尚硅谷大数据技术之 Hadoop（HDFS）

(作者：尚硅谷大数据研发部)

版本 V3.0

第 1 章 HDFS 概述

1.1 HDFS 产出背景及定义

HDFS概述

尚硅谷

1.1 HDFS□□□□

随着数据量越来越大，在一个操作系统存不下所有的数据，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。HDFS只是分布式文件管理系统中的一种。

1.2 HDFS□□

HDFS（Hadoop Distributed File System），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。

HDFS的使用场景：适合一次写入，多次读出的场景，且不支持文件的修改。适合用来做数据分析，并不适合用来做网盘应用。

图 1-1 HDFS 概述

1.2 HDFS 优缺点

HDFS优缺点

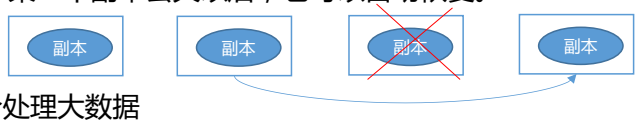
尚硅谷

1.2.1 优点

1）高容错性

（1）数据自动保存多个副本。它通过增加副本的形式，提高容错性。

（2）某一个副本丢失以后，它可以自动恢复。



2）适合处理大数据

（1）数据规模：能够处理数据规模达到GB、TB、甚至PB级别的数据；

（2）文件规模：能够处理百万规模以上的文件数量，数量相当之大。

3）可构建在廉价机器上，通过多副本机制，提高可靠性。

图 1-2 HDFS 优缺点

HDFS优缺点

1.2.2 缺点

1）不适合低延时数据访问，比如毫秒级的存储数据，是做不到的。

2）无法高效的对大量小文件进行存储。

（1）存储大量小文件的话，它会占用NameNode大量的内存来存储文件目录和块信息。这样是不可取的，因为NameNode的内存总是有限的；

（2）小文件存储的寻址时间会超过读取时间，它违反了HDFS的设计目标。

3）不支持并发写入、文件随机修改。

（1）一个文件只能有一个写，不允许多个线程同时写；

（2）仅支持数据append（追加），不支持文件的随机修改。

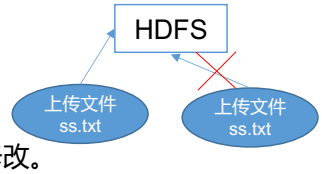
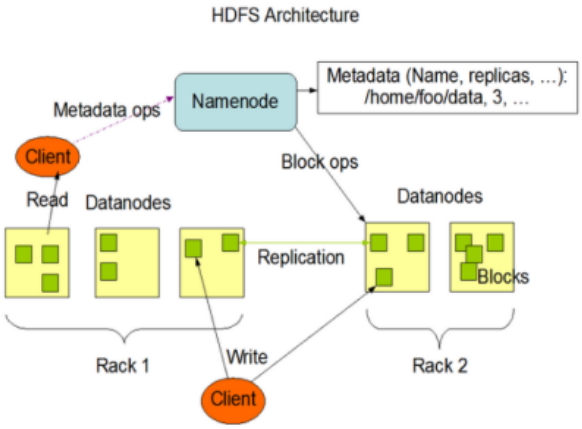


图 1-3 HDFS 优缺点

1.3 HDFS 组成架构

HDFS□□□□



1）NameNode（nn）：就是Master，它是一个主管、管理者。

（1）管理HDFS的名称空间；

（2）配置副本策略；

（3）管理数据块（Block）映射信息；

（4）处理客户端读写请求。

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

（1）存储实际的数据块；

（2）执行数据块的读/写操作。

图 1-4 HDFS 组成架构

HDFS□□□□

3）Client：就是客户端。

（1）文件切分。文件上传HDFS的时候，Client将文件切分成一个一个的Block，然后进行上传；

（2）与NameNode交互，获取文件的位置信息；

（3）与DataNode交互，读取或者写入数据；

（4）Client提供一些命令来管理HDFS，比如NameNode格式化；

（5）Client可以通过一些命令来访问HDFS，比如对HDFS增删查改操作；

4）Secondary NameNode：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）辅助NameNode，分担其工作量，比如定期合并Fsimage和Edits，并推送给NameNode ；

（2）在紧急情况下，可辅助恢复NameNode。

图 1-5 HDFS 组成架构

1.4 HDFS 文件块大小（面试重点）

HDFS 文件块大小

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Hadoop3.1.3版本中是128M，hadoop2.x也是128M，hadopp1.x是64M。

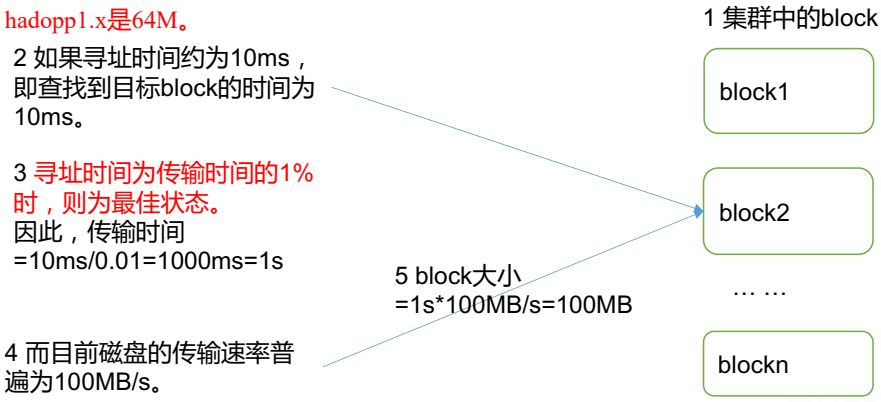


图 1-6 HDFS 文件块大小

真实企业往往是 256 或 512. 一般由磁盘传输速率决定.

思考：为什么块的大小不能设置太小，也不能设置太大？

（1）HDFS的块设置太小，会增加寻址时间，程序一直在找块的开始位置；

（2）如果块设置的太大，从磁盘传输数据的时间会明显大于定位这个块开始位置所需的时间。导致程序在处理这块数据时，会非常慢。

总结：HDFS块的大小设置主要取决于磁盘传输速率。

图 1-7 HDFS 文件块大小

第 2 章 HDFS 的 Shell 操作（开发重点）

1）基本语法

bin/hadoop fs 具体命令 OR bin/hdfs dfs 具体命令

两个是完全相同的。

2）命令大全

```
[atguigu@hadoop102 hadoop-3.1.3]$ bin/hadoop fs

[-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 ...]]
```

2）常用命令实操

（0）启动 Hadoop 集群（方便后续的测试）

```
[atguigu@hadoop102 hadoop-3.1.3]$ sbin/start-dfs.sh
[atguigu@hadoop103 hadoop-3.1.3]$ sbin/start-yarn.sh
```

（1）-help：输出这个命令参数

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -help rm
```

（2）-ls: 显示目录信息

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -ls /
```

（3）-mkdir：在 HDFS 上创建目录

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -mkdir -p
/sanguo/shuguo
```

（4）-moveFromLocal：从本地剪切粘贴到 HDFS

```
[atguigu@hadoop102 hadoop-3.1.3]$ touch kongming.txt
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs  -
moveFromLocal  ./kongming.txt  /sanguo/shuguo
```

（5）-appendToFile：追加一个文件到已经存在的文件末尾

```
[atguigu@hadoop102 hadoop-3.1.3]$ touch liubei.txt
[atguigu@hadoop102 hadoop-3.1.3]$ vi liubei.txt
输入
san gu mao lu
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -appendToFile
liubei.txt /sanguo/shuguo/kongming.txt
```

（6）-cat：显示文件内容

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -cat
/sanguo/shuguo/kongming.txt
```

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

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs  -chmod  666
/sanguo/shuguo/kongming.txt
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs  -chown
atguigu:atguigu   /sanguo/shuguo/kongming.txt
```

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

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -copyFromLocal
README.txt /
```

（9）-copyToLocal：从 HDFS 拷贝到本地

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -copyToLocal
/sanguo/shuguo/kongming.txt ./
```

（10）-cp ：从 HDFS 的一个路径拷贝到 HDFS 的另一个路径

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -cp
/sanguo/shuguo/kongming.txt /zhuge.txt
```

（11）-mv：在 HDFS 目录中移动文件

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -mv /zhuge.txt
/sanguo/shuguo/
```

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

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -get
/sanguo/shuguo/kongming.txt ./
```

（13）-getmerge：合并下载多个文件，比如 HDFS 的目录 /user/atguigu/test 下有多个文件:log.1, log.2,log.3,...

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -getmerge
/user/atguigu/test/* ./zaiyiqi.txt
```

（14）-put：等同于 copyFromLocal

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -put ./zaiyiqi.txt
/user/atguigu/test/
```

（15）-tail：显示一个文件的末尾

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -tail
/sanguo/shuguo/kongming.txt
```

（16）-rm：删除文件或文件夹

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -rm
/user/atguigu/test/jinlian2.txt
```

（17）-rmdir：删除空目录

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -mkdir /test
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -rmdir /test
```

（18）-du 统计文件夹的大小信息

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -du -s -h
/user/atguigu/test
2.7 K  /user/atguigu/test

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -du  -h
/user/atguigu/test
1.3 K  /user/atguigu/test/README.txt
15     /user/atguigu/test/jinlian.txt
1.4 K  /user/atguigu/test/zaiyiqi.txt
```

（19）-setrep：设置 HDFS 中文件的副本数量

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -setrep 5
/README.txt
```

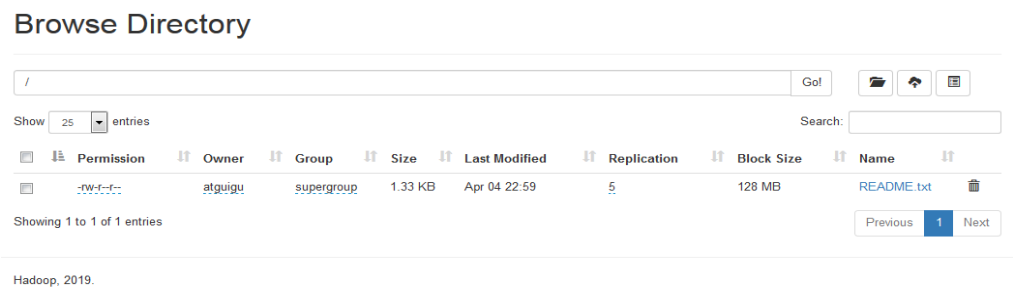


图 2-1 HDFS 副本数量

这里设置的副本数只是记录在 NameNode 的元数据中，是否真的会有这么多副本，还得看 DataNode 的数量。因为目前只有 3 台设备，最多也就 3 个副本，只有节点数的增加到至少 5 台时，副本数才能达到 5。

（20）解决 web 页面中操作没有权限问题，如图 2-2

图 2-2 web 页面操作报错

hadoop 默认情况下开启了权限检查，且默认使用 dir.who 作为 http 访问的静态用户,因此可通过关闭权限检查或者配置 http 访问的静态用户为 atguigu，二选一即可.

在 core-site.xml 中修改 http 访问的静态用户为 atguigu

