Day02_HDFS

大数据-张军锋 Day02 元数据 hdfs

集群中扮演的角色

hdfs

> 主节点 namenode ，hdfs的端口号是50070

namenode： 处理请求，分配处理任务；负责心跳连接；负责均衡；副本
datanote：数据的读写请求执行和数据的保存操作
secondarynamenode ： 备份NameNode上的数据，合成fsimage和fsedits文件，是namenode的助手

yarn

> 主节点 resourceManager ， yarn主节点端口号是8088

resourceManager :在YARN中，ResourceManager负责集群中所有资源的统一管理和分配，它接收来自各个节点（NodeManager）的资源汇报信息，并把这些信息按照一定的策略分配给各个应用程序
nodeManager ： NodeManager是运行在单个节点上的代理，它管理Hadoop集群中单个计算节点，功能包括与ResourceManager保持通信，管理Container的生命周期、监控每个Container的资源使用(内存、CPU等）情况、追踪节点健康状况、管理日志和不同应用程序用到的附属服务等。

元数据

> 元数据就是描述数据的数据
> 数据字典表就是存储元素数据的表
>
> 注意：在数据库中将元数据存储在数据字典表中，也就是存储在磁盘文件中，但是在hdfs中，将元数据存储在内存中，因此对于主节点内存的要求就很高了，因此hdfs的缺点就是尽量不要存储小文件，减少元数据的产生

在hadoop生态圈中，对于hdfs中元数据是我们自己指定存储位置的，具体配置在hdfs-site.xml文件中

```
-rw-r--r--. 1 20415 101  2250 Jul 31 17:35 yarn-env.cmd
-rw-r--r--. 1 20415 101  4572 Oct 10 01:58 yarn-env.sh
-rw-r--r--. 1 20415 101  1668 Oct 10 06:17 yarn-site.xml
[root@master hadoop]# vim hdfs-site.xml
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
<name>dfs.namenode.secondary.http-address</name>
<value>master:9001</value>
</property>

<property>
<name>dfs.namenode.name.dir</name>
<value>file:/usr/dfs/name</value>
</property>

<property>
<name>dfs.datanode.data.dir</name>
<value>file:/usr/dfs/data</value>
```

元数据存放位置

在data目录下元数据存放在两个文件中，分别是fsimage和fsedits文件中

fsimage 文件存放元数据的具体内容包含以下信息：文件名称、文件大小、路径、创建者、时间、文件的分块信息

edits 文件存放元数据的操作日志：创建一个文件、删除、修改，将操作记录的形式保存早edits里面

```
18:24 edits_0000000000000086911-0000000000000086988
19:24 edits_0000000000000086989-0000000000000087033
20:24 edits_0000000000000087034-0000000000000087235
20:29 edits_0000000000000087236-0000000000000087254
15:53 edits_0000000000000087255-0000000000000087271
16:54 edits_0000000000000087272-0000000000000087287
17:54 edits_0000000000000087288-0000000000000087298
17:54 edits_0000000000000087299-0000000000000087299
20:11 edits_0000000000000087300-0000000000000087325
20:22 edits_0000000000000087326-0000000000000087334
15:50 edits_0000000000000087335-0000000000000087344
16:50 edits_0000000000000087345-0000000000000087383
17:50 edits_0000000000000087384-0000000000000087463
17:50 edits_0000000000000087464-0000000000000087464
15:30 edits_0000000000000087465-0000000000000087466
16:30 edits_0000000000000087467-0000000000000088231
17:00 edits_0000000000000088232-0000000000000100559
10:37 edits_0000000000000100560-0000000000000100561
11:37 edits_0000000000000100562-0000000000000100568
11:37 edits_inprogress_0000000000000100569
10:37 fsimage_0000000000000100561
10:37 fsimage_0000000000000100561.md5
11:37 fsimage_0000000000000100568
11:37 fsimage_0000000000000100568.md5
12:07 fsimage.ckpt_0000000000000023039
11:37 seen_txid
10:02 VERSION
```

