HDFS分布式文件系统

1 HDFS概述

- **HDFS定义**

  HDFS（Hadoop Distributed File System）hadoop分布式文件系统，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。

- **HDFS优点**

  1. 高容错性
     - 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。
     - 某一个副本丢失以后，它可以自动恢复。
  2. 适合处理大数据
     - 数据规模：能够处理数据规模达到GB、TB、甚至PB级别的数据；
     - 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。

- **HDFS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。
     - 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；
     - 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。
     - 一个文件只能有一个写，不允许多个线程同时写；
     - 仅支持数据append（追加），不支持文件的随机修改。

- **HDFS组成架构**

  1. NameNode（nn）：就是Master，它是一个主管、管理者。

     负责执行有关 `文件系统命名空间` 的操作，例如打开，关闭、重命名文件和目录等。它同时还负责集群元数据的存储，记录着文件中各个数据块的位置信息。
     - 管理HDFS的名称空间
     - 配置副本策略
     - 管理数据块（Block）映射信息
     - 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。

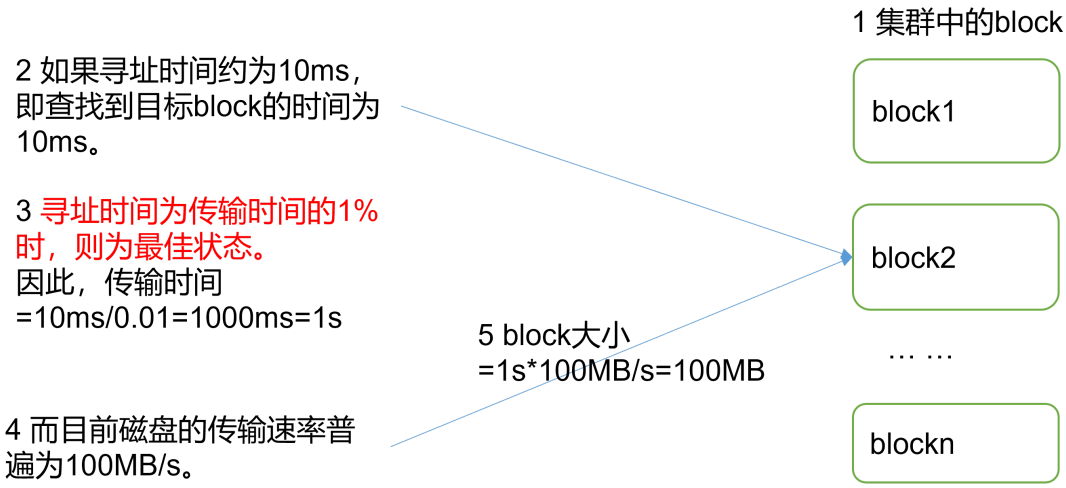
     负责提供来自文件系统客户端的读写请求，执行块的创建，删除等操作。
     - 存储实际的数据块
     - 执行数据块的读/写操作
  3. Client：就是客户端。
     - 文件切分。文件上传HDFS的时候，Client将文件切分成一个一个的Block，然后进行上传
     - 与NameNode交互，获取文件的位置信息

- 与DataNode交互，读取或者写入数据
- Client提供一些命令来管理HDFS，比如NameNode格式化
- 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

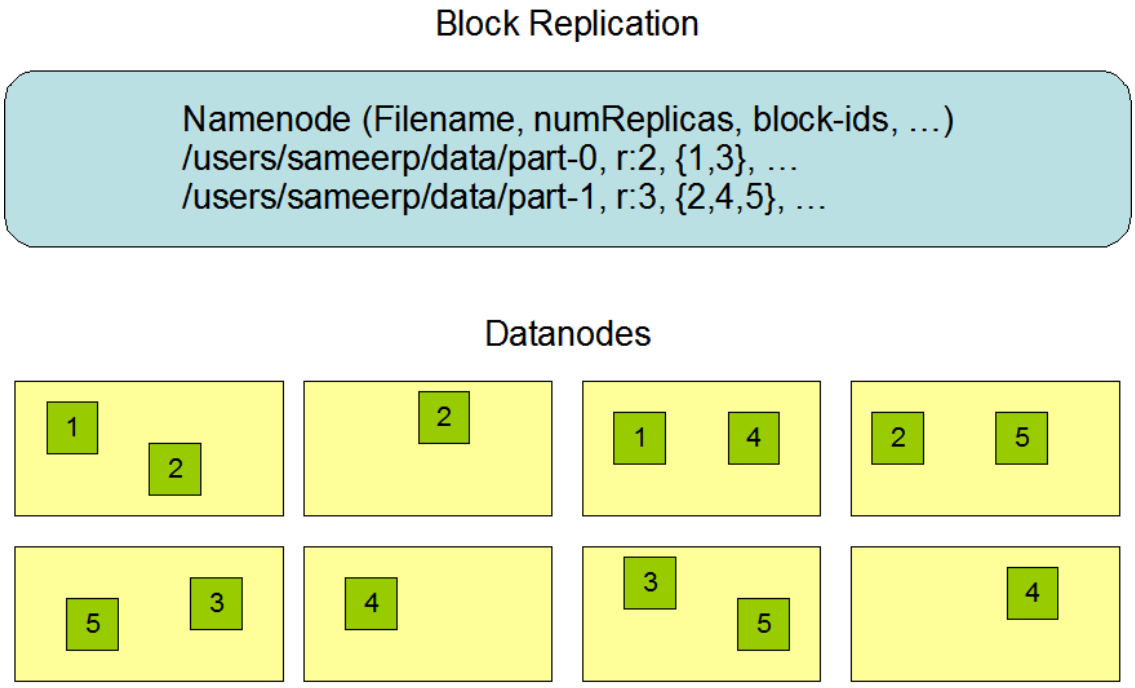
- 辅助NameNode，分担其工作量，比如定期合并Fsimage和Edits，并推送给NameNode
- 在紧急情况下，可辅助恢复NameNode

