  [-getmerge [-nl] <src> <localdst>]  [-help [cmd ...]]  [-ls [-d] [-h] [-R] [<path> ...]]  [-mkdir [-p] <path> ...]  [-moveFromLocal <localsrc> ... <dst>]  [-moveToLocal <src> <localdst>]  [-mv <src> ... <dst>]  [-put [-f] [-p] <localsrc> ... <dst>]  [-renameSnapshot <snapshotDir> <oldName> <newName>]  [-rm [-f] [-r|-R] [-skipTrash] <src> ...]  [-rmdir [--ignore-fail-on-non-empty] <dir> ...]  [-setfacl [-R] [{-b|-k} {-m|-x <acl\_spec>} <path>]|[--set <acl\_spec> <path>]]  [-setrep [-R] [-w] <rep> <path> ...]  [-stat [format] <path> ...]  [-tail [-f] <file>]  [-test -[defsz] <path>]  [-text [-ignoreCrc] <src> ...]  [-touchz <path> ...]  [-usage [cmd ...]] |

## 3.2 常用命令参数介绍

|  |
| --- |
| -help  功能：输出这个命令参数手册 |
| **-ls**  **功能：显示目录信息**  *示例： hadoop fs -ls hdfs://hadoop-server01:9000/*  *备注：这些参数中，所有的hdfs路径都可以简写*  *-->hadoop fs -ls / 等同于上一条命令的效果* |
| **-mkdir**  **功能：在hdfs上创建目录**  *示例：hadoop fs -mkdir -p /aaa/bbb/cc/dd* |
| **-moveFromLocal**  **功能：从本地剪切粘贴到hdfs**  *示例：hadoop fs - moveFromLocal /home/hadoop/a.txt /aaa/bbb/cc/dd*  **-moveToLocal**  **功能：从hdfs剪切粘贴到本地**  *示例：hadoop fs - moveToLocal /aaa/bbb/cc/dd /home/hadoop/a.txt* |
| **--appendToFile**  **功能：追加一个文件到已经存在的文件末尾**  *示例：hadoop fs -appendToFile ./hello.txt hdfs://hadoop-server01:9000/hello.txt*  *可以简写为：*  *Hadoop fs -appendToFile ./hello.txt /hello.txt* |
| **-cat**  **功能：显示文件内容**  *示例：hadoop fs -cat /hello.txt*  **-tail**  **功能：显示一个文件的末尾**  *示例：hadoop fs -tail /weblog/access\_log.1*  **-text**  **功能：以字符形式打印一个文件的内容**  *示例：hadoop fs -text /weblog/access\_log.1* |
| **-chgrp**  **-chmod**  **-chown**  **功能：linux文件系统中的用法一样，对文件所属权限**  *示例：*  *hadoop fs -chmod 666 /hello.txt*  *hadoop fs -chown someuser:somegrp /hello.txt* |
| **-copyFromLocal**  **功能：从本地文件系统中拷贝文件到hdfs路径去**  *示例：hadoop fs -copyFromLocal ./jdk.tar.gz /aaa/*  **-copyToLocal**  **功能：从hdfs拷贝到本地**  *示例：hadoop fs -copyToLocal /aaa/jdk.tar.gz* |
| **-cp**  **功能：从hdfs的一个路径拷贝hdfs的另一个路径**  *示例： hadoop fs -cp /aaa/jdk.tar.gz /bbb/jdk.tar.gz.2*  **-mv**  **功能：在hdfs目录中移动文件**  *示例： hadoop fs -mv /aaa/jdk.tar.gz /* |
| **-get**  **功能：等同于copyToLocal，就是从hdfs下载文件到本地**  示例：hadoop fs -get /aaa/jdk.tar.gz  **-getmerge**  **功能：合并下载多个文件**  *示例：比如hdfs的目录 /aaa/下有多个文件:log.1, log.2,log.3,...*  hadoop fs -getmerge /aaa/log.\* ./log.sum |
| **-put**  **功能：等同于copyFromLocal**  *示例：hadoop fs -put /aaa/jdk.tar.gz /bbb/jdk.tar.gz.2* |
| **-rm**  **功能：删除文件或文件夹**  *示例：hadoop fs -rm -r /aaa/bbb/*  **-rmdir**  **功能：删除空目录**  *示例：hadoop fs -rmdir /aaa/bbb/ccc* |
| **-df**  **功能：统计文件系统的可用空间信息**  *示例：hadoop fs -df -h /*  **-du**  **功能：统计文件夹的大小信息**  *示例：*  *hadoop fs -du -s -h /aaa/\** |
| **-count**  **功能：统计一个指定目录下的文件节点数量**  *示例：hadoop fs -count /aaa/* |
| **-setrep**  **功能：设置hdfs中文件的副本数量**  *示例：hadoop fs -setrep 3 /aaa/jdk.tar.gz* |

