尚硅谷大数据技术之 Hadoop（HDFS）

(作者：尚硅谷大数据研发部)

版本：V2.0

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产生背景

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定义

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。

1.2 HDFS 优缺点

HDFS优缺点

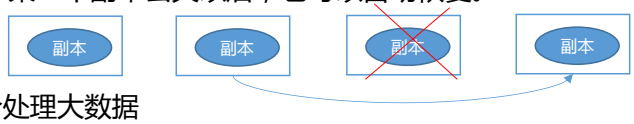
尚硅谷

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。

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。

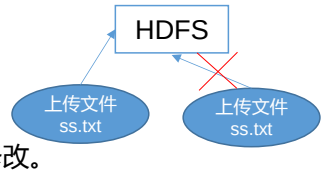
（1）存储大量小文件的话，它会占用NameNode大量的内存来存储文件目录和块信息。这样是不可取的，因为NameNode的内存总是有限的；

（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。

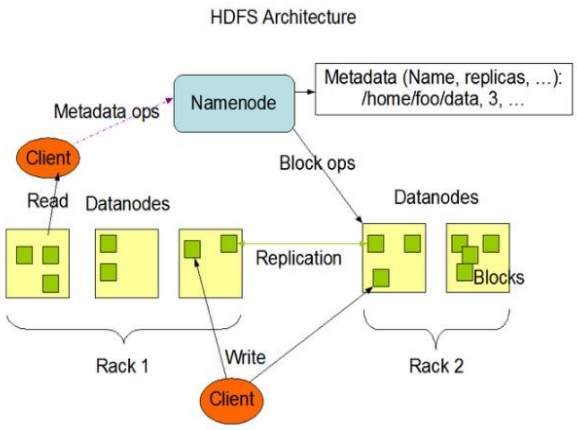
3）不支持并发写入、文件随机修改。

（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。



1.3 HDFS 组成架构



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。

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；

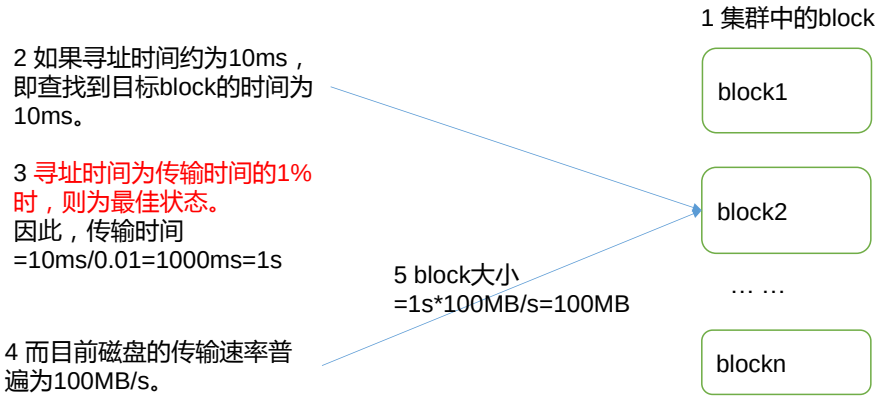
4）Secondary NameNode：并非NameNode的热备。当NameNode挂掉的时候，它并不能马上替换NameNode并提供服务。

（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。

1.4 HDFS 文件块大小（面试重点）

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。



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。

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

dfs 是 fs 的实现类。

2. 命令大全

```
[atguigu@hadoop102 hadoop-2.7.2]$ 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 ...]]
```

3．常用命令实操

（0）启动 Hadoop 集群（方便后续的测试）

```
[atguigu@hadoop102 hadoop-2.7.2]$ sbin/start-dfs.sh
[atguigu@hadoop103 hadoop-2.7.2]$ sbin/start-yarn.sh
```

（1）-help：输出这个命令参数

```
[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -help rm
```

（2）-ls: 显示目录信息

```
[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -ls /
```

（3）-mkdir：在 HDFS 上创建目录

```
[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -mkdir -p
/sanguo/shuguo
```

（4）-moveFromLocal：从本地剪切粘贴到 HDFS

```
[atguigu@hadoop102 hadoop-2.7.2]$ touch kongming.txt
[atguigu@hadoop102 hadoop-2.7.2]$ 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-2.7.2]$ touch liubei.txt
[atguigu@hadoop102 hadoop-2.7.2]$ vi liubei.txt
输入
san gu mao lu
[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -appendToFile
liubei.txt /sanguo/shuguo/kongming.txt
```

（6）-cat：显示文件内容

```
[atguigu@hadoop102 hadoop-2.7.2]$ 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-2.7.2]$ hadoop fs -chmod 666
/sanguo/shuguo/kongming.txt
[atguigu@hadoop102 hadoop-2.7.2]$ 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-2.7.2]$ hadoop fs -copyFromLocal
README.txt /
```

（9）-copyToLocal：从 HDFS 拷贝到本地

```
[atguigu@hadoop102 hadoop-2.7.2]$ 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-2.7.2]$ hadoop fs -cp
/sanguo/shuguo/kongming.txt /zhuge.txt
```

（11）-mv：在 HDFS 目录中移动文件

```
[atguigu@hadoop102 hadoop-2.7.2]$ 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-2.7.2]$ 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-2.7.2]$ hadoop fs -getmerge
/user/atguigu/test/* ./zaiyiqi.txt
```

（14）-put：等同于 copyFromLocal

```
[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -put ./zaiyiqi.txt
/user/atguigu/test/
```

（15）-tail：显示一个文件的末尾

```
[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -tail
/sanguo/shuguo/kongming.txt
```

（16）-rm：删除文件或文件夹

```
[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -rm
/user/atguigu/test/jinlian2.txt
```

（17）-rmdir：删除空目录

```
[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -mkdir /test
[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -rmdir /test
```

（18）-du 统计文件夹的大小信息

```
[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -du -s -h
/user/atguigu/test
2.7 K  /user/atguigu/test

[atguigu@hadoop102 hadoop-2.7.2]$ 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-2.7.2]$ hadoop fs -setrep 10
/sanguo/shuguo/kongming.txt
```

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	atguigu	supergroup	0 B	2018/1/23 下午4:11:10	10	128 MB	kongming.txt

