# 目标成绩： A

# 大数据实验报告（实验2）

 班级 21计算机 4 班  学号 2021329600006 姓名 陈昊天

实验时间： 2024.04.09

一、实验名称：

HDFS基本操作及API编程

二、实验目的

要求学生

1、具备使用Hadoop命令来操作分布式文件系统的能力。

2、利用HDFS文件系统开放的API对HDFS系统进行文件的创建和读写。为此完成以下二个模块的工作：

**模块一 HDFS的Shell客户端操作**

1 完成HDFS环境搭建

2 通过HDFS的Shell客户端，掌握对HDFS所提供的文件接口的操作能力。

3 设计一个实验方案，以验证HDFS的组成架构和文件块大小等核心概念。

**模块二 HDFS的Java客户端API编程**

1、HDFS的JAVA客户端环境准备。

2、通过HDFS的JAVA客户端，掌握对HDFS的编程接口API对文件的操作能力

3、设计一个实验方案，通过HDFS的API操作，实现文件上传、文件下载、文件夹删除、文件名更改、文件和文件夹判断，以及用I/O流操作HDFS。

4、通过对HDFS写数据流的流程知识，详细分析自已实验方案中“文件写入”的过程。

三、实验设计

（在此，描述你根据实验目的和老师给的参考，分别对模块一和模块二设定的实验步骤）

#### 模块一 HDFS的Shell客户端操作

##### 1 完成HDFS环境搭建

以Debian 12为例

###### （1）安装JDK8

使用adoptium源安装temurin-8-jdk

###### （2）设置环境变量

包括JAVA\_HOME、HADOOP\_HOME和PATH

###### （3）下载Hadoop

下载并解压hadoop-3.3.6

###### （4）配置SSH免密登录

使用ssh-keygen生成密钥并复制到集群，包括namenode

###### （5）配置core-site.xml、hdfs-site.xml、workers和hadoop-env.sh

在core-site.xml设置fs.defaultFS

在hdfs-site.xml设置dfs.replication、dfs.namenode.name.dir、dfs.datanode.data.dir和dfs.namenode.secondary.http-address

在workers设置datanode的主机名

在hadoop-env.sh追加环境变量。

###### （6）在datanode上进行（1）（2）（3）（5）步。

###### （7）配置namenode和datanode的host文件

###### （8）格式化 NameNode

在 NameNode 上执行格式化命令，只需执行一次

###### （9）启动 Hadoop 集群

在NameNode执行启动命令

###### （10）验证集群状态

在NameNode查看集群健康状态

##### 2 通过HDFS的Shell客户端，掌握对HDFS所提供的文件接口的操作能力。

此部分包括hadoop常用命令的实操，如hadoop fs -ls / -mkdir / -rm / cat 等。

##### 3 设计一个实验方案，以验证HDFS的组成架构和文件块大小等核心概念。

###### 3.1 组成架构

###### （1） 使用 hdfs dfs -ls 验证NameNode

这一操作说明NameNode能够查询文件系统的元数据来获取目录列表。

###### （2） 使用 hdfs dfs -mkdir 创建目录

这证明了NameNode能够接收更改并更新文件系统的元数据。

###### （3） 使用 hdfs dfs -put 上传文件

这需要DataNode存储数据块，而NameNode则需要更新关于这些数据块的元数据。

###### 3.2 文件块大小

###### （1） 查看当前文件块大小

使用hdfs getconf -confKey dfs.blocksize命令

###### （2）上传文件到HDFS，记录上传时间

使用如下命令，dd创建一个1GB文件，上传到hdfs并记录时间

dd if=/dev/zero of=1GB.bin bs=1G count=1

START=$(date +%s%N)

hadoop fs -put 1GB.bin /

END=$(date +%s%N)

UPLOAD\_TIME=$(awk "BEGIN {print ($END - $START)/1000000}")

echo "Upload time: $UPLOAD\_TIME ms"

###### （3）修改块大小

在/usr/local/hadoop/etc/hadoop/hdfs-site.xml添加以下内容：

vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml

<property>

<name>dfs.blocksize</name>

<value>1048576</value> <!-- 1MB-->

</property>

重启Hadoop

使用hdfs getconf -confKey dfs.blocksize验证更改

###### （4）再次上传文件到HDFS，记录上传时间

#### 模块二 HDFS的Java客户端API编程

##### 1、HDFS的JAVA客户端环境准备。

此部分在模块一有所体现，包括安装temurin-8-jdk，设置JAVA\_HOME以及在core-site.xml中设置fs.defaultFS属性等。

###### 2、通过HDFS的JAVA客户端，掌握对HDFS的编程接口API对文件的操作能力

（1）在IDEA中新建Java Maven JDK1.8项目，设置pom.xml添加依赖：

（2）创建HdfsClient类测试连接

###### 3、设计一个实验方案，通过HDFS的API操作，实现文件上传、文件下载、文件夹删除、文件名更改、文件和文件夹判断，以及用I/O流操作HDFS。

3.1 文件上传

public void uploadFile(String source, String destination) throws Exception {  
 FileSystem fs = getFileSystem();  
 fs.copyFromLocalFile(new Path(source), new Path(destination));  
 fs.close();  
}

3.2 文件下载

public void downloadFile(String hdfsPath, String localPath) throws Exception {  
 FileSystem fs = getFileSystem();  
 fs.copyToLocalFile(new Path(hdfsPath), new Path(localPath));  
 fs.close();  
}