```
<property>
       <name>hadoop.http.staticuser.user</name>
       <value>atguigu</value>
 </property>
```

在 hdfs-site.xml 中关闭权限检查

```
<property>
       <name>dfs.permissions.enabled</name>
       <value>false</value>
</property>
```

第 3 章 HDFS 客户端操作（开发重点）

3.1 HDFS 客户端环境准备

1）找到资料目录下的 Windows 依赖目录，打开：

图 3-1 windows 依赖

选择 Hadoop-3.1.0，拷贝到其他地方(比如 E:\hadoop\)。

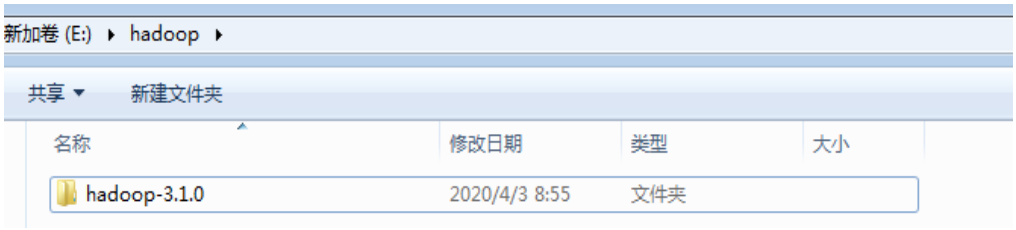


图 3-2 windows 依赖存放

2）配置 HADOOP_HOME 环境变量，如图 3-3 所示。

图 3-3　配置 HADOOP_HOME 环境变量

3）配置 Path 环境变量，如图 3-4 所示。然后重启电脑

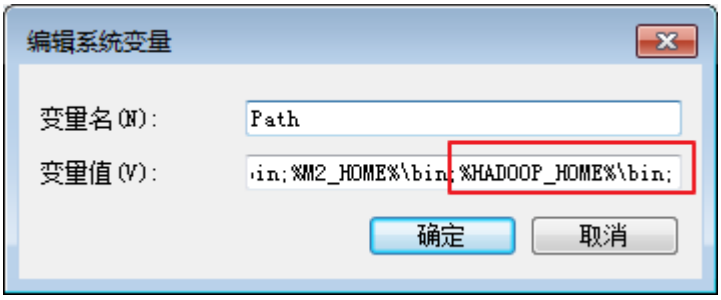


图 3-4　配置 Path 环境变量

4）创建一个 Maven 工程

5）导入相应的依赖坐标+日志添加

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-slf4j-impl</artifactId>
        <version>2.12.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-client</artifactId>
        <version>3.1.3</version>
    </dependency>
```

```
</dependencies>
```

在项目的 src/main /resources 目录下，新建一个文件，命名为“log4j2.xml”，在文件中填入

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="error" strict="true" name="XMLConfig">
    <Appenders>
        <!-- 类型名为 Console，名称为必须属性 -->
        <Appender type="Console" name="STDOUT">
            <!-- 布局为 PatternLayout 的方式，
            输出样式为[INFO] [2018-01-22
17:34:01][org.test.Console]I'm here -->
            <Layout type="PatternLayout"
                    pattern="[%p] [%d{yyyy-MM-dd
HH:mm:ss}][%c{10}]%m%n" />
        </Appender>

    </Appenders>

    <Loggers>
        <!-- 可加性为 false -->
        <Logger name="test" level="info" additivity="false">
            <AppenderRef ref="STDOUT" />
        </Logger>

        <!-- root loggerConfig 设置 -->
        <Root level="info">
            <AppenderRef ref="STDOUT" />
        </Root>
    </Loggers>

</Configuration>
```

6）创建包名：com.atguigu.hdfs

7）创建 HdfsClient 类

```
public class HdfsClient{
   @Test
    public    void    testHdfsClient()    throws    IOException,
InterruptedException {
        //1. 创建 HDFS 客户端对象,传入 uri， configuration , user
        FileSystem fileSystem =

FileSystem.get(URI.create("hdfs://hadoop102:9820"),          new
Configuration(), "atguigu");
        //2. 操作集群
        // 例如：在集群的/目录下创建 testHDFS 目录
        fileSystem.mkdirs(new Path("/testHDFS"));
        //3. 关闭资源
        fileSystem.close();
    }
}
```

8）执行程序

3.2 HDFS 的 API 操作

3.2.1 HDFS 文件上传（测试参数优先级）

1）编写源代码

```
@Test
public void testCopyFromLocalFile() throws IOException,
InterruptedException, URISyntaxException {

        // 1 获取文件系统
        Configuration configuration = new Configuration();
        // 设置副本数为 2 个
        configuration.set("dfs.replication", "2");
        FileSystem fs = FileSystem.get(new
URI("hdfs://hadoop102:9820"), configuration, "atguigu");

        // 2 上传文件
        fs.copyFromLocalFile(new Path("e:/banzhang.txt"), new
Path("/banzhang.txt"));

        // 3 关闭资源
        fs.close();

}
```

2）在项目的 resources 中新建 hdfs-site.xml 文件，并将如下内容拷贝进去，再次测试

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
</configuration>
```

3）参数优先级

参数优先级排序：（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）然后是服务器的自定义配置(xxx-site.xml)>（4）服务器的默认配置(xxx-default.xml)

3.2.2 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://hadoop102:9820"), configuration, "atguigu");
```

```
        // 2 执行下载操作
        // boolean delSrc 指是否将原文件删除
        // Path src 指要下载的文件路径
        // Path dst 指将文件下载到的路径
        // boolean useRawLocalFileSystem 是否开启文件校验
        fs.copyToLocalFile(false, new Path("/banzhang.txt"), new
Path("e:/banhua.txt"), true);

        // 3 关闭资源
        fs.close();
}
```

3.2.3 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://hadoop102:9820"), configuration, "atguigu");

    // 2 执行删除
    fs.delete(new Path("/0213/"), true);

    // 3 关闭资源
    fs.close();
}
```

3.2.4 HDFS 文件名更改/移动

```
@Test
public      void      testRename()      throws      IOException,
InterruptedException, URISyntaxException{

    // 1 获取文件系统
    Configuration configuration = new Configuration();
    FileSystem          fs          =          FileSystem.get(new
URI("hdfs://hadoop102:9820"), configuration, "atguigu");

    // 2 修改文件名称
    fs.rename(new           Path("/banzhang.txt"),           new
Path("/banhua.txt"));

    // 3 关闭资源
    fs.close();
}
```

3.2.5 HDFS 文件详情查看

查看文件名称、权限、长度、块信息

```
@Test
public     void     testListFiles()     throws     IOException,
InterruptedException, URISyntaxException{
```

```
    // 1 获取文件系统
    Configuration configuration = new Configuration();
    FileSystem fs = FileSystem.get(new
URI("hdfs://hadoop102:9820"), configuration, "atguigu");

    // 2 获取文件详情
    RemoteIterator<LocatedFileStatus> listFiles =
fs.listFiles(new Path("/"), true);

    while(listFiles.hasNext()){
        LocatedFileStatus status = listFiles.next();

        // 输出详情
        // 文件名称
        System.out.println(status.getPath().getName());
        // 长度
        System.out.println(status.getLen());
        // 权限
        System.out.println(status.getPermission());
        // 分组
        System.out.println(status.getGroup());

        // 获取存储的块信息
        BlockLocation[] blockLocations =
status.getBlockLocations();

        for (BlockLocation blockLocation : blockLocations) {

            // 获取块存储的主机节点
            String[] hosts = blockLocation.getHosts();

            for (String host : hosts) {
                System.out.println(host);
            }
        }

        System.out.println("-----------漂亮的分割线----------");
    }

    // 3 关闭资源
    fs.close();
}
```

3.2.6 HDFS 文件和文件夹判断

```
@Test
public void testListStatus() throws IOException,
InterruptedException, URISyntaxException{

    // 1 获取文件配置信息
    Configuration configuration = new Configuration();
    FileSystem fs = FileSystem.get(new
URI("hdfs://hadoop102:9820"), configuration, "atguigu");

    // 2 判断是文件还是文件夹
```

```
    FileStatus[] listStatus = fs.listStatus(new Path("/"));

    for (FileStatus fileStatus : listStatus) {

        // 如果是文件
        if (fileStatus.isFile()) {

    System.out.println("f:"+fileStatus.getPath().getName());
            }else {

    System.out.println("d:"+fileStatus.getPath().getName());
            }
        }

    // 3 关闭资源
    fs.close();
}
```

第 4 章 HDFS 的数据流（面试重点）

4.1 HDFS 写数据流程

4.1.1 剖析文件写入

HDFS 写数据流程，如图 4-1 所示。

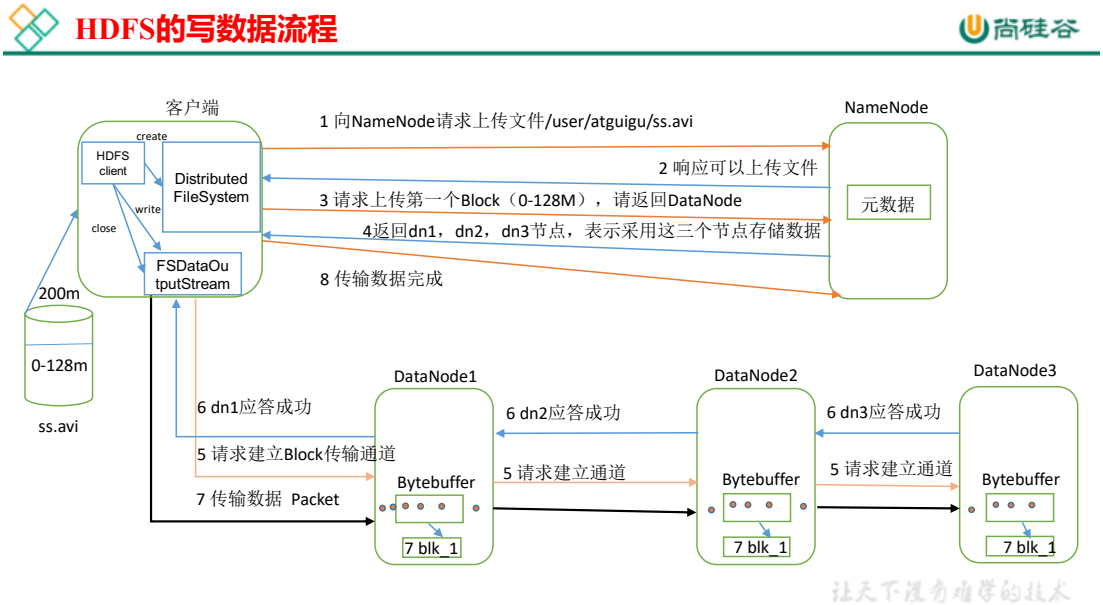


图 4-1 配置用户名称

1）客户端通过 Distributed FileSystem 模块向 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）客户端通过 FSDataOutputStream 模块请求 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）。