图 3-3　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 10 台时，副本数才能达到 10。

第 3 章 HDFS 客户端操作（开发重点）

3.1 HDFS 客户端环境准备

1．根据自己电脑的操作系统拷贝对应的编译后的 hadoop jar 包到非中文路径（例如：D:\Develop\hadoop-2.7.2），如图 3-4 所示。

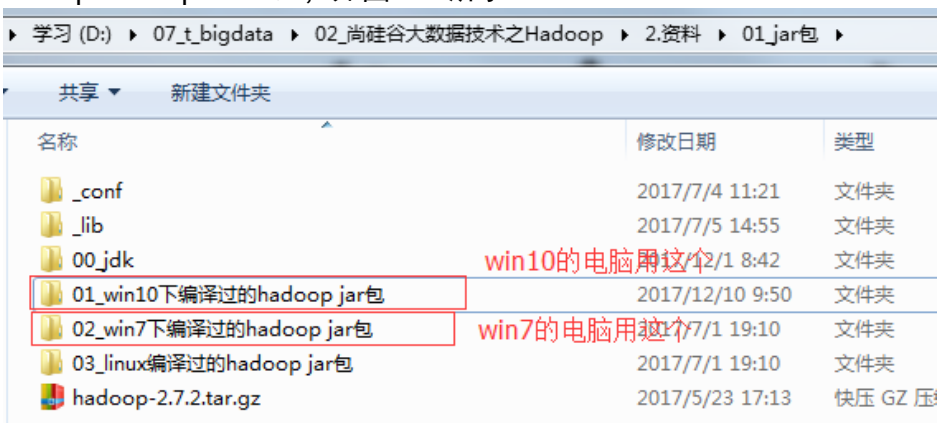


图 3-4　编译后的 hadoop jar 包

2．配置 HADOOP_HOME 环境变量，如图 3-5 所示。

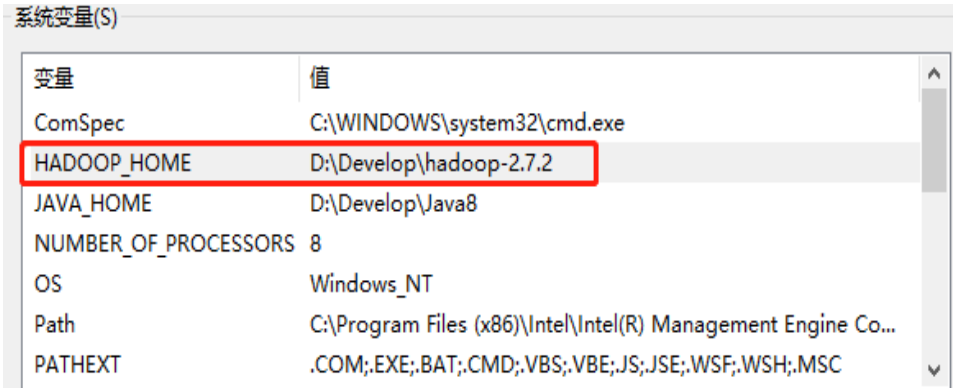


图 3-5　配置 HADOOP_HOME 环境变量

3. 配置 Path 环境变量，如图 3-6 所示。

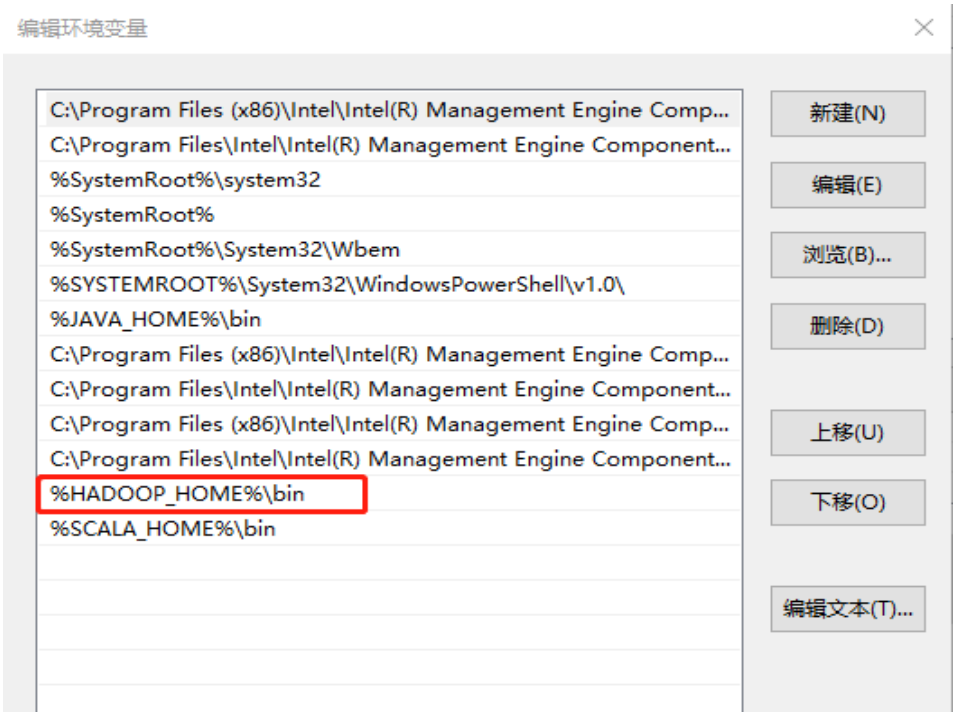


图 3-6　配置 Path 环境变量

4．创建一个 Maven 工程 HdfsClientDemo

5．导入相应的依赖坐标+日志添加

```
<dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.8.2</version>
        </dependency>
        <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>
        <dependency>
            <groupId>jdk.tools</groupId>
            <artifactId>jdk.tools</artifactId>
```

```
            <version>1.8</version>
            <scope>system</scope>
            <systemPath>${JAVA_HOME}/lib/tools.jar</systemPath>
        </dependency>
</dependencies>
```

注意：如果 Eclipse/Idea 打印不出日志，在控制台上只显示

```
1.log4j:WARN No appenders could be found for logger (org.apach
e.hadoop.util.Shell).
2.log4j:WARN Please initialize the log4j system properly.
3.log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#
noconfig for more info.
```

需要在项目的 src/main/resources 目录下，新建一个文件，命名为“log4j.properties”，在文件中填入

```
log4j.rootLogger=INFO, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d    %p    [%c]
- %m%n
log4j.appender.logfile=org.apache.log4j.FileAppender
log4j.appender.logfile.File=target/spring.log
log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
log4j.appender.logfile.layout.ConversionPattern=%d    %p    [%c]
- %m%n
```

6．创建包名：com.atguigu.hdfs

7．创建 HdfsClient 类

```
public class HdfsClient{
@Test
public      void      testMkdirs()      throws      IOException,
InterruptedException, URISyntaxException{

        // 1 获取文件系统
        Configuration configuration = new Configuration();
        // 配置在集群上运行
        //                      configuration.set("fs.defaultFS",
"hdfs://hadoop102:9000");
        // FileSystem fs = FileSystem.get(configuration);

        FileSystem         fs         =         FileSystem.get(new
URI("hdfs://hadoop102:9000"), configuration, "atguigu");

        // 2 创建目录
        fs.mkdirs(new Path("/1108/daxian/banzhang"));

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

8．执行程序

运行时需要配置用户名称，如图 3-7 所示

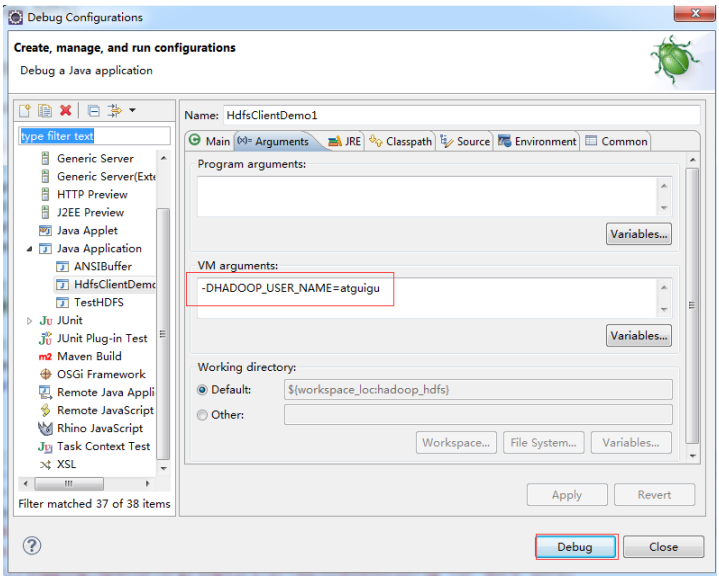


图 3-7　配置用户名称

客户端去操作 HDFS 时，是有一个用户身份的。默认情况下，HDFS 客户端 API 会从 JVM 中获取一个参数来作为自己的用户身份：-DHADOOP_USER_NAME=atguigu，atguigu 为用户名称。

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();
	configuration.set("dfs.replication", "2");
	FileSystem fs = FileSystem.get(new
URI("hdfs://hadoop102:9000"), 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();

	System.out.println("over");
}
```

2．将 hdfs-site.xml 拷贝到项目的根目录下

```
<?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）然后是服务器的默认配置

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://hadoop102:9000"), 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://hadoop102:9000"), configuration, "atguigu");

    // 2 执行删除
    fs.delete(new Path("/0508/"), true);

    // 3 关闭资源
    fs.close();
}
```

3.2.4 HDFS 文件名更改

```
@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://hadoop102:9000"), 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://hadoop102:9000"), 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("-----------班长的分割线----------");
}

// 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://hadoop102:9000"), 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();
}
```

3.3 HDFS 的 I/O 流操作

上面我们学的 API 操作 HDFS 系统都是框架封装好的。那么如果我们想自己实现上述 API 的操作该怎么实现呢？

我们可以采用 IO 流的方式实现数据的上传和下载。

3.3.1 HDFS 文件上传

1．需求：把本地 e 盘上的 banhua.txt 文件上传到 HDFS 根目录

2．编写代码

```
@Test
public void putFileToHDFS() throws IOException,
```

```
InterruptedException, URISyntaxException {

    // 1 获取文件系统
    Configuration configuration = new Configuration();
    FileSystem fs = FileSystem.get(new
URI("hdfs://hadoop102:9000"), configuration, "atguigu");

    // 2 创建输入流
    FileInputStream fis = new FileInputStream(new
File("e:/banhua.txt"));

    // 3 获取输出流
    FSDataOutputStream fos = fs.create(new Path("/banhua.txt"));

    // 4 流对拷
    IOUtils.copyBytes(fis, fos, configuration);

    // 5 关闭资源
    IOUtils.closeStream(fos);
    IOUtils.closeStream(fis);
    fs.close();
}
```

3.3.2 HDFS 文件下载

1．需求：从 HDFS 上下载 banhua.txt 文件到本地 e 盘上

2．编写代码

```
// 文件下载
@Test
public void getFileFromHDFS() throws IOException,
InterruptedException, URISyntaxException{

    // 1 获取文件系统
    Configuration configuration = new Configuration();
    FileSystem fs = FileSystem.get(new
URI("hdfs://hadoop102:9000"), configuration, "atguigu");

    // 2 获取输入流
    FSDataInputStream fis = fs.open(new Path("/banhua.txt"));

    // 3 获取输出流
    FileOutputStream fos = new FileOutputStream(new
File("e:/banhua.txt"));

    // 4 流的对拷
    IOUtils.copyBytes(fis, fos, configuration);

    // 5 关闭资源
    IOUtils.closeStream(fos);
    IOUtils.closeStream(fis);
    fs.close();
}
```

3.3.3 定位文件读取

1．需求：分块读取 HDFS 上的大文件，比如根目录下的/hadoop-2.7.2.tar.gz

2．编写代码

（1）下载第一块

```
@Test
public void readFileSeek1() throws IOException, InterruptedException, URISyntaxException{

    // 1 获取文件系统
    Configuration configuration = new Configuration();
    FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9000"), configuration, "atguigu");

    // 2 获取输入流
    FSDataInputStream fis = fs.open(new Path("/hadoop-2.7.2.tar.gz"));

    // 3 创建输出流
    FileOutputStream fos = new FileOutputStream(new File("e:/hadoop-2.7.2.tar.gz.part1"));

    // 4 流的拷贝
    byte[] buf = new byte[1024];

    for(int i =0 ; i < 1024 * 128; i++){
        fis.read(buf);
        fos.write(buf);
    }

    // 5 关闭资源
    IOUtils.closeStream(fis);
    IOUtils.closeStream(fos);
    fs.close();
}
```

（2）下载第二块

```
@Test
public void readFileSeek2() throws IOException, InterruptedException, URISyntaxException{

    // 1 获取文件系统
    Configuration configuration = new Configuration();
    FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9000"), configuration, "atguigu");

    // 2 打开输入流
    FSDataInputStream fis = fs.open(new Path("/hadoop-2.7.2.tar.gz"));

    // 3 定位输入数据位置
    fis.seek(1024*1024*128);
```

```
    // 4 创建输出流
    FileOutputStream fos = new FileOutputStream(new File("e:/hadoop-2.7.2.tar.gz.part2"));

    // 5 流的对拷
    IOUtils.copyBytes(fis, fos, configuration);

    // 6 关闭资源
    IOUtils.closeStream(fis);
    IOUtils.closeStream(fos);
}
```