# \*\*\*\*\*\*HDFS原理篇\*\*\*\*\*\*

# 4. hdfs的工作机制

*（工作机制的学习主要是为加深对分布式系统的理解，以及增强遇到各种问题时的分析解决能力，形成一定的集群运维能力）*

*注：很多不是真正理解hadoop技术体系的人会常常觉得HDFS可用于网盘类应用，但实际并非如此。要想将技术准确用在恰当的地方，必须对技术有深刻的理解*

## 4.1 概述

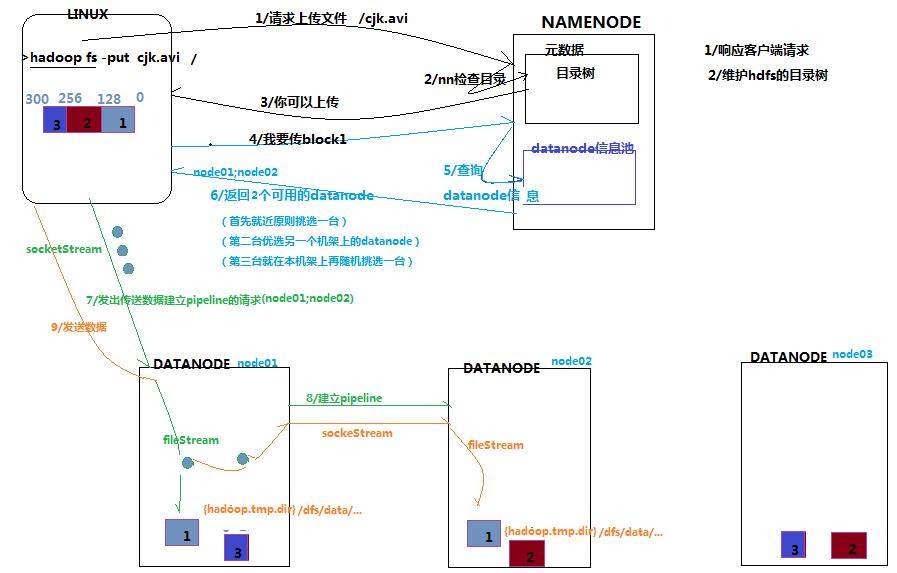
1. HDFS集群分为两大角色：NameNode、DataNode (Secondary Namenode)
2. NameNode负责管理整个文件系统的元数据
3. DataNode 负责管理用户的文件数据块
4. 文件会按照固定的大小（blocksize）切成若干块后分布式存储在若干台datanode上
5. 每一个文件块可以有多个副本，并存放在不同的datanode上
6. Datanode会定期向Namenode汇报自身所保存的文件block信息，而namenode则会负责保持文件的副本数量
7. HDFS的内部工作机制对客户端保持透明，客户端请求访问HDFS都是通过向namenode申请来进行

## 4.2 HDFS写数据流程

### 4.2.1 概述

客户端要向HDFS写数据，首先要跟namenode通信以确认可以写文件并获得接收文件block的datanode，然后，客户端按顺序将文件逐个block传递给相应datanode，并由接收到block的datanode负责向其他datanode复制block的副本

### 4.2.2 详细步骤图



### 4.2.3 详细步骤解析

1、根namenode通信请求上传文件，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、client请求第一个 block该传输到哪些datanode服务器上

4、namenode返回3个datanode服务器ABC

5、client请求3台dn中的一台A上传数据（本质上是一个RPC调用，建立pipeline），A收到请求会继续调用B，然后B调用C，将真个pipeline建立完成，逐级返回客户端

6、client开始往A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，A收到一个packet就会传给B，B传给C；A每传一个packet会放入一个应答队列等待应答