4.1.2 网络拓扑-节点距离计算

在 HDFS 写数据的过程中，NameNode 会选择距离待上传数据最近距离的 DataNode 接收数据。那么这个最近距离怎么计算呢？

节点距离：两个节点到达最近的共同祖先的距离总和。

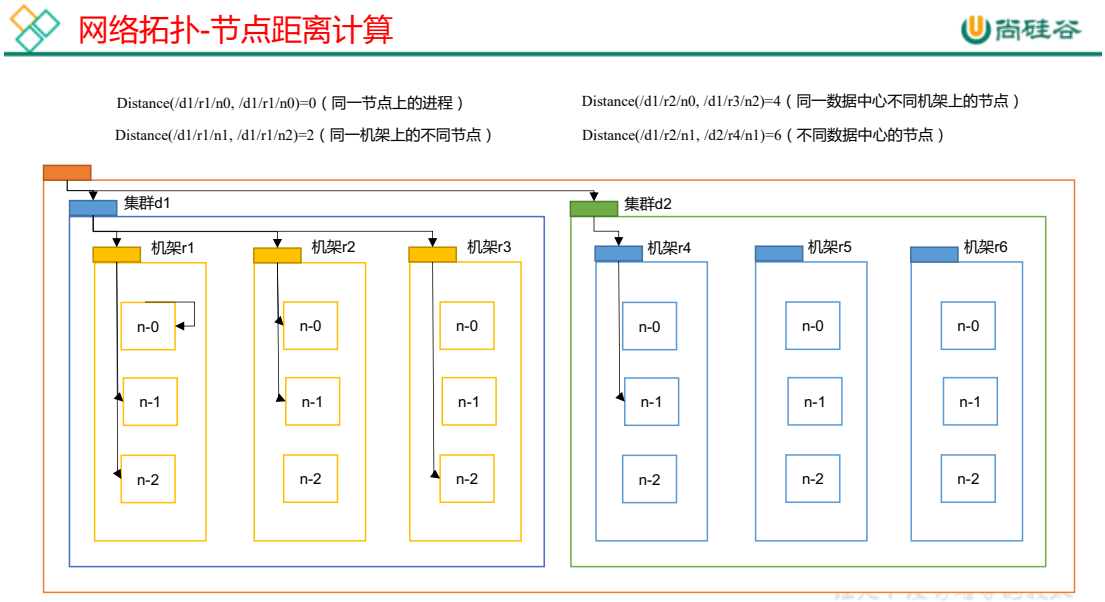


图 4-2　网络拓扑概念

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，如图 4-2 所示。

大家算一算每两个节点之间的距离，如图 4-3 所示。

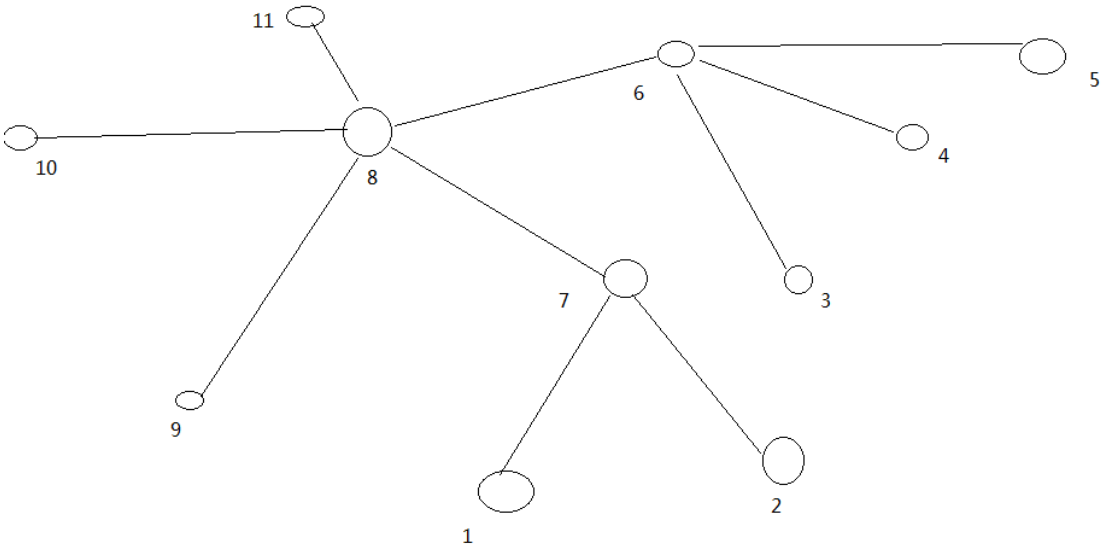


图 4-3　网络拓扑

4.1.3 机架感知（副本存储节点选择）

1）机架感知说明

http://hadoop.apache.org/docs/r3.1.3/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html#Data_Replication

```
For the common case, when the replication factor is three,
HDFS's placement policy is to put one replica on the local
machine if the writer is on a datanode, otherwise on a random
datanode, another replica on a node in a different (remote)
rack, and the last on a different node in the same remote
rack.
```

2）Hadoop3. 1. 3 副本节点选择

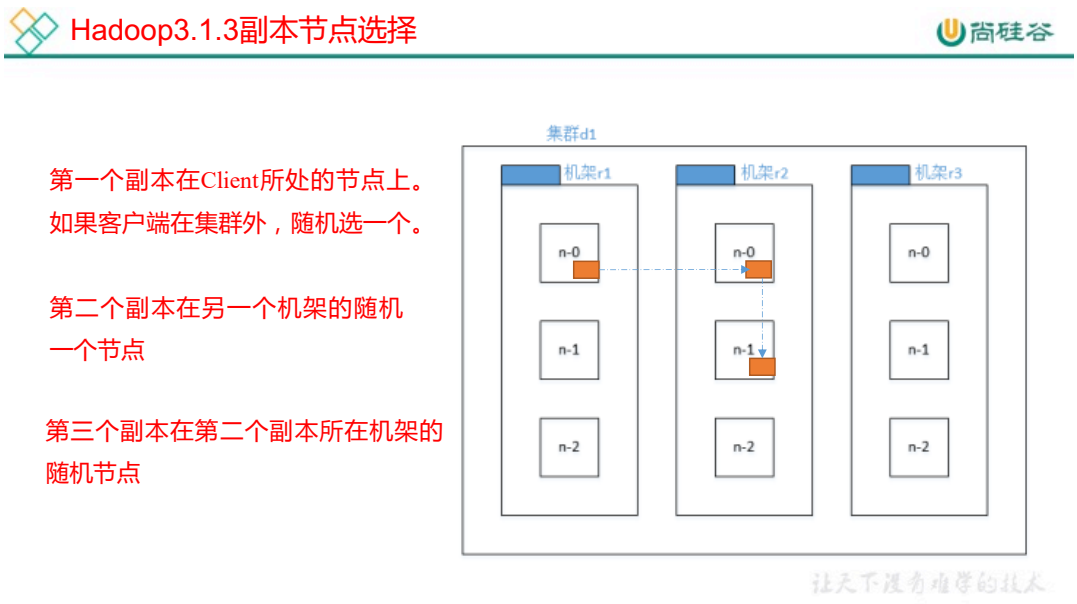


图 4-4　副本节点选择

4.2 HDFS 读数据流程

HDFS 的读数据流程，如图 4-5 所示。

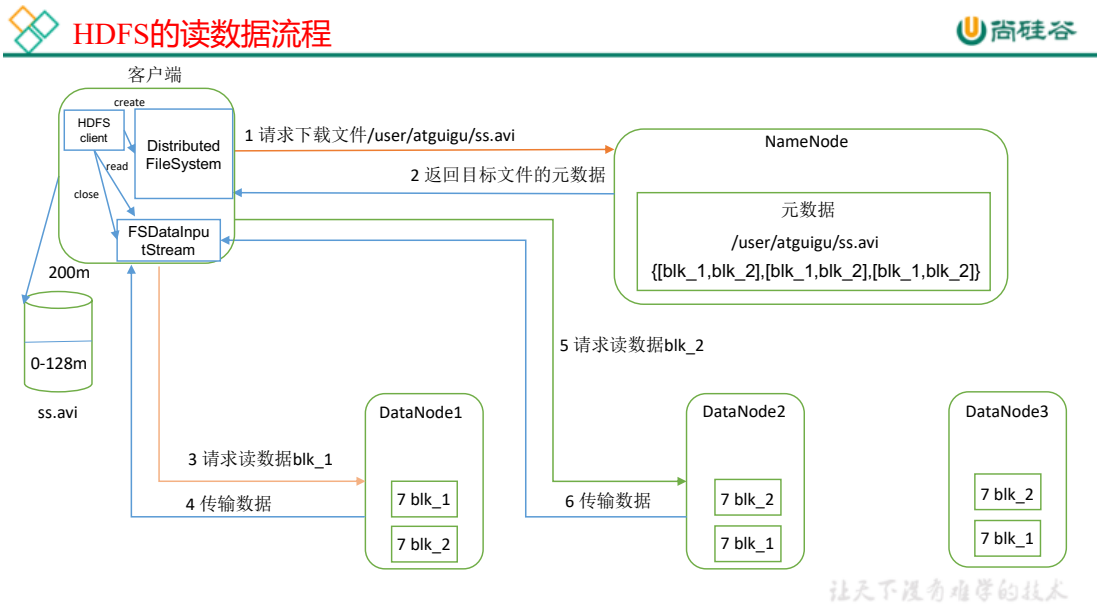


图 4-5　HDFS 读数据流程

1）客户端通过 Distributed FileSystem 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输入流，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。

