HDFS shell 命令和 Java API

刘磊

2020 年 8 月

1 HDFS shell 命令

调用文件系统(FS)Shell 命令使用

```
hadoop fs -cmd <args:uri>
```

的形式。 所有的 FS shell 命令使用 URI 路径作为参数。URI 格式是

```
scheme://authority/path
```

对 HDFS 文件系统，scheme 是 *hdfs*，对本地文件系统，scheme 是 *file*。其中 scheme 和 authority 参数都是可选的，如果未加指定，就会使用配置中指定的默认 scheme。一个 HDFS 文件或目录比如*/parent/child* 可以表示成

```
hdfs://namenode:namenodeport/parent/child
```

如果配置文件中设置了默认值 *namenode:namenodeport*，比如我们在安装 Hadoop 时，在 core-site.xml 文件中设置的 localhost:9000，

```
<property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
</property>
</configuration>
```

RUI 可以简写为

```
/parent/child
```

大多数 FS Shell 命令的行为和对应的 Linux Shell 命令类似。例如，查看本地目录 /home/lei/big_data_tools 中内容

```
hadoop fs -ls file:///home/lei/big_data_tools
```

```
lei@ubuntu:~$ hadoop fs -ls file:///home/lei/big_data_tools/
Found 13 items
-rwxr-x---   1 lei lei    1165347 2020-09-13 06:50 file:///home/lei/big_data_too
ls/IKAnalyzer2012_u6.jar
-rwxr-x---   1 lei lei  232027212 2020-09-13 06:50 file:///home/lei/big_data_too
ls/apache-hive-2.3.5-bin.tar.gz
-rwxr-x---   1 lei lei   20182998 2020-09-13 06:50 file:///home/lei/big_data_too
ls/apache-storm-0.9.4.tar.gz
-rwxr-x---   1 lei lei  214091931 2020-09-13 06:50 file:///home/lei/big_data_too
ls/eclipse-java-2020-06-R-linux-gtk-x86_64.tar.gz
-rwxr-x---   1 lei lei  306392917 2020-09-13 06:50 file:///home/lei/big_data_too
ls/hadoop-3.0.0.tar.gz
```

注：URI，统一资源标志符(Uniform Resource Identifier， URI)，表示的是 web 上每一种可用的资源，如 HTML 文档、图像、视频片段、程序等都由一个 URI 进行标识的。

为了演示后面的命令，在本地/data 目录下新建两个文件，/data/testfile1 和 /data/testfile2，内容分别为 Hello big data!和 Hello hadoop!

```
lei@ubuntu:~$ cat /data/testfile1 /data/testfile2
Hello big data!
Hello hadoop!
```

mkdir

在 HDFS 文件系统创建新目录，只能创建一级目录。创建多级目录上一级目录必须存在，或使用-p 参数。

使用方法：`hadoop fs -mkdir <paths>`

例如，在 HDFS 根目录下的 input 目录下创建 files 目录。如果/input 目录不存在，需要加-p 参数。

```
hadoop fs -mkdir -p /input/files
```

put

上传本地文件系统中单个或多个文件到 HDFS 文件系统。

使用方法：hadoop fs -put <local:Files> <hdfs:Directory>

例如，上传/data/testfile1 和/data/testfile2 到新创建的目录/input/files 中，可以两个文件分别上传，

```
hadoop fs -put /data/testfile1 /input/files
hadoop fs -put /data/testfile2 /input/files
```

也可以使用一条命令，同时上传两个文件

```
hadoop fs -put /data/testfile1 /data/testfile2 /input/files
```

ls

查看文件或目录，如果是文件，返回文件的信息，如果是目录，返回它直接子目录的文件列表。加上-R 选项以方式递归查看目录下所有文件

使用方法：`hadoop fs -ls <hdfs:Directory>`

例如，查看/input/files 目录，确认我们的两个文件是否上传成功。

```
hadoop fs -ls /input/files
```

```
lei@ubuntu:~$ hadoop fs -ls /input/files
Found 2 items
-rw-r--r--   1 lei supergroup         16 2020-08-27 11:24 /input/files/testfile1
-rw-r--r--   1 lei supergroup         14 2020-08-27 11:24 /input/files/testfile2
```

查看文件/input/files/testfile1 的信息

```
hadoop fs -ls /input/files/testfile1
```

```
lei@ubuntu:~$ hadoop fs -ls /input/files/testfile1
-rw-r--r--   1 lei supergroup         16 2020-08-27 11:24 /input/files/testfile1
```

查看根目录下所有文件

```
hadoop fs -ls -R /
```

cat

查看指定文件的内容

使用方法：hadoop fs -cat URI [URI …]

例如，查看/input/files/testfile1 的内容

```
hadoop fs -cat /input/files/testfile1
```

```
lei@ubuntu:~$ hadoop fs -cat /input/files/testfile1
Hello big data!
```

查看一个目录下所有文件的内容可以使用通配符*

例如，查看目录/input/files/的所有文件内容，命令为

```
hadoop fs -cat /input/files/*
```

```
lei@ubuntu:~$ hadoop fs -cat /input/files/*
Hello big data!
Hello hadoop!
```

rm

删除指定的文件，加上-r 选项可以删除指定目录。

使用方法：hadoop fs -rm <hdfs:File>

例如，删除文件/myhadoop/testfile1（如果不存在，先创建）

```
hadoop fs -rm /myhadoop/testfile1
```

```
lei@ubuntu:~$ hadoop fs -rm /myhadoop/testfile1
Deleted /myhadoop/testfile1
```