3.3 文件夹删除

public void deleteDirectory(String directory) throws Exception {  
 FileSystem fs = getFileSystem();  
 fs.delete(new Path(directory), true); // true 表示递归删除  
 fs.close();  
}

3.4 文件名更改

public void renameFile(String source, String destination) throws Exception {  
 FileSystem fs = getFileSystem();  
 fs.rename(new Path(source), new Path(destination));  
 fs.close();  
}

3.5 文件和文件夹判断

public void checkFileOrDirectory(String path) throws Exception {  
 FileSystem fs = getFileSystem();  
 FileStatus status = fs.getFileStatus(new Path(path));  
 if (status.isDirectory()) {  
 System.out.println(path + " 是一个文件夹");  
 } else {  
 System.out.println(path + " 是一个文件");  
 }  
 fs.close();  
}

3.6 I/O流操作HDFS

（1）文件上传

public void putFileToHDFS() throws Exception {  
 // 1. 获取对象  
 Configuration conf = new Configuration();  
 FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf, "root");  
 // 2. 输入流  
 FileInputStream fis = new FileInputStream(new File("testio.txt"));  
 // 3. 输出流  
 FSDataOutputStream fos = fs.create(new Path("/testio.txt"));  
 // 4. 输入输出流相互拷贝  
 IOUtils.copyBytes(fis, fos, conf);  
 // 5. 关闭流  
 IOUtils.closeStream(fos);  
 IOUtils.closeStream(fis);  
 fs.close();  
}

（2）文件下载

public void getFileFromHDFS() throws IOException, InterruptedException, URISyntaxException {  
 // 1. 创建配置对象  
 Configuration conf = new Configuration();  
 FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf, "root");  
 // 2. 输入流  
 FSDataInputStream fis = fs.open(new Path("/testio.txt"));  
 // 3. 输出流  
 FileOutputStream fos = new FileOutputStream(new File("testio1.txt"));  
 // 4. 流互拷贝  
 IOUtils.copyBytes(fis, fos, conf);  
 // 5. 关闭流对象  
 IOUtils.closeStream(fos);  
 IOUtils.closeStream(fis);  
 fs.close();  
}

（3）定位文件读取

@Test  
/\*\*  
 \* 下载第1块内容  
 \*/  
public void readFileSeek1() throws IOException, InterruptedException, URISyntaxException {  
 // 1. 获取对象  
 Configuration conf = new Configuration();  
 FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf, "root");  
 // 2.获取输入流  
 FSDataInputStream fis = fs.open(new Path("/hadoop-3.3.6.tar.gz"));  
 // 3. 获取输出流  
 FileOutputStream fos = new FileOutputStream(new File("hadoop-3.3.6.tar.gz.part1"));  
 // 4. 流的互拷贝（这里只拷贝指定大小的数据流128M）  
 byte[] buffer = new byte[1024];  
 for (int i = 0; i < 1024 \* 128; i++) {  
 fis.read(buffer);  
 fos.write(buffer);  
 }  
 // 5. 关闭资源  
 IOUtils.closeStream(fos);  
 IOUtils.closeStream(fis);  
 fs.close();  
}  
/\*\*  
 \* 下载第2块内容  
 \*/  
@Test  
public void readFileSeek2() throws IOException, InterruptedException, URISyntaxException {  
 // 1. 获取对象  
 Configuration conf = new Configuration();  
 FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf, "root");  
 // 2. 获取输入流  
 FSDataInputStream fis = fs.open(new Path("/hadoop-3.3.6.tar.gz"));  
 // 3. 指定输入流读取位置  
 fis.seek(1024\*1024\*128);  
 // 4. 获取输出流  
 FileOutputStream fos = new FileOutputStream(new File("hadoop-3.3.6.tar.gz.part2"));  
 // 5. 流的互拷贝  
 IOUtils.copyBytes(fis, fos, conf);  
 // 6. 关闭资源  
 IOUtils.closeStream(fos);  
 IOUtils.closeStream(fis);  
 fs.close();  
}

**合并文件**

cat hadoop-3.3.6.tar.gz.part2 >> hadoop-3.3.6.tar.gz.part1

合并完成后进行解压和哈希值比对，发现与源文件相同。

四、实验效果及结果分析：

（对模块一，要求能够验证HDFS的组成架构和文件块大小等核心概念，对模块二通，要求过对HDFS写数据流的流程知识，详细分析自已实验方案中“文件写入”过程及所获得的结果）

#### 模块一 HDFS的Shell客户端操作

##### 1 完成HDFS环境搭建

以Debian 12为例

###### （1）安装JDK8

使用adoptium源安装temurin-8-jdk

sudo apt-get install -y wget apt-transport-https gnupg

wget -O - https://packages.adoptium.net/artifactory/api/gpg/key/public | sudo apt-key add -

echo "deb https://packages.adoptium.net/artifactory/deb $(lsb\_release -sc) main" | sudo tee /etc/apt/sources.list.d/adoptium.list

sudo apt-get update

sudo apt-get install temurin-8-jdk

###### （2）设置环境变量

包括JAVA\_HOME、HADOOP\_HOME和PATH

echo 'export JAVA\_HOME=/usr/lib/jvm/temurin-8-jdk-amd64' >> ~/.bashrc

echo 'export HADOOP\_HOME=/usr/local/hadoop' >> ~/.bashrc

echo 'export PATH=$PATH:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin' >> ~/.bashrc