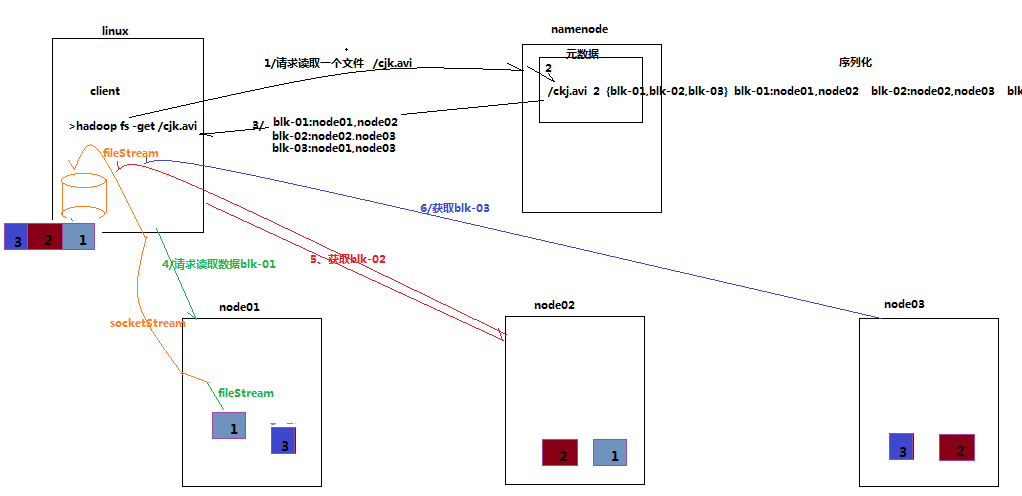
7、当一个block传输完成之后，client再次请求namenode上传第二个block的服务器。

## 4.3. HDFS读数据流程

### 4.3.1 概述

客户端将要读取的文件路径发送给namenode，namenode获取文件的元信息（主要是block的存放位置信息）返回给客户端，客户端根据返回的信息找到相应datanode逐个获取文件的block并在客户端本地进行数据追加合并从而获得整个文件

### 4.3.2 详细步骤图



### 4.3.3 详细步骤解析

1、跟namenode通信查询元数据，找到文件块所在的datanode服务器

2、挑选一台datanode（就近原则，然后随机）服务器，请求建立socket流

3、datanode开始发送数据（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，现在本地缓存，然后写入目标文件

# 5. NAMENODE工作机制

学习目标：理解namenode的工作机制尤其是**元数据管理**机制，以增强对HDFS工作原理的理解，及培养hadoop集群运营中“性能调优”、“namenode”故障问题的分析解决能力

*问题场景：*

*1、集群启动后，可以查看文件，但是上传文件时报错，打开web页面可看到namenode正处于safemode状态，怎么处理？*

*2、Namenode服务器的磁盘故障导致namenode宕机，如何挽救集群及数据？*

*3、Namenode是否可以有多个？namenode内存要配置多大？namenode跟集群数据存储能力有关系吗？*

*4、文件的blocksize究竟调大好还是调小好？*

*……*

*诸如此类问题的回答，都需要基于对namenode自身的工作原理的深刻理解*

## 5.1 NAMENODE职责

NAMENODE职责：

负责客户端请求的响应

元数据的管理（查询，修改）

## 5.2 元数据管理

namenode对数据的管理采用了三种存储形式：

内存元数据(NameSystem)

磁盘元数据镜像文件

数据操作日志文件（可通过日志运算出元数据）

### 5.2.1 元数据存储机制

A、内存中有一份完整的元数据(**内存meta data**)

B、磁盘有一个“准完整”的元数据镜像（**fsimage**）文件(在namenode的工作目录中)

C、用于衔接内存metadata和持久化元数据镜像fsimage之间的操作日志（**edits文件**）*注：当客户端对hdfs中的文件进行新增或者修改操作，操作记录首先被记入edits日志文件中，当客户端操作成功后，相应的元数据会更新到内存meta.data中*

