尚硅谷大数据技术之 Hadoop
（HDFS 文件系统）

(作者：大海哥)

官网：www.atguigu.com

版本：V1.3

一 HDFS 概述

1.1 HDFS 产生背景

随着数据量越来越大，在一个操作系统管辖的范围内存不下了，那么就分配到更多的操作系统管理的磁盘中，但是不方便管理和维护，迫切需要一种系统来管理多台机器上的文件，这就是分布式文件管理系统。HDFS 只是分布式文件管理系统中的一种。

1.2 HDFS 概念

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

HDFS 的设计适合一次写入，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。

1.3 HDFS 优缺点

1.3.1 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）适合大数据处理

（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）流式数据访问

（1）一次写入，多次读取，不能修改，只能追加。

（2）它能保证数据的一致性。

4）可构建在廉价机器上，通过多副本机制，提高可靠性。

1.3.2 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。

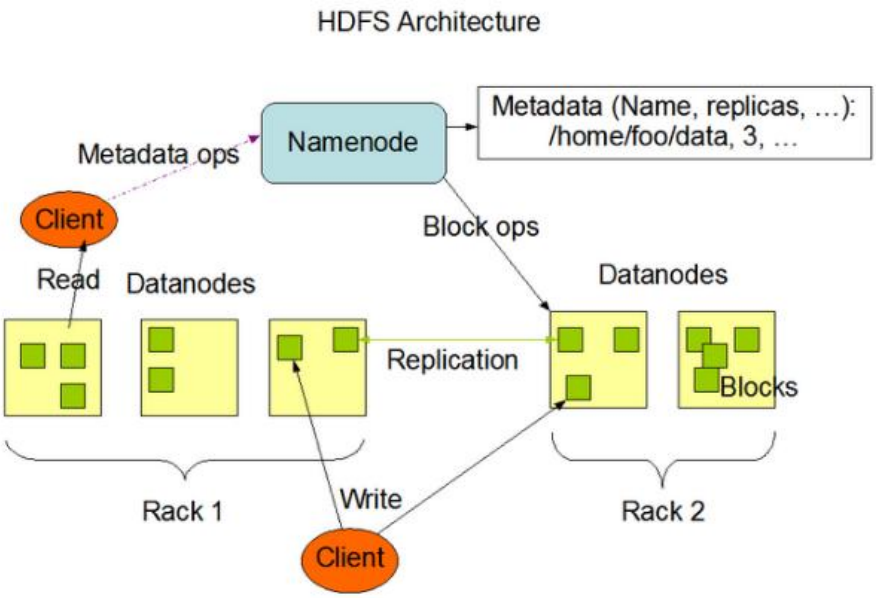
（2）小文件存储的寻址时间会超过读取时间，它违反了 HDFS 的设计目标。

3）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。

1.4 HDFS 架构



HDFS 的架构图

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

1）Client：就是客户端。

（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，然后进行存储。

（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，比如启动或者关闭 HDFS。

（5）Client 可以通过一些命令来访问 HDFS。

2）NameNode：就是 master，它是一个主管、管理者。

（1）管理 HDFS 的名称空间。

（2）管理数据块（Block）映射信息

（3）配置副本策略

（4）处理客户端读写请求。

3） 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。

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。 [sekəndri]

（1）辅助 NameNode，分担其工作量。

（2）定期合并 Fsimage 和 Edits，并推送给 NameNode。

（3）在紧急情况下，可辅助恢复 NameNode。

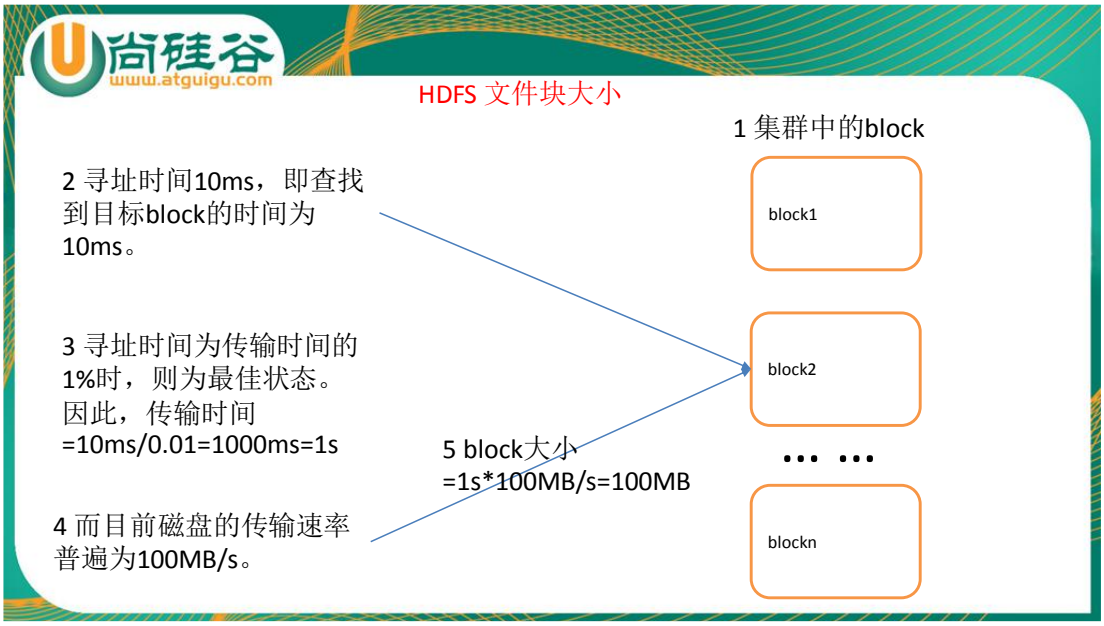
1.5 HDFS 文件块大小

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 hadoop2.x 版本中是 128M，老版本中是 64M。

HDFS 的块比磁盘的块大，其目的是为了最小化寻址开销。如果块设置得足够大，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。因而，传输一个由多个块组成的文件的时间取决于磁盘传输速率。

如果寻址时间约为 10ms，而传输速率为 100MB/s，为了使寻址时间仅占传输时间的 1%，我们要将块大小设置约为 100MB。默认的块大小 128MB。

块的大小：10ms*100*100M/s = 100M 1s=1000ms



二 HFDS 命令行操作

1）基本语法

bin/hadoop fs 具体命令

2）参数大全

[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 / ← hadoop fs -lsr 递归查看

（3）-mkdir：在 hdfs 上创建目录

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -mkdir -p /user/atguigu/test

（4）-moveFromLocal 从本地剪切粘贴到 hdfs

[atguigu@hadoop102 hadoop-2.7.2]$ touch jinlian.txt

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -moveFromLocal ./jinlian.txt /user/atguigu/test

（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 ximen.txt

[atguigu@hadoop102 hadoop-2.7.2]$ vi ximen.txt

输入

wo ai jinlian

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -appendToFile ximen.txt /user/atguigu/test/jinlian.txt

（6）-cat ：显示文件内容

（7）-tail：显示一个文件的末尾

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -tail /user/atguigu/test/jinlian.txt

（8）-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 /user/atguigu/test/jinlian.txt

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -chown atguigu:atguigu /user/atguigu/test/jinlian.txt

（9）-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 /user/atguigu/test

（10）-copyToLocal：从 hdfs 拷贝到本地

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -copyToLocal /user/atguigu/test/jinlian.txt ./jinlian.txt

（11）-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 /user/atguigu/test/jinlian.txt /jinlian2.txt

（12）-mv：在 hdfs 目录中移动文件

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

（13）-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 /user/atguigu/test/jinlian2.txt ./

（14）-getmerge ：合并下载多个文件，比如 hdfs 的目录 /aaa/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

（15）-put：等同于 copyFromLocal

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

（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）-df ：统计文件系统的可用空间信息

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -df -h /

（19）-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

（20）-setrep：设置 hdfs 中文件的副本数量

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -setrep 2 /user/atguigu/test/jinlian.txt