元数据的操作日志
创建一个文件
删除 修改
以操作记录的形式保存在edits里面

元数据的具体内容
文件名称
文件大小
路径
创建者
时间
文件的分块信息

每一个block的位置：在集群中的哪一个节点上

元数据的管理

> 元数据存储在namenode中,用于记录数据的block存储到哪个datanode中

- 因为客户端程序在对集群中的文件进行读取和存储的过程中都需要访问元数据,所以对元数据的实时性要求比较高,所以元数据信息是存放到内存中的
- 1.客户端上传文件的时候,首先请求 `namenode` ,获取块和 `datanode` 的信息
- 2. `namenode` 还会将这些信息存储到 `edits log` 日志文件(永远是64M大小)中进行记录
- 3.当客户端将对应的 `block` 写入 `datanode` 中,会告知 `namenode` 写入完成
- 4. `namenode` 会将记录在 `edits log` 日志文件中的元数据信息放入内存中
    - 当客户端要去下载文件就可以直接读取内存中的元数据信息

- 5.每当 `edits log` 中的元数据满的时候,将 `edits log` 中的数据和 `fsimage` 中的数据进行合并,然后生成新的 `edits log` ,是一个空的文件
    - 因为 `edits log` 和 `fsimage` 中的数据格式不同,当合并元数据信息的时候,需要计算相关内容,这个计算的过程是在 `secondarynamenode` 中进行
    - 当 `edits log` 满的时候, `namenode` 会通知 `secondarynamenode` 进行 `checkpoint` 的操作, `secondarynamenode` 会通知 `namenode` 进行停止操作, `namenode` 会生成一个新的文件 `edits.new` 将此过程中产生的新的数据写入此文件中
    - `secondarynamenode` 会将 `namenode` 中的 `edits log` 和 `fsimage` 进行下载,然后进行合并生成 `fsimage.checkpoint` 文件,将新的镜像上传给 `namenode`
    - `namenode` 会将 `fsimage` 进行替换,并将 `edits.new` 重命名为 `edits log`

元数据的格式

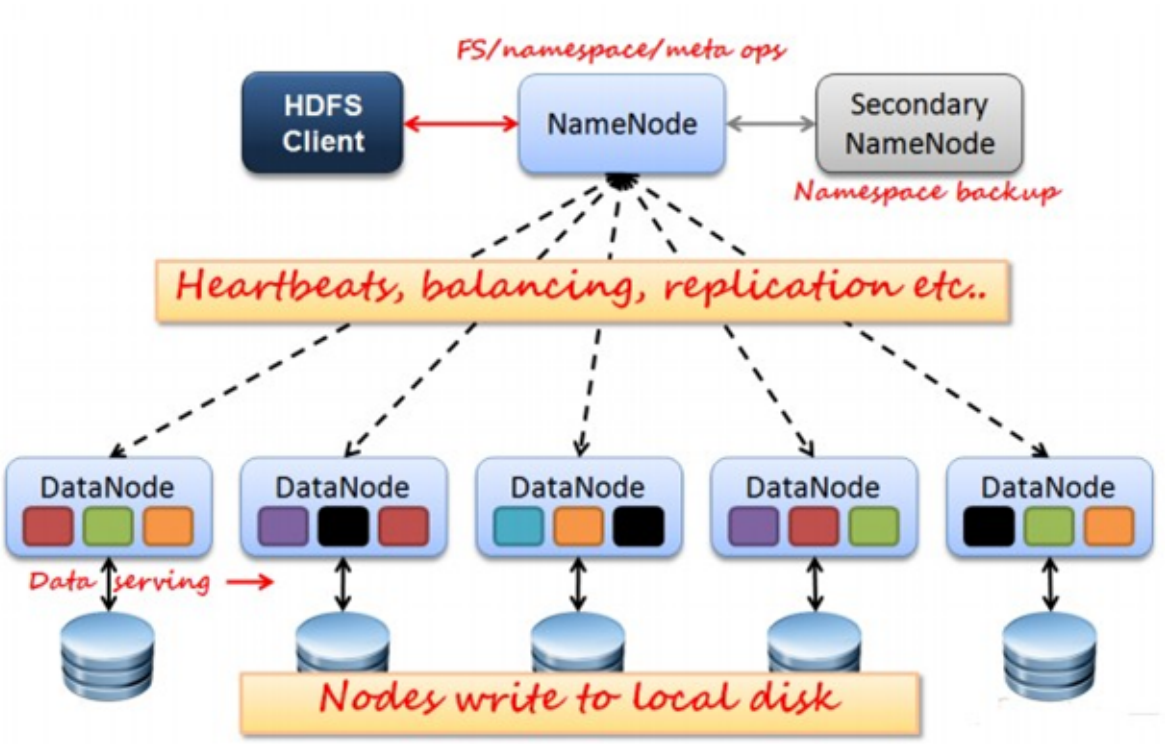
- 元数据的数据格式为 `NameNode(FileName, replicas, block-ids,id2host...)` 如虚拟目录下 `/dir/abc.txt` 文件被分成两个块id为 `blk_1` 和 `blk_2` ,每个块的分为3个备份, `blk_1` 存在 `h0,h1,h3` 三个节点中, `blk_2` 存在 `h0,h2,h4` 三个节点中
- 具体格式为: `/dir/abc.txt, 3 ,{blk_1,blk_2}, [{blk_1:[h0,h1,h3]},{blk_2:[h0,h2,h4]}]`