第 5 章　NameNode 和 SecondaryNameNode（面试开发重点）

5.1 NN 和 2NN 工作机制

思考：NameNode 中的元数据是存储在哪里的？

首先，我们做个假设，如果存储在 NameNode 节点的磁盘中，因为经常需要进行随机访问，还有响应客户请求，必然是效率过低。因此，元数据需要存放在内存中。但如果只存在内存中，一旦断电，元数据丢失，整个集群就无法工作了。因此产生在磁盘中备份元数据的 FsImage。

这样又会带来新的问题，当在内存中的元数据更新时，如果同时更新 FsImage，就会导致效率过低，但如果不更新，就会发生一致性问题，一旦 NameNode 节点断电，就会产生数

据丢失。因此，引入 Edits 文件(只进行追加操作，效率很高)。每当元数据有更新或者添加元数据时，修改内存中的元数据并追加到 Edits 中。这样，一旦 NameNode 节点断电，可以通过 FsImage 和 Edits 的合并，合成元数据。

但是，如果长时间添加数据到 Edits 中，会导致该文件数据过大，效率降低，而且一旦断电，恢复元数据需要的时间过长。因此，需要定期进行 FsImage 和 Edits 的合并，如果这个操作由 NameNode 节点完成，又会效率过低。因此，引入一个新的节点 SecondaryNamenode，专门用于 FsImage 和 Edits 的合并。

NN 和 2NN 工作机制，如图 5-1 所示。

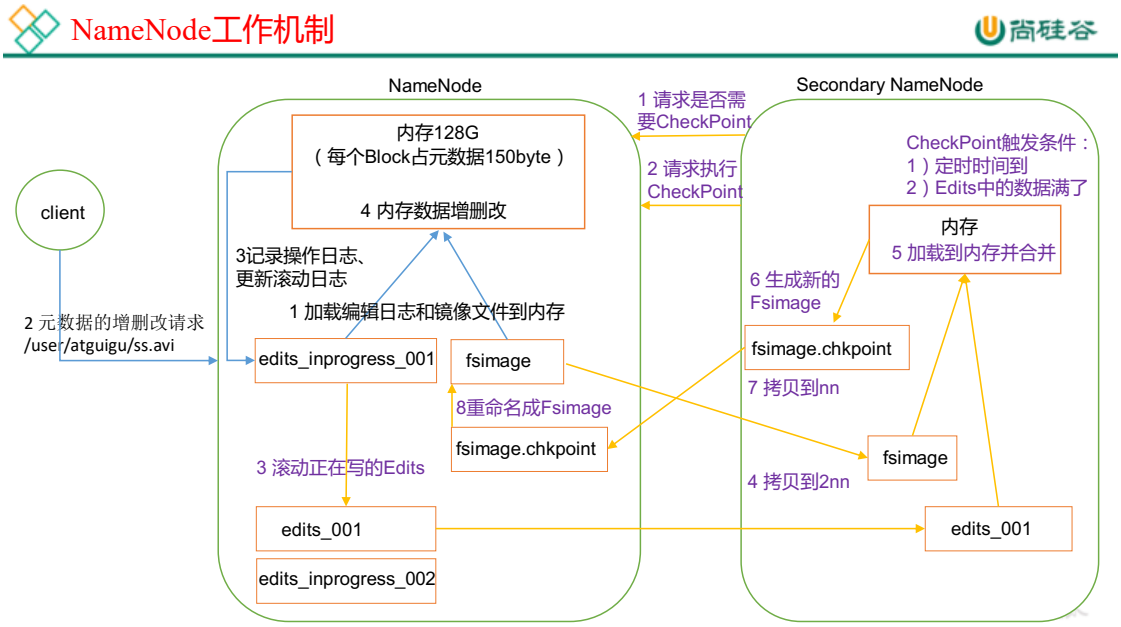


图 5-1　NN 和 2NN 工作机制

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 在内存中对元数据进行增删改。

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。

NN 和 2NN 工作机制详解：

Fsimage：NameNode 内存中元数据序列化后形成的文件。

Edits：记录客户端更新元数据信息的每一步操作（可通过 Edits 运算出元数据）。

NameNode 启动时，先滚动 Edits 并生成一个空的 edits.inprogress，然后加载 Edits 和 Fsimage 到内存中，此时 NameNode 内存就持有最新的元数据信息。Client 开始对 NameNode 发送元数据的增删改的请求，这些请求的操作首先会被记录到 edits.inprogress 中（查询元数据的操作不会被记录在 Edits 中，因为查询操作不会更改元数据信息），如果此时 NameNode 挂掉，重启后会从 Edits 中读取元数据的信息。然后，NameNode 会在内存中执行元数据的增删改的操作。

由于 Edits 中记录的操作会越来越多，Edits 文件会越来越大，导致 NameNode 在启动加载 Edits 时会很慢，所以需要对 Edits 和 Fsimage 进行合并（所谓合并，就是将 Edits 和 Fsimage 加载到内存中，照着 Edits 中的操作一步步执行，最终形成新的 Fsimage）。

SecondaryNameNode 的作用就是帮助 NameNode 进行 Edits 和 Fsimage 的合并工作。

SecondaryNameNode 首先会询问 NameNode 是否需要 CheckPoint（触发 CheckPoint 需要满足两个条件中的任意一个，定时时间到和 Edits 中数据写满了）。直接带回 NameNode 是否检查结果。SecondaryNameNode 执行 CheckPoint 操作，首先会让 NameNode 滚动 Edits 并生成一个空的 edits.inprogress，滚动 Edits 的目的是给 Edits 打个标记，以后所有新的操作都写入 edits.inprogress，其他未合并的 Edits 和 Fsimage 会拷贝到 SecondaryNameNode 的本地，然后将拷贝的 Edits 和 Fsimage 加载到内存中进行合并，生成 fsimage.chkpoint，然后将 fsimage.chkpoint 拷贝给 NameNode，重命名为 Fsimage 后替换掉原来的 Fsimage。NameNode 在启动时就只需要加载之前未合并的 Edits 和 Fsimage 即可，因为合并过的 Edits 中的元数据信息已经被记录在 Fsimage 中。

5.2 Fsimage 和 Edits 解析

1）概念

Fsimage和Edits概念

NameNode被格式化之后，将在/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inode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，加载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。

图 5-2 Fsimage 和 Edits

2） oiv 查看 Fsimage 文件

（1）查看 oiv 和 oev 命令

```
[atguigu@hadoop102 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

```
[atguigu@hadoop102 current]$ pwd
/opt/module/hadoop-3.1.3/data/tmp/dfs/name/current

[atguigu@hadoop102 current]$ hdfs oiv -p XML -i
fsimage_0000000000000000025 -o /opt/module/hadoop-
3.1.3/fsimage.xml

[atguigu@hadoop102 current]$ cat /opt/module/hadoop-
3.1.3/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。

```
<inode>
    <id>16386</id>
    <type>DIRECTORY</type>
    <name>user</name>
    <mtime>1512722284477</mtime>
    <permission>atguigu:supergroup:rwxr-xr-x</permission>
    <nsquota>-1</nsquota>
```

```
    <dsquota>-1</dsquota>
</inode>
<inode>
    <id>16387</id>
    <type>DIRECTORY</type>
    <name>atguigu</name>
    <mtime>1512790549080</mtime>
    <permission>atguigu:supergroup:rwxr-xr-x</permission>
    <nsquota>-1</nsquota>
    <dsquota>-1</dsquota>
</inode>
<inode>
    <id>16389</id>
    <type>FILE</type>
    <name>wc.input</name>
    <replication>3</replication>
    <mtime>1512722322219</mtime>
    <atime>1512722321610</atime>
    <perferredBlockSize>134217728</perferredBlockSize>
    <permission>atguigu:supergroup:rw-r--r--</permission>
    <blocks>
        <block>
            <id>1073741825</id>
            <genstamp>1001</genstamp>
            <numBytes>59</numBytes>
        </block>
    </blocks>
</inode >
```

思考：可以看出，Fsimage 中没有记录块所对应 DataNode，为什么？

在集群启动后，要求 DataNode 上报数据块信息，并间隔一段时间后再次上报。

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

```
[atguigu@hadoop102 current]$ hdfs oev -p XML -i
edits_0000000000000000012-0000000000000000013 -o
/opt/module/hadoop-3.1.3/edits.xml

[atguigu@hadoop102 current]$ cat /opt/module/hadoop-
3.1.3/edits.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。显示结果如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<EDITS>
    <EDITS_VERSION>-63</EDITS_VERSION>
    <RECORD>
        <OPCODE>OP_START_LOG_SEGMENT</OPCODE>
        <DATA>
            <TXID>129</TXID>
        </DATA>
    </RECORD>
```

```
    <RECORD>
        <OPCODE>OP_ADD</OPCODE>
        <DATA>
            <TXID>130</TXID>
            <LENGTH>0</LENGTH>
            <INODEID>16407</INODEID>
            <PATH>/hello7.txt</PATH>
            <REPLICATION>2</REPLICATION>
            <MTIME>1512943607866</MTIME>
            <ATIME>1512943607866</ATIME>
            <BLOCKSIZE>134217728</BLOCKSIZE>
            <CLIENT_NAME>DFSClient_NONMAPREDUCE_-
1544295051_1</CLIENT_NAME>
            <CLIENT_MACHINE>192.168.1.5</CLIENT_MACHINE>
            <OVERWRITE>true</OVERWRITE>
            <PERMISSION_STATUS>
                <USERNAME>atguigu</USERNAME>
                <GROUPNAME>supergroup</GROUPNAME>
                <MODE>420</MODE>
            </PERMISSION_STATUS>
            <RPC_CLIENTID>908eafd4-9aec-4288-96f1-
e8011d181561</RPC_CLIENTID>
            <RPC_CALLID>0</RPC_CALLID>
        </DATA>
    </RECORD>
    <RECORD>
        <OPCODE>OP_ALLOCATE_BLOCK_ID</OPCODE>
        <DATA>
            <TXID>131</TXID>
            <BLOCK_ID>1073741839</BLOCK_ID>
        </DATA>
    </RECORD>
    <RECORD>
        <OPCODE>OP_SET_GENSTAMP_V2</OPCODE>
        <DATA>
            <TXID>132</TXID>
            <GENSTAMPV2>1016</GENSTAMPV2>
        </DATA>
    </RECORD>
    <RECORD>
        <OPCODE>OP_ADD_BLOCK</OPCODE>
        <DATA>
            <TXID>133</TXID>
            <PATH>/hello7.txt</PATH>
            <BLOCK>
                <BLOCK_ID>1073741839</BLOCK_ID>
                <NUM_BYTES>0</NUM_BYTES>
                <GENSTAMP>1016</GENSTAMP>
            </BLOCK>
            <RPC_CLIENTID></RPC_CLIENTID>
            <RPC_CALLID>-2</RPC_CALLID>
        </DATA>
    </RECORD>
    <RECORD>
        <OPCODE>OP_CLOSE</OPCODE>
        <DATA>
            <TXID>134</TXID>
            <LENGTH>0</LENGTH>
```

```
            <INODEID>0</INODEID>
            <PATH>/hello7.txt</PATH>
            <REPLICATION>2</REPLICATION>
            <MTIME>1512943608761</MTIME>
            <ATIME>1512943607866</ATIME>
            <BLOCKSIZE>134217728</BLOCKSIZE>
            <CLIENT_NAME></CLIENT_NAME>
            <CLIENT_MACHINE></CLIENT_MACHINE>
            <OVERWRITE>false</OVERWRITE>
            <BLOCK>
                <BLOCK_ID>1073741839</BLOCK_ID>
                <NUM_BYTES>25</NUM_BYTES>
                <GENSTAMP>1016</GENSTAMP>
            </BLOCK>
            <PERMISSION_STATUS>
                <USERNAME>atguigu</USERNAME>
                <GROUPNAME>supergroup</GROUPNAME>
                <MODE>420</MODE>
            </PERMISSION_STATUS>
        </DATA>
    </RECORD>
</EDITS >
```