（3）合并文件

在 Window 命令窗口中进入到目录 E:\，然后执行如下命令，对数据进行合并

type hadoop-2.7.2.tar.gz.part2 >> hadoop-2.7.2.tar.gz.part1

合并完成后，将 hadoop-2.7.2.tar.gz.part1 重新命名为 hadoop-2.7.2.tar.gz。解压发现该 tar 包非常完整。

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 3-8 所示。

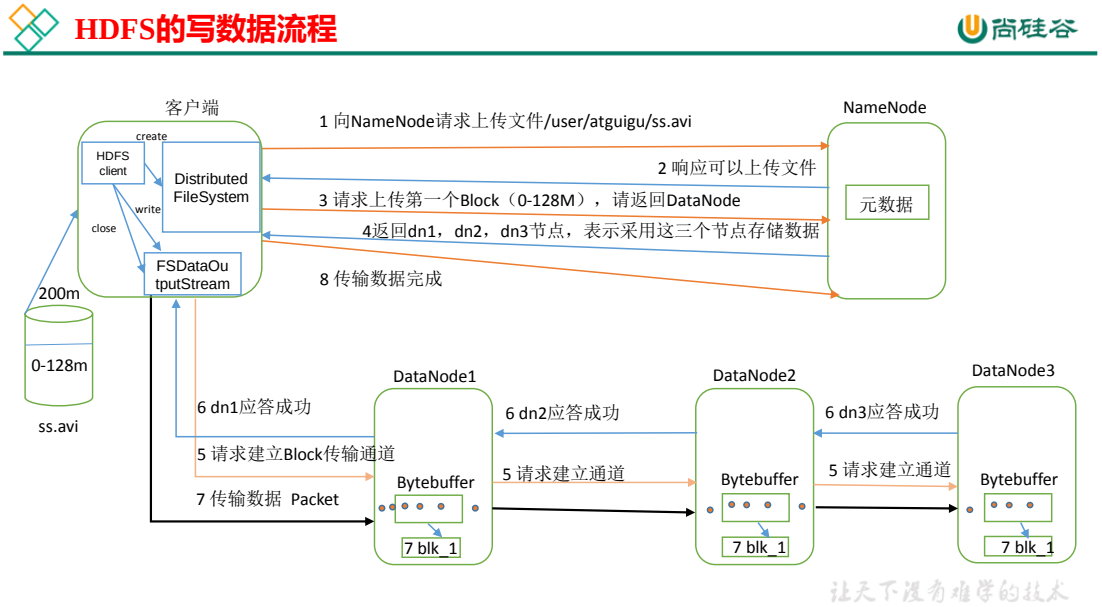


图 3-8　配置用户名称

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。

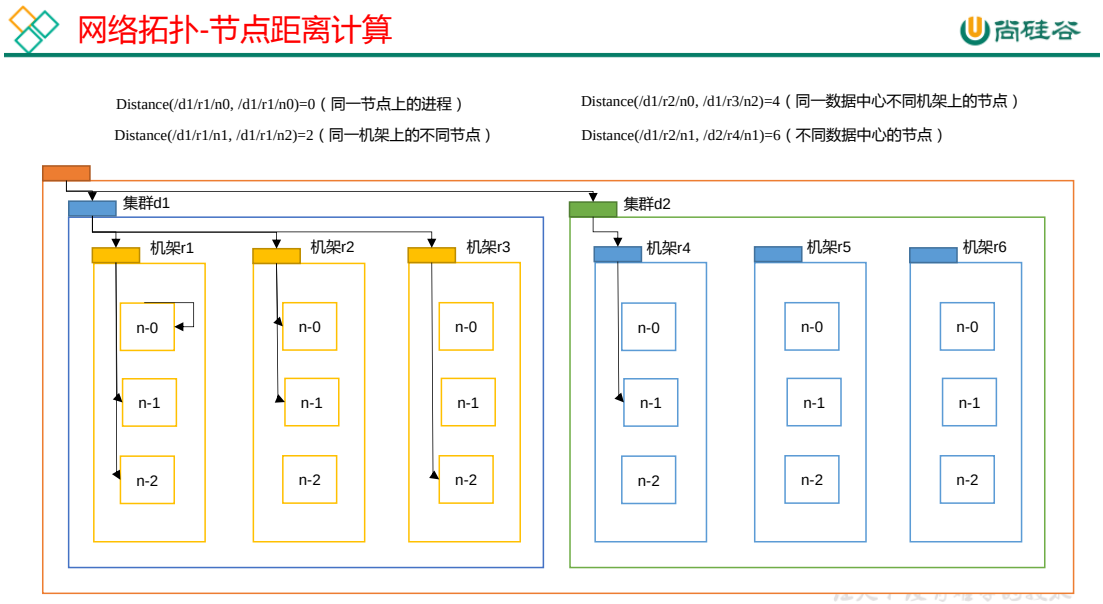


图 3-9　网络拓扑概念

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 3-9 所示。

大家算一算每两个节点之间的距离，如图 3-10 所示。

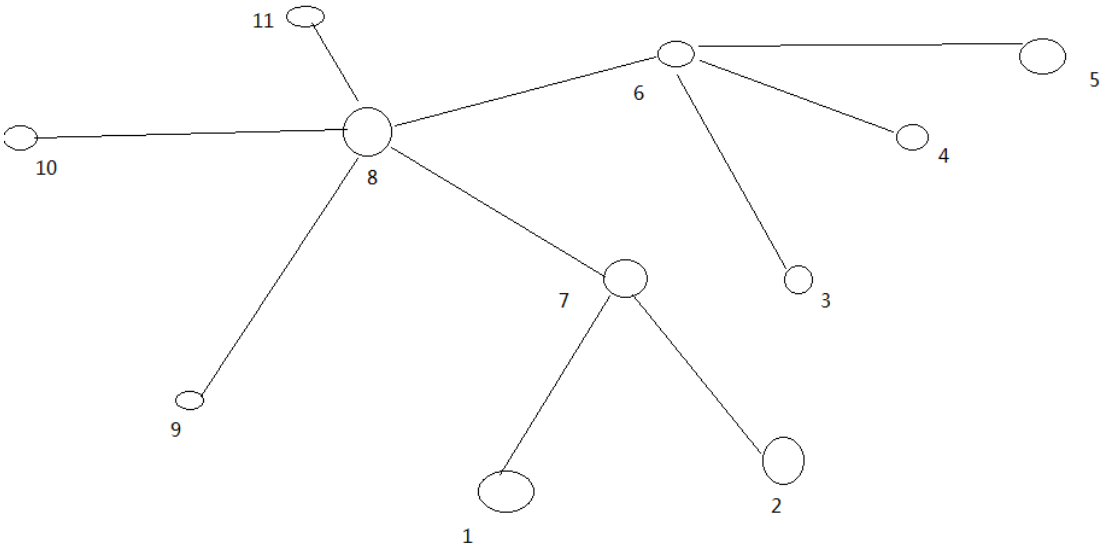


图 3-10　网络拓扑

4.1.3 机架感知（副本存储节点选择）

1. 官方 ip 地址

机架感知说明

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

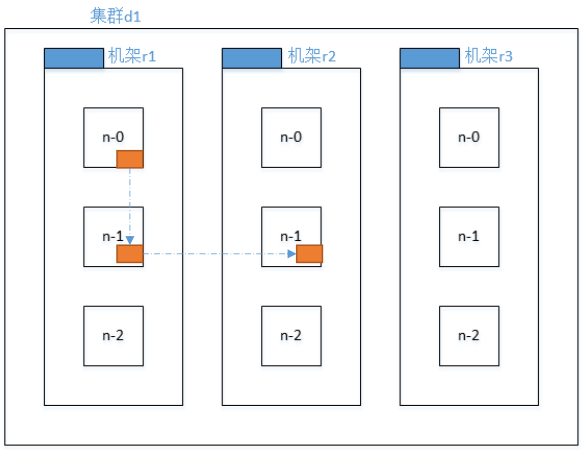
```
For the common case, when the replication factor is three, HDFS's
placement policy is to put one replica on one node in the local
rack, another on a different node in the local rack, and the
last on a different node in a different rack.
```