- **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。



- **数据复制及其实现原理**

由于 Hadoop 被设计运行在廉价的机器上，这意味着硬件是不可靠的，为了保证容错性，HDFS 提供了数据复制机制。HDFS 将每一个文件存储为一系列**块**，每个块由多个副本来保证容错，块的大小和复制因子可以自行配置（默认情况下，块大小是 128M，默认复制因子是 3）。



数据复制的实现原理

大型的 HDFS 实例在通常分布在多个机架的多台服务器上，不同机架上的两台服务器之间通过交换机进行通讯。在大多数情况下，同一机架中的服务器间的网络带宽大于不同机架中的服务器之间的带宽。因此 HDFS 采用机架感知副本放置策略，对于常见情况，当复制因子为 3 时，HDFS 的放置策略是：

在写入程序位于 `datanode` 上时，就优先将写入文件的一个副本放置在该 `datanode` 上，否则放在随机 `datanode` 上。之后在另一个远程机架上的任意一个节点上放置另一个副本，并在该机架上的另一个节点上放置最后一个副本。此策略可以减少机架间的写入流量，从而提高写入性能。

如果复制因子大于 3，则随机确定第 4 个和之后副本的放置位置，同时保持每个机架的副本数量低于上限，上限值通常为 `（复制系数 - 1）/机架数量 + 2`，需要注意的是不允许同一个 `dataNode` 上具有同一个块的多个副本。

- **架构的稳定性**

  1. 心跳机制和重新复制

     每个 DataNode 定期向 NameNode 发送心跳消息，如果超过指定时间没有收到心跳消息，则将 DataNode 标记为死亡。NameNode 不会将任何新的 IO 请求转发给标记为死亡的 DataNode，也不会再使用这些 DataNode 上的数据。 由于数据不再可用，可能会导致某些块的复制因子小于其指定值，NameNode 会跟踪这些块，并在必要的时候进行重新复制。

  2. 数据的完整性

     由于存储设备故障等原因，存储在 DataNode 上的数据块也会发生损坏。为了避免读取到已经损坏的数据而导致错误，HDFS 提供了数据完整性校验机制来保证数据的完整性，具体操作如下：

     当客户端创建 HDFS 文件时，它会计算文件的每个块的 `校验和`，并将 `校验和` 存储在同一 HDFS 命名空间下的单独的隐藏文件中。当客户端检索文件内容时，它会验证从每个 DataNode 接收的数据是否与存储在关联校验和文件中的 `校验和` 匹配。如果匹配失败，则证明数据已经损坏，此时客户端会选择从其他 DataNode 获取该块的其他可用副本。

  3. 元数据的磁盘故障

     `FsImage` 和 `EditLog` 是 HDFS 的核心数据，这些数据的意外丢失可能会导致整个 HDFS 服务不可用。为了避免这个问题，可以配置 NameNode 使其支持 `FsImage` 和 `EditLog` 多副本同步，这样 `FsImage` 或 `EditLog` 的任何改变都会引起每个副本 `FsImage` 和 `EditLog` 的同步更新。

  4. 支持快照

     快照支持在特定时刻存储数据副本，在数据意外损坏时，可以通过回滚操作恢复到健康的数据状态。

- **思考：为什么块的大小不能设置太小，也不能设置太大?**
  - 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
  - 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

---

2 HDFS的Shell操作

2.1 hdfs命令

- 基本语法

```
hadoop fs 具体命令
# OR
hdfs dfs 具体命令
```

以上两个命令是完全相同的。

- 命令大全

```
$ bin/hadoop fs

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

2.2 上传

首先需要启动 `hdsf`

```
start-dfs.sh
```

在浏览器输入 `hadoop102:9870` 访问 `hdfs` 网页，创建 `sanguo` 目录

1. -copyFromLocal：从本地文件系统中拷贝文件到HDFS路径去

   ```
   vim liubei.txt # 写入相关文字
   hadoop fs -copyFromLocal ./liubei.txt /sanguo # 执行后可在网页查看
   ```

2. -moveFromLocal：从本地剪切粘贴到HDFS

   ```
   vim guanyu.txt # 写入相关文字
   hadoop fs -moveFromLocal ./guanyu.txt /sanguo  # 执行后可在网页查看，本地文件上传后就被删除了
   ```

3. -appendToFile：追加一个文件到已经存在的文件末尾

   ```
   vim zhangfei.txt # 写入相关文字
   hadoop fs -appendToFile ./zhangfei /sanguo/liubei.txt
   # 将zhangfei.txt内容追加到liubei.txt中
   ```

4. -put：等同于copyFromLocal

   ```
   hadoop fs -put ./zhangfei.txt /sanguo
   ```

2.3 下载

1. -copyToLocal：从HDFS拷贝到本地

   ```
   hadoop fs -copyToLocal /sanguo/guanyu.txt ./
   # 将guanyu.txt删除，方便下面测试
   rm -rf guanyu.txt
   ```

2. -get：等同于copyToLocal，就是从HDFS下载文件到本地

   ```
   hadoop fs -get /sanguo/guanyu.txt ./
   ```

3. -getmerge：合并下载多个文件，比如HDFS的目录 `/sanguo` 下有多个文件:`liubei.txt`,`guanyu.txt`,`zhangfei.txt`,下载到本地`xiongdi.txt`

   ```
   hadoop fs -getmerge /sanguo/liubei.txt /sanguo/guanyu.txt
   /sanguo/zhagfei.txt ./xiongdi.txt
   ```

2.4 HDFS直接操作

1. -ls: 显示目录信息