思考：NameNode 如何确定下次开机启动的时候合并哪些 Edits？

5.3 CheckPoint 时间设置

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，3 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 >
```

5.4 NameNode 故障处理（扩展）

NameNode 故障后，可以采用如下两种方法恢复数据。

1）方法一：将 SecondaryNameNode 中数据拷贝到 NameNode 存储数据的目录

（1）kill -9 NameNode 进程

（2）删除NameNode存储的数据(/opt/module/hadoop-3.1.3/data/tmp/dfs/name)

```
[atguigu@hadoop102 hadoop-3.1.3]$ rm -rf /opt/module/hadoop-
3.1.3/data/tmp/dfs/name/*
```

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

```
[atguigu@hadoop102 dfs]$ scp -r
atguigu@hadoop104:/opt/module/hadoop-
3.1.3/data/tmp/dfs/namesecondary/* ./name/
```

（4）重新启动 NameNode

```
[atguigu@hadoop102 hadoop-3.1.3]$ hdfs --daemon start namenode
```

2） 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）修改 hdfs-site.xml 中的

```
<property>
  <name>dfs.namenode.checkpoint.period</name>
  <value>120</value>
</property>

<property>
  <name>dfs.namenode.name.dir</name>
  <value>/opt/module/hadoop-3.1.3/data/tmp/dfs/name</value>
</property>
```

（2）kill -9 NameNode 进程

(3)删除 NameNode 存储的数据(/opt/module/hadoop-3.1.3/data/tmp/dfs/name)

```
[atguigu@hadoop102 hadoop-3.1.3]$ rm -rf /opt/module/hadoop-
3.1.3/data/tmp/dfs/name/*
```

（4）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，

并删除 in_use.lock 文件

```
[atguigu@hadoop102 dfs]$ scp -r
atguigu@hadoop104:/opt/module/hadoop-
3.1.3/data/tmp/dfs/namesecondary ./

[atguigu@hadoop102 namesecondary]$ rm -rf in_use.lock

[atguigu@hadoop102 dfs]$ pwd
/opt/module/hadoop-3.1.3/data/tmp/dfs

[atguigu@hadoop102 dfs]$ ls
data  name  namesecondary
```

（5）导入检查点数据（等待一会 ctrl+c 结束掉）

```
[atguigu@hadoop102 hadoop-3.1.3]$ bin/hdfs namenode -
importCheckpoint
```

（6）启动 NameNode

```
[atguigu@hadoop102 hadoop-3.1.3]$ hdfs --daemon start namenode
```

5.5 集群安全模式

1）概述

集群安全模式

1、NameNode启动

NameNode启动时，首先将镜像文件（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。这个过程期间，NameNode一直运行在安全模式，即NameNode的文件系统对于客户端来说是只读的。

2、DataNode启动

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。

3、安全模式退出判断

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。

图 5-3 集群安全模式

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

（1）查看当前模式

```
[atguigu@hadoop102 hadoop-3.1.3]$ hdfs dfsadmin -safemode get
Safe mode is OFF
```

（2）先进入安全模式

```
[atguigu@hadoop102 hadoop-3.1.3]$ bin/hdfs dfsadmin -safemode
enter
```

（3）创建并执行下面的脚本

在/opt/module/hadoop-3.1.3 路径上，编辑一个脚本 safemode.sh

```
[atguigu@hadoop102 hadoop-3.1.3]$ touch safemode.sh
[atguigu@hadoop102 hadoop-3.1.3]$ vim safemode.sh

#!/bin/bash
hdfs dfsadmin -safemode wait
hdfs dfs -put /opt/module/hadoop-3.1.3/README.txt /
```

```
[atguigu@hadoop102 hadoop-3.1.3]$ chmod 777 safemode.sh

[atguigu@hadoop102 hadoop-3.1.3]$ ./safemode.sh
```

（4）再打开一个窗口，执行

```
[atguigu@hadoop102 hadoop-3.1.3]$ bin/hdfs dfsadmin -safemode
leave
```

（5）观察

（a）再观察上一个窗口

Safe mode is OFF

（b）HDFS 集群上已经有上传的数据了。

5.6 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

（1）在 hdfs-site.xml 文件中修改如下内容

```
<property>
    <name>dfs.namenode.name.dir</name>
<value>file:///${hadoop.tmp.dir}/name1,file:///${hadoop.tmp.dir}/name2<
/value>
</property>
```

（2）停止集群，删除 data 和 logs 中所有数据。

```
[atguigu@hadoop102 hadoop-3.1.3]$ rm -rf data/ logs/
[atguigu@hadoop103 hadoop-3.1.3]$ rm -rf data/ logs/
[atguigu@hadoop104 hadoop-3.1.3]$ rm -rf data/ logs/
```

（3）格式化集群并启动。

```
[atguigu@hadoop102 hadoop-3.1.3]$ bin/hdfs namenode –format
[atguigu@hadoop102 hadoop-3.1.3]$ sbin/start-dfs.sh
```

（4）查看结果

```
[atguigu@hadoop102 dfs]$ ll
总用量 12
drwx------. 3 atguigu atguigu 4096 12 月 11 08:03 data
drwxrwxr-x. 3 atguigu atguigu 4096 12 月 11 08:03 name1
drwxrwxr-x. 3 atguigu atguigu 4096 12 月 11 08:03 name2
```

第 6 章 DataNode（面试开发重点）

6.1 DataNode 工作机制

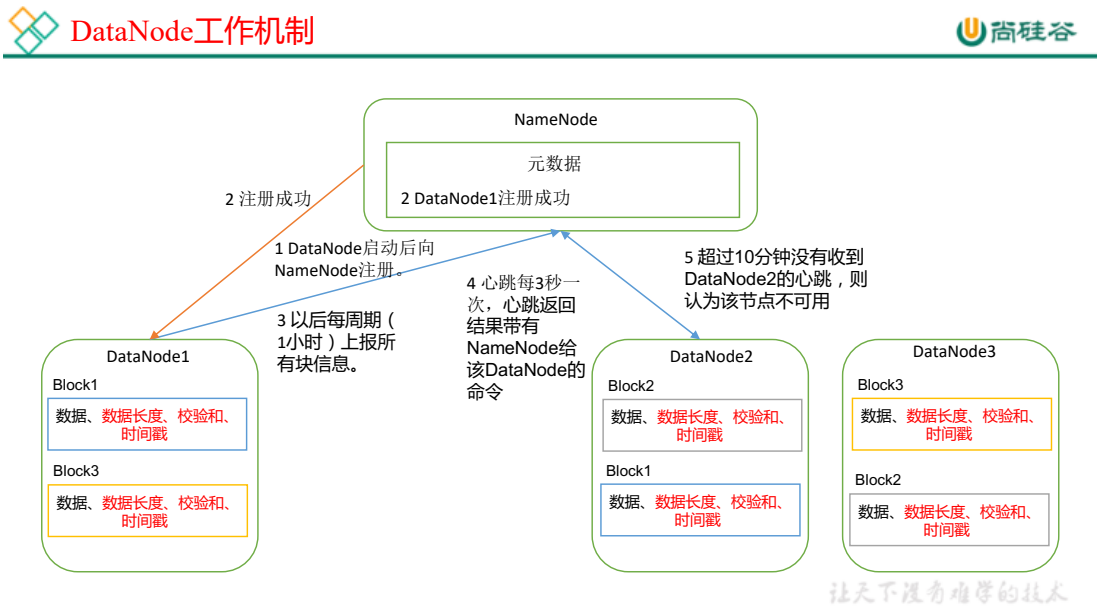


图 6-1 DataNode 工作机制

1）一个数据块在 DataNode 上以文件形式存储在磁盘上，包括两个文件，一个是数据本身，一个是元数据包括数据块的长度，块数据的校验和，以及时间戳。

2）DataNode 启动后向 NameNode 注册，通过后，周期性（1 小时）的向 NameNode 上报所有的块信息。

3）心跳是每 3 秒一次，心跳返回结果带有 NameNode 给该 DataNode 的命令如复制块数据到另一台机器，或删除某个数据块。如果超过 10 分钟没有收到某个 DataNode 的心跳，则认为该节点不可用。

4）集群运行中可以安全加入和退出一些机器。

6.2 数据完整性

思考：如果电脑磁盘里面存储的数据是控制高铁信号灯的红灯信号（1）和绿灯信号（0），但是存储该数据的磁盘坏了，一直显示是绿灯，是否很危险？同理 DataNode 节点上的数据损坏了，却没有发现，是否也很危险，那么如何解决呢？

如下是 DataNode 节点保证数据完整性的方法。

1）当 DataNode 读取 Block 的时候，它会计算 CheckSum。

2）如果计算后的 CheckSum，与 Block 创建时值不一样，说明 Block 已经损坏。

3）Client 读取其他 DataNode 上的 Block。

4）常见的校验算法 crc（32） md5（128） sha1（160）

5）DataNode 在其文件创建后周期验证 CheckSum，如图 6-2 所示。

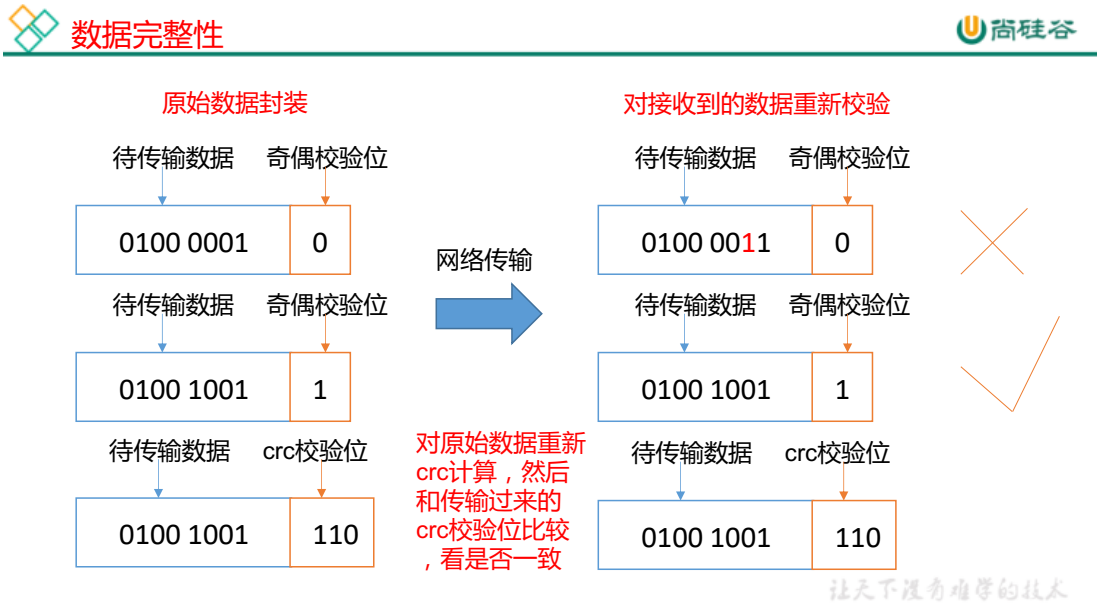


图 6-2 校验和

6.3 掉线时限参数设置

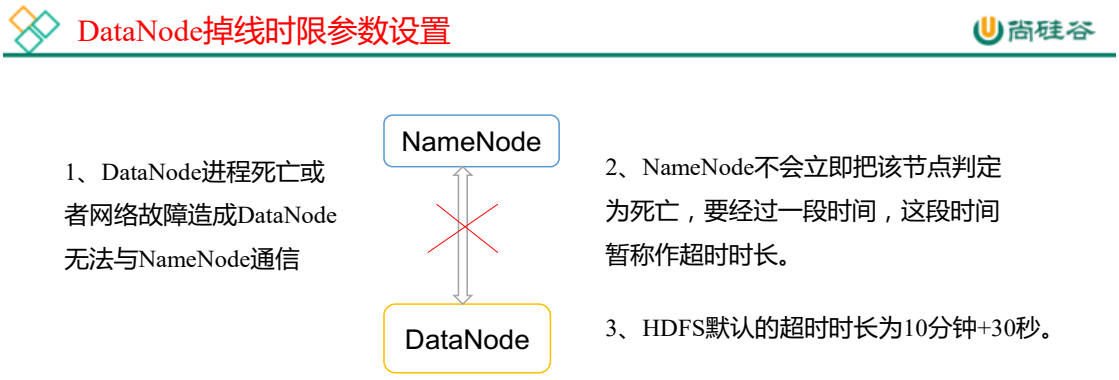


图 6-3 掉线时限

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。