2. Hadoop2.7.2 副本节点选择

第一个副本在Client所处的节点上。如果客户端在集群外，随机选一个。

第二个副本和第一个副本位于相同机架，随机节点。

第三个副本位于不同机架，随机节点。



4.2 HDFS 读数据流程

HDFS 的读数据流程，如图 3-13 所示。

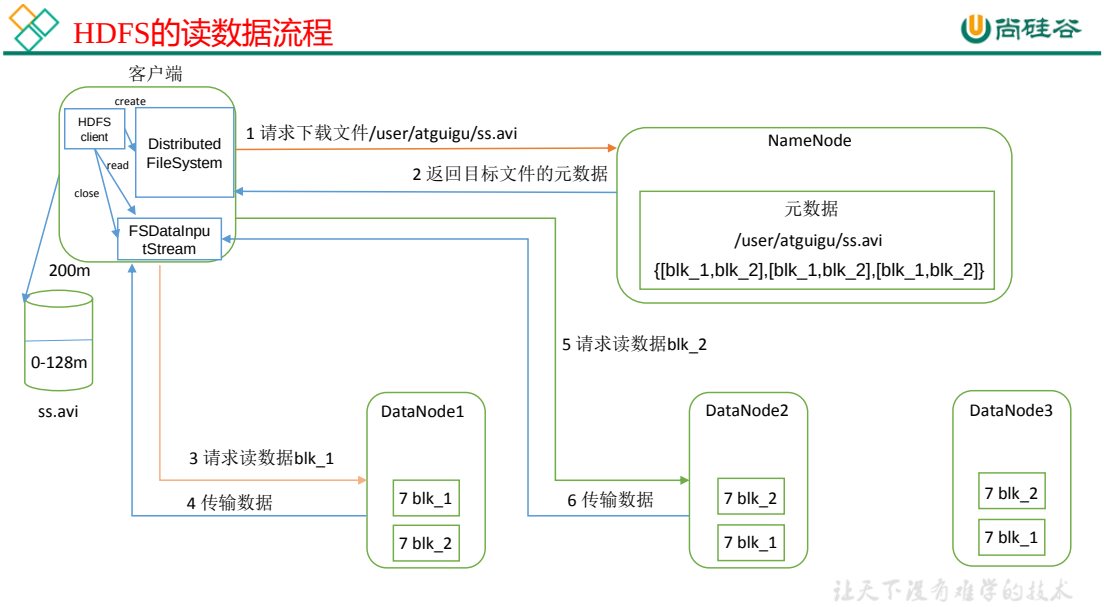


图 3-13　HDFS 读数据流程

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 3-14 所示。

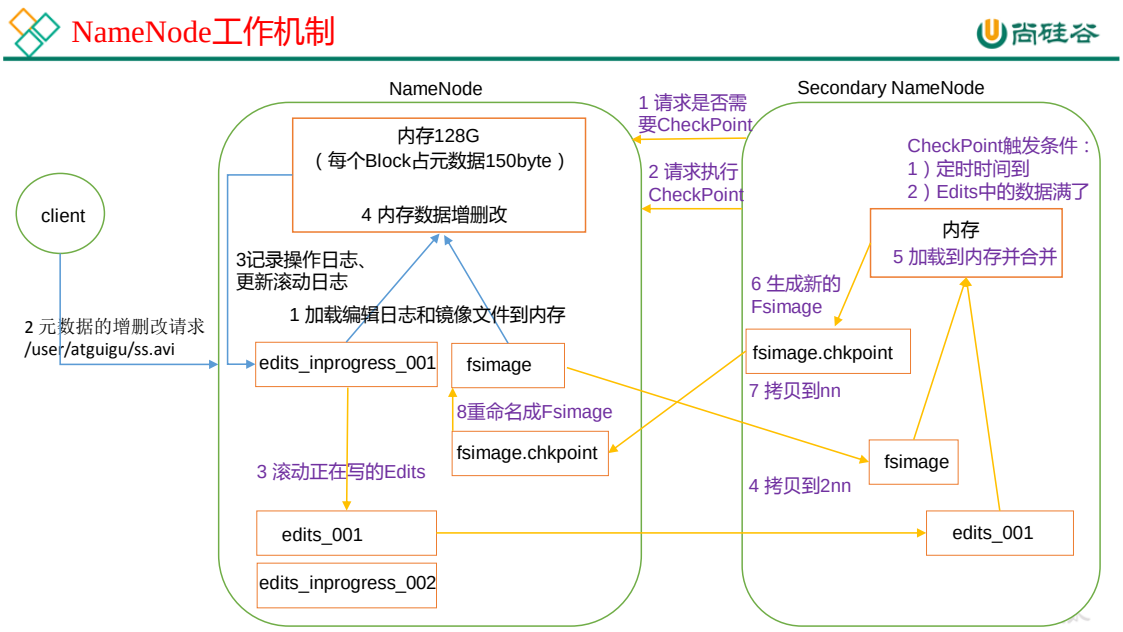


图 3-14　NN 和 2NN 工作机制

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。

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

尚硅谷

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。

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-2.7.2/data/tmp/dfs/name/current

[atguigu@hadoop102 current]$ hdfs oiv -p XML -i fsimage_0000000000000000025 -o /opt/module/hadoop-2.7.2/fsimage.xml

[atguigu@hadoop102 current]$ cat /opt/module/hadoop-2.7.2/fsimage.xml
```

将显示的 xml 文件内容拷贝到 Eclipse 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-2.7.2/edits.xml

[atguigu@hadoop102 current]$ cat /opt/module/hadoop-
2.7.2/edits.xml
```

将显示的 xml 文件内容拷贝到 Eclipse 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

NameNode 故障后，可以采用如下两种方法恢复数据。

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-2.7.2/data/tmp/dfs/name）

```
[atguigu@hadoop102 hadoop-2.7.2]$ rm -rf /opt/module/hadoop-2.7.2/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-2.7.2/data/tmp/dfs/namesecondary/* ./name/
```

4. 重新启动 NameNode

```
[atguigu@hadoop102 hadoop-2.7.2]$ sbin/hadoop-daemon.sh start namenode
```

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-2.7.2/data/tmp/dfs/name</value>
</property>
```

2. kill -9 NameNode 进程

3. 删除 NameNode 存储的数据（/opt/module/hadoop-2.7.2/data/tmp/dfs/name）

```
[atguigu@hadoop102 hadoop-2.7.2]$ rm -rf /opt/module/hadoop-2.7.2/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-2.7.2/data/tmp/dfs/namesecondary ./

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

[atguigu@hadoop102 dfs]$ pwd
/opt/module/hadoop-2.7.2/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-2.7.2]$ bin/hdfs namenode -importCheckpoint
```

6. 启动 NameNode

```
[atguigu@hadoop102 hadoop-2.7.2]$ sbin/hadoop-daemon.sh start namenode
```

5.5 集群安全模式

1. 概述

集群安全模式

尚硅谷

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。

2. 基本语法

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

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

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

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

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

3. 案例

模拟等待安全模式

（1）查看当前模式

```
[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfsadmin -safemode get
Safe mode is OFF
```

（2）先进入安全模式

```
[atguigu@hadoop102 hadoop-2.7.2]$ bin/hdfs dfsadmin -safemode enter
```

（3）创建并执行下面的脚本

在/opt/module/hadoop-2.7.2 路径上，编辑一个脚本 safemode.sh

```
[atguigu@hadoop102 hadoop-2.7.2]$ touch safemode.sh
[atguigu@hadoop102 hadoop-2.7.2]$ vim safemode.sh

#!/bin/bash
hdfs dfsadmin -safemode wait
```

```
hdfs dfs -put /opt/module/hadoop-2.7.2/README.txt /

[atguigu@hadoop102 hadoop-2.7.2]$ chmod 777 safemode.sh

[atguigu@hadoop102 hadoop-2.7.2]$ ./safemode.sh
```

（4）再打开一个窗口，执行

```
[atguigu@hadoop102 hadoop-2.7.2]$ 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 文件中增加如下内容

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

（2）停止集群，删除 data 和 logs 中所有数据。

```
[atguigu@hadoop102 hadoop-2.7.2]$ rm -rf data/ logs/
[atguigu@hadoop103 hadoop-2.7.2]$ rm -rf data/ logs/
[atguigu@hadoop104 hadoop-2.7.2]$ rm -rf data/ logs/
```

（3）格式化集群并启动。

```
[atguigu@hadoop102 hadoop-2.7.2]$ bin/hdfs namenode -format
[atguigu@hadoop102 hadoop-2.7.2]$ 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

DataNode 工作机制，如图 3-15 所示。

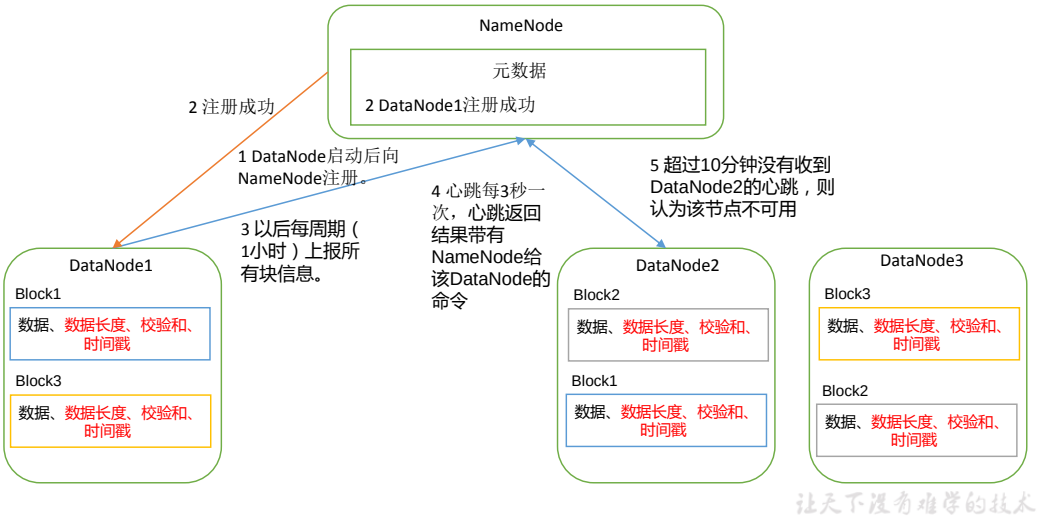


图 3-15　DataNode 工作机制

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）DataNode 在其文件创建后周期验证 CheckSum，如图 3-16 所示。

数据完整性

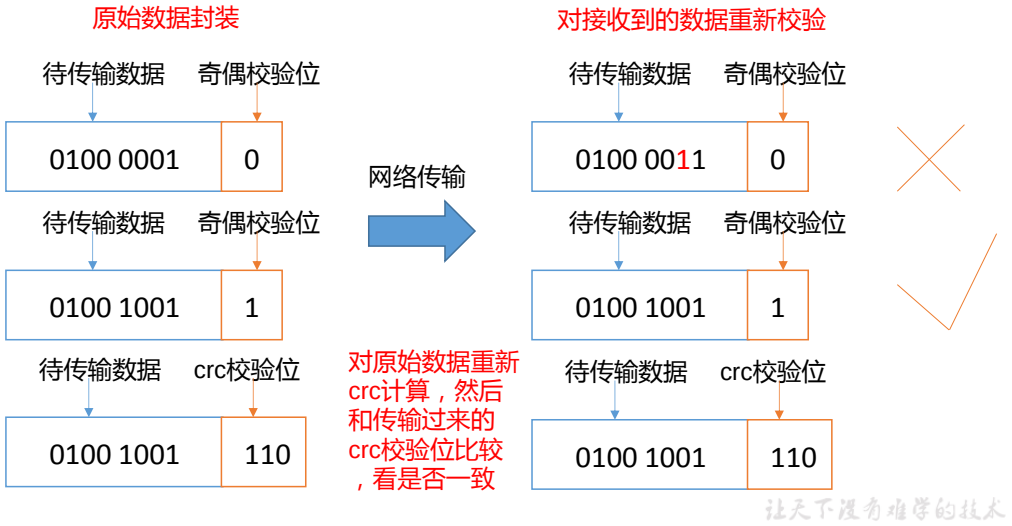
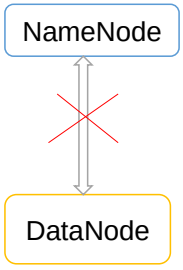


图 3-16　校验和

6.3 掉线时限参数设置

DataNode掉线时限参数设置

1、DataNode进程死亡或者网络故障造成DataNode无法与NameNode通信



2、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。

3、HDFS默认的超时时长为10分钟+30秒。

4、如果定义超时时间为TimeOut，则超时时长的计算公式为：

TimeOut = 2 * dfs.namenode.heartbeat.recheck-interval + 10 * dfs.heartbeat.interval。

而默认的dfs.namenode.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。