source ~/.bashrc

###### （3）下载Hadoop

下载并解压hadoop-3.3.6

wget https://dlcdn.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz

sudo tar zxvf hadoop-3.3.6.tar.gz -C /usr/local/

sudo mv /usr/local/hadoop-3.3.6 /usr/local/hadoop

rm hadoop-3.3.6.tar.gz

###### (4)配置SSH免密登录

使用ssh-keygen生成密钥并复制到集群，包括namenode

ssh-keygen -t rsa -P ''

ssh-copy-id -p 2200 root@namenode.vayki.com

ssh-copy-id -p 2200 root@datanode1.vayki.com

ssh-copy-id -p 2200 root@datanode2.vayki.com

这里使用了SSH非标准端口，还需编辑~/.ssh/config文件以配置hadoop正常运行。

vim ~/.ssh/config

填入以下内容：

Host namenode.vayki.com

HostName namenode.vayki.com

Port 2200

User root

Host datanode1.vayki.com

HostName datanode1.vayki.com

Port 2200

User root

Host datanode2.vayki.com

HostName datanode2.vayki.com

Port 2200

User root

然后更改权限600以保证安全

chmod 600 ~/.ssh/config

###### （5）配置core-site.xml、hdfs-site.xml、workers和hadoop-env.sh

在core-site.xml设置fs.defaultFS：

vim /usr/local/hadoop/etc/hadoop/core-site.xml

<property>

<name>fs.defaultFS</name>

<value>hdfs://namenode.vayki.com:59000</value>

</property>

在hdfs-site.xml设置dfs.replication、dfs.namenode.name.dir、dfs.datanode.data.dir和dfs.namenode.secondary.http-address：

vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>file:///usr/local/hadoop/hdfs/namenode</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>file:///usr/local/hadoop/hdfs/datanode</value>

</property>

<property>

<name>dfs.namenode.secondary.http-address</name>

<value>namenode.vayki.com:50090</value>

</property>

在workers设置datanode的主机名

vim /usr/local/hadoop/etc/hadoop/workers

datanode1.vayki.com

datanode2.vayki.com

在hadoop-env.sh追加环境变量。

echo 'export HDFS\_NAMENODE\_USER=root' >> /usr/local/hadoop/etc/hadoop/hadoop-env.sh

echo 'export HDFS\_DATANODE\_USER=root' >> /usr/local/hadoop/etc/hadoop/hadoop-env.sh

echo 'export HDFS\_SECONDARYNAMENODE\_USER=root' >> /usr/local/hadoop/etc/hadoop/hadoop-env.sh

echo 'export YARN\_RESOURCEMANAGER\_USER=root' >> /usr/local/hadoop/etc/hadoop/hadoop-env.sh

echo 'export YARN\_NODEMANAGER\_USER=root' >> /usr/local/hadoop/etc/hadoop/hadoop-env.sh

echo 'export JAVA\_HOME=/usr/lib/jvm/temurin-8-jdk-amd64' >> /usr/local/hadoop/etc/hadoop/hadoop-env.sh