```
hadoop fs -ls /
```

2. -mkdir：在HDFS上创建目录

```
hadoop fs -mkdir /xiyou # 创建目录
hadoop fs -mkdir -p /shuihu/liangsan #创建多层目录需要加上 -p
```

3. -cat：显示文件内容

```
hadoop fs -cat /sanguo/guanyu.txt
```

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

```
hadoop fs  -chmod  666 /sanguo/guanyu.txt
hadoop fs  -chown  xu1an:xu1an   /sanguo/zhangfei.txt
```

5. -cp ： 从HDFS的一个路径拷贝到HDFS的另一个路径

```
hadoop fs -cp /sanguo/zhangfei.txt /xiyou # 将zhangfei.txt复制到xiyou下
```

6. -mv：在HDFS目录中移动文件

```
hadoop fs -mv /sanguo/guanyu.txt /xiyou # 将guanyu.txt移动到xiyou下
hadoop fs -mv /sanguo/liubei.txt /sanguo/zhugong.txt #将liubei.txt改名为
zhugong.txt
```

7. -tail：显示一个文件的末尾1kb的数据

```
hadoop fs -tail /sanguo/zhangfei.txt # 显示头部 不支持 -n
hadoop fs -head /sanhuo/zhangfei.txt # 显示尾部
```

8. -rm：删除文件或文件夹

```
hadoop fs -rm /xiyou/zhangfei.txt # 删除文件
hadoop fs -rm -r /xiyou # 删除文件夹
```

9. -rmdir：删除空目录

```
hadoop fs -rmdir /shuihu/liangsan
```

10. -du统计文件夹的大小信息

```
hadoop fs -du /
hadoop fs -du -h / #有单位 52  156  /sanguo 三倍关系：有三个副本
```

11. -setrep：设置HDFS中文件的副本数量

```
hadoop fs -setrep 6 /sanguo/zhangfei.txt # 设置副本数为6
```

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。

---

3 HDFS客户端操作

3.1 准备Windows开发环境

1. 配置环境变量
2. 创建Maven项目，并导入相应的依赖

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

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

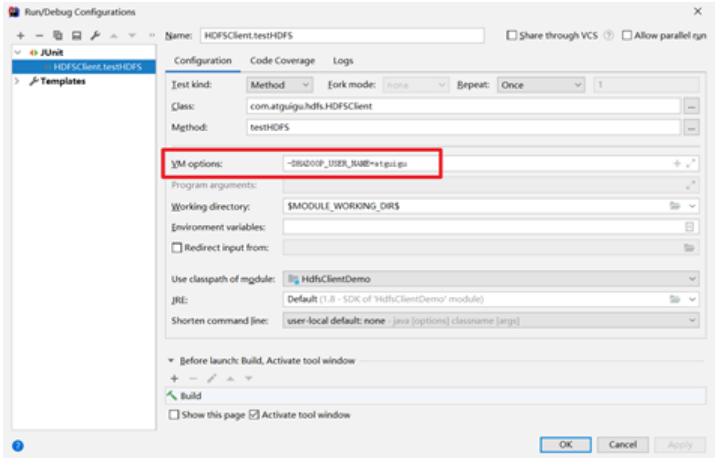
```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="error" strict="true" name="XMLConfig">
    <Appenders>
        <!-- 类型名为Console，名称为必须属性 -->
        <Appender type="Console" name="STDOUT">
            <!-- 布局为PatternLayout的方式，
            输出样式为[INFO] [2018-01-22 17:34:01][org.test.Console]I'm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

    </Appenders>

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

        <!-- root loggerConfig设置 -->
```

```
        <Root level="info">
            <AppenderRef ref="STDOUT" />
        </Root>
    </Loggers>
</Configuration>
```

4. 创建com.xu1an.hdfs.HdfsClinet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://hadoop102:9820");
        // FileSystem fs = FileSystem.get(configuration);

        FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9820"),
configuration, "xu1an");

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

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

5. 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=xu1an，xu1an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(){
    //1、 获取文件系统
    Configuration conf = new Configuration();
    conf.set("dfs.replication","5");
    FileSystem fs = FileSystem.get(new URI(hdfs://hadoop:9820), conf,
"xu1an" );
    //2、上传文件
    fs.copyFromLocalFile(false,true,new
Path("D:\\web\\bigdataDemo\\src\\main\\resources\\hello.txt"),new
Path("/client_test"));
     //3、 关闭资源
     fs.close;

}
```

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

   测试配置的优先级`Confrguration` > `hdfs-site.xml` > `hdfs-default.xml`

3.2.2 HDFS文件下载

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

        // 1 获取文件系统
        Configuration configuration = new Configuration();
        FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9820"),
configuration, "xu1an");

