

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，详细分析自己实验方案中“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，详细分析自己实验方案中“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
```

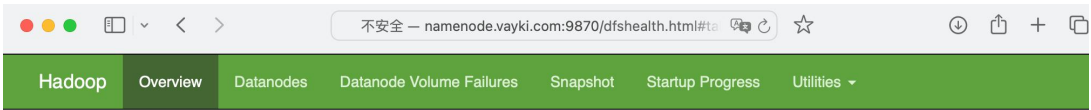
```
root@namenode:~# start-dfs.sh
Starting namenodes on [namenode.vayki.com]
Starting datanodes
Starting secondary namenodes [namenode.vayki.com]
root@namenode:~# hdfs dfsadmin -report
Safe mode is ON
Configured Capacity: 19999629312 (18.63 GB)
Present Capacity: 7336898560 (6.83 GB)
DFS Remaining: 5865086976 (5.46 GB)
DFS Used: 1471811584 (1.37 GB)
DFS Used%: 20.06%
Replicated Blocks:
    Under replicated blocks: 1
    Blocks with corrupt replicas: 0
    Missing blocks: 0
    Missing blocks (with replication factor 1): 0
    Low redundancy blocks with highest priority to recover: 0
    Pending deletion blocks: 0
Erasure Coded Block Groups:
    Low redundancy block groups: 0
    Block groups with corrupt internal blocks: 0
    Missing block groups: 0
    Low redundancy blocks with highest priority to recover: 0
    Pending deletion blocks: 0
```

Live datanodes (2):

Name: 185.212.62.40:9866 (datanode1.vayki.com)
Hostname: datanode1.vayki.com
Decommission Status : Normal
Configured Capacity: 10017955840 (9.33 GB)
DFS Used: 735899648 (701.81 MB)
Non DFS Used: 4703031296 (4.38 GB)
DFS Remaining: 4119486464 (3.84 GB)
DFS Used%: 7.35%
DFS Remaining%: 41.12%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 0
Last contact: Mon Apr 08 12:00:18 UTC 2024
Last Block Report: Mon Apr 08 12:00:15 UTC 2024
Num of Blocks: 9

Name: 82.115.31.90:9866 (datanode2.vayki.com)
Hostname: datanode2.vayki.com
Decommission Status : Normal
Configured Capacity: 9981673472 (9.30 GB)
DFS Used: 735911936 (701.82 MB)
Non DFS Used: 7027138560 (6.54 GB)
DFS Remaining: 1745600512 (1.63 GB)
DFS Used%: 7.37%
DFS Remaining%: 17.49%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 0
Last contact: Mon Apr 08 12:00:20 UTC 2024
Last Block Report: Mon Apr 08 12:00:11 UTC 2024
Num of Blocks: 9

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



Overview 'namenode.vayki.com:59000' (✓active)

Started:	Tue Apr 09 08:08:26 +0800 2024
Version:	3.3.6, r1be78238728da9266a4f88195058f08fd012bf9c
Compiled:	Sun Jun 18 16:22:00 +0800 2023 by ubuntu from (HEAD detached at release-3.3.6-RC1)
Cluster ID:	CID-f0601f23-85c6-4419-aa51-a949cc7895dd
Block Pool ID:	BP-298277588-127.0.1.1-1712549815795

Summary

Security is off.
Safemode is off.
8 files and directories, 11 blocks (11 replicated blocks, 0 erasure coded block groups) = 19 total filesystem object(s).
Heap Memory used 111.29 MB of 192 MB Heap Memory. Max Heap Memory is 441 MB.
Non Heap Memory used 59.4 MB of 60.56 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	18.63 GB
Configured Remote Capacity:	0 B

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:~# <code>hadoop fs -ls /</code> 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:~# <code>hadoop fs -mkdir /directory</code>
删除目录	root@namenode:~# <code>hadoop fs -rm -r /directory</code> 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	<pre> 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 </pre>
从HDFS复制文件到本地	<pre> root@namenode:~# rm localfile root@namenode:~# hadoop fs -get / localfile root@namenode:~# ls 1.txt hdfs input localfile run-wordcount.sh start-hadoop.sh </pre>
复制文件	<pre> 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 </pre>
移动文件	<pre> 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 </pre>
查看文件	<pre> root@namenode:~# hadoop fs -cat /hello Hello, hadoop! </pre>
更改文件或目录的权限	<pre> root@namenode:~# hadoop fs -chmod 755 /hello </pre>
统计目录中的文件数量和占用空间	<pre> root@namenode:~# hadoop fs -count / 13 10 156456 / </pre>
设置文件的副本数	<pre> root@namenode:~# hadoop fs -setrep 5 /hello Replication 5 set: /hello </pre>

量	
---	--

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");
    }
}