例如，删除目录/myhadoop

```
hadoop fs -rm -r /myhadoop/
```

```
lei@ubuntu:~$ hadoop fs -rm -r /myhadoop/
Deleted /myhadoop
```

cp

将文件从源路径复制到目标路径。这个命令允许有多个源路径，此时目标路径必须是一个目录。

使用方法：`hadoop fs -cp <hdfs:File> <hdfs:Directory>`

例如，复制/input/files/目录中的两个文件到 HDFS 根目录下

```
hadoop fs -cp /input/files/testfile1 /input/files/testfile2 /
```

```
lei@ubuntu:~$ hadoop fs -cp /input/files/testfile1 /input/files/testfile2 /
lei@ubuntu:~$ hadoop fs -ls /
Found 6 items
drwxr-xr-x   - lei supergroup          0 2020-08-27 11:23 /input
-rw-r--r--   3 lei supergroup         28 2020-08-27 00:01 /testcreate
-rw-r--r--   1 lei supergroup         16 2020-08-27 12:02 /testfile1
-rw-r--r--   1 lei supergroup         14 2020-08-27 12:02 /testfile2
drwx------   - lei supergroup          0 2020-08-25 23:58 /tmp
drwxr-xr-x   - lei supergroup          0 2020-08-25 23:58 /user
```

get

从 HDFS 文件系统下载文件到本地目录，本地目录必须存在。

使用方法：`hadoop fs -get <hdfs:File> <local:Directory>`

例如，下载/input/files/testfile1 文件到当前用户家目录下

```
hadoop fs -get /input/files/testfile1 ~/
```

```
lei@ubuntu:~$ ls
big_data_tools  Documents  eclipse-workspace  Music     Public     testdir
Desktop         Downloads  examples.desktop   Pictures  Templates  Videos
lei@ubuntu:~$ hadoop fs -get /input/files/testfile1 ~
lei@ubuntu:~$ ls
big_data_tools  Downloads          Music     Templates  Videos
Desktop         eclipse-workspace  Pictures  testdir
Documents       examples.desktop   Public    testfile1
```

mv

将文件从源路径移动到目标路径。这个命令允许有多个源路径，此时目标路径必须是一个目录。移动的同时对文件进行改名。

使用方法：`hadoop fs -mv <hdfs:source> <hdfs:target>`

例如：将文件/testfile1 移动到目录/input

```
hadoop fs -mv /testfile1 /input/
```

```
lei@ubuntu:~$ hadoop fs -mv /testfile1 /input/
lei@ubuntu:~$ hadoop fs -ls /input
Found 2 items
drwxr-xr-x   - lei supergroup          0 2020-08-27 12:56 /input/files
-rw-r--r--   1 lei supergroup         16 2020-08-27 12:56 /input/testfile1
```

例如：将文件/testfile2 移动到目录/input/files，并改名为 testfile3。

```
hadoop fs -mv /testfile2 /input/files/testfile3
```

```
lei@ubuntu:~$ hadoop fs -mv /testfile2 /input/files/testfile3
lei@ubuntu:~$ hadoop fs -ls /input/files/
Found 3 items
-rw-r--r--   1 lei supergroup         16 2020-08-27 11:24 /input/files/testfile1
-rw-r--r--   1 lei supergroup         14 2020-08-27 11:24 /input/files/testfile2
-rw-r--r--   1 lei supergroup         14 2020-08-27 12:02 /input/files/testfile3
```

test

检查一个文件或目录是否存在，存在返回 0，否则返回 1。

使用方法：`hadoop fs -test -[ed] <hdfs:target>`

选项：

`-e` 检查文件是否存在

`-d` 检查目录是否存在

例如：检查`/input/files/testfile1` 是否存在

```
hadoop fs -test -e /input/files/testfile1
```

```
lei@ubuntu:~$ hadoop fs -test -e /input/files/testfile1
lei@ubuntu:~$ echo $?
0
```

检查目录/input 是否存在

```
hadoop fs -test -d /input/
```

```
lei@ubuntu:~$ hadoop fs -test -d /input/
lei@ubuntu:~$ echo $?
0
```

命令的返回值没有直接显示在终端中，需要使用 `echo` 命令查看。返回 `0`，说明存在，返回 `1`，说明不存在。

```
echo $?
```

使用 Shell 命令执行 Hadoop 自带的 WordCount

1. 将 HDFS 上/inputs/files/目录中三个的文件作为输入，

   ```
   lei@ubuntu:~$ hadoop fs -ls /input/files/
   Found 3 items
   -rw-r--r--   1 lei supergroup         16 2020-08-27 11:24 /input/files/testfile1
   -rw-r--r--   1 lei supergroup         14 2020-08-27 11:24 /input/files/testfile2
   -rw-r--r--   1 lei supergroup         14 2020-08-27 12:02 /input/files/testfile3
   ```

   三个文件的内容为

   ```
   lei@ubuntu:~$ hadoop fs -cat /input/files/*
   Hello big data!
   Hello hadoop!
   Hello hadoop!
   ```

2. 词频统计结果将输出到目录/output/wordcount，注意/output/wordcount 不能提前存在。

3. 执行 hadoop jar 命令，在 hadoop 的 /apps/hadoop/share/hadoop/mapreduce 路径下存在 hadoop-mapreduce-examples-3.0.0.jar 包，执行其中的 worldcount 类，输入为 HDFS 的 /input/files 目录下的文件，结果输出到 HDFS 的/output/wordcount 目录。

```
hadoop jar /apps/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.0.0.jar wordcount /input/files /output/wordcount
```

```
2020-08-27 13:17:33,201 INFO mapreduce.Job:  map 67% reduce 0%
2020-08-27 13:17:39,225 INFO mapreduce.Job:  map 100% reduce 0%
2020-08-27 13:17:40,229 INFO mapreduce.Job:  map 100% reduce 100%
2020-08-27 13:17:40,234 INFO mapreduce.Job: Job job_1598495233491_0001 completed
 successfully
```

查看 HDFS 中的/output/wordcount 目录

```
hadoop fs -ls /output/wordcount/
```

```
lei@ubuntu:~$ hadoop fs -ls /output/wordcount/
Found 2 items
-rw-r--r--   1 lei supergroup          0 2020-08-27 13:17 /output/wordcount/_SUC
CESS
-rw-r--r--   1 lei supergroup         32 2020-08-27 13:17 /output/wordcount/part
-r-00000
```

查看结果，结果保存在 part-r-00000 中。

```
hadoop fs -cat /output/wordcount/part-r-00000
```

```
lei@ubuntu:~$ hadoop fs -cat /output/wordcount/part-r-00000
Hello   3
big     1
data!   1
hadoop! 2
```