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	atguigu	supergroup	15 B	2017/12/5 下午4:41:31	2	128 MB	jinlian.txt

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。

三 HDFS 客户端操作

3.1 HDFS 客户端环境准备

3.1.1 jar 包准备

1）解压 hadoop-2.7.2.tar.gz 到非中文目录

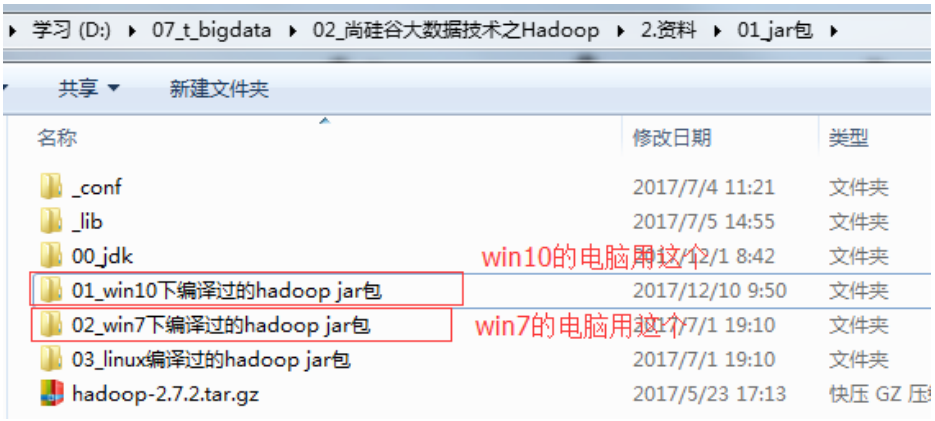
2）进入 share 文件夹，查找所有 jar 包，并把 jar 包拷贝到_lib 文件夹下

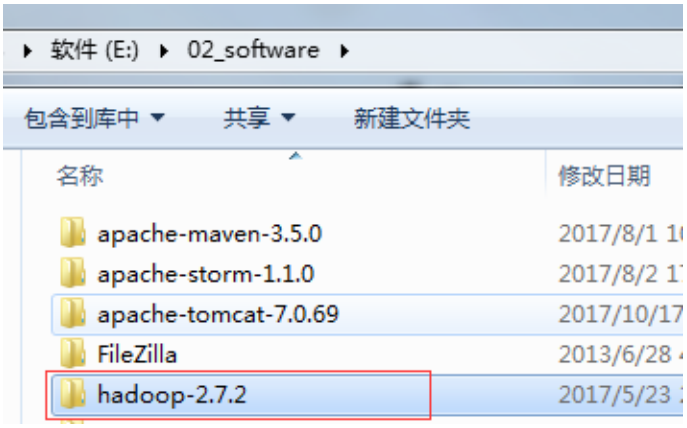
3）在全部 jar 包中查找 sources.jar，并剪切到_source 文件夹。

4）在全部 jar 包中查找 tests.jar，并剪切到_test 文件夹。

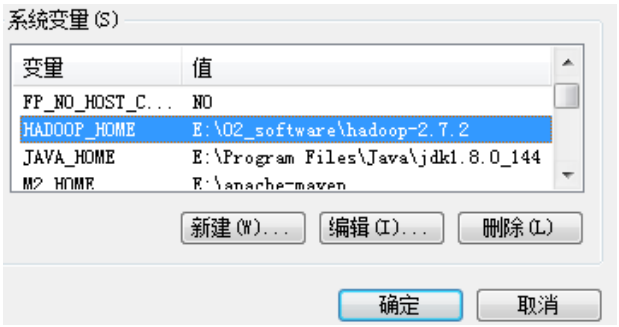
3.1.2 Eclipse 准备

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：E:\02_software\hadoop-2.7.2）。（如果不生效，重新启动 eclipse）



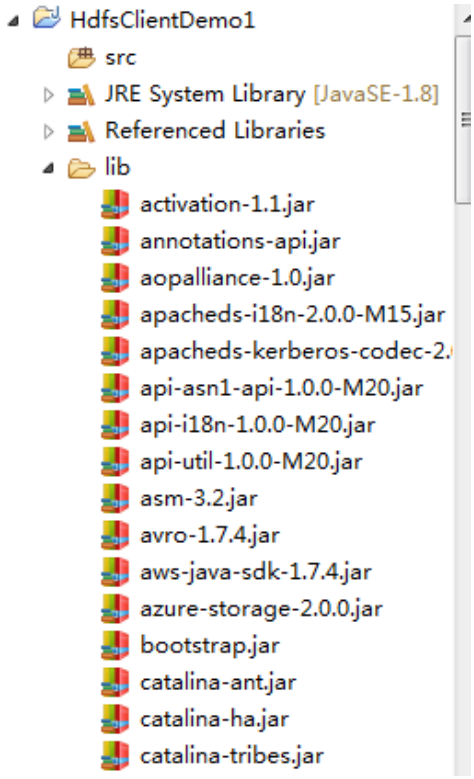


2）配置 HADOOP_HOME 环境变量



3）创建第一个 java 工程 HdfsClientDemo1

4）创建 lib 文件夹，然后添加 jar 包



5）创建包名：com.atguigu.hdfs

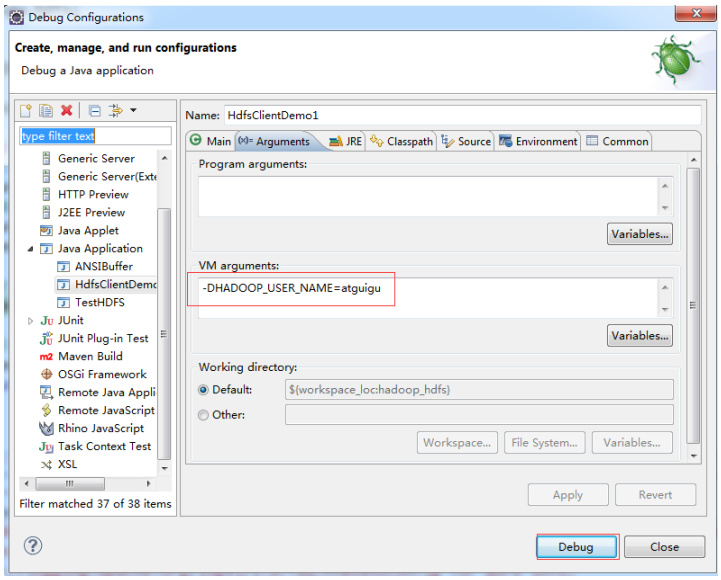
6）创建 HdfsClient 类

```
public class HdfsClient {

    // 上传文件
    public static void main(String[] args) 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 上传文件
        fs.copyFromLocalFile(new Path("e:/hello.txt"), new Path("/hello2.txt"));

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

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

7）执行程序

运行时需要配置用户名称



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。

8）注意：如果 eclipse 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.apache.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 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
```

3.2 通过 API 操作 HDFS

3.2.1 HDFS 获取文件系统

1）详细代码

```
    @Test
    public void initHDFS() throws Exception{
        // 1 创建配置信息对象
        Configuration configuration = new Configuration();

        // 2 获取文件系统
        FileSystem fs = FileSystem.get(configuration);

        // 3 打印文件系统
        System.out.println(fs.toString());
    }
```

3.2.2 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:/hello.txt"), new Path("/hello5.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）测试参数优先级

参数优先级：（1）客户端代码中设置的值 >（2）classpath 下的用户自定义配置文件 >（3）然后是服务器的默认配置

3.2.3 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 是否开启文件效验
        fs.copyToLocalFile(false, new Path("/hello1.txt"), new Path("e:/hello1.txt"), true);

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

3.2.4 HDFS 目录创建

```
    @Test
    public void testMkdirs() 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 创建目录
        fs.mkdirs(new Path("/0906/daxian/banzhang"));

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

3.2.5 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("/0906/"), true);

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

3.2.6 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("/hello.txt"), new Path("/hello6.txt"));

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

3.2.7 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());
            // z 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----------");
        }
    }
```