checkpoint触发机制

> 对于secondarynamenode触发checkpoint有两种机制,可以在hdfs-site.xml中进行设置

- 1.fs.checkpoint.period 指定两次checkpoint的最大时间间隔，默认3600秒。
- 2.fs.checkpoint.size 规定edits文件的最大值，一旦超过这个值则强制checkpoint，不管是否到达最大时间间隔。默认大小是64M。
- 以上两种机制哪种先达到按照哪种执行

HDFS的启动过程

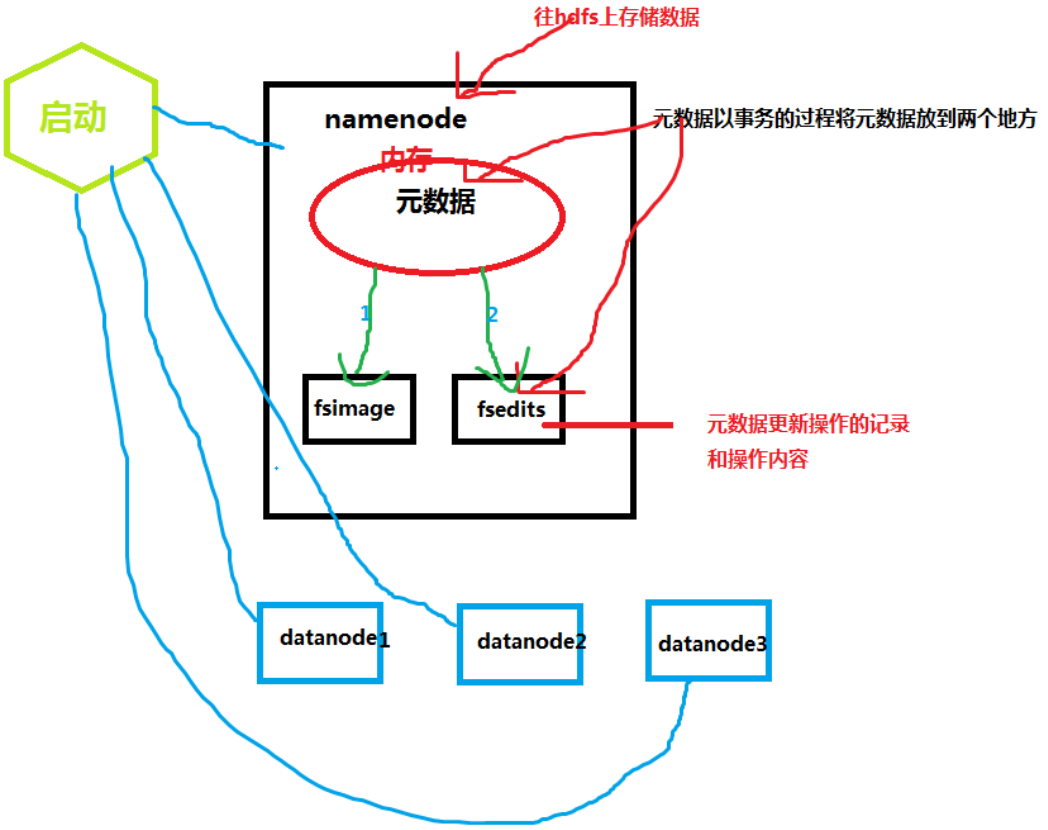


> 如上图所示，HDFS也是按照Master和Slave的结构。分NameNode、SecondaryNameNode、DataNode这几个角色。

- NameNode：是Master节点，是大领导。管理数据块映射；处理客户端的读写请求；配置副本策略；管理HDFS的名称空间；
- SecondaryNameNode：是一个小弟，分担大哥namenode的工作量；是NameNode的冷备份；合并fsimage和fsedits然后再发给namenode。
- DataNode：Slave节点，奴隶，干活的。负责存储client发来的数据块block；执行数据块的读写操作。
- 热备份：b是a的热备份，如果a坏掉。那么b马上运行代替a的工作。
- 冷备份：b是a的冷备份，如果a坏掉。那么b不能马上代替a工作。但是b上存储a的一些信息，减少a坏掉之后的损失。
- fsimage:元数据镜像文件（文件系统的目录树。）
- edits：元数据的操作日志（针对文件系统做的修改操作记录）
- namenode内存中存储的是=fsimage+edits。

SecondaryNameNode负责定时默认1小时，从namenode上，获取fsimage和edits来进行合并，然后再发送给namenode。减少namenode的工作量。