查看 hadoop 自带的有哪些例子

```
hadoop jar /apps/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.0.0.jar
```

```
lei@lei-VirtualBox:~$ hadoop jar /apps/hadoop/share/hadoop/mapreduce/hadoop-mapr
educe-examples-3.0.0.jar
An example program must be given as the first argument.
Valid program names are:
  aggregatewordcount: An Aggregate based map/reduce program that counts the word
s in the input files.
  aggregatewordhist: An Aggregate based map/reduce program that computes the his
togram of the words in the input files.
  bbp: A map/reduce program that uses Bailey-Borwein-Plouffe to compute exact di
gits of Pi.
  dbcount: An example job that count the pageview counts from a database.
  distbbp: A map/reduce program that uses a BBP-type formula to compute exact bi
ts of Pi.
  grep: A map/reduce program that counts the matches of a regex in the input.
  join: A job that effects a join over sorted, equally partitioned datasets
  multifilewc: A job that counts words from several files.
  pentomino: A map/reduce tile laying program to find solutions to pentomino pro
blems.
```

安全模式

在分布式文件系统启动的时候，开始的时候会有安全模式，当分布式文件系统处于安

全模式的情况下，文件系统中的内容不允许修改也不允许删除，直到安全模式结束。安全模式主要是为了系统启动的时候检查各个 DataNode 上数据块的有效性，同时根据策略必要的复制或者删除部分数据块。运行期通过命令也可以进入安全模式。在实践过程中，系统启动的时候去修改和删除文件也会有安全模式不允许修改的出错提示，只需要等待一会儿即可。

进入 Hadoop 安全模式

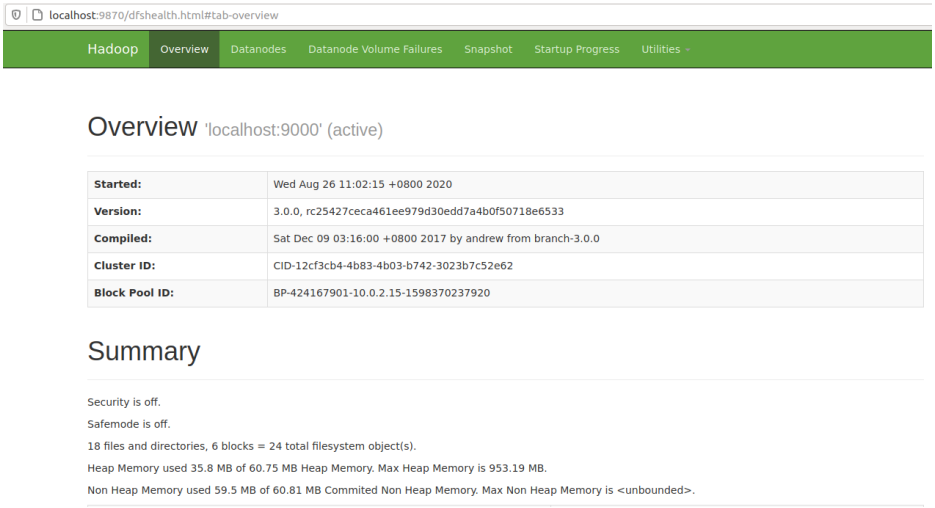
```
hdfs dfsadmin -safemode enter
```

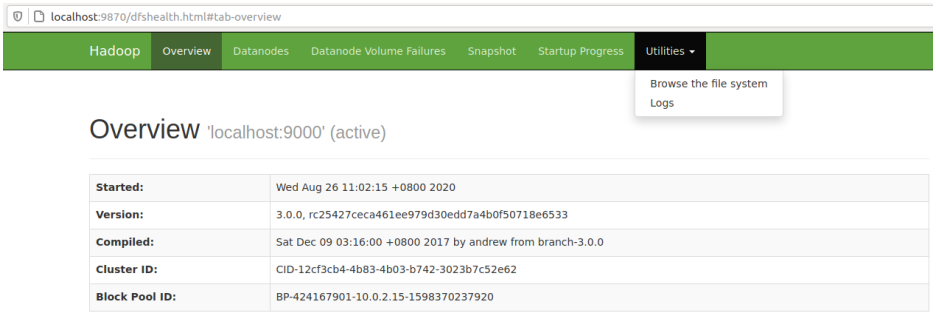
退出 Hadoop 安全模式

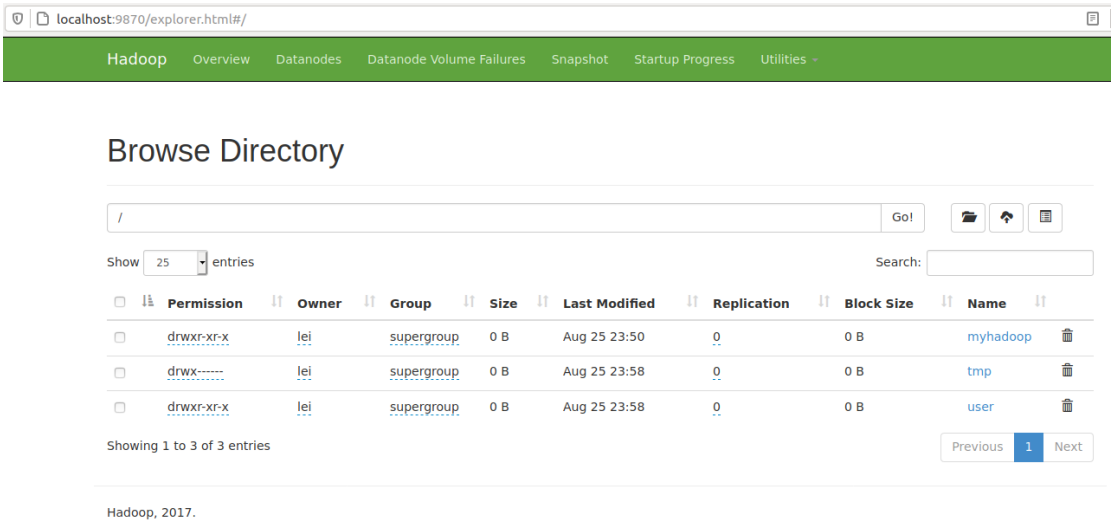
```
hdfs dfsadmin -safemode leave
```

HDFS Web 界面

在浏览器中访问 http://localhost:9870，可以查看 HDFS 相关信息，浏览 HDFS 上的文件系统。







2 HDFS Java API

HDFS 设计的主要目的是对海量数据进行存储，也就是说在其上能够存储很大量的文件。HDFS 将这些文件分割之后，存储在不同的 DataNode 上，HDFS 提供了通过 Java API 对 HDFS 里面的文件进行操作的功能，数据块在 DataNode 上的存放位置，对于开发者来说是透明的。