```
<property>
    <name>dfs.namenode.heartbeat.recheck-interval</name>
```

```
    <value>300000</value>
</property>
<property>
    <name>dfs.heartbeat.interval</name>
    <value>3</value>
</property>
```

6.4 服役新数据节点

1）需求

随着公司业务的增长，数据量越来越大，原有的数据节点的容量已经不能满足存储数据的需求，需要在原有集群基础上动态添加新的数据节点。

2）环境准备

（1）在 hadoop104 主机上再克隆一台 hadoop105 主机

（2）修改 IP 地址和主机名称

（3）**删除原来 HDFS 文件系统留存的文件（/opt/module/hadoop-3.1.3/data 和 log）**

（4）source 一下配置文件

```
[atguigu@hadoop105 hadoop-3.1.3]$ source /etc/profile
```

3）服役新节点具体步骤

（1）直接启动 DataNode，即可关联到集群

```
[atguigu@hadoop105 hadoop-3.1.3]$ hdfs --daemon start datanode
[atguigu@hadoop105 hadoop-3.1.3]$yarn --daemon start
nodemanager
```

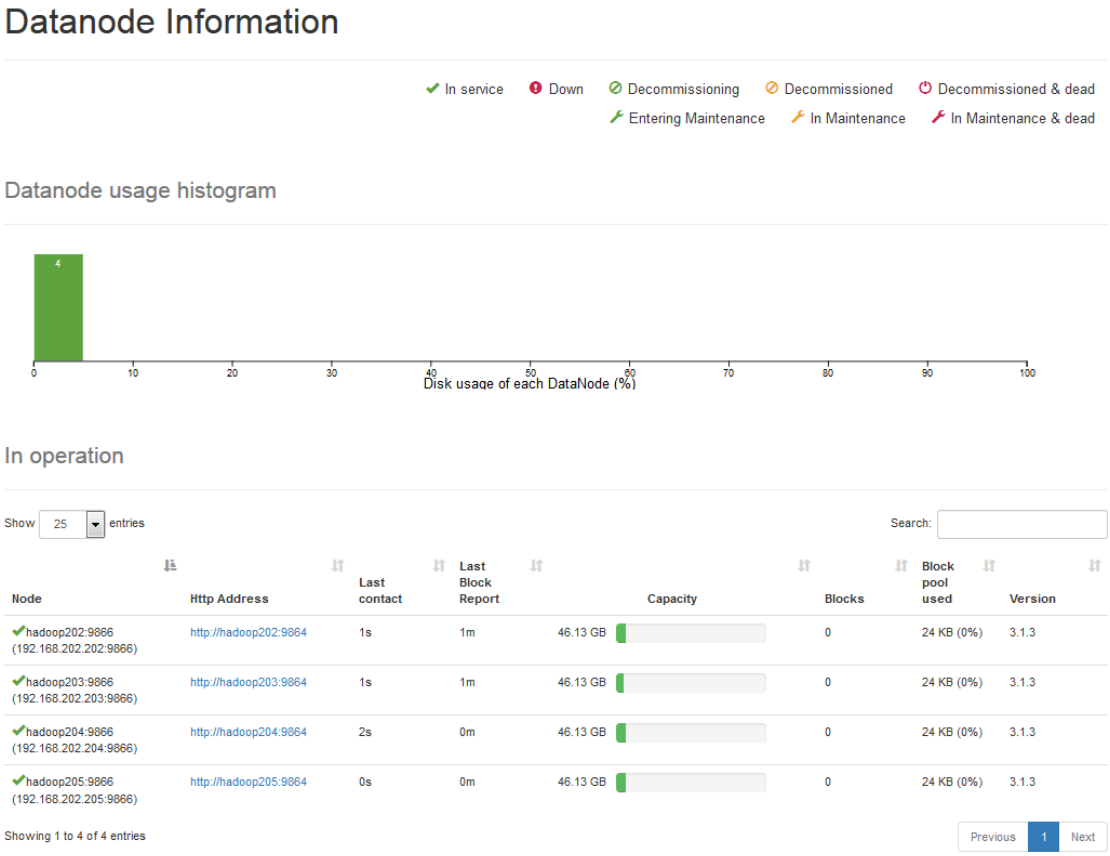


图 6-4 DataNode 节点

（2）如果数据不均衡，可以用命令实现集群的再平衡

```
[atguigu@hadoop102 hadoop-3.1.3]$ sbin/start-balancer.sh
```

6.5 退役旧数据节点

6.5.1 添加白名单 和 黑名单

添加到白名单的主机节点，都允许访问 NameNode，不在白名单的主机节点，都会被直接退出。

添加到黑名单的主机节点，不允许访问 NameNode，会在数据迁移后退出。

实际情况下，白名单用于确定允许访问 NameNode 的 DataNode 节点，内容配置一般与 workers 文件内容一致。 黑名单用于在集群运行过程中退役 DataNode 节点。

配置白名单和黑名单的具体步骤如下：

1）在 NameNode 的/opt/module/hadoop-3.1.3/etc/hadoop 目录下分别创建 whitelist 和 blacklist 文件

```
[atguigu@hadoop102 hadoop]$ pwd
/opt/module/hadoop-3.1.3/etc/hadoop
[atguigu@hadoop102 hadoop]$ touch whitelist
[atguigu@hadoop102 hadoop]$ touch blacklist
```

在 whitelist 中添加如下主机名称,假如集群正常工作的节点为 102 103 104 105

```
hadoop102
hadoop103
hadoop104
hadoop105
```

黑名单暂时为空。

2）在 NameNode 的 hdfs-site.xml 配置文件中增加 dfs.hosts 和 dfs.hosts.exclude 配置

```
<property>
<name>dfs.hosts</name>
<value>/opt/module/hadoop-3.1.3/etc/hadoop/whitelist</value>
</property>

<property>
<name>dfs.hosts.exclude</name>
<value>/opt/module/hadoop-3.1.3/etc/hadoop/blacklist</value>
</property>
```

3）配置文件分发

```
[atguigu@hadoop102 hadoop]$ xsync hdfs-site.xml
```

4）重新启动集群

```
[atguigu@hadoop102 hadoop-3.1.3]$ stop-dfs.sh
[atguigu@hadoop102 hadoop-3.1.3]$ start-dfs.sh
注意：因为 workers 中没有配置 105,需要单独在 105 启动 DN
```

5）web 端查看目前正常工作的 DN 节点

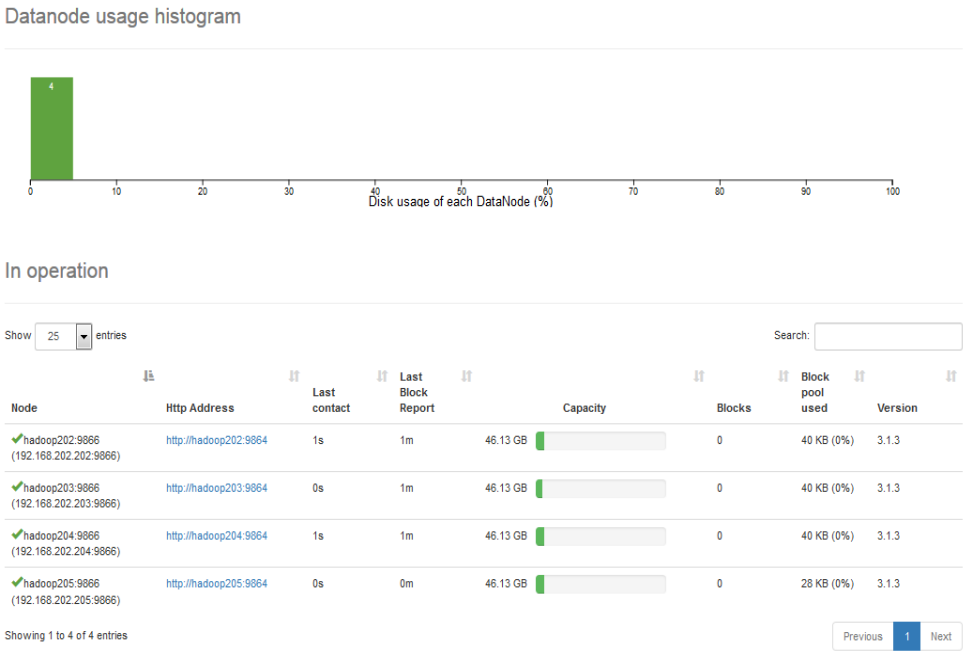


图 6-5 DataNode 节点

6.5.2 黑名单退役

1）准备使用黑名单退役 105

编辑 blacklist 文件，添加 105