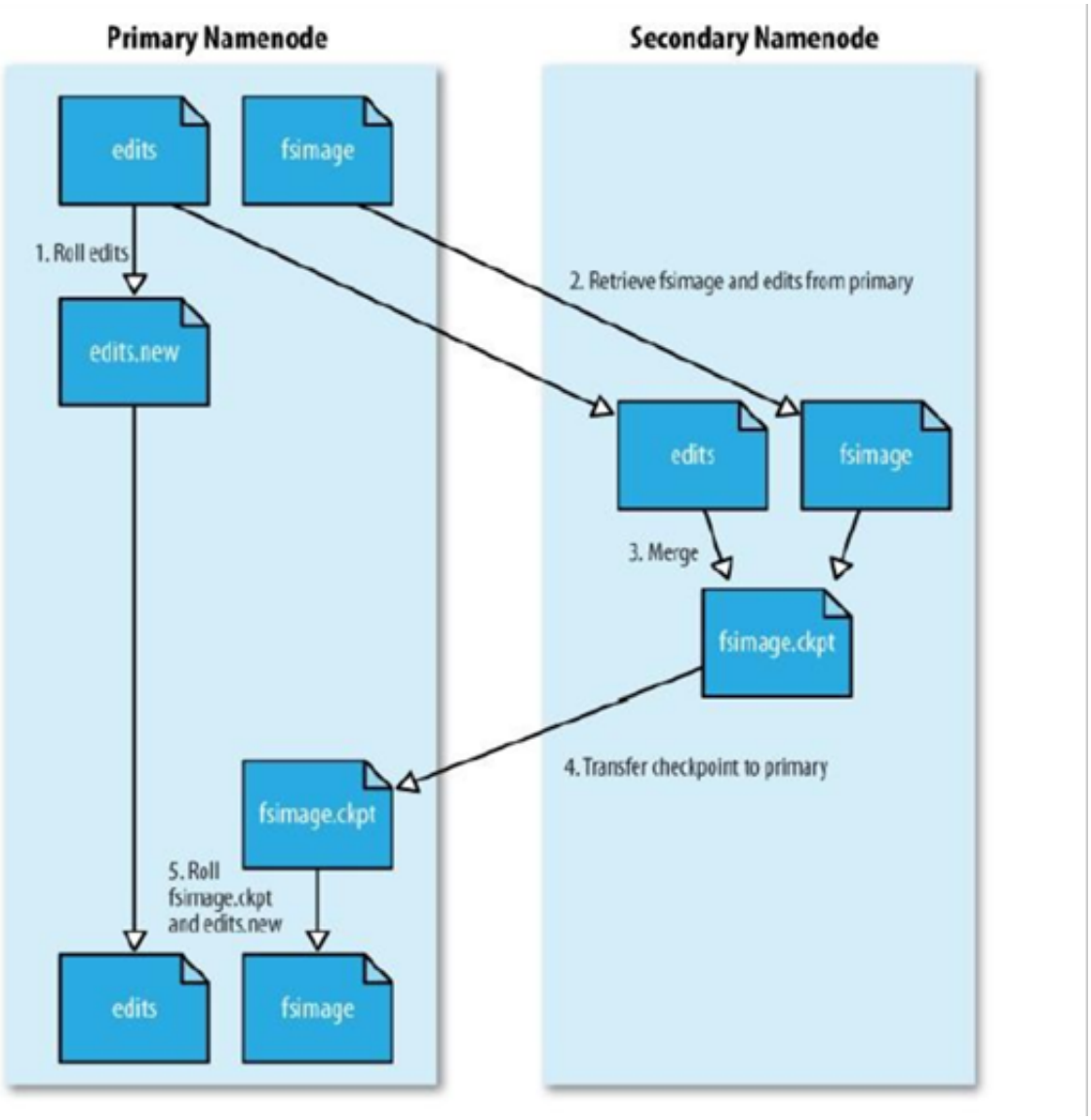
HDFS启动



> 当启动hdfs时加载namenode和datanode.

- 加载namenode
    - 首先加载fsimage文件
    - 加载edits文件
    - 当向hdfs中写数据时，fsedits文件会记录操作的过程，时间久了，就会产生大量的文件，当下次重新启动hdfs，hdfs会读取fsimage文件和fsedits文件，因为文件过大，需要读取很长时间。当然，这种现象不是我们想要看到的。因此，hdfs会将这些任务交给secondarynamenode进行监控，可以通过设置一段时间(默认是一个小时)内进行合并，也可以设置当文件达到默认值128M时，进行合并。secondarynamenode的工作原理见下文。
- 加载datanode信息，将datanode上存储的信息返回给namenode，其中包含当前节点上存储的数据块的信息

SecondaryNameNode工作流程



1. namenode 响应 Secondary namenode 请求，将 edit log 推送给 Secondary namenode ， 开始重新写一个新的 edit log
2. Secondary namenode 收到来自 namenode 的 fsimage 文件和 edit log
3. Secondary namenode 将 fsimage 加载到内存，应用 edit log ， 并生成一 个新的 fsimage 文件
4. Secondary namenode 将新的 fsimage 推送给 Namenode
5. Namenode 用新的 fsimage 取代旧的 fsimage ， 在 fstime 文件中记下检查 点发生的时间

HDFS读写操作原理

写操作

> 有一个文件FileA，100M大小。Client将FileA写入到HDFS上。
> HDFS按默认配置。
> HDFS分布在三个机架上Rack1，Rack2，Rack3。

a. Client将FileA按64M分块。分成两块，block1和Block2;
b. Client向nameNode发送写数据请求，如图蓝色虚线①——>。
c. NameNode节点，记录block信息。并返回可用的DataNode，如粉色虚线②———>。