使用 Java API 可以完成对 HDFS 的各种操作，如新建文件、删除文件、读取文件内容等。下面将介绍 HDFS 常用的 Java API 及其编程实例。

下表显示了对 HDFS 中的文件操作主要涉及的类

名称	作用
org.apache.hadoop.con.Configuration	该类的对象封装了客户端或者服务器的配置。
org.apache.hadoop.fs.FileSystem	该类的对象是一个文件系统对象，可以用该对象的一些方法来对文件进行操作。
org.apache.hadoop.fs.FileStatus	该类用于向客户端展示系统中文件和目录的元数据，具体包括文件大小、块大小、副本信

名称	作用
	息、所有者、修改时间等。
org.apache.hadoop.fs.FSDataInputStream	该类是 HDFS 中的输入流，用于读取 Hadoop 文件。
org.apache.hadoop.fs.FSDataOutputStream	该类是 HDFS 中的输出流，用于写 Hadoop 文件;
org.apache.hadoop.fs.Path	该类用于表示 Hadoop 文件系统中的文件或者目录的路径。

注:

- FileSystem 类：该类的对象是一个文件系统对象，可以用该对象的一些方法来对文件进行操作。FileSystem fs = FileSystem.get(conf)；通过 FileSystem 的静态方法 get 获得该对象。
- FSDataInputStream 和 FSDataOutputStream：这两个类是 HDFS 中的输入输出流。分别通过 FileSystem 的 open 方法和 create 方法获得。

实验过程

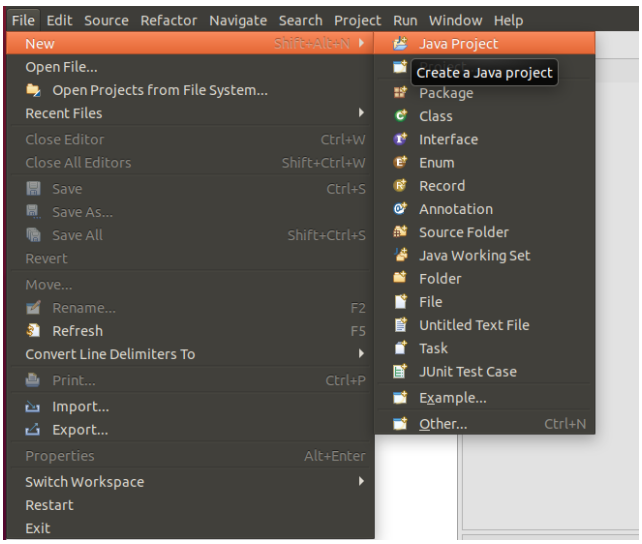
1. 准备需要的 jar 文件

新建目录~/big_data_tools /hadoop3lib

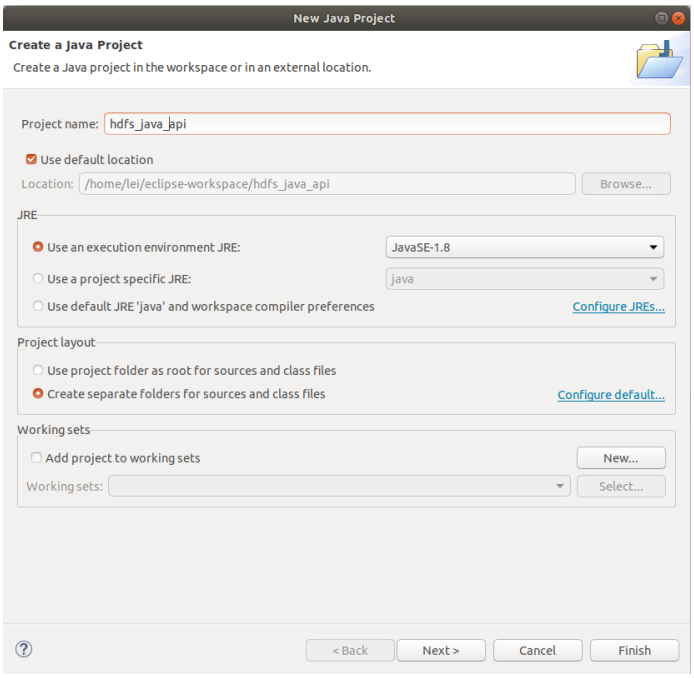
```
mkdir ~/big_data_tools/hadoop3lib
```

将/apps/hadoop/share/hadoop 目录下的 common，hdfs，mapreduce，yarn 4 个子目录中的 jar 文件以及这 4 个子目录下 lib 文件夹中的所有 jar 文件复制到~/big_data_tools/hadoop3lib 中。这些 jar 文件将作为外部的 jar 文件添加到工程中。

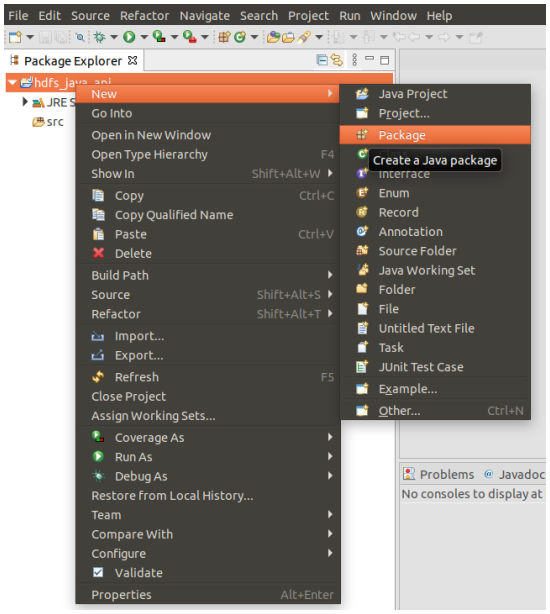
2. 新建 Java Project。点击【File】=>【New】=>【Java Project】。