3.2.8 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 通过 IO 流操作 HDFS

3.3.1 HDFS 文件上传

```
    @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:/hello.txt"));

        // 3 获取输出流
        FSDataOutputStream fos = fs.create(new Path("/hello4.txt"));

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

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

3.3.2 HDFS 文件下载

1）需求：从 HDFS 上下载文件到本地控制台上。

2）编写代码：

```
    @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("/hello4.txt"));

        // 3 获取输出流
        IOUtils.copyBytes(fis, System.out, configuration);
        // 4 流对拷

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

3.3.3 定位文件读取

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);
        }
```

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

之后part2会自动添加在part1之上，我们将part1名称后缀part去掉，解压发现和我们上传的一样

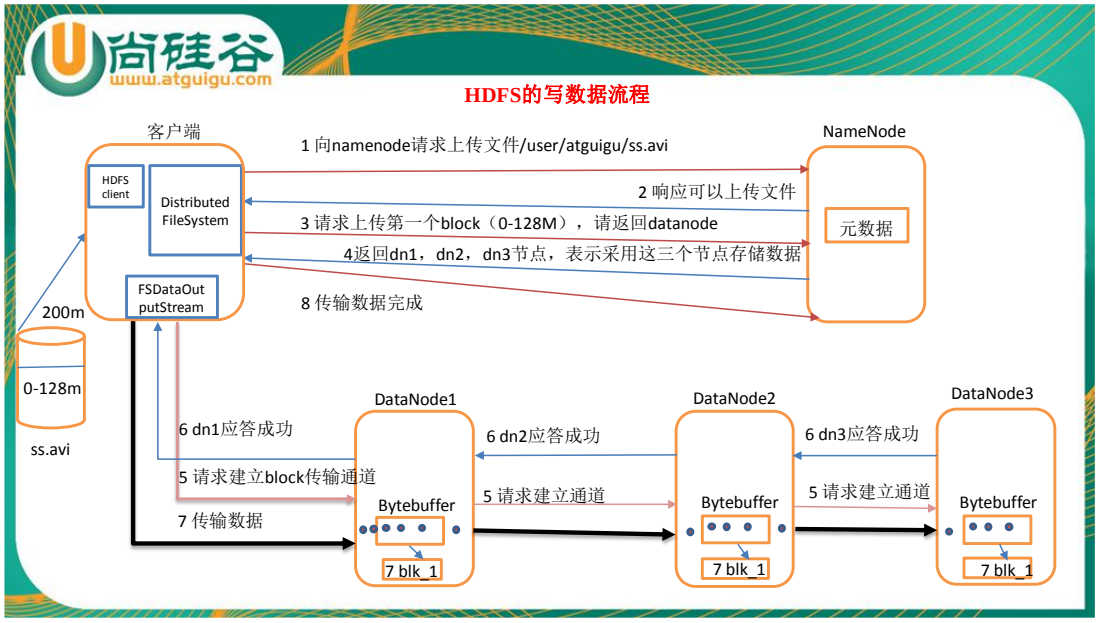
在 window 命令窗口中执行

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

四 HDFS 的数据流

4.1 HDFS 写数据流程

4.1.1 剖析文件写入



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。