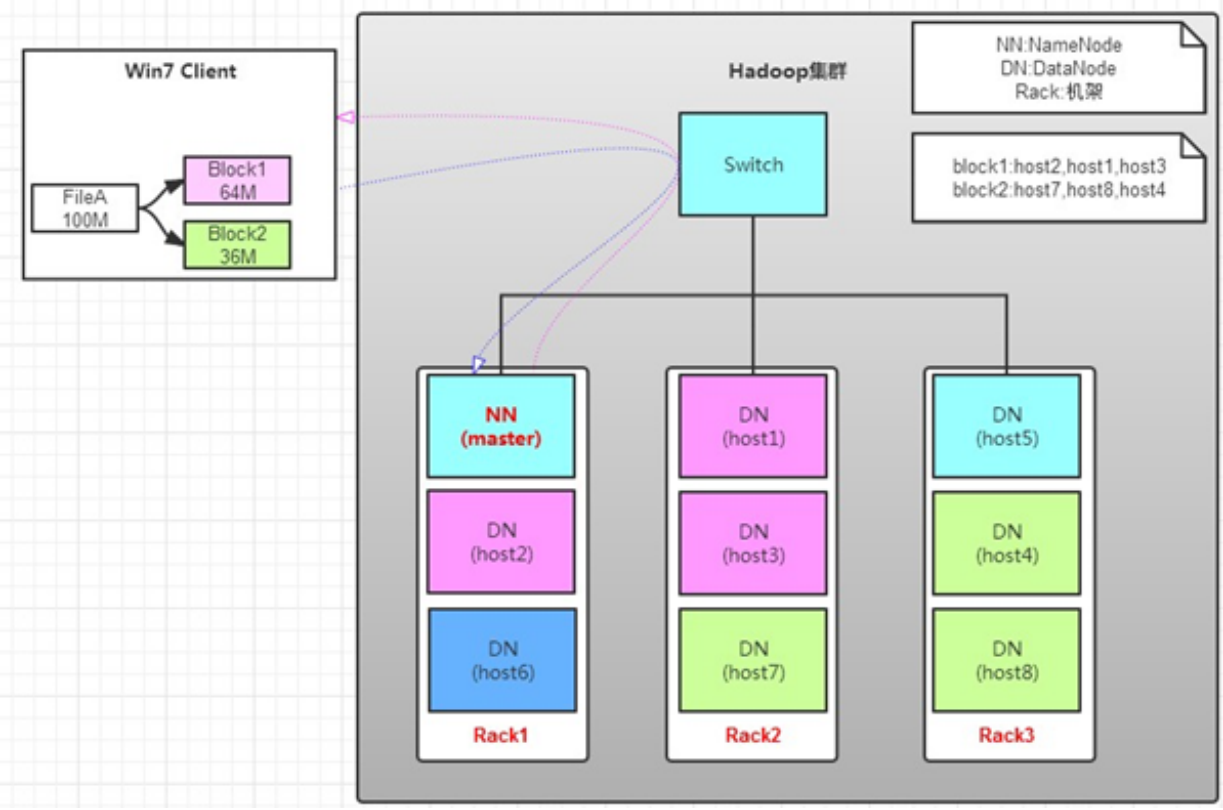
> Block1: host2,host1,host3
> Block2: host7,host8,host4

原理：
NameNode具有RackAware机架感知功能，这个可以配置。
若client为DataNode节点，那存储block时，规则为：副本1，同client的节点上；副本2，不同机架节点上；副本3，同第二个副本机架的另一个节点上；其他副本随机挑选。
若client不为DataNode节点，那存储block时，规则为：副本1，随机选择一个节点上；副本2，不同副本1，机架上；副本3，同副本2相同的另一个节点上；其他副本随机挑选。
d. client向DataNode发送block1；发送过程是以流式写入。

流式写入过程
1. 将64M的block1按64k的package划分;
2. 然后将第一个package发送给host2;
3. host2接收完后，将第一个package发送给host1，同时client向host2发送第二个package；
4. host1接收完第一个package后，发送给host3，同时接收host2发来的第二个package。
5. 以此类推，如图红线实线所示，直到将block1发送完毕。
6. host2,host1,host3向NameNode，host2向Client发送通知，说“消息发送完了”。如图粉红颜色实线所示。
7. client收到host2发来的消息后，向namenode发送消息，说我写完了。这样就真完成了。如图黄色粗实线
8. 发送完block1后，再向host7，host8，host4发送block2，如图蓝色实线所示。
9. 发送完block2后，host7,host8,host4向NameNode，host7向Client发送通知，如图浅绿色实线所示。
10client向NameNode发送消息，说我写完了，如图黄色粗实线。。。这样就完毕了。
分析，通过写过程，我们可以了解到：

① 写1T文件，我们需要3T的存储，3T的网络流量带宽。
② 在执行读或写的过程中，NameNode和DataNode通过HeartBeat进行保存通信，确定DataNode活着。如果发现DataNode死掉了，就将死掉的DataNode上的数据，放到其他节点去。读取时，要读其他节点去。
③ 挂掉一个节点，没关系，还有其他节点可以备份；甚至，挂掉某一个机架，也没关系；其他机架上，也有备份。