```
<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

0. 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。

1. 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-2.7.2/data 和 log）**

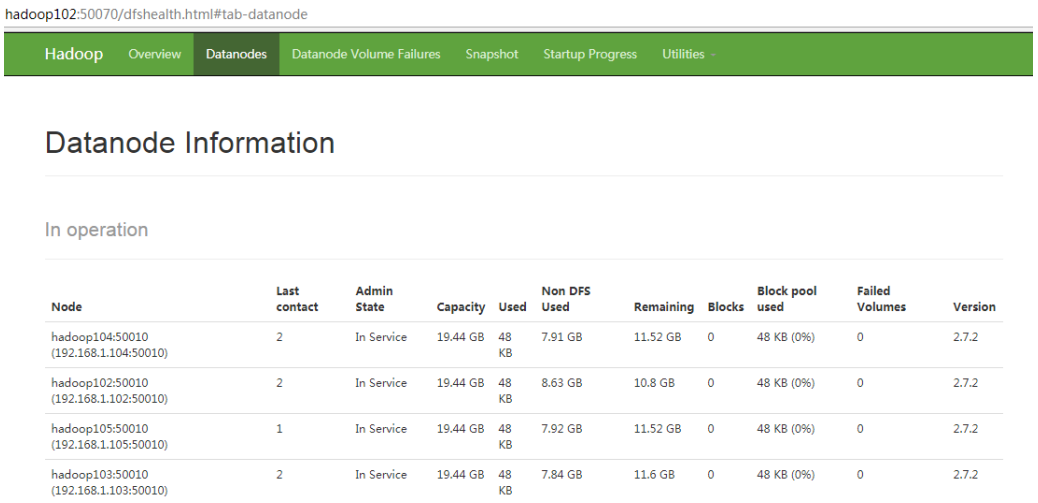
（4）source 一下配置文件

```
[atguigu@hadoop105 hadoop-2.7.2]$ source /etc/profile
```

2. 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-2.7.2]$ sbin/hadoop-daemon.sh start datanode
[atguigu@hadoop105 hadoop-2.7.2]$ sbin/yarn-daemon.sh start nodemanager
```



（2）在 hadoop105 上上传文件

```
[atguigu@hadoop105 hadoop-2.7.2]$ hadoop fs -put /opt/module/hadoop-2.7.2/LICENSE.txt /
```

（3）如果数据不均衡，可以用命令实现集群的再平衡

```
[atguigu@hadoop102 sbin]$ ./start-balancer.sh
starting balancer, logging to /opt/module/hadoop-2.7.2/logs/hadoop-atguigu-balancer-hadoop102.out
Time Stamp Iteration# Bytes Already Moved Bytes Left To Move Bytes Being Moved
```

6.5 退役旧数据节点

6.5.1 添加白名单

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，都会被退出。

配置白名单的具体步骤如下：

（1）在 NameNode 的/opt/module/hadoop-2.7.2/etc/hadoop 目录下创建 dfs.hosts 文件

```
[atguigu@hadoop102 hadoop]$ pwd
/opt/module/hadoop-2.7.2/etc/hadoop
[atguigu@hadoop102 hadoop]$ touch dfs.hosts
[atguigu@hadoop102 hadoop]$ vi dfs.hosts
```

添加如下主机名称（不添加 hadoop105）

```
hadoop102
hadoop103
hadoop104
```

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

```
<property>
<name>dfs.hosts</name>
<value>/opt/module/hadoop-2.7.2/etc/hadoop/dfs.hosts</value>
</property>
```

（3）配置文件分发

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

（4）刷新 NameNode

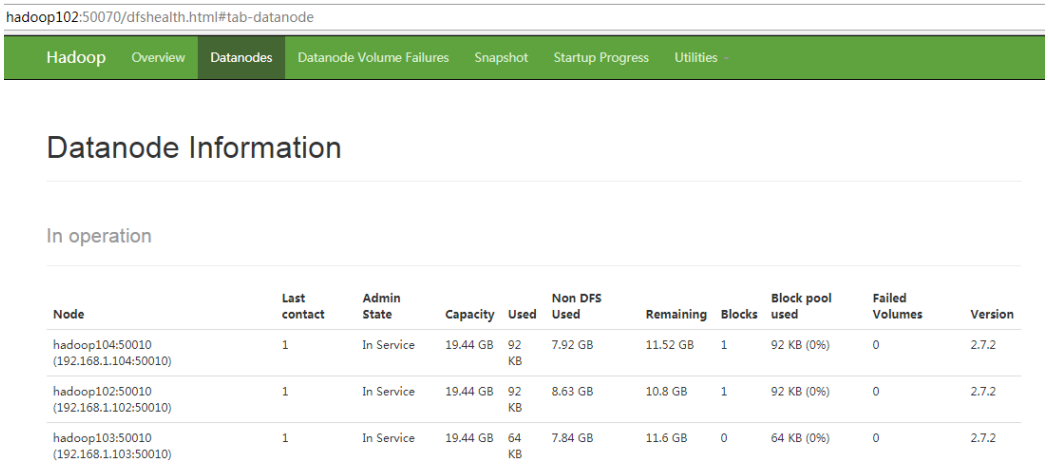
```
[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfsadmin -refreshNodes
```

Refresh nodes successful

（5）更新 ResourceManager 节点

```
[atguigu@hadoop102 hadoop-2.7.2]$ yarn rmadmin -refreshNodes
17/06/24 14:17:11 INFO client.RMProxy: Connecting to
ResourceManager at hadoop103/192.168.1.103:8033
```

（6）在 web 浏览器上查看



4. 如果数据不均衡，可以用命令实现集群的再平衡

```
[atguigu@hadoop102 sbin]$ ./start-balancer.sh
starting balancer, logging to /opt/module/hadoop-2.7.2/logs/hadoop-atguigu-balancer-hadoop102.out
Time Stamp Iteration# Bytes Already Moved Bytes Left To Move Bytes Being Moved
```

6.5.2 黑名单退役

在黑名单上面的主机都会被强制退出。

1. 在 NameNode 的 /opt/module/hadoop-2.7.2/etc/hadoop 目录下创建 dfs.hosts.exclude 文件

```
[atguigu@hadoop102 hadoop]$ pwd
/opt/module/hadoop-2.7.2/etc/hadoop
[atguigu@hadoop102 hadoop]$ touch dfs.hosts.exclude
[atguigu@hadoop102 hadoop]$ vi dfs.hosts.exclude
```

添加如下主机名称（要退役的节点）

```
hadoop105
```

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

```
<property>
<name>dfs.hosts.exclude</name>
     <value>/opt/module/hadoop-2.7.2/etc/hadoop/dfs.hosts.exclude</value>
</property>
```

3. 刷新 NameNode、刷新 ResourceManager

```
[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfsadmin -refreshNodes
Refresh nodes successful

[atguigu@hadoop102 hadoop-2.7.2]$ yarn rmadmin -refreshNodes
17/06/24 14:55:56 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.1.103:8033
```

4. 检查 Web 浏览器，退役节点的状态为 decommission in progress（退役中），说明数据节点正在复制块到其他节点，如图 3-17 所示

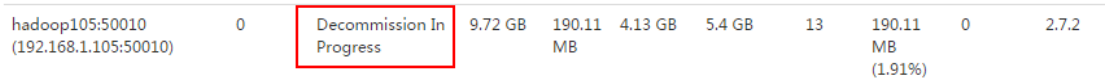


图 3-17 退役中

5. 等待退役节点状态为 decommissioned（所有块已经复制完成），停止该节点及节点资源管理器。注意：如果副本数是 3，服役的节点小于等于 3，是不能退役成功的，需要修改副本数后才能退役，如图 3-18 所示

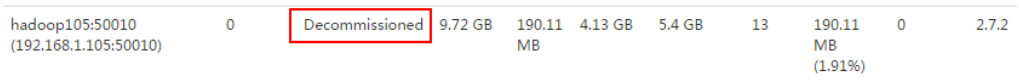


图 3-18 已退役

```
[atguigu@hadoop105 hadoop-2.7.2]$ sbin/hadoop-daemon.sh stop datanode
```

stopping datanode

```
[atguigu@hadoop105 hadoop-2.7.2]$ sbin/yarn-daemon.sh stop
nodemanager
```

stopping nodemanager

6. 如果数据不均衡，可以用命令实现集群的再平衡