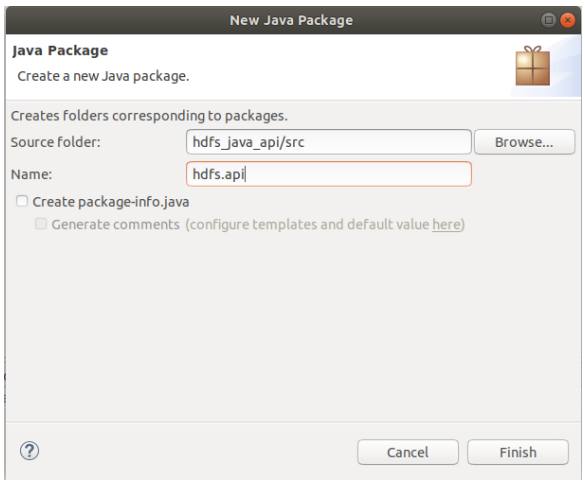
3. 在【New Java Project】对话框中填写 Project name，点击【Finish】。



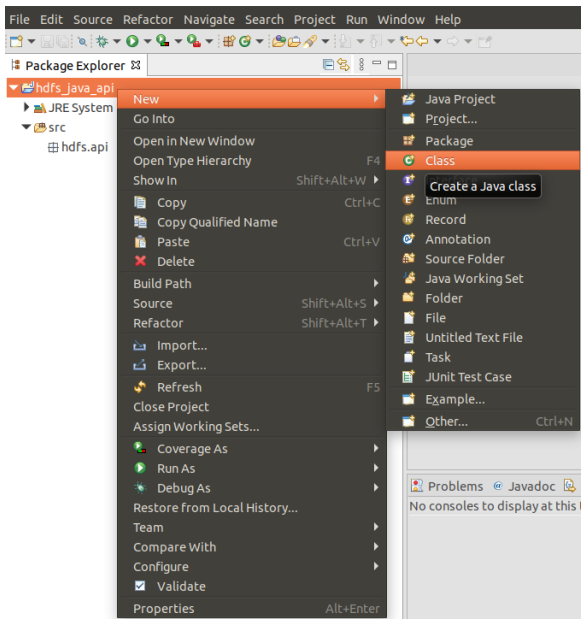
4. 新建 Package。在左边侧边栏工程名上点击右键，选择【New】=>【Package】。



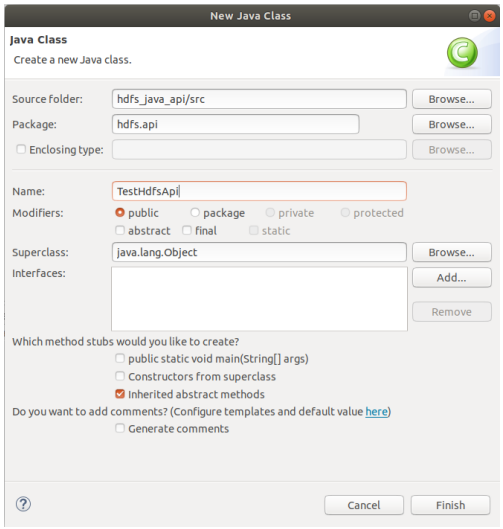
5. 在【New Java Package】对话框中填写 Package Name，其它保持默认值，点击【Finish】。



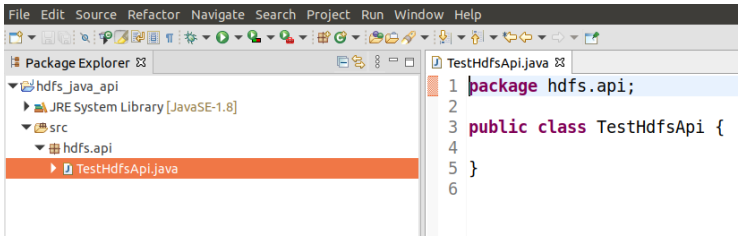
6. 新建 Class。在工程名上点击右键，选择【New】=>【Class】



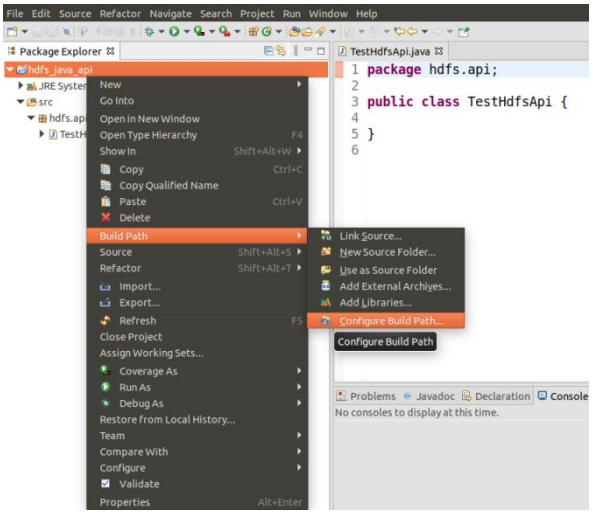
在【New Java Class】对话框中填写类名 TestHdfsApi，其它保持默认值，点击【Finish】。