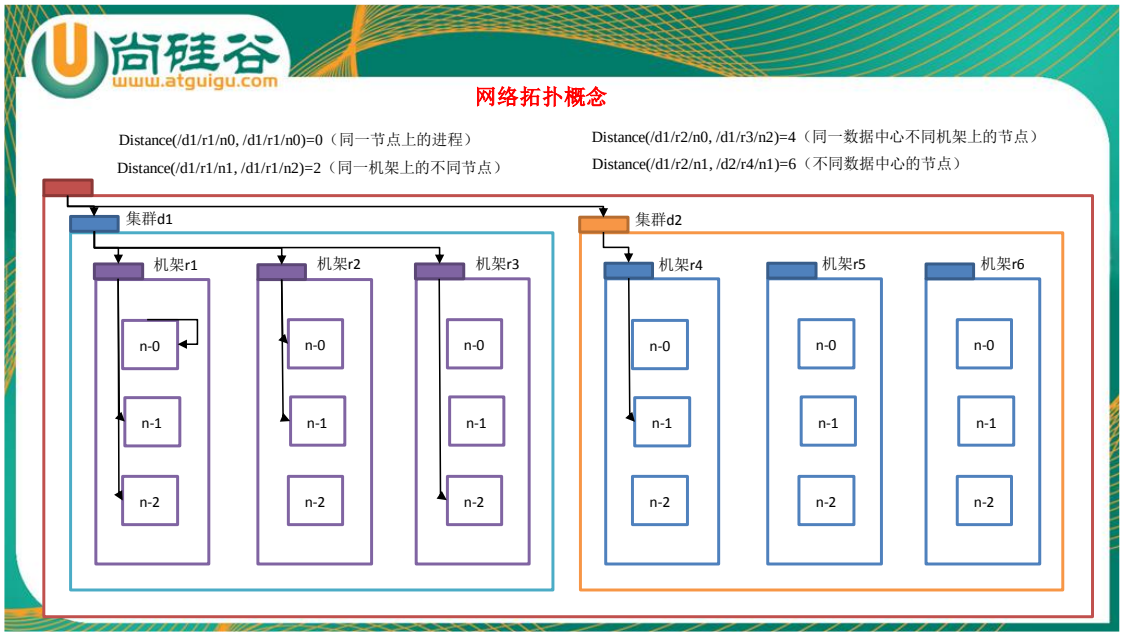
8）当一个 block 传输完成之后，客户端再次请求 namenode 上传第二个 block 的服务器。（重复执行 3-7 步）。

4.1.2 网络拓扑概念

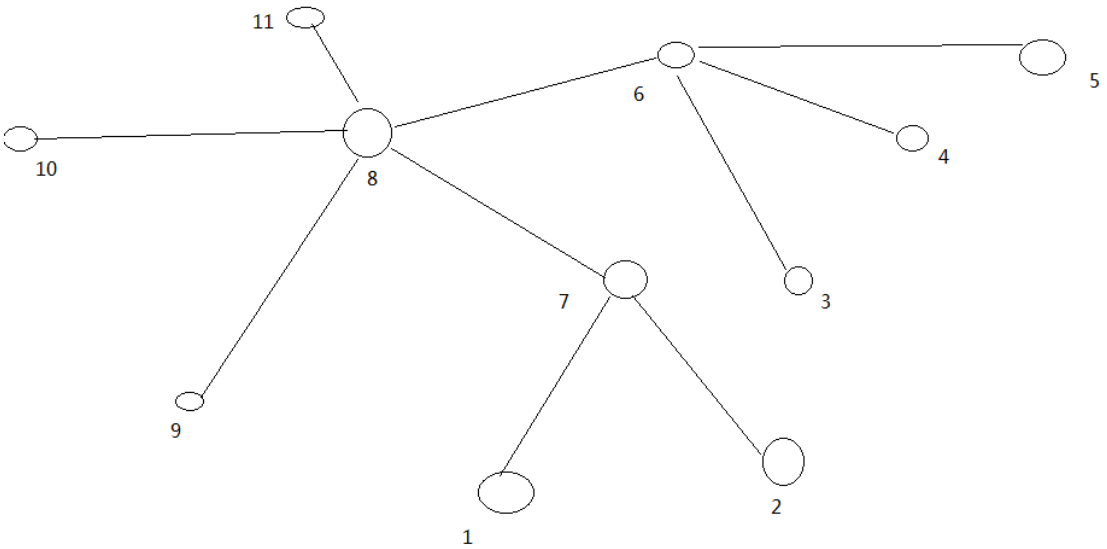
在本地网络中，两个节点被称为“彼此近邻”是什么意思？在海量数据处理中，其主要限制因素是节点之间数据的传输速率——带宽很稀缺。这里的想法是将两个节点间的带宽作为距离的衡量标准。

节点距离：两个节点到达最近的共同祖先的距离总和。

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。



大家算一算每两个节点之间的距离。



4.1.3 机架感知（副本节点选择）

1）官方 ip 地址：

http://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/RackAwareness.html

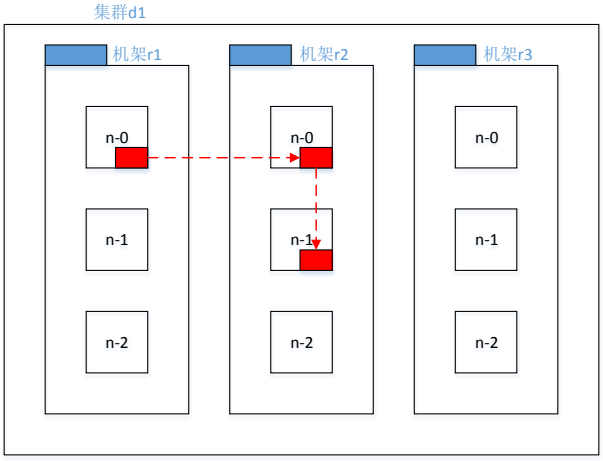
http://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html#Data_Replication

2）低版本 Hadoop 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不相同机架的随机节点上。

第三个副本和第二个副本位于相同机架，节点随机。

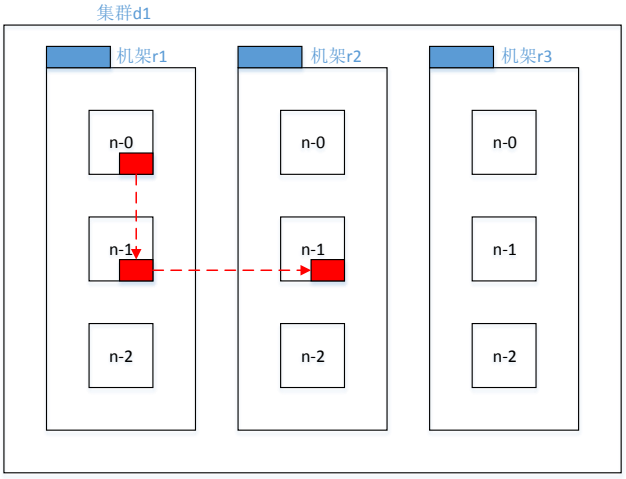


3）Hadoop2.7.2 副本节点选择

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。

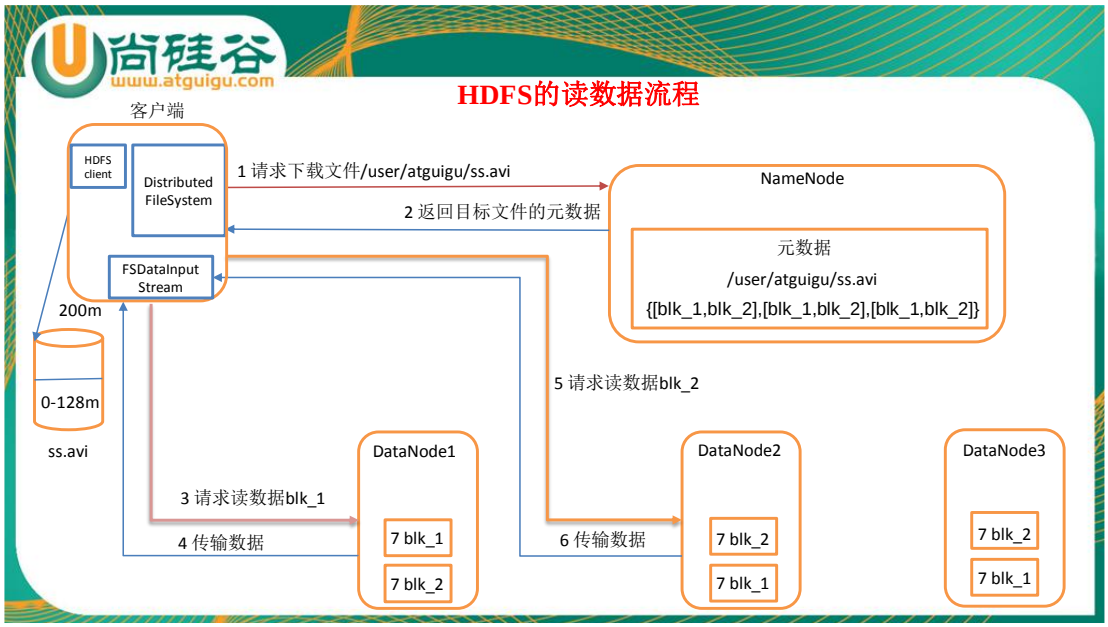
第三个副本位于不同机架，随机节点。



4）HDFS 如何控制客户端读取哪个副本节点数据

HDFS 满足客户端访问副本数据的最近原则。即客户端距离哪个副本数据最近，HDFS 就让哪个节点把数据给客户端。

4.2 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。

4.3 一致性模型

1）debug 调试如下代码