### 5.2.2 元数据手动查看

可以通过hdfs的一个工具来查看edits中的信息

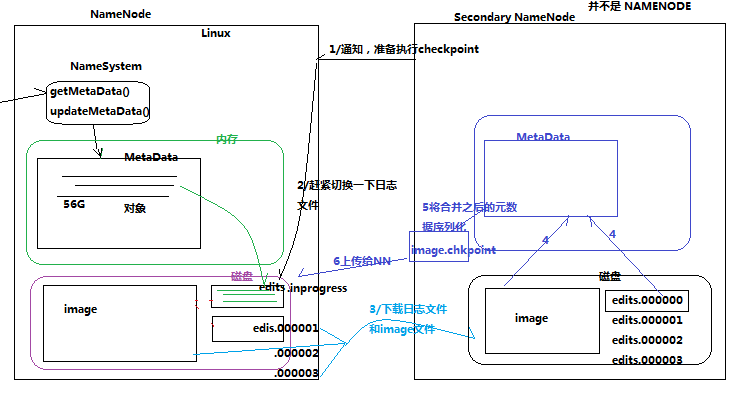
bin/hdfs oev -i edits -o edits.xml

bin/hdfs oiv -i fsimage\_0000000000000000087 -p XML -o fsimage.xml

### 5.2.3 元数据的checkpoint

每隔一段时间，会由secondary namenode将namenode上积累的所有edits和一个最新的fsimage下载到本地，并加载到内存进行merge（这个过程称为checkpoint）

#### checkpoint的详细过程



#### checkpoint操作的触发条件配置参数

|  |
| --- |
| dfs.namenode.checkpoint.check.period=60 #检查触发条件是否满足的频率，60秒  dfs.namenode.checkpoint.dir=file://${hadoop.tmp.dir}/dfs/namesecondary  #以上两个参数做checkpoint操作时，secondary namenode的本地工作目录  dfs.namenode.checkpoint.edits.dir=${dfs.namenode.checkpoint.dir}  dfs.namenode.checkpoint.max-retries=3 #最大重试次数  dfs.namenode.checkpoint.period=3600 #两次checkpoint之间的时间间隔3600秒  dfs.namenode.checkpoint.txns=1000000 #两次checkpoint之间最大的操作记录 |

#### checkpoint的附带作用

namenode和secondary namenode的工作目录存储结构完全相同，所以，当namenode故障退出需要重新恢复时，可以从secondary namenode的工作目录中将fsimage拷贝到namenode的工作目录，以恢复namenode的元数据

### 5.2.4 元数据目录说明

在第一次部署好Hadoop集群的时候，我们需要在NameNode（NN）节点上格式化磁盘：

|  |
| --- |
| $HADOOP\_HOME/bin/hdfs namenode -format |

格式化完成之后，将会在$dfs.namenode.name.dir/current目录下如下的文件结构

|  |
| --- |
| current/  |-- VERSION  |-- edits\_\*  |-- fsimage\_0000000000008547077  |-- fsimage\_0000000000008547077.md5  `-- seen\_txid |

其中的dfs.name.dir是在hdfs-site.xml文件中配置的，默认值如下：

|  |
| --- |
| <property>  <name>dfs.name.dir</name>  <value>file://${hadoop.tmp.dir}/dfs/name</value>  </property>  hadoop.tmp.dir是在core-site.xml中配置的，默认值如下  <property>  <name>hadoop.tmp.dir</name>  <value>/tmp/hadoop-${user.name}</value>  <description>A base for other temporary directories.</description>  </property> |

dfs.namenode.name.dir属性可以配置多个目录，

如/data1/dfs/name,/data2/dfs/name,/data3/dfs/name,....。各个目录存储的文件结构和内容都完全一样，相当于备份，这样做的好处是当其中一个目录损坏了，也不会影响到Hadoop的元数据，特别是当其中一个目录是NFS（网络文件系统Network File System，NFS）之上，即使你这台机器损坏了，元数据也得到保存。  
下面对$dfs.namenode.name.dir/current/目录下的文件进行解释。  
1、VERSION文件是Java属性文件，内容大致如下：

|  |
| --- |
| #Fri Nov 15 19:47:46 CST 2013  namespaceID=934548976  clusterID=CID-cdff7d73-93cd-4783-9399-0a22e6dce196  cTime=0  storageType=NAME\_NODE  blockpoolID=BP-893790215-192.168.24.72-1383809616115  layoutVersion=-47 |