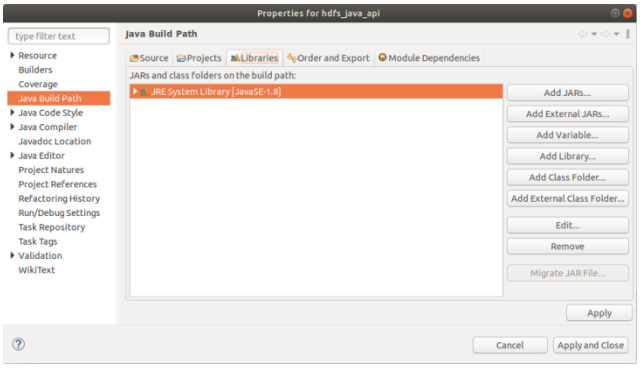
Eclipse 自动打开新建的 Class 文件。



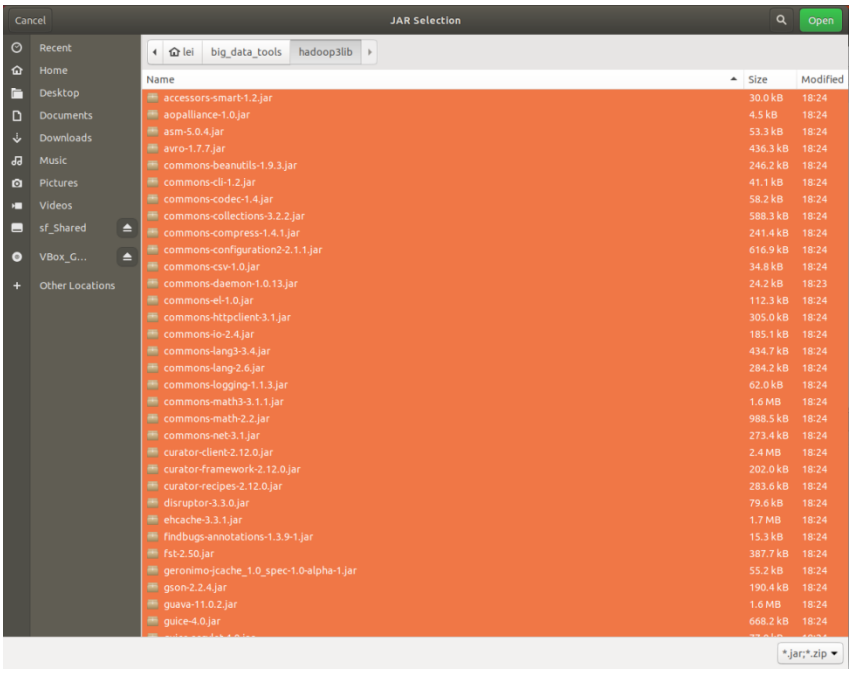
7. 添加外部 jar 文件。在工程名上点击右键，选择【Build Path】=>【Configure Build Path...】。



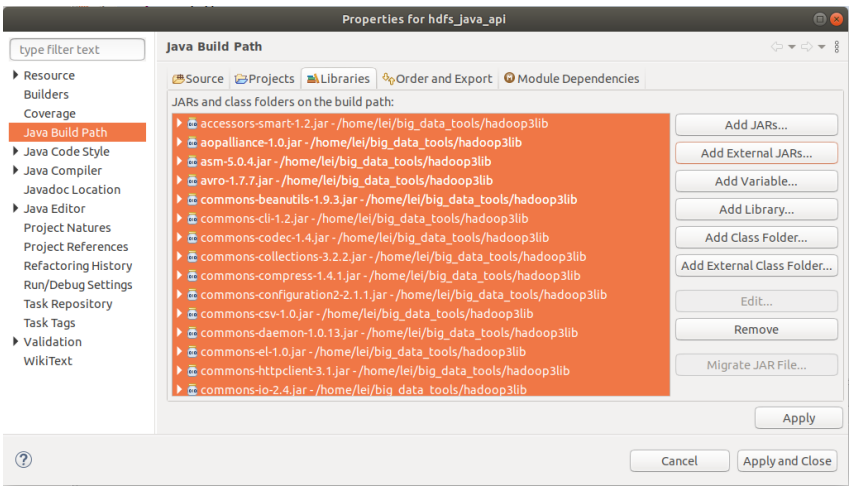
在【Java Build Path】对话框中，选择【Libraries】标签，点击右侧的【Add External JARs...】按钮。



在弹出的 JAR 文件选择窗口，切换到目录~/big_data_tools/hadoop3lib，全选目录中的 JAR 文件。点击【Open】。



回到【Java Build Path】对话框，点击【Apply and Close】。



8. 将下面代码添加到 TestHdfsApi.java 中。

```
package hdfs.api;

import java.net.URI;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.BlockLocation;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.hdfs.DistributedFileSystem;
import org.apache.hadoop.hdfs.protocol.DatanodeInfo;
import org.apache.hadoop.io.IOUtils;

public class TestHdfsApi {

        public static void main(String[] args) throws Exception {
                uploadFile();
                createFile();
                createDir();
                fileRename();
                deleteFile();
                readFile();
                isFileExists();
                fileLastModify();
                fileLocation();
                nodeList();
        }

        static FileSystem getFileSystem() throws Exception {
                URI uri = new URI("hdfs://localhost:9000/");
                // 使用 HDFS 文件系统并提供服务器路径，端口号在 core-site.xml 中配置
                FileSystem fileSystem = FileSystem.get(uri, new
Configuration());
                return fileSystem;
        }

        public static void uploadFile() throws Exception {

                FileSystem hdfs = getFileSystem();
                Path src = new Path("/data/testfile");
                Path dst = new Path("/myhadoop");
                FileStatus files[] = hdfs.listStatus(dst);
                for (FileStatus file : files) {
```

```
                        System.out.println(file.getPath());
                }
                System.out.println("------------after upload---------------
-----");
                // FileSystem.copyFromLocalFile 将本地文件上传到 HDFS 中
                hdfs.copyFromLocalFile(src, dst);
                files = hdfs.listStatus(dst);
                for (FileStatus file : files) {
                        // FileStatus.getPath 查看指定目录下的所有文件
                        System.out.println(file.getPath());
                }
        }

        public static void createFile() throws Exception {

                byte[] buff = "Hello Hadoop 888@Chinasofti\n".getBytes();
                FileSystem hdfs = getFileSystem();
                Path dfs = new Path("/testcreate");
                // FileSystem.create 可在 HDFS 上创建文件
                FSDataOutputStream outputStream = hdfs.create(dfs);
                outputStream.write(buff, 0, buff.length);
                outputStream.close();

        }

        public static void createDir() throws Exception {
                FileSystem hdfs = getFileSystem();
                Path dfs = new Path("/TestDir");
                // FileSystem.mkdir 可在 HDFS 上创建文件夹
                hdfs.mkdirs(dfs);
        }

        public static void fileRename() throws Exception {

                FileSystem hdfs = getFileSystem();
                Path frpaht = new Path("/myhadoop/testfile");
                Path topath = new Path("/myhadoop/testfile1");
                // FileSystem.rename 可为制定的文件重命名
                boolean isRename = hdfs.rename(frpaht, topath);
                String result = isRename ? "成功" : "失败";
                System.out.println("文件重命名结果为: " + result);
        }

        public static void deleteFile() throws Exception {
```