```
    // 向 HDFS 上写数据
    @Test
    public void writeFile() 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 获取输出流
        FSDataOutputStream fos = fs.create(new Path("/hello.txt"));

        // 3 写数据
        fos.write("hello".getBytes());

        // 4 一致性刷新
        fos.hflush();

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

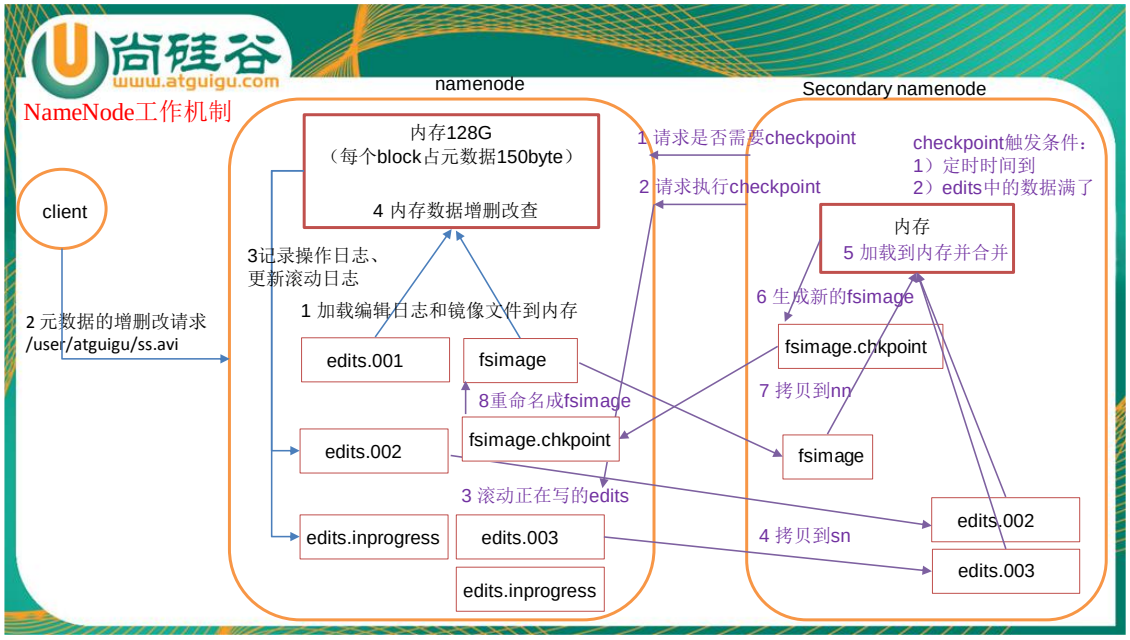
2）总结

写入数据时，如果希望数据被其他 client 立即可见，调用如下方法

FsDataOutputStream. hflush (); //清理客户端缓冲区数据，被其他 client 立即可见

五 NameNode 工作机制

5.1 NameNode&Secondary NameNode 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。

编辑日志：记录客户端对集群的增删改操作，查操作是不走编辑日志的。
镜像文件：集群现存数据的一个镜像记录。我们根据镜像记录可以基本上恢复集群数据。

（2）客户端对元数据进行增删改的请求。

（3）namenode 记录操作日志，更新滚动日志。

（4）namenode 在内存中对数据进行增删改查。

2）第二阶段：Secondary NameNode 工作

（1）Secondary NameNode 询问 namenode 是否需要 checkpoint。直接带回 namenode 是否检查结果。

（2）Secondary NameNode 请求执行 checkpoint。

滚动日志：当前编辑日志执行到哪一步，依次为标志。记录一下编辑日志当前的状态。

（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。

Secondary NameNode将编辑日志和镜像文件进行合并，合并出一个新的镜像文件。然后将新的镜像文件拷贝回NameNode

（7）拷贝 fsimage.chkpoint 到 namenode。

（8）namenode 将 fsimage.chkpoint 重新命名成 fsimage。

5.2 镜像文件和编辑日志文件

1）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

edits_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 idnode 的序列化信息。

（2）Edits 文件：存放 HDFS 文件系统的所有更新操作的路径，文件系统客户端执行的所有写操作首先会被记录到 edits 文件中。

（3）seen_txid 文件保存的是一个数字，就是最后一个 edits_的数字

（4）每次 Namenode 启动的时候都会将 fsimage 文件读入内存，并从 00001 开始到 seen_txid 中记录的数字依次执行每个 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>
```

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>
```

5.3 滚动编辑日志

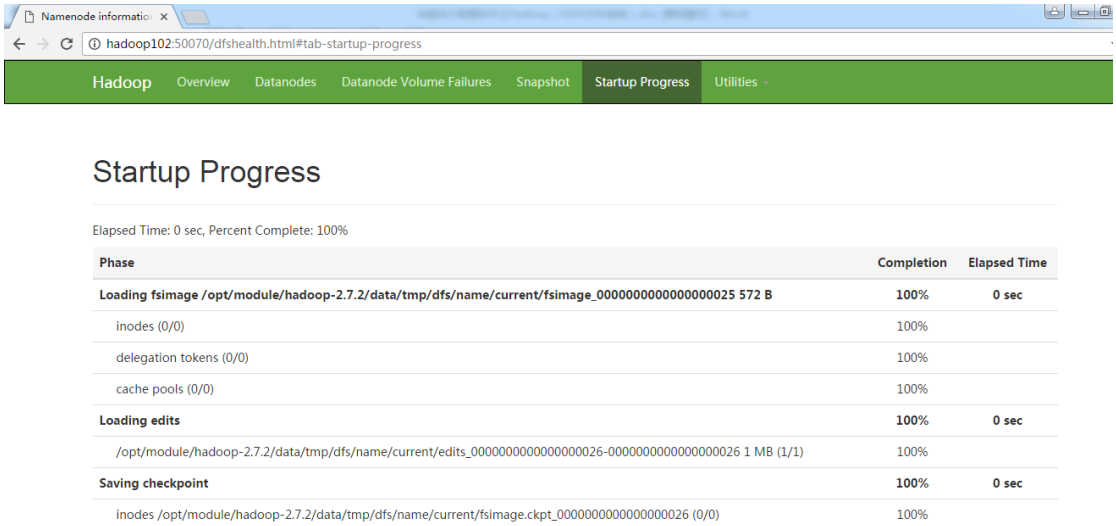
正常情况 HDFS 文件系统有更新操作时，就会滚动编辑日志。也可以用命令强制滚动编辑日志。

1）滚动编辑日志（前提必须启动集群）

```
[atguigu@hadoop102 current]$ hdfs dfsadmin -rollEdits
```

2）镜像文件什么时候产生

Namenode 启动时加载镜像文件和编辑日志



5.4 Namenode 版本号

1）查看 namenode 版本号

在/opt/module/hadoop-2.7.2/data/tmp/dfs/name/current 这个目录下查看 VERSION

namespaceID=1933630176

clusterID=CID-1f2bf8d1-5ad2-4202-af1c-6713ab381175

cTime=0

storageType=NAME_NODE

blockpoolID=BP-97847618-192.168.10.102-1493726072779

layoutVersion=-63

2）namenode 版本号具体解释

（1） namespaceID 在 HDFS 上，会有多个 Namenode，所以不同 Namenode 的 namespaceID 是不同的，分别管理一组 blockpoolID。

（2）clusterID 集群 id，全局唯一

（3）cTime 属性标记了 namenode 存储系统的创建时间，对于刚刚格式化的存储系统，这个属性为 0；但是在文件系统升级之后，该值会更新到新的时间戳。

（4）storageType 属性说明该存储目录包含的是 namenode 的数据结构。

（5）blockpoolID：一个 block pool id 标识一个 block pool，并且是跨集群的全局唯一。当一个新的 Namespace 被创建的时候(format 过程的一部分)会创建并持久化一个唯一 ID。在创建过程构建全局唯一的 BlockPoolID 比人为的配置更可靠一些。NN 将 BlockPoolID 持久化到磁盘中，在后续的启动过程中，会再次 load 并使用。

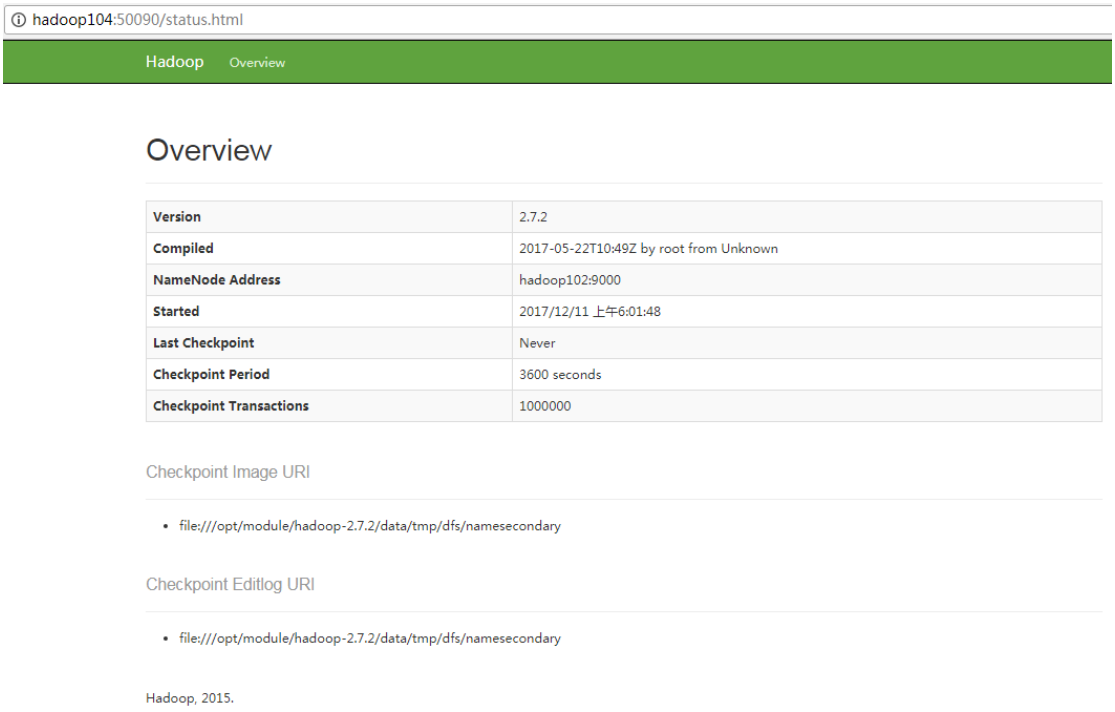
（6）layoutVersion 是一个负整数。通常只有 HDFS 增加新特性时才会更新这个版本号。

5.5 web 端访问 SecondaryNameNode 端口号

（1）启动集群。

（2）浏览器中输入：http://hadoop104:50090/status.html

（3）查看 SecondaryNameNode 信息。



5.6 chkpoint 检查时间参数设置

（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，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>
```

5.7 SecondaryNameNode 目录结构

Secondary NameNode 用来监控 HDFS 状态的辅助后台程序，每隔一段时间获取 HDFS

元数据的快照。

在 /opt/module/hadoop-2.7.2/data/tmp/dfs/namesecondary/current 这个目录中查看 SecondaryNameNode 目录结构。