source /usr/local/hadoop/etc/hadoop/hadoop-env.sh

###### （6）在DataNode上进行（1）（2）（3）（5）步。

###### （7）配置NameNode和DataNode的hosts文件

vim /etc/hosts

127.0.0.1 localhost

154.9.239.202 namenode.vayki.com namenode

185.212.62.40 datanode1.vayki.com datanode1

82.115.31.90 datanode2.vayki.com datanode2

###### （8）格式化 NameNode

在 NameNode 上执行格式化命令，只需执行一次

hdfs namenode -format

###### （9）启动 Hadoop 集群

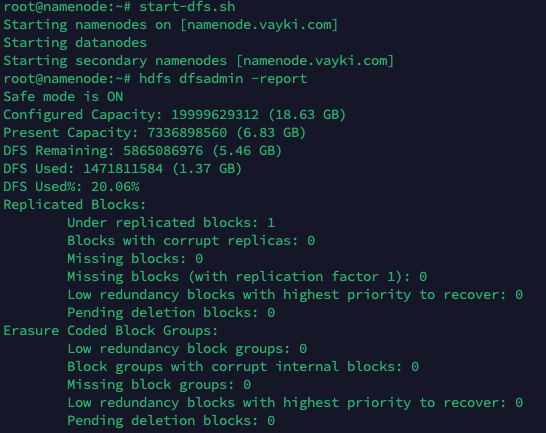
在NameNode执行启动命令

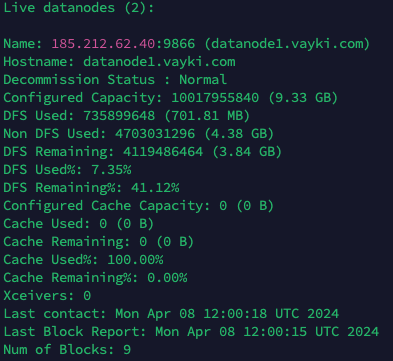
start-dfs.sh

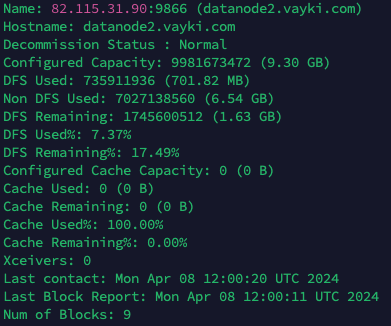
###### （10）验证集群状态

在NameNode查看集群健康状态

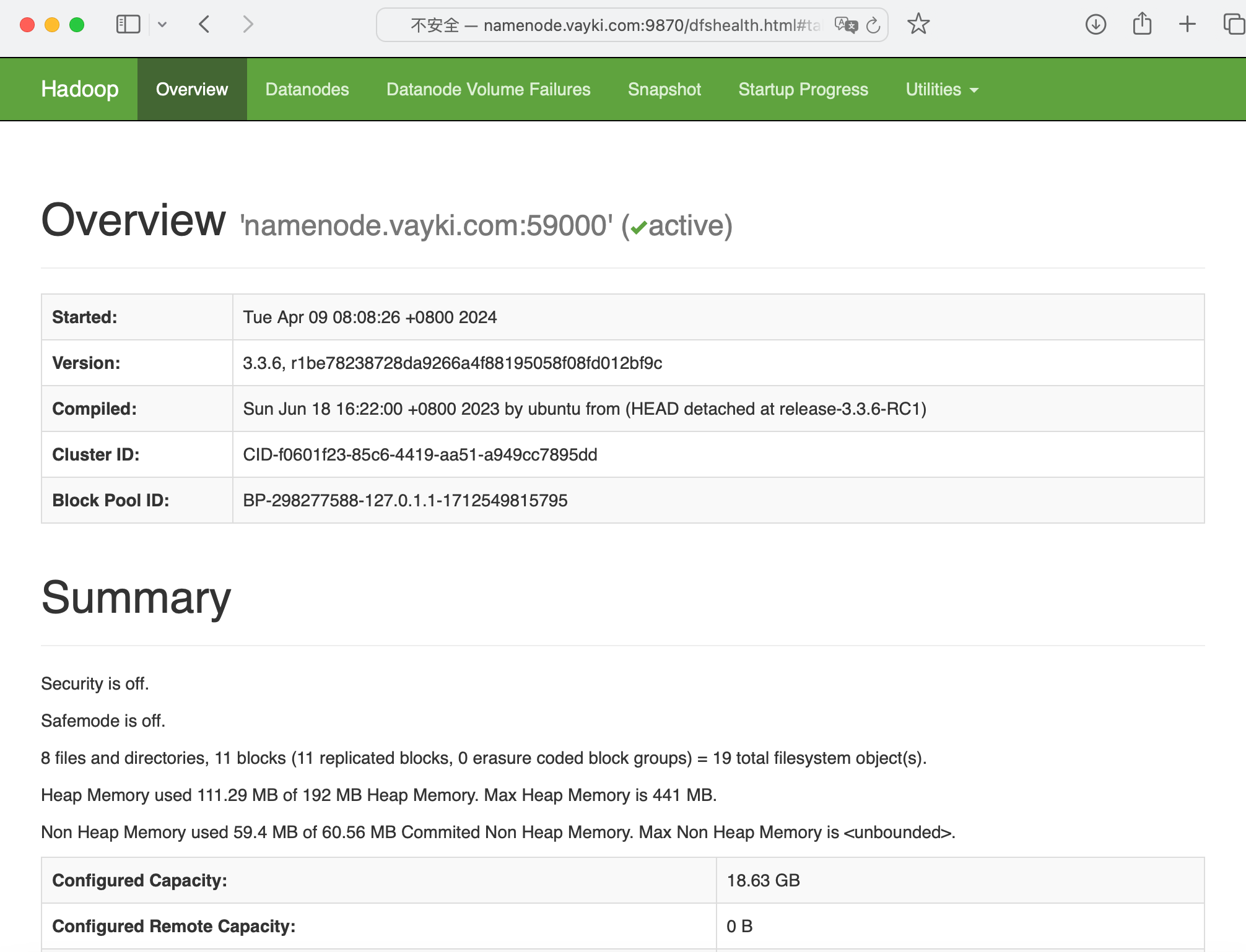
hdfs dfsadmin -report







浏览器打开 http://namenode.vayki.com:9870/ 即可访问 Web UI。



##### 2 通过HDFS的Shell客户端，掌握对HDFS所提供的文件接口的操作能力。

此部分包括hadoop常用命令的实操，如hadoop fs -ls / -mkdir / -rm / cat 等。

|  |  |
| --- | --- |
| 常用命令 | 运行结果 |
| 列出目录内容 | root@namenode:~# hadoop fs -ls / Found 2 items drwx------ - root supergroup 0 2024-04-06 08:25 /tmp drwxr-xr-x - root supergroup 0 2024-04-06 08:25 /user |
| 创建目录 | root@namenode:~# hadoop fs -mkdir /directory |
| 删除目录 | root@namenode:~# hadoop fs -rm -r /directory 24/04/06 08:46:47 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes. Deleted /directory |
| 上传本地文件到HDFS | root@namenode:~# touch localfile root@namenode:~# hadoop fs -put localfile / root@namenode:~# hadoop fs -ls / Found 3 items -rw-r--r-- 2 root supergroup 0 2024-04-06 08:48 /localfile drwx------ - root supergroup 0 2024-04-06 08:25 /tmp drwxr-xr-x - root supergroup 0 2024-04-06 08:25 /user |
| 从HDFS复制文件到本地 | root@namenode:~# rm localfile  root@namenode:~# hadoop fs -get / localfile root@namenode:~# ls 1.txt hdfs input localfile run-wordcount.sh start-hadoop.sh |
| 复制文件 | root@namenode:~# hadoop fs -cp /localfile /copyfile root@namenode:~# hadoop fs -ls / Found 4 items -rw-r--r-- 2 root supergroup 0 2024-04-06 08:51 /copyfile -rw-r--r-- 2 root supergroup 0 2024-04-06 08:48 /localfile drwx------ - root supergroup 0 2024-04-06 08:25 /tmp drwxr-xr-x - root supergroup 0 2024-04-06 08:25 /user |
| 移动文件 | root@namenode:~# hadoop fs -mv /localfile /movefile root@namenode:~# hadoop fs -ls / Found 4 items -rw-r--r-- 2 root supergroup 0 2024-04-06 08:51 /copyfile -rw-r--r-- 2 root supergroup 0 2024-04-06 08:48 /movefile drwx------ - root supergroup 0 2024-04-06 08:25 /tmp drwxr-xr-x - root supergroup 0 2024-04-06 08:25 /user |