        // 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删除文件和目录

`recursive=true`递归删除目录路径下的文件夹和文件

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

    // 1 获取文件系统
    Configuration configuration = new Configuration();
    FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9820"),
configuration, "xu1an");

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

    // 3 关闭资源
    fs.close();
}
```

3.2.4 HDFS文件更名和移动

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

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

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

    // 3 关闭资源
    fs.close();
}
```

3.2.5 HDFS文件详情查看

`recursive=true`递归查看目录路径及其子路径下文件详情

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

    // 1获取文件系统
    Configuration configuration = new Configuration();
    FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9820"),
configuration, "xu1an");

    // 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

只会判读同个目录下的文件或文件夹

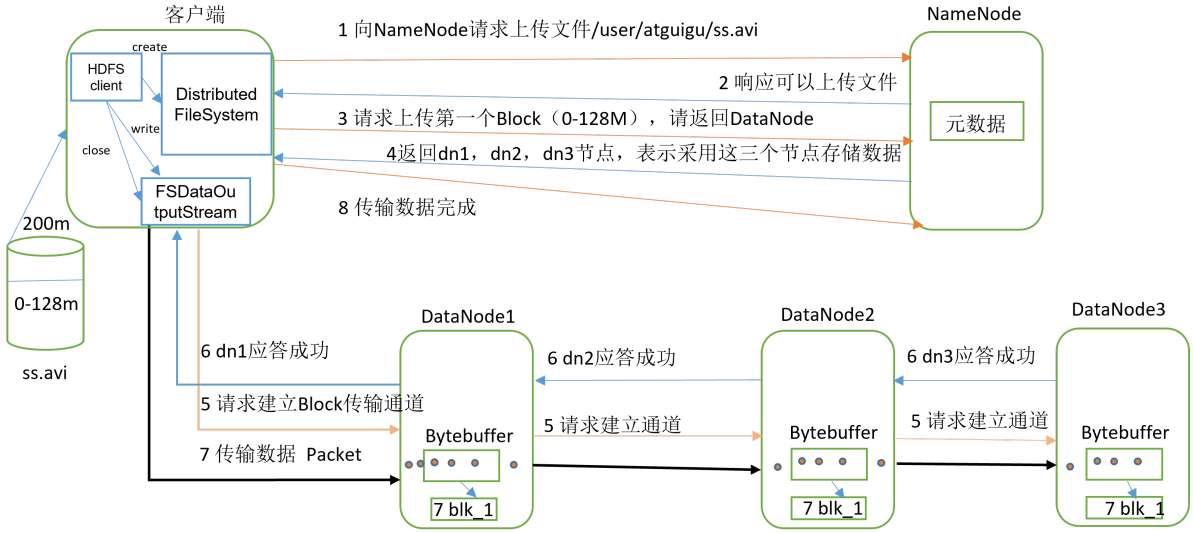
```
@Test
public void testListStatus() throws IOException, InterruptedException,
URISyntaxException{

    // 1 获取文件配置信息
    Configuration configuration = new Configuration();
    FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9820"),
configuration, "xu1an");

    // 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();
}
```

4 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）。

源码解析：org.apache.hadoop.hdfs.DFSOutputStream

- HDFS根据请求返回DataNode的节点的策略？-- 机架感知

  如果当前Client所在机器有DataNode节点，那就返回当前机器DN1,否则从集群中随机一台。根据第一台机器的位置，然后再其他机架上随机一台，在第二台机器所在机架上再随机一台。 以上策略的缘由：为了提高数据的可靠性，同时一定程度也保证数据传输的效率！

- 客户端建立传输通道的时候如何确定和哪一台DataNode先建立连接？-- 网络拓扑

找离client最近的一台机器先建立通道。

- Client为什么是以串行的方式建立通道?

  本质上就是为了降低client的IO开销

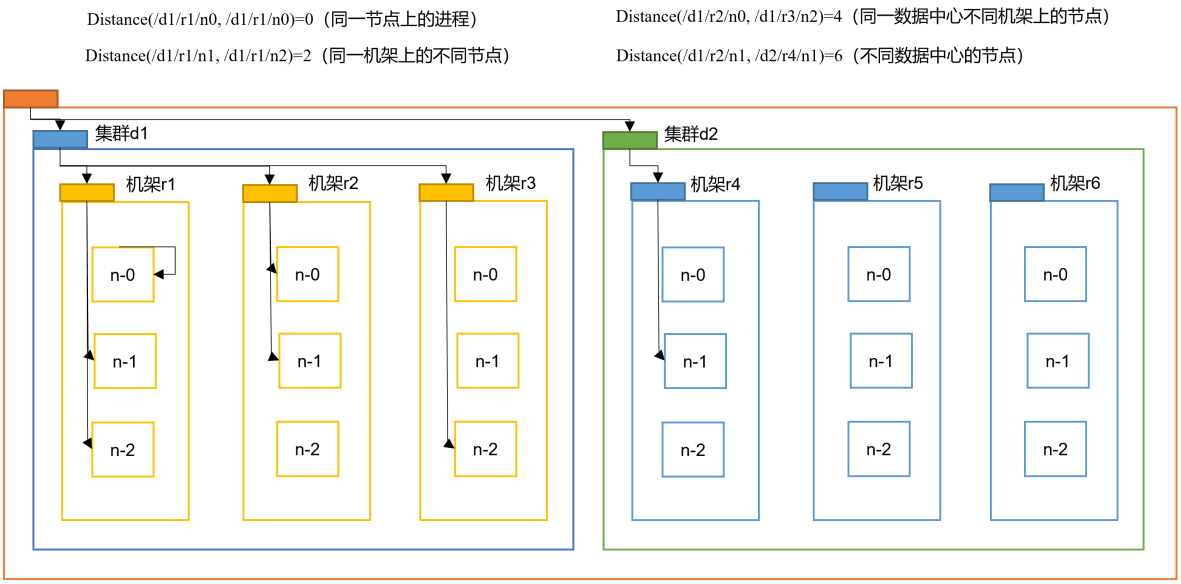
- 数据传输的时候如何保证数据成功?