```
edits_0000000000000000001-0000000000000000002
fsimage_0000000000000000002
fsimage_0000000000000000002.md5
VERSION
```

SecondaryNameNode 的 namesecondary/current 目录和主 namenode 的 current 目录的布局相同。

好处：在主 namenode 发生故障时（假设没有及时备份数据），可以从 SecondaryNameNode 恢复数据。

5.8 Namenode 故障处理方法

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；

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。

5.8.1 手动拷贝 SecondaryNameNode 数据：

模拟 namenode 故障，并采用方法一，恢复 namenode 数据

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
```

5.8.2 采用 importCheckpoint 命令拷贝 SecondaryNameNode 数据

模拟 namenode 故障，并采用方法二，恢复 namenode 数据

0）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>
```

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）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
```

4）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
```

5）启动 namenode

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

5.9 集群安全模式操作

1）概述

Namenode 启动时，首先将映像文件（fsimage）载入内存，并执行编辑日志（edits）中的各项操作。一旦在内存中成功建立文件系统元数据的映像，则创建一个新的 fsimage 文件和一个空的编辑日志。此时，namenode 开始监听 datanode 请求。但是此刻，namenode 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。

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。

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）先进入安全模式

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

2）执行下面的脚本

编辑一个脚本

```
#!/bin/bash
bin/hdfs dfsadmin -safemode wait
bin/hdfs dfs -put ~/hello.txt /root/hello.txt
```

3）再打开一个窗口，执行

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

5.10 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</v
        alue>
</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

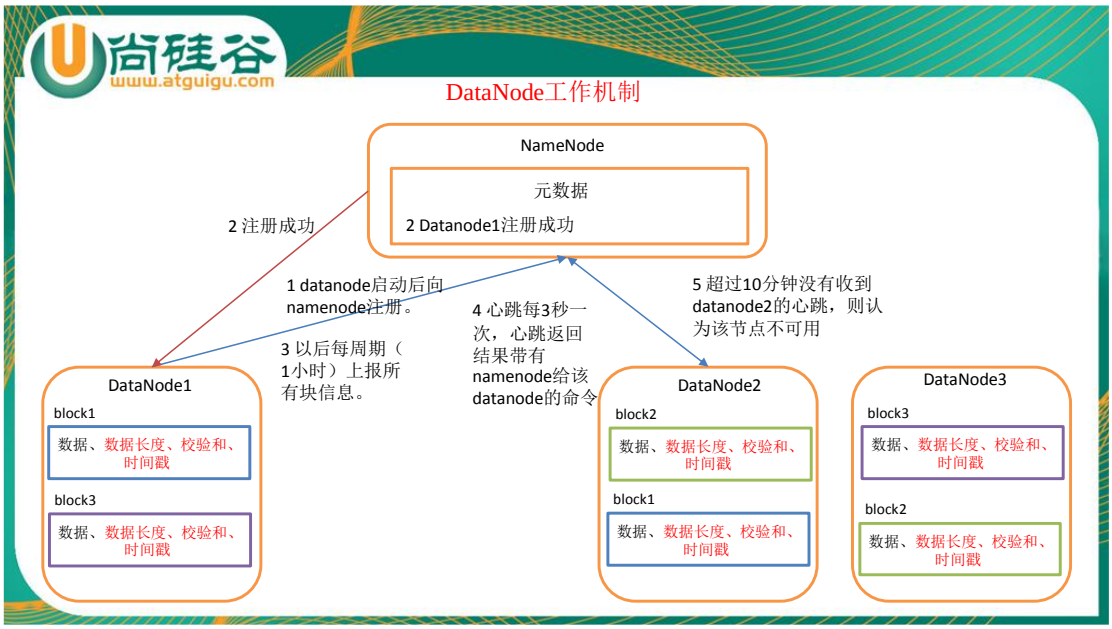
drwx------. 3 atguigu atguigu 4096 12 月 11 08:03 data

drwxrwxr-x. 3 atguigu atguigu 4096 12 月 11 08:03 name1

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

六 DataNode 工作机制

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

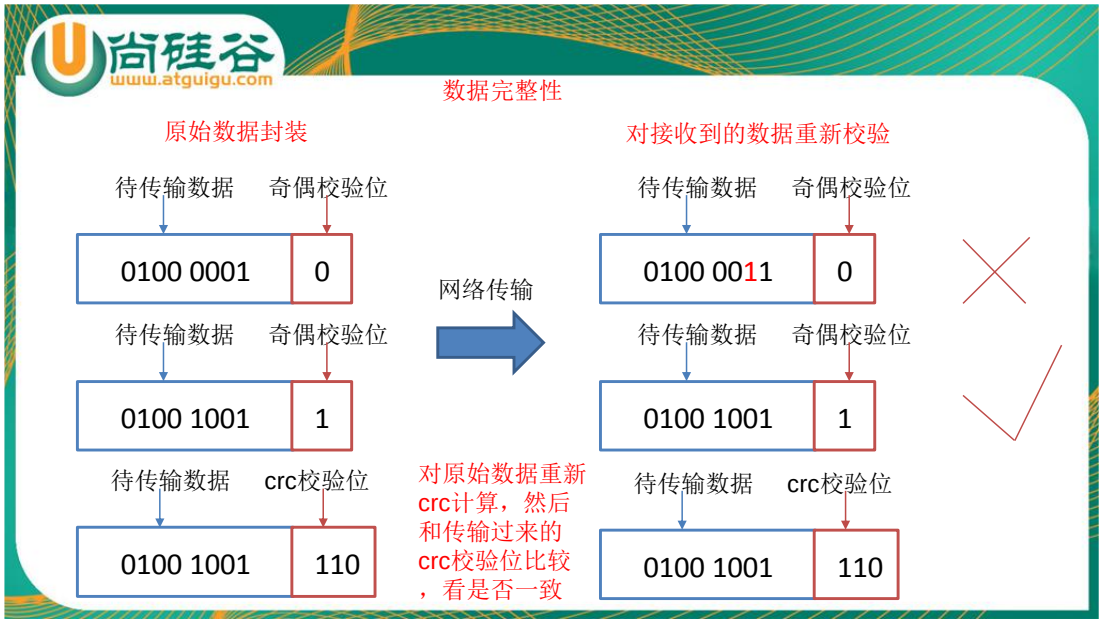
6.2 数据完整性

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



6.3 掉线时限参数设置

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

timeout = 2 * 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 DataNode 的目录结构

和 namenode 不同的是，datanode 的存储目录是初始阶段自动创建的，不需要额外格式化。

1）在/opt/module/hadoop-2.7.2/data/tmp/dfs/data/current 这个目录下查看版本号

[atguigu@hadoop102 current]$ cat VERSION

storageID=DS-1b998a1d-71a3-43d5-82dc-c0ff3294921b

clusterID=CID-1f2bf8d1-5ad2-4202-af1c-6713ab381175

cTime=0

datanodeUuid=970b2daf-63b8-4e17-a514-d81741392165

storageType=DATA_NODE

layoutVersion=-56

2）具体解释

（1）storageID：存储 id 号

（2）clusterID 集群 id，全局唯一

（3）cTime 属性标记了 datanode 存储系统的创建时间，对于刚刚格式化的存储系统，这个属性为 0；但是在文件系统升级之后，该值会更新到新的时间戳。

（4）datanodeUuid：datanode 的唯一识别码

（5）storageType：存储类型

（6）layoutVersion 是一个负整数。通常只有 HDFS 增加新特性时才会更新这个版本号。

3）在 /opt/module/hadoop-2.7.2/data/tmp/dfs/data/current/BP-97847618-192.168.10.102-1493726072779/current 这个目录下查看该数据块的版本号

[atguigu@hadoop102 current]$ cat VERSION

#Mon May 08 16:30:19 CST 2017

namespaceID=1933630176

cTime=0

blockpoolID=BP-97847618-192.168.10.102-1493726072779

layoutVersion=-56

4）具体解释

（1）namespaceID：是 datanode 首次访问 namenode 的时候从 namenode 处获取的 storageID 对每个 datanode 来说是唯一的（但对于单个 datanode 中所有存储目录来说则是相同的），namenode 可用这个属性来区分不同 datanode。

（2）cTime 属性标记了 datanode 存储系统的创建时间，对于刚刚格式化的存储系统，这个属性为 0；但是在文件系统升级之后，该值会更新到新的时间戳。

（3）blockpoolID：一个 block pool id 标识一个 block pool，并且是跨集群的全局唯一。当一个新的 Namespace 被创建的时候(format 过程的一部分)会创建并持久化一个唯一 ID。在创建过程构建全局唯一的 BlockPoolID 比人为的配置更可靠一些。NN 将 BlockPoolID 持久化到磁盘中，在后续的启动过程中，会再次 load 并使用。

（4）layoutVersion 是一个负整数。通常只有 HDFS 增加新特性时才会更新这个版本号。

6.5 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）克隆一台虚拟机

（2）修改 ip 地址和主机名称

（3）修改 xcall 和 xsync 文件，增加新`增节点的同步 ssh

（4）删除原来 HDFS 文件系统留存的文件

/opt/module/hadoop-2.7.2/data

2）服役新节点具体步骤

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

[atguigu@hadoop105 hadoop]$ pwd

/opt/module/hadoop-2.7.2/etc/hadoop

[atguigu@hadoop105 hadoop]$ touch dfs.hosts

[atguigu@hadoop105 hadoop]$ vi dfs.hosts

添加如下主机名称（包含新服役的节点）

hadoop102

hadoop103

hadoop104

hadoop105

（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）刷新 namenode

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

Refresh nodes successful

（4）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

（5）在 namenode 的 slaves 文件中增加新主机名称

增加 105　不需要分发

hadoop102

hadoop103

hadoop104

hadoop105

（6）单独命令启动新的数据节点和节点管理器

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

starting datanode, logging to /opt/module/hadoop-2.7.2/logs/hadoop-atguigu-datanode-hadoop105.out

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

starting nodemanager, logging to /opt/module/hadoop-2.7.2/logs/yarn-atguigu-nodemanager-hadoop105.out

（7）在 web 浏览器上检查是否 ok

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

6.6 退役旧数据节点

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

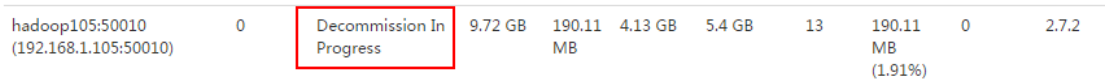
[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfsadmin -refreshNodes

Refresh nodes successful

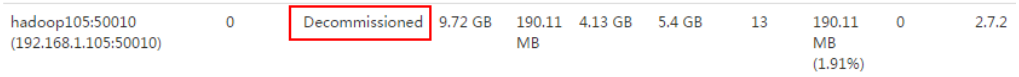
[atguigu@hadoop102 hadoop-2.7.2]$ yarn rmadmin -refreshNodes

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。



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。 •



hadoop105:50010 (192.168.1.105:50010)	0	Decommissioned	9.72 GB	190.11 MB	4.13 GB	5.4 GB	13	190.11 MB (1.91%)	0	2.7.2

[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）从 include 文件中删除退役节点，再运行刷新节点的命令

（1）从 namenode 的 dfs.hosts 文件中删除退役节点 hadoop105

hadoop102

hadoop103

hadoop104

（2）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

7）从 namenode 的 slave 文件中删除退役节点 hadoop105

hadoop102

hadoop103

hadoop104

8）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

6.7 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.tmp.dir}/dfs/data2</value>
</property>
```

