|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 《大数据技术原理与应用》实验报告 | | | | |
| 题目： | 上机2 | 姓名 |  | 日期： |
| 实验环境：  操作系统：  PRETTY\_NAME="Ubuntu 22.04.1 LTS"  NAME="Ubuntu"  VERSION\_ID="22.04"  VERSION="22.04.1 LTS (Jammy Jellyfish)"  VERSION\_CODENAME=jammy  ID=ubuntu  ID\_LIKE=debian  HOME\_URL="https://www.ubuntu.com/"  SUPPORT\_URL="https://help.ubuntu.com/"  BUG\_REPORT\_URL="https://bugs.launchpad.net/ubuntu/"  PRIVACY\_POLICY\_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"  UBUNTU\_CODENAME=jammy  Hadoop 版本：3.3.6。  JDK 版本：1.8。  Java IDE：IntelliJ Idea Umltimate 远程开发） | | | | |
| 实验内容与完成情况：  （一）编程实现以下功能，并利用Hadoop提供的Shell命令完成相同任务：   1. 向HDFS中上传任意文本文件，如果指定的文件在HDFS中已经存在，则由用户来指定是追加到原有文件末尾还是覆盖原有的文件；  * Shell命令   检查文件是否存在，可以使用如下命令:   |  | | --- | | $ cd /usr/local/hadoop  $ ./bin/hdfs dfs -test -e text.txt |     执行完上述命令不会输出结果，需要继续输入命令查看结果：   |  | | --- | | $ echo $? |   如果结果显示文件已经存在，则用户可以选择追加到原来文件末尾或者覆盖原来文件，具体命令如下：   |  | | --- | | $ cd /usr/local/hadoop  $ ./bin/hdfs dfs -appendToFile local.txt text.txt #追加到原文件末尾  $ ./bin/hdfs dfs -copyFromLocal -f local.txt text.txt #覆盖原来文件，第一种命令形式  $ ./bin/hdfs dfs -cp -f file:///home/hadoop/local.txt text.txt #覆盖原来文件，第二种命令形式 |     实际上，也可以不用上述方式，而是采用如下命令来实现：   |  | | --- | | $ if $(hdfs dfs -test -e text.txt);  $ then $(hdfs dfs -appendToFile local.txt text.txt);  $ else $(hdfs dfs -copyFromLocal -f local.txt text.txt);  $ fi |     上述代码可视为一行代码，在终端中输入第一行代码后，代码不会立即被执行，可以继续输入第2行代码和第3行代码，直到输入 fi以后，上述代码才会真正执行。另外，上述代码中，直接使用了hdfs命令，而没有给出命令的路径，因为，这里假设已经配置了PATH环境变量，把hdfs命令的路径“/usr/local/hadoop/bin”写入了PATH环境变量中。   * Java代码  |  | | --- | | import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.\*;  import java.io.\*;  public class HDFSApi {  /\*\*  \* 判断路径是否存在  \*/  public static boolean test(Configuration conf, String path) throws IOException {  FileSystem fs = FileSystem.get(conf);  return fs.exists(new Path(path));  }  /\*\*  \* 复制文件到指定路径  \* 若路径已存在，则进行覆盖  \*/  public static void copyFromLocalFile(Configuration conf, String localFilePath, String remoteFilePath) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path localPath = new Path(localFilePath);  Path remotePath = new Path(remoteFilePath);  /\* fs.copyFromLocalFile 第一个参数表示是否删除源文件，第二个参数表示是否覆盖 \*/  fs.copyFromLocalFile(false, true, localPath, remotePath);  fs.close();  }    /\*\*  \* 追加文件内容  \*/  public static void appendToFile(Configuration conf, String localFilePath, String remoteFilePath) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path remotePath = new Path(remoteFilePath);  /\* 创建一个文件读入流 \*/  FileInputStream in = new FileInputStream(localFilePath);  /\* 创建一个文件输出流，输出的内容将追加到文件末尾 \*/  FSDataOutputStream out = fs.append(remotePath);  /\* 读写文件内容 \*/  byte[] data = new byte[1024];  int read = -1;  while ( (read = in.read(data)) > 0 ) {  out.write(data, 0, read);  }  out.close();  in.close();  fs.close();  }    /\*\*  \* 主函数  \*/  public static void main(String[] args) {  Configuration conf = new Configuration();  conf.set("fs.default.name","hdfs://localhost:9000");  String localFilePath = "/home/hadoop/text.txt"; // 本地路径  String remoteFilePath = "/user/hadoop/text.txt"; // HDFS路径  String choice = "append"; // 若文件存在则追加到文件末尾  // String choice = "overwrite"; // 若文件存在则覆盖    try {  /\* 判断文件是否存在 \*/  Boolean fileExists = false;  if (HDFSApi.test(conf, remoteFilePath)) {  fileExists = true;  System.out.println(remoteFilePath + " 已存在.");  } else {  System.out.println(remoteFilePath + " 不存在.");  }  /\* 进行处理 \*/  if ( !fileExists) { // 文件不存在，则上传  HDFSApi.copyFromLocalFile(conf, localFilePath, remoteFilePath);  System.out.println(localFilePath + " 已上传至 " + remoteFilePath);  } else if ( choice.equals("overwrite") ) { // 选择覆盖  HDFSApi.copyFromLocalFile(conf, localFilePath, remoteFilePath);  System.out.println(localFilePath + " 已覆盖 " + remoteFilePath);  } else if ( choice.equals("append") ) { // 选择追加  HDFSApi.appendToFile(conf, localFilePath, remoteFilePath);  System.out.println(localFilePath + " 已追加至 " + remoteFilePath);  }  } catch (Exception e) {  e.printStackTrace();  }  }  } |      1. 