| 查看文件 | root@namenode:~# hadoop fs -cat /hello  Hello, hadoop! |
| 更改文件或目录的权限 | root@namenode:~# hadoop fs -chmod 755 /hello |
| 统计目录中的文件数量和占用空间 | root@namenode:~# hadoop fs -count /  13 10 156456 / |
| 设置文件的副本数量 | root@namenode:~# hadoop fs -setrep 5 /hello Replication 5 set: /hello |

##### 3 设计一个实验方案，以验证HDFS的组成架构和文件块大小等核心概念。

###### 3.1 组成架构

###### （1） 使用 hdfs dfs -ls 验证NameNode

这一操作说明NameNode能够查询文件系统的元数据来获取目录列表。

root@namenode:~# hdfs dfs -ls /

Found 4 items

-rw-r--r-- 2 root supergroup 4 2024-04-08 04:34 /1234

-rw-r--r-- 2 root supergroup 5 2024-04-08 04:38 /2.txt

-rw-r--r-- 2 root supergroup 730107476 2024-04-08 10:00 /hadoop-3.3.6.tar.gz

-rw-r--r-- 3 root supergroup 6 2024-04-08 09:46 /testio.txt

###### （2） 使用 hdfs dfs -mkdir 创建目录

这证明了NameNode能够接收更改并更新文件系统的元数据。

root@namenode:~# hdfs dfs -mkdir /dir

root@namenode:~# hdfs dfs -ls /

Found 5 items

-rw-r--r-- 2 root supergroup 4 2024-04-08 04:34 /1234

-rw-r--r-- 2 root supergroup 5 2024-04-08 04:38 /2.txt

drwxr-xr-x - root supergroup 0 2024-04-08 12:07 /dir

-rw-r--r-- 2 root supergroup 730107476 2024-04-08 10:00 /hadoop-3.3.6.tar.gz

-rw-r--r-- 3 root supergroup 6 2024-04-08 09:46 /testio.txt

###### （3） 使用 hdfs dfs -put 上传文件

这需要DataNode存储数据块，而NameNode则需要更新关于这些数据块的元数据。

root@namenode:~# hdfs dfs -put 123 /789

root@namenode:~# hdfs dfs -ls /

Found 6 items

-rw-r--r-- 2 root supergroup 4 2024-04-08 04:34 /1234

-rw-r--r-- 2 root supergroup 5 2024-04-08 04:38 /2.txt

-rw-r--r-- 2 root supergroup 4 2024-04-08 12:08 /789

drwxr-xr-x - root supergroup 0 2024-04-08 12:07 /dir

-rw-r--r-- 2 root supergroup 730107476 2024-04-08 10:00 /hadoop-3.3.6.tar.gz

-rw-r--r-- 3 root supergroup 6 2024-04-08 09:46 /testio.txt

###### 3.2 文件块大小

###### （1） 查看当前文件块大小

使用hdfs getconf -confKey dfs.blocksize命令

root@namenode:~# hdfs getconf -confKey dfs.blocksize

134217728

###### （2）上传文件到HDFS，记录上传时间

使用如下命令，dd创建一个1GB文件，上传到hdfs并记录时间

dd if=/dev/zero of=1GB.bin bs=1G count=1

START=$(date +%s%N)

hadoop fs -put 1GB.bin /

END=$(date +%s%N)

UPLOAD\_TIME=$(awk "BEGIN {print ($END - $START)/1000000}")

echo "Upload time: $UPLOAD\_TIME ms"

root@namenode:~# START=$(date +%s%N)

root@namenode:~# hadoop fs -put 1GB.bin /

root@namenode:~# END=$(date +%s%N)

root@namenode:~# UPLOAD\_TIME=$(awk "BEGIN {print ($END - $START)/1000000}")

root@namenode:~# echo "Upload time: $UPLOAD\_TIME ms"