七 HDFS 其他功能

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 discp 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 Hadoop 存档

1）理论概述

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 hadoop 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存储这些文件原始内容所需要的磁盘空间相比也不会增多。例如，一个 1MB 的文件以大小为 128MB 的块存储，使用的是 1MB 的磁盘空间，而不是 128MB。

Hadoop 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，Hadoop 存档文件可以用作 MapReduce 的输入。

2）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

归档成一个叫做 xxx.har 的文件夹，该文件夹下有相应的数据文件。Xx.har 目录是一个整体，该目录看成是一个归档文件即可。

[atguigu@hadoop102 hadoop-2.7.2]$ bin/hadoop archive -archiveName myhar.har -p /user/atguigu /user/my

（3）查看归档

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -lsr /user/my/myhar.har

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -lsr har:///myhar.har

（4）解归档文件

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -cp har:/// user/my/myhar.har /* /user/atguigu

7.3 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，而是指向同一个文件。当写入发生时，才会产生新文件。

1）基本语法

（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/data

[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfsadmin -disallowSnapshot

/user/atguigu/data

（2）对目录创建快照

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

通过 web 访问 hdfs://hadoop102:9000/user/atguigu/data/.snapshot/s…..// 快照和源文件使用相同数据块

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

（3）指定名称创建快照

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

（4）重命名快照

[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfs -renameSnapshot /user/atguigu/data/ 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/data/ . .snapshot/atguigu170508

（7）恢复快照

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

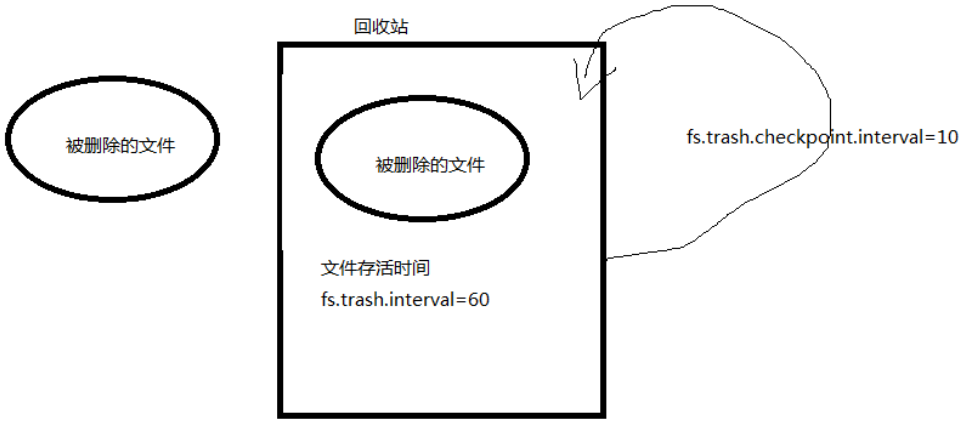
7.4 回收站

1）默认回收站

默认值 fs.trash.interval=0，0 表示禁用回收站，可以设置删除文件的存活时间。

默认值 fs.trash.checkpoint.interval=0，检查回收站的间隔时间。

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



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-2.7.2]$ hadoop fs -mv /user/atguigu/.Trash/Current/user/atguigu/input /user/atguigu/input

7）清空回收站

[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfs -expunge

八 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

1）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。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。