其中  
　　（1）、namespaceID是文件系统的唯一标识符，在文件系统首次格式化之后生成的；  
　　（2）、storageType说明这个文件存储的是什么进程的数据结构信息（如果是DataNode，storageType=DATA\_NODE）；  
　　（3）、cTime表示NameNode存储时间的创建时间，由于我的NameNode没有更新过，所以这里的记录值为0，以后对NameNode升级之后，cTime将会记录更新时间戳；  
　　（4）、layoutVersion表示HDFS永久性数据结构的版本信息， 只要数据结构变更，版本号也要递减，此时的HDFS也需要升级，否则磁盘仍旧是使用旧版本的数据结构，这会导致新版本的NameNode无法使用；  
　　（5）、clusterID是系统生成或手动指定的集群ID，在-clusterid选项中可以使用它；如下说明

1. 使用如下命令格式化一个Namenode：

$HADOOP\_HOME/bin/hdfs namenode -format [-clusterId <cluster\_id>]

选择一个唯一的cluster\_id，并且这个cluster\_id不能与环境中其他集群有冲突。如果没有提供cluster\_id，则会自动生成一个唯一的ClusterID。

b、使用如下命令格式化其他Namenode：

$HADOOP\_HOME/bin/hdfs namenode -format -clusterId <cluster\_id>

c、升级集群至最新版本。在升级过程中需要提供一个ClusterID，例如：

$HADOOP\_PREFIX\_HOME/bin/hdfs start namenode --config $HADOOP\_CONF\_DIR  -upgrade -clusterId <cluster\_ID>

如果没有提供ClusterID，则会自动生成一个ClusterID。

　　（6）、blockpoolID：是针对每一个Namespace所对应的blockpool的ID，上面的这个BP-893790215-192.168.24.72-1383809616115就是在我的ns1的namespace下的存储块池的ID，这个ID包括了其对应的NameNode节点的ip地址。  
　　  
2、$dfs.namenode.name.dir/current/seen\_txid非常重要，是存放transactionId的文件，format之后是0，它代表的是namenode里面的edits\_\*文件的尾数，namenode重启的时候，会按照seen\_txid的数字，循序从头跑edits\_0000001~到seen\_txid的数字。所以当你的hdfs发生异常重启的时候，一定要比对seen\_txid内的数字是不是你edits最后的尾数，不然会发生建置namenode时metaData的资料有缺少，导致误删Datanode上多余Block的资讯。

3、$dfs.namenode.name.dir/current目录下在format的同时也会生成fsimage和edits文件，及其对应的md5校验文件。

补充：seen\_txid

文件中记录的是edits滚动的序号，每次重启namenode时，namenode就知道要将哪些edits进行加载edits

# 6. DATANODE的工作机制

*问题场景：*

*1、集群容量不够，怎么扩容？*

*2、如果有一些datanode宕机，该怎么办？*

*3、datanode明明已启动，但是集群中的可用datanode列表中就是没有，怎么办？*

*以上这类问题的解答，有赖于对datanode工作机制的深刻理解*

## 6.1 概述

1、Datanode工作职责：

存储管理用户的文件块数据

定期向namenode汇报自身所持有的block信息（通过心跳信息上报）

（这点很重要，因为，当集群中发生某些block副本失效时，集群如何恢复block初始副本数量的问题）

|  |
| --- |
| <property>  <name>dfs.blockreport.intervalMsec</name>  <value>3600000</value>  <description>Determines block reporting interval in milliseconds.</description>  </property> |

2、Datanode掉线判断时限参数

datanode进程死亡或者网络故障造成datanode无法与namenode通信，namenode不会立即把该节点判定为死亡，要经过一段时间，这段时间暂称作超时时长。HDFS默认的超时时长为10分钟+30秒。如果定义超时时间为timeout，则超时时长的计算公式为：

timeout = 2 \* heartbeat.recheck.interval + 10 \* dfs.heartbeat.interval。

而默认的heartbeat.recheck.interval 大小为5分钟，dfs.heartbeat.interval默认为3秒。

需要注意的是hdfs-site.xml 配置文件中的heartbeat.recheck.interval的单位为毫秒，dfs.heartbeat.interval的单位为秒。所以，举个例子，如果heartbeat.recheck.interval设置为5000（毫秒），dfs.heartbeat.interval设置为3（秒，默认），则总的超时时间为40秒。

|  |
| --- |
| <property>  <name>heartbeat.recheck.interval</name>  <value>2000</value>  </property>  <property>  <name>dfs.heartbeat.interval</name>  <value>1</value>  </property> |