```
[atguigu@hadoop102 hadoop] vim blacklist
hadoop105
```

2）刷新 NameNode

```
[atguigu@hadoop102 hadoop] hdfs dfsadmin -refreshNodes
```

3）在 web 端查看 DN 状态，105 正在退役中…进行数据的迁移

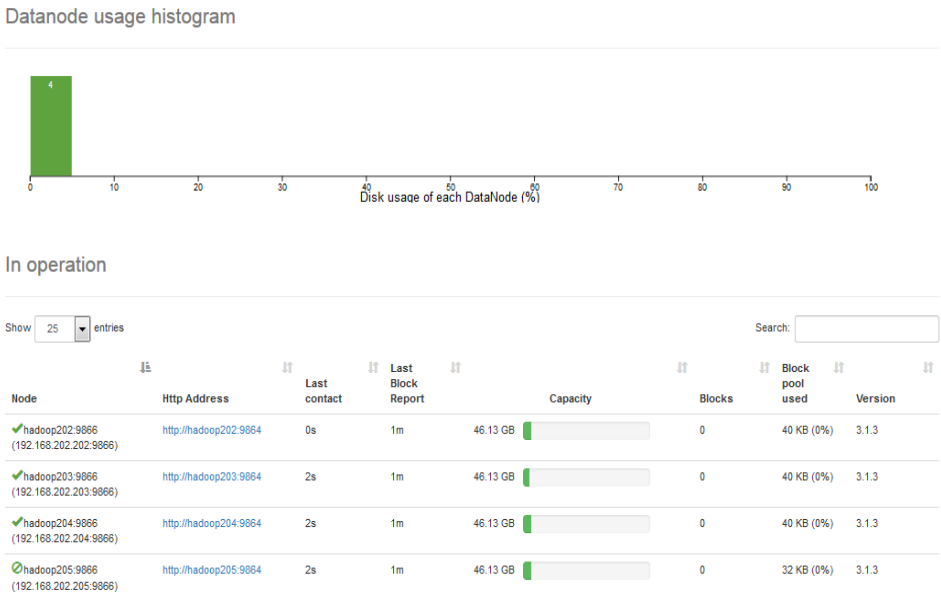


图 6-6 DataNode 退役中

4）如果 105 也启动的 NodeManager，也可以刷新 yarn 状态。【可选查看】

```
[atguigu@hadoop102 hadoop-3.1.3]$ yarn rmadmin -refreshNodes
```

6.5.3 白名单退役[不推荐]

白名单退役会直接将节点抛弃，没有迁移数据的过程，会造成数据丢失。

1）删除 blacklist 的中的内容，恢复 102 103 104 105 正常工作，如图

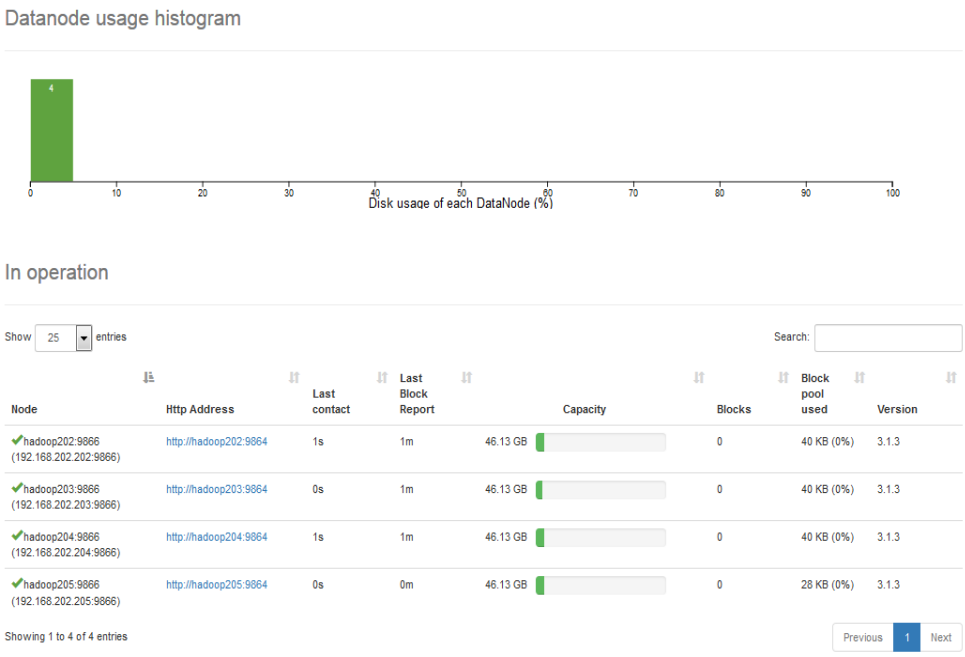


图 6-7 退役前 DataNode 节点

2）修改 whitelist，将 105 删除，保留 102 103 104

```
[atguigu@hadoop102 hadoop]$ vim whitelist
hadoop102
hadoop103
hadoop104
```

3）刷新 NameNode

```
[atguigu@hadoop102 hadoop-3.1.3]$ hdfs dfsadmin -refreshNodes
```

4）web 端查看，发现 105 节点直接从集群列表中丢弃

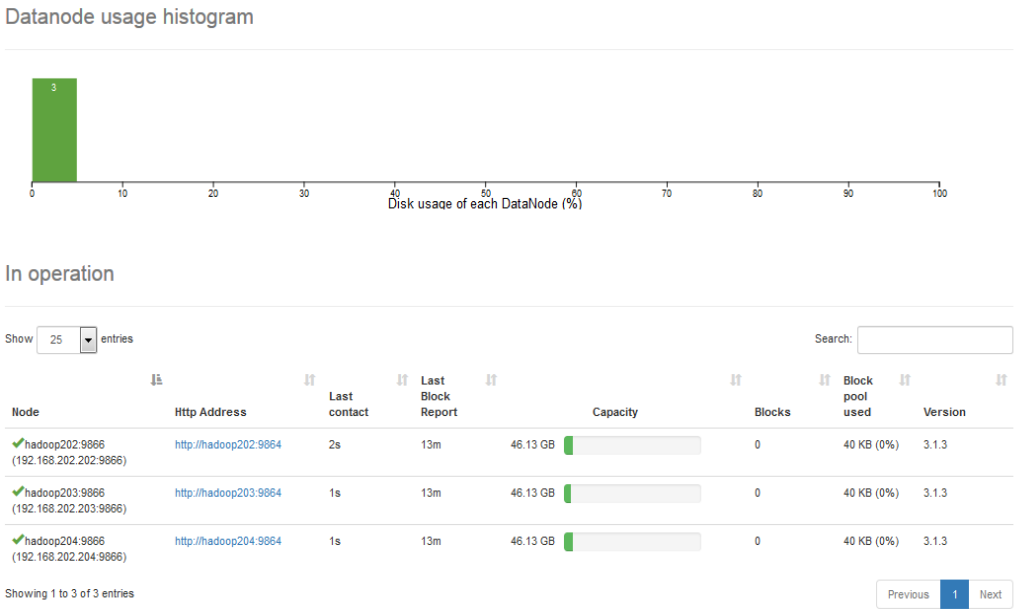


图 6-8 退役后 DataNode 节点

6.6 Datanode 多目录配置

1）DataNode 也可以配置成多个目录，每个目录存储的数据不一样。即：数据不是副本

2）具体配置如下

（1）在 hdfs-site.xml 中修改如下内容:

```
<property>
        <name>dfs.datanode.data.dir</name>
<value>file:///${hadoop.tmp.dir}/data1,file:///${hadoop.tmp.di
r}/data2</value>
</property>
```

（2）停止集群，删除 data 和 logs 中所有数据。

```
[atguigu@hadoop102 hadoop-3.1.3]$ rm -rf data/ logs/
[atguigu@hadoop103 hadoop-3.1.3]$ rm -rf data/ logs/
[atguigu@hadoop104 hadoop-3.1.3]$ rm -rf data/ logs/
```

（3）格式化集群并启动。

```
[atguigu@hadoop102 hadoop-3.1.3]$ bin/hdfs namenode -format
[atguigu@hadoop102 hadoop-3.1.3]$ sbin/start-dfs.sh
```

（4）查看结果

```
[atguigu@hadoop102 dfs]$ ll
总用量 12
drwx------. 3 atguigu atguigu 4096 4 月   4 14:22 data1
drwx------. 3 atguigu atguigu 4096 4 月   4 14:22 data2
drwxrwxr-x. 3 atguigu atguigu 4096 12 月 11 08:03 name1
```

```
drwxrwxr-x. 3 atguigu atguigu 4096 12 月 11 08:03 name2
```

第 7 章 小文件存档

小文件存档

1、HDFS存储小文件弊端

每个文件均按块存储，每个块的元数据存储在NameNode的内存中，因此HDFS存储小文件会非常低效。因为大量的小文件会耗尽NameNode中的大部分内存。但注意，存储小文件所需要的磁盘容量和数据块的大小无关。例如，一个1MB的文件设置为128MB的块存储，实际使用的是1MB的磁盘空间，而不是128MB。

2、解决存储小文件办法之一

HDFS存档文件或HAR文件，是一个更高效的文件存档工具，它将文件存入HDFS块，在减少NameNode内存使用的同时，允许对文件进行透明的访问。具体说来，HDFS存档文件对内还是一个一个独立文件，对NameNode而言却是一个整体，减少了NameNode的内存。

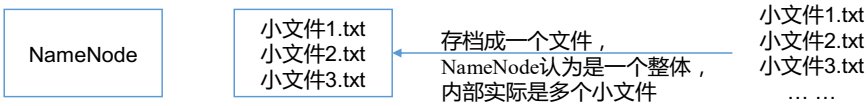


图 7-1 小文件存档

1）案例实操

（1）需要启动 YARN 进程