  采用了ack回执的策略保证了数据完整成功上传。

4.1.2 网络拓扑-节点距离计算

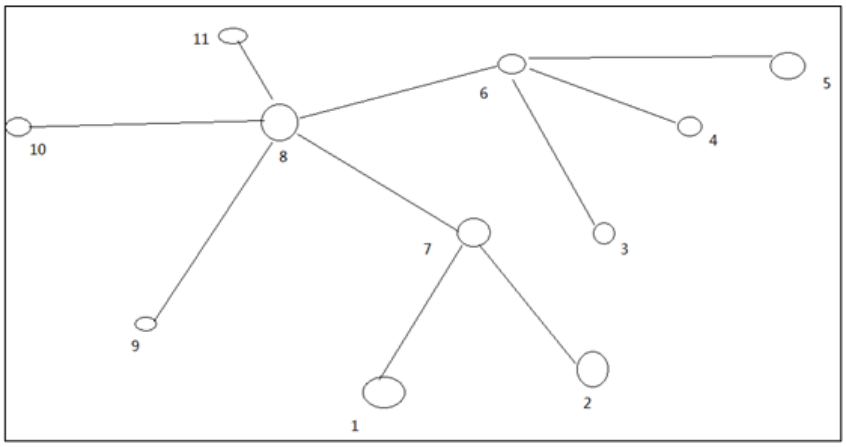
在HDFS写数据的过程中，NameNode会选择距离待上传数据最近距离的DataNode接收数据。那么这个最近距离怎么计算呢?

节点距离：两个节点到达最近的共同祖先的距离总和。



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。

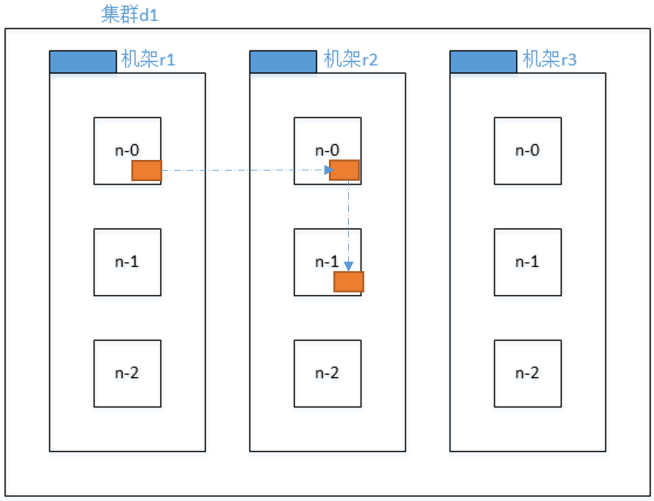


4.1.3 机架感知（副本存储节 点选择）

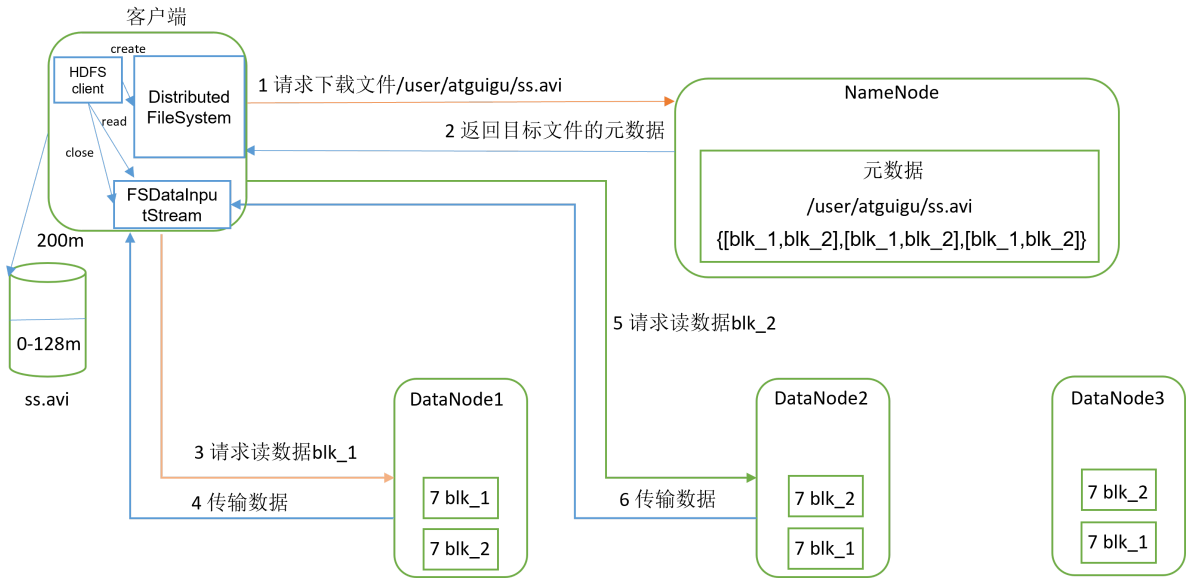
第一个副本在Client所处的节点上。如果客户端在集群外，随机选一个。

第二个副本在另一个机架的随机一个节点

第三个副本在第二个副本所在机架的随机节点



4.2 HDFS读数据流程



（1）客户端通过DistributedFileSystem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