Upload time: 4902.21 ms

###### （3）修改块大小

在/usr/local/hadoop/etc/hadoop/hdfs-site.xml添加以下内容：

vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml

<property>

<name>dfs.blocksize</name>

<value>1048576</value> <!-- 1MB-->

</property>

重启Hadoop

/usr/local/hadoop/sbin/stop-dfs.sh

/usr/local/hadoop/sbin/start-dfs.sh

使用hdfs getconf -confKey dfs.blocksize验证更改

root@namenode:~# hdfs getconf -confKey dfs.blocksize

1048576

###### （4）再次上传文件到HDFS，记录上传时间

可见当文件块过小时，上传时间显著增加

root@namenode:~# START=$(date +%s%N)

root@namenode:~# hadoop fs -put 1GB.bin /

root@namenode:~# END=$(date +%s%N)

root@namenode:~# UPLOAD\_TIME=$(awk "BEGIN {print ($END - $START)/1000000}")

root@namenode:~# echo "Upload time: $UPLOAD\_TIME ms"

Upload time: 10947.8 ms

#### 模块二 HDFS的Java客户端API编程

##### 1、HDFS的JAVA客户端环境准备。

此部分在模块一有所体现，包括安装temurin-8-jdk，设置JAVA\_HOME以及在core-site.xml中设置fs.defaultFS属性等。

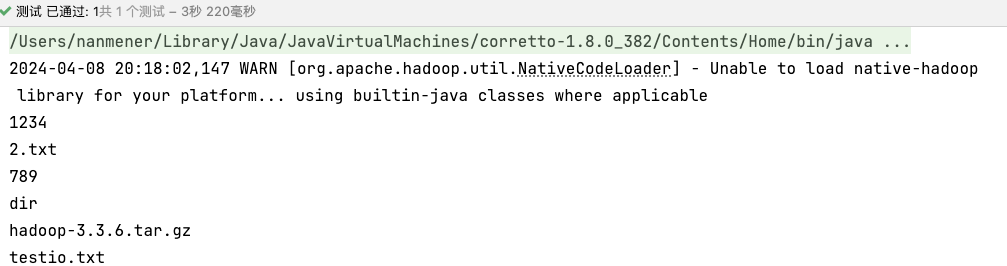
###### 2、通过HDFS的JAVA客户端，掌握对HDFS的编程接口API对文件的操作能力

（1）在IDEA中新建Java Maven JDK1.8项目，设置pom.xml添加依赖：

<dependencies>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.logging.log4j</groupId>  
 <artifactId>log4j-core</artifactId>  
 <version>2.8.2</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.hadoop</groupId>  
 <artifactId>hadoop-common</artifactId>  
 <version>2.7.2</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.hadoop</groupId>  
 <artifactId>hadoop-client</artifactId>  
 <version>2.7.2</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.hadoop</groupId>  
 <artifactId>hadoop-hdfs</artifactId>  
 <version>2.7.2</version>  
 </dependency>  
</dependencies>

（2）创建HdfsClient类测试连接

public class HdfsClient {  
 @Test  
 public void check() throws Exception {  
// uploadFile("hello.txt", "/hello.txt");  
 printFileList("/");  
 }  
 public FileSystem getFileSystem() throws Exception {  
 Configuration configuration = new Configuration();  
 String fileSystemURL = "hdfs://namenode.vayki.com:59000";  
 return FileSystem.get(new URI(fileSystemURL), configuration, "root");  
 }



###### 3、设计一个实验方案，通过HDFS的API操作，实现文件上传、文件下载、文件夹删除、文件名更改、文件和文件夹判断，以及用I/O流操作HDFS。

3.1 文件上传

public void uploadFile(String source, String destination) throws Exception {  
 FileSystem fs = getFileSystem();  
 fs.copyFromLocalFile(new Path(source), new Path(destination));  
 fs.close();  
}

3.2 文件下载

public void downloadFile(String hdfsPath, String localPath) throws Exception {  
 FileSystem fs = getFileSystem();  
 fs.copyToLocalFile(new Path(hdfsPath), new Path(localPath));  
 fs.close();  
}

3.3 文件夹删除

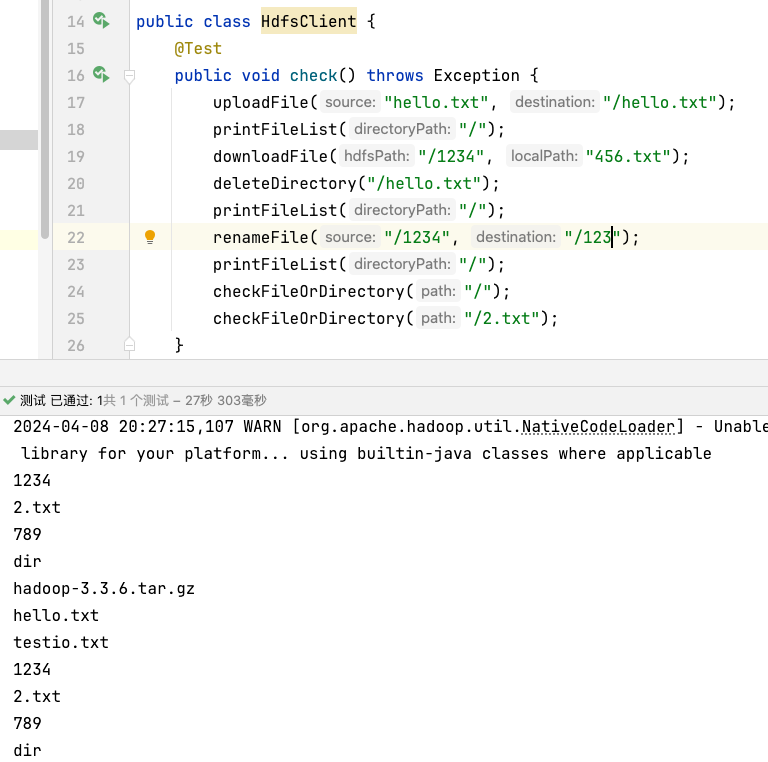
public void deleteDirectory(String directory) throws Exception {  
 FileSystem fs = getFileSystem();  
 fs.delete(new Path(directory), true); // true 表示递归删除  
 fs.close();  
}