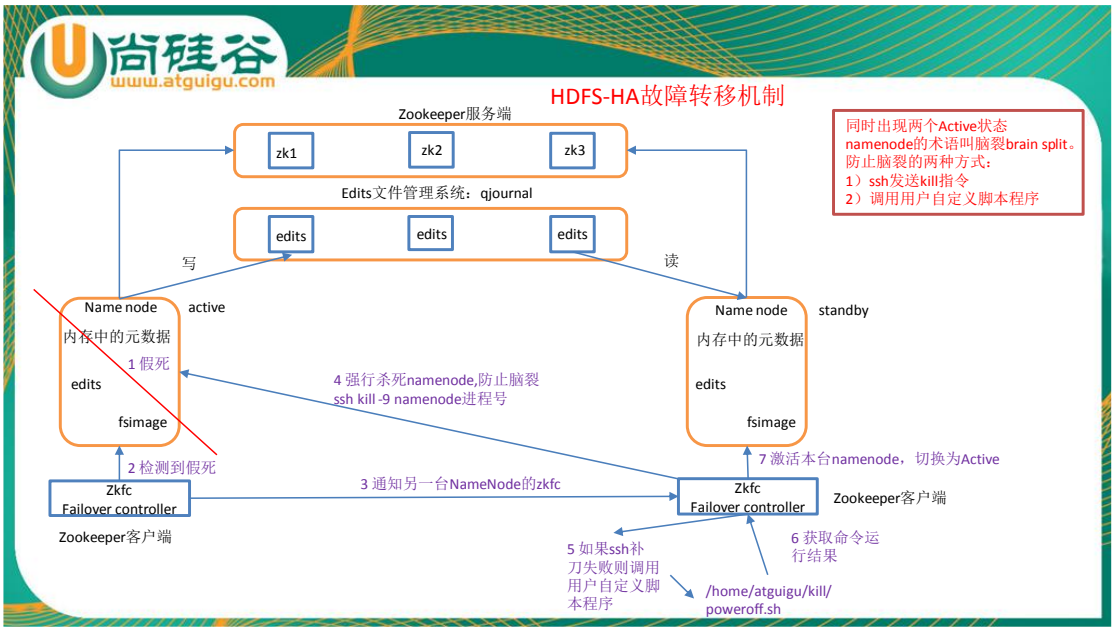
2）现役 NameNode 选择：ZooKeeper 提供了一个简单的机制用于唯一的选择一个节点为 active 状态。如果目前现役 NameNode 崩溃，另一个节点可能从 ZooKeeper 获得特殊的排外锁以表明它应该成为现役 NameNode。

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。



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

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

0）集群规划

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

1）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

2）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。

3）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;hadoop104:8485/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
```

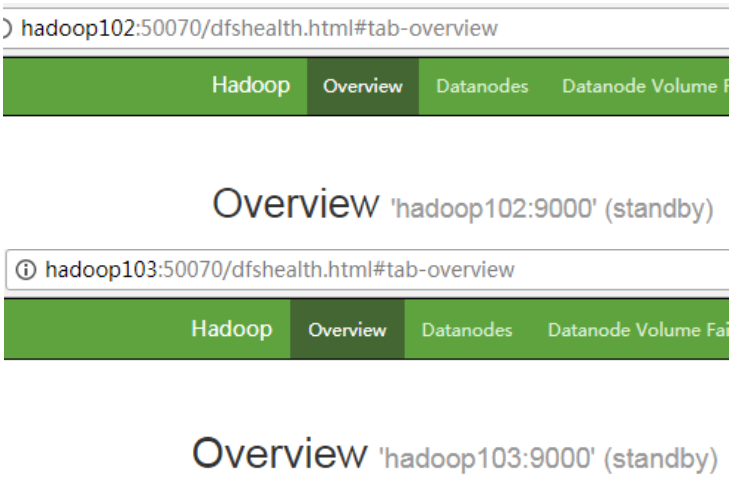
3）在[nn2]上，同步 nn1 的元数据信息：

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

4）启动[nn2]：

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

5）查看 web 页面显示



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

8）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

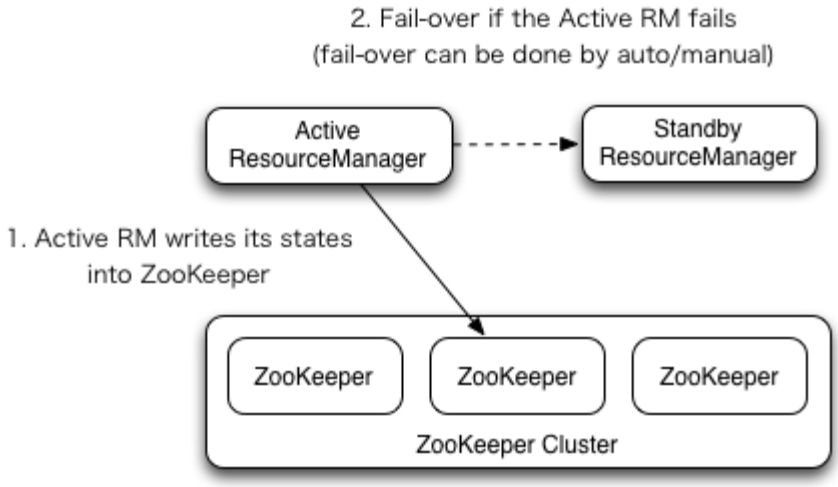
8.4 YARN-HA 配置

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



8.4.2 配置 YARN-HA 集群

0）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

1）规划集群

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

ResourceManager	ResourceManager	
NodeManager	NodeManager	NodeManager

2）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.recovery.ZKRMStateStore<
        /value>
    </property>

</configuration>
```

（2）同步更新其他节点的配置信息

3）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
```

4）启动 yarn

（1）在 hadoop102 中执行：

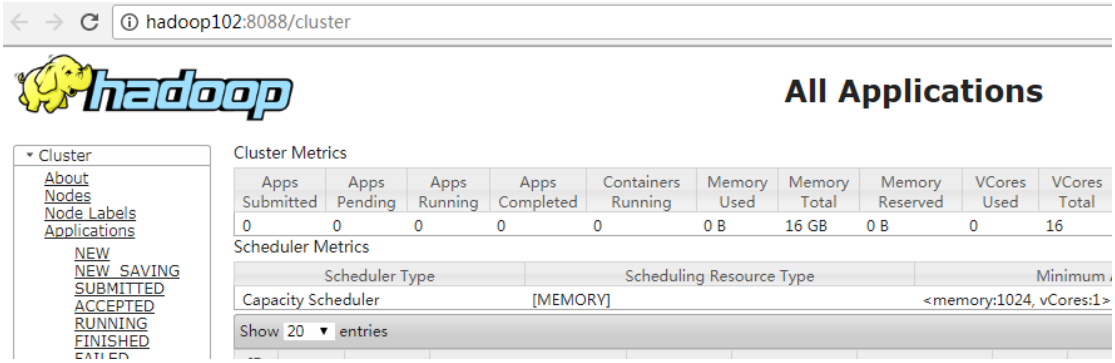
```
sbin/start-yarn.sh
```

（2）在 hadoop103 中执行：

sbin/yarn-daemon.sh start resourcemanager

（3）查看服务状态

bin/yarn rmadmin -getServiceState rm1



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

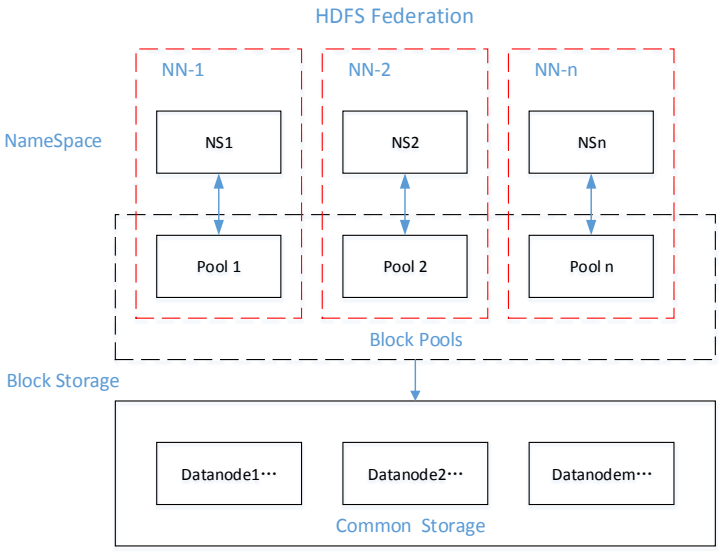
由于是单个 namenode 的 HDFS 架构，因此整个 HDFS 文件系统的吞吐量受限于单个 namenode 的吞吐量。

2）HDFS Federation 架构设计

能不能有多个 NameNode

NameNode	NameNode	NameNode
元数据	元数据	元数据

Log machine 电商数据/话单数据



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）