---

5 NN和2NN

5.1 NN和2NN工作机制

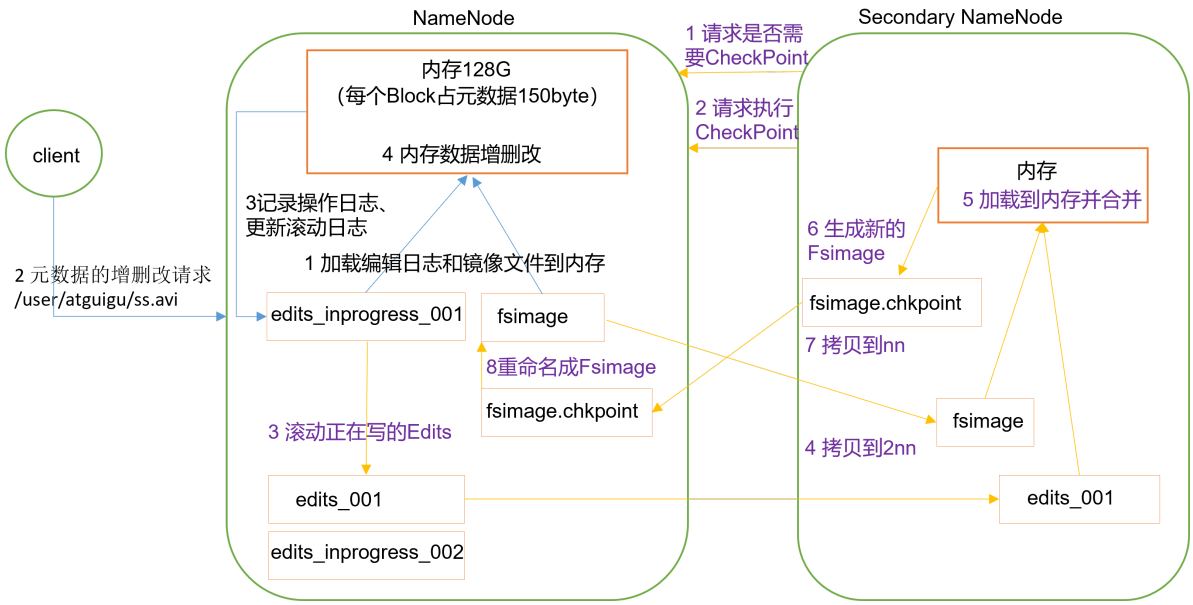
思考：NameNode中的元数据是存储在哪里的？

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。

```
1. 元数据信息要保存在哪？
   1.1 保存到磁盘
       -- 不足：读写速度慢 效率低！
   1.2 保存内存
       -- 不足：数据不安全
   1.3 最终的解决方案： 磁盘 + 内存

2. 内存中的元数据和磁盘中的元数据如何进行同步。（元数据的维护策略）
       当我们对元数据进行操作的时候，首先在内存进行合并，其次还要把相关操作记录追加到edits编辑日志文件中，在满足一定条件下，将edits文件中的记录合并到元数据信息文件中 fsimage

3. 谁负责对NN的元数据信息进行合并？
       2NN主要负责对NN的元数据精心合并，当满足一定条件的下，2NN会检测本地时间，每隔一个小时会主动对NN的edits文件和fsimage文件进行一次合并。合并的时候，首先会通知NN,这时候NN就会停止对正在使用的edits文件的追加，同时会新建一个新的edits编辑日志文件，保证NN的正常工作。接下来 2NN会把NN本地的fsimage文件和edits编辑日志拉取2NN的本地，在内存中对二者进行合并，最后产生最新fsimage文件。把最新的fsimage文件再发送给NN的本地。注意还有一个情况，当NN的edits文件中的操作次数累计达到100万次，即便还没到1小时，2NN（每隔60秒会检测一次NN方的edits文件的操作次数）也会进行合并。
       2NN 也会自己把最新的fsimage文件备份一份。
```



1）第一阶段：NameNode启动

（1）第一次启动NameNode格式化后，创建Fsimage和Edits文件。如果不是第一次启动，直接加载编辑日志和镜像文件到内存。

（2）客户端对元数据进行增删改的请求。

（3）NameNode记录操作日志，更新滚动日志。

（4）NameNode在内存中对元数据进行增删改。

2）第二阶段：Secondary NameNode工作

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

（2）Secondary NameNode请求执行CheckPoint。

（3）NameNode滚动正在写的Edits日志。

（4）将滚动前的编辑日志和镜像文件拷贝到Secondary NameNode。

（5）Secondary NameNode加载编辑日志和镜像文件到内存，并合并。

（6）生成新的镜像文件fsimage.chkpoint。

（7）拷贝fsimage.chkpoint到NameNode。

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

NN和2NN工作机制详解：

Fsimage：NameNode内存中元数据序列化后形成的文件。

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

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

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

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

5.2 Fsimage和Edits解析

1）oiv查看Fsimage文件

（1）查看oiv和oev命令

```
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

```
pwd
/opt/module/hadoop-3.1.3/data/dfs/name/current
hdfs oiv -p XML -i fsimage_0000000000000000025 -o /opt/module/hadoop-
3.1.3/fsimage.xml
cat /opt/module/hadoop-3.1.3/fsimage.xml
cd /opt/module/hadoop-3.1.3
sz fsimage.xml # 下载到本地
```

将显示的xml文件内容拷贝到IDEA中创建的xml文件中，并格式化。部分显示结果如下。