```
[atguigu@hadoop102 hadoop-2.7.2]$ sbin/start-balancer.sh
starting balancer, logging to /opt/module/hadoop-
2.7.2/logs/hadoop-atguigu-balancer-hadoop102.out
Time Stamp Iteration# Bytes Already Moved
Bytes Left To Move Bytes Being Moved
```

注意：不允许白名单和黑名单中同时出现同一个主机名称。

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

hdfs-site.xml

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

第 7 章 HDFS 2.X 新特性

7.1 集群间数据拷贝

1. scp 实现两个远程主机之间的文件复制

scp -r hello.txt root@hadoop103:/user/atguigu/hello.txt // 推 push

scp -r root@hadoop103:/user/atguigu/hello.txt hello.txt // 拉 pull

scp -r root@hadoop103:/user/atguigu/hello.txt root@hadoop104:/user/atguigu //是通过本地主机中转实现两个远程主机的文件复制；如果在两个远程主机之间 ssh 没有配置的情况下可以使用该方式。

2. 采用 distcp 命令实现两个 Hadoop 集群之间的递归数据复制

```
[atguigu@hadoop102 hadoop-2.7.2]$ bin/hadoop distcp
hdfs://haoop102:9000/user/atguigu/hello.txt
hdfs://hadoop103:9000/user/atguigu/hello.txt
```

7.2 小文件存档

小文件存档

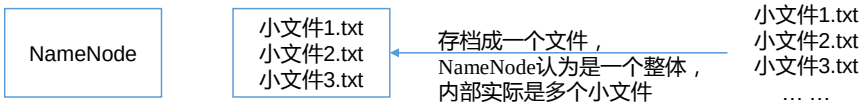
尚硅谷

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。



3．案例实操

（1）需要启动 YARN 进程

```
[atguigu@hadoop102 hadoop-2.7.2]$ 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-2.7.2]$ bin/hadoop archive -archiveName input.har -p /user/atguigu/input /user/atguigu/output
```

（3）查看归档

```
[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -lsr /user/atguigu/output/input.har
[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -lsr har:///user/atguigu/output/input.har
```

（4）解归档文件

```
[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -cp har:///user/atguigu/output/input.har/* /user/atguigu
```

7.3 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：

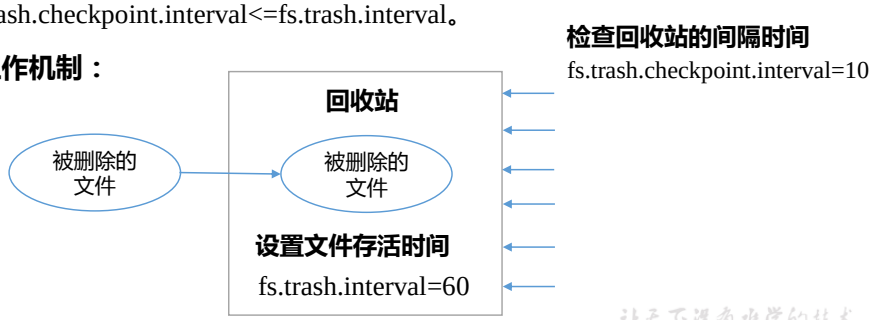


图 3-19 回收站

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);

5. 恢复回收站数据

```
[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -mv
/user/atguigu/.Trash/Current/user/atguigu/input
/user/atguigu/input
```

6. 清空回收站

```
[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -expunge
```

7.4 快照管理

快照管理

快照相当于对目录做一个备份。并不会立即复制所有文件，而是记录文件变化。

(1) hdfs dfsadmin -allowSnapshot 路径 （功能描述：开启指定目录的快照功能）

(2) hdfs dfsadmin -disallowSnapshot 路径（功能描述：禁用指定目录的快照功能，默认是禁用）

(3) hdfs dfs -createSnapshot 路径 （功能描述：对目录创建快照）

(4) hdfs dfs -createSnapshot 路径 名称 （功能描述：指定名称创建快照）

(5) hdfs dfs -renameSnapshot 路径 旧名称 新名称（功能描述：重命名快照）

(6) hdfs lsSnapshottableDir （功能描述：列出当前用户所有可快照目录）

(7) hdfs snapshotDiff 路径1 路径2 （功能描述：比较两个快照目录的不同之处）

(8) hdfs dfs -deleteSnapshot <path> <snapshotName> （功能描述：删除快照）

2. 案例实操

（1）开启/禁用指定目录的快照功能

```
[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfsadmin -allowSnapshot /user/atguigu/input

[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfsadmin -disallowSnapshot /user/atguigu/input
```

（2）对目录创建快照

```
[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfs -createSnapshot /user/atguigu/input
```

通过 web 访问 hdfs://hadoop102:50070/user/atguigu/input/.snapshot/s…..// 快照和源文件使用相同数据

```
[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfs -lsr /user/atguigu/input/.snapshot/
```

（3）指定名称创建快照

```
[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfs -createSnapshot /user/atguigu/input miao170508
```

（4）重命名快照

```
[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfs -renameSnapshot /user/atguigu/input/ miao170508 atguigu170508
```

（5）列出当前用户所有可快照目录

```
[atguigu@hadoop102 hadoop-2.7.2]$ hdfs lsSnapshottableDir
```

（6）比较两个快照目录的不同之处

```
[atguigu@hadoop102 hadoop-2.7.2]$ hdfs snapshotDiff /user/atguigu/input/ . .snapshot/atguigu170508
```

（7）恢复快照

```
[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfs -cp
/user/atguigu/input/.snapshot/s20170708-134303.027 /user
```

第 8 章 HDFS HA 高可用

8.1 HA 概述

1）所谓 HA（High Available），即高可用（7*24 小时不中断服务）。

2）实现高可用最关键的策略是消除单点故障。HA 严格来说应该分成各个组件的 HA 机制：HDFS 的 HA 和 YARN 的 HA。

3）Hadoop2.0 之前，在 HDFS 集群中 NameNode 存在单点故障（SPOF）。

4）NameNode 主要在以下两个方面影响 HDFS 集群

NameNode 机器发生意外，如宕机，集群将无法使用，直到管理员重启

NameNode 机器需要升级，包括软件、硬件升级，此时集群也将无法使用

HDFS HA 功能通过配置 Active/Standby 两个 NameNodes 实现在集群中对 NameNode 的热备来解决上述问题。如果出现故障，如机器崩溃或机器需要升级维护，这时可通过此种方式将 NameNode 很快的切换到另外一台机器。

8.2 HDFS-HA 工作机制

通过双 NameNode 消除单点故障

8.2.1 HDFS-HA 工作要点

1. 元数据管理方式需要改变

内存中各自保存一份元数据；

Edits 日志只有 Active 状态的 NameNode 节点可以做写操作；

两个 NameNode 都可以读取 Edits；

共享的 Edits 放在一个共享存储中管理（qjournal 和 NFS 两个主流实现）；

2. 需要一个状态管理功能模块

实现了一个 zkfailover，常驻在每一个 namenode 所在的节点，每一个 zkfailover 负责监控自己所在 NameNode 节点，利用 zk 进行状态标识，当需要进行状态切换时，由 zkfailover 来负责切换，切换时需要防止 brain split 现象的发生。

3. 必须保证两个 NameNode 之间能够 ssh 无密码登录

4. 隔离（Fence），即同一时刻仅仅有一个 NameNode 对外提供服务

8.2.2 HDFS-HA 自动故障转移工作机制

前面学习了使用命令 hdfs haadmin -failover 手动进行故障转移，在该模式下，即使现役 NameNode 已经失效，系统也不会自动从现役 NameNode 转移到待机 NameNode，下面学习如何配置部署 HA 自动进行故障转移。自动故障转移为 HDFS 部署增加了两个新组件：ZooKeeper 和 ZKFailoverController（ZKFC）进程，如图 3-20 所示。ZooKeeper 是维护少量协调数据，通知客户端这些数据的改变和监视客户端故障的高可用服务。HA 的自动故障转移依赖于 ZooKeeper 的以下功能：

1）故障检测：集群中的每个 NameNode 在 ZooKeeper 中维护了一个持久会话，如果机器崩溃，ZooKeeper 中的会话将终止，ZooKeeper 通知另一个 NameNode 需要触发故障转移。

2）现役 NameNode 选择：ZooKeeper 提供了一个简单的机制用于唯一的选择一个节点为 active 状态。如果目前现役 NameNode 崩溃，另一个节点可能从 ZooKeeper 获得特殊的排外锁以表明它应该成为现役 NameNode。

ZKFC 是自动故障转移中的另一个新组件，是 ZooKeeper 的客户端，也监视和管理 NameNode 的状态。每个运行 NameNode 的主机也运行了一个 ZKFC 进程，ZKFC 负责：

1）健康监测：ZKFC 使用一个健康检查命令定期地 ping 与之在相同主机的 NameNode，只要该 NameNode 及时地回复健康状态，ZKFC 认为该节点是健康的。如果该节点崩溃，冻结或进入不健康状态，健康监测器标识该节点为非健康的。

2）ZooKeeper 会话管理：当本地 NameNode 是健康的，ZKFC 保持一个在 ZooKeeper 中打开的会话。如果本地 NameNode 处于 active 状态，ZKFC 也保持一个特殊的 znode 锁，该锁使用了 ZooKeeper 对短暂节点的支持，如果会话终止，锁节点将自动删除。

3）基于 ZooKeeper 的选择：如果本地 NameNode 是健康的，且 ZKFC 发现没有其它的节点当前持有 znode 锁，它将为自己获取该锁。如果成功，则它已经赢得了选择，并负责运行故障转移进程以使它的本地 NameNode 为 Active。故障转移进程与前面描述的手动故障转移相似，首先如果必要保护之前的现役 NameNode，然后本地 NameNode 转换为 Active 状态。

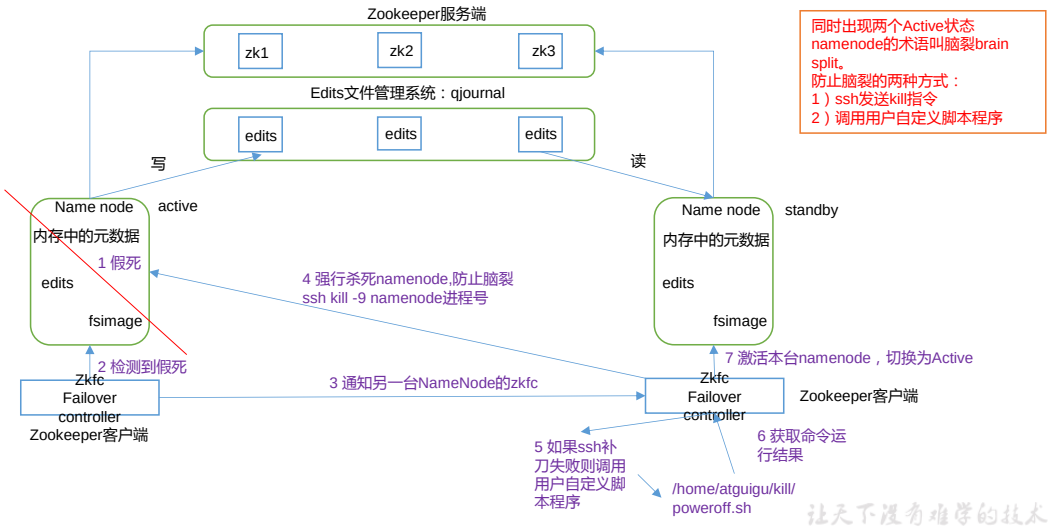


图 3-20 HDFS-HA 故障转移机制

8.3 HDFS-HA 集群配置

8.3.1 环境准备

1. 修改 IP
2. 修改主机名及主机名和 IP 地址的映射
3. 关闭防火墙
4. ssh 免密登录
5. 安装 JDK，配置环境变量等

8.3.2 规划集群

表 3-1

hadoop102	hadoop103	hadoop104
NameNode	NameNode	
JournalNode	JournalNode	JournalNode
DataNode	DataNode	DataNode
ZK	ZK	ZK
	ResourceManager	
NodeManager	NodeManager	NodeManager

8.3.3 配置 Zookeeper 集群

1. 集群规划

在 hadoop102、hadoop103 和 hadoop104 三个节点上部署 Zookeeper。

2. 解压安装

（1）解压 Zookeeper 安装包到/opt/module/目录下