3.4 文件名更改

public void renameFile(String source, String destination) throws Exception {  
 FileSystem fs = getFileSystem();  
 fs.rename(new Path(source), new Path(destination));  
 fs.close();  
}

3.5 文件和文件夹判断

public void checkFileOrDirectory(String path) throws Exception {  
 FileSystem fs = getFileSystem();  
 FileStatus status = fs.getFileStatus(new Path(path));  
 if (status.isDirectory()) {  
 System.out.println(path + " 是一个文件夹");  
 } else {  
 System.out.println(path + " 是一个文件");  
 }  
 fs.close();  
}



3.6 I/O流操作HDFS

（1）文件上传

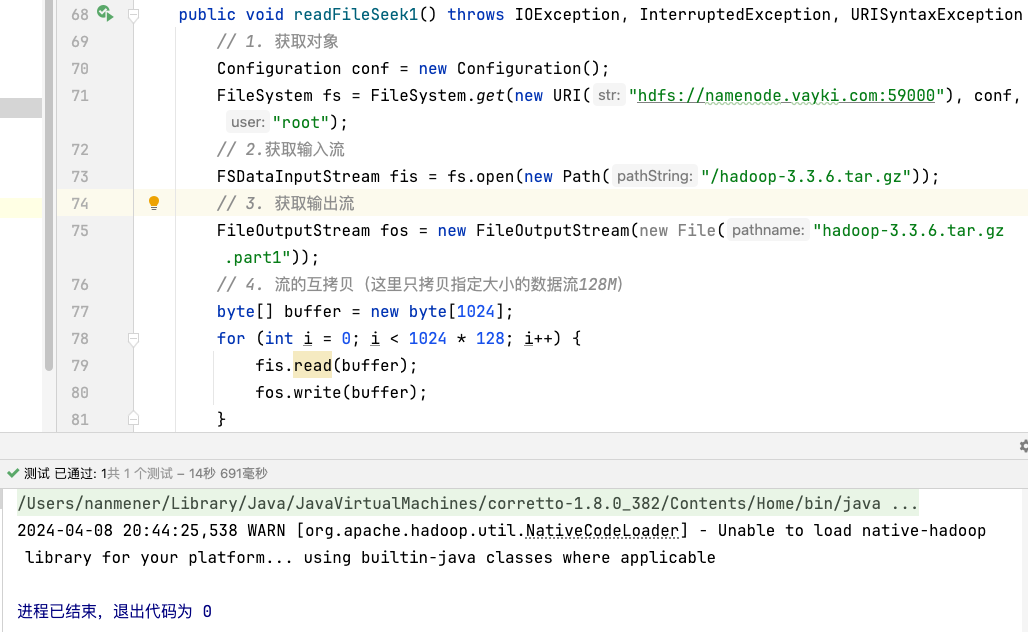
public void putFileToHDFS() throws Exception {  
 // 1. 获取对象  
 Configuration conf = new Configuration();  
 FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf, "root");  
 // 2. 输入流  
 FileInputStream fis = new FileInputStream(new File("testio.txt"));  
 // 3. 输出流  
 FSDataOutputStream fos = fs.create(new Path("/testio.txt"));  
 // 4. 输入输出流相互拷贝  
 IOUtils.copyBytes(fis, fos, conf);  
 // 5. 关闭流  
 IOUtils.closeStream(fos);  
 IOUtils.closeStream(fis);  
 fs.close();  
}

（2）文件下载

public void getFileFromHDFS() throws IOException, InterruptedException, URISyntaxException {  
 // 1. 创建配置对象  
 Configuration conf = new Configuration();  
 FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf, "root");  
 // 2. 输入流  
 FSDataInputStream fis = fs.open(new Path("/testio.txt"));  
 // 3. 输出流  
 FileOutputStream fos = new FileOutputStream(new File("testio1.txt"));  
 // 4. 流互拷贝  
 IOUtils.copyBytes(fis, fos, conf);  
 // 5. 关闭流对象  
 IOUtils.closeStream(fos);  
 IOUtils.closeStream(fis);  
 fs.close();  
}