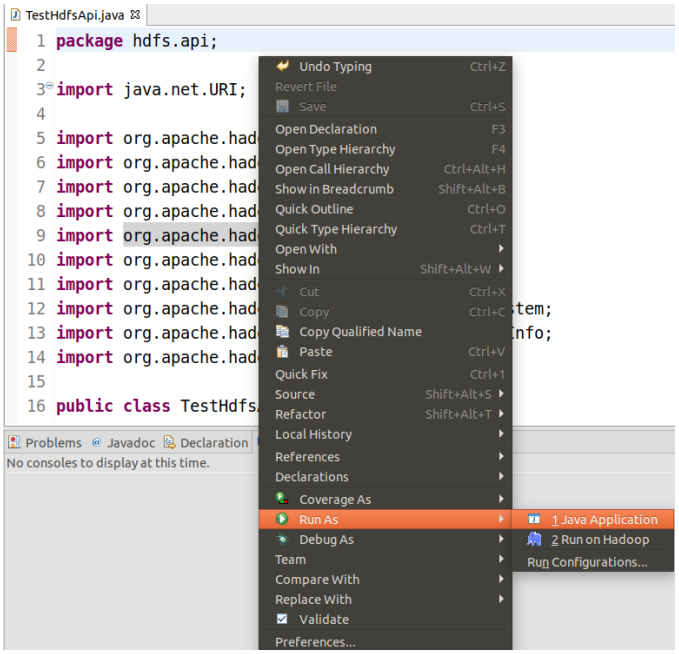
```
                FileSystem hdfs = getFileSystem();
                Path delef = new Path("/TestDir");
                // FileSystem.delete 删除指定的 HDFS 文件
                boolean isDeleted = hdfs.delete(delef, false);
                // 递归删除
                // boolean isDeleted=hdfs.delete(delef,true);
                System.out.println("Delete?" + isDeleted);
        }

        public static void readFile() throws Exception {
                FileSystem fileSystem = getFileSystem();
                // FileSystem.open 打开指定的文件进行读取
                FSDataInputStream openStream = fileSystem.open(new
Path("/testcreate"));
                IOUtils.copyBytes(openStream, System.out, 1024, false);
                IOUtils.closeStream(openStream);

        }

        public static void isFileExists() throws Exception {
                FileSystem hdfs = getFileSystem();
                Path findf = new Path("/myhadoop");
                // FileSystem.exists 判断指定的文件是否存在
                boolean isExists = hdfs.exists(findf);
                System.out.println("Exist?" + isExists);
        }

        public static void fileLastModify() throws Exception {
                FileSystem hdfs = getFileSystem();
                Path fpath = new Path("/testcreate");
                FileStatus fileStatus = hdfs.getFileStatus(fpath);
                // FileSystem.getModificationTime 可以查看指定文件的修改时间
                long modiTime = fileStatus.getModificationTime();
                System.out.println("testcreate 的修改时间是" + modiTime);
        }

        public static void fileLocation() throws Exception {
                FileSystem hdfs = getFileSystem();
                Path fpath = new Path("/testcreate");
                FileStatus filestatus = hdfs.getFileStatus(fpath);
                // FileSystem.getFileBlockLocations 可以查看指定文件在集群上的位
置
                BlockLocation[] blkLocations =
hdfs.getFileBlockLocations(filestatus,
```

```
                                0, filestatus.getLen());
                int blockLen = blkLocations.length;
                for (int i = 0; i < blockLen; i++) {
                        String[] hosts = blkLocations[i].getHosts();
                        System.out.println("block_" + i + "_location:" +
hosts[0]);
                }
        }

        public static void nodeList() throws Exception {
                FileSystem fs = getFileSystem();
                DistributedFileSystem hdfs = (DistributedFileSystem) fs;

                DatanodeInfo[] dataNodeStats = hdfs.getDataNodeStats();
                for (int i = 0; i < dataNodeStats.length; i++) {
                        // DatanodeInfo.getHostName 获取集群上的所有节点名称
                        System.out.println("DataNode_" + i + "_Name:"
                                        + dataNodeStats[i].getHostName());
                }
        }
}
```

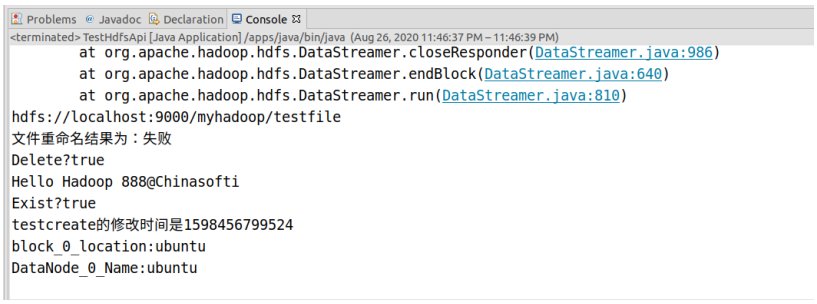
9. 创建文件 testfile 用于上传到 HDFS

```
touch /data/testfile
```