## 6.2 观察验证DATANODE功能

上传一个文件，观察文件的block具体的物理存放情况：

在每一台datanode机器上的这个目录中能找到文件的切块：

/home/hadoop/app/hadoop-2.4.1/tmp/dfs/data/current/BP-193442119-192.168.2.120-1432457733977/current/finalized

# \*\*\*\*\*\*HDFS应用开发篇\*\*\*\*\*\*

# 7. HDFS的java操作

*hdfs在生产应用中主要是客户端的开发，其核心步骤是从hdfs提供的api中构造一个HDFS的访问客户端对象，然后通过该客户端对象操作（增删改查）HDFS上的文件*

## 7.1 搭建开发环境

1、引入依赖

|  |  |
| --- | --- |
| |  | | --- | | <dependency>  <groupId>org.apache.hadoop</groupId>  <artifactId>hadoop-client</artifactId>  <version>2.6.1</version>  </dependency> | |

*注：如需手动引入jar包，hdfs的jar包----hadoop的安装目录的share下*

2、window下开发的说明

建议在linux下进行hadoop应用的开发，不会存在兼容性问题。如在window上做客户端应用开发，需要设置以下环境：

1. 在windows的某个目录下解压一个hadoop的安装包
2. 将安装包下的lib和bin目录用对应windows版本平台编译的本地库替换
3. 在window系统中配置HADOOP\_HOME指向你解压的安装包
4. 在windows系统的path变量中加入hadoop的bin目录

## 7.2 获取api中的客户端对象

在java中操作hdfs，首先要获得一个客户端实例

|  |
| --- |
| Configuration conf = new Configuration()  FileSystem fs = FileSystem.get(conf) |

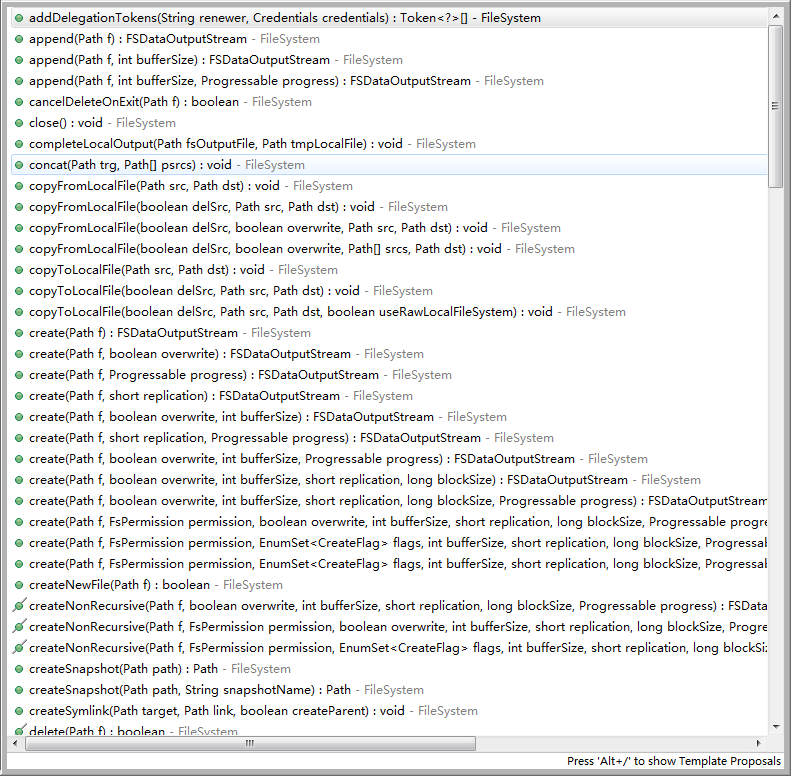
而我们的操作目标是HDFS，所以获取到的fs对象应该是DistributedFileSystem的实例；

get方法是从何处判断具体实例化那种客户端类呢？

**——从conf中的一个参数 fs.defaultFS的配置值判断；**

如果我们的代码中没有指定fs.defaultFS，并且工程classpath下也没有给定相应的配置，conf中的默认值就来自于hadoop的jar包中的core-default.xml，默认值为： file:///，则获取的将不是一个DistributedFileSystem的实例，而是一个本地文件系统的客户端对象