写操作



读操作就简单一些了，如上图所示，client要从datanode上，读取FileA。而FileA由block1和block2组成。

读操作流程为：

1. client向namenode发送读请求。
2. namenode查看Metadata信息，返回fileA的block的位置。

block1:host2,host1,host3
block2:host7,host8,host4

3. block的位置是有先后顺序的，先读block1，再读block2。而且block1去host2上读取；然后block2，去host7上读取；

上面例子中，client位于机架外，那么如果client位于机架内某个DataNode上，例如,client是host6。那么读取的时候，遵循的规律是：**优选读取本机架上的数据**

HDFS通信协议

> 所有的 HDFS 通讯协议都是构建在 TCP/IP 协议上。客户端通过一个可 配置的端口连接到 Namenode，通过 ClientProtocol 与 Namenode 交互。而 Datanode 是使用 DatanodeProtocol 与 Namenode 交互。再设计上， DataNode 通过周期性的向 NameNode 发送心跳和数据块来保持和 NameNode 的通信，数据块报告的信息包括数据块的属性，即数据块属于哪 个文件，数据块 ID ，修改时间等， NameNode 的 DataNode 和数据块的映射 关系就是通过系统启动时 DataNode 的数据块报告建立的。从 ClientProtocol 和 Datanodeprotocol 抽象出一个远程调用 (RPC) ， 在设计上， Namenode 不会主动发起 RPC ， 而是是响应来自客户端和 Datanode 的 RPC 请求。

HDFS文件权限

HDFS文件权限与Linux文件权限类似
r:read；w:write；x：execute

> 如果Linux系统用用户xxx使用hadoop命令创建一个文件，那么，在hdfs中这个文件的owner就是xxx
> HDFS的权限目的是将控制权交出去，本身只判断用户和权限，至于用户是不是真的，不管。

HDFS安全模式

> NameNode是运行在安全模式的。即对外（客户端）只读，所以此段时间内对hdfs的写入、删除、重命名都会失败
> 可以通过命令进出安全模式 `hdfs dfadmin -safemode enter` and `hdfs dfsadmin -safemode leave`

Overview 'master:9000' (active)

Started:	Wed Oct 11 03:23:27 PDT 2017
Version:	2.7.4, rcd915e1e8d9d0131462a0b7301586c175728a282
Compiled:	2017-08-01T00:29Z by kshvachk from branch-2.7.4
Cluster ID:	CID-013b8d3a-85ad-4fbf-862b-85417b203d36
Block Pool ID:	BP-1405003921-192.168.159.137-1507717383213

Summary

Security is off.

Safe mode is ON. It was turned on manually. Use "hdfs dfsadmin -safemode leave" to turn safe mode off.

3 files and directories, 3 blocks = 6 total filesystem object(s).

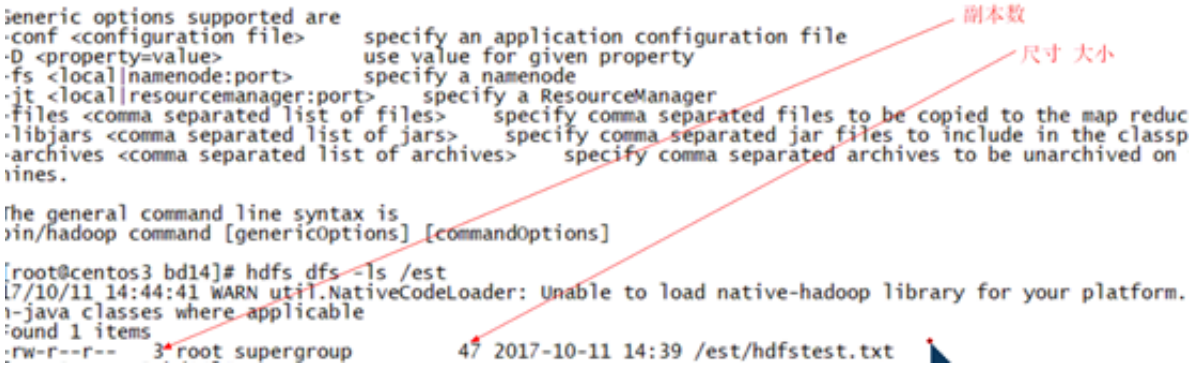
Heap Memory used 31.66 MB of 61.55 MB Heap Memory. Max Heap Memory is 966.69 MB.