```
<inode>
    <id>16386</id>
    <type>DIRECTORY</type>
    <name>user</name>
    <mtime>1512722284477</mtime>
    <permission>xu1an:supergroup:rwxr-xr-x</permission>
    <nsquota>-1</nsquota>
    <dsquota>-1</dsquota>
</inode>
<inode>
    <id>16387</id>
    <type>DIRECTORY</type>
    <name>xu1an</name>
    <mtime>1512790549080</mtime>
    <permission>xu1an: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>xu1an: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。

2）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

```
hdfs oev -p XML -i edits_0000000000000000012-0000000000000000013 -o
/opt/module/hadoop-3.1.3/edits.xml
cat /opt/module/hadoop-3.1.3/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>3600s</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>60s</value>
<description> 1分钟检查一次操作次数</description>
</property >
```

5.4 NameNode故障处理（扩展）

NameNode故障后，可以采用如下两种方法恢复数据。

1）将SecondaryNameNode中数据拷贝到NameNode存储数据的目录；

（1）kill -9 NameNode进程

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

```
rm -rf /opt/module/hadoop-3.1.3/data/dfs/name/*
```

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

```
scp -r atguigu@hadoop104:/opt/module/hadoop-3.1.3/data/dfs/namesecondary/*
./name/
```

（4）重新启动NameNode

```
hdfs --daemon start namenode
```

2）使用-importCheckpoint选项启动NameNode守护进程，从而将SecondaryNameNode中数据拷贝到NameNode目录中。

（1）修改hdfs-site.xml中的

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

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

（2）kill -9 NameNode进程

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

```
rm -rf /opt/module/hadoop-3.1.3/data/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

```
 scp -r atguigu@hadoop104:/opt/module/hadoop-3.1.3/data/dfs/namesecondary ./

 rm -rf in_use.lock

 pwd
/opt/module/hadoop-3.1.3/data/dfs

ls
data  name  namesecondary
```

（5）导入检查点数据（等待一会ctrl+c结束掉）

```
bin/hdfs namenode -importCheckpoint
```

（6）启动NameNode

```
hdfs --daemon start namenode
```

5.5 集群安全模式

1、NameNode启动

NameNode启动时，首先将镜像文件（Fsimage）载入内存，并执行编辑日志（Edits）中的各项操作。一旦在内存中成功建立文件系统元数据的映像，则创建一个空的编辑日志。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。

1）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）
```

2）案例

模拟等待安全模式

3）查看当前模式

```
hdfs dfsadmin -safemode get
Safe mode is OFF
```

4）先进入安全模式

```
bin/hdfs dfsadmin -safemode enter
```

5）创建并执行下面的脚本

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

```
touch safemode.sh
vim safemode.sh

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

chmod 777 safemode.sh

./safemode.sh
```

6）再打开一个窗口，执行

7）观察

8）再观察上一个窗口

Safe mode is OFF

9）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文件中添加如下内容

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

（2）停止集群，删除三台节点的data和logs中所有数据。

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

（3）格式化集群并启动。

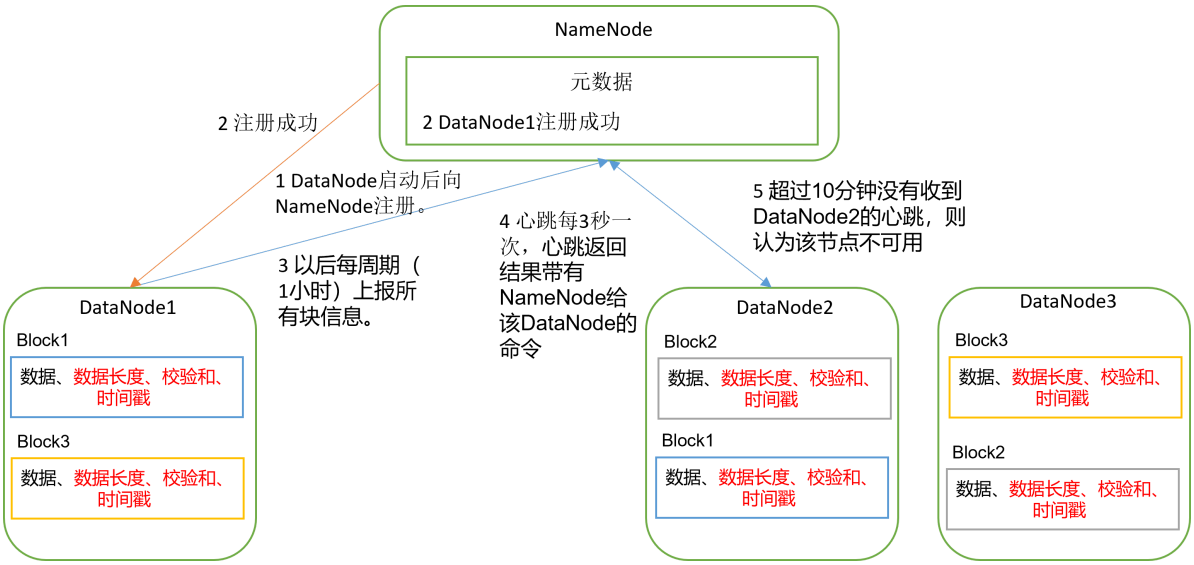
```
[@hadoop102 hadoop-3.1.3]$ bin/hdfs namenode -format
[@hadoop102 hadoop-3.1.3]$ sbin/start-dfs.sh
```