```
[atguigu@hadoop102 software]$ tar -zxvf zookeeper-3.4.10.tar.gz -C /opt/module/
```

（2）在/opt/module/zookeeper-3.4.10/这个目录下创建 zkData

```
mkdir -p zkData
```

（3）重命名/opt/module/zookeeper-3.4.10/conf 这个目录下的 zoo_sample.cfg 为 zoo.cfg

```
mv zoo_sample.cfg zoo.cfg
```

3. 配置 zoo.cfg 文件

（1）具体配置

```
dataDir=/opt/module/zookeeper-3.4.10/zkData
```

增加如下配置

```
#######################cluster##########################
server.2=hadoop102:2888:3888
server.3=hadoop103:2888:3888
server.4=hadoop104:2888:3888
```

（2）配置参数解读

Server.A=B:C:D。

A 是一个数字，表示这个是第几号服务器；

B 是这个服务器的 IP 地址；

C 是这个服务器与集群中的 Leader 服务器交换信息的端口；

D 是万一集群中的 Leader 服务器挂了，需要一个端口来重新进行选举，选出一个新的 Leader，而这个端口就是用来执行选举时服务器相互通信的端口。

集群模式下配置一个文件 myid，这个文件在 dataDir 目录下，这个文件里面有一个数据就是 A 的值，Zookeeper 启动时读取此文件，拿到里面的数据与 zoo.cfg 里面的配置信息比较从而判断到底是哪个 server。

4. 集群操作

（1）在/opt/module/zookeeper-3.4.10/zkData 目录下创建一个 myid 的文件

```
touch myid
```

添加 myid 文件，注意一定要在 linux 里面创建，在 notepad++里面很可能乱码

（2）编辑 myid 文件

```
vi myid
```

在文件中添加与 server 对应的编号：如 2

（3）拷贝配置好的 zookeeper 到其他机器上

```
scp -r zookeeper-3.4.10/ root@hadoop103.atguigu.com:/opt/app/
```

```
scp -r zookeeper-3.4.10/ root@hadoop104.atguigu.com:/opt/app/
```

并分别修改 myid 文件中内容为 3、4

（4）分别启动 zookeeper

```
[root@hadoop102 zookeeper-3.4.10]# bin/zkServer.sh start
[root@hadoop103 zookeeper-3.4.10]# bin/zkServer.sh start
[root@hadoop104 zookeeper-3.4.10]# bin/zkServer.sh start
```

（5）查看状态

```
[root@hadoop102 zookeeper-3.4.10]# bin/zkServer.sh status
JMX enabled by default
Using              config:              /opt/module/zookeeper-
3.4.10/bin/../conf/zoo.cfg
Mode: follower
[root@hadoop103 zookeeper-3.4.10]# bin/zkServer.sh status
JMX enabled by default
Using              config:              /opt/module/zookeeper-
3.4.10/bin/../conf/zoo.cfg
Mode: leader
[root@hadoop104 zookeeper-3.4.5]# bin/zkServer.sh status
JMX enabled by default
Using              config:              /opt/module/zookeeper-
3.4.10/bin/../conf/zoo.cfg
Mode: follower
```

8.3.4 配置 HDFS-HA 集群

1. 官方地址：http://hadoop.apache.org/
2. 在 opt 目录下创建一个 ha 文件夹

```
mkdir ha
```

3. 将/opt/app/下的 hadoop-2.7.2 拷贝到/opt/ha 目录下

```
cp -r hadoop-2.7.2/ /opt/ha/
```

4. 配置 hadoop-env.sh

```
export JAVA_HOME=/opt/module/jdk1.8.0_144
```

5. 配置 core-site.xml

```
<configuration>
<!-- 把两个 NameNode）的地址组装成一个集群 mycluster -->
     <property>
        <name>fs.defaultFS</name>
        <value>hdfs://mycluster</value>
     </property>

     <!-- 指定 hadoop 运行时产生文件的存储目录 -->
     <property>
        <name>hadoop.tmp.dir</name>
        <value>/opt/ha/hadoop-2.7.2/data/tmp</value>
     </property>
</configuration>
```

6. 配置 hdfs-site.xml

```
<configuration>
  <!-- 完全分布式集群名称 -->
```

```
  <property>
     <name>dfs.nameservices</name>
     <value>mycluster</value>
  </property>

  <!-- 集群中 NameNode 节点都有哪些 -->
  <property>
     <name>dfs.ha.namenodes.mycluster</name>
     <value>nn1,nn2</value>
  </property>

  <!-- nn1 的 RPC 通信地址 -->
  <property>
     <name>dfs.namenode.rpc-address.mycluster.nn1</name>
     <value>hadoop102:9000</value>
  </property>

  <!-- nn2 的 RPC 通信地址 -->
  <property>
     <name>dfs.namenode.rpc-address.mycluster.nn2</name>
     <value>hadoop103:9000</value>
  </property>

  <!-- nn1 的 http 通信地址 -->
  <property>
     <name>dfs.namenode.http-address.mycluster.nn1</name>
     <value>hadoop102:50070</value>
  </property>

  <!-- nn2 的 http 通信地址 -->
  <property>
     <name>dfs.namenode.http-address.mycluster.nn2</name>
     <value>hadoop103:50070</value>
  </property>

  <!-- 指定 NameNode 元数据在 JournalNode 上的存放位置 -->
  <property>
     <name>dfs.namenode.shared.edits.dir</name>
  <value>qjournal://hadoop102:8485;hadoop103:8485;hadoop10
4:8485/mycluster</value>
  </property>

  <!-- 配置隔离机制，即同一时刻只能有一台服务器对外响应 -->
  <property>
     <name>dfs.ha.fencing.methods</name>
     <value>sshfence</value>
  </property>

  <!-- 使用隔离机制时需要 ssh 无秘钥登录-->
  <property>
     <name>dfs.ha.fencing.ssh.private-key-files</name>
     <value>/home/atguigu/.ssh/id_rsa</value>
  </property>

  <!-- 声明 journalnode 服务器存储目录-->
  <property>
     <name>dfs.journalnode.edits.dir</name>
```

```
    <value>/opt/ha/hadoop-2.7.2/data/jn</value>
  </property>

  <!-- 关闭权限检查-->
  <property>
      <name>dfs.permissions.enable</name>
      <value>false</value>
  </property>

  <!-- 访问代理类：client，mycluster，active 配置失败自动切换实现方式-->
  <property>

  <name>dfs.client.failover.proxy.provider.mycluster</name>
  <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>
  </property>
</configuration>
```

7. 拷贝配置好的 hadoop 环境到其他节点

8.3.5 启动 HDFS-HA 集群

1. 在各个 JournalNode 节点上，输入以下命令启动 journalnode 服务
```
sbin/hadoop-daemon.sh start journalnode
```
2. 在[nn1]上，对其进行格式化，并启动
```
bin/hdfs namenode -format
sbin/hadoop-daemon.sh start namenode
```
3. 在[nn2]上，同步 nn1 的元数据信息
```
bin/hdfs namenode -bootstrapStandby
```
4. 启动[nn2]
```
sbin/hadoop-daemon.sh start namenode
```
5. 查看 web 页面显示，如图 3-21，3-22 所示

图 3-21 hadoop102(standby)

图 3-22 hadoop103(standby)

6. 在[nn1]上，启动所有 datanode

```
sbin/hadoop-daemons.sh start datanode
```

7. 将[nn1]切换为 Active

```
bin/hdfs haadmin -transitionToActive nn1
```

7. 查看是否 Active

```
bin/hdfs haadmin -getServiceState nn1
```

8.3.6 配置 HDFS-HA 自动故障转移

1. 具体配置

（1）在 hdfs-site.xml 中增加