## 7.3 DistributedFileSystem实例对象所具备的方法



## 7.4 HDFS客户端操作数据代码示例：

### 7.4.1 文件的增删改查

|  |
| --- |
| public class HdfsClient {  FileSystem fs = null;  @Before  public void init() throws Exception {  // 构造一个配置参数对象，设置一个参数：我们要访问的hdfs的URI  // 从而FileSystem.get()方法就知道应该是去构造一个访问hdfs文件系统的客户端，以及hdfs的访问地址  // new Configuration();的时候，它就会去加载jar包中的hdfs-default.xml  // 然后再加载classpath下的hdfs-site.xml  Configuration conf = new Configuration();  conf.set("fs.defaultFS", "hdfs://hdp-node01:9000");  /\*\*  \* 参数优先级： 1、客户端代码中设置的值 2、classpath下的用户自定义配置文件 3、然后是服务器的默认配置  \*/  conf.set("dfs.replication", "3");  // 获取一个hdfs的访问客户端，根据参数，这个实例应该是DistributedFileSystem的实例  // fs = FileSystem.get(conf);  // 如果这样去获取，那conf里面就可以不要配"fs.defaultFS"参数，而且，这个客户端的身份标识已经是hadoop用户  fs = FileSystem.get(new URI("hdfs://hdp-node01:9000"), conf, "hadoop");  }  /\*\*  \* 往hdfs上传文件  \*  \* @throws Exception  \*/  @Test  public void testAddFileToHdfs() throws Exception {  // 要上传的文件所在的本地路径  Path src = new Path("g:/redis-recommend.zip");  // 要上传到hdfs的目标路径  Path dst = new Path("/aaa");  fs.copyFromLocalFile(src, dst);  fs.close();  }  /\*\*  \* 从hdfs中复制文件到本地文件系统  \*  \* @throws IOException  \* @throws IllegalArgumentException  \*/  @Test  public void testDownloadFileToLocal() throws IllegalArgumentException, IOException {  fs.copyToLocalFile(new Path("/jdk-7u65-linux-i586.tar.gz"), new Path("d:/"));  fs.close();  }  @Test  public void testMkdirAndDeleteAndRename() throws IllegalArgumentException, IOException {  // 创建目录  fs.mkdirs(new Path("/a1/b1/c1"));  // 删除文件夹 ，如果是非空文件夹，参数2必须给值true  fs.delete(new Path("/aaa"), true);  // 重命名文件或文件夹  fs.rename(new Path("/a1"), new Path("/a2"));  }  /\*\*  \* 查看目录信息，只显示文件  \*  \* @throws IOException  \* @throws IllegalArgumentException  \* @throws FileNotFoundException  \*/  @Test  public void testListFiles() throws FileNotFoundException, IllegalArgumentException, IOException {  // 思考：为什么返回迭代器，而不是List之类的容器  RemoteIterator<LocatedFileStatus> listFiles = fs.listFiles(new Path("/"), true);  while (listFiles.hasNext()) {  LocatedFileStatus fileStatus = listFiles.next();  System.out.println(fileStatus.getPath().getName());  System.out.println(fileStatus.getBlockSize());  System.out.println(fileStatus.getPermission());  System.out.println(fileStatus.getLen());  BlockLocation[] blockLocations = fileStatus.getBlockLocations();  for (BlockLocation bl : blockLocations) {  System.out.println("block-length:" + bl.getLength() + "--" + "block-offset:" + bl.getOffset());  String[] hosts = bl.getHosts();  for (String host : hosts) {  System.out.println(host);  }  }  System.out.println("--------------为angelababy打印的分割线--------------");  }  }  /\*\*  \* 查看文件及文件夹信息  \*  \* @throws IOException  \* @throws IllegalArgumentException  \* @throws FileNotFoundException  \*/  @Test  public void testListAll() throws FileNotFoundException, IllegalArgumentException, IOException {  FileStatus[] listStatus = fs.listStatus(new Path("/"));  String flag = "d-- ";  for (FileStatus fstatus : listStatus) {  if (fstatus.isFile()) flag = "f-- ";  System.out.println(flag + fstatus.getPath().getName());  }  }  } |