```

✓ 测试 已通过: 1共 1 个测试 - 3秒 220毫秒

```

/Users/nanmener/Library/Java/JavaVirtualMachines/corretto-1.8.0_382/Contents/Home/bin/java ...
2024-04-08 20:18:02,147 WARN [org.apache.hadoop.util.NativeCodeLoader] - Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
1234
2.txt
789
dir
hadoop-3.3.6.tar.gz
testio.txt

```

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();
}
```

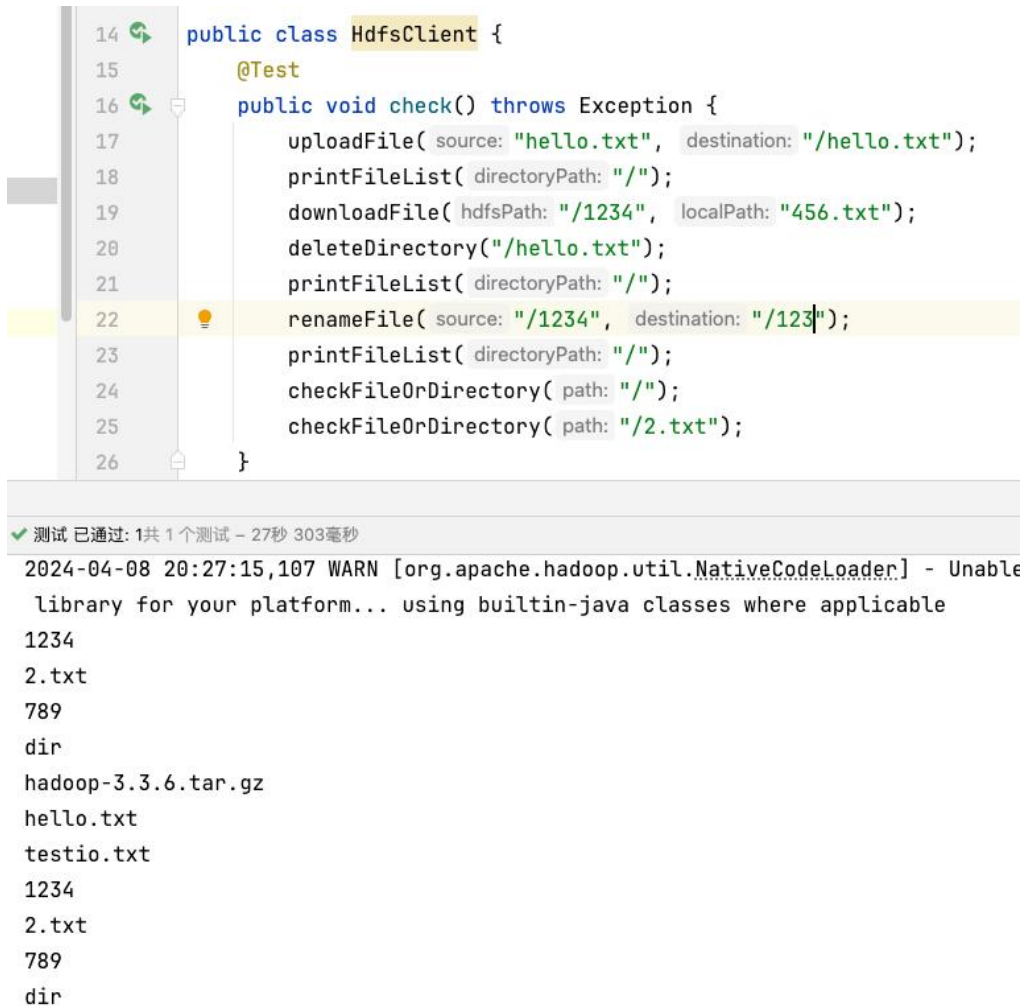
3.5 文件和文件夹判断