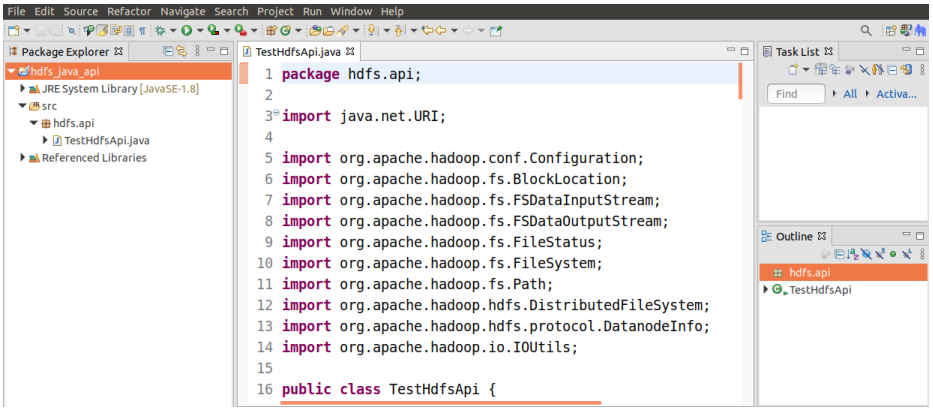
10. 在源代码窗口点击右键，选择【Run AS】=> 【Java Application】。



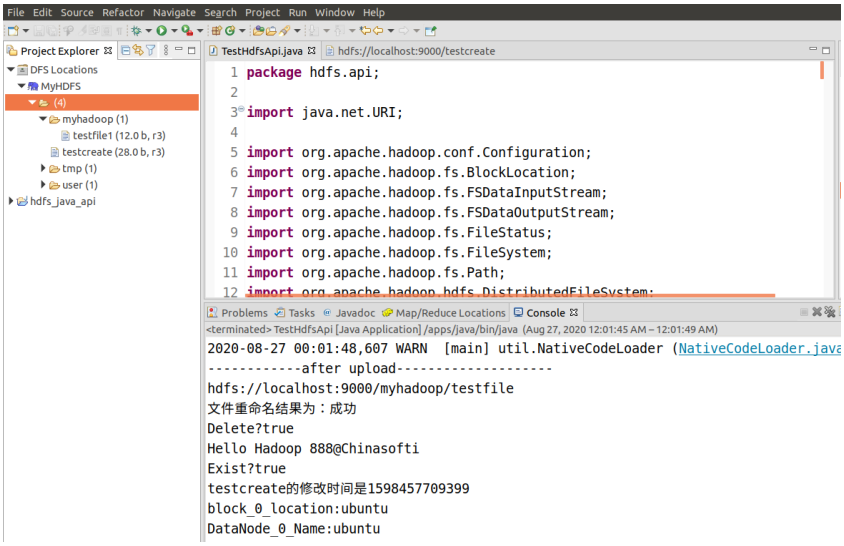
结果显示在下方的 Console 窗口中

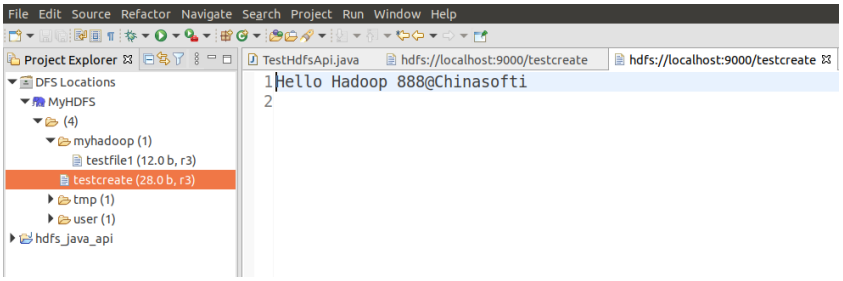


点击右上角蓝色小象图标，左边的侧边栏将由【Package Explorer】变为【Project Explorer】。



点开【DFS Locations】，可以看到我们程序上传到目录/myhadoop 的文件 testfile 已经被重命名为 testfile1，以及新建的文件 testcreate。在文件名上双击可以打开文件，可以看到其中的内容与我们添加的内容一致。没有看到新建的目录/TestDir，因为通过运行结果可知已经被成功删除。程序还判断了目录/myhadoop 是否存在，程序结果显示是存在的，与实际相符。综上可知，程序运行结果正常。


File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
DFS Locations
MyHDFS
(4)
myhadoop (1)
testfile1 (12.0 b, r3)
testcreate (28.0 b, r3)
tmp (1)
user (1)
hdfs_java_api
TestHdfsApi.java
hdfs://localhost:9000/testcreate
1 package hdfs.api;
2
3 import java.net.URI;
4
5 import org.apache.hadoop.conf.Configuration;
6 import org.apache.hadoop.fs.BlockLocation;
7 import org.apache.hadoop.fs.FSDataInputStream;
8 import org.apache.hadoop.fs.FSDataOutputStream;
9 import org.apache.hadoop.fs.FileStatus;
10 import org.apache.hadoop.fs.FileSystem;
11 import org.apache.hadoop.fs.Path;
12 import org.apache.hadoop.hdfs.DistributedFileSystem;
Problems Tasks Javadoc Map/Reduce Locations Console
<terminated> TestHdfsApi [Java Application] /apps/java/bin/java (Aug 27, 2020 12:01:45 AM – 12:01:49 AM)
2020-08-27 00:01:48,607 WARN [main] util.NativeCodeLoader (NativeCodeLoader.java
----------after upload----------
hdfs://localhost:9000/myhadoop/testfile
文件重命名结果为：成功
Delete?true
Hello Hadoop 888@Chinasofti
Exist?true
testcreate的修改时间是1598457709399
block_0_location:ubuntu
DataNode_0_Name:ubuntu


File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
DFS Locations
MyHDFS
(4)
myhadoop (1)
testfile1 (12.0 b, r3)
testcreate (28.0 b, r3)
tmp (1)
user (1)
hdfs_java_api
TestHdfsApi.java
hdfs://localhost:9000/testcreate
hdfs://localhost:9000/testcreate
1 Hello Hadoop 888@Chinasofti
2