Non Heap Memory used 53.76 MB of 55 MB Commited Non Heap Memory. Max Non Heap Memory is -1 B.

Configured Capacity:	52.47 GB
DFS Used:	537.84 MB (1%)
Non DFS Used:	11.84 GB

HDFS中常用到的命令

hdfs dfs -ls / ：列举出根目录下的内容
hdfs dfs -mkdir /test ：创建文件夹
hdfs dfs -put htfstest.txt /test/ ： 上传文件
hdfs dfs -cat /test/htfstest.txt ： 读取文件
hdfs dfs -get /test/htfstest.txt /root/a.txt ：下载文件
hdfs dfs –help ：查看htfs文档
hdfs dfs -chmod 777 /test/htfstest.txt ：修改文件权限
hdfs dfs -checksum /test/htfstest.txt ：查看MD5信息
hdfs dfs –df ：查看磁盘利用率

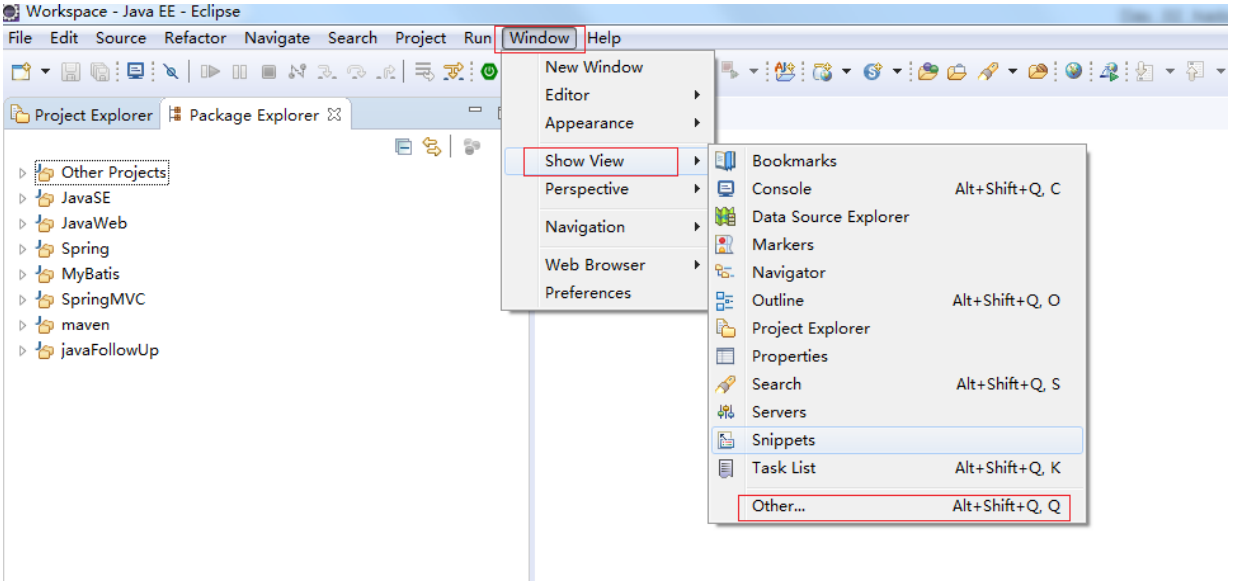


hdfs dfsadmin -safemode enter : 进入安全模式，安全模式只能读，不能写
hdfs dfsadmin -safemode leave ： 退出安全模式
hdfs fsck / ： 查看目录基本信息

start-balancer.sh : 负载均衡,可以使DataNode节点上选择策略重新平衡DataNode上的数据块的分布

eclipse 连接hdfs

1. 下载eclipse连接htfs所需要的jar包，下载地址 https://github.com/winghc/hadoop2x-eclipse-plugin/blob/master/release/hadoop-eclipse-plugin-2.6.0.jar
2. 将下载好的jar包放在eclipse安装路径下的plugins文件下，启动eclipse
3. [window–>Show View –> Other]



4. 点击右上角的图标

5.填写如下图所示信息

6.出现下图所示信息，说明连接成功