（4）查看结果

```
[@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
```

6 DataNode

6.1 DataNode工作机制



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

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

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

（4）集群运行中可以安全加入和退出一些机器。

6.2 数据完整性

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

如下是DataNode节点保证数据完整性的方法。

（1）当DataNode读取Block的时候，它会计算CheckSum。

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

（3）Client读取其他DataNode上的Block。

（4）常见的校验算法 crc（32），md5（128），sha1（160）

（5）DataNode在其文件创建后周期验证CheckSum。

6.3 掉线时限参数设置

需要注意的是hdfs-site.xml 配置文件中的heartbeat.recheck.interval的单位为毫秒，dfs.heartbeat.interval的单位为秒

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

6.4 服役新数据节点

1）需求

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

2）环境准备

（1）在hadoop104主机上再克隆一台hadoop105主机

（2）修改IP地址和主机名称

（3）删除原来HDFS文件系统留存的文件（/opt/module/hadoop-3.1.3/data和logs）如果没有删掉，nn就会认为104和105是同一台机器

（4）source一下配置文件

```
 source /etc/profile
```

3）服役新节点具体步骤

（1）直接启动DataNode，即可关联到集群

```
hdfs --daemon start datanode
yarn --daemon start nodemanager
```

（2）在hadoop105上上传文件

```
hadoop fs -put /opt/module/hadoop-3.1.3/LICENSE.txt
```

（3）如果数据不均衡，可以用命令实现集群的再平衡

```
 ./start-balancer.sh
starting balancer, logging to /opt/module/hadoop-3.1.3/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 添加白名单和黑名单

白名单和黑名单是hadoop管理集群主机的一种机制。

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。添加到黑名单的主机节点，不允许访问NameNode，会在数据迁移后退出。

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

配置白名单和黑名单的具体步骤如下：

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

```
pwd
/opt/module/hadoop-3.1.3/etc/hadoop
touch whitelist
touch blacklist
```

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

```
hadoop102
hadoop103
hadoop104
hadoop105
```

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

```
<!-- 白名单 -->
<property>
<name>dfs.hosts</name>
<value>/opt/module/hadoop-3.1.3/etc/hadoop/whitelist</value>
</property>
<!-- 黑名单 -->
<property>
<name>dfs.hosts.exclude</name>
<value>/opt/module/hadoop-3.1.3/etc/hadoop/blacklist</value>
</property>
```

3）分发配置文件whitelist，blacklist，hdfs-site.xml (注意：105节点也要发一份)

```
my_rsync hadoop/
rsync -av hadoop/ xu1an@hadoop105:/opt/module/hadoop-3.1.3/etc/hadoop/
```

4）重新启动集群(注意：105节点没有添加到workers，因此要单独起停)

```
[hadoop102 hadoop-3.1.3]$ stop-dfs.sh
[hadoop102 hadoop-3.1.3]$ start-dfs.sh
[hadoop105 hadoop-3.1.3]$ hdfs –daemon start datanode
```

5）在web浏览器上查看目前正常工作的DN节点

6.5.2 黑名单退役

1）编辑/opt/module/hadoop-3.1.3/etc/hadoop目录下的blacklist文件

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

添加如下主机名称（要退役的节点）

```
hadoop105
```

2）分发blacklist到所有节点

```
[@hadoop102 etc]$ my_rsync hadoop/
[@hadoop102 etc]$ rsync -av hadoop/ atguigu@hadoop105:/opt/module/hadoop-
3.1.3/etc/hadoop/
```

3）刷新NameNode、刷新ResourceManager

```
[@hadoop102 hadoop-3.1.3]$ hdfs dfsadmin -refreshNodes
Refresh nodes successful

[@hadoop102 hadoop-3.1.3]$ 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 hadoop-3.1.3]$ hdfs --daemon stop datanode
```

```
[@hadoop105 hadoop-3.1.3]$ yarn --daemon stop nodemanager
```

6）如果数据不均衡，可以用命令实现集群的再平衡

```
[@hadoop102 hadoop-3.1.3]$ sbin/start-balancer.sh
starting balancer, logging to /opt/module/hadoop-3.1.3/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，既然使用了黑名单blacklist成功退役了hadoop105节点，因此要将白名单whitelist里面的hadoop105去掉。

白名单退役编辑whitelist

```
hadoop102
hadoop103
hadoop104
```

6.6 DataNode多目录配置

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

2）具体配置如下

（1）在hdfs-site.xml文件中添加如下内容

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

（2）停止集群，删除三台节点的data和logs中所有数据。

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

（3）格式化集群并启动。

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

（4）查看结果

```
[@hadoop102 dfs]$ ll
```