```
[atguigu@hadoop102 hadoop-3.1.3]$ start-yarn.sh
```

（2）归档文件

把/user/atguigu/input 目录里面的所有文件归档成一个叫 input.har 的归档文件，并把归档后文件存储到/user/atguigu/output 路径下。

```
[atguigu@hadoop102 hadoop-3.1.3]$ bin/hadoop archive -
archiveName input.har -p  /user/atguigu/input
/user/atguigu/output
```

（3）查看归档

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -lsr
/user/atguigu/output/input.har
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -lsr
har:///user/atguigu/output/input.har
```

（4）解归档文件

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -cp har:///
user/atguigu/output/input.har/*    /user/atguigu
```

第 8 章 回收站

开启回收站功能，可以将删除的文件在不超时的情况下，恢复原数据，起到防止误删除、备份等作用。

1）回收站参数设置及工作机制

回收站

一、开启回收站功能参数说明：

1、默认值fs.trash.interval=0，0表示禁用回收站;其他值表示设置文件的存活时间。

2、默认值fs.trash.checkpoint.interval=0，检查回收站的间隔时间。如果该值为0，则该值设置和fs.trash.interval的参数值相等。

3、要求fs.trash.checkpoint.interval<=fs.trash.interval。

二、回收站工作机制：

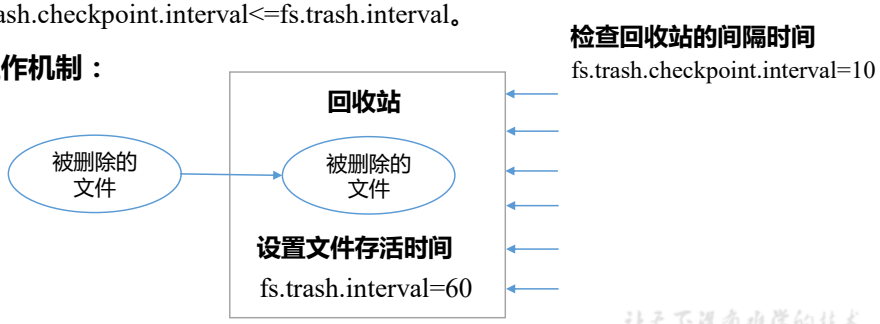


图 回收站

2）启用回收站

修改 core-site.xml，配置垃圾回收时间为 1 分钟。

```
<property>
  <name>fs.trash.interval</name>
<value>1</value>
</property>
```

3）查看回收站

回收站在集群中的路径：/user/atguigu/.Trash/….

4）修改访问垃圾回收站用户名称

进入垃圾回收站用户名称，默认是 dr.who，修改为 atguigu 用户

[core-site.xml]

```
<property>
  <name>hadoop.http.staticuser.user</name>
  <value>atguigu</value>
</property>
```

5）通过程序删除的文件不会经过回收站，需要调用 moveToTrash()才进入回收站

```
Trash trash = New Trash(conf);
trash.moveToTrash(path);
```

6）恢复回收站数据

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -mv
/user/atguigu/.Trash/Current/user/atguigu/input
/user/atguigu/input
```

7）清空回收站

```
[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -expunge
```

第 9 章 纠删码(擦除编码)机制[了解]

参考资料: https://issues.apache.org/jira/browse/HDFS-7285

HDFS 为擦除编码（EC）提供了支持，以更有效地存储数据。与默认三个副本机制相比， EC 策略可以节省约 50%左右的存储空间。
但不可忽略的是编解码的运算会消耗 CPU 资源。纠删码的编解码性能对其在 HDFS 中的应用起着至关重要的作用，如果不利用硬件方面的优化就很难得到理想的性能。英特尔的智能存储加速库（ISA-L）提供了对纠删码编解码的优化，极大的提升了其性能。

纠删码是 hadoop3.x 新加入的功能，之前的 hdfs 都是采用副本方式容错，默认情况下，一个文件有 3 个副本，可以容忍任意 2 个副本（datanode）不可用，这样提高了数据的可用性，但也带来了 2 倍的冗余开销。例如 3TB 的空间，只能存储 1TB 的有效数据。而纠删码则可以在同等可用性的情况下，节省更多的空间，以 RS-6-3-1024K 这种纠删码策略为例子，6 份原始数据，编码后生成 3 份校验数据，一共 9 份数据，只要最终有 6 份数据存在，就可以得到原始数据，它可以容忍任意 3 份数据不可用.

9.1 查看当前支持的纠删码策略

1）查看当前支持的纠删码策略

```
[atguigu@hadoop102 hadoop-3.1.3] hdfs ec -listPolicies

Erasure Coding Policies:
ErasureCodingPolicy=[Name=RS-10-4-1024k,
Schema=[ECSchema=[Codec=rs, numDataUnits=10,
numParityUnits=4]], CellSize=1048576, Id=5], State=DISABLED

ErasureCodingPolicy=[Name=RS-3-2-1024k,
Schema=[ECSchema=[Codec=rs, numDataUnits=3,
numParityUnits=2]], CellSize=1048576, Id=2], State=DISABLED

ErasureCodingPolicy=[Name=RS-6-3-1024k,
Schema=[ECSchema=[Codec=rs, numDataUnits=6,
numParityUnits=3]], CellSize=1048576, Id=1], State=ENABLED

ErasureCodingPolicy=[Name=RS-LEGACY-6-3-1024k,
Schema=[ECSchema=[Codec=rs-legacy, numDataUnits=6,
```

```
numParityUnits=3]], CellSize=1048576, Id=3], State=DISABLED

ErasureCodingPolicy=[Name=XOR-2-1-1024k,
Schema=[ECSchema=[Codec=xor, numDataUnits=2,
numParityUnits=1]], CellSize=1048576, Id=4], State=DISABLED
```

2）纠删码策略解释：

RS-10-4-1024k：使用 RS 编码，每 10 个数据单元（cell），生成 4 个校验单元，共 14 个单元，也就是说：这 14 个单元中，只要有任意的 10 个单元存在（不管是数据单元还是校验单元，只要总数=10），就可以得到原始数据。每个单元的大小是 1024k=1024*1024=1048576。

RS-3-2-1024k：使用 RS 编码，每 3 个数据单元，生成 2 个校验单元，共 5 个单元，也就是说：这 5 个单元中，只要有任意的 3 个单元存在（不管是数据单元还是校验单元，只要总数=3），就可以得到原始数据。每个单元的大小是 1024k=1024*1024=1048576。

RS-6-3-1024k：使用 RS 编码，每 6 个数据单元，生成 3 个校验单元，共 9 个单元，也就是说：这 9 个单元中，只要有任意的 6 个单元存在（不管是数据单元还是校验单元，只要总数=6），就可以得到原始数据。每个单元的大小是 1024k=1024*1024=1048576。

RS-LEGACY-6-3-1024k：策略和上面的 RS-6-3-1024k 一样，只是编码的算法用的是 rs-legacy。

XOR-2-1-1024k：使用 XOR 编码（速度比 RS 编码快），每 2 个数据单元，生成 1 个校验单元，共 3 个单元，也就是说：这 3 个单元中，只要有任意的 2 个单元存在（不管是数据单元还是校验单元，只要总数=2），就可以得到原始数据。每个单元的大小是 1024k=1024*1024=1048576。

9.2 设置纠删码策略

纠删码策略是与具体的路径（path）相关联的。也就是说，如果我们要使用纠删码，则要给一个具体的路径设置纠删码策略，后续，所有往此目录下存储的文件，都会执行此策略。

默认只开启对 RS-6-3-1024k 策略的支持，如要使用别的策略需要先启用.

1）纠删码操作相关的命令

```
[atguigu@hadoop202 hadoop-3.1.3]$ hdfs ec
Usage: bin/hdfs ec [COMMAND]
          [-listPolicies]
          [-addPolicies -policyFile <file>]
          [-getPolicy -path <path>]
          [-removePolicy -policy <policy>]
          [-setPolicy -path <path> [-policy <policy>] [-
replicate]]
          [-unsetPolicy -path <path>]
          [-listCodecs]
          [-enablePolicy -policy <policy>]
          [-disablePolicy -policy <policy>]
```

```
        [-help <command-name>].
```

2）开启对 RS-3-2-1024k 策略的支持

为啥不用默认的 RS-6-3-1024k 策略呢，理论情况下，该策略需要 9 台 DN 的支持，而 RS-3-2-1024k 策略需要 5 台 DN 的支持，所以，你们懂的！！！

```
[atguigu@hadoop202 hadoop-3.1.3]$  hdfs ec -enablePolicy  -
policy RS-3-2-1024k
Erasure coding policy RS-3-2-1024k is enabled
```

3）在 HDFS 创建目录，并设置擦除策略

```
[atguigu@hadoop202  hadoop-3.1.3]$  hdfs dfs -mkdir /input
[atguigu@hadoop202 hadoop-3.1.3]$ hdfs ec -setPolicy -path
/input -policy RS-3-2-1024k
[atguigu@hadoop202 hadoop-3.1.3]$ hdfs ec -getPolicy -path
/input
```

4）上传文件，并查看文件编码后的存储情况

```
[atguigu@hadoop202 hadoop-3.1.3]$ hdfs dfs -put README.txt
/input
[atguigu@hadoop202  hadoop-3.1.3]$ hdfs fsck /input/README.txt
-files -blocks -locations

Connecting to namenode via
http://hadoop202:9870/fsck?ugi=atguigu&files=1&blocks=1&locati
ons=1&path=%2Finput%2FREADME.txt
FSCK started by atguigu (auth:SIMPLE) from /192.168.202.202
for path /input/README.txt at Fri Apr 10 22:03:27 CST 2020
/input/README.txt 1366 bytes, erasure-coded: policy=RS-3-2-
1024k, 1 block(s):  OK
0. BP-1572777420-192.168.202.202-1586521347125:blk_-
9223372036854775760_1003 len=1366 Live_repl=3  [blk_-
9223372036854775760:DatanodeInfoWithStorage[192.168.202.202:98
66,DS-0d2d459f-331a-44b9-9e05-c09f971bfcfc,DISK], blk_-
9223372036854775757:DatanodeInfoWithStorage[192.168.202.206:98
66,DS-ca91c898-1e07-46de-a0eb-633fdab64bca,DISK], blk_-
9223372036854775756:DatanodeInfoWithStorage[192.168.202.204:98
66,DS-6ea8118c-fea4-4a69-bb96-7d5a1ffc22d1,DISK]]
```