```
<property>
  <name>dfs.ha.automatic-failover.enabled</name>
  <value>true</value>
</property>
```

（2）在 core-site.xml 文件中增加

```
<property>
  <name>ha.zookeeper.quorum</name>
  <value>hadoop102:2181,hadoop103:2181,hadoop104:2181</value>
</property>
```

2. 启动

（1）关闭所有 HDFS 服务：

```
sbin/stop-dfs.sh
```

（2）启动 Zookeeper 集群：

```
bin/zkServer.sh start
```

（3）初始化 HA 在 Zookeeper 中状态：

```
bin/hdfs zkfc -formatZK
```

（4）启动 HDFS 服务：

```
sbin/start-dfs.sh
```

（5）在各个 NameNode 节点上启动 DFSZK Failover Controller，先在哪台机器启动，哪个机器的 NameNode 就是 Active NameNode

```
sbin/hadoop-daemin.sh start zkfc
```

3. 验证

（1）将 Active NameNode 进程 kill

```
kill -9 namenode 的进程 id
```

（2）将 Active NameNode 机器断开网络

```
service network stop
```

8.4 YARN-HA 配置

8.4.1 YARN-HA 工作机制

1. 官方文档：

http://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/ResourceManagerHA.html

2. YARN-HA 工作机制，如图 3-23 所示

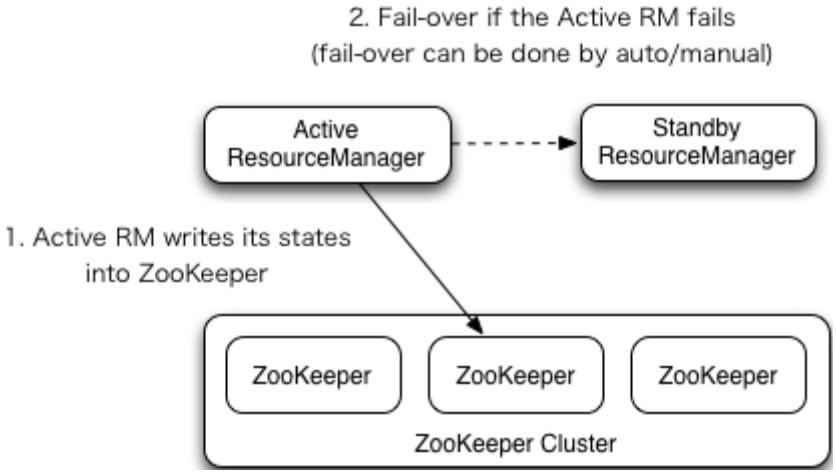


图 3-22 YARN-HA 工作机制

8.4.2 配置 YARN-HA 集群

1. 环境准备

（1）修改 IP

（2）修改主机名及主机名和 IP 地址的映射

（3）关闭防火墙

（4）ssh 免密登录

（5）安装 JDK，配置环境变量等

（6）配置 Zookeeper 集群

2. 规划集群

表 3-2

hadoop102	hadoop103	hadoop104
NameNode	NameNode	
JournalNode	JournalNode	JournalNode
DataNode	DataNode	DataNode
ZK	ZK	ZK

ResourceManager	ResourceManager	
NodeManager	NodeManager	NodeManager

3. 具体配置

（1）yarn-site.xml

```
<configuration>

    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>

    <!--启用 resourcemanager ha-->
    <property>
        <name>yarn.resourcemanager.ha.enabled</name>
        <value>true</value>
    </property>

    <!--声明两台 resourcemanager 的地址-->
    <property>
        <name>yarn.resourcemanager.cluster-id</name>
        <value>cluster-yarn1</value>
    </property>

    <property>
        <name>yarn.resourcemanager.ha.rm-ids</name>
        <value>rm1,rm2</value>
    </property>

    <property>
        <name>yarn.resourcemanager.hostname.rm1</name>
        <value>hadoop102</value>
    </property>

    <property>
        <name>yarn.resourcemanager.hostname.rm2</name>
        <value>hadoop103</value>
    </property>

    <!--指定 zookeeper 集群的地址-->
    <property>
        <name>yarn.resourcemanager.zk-address</name>

<value>hadoop102:2181,hadoop103:2181,hadoop104:2181</value
>
    </property>

    <!--启用自动恢复-->
    <property>
        <name>yarn.resourcemanager.recovery.enabled</name>
        <value>true</value>
    </property>

    <!--指定 resourcemanager 的状态信息存储在 zookeeper 集群-->
```

```
    <property>
        <name>yarn.resourcemanager.store.class</name>
<value>org.apache.hadoop.yarn.server.resourcemanager.recov
ery.ZKRMStateStore</value>
</property>

</configuration>
```

（2）同步更新其他节点的配置信息

4. 启动 hdfs

（1）在各个 JournalNode 节点上，输入以下命令启动 journalnode 服务：

```
sbin/hadoop-daemon.sh start journalnode
```

（2）在[nn1]上，对其进行格式化，并启动：

```
bin/hdfs namenode -format
sbin/hadoop-daemon.sh start namenode
```

（3）在[nn2]上，同步 nn1 的元数据信息：

```
bin/hdfs namenode -bootstrapStandby
```

（4）启动[nn2]：

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

（5）启动所有 DataNode

```
sbin/hadoop-daemons.sh start datanode
```

（6）将[nn1]切换为 Active

```
bin/hdfs haadmin -transitionToActive nn1
```

5. 启动 YARN

（1）在 hadoop102 中执行：

```
sbin/start-yarn.sh
```

（2）在 hadoop103 中执行：

```
sbin/yarn-daemon.sh start resourcemanager
```

（3）查看服务状态，如图 3-24 所示

```
bin/yarn rmadmin -getServiceState rm1
```

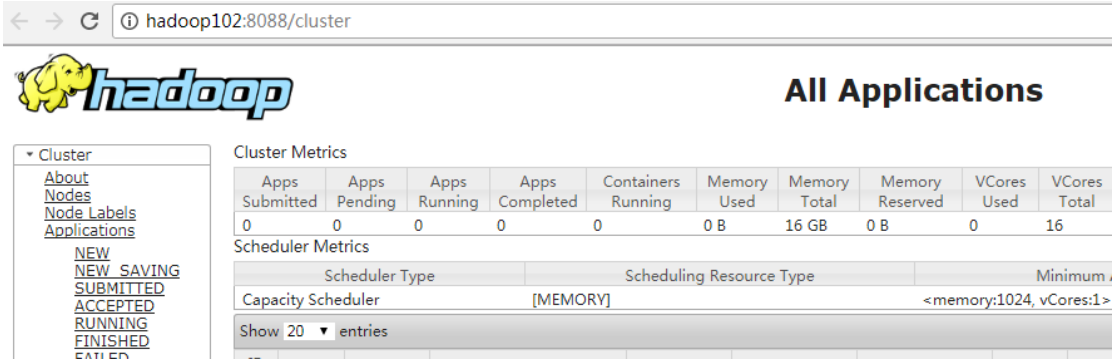


图 3-24　YARN 的服务状态

8.5 HDFS Federation 架构设计

1. NameNode 架构的局限性

（1）Namespace（命名空间）的限制

由于 NameNode 在内存中存储所有的元数据（metadata），因此单个 NameNode 所能存储的对象（文件+块）数目受到 NameNode 所在 JVM 的 heap size 的限制。50G 的 heap 能够存储 20 亿（200million）个对象，这 20 亿个对象支持 4000 个 DataNode，12PB 的存储（假设文件平均大小为 40MB）。随着数据的飞速增长，存储的需求也随之增长。单个 DataNode 从 4T 增长到 36T，集群的尺寸增长到 8000 个 DataNode。存储的需求从 12PB 增长到大于 100PB。

（2）隔离问题

由于 HDFS 仅有一个 NameNode，无法隔离各个程序，因此 HDFS 上的一个实验程序就很有可能影响整个 HDFS 上运行的程序。

（3）性能的瓶颈

由于是单个 NameNode 的 HDFS 架构，因此整个 HDFS 文件系统的吞吐量受限于单个 NameNode 的吞吐量。

2. HDFS Federation 架构设计，如图 3-25 所示

能不能有多个 NameNode

表 3-3

NameNode	NameNode	NameNode
元数据	元数据	元数据
Log	machine	电商数据/话单数据

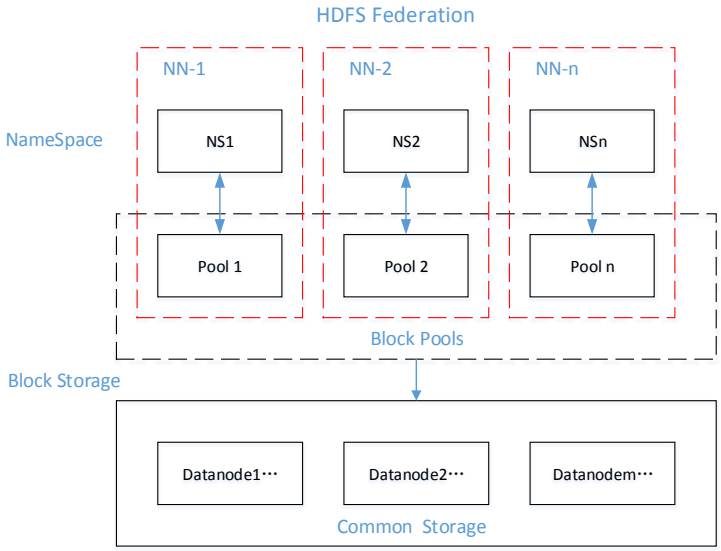


图 3-25　HDFS Federation 架构设计

3. HDFS Federation 应用思考

不同应用可以使用不同 NameNode 进行数据管理

图片业务、爬虫业务、日志审计业务

Hadoop 生态系统中，不同的框架使用不同的 NameNode 进行管理 NameSpace。（隔离性）