```
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();
}

```



```

14 public class HdfsClient {
15     @Test
16     public void check() throws Exception {
17         uploadFile( source: "hello.txt", destination: "/hello.txt");
18         printFileList( directoryPath: "/");
19         downloadFile( hdfsPath: "/1234", localPath: "456.txt");
20         deleteDirectory("/hello.txt");
21         printFileList( directoryPath: "/");
22         renameFile( source: "/1234", destination: "/123");
23         printFileList( directoryPath: "/");
24         checkFileOrDirectory( path: "/");
25         checkFileOrDirectory( path: "/2.txt");
26     }
}

```

✓ 测试 已通过: 1共 1 个测试 - 27秒 303毫秒

2024-04-08 20:27:15,107 WARN [org.apache.hadoop.util.NativeCodeLoader] - Unable to load native code library for your platform... using builtin-java classes where applicable

```

1234
2.txt
789
dir
hadoop-3.3.6.tar.gz
hello.txt
testio.txt
1234
2.txt
789
dir

```

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();
    }
}

```

```
68 public void readFileSeek1() throws IOException, InterruptedException, URISyntaxException
69 // 1. 获取对象
70 Configuration conf = new Configuration();
71 FileSystem fs = FileSystem.get(new URI( str: "hdfs://namenode.vayki.com:59000"), conf,
72     user: "root");
73 // 2. 获取输入流
74 FSDDataInputStream fis = fs.open(new Path( pathString: "/hadoop-3.3.6.tar.gz"));
75 // 3. 获取输出流
76 FileOutputStream fos = new FileOutputStream(new File( pathname: "hadoop-3.3.6.tar.gz
77     .part1"));
78 // 4. 流的互拷贝 (这里只拷贝指定大小的数据流128M)
79 byte[] buffer = new byte[1024];
80 for (int i = 0; i < 1024 * 128; i++) {
81     fis.read(buffer);
82     fos.write(buffer);
83 }
```

测试 已通过: 1共 1 个测试 - 14秒 691毫秒

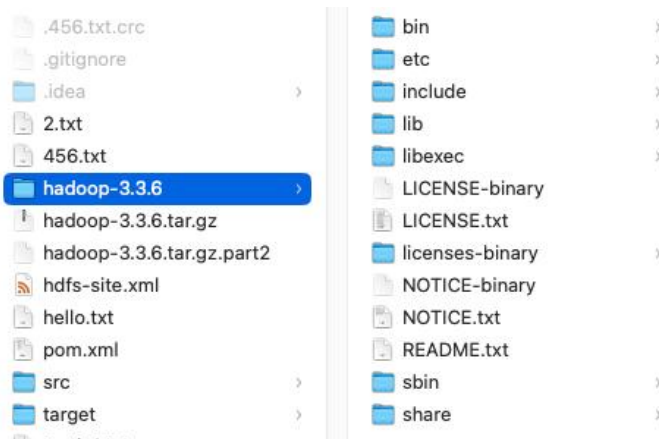
/Users/nanmener/Library/Java/JavaVirtualMachines/corretto-1.8.0_382/Contents/Home/bin/java ...
2024-04-08 20:44:25,538 WARN [org.apache.hadoop.util.NativeCodeLoader] - Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable

进程已结束, 退出代码为 0

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, 详细分析自己实验方案中“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 FSDDataOutputStream 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/cheng2438/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。