（3）定位文件读取

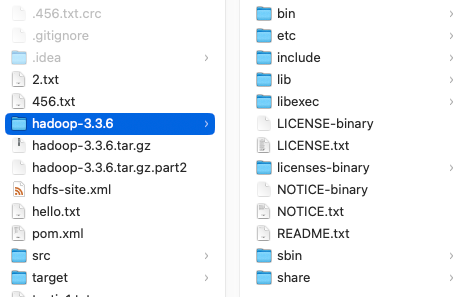
@Test  
/\*\*  
 \* 下载第1块内容  
 \*/  
public void readFileSeek1() throws IOException, InterruptedException, URISyntaxException {  
 // 1. 获取对象  
 Configuration conf = new Configuration();  
 FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf, "root");  
 // 2.获取输入流  
 FSDataInputStream fis = fs.open(new Path("/hadoop-3.3.6.tar.gz"));  
 // 3. 获取输出流  
 FileOutputStream fos = new FileOutputStream(new File("hadoop-3.3.6.tar.gz.part1"));  
 // 4. 流的互拷贝（这里只拷贝指定大小的数据流128M）  
 byte[] buffer = new byte[1024];  
 for (int i = 0; i < 1024 \* 128; i++) {  
 fis.read(buffer);  
 fos.write(buffer);  
 }  
 // 5. 关闭资源  
 IOUtils.closeStream(fos);  
 IOUtils.closeStream(fis);  
 fs.close();  
}  
/\*\*  
 \* 下载第2块内容  
 \*/  
@Test  
public void readFileSeek2() throws IOException, InterruptedException, URISyntaxException {  
 // 1. 获取对象  
 Configuration conf = new Configuration();  
 FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf, "root");  
 // 2. 获取输入流  
 FSDataInputStream fis = fs.open(new Path("/hadoop-3.3.6.tar.gz"));  
 // 3. 指定输入流读取位置  
 fis.seek(1024\*1024\*128);  
 // 4. 获取输出流  
 FileOutputStream fos = new FileOutputStream(new File("hadoop-3.3.6.tar.gz.part2"));  
 // 5. 流的互拷贝  
 IOUtils.copyBytes(fis, fos, conf);  
 // 6. 关闭资源  
 IOUtils.closeStream(fos);  
 IOUtils.closeStream(fis);  
 fs.close();  
}



**合并文件**

cat hadoop-3.3.6.tar.gz.part2 >> hadoop-3.3.6.tar.gz.part1

合并完成后进行解压和哈希值比对，发现与源文件相同。



###### 4、通过对HDFS写数据流的流程知识，详细分析自已实验方案中“文件写入”的过程。

文件写入的流程包括：

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2个datanode节点，分别为datanode1、datanode2。

5）客户端通过FSDataOutputStream模块请求datanode1上传数据，datanode1收到请求会继续调用datanode2，将这个通信管道建立完成。

6）datanode1、datanode2逐级应答客户端。

7）客户端开始往datanode1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，datanode1收到一个packet就会传给datanode2；

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）。

五、结论

本次实验完成了HDFS基本操作及API编程的学习目标。通过两个主要模块的深入探索，理解HDFS的核心概念和架构，掌握如何通过Hadoop命令和Java API进行有效的文件系统操作。

在模块一中，通过在Debian 12环境下搭建HDFS并配置必要的环境变量和文件系统参数，验证HDFS的组成架构和文件块大小等核心概念。实验过程包括SSH免密登录配置、core-site.xml和hdfs-site.xml的设置，Hadoop集群的启动和验证。通过这些步骤加深对HDFS架构的理解，通过Shell客户端操作掌握对HDFS文件接口的操作能力。这一模块的实践验证了HDFS的高可靠性和高扩展性特性。

在模块二中，通过HDFS的Java客户端API编程了解如何利用HDFS的编程接口API进行文件操作。通过设置Maven依赖和编写Java测试类，实现文件上传、下载、文件夹删除、文件名更改、文件和文件夹判断，以及使用I/O流操作HDFS的功能。详细分析文件写入的流程，从客户端请求上传文件到NameNode，再到数据的实际存储在DataNode上，整个过程揭示了HDFS的数据流转机制，体现HDFS的设计哲学和高效性。

本次实验遇到的问题与解决方式：

（1）开始试图使用docker搭建hadoop集群，发现 https://hub.docker.com/r/rancher/hadoop-base 在8年前停止更新，遂寻找新的镜像，找到了 https://hub.docker.com/r/apache/hadoop 镜像，发现其不提供 arm 版本镜像，无法在 M1 Macbook 上运行。

后找到 https://github.com/kiwenlau/hadoop-cluster-docker 项目，通过修改 Dockerfile 可以构建 arm 架构的镜像。发现该项目使用jdk7，于是修改 Dockerfile，升级ubuntu版本并使用jdk8。成功构建了 arm64 jdk8 的镜像，并发布到 https://hub.docker.com/r/chen2438/hadoop/tags 。

但在使用Java API时，发现 Macbook 宿主机无法通过 docker 网络访问容器（无法ping通），两者是隔离的。问题详见 https://www.cnblogs.com/luo-c/p/15830769.html ，在搜索解决方案后，决定放弃使用 M1 Macbook + Docker 环境，搭建真实的hadoop集群。

（2）在搭建真实的hadoop集群时遇到的问题：

1. Debian12官方源中已不支持JDK11，而hadoop 3.3.6需要JDK8（编译&运行）或JDK11（运行）。于是引入第三方源adoptium。

2. 使用的三台公网服务器使用SSH非标准端口2200，在hadoop运行前需要编辑~/.ssh/config文件。

3. 在使用rsync复制hadoop分发到datanode时操作不当，导致Incompatible clusterIDs报错，清楚datanode的数据目录后解决。

4. 由于主机名和hosts文件设置不当，导致 Datanode denied communication with namenode 报错，正确配置后解决。