### 7.4.2 通过流的方式访问hdfs

|  |
| --- |
| */\*\**  *\* 相对那些封装好的方法而言的更底层一些的操作方式*  *\* 上层那些mapreduce spark等运算框架，去hdfs中获取数据的时候，就是调的这种底层的api*  *\* @author*  *\**  *\*/*  *public class StreamAccess {*    *FileSystem fs = null;*  *@Before*  *public void init() throws Exception {*  *Configuration conf = new Configuration();*  *fs = FileSystem.get(new URI("hdfs://hdp-node01:9000"), conf, "hadoop");*  *}*    */\*\**  *\* 通过流的方式上传文件到hdfs*  *\* @throws Exception*  *\*/*  *@Test*  *public void testUpload() throws Exception {*    *FSDataOutputStream outputStream = fs.create(new Path("/angelababy.love"), true);*  *FileInputStream inputStream = new FileInputStream("c:/angelababy.love");*    *IOUtils.copy(inputStream, outputStream);*    *}*    *@Test*  *public void testDownLoadFileToLocal() throws IllegalArgumentException, IOException{*    *//先获取一个文件的输入流----针对hdfs上的*  *FSDataInputStream in = fs.open(new Path("/jdk-7u65-linux-i586.tar.gz"));*    *//再构造一个文件的输出流----针对本地的*  *FileOutputStream out = new FileOutputStream(new File("c:/jdk.tar.gz"));*    *//再将输入流中数据传输到输出流*  *IOUtils.copyBytes(in, out, 4096);*      *}*      */\*\**  *\* hdfs支持随机定位进行文件读取，而且可以方便地读取指定长度*  *\* 用于上层分布式运算框架并发处理数据*  *\* @throws IllegalArgumentException*  *\* @throws IOException*  *\*/*  *@Test*  *public void testRandomAccess() throws IllegalArgumentException, IOException{*  *//先获取一个文件的输入流----针对hdfs上的*  *FSDataInputStream in = fs.open(new Path("/iloveyou.txt"));*      *//可以将流的起始偏移量进行自定义*  *in.seek(22);*    *//再构造一个文件的输出流----针对本地的*  *FileOutputStream out = new FileOutputStream(new File("c:/iloveyou.line.2.txt"));*    *IOUtils.copyBytes(in,out,19L,true);*    *}*        */\*\**  *\* 显示hdfs上文件的内容*  *\* @throws IOException*  *\* @throws IllegalArgumentException*  *\*/*  *@Test*  *public void testCat() throws IllegalArgumentException, IOException{*    *FSDataInputStream in = fs.open(new Path("/iloveyou.txt"));*    *IOUtils.copyBytes(in, System.out, 1024);*  *}*  *}* |

### 7.4.3 场景编程

在mapreduce 、spark等运算框架中，有一个核心思想就是将运算移往数据，或者说，就是要在并发计算中尽可能让运算本地化，这就需要获取数据所在位置的信息并进行相应范围读取

以下模拟实现：获取一个文件的所有block位置信息，然后读取指定block中的内容

|  |
| --- |
| *@Test*  *public void testCat() throws IllegalArgumentException, IOException{*    *FSDataInputStream in = fs.open(new Path("/weblog/input/access.log.10"));*  *//拿到文件信息*  *FileStatus[] listStatus = fs.listStatus(new Path("/weblog/input/access.log.10"));*  *//获取这个文件的所有block的信息*  *BlockLocation[] fileBlockLocations = fs.getFileBlockLocations(listStatus[0], 0L, listStatus[0].getLen());*  *//第一个block的长度*  *long length = fileBlockLocations[0].getLength();*  *//第一个block的起始偏移量*  *long offset = fileBlockLocations[0].getOffset();*    *System.out.println(length);*  *System.out.println(offset);*    *//获取第一个block写入输出流*  *// IOUtils.copyBytes(in, System.out, (int)length);*  *byte[] b = new byte[4096];*    *FileOutputStream os = new FileOutputStream(new File("d:/block0"));*  *while(in.read(offset, b, 0, 4096)!=-1){*  *os.write(b);*  *offset += 4096;*  *if(offset>=length) return;*  *};*  *os.flush();*  *os.close();*  *in.close();*  *}* |