从HDFS中下载指定文件，如果本地文件与要下载的文件名称相同，则自动对下载的文件重命名；  * Shell命令  |  | | --- | | $ if $(hdfs dfs -test -e file:///home/hadoop/text.txt);  $ then $(hdfs dfs -copyToLocal text.txt ./text2.txt);  $ else $(hdfs dfs -copyToLocal text.txt ./text.txt);  $ fi |      * Java代码  |  | | --- | | import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.\*;  import java.io.\*;  public class HDFSApi {  /\*\*  \* 下载文件到本地  \* 判断本地路径是否已存在，若已存在，则自动进行重命名  \*/  public static void copyToLocal(Configuration conf, String remoteFilePath, String localFilePath) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path remotePath = new Path(remoteFilePath);  File f = new File(localFilePath);  /\* 如果文件名存在，自动重命名(在文件名后面加上 \_0, \_1 ...) \*/  if (f.exists()) {  System.out.println(localFilePath + " 已存在.");  Integer i = 0;  while (true) {  f = new File(localFilePath + "\_" + i.toString());  if (!f.exists()) {  localFilePath = localFilePath + "\_" + i.toString();  break;  }  }  System.out.println("将重新命名为: " + localFilePath);  }    // 下载文件到本地  Path localPath = new Path(localFilePath);  fs.copyToLocalFile(remotePath, localPath);  fs.close();  }    /\*\*  \* 主函数  \*/  public static void main(String[] args) {  Configuration conf = new Configuration();  conf.set("fs.default.name","hdfs://localhost:9000");  String localFilePath = "/home/hadoop/text.txt"; // 本地路径  String remoteFilePath = "/user/hadoop/text.txt"; // HDFS路径    try {  HDFSApi.copyToLocal(conf, remoteFilePath, localFilePath);  System.out.println("下载完成");  } catch (Exception e) {  e.printStackTrace();  }  }  } |      1. 将HDFS中指定文件的内容输出到终端中；  * Shell命令  |  | | --- | | $ hdfs dfs -cat text.txt |      * Java代码  |  | | --- | | import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.\*;  import java.io.\*;  public class HDFSApi {  /\*\*  \* 读取文件内容  \*/  public static void cat(Configuration conf, String remoteFilePath) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path remotePath = new Path(remoteFilePath);  FSDataInputStream in = fs.open(remotePath);  BufferedReader d = new BufferedReader(new InputStreamReader(in));  String line = null;  while ( (line = d.readLine()) != null ) {  System.out.println(line);  }  d.close();  in.close();  fs.close();  }    /\*\*  \* 主函数  \*/  public static void main(String[] args) {  Configuration conf = new Configuration();  conf.set("fs.default.name","hdfs://localhost:9000");  String remoteFilePath = "/user/hadoop/text.txt"; // HDFS路径    try {  System.out.println("读取文件: " + remoteFilePath);  HDFSApi.cat(conf, remoteFilePath);  System.out.println("\n读取完成");  } catch (Exception e) {  e.printStackTrace();  }  }  } |      1. 显示HDFS中指定的文件的读写权限、大小、创建时间、路径等信息；  * Shell命令  |  | | --- | | $ hdfs dfs -ls -h text.txt |      * Java代码  |  | | --- | | import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.\*;  import java.io.\*;  import java.text.SimpleDateFormat;  public class HDFSApi {  /\*\*  \* 显示指定文件的信息  \*/  public static void ls(Configuration conf, String remoteFilePath) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path remotePath = new Path(remoteFilePath);  FileStatus[] fileStatuses = fs.listStatus(remotePath);  for (FileStatus s : fileStatuses) {  System.out.println("路径: " + s.getPath().toString());  System.out.println("权限: " + s.getPermission().toString());  System.out.println("大小: " + s.getLen());  /\* 返回的是时间戳,转化为时间日期格式 \*/  Long timeStamp = s.getModificationTime();  SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  String date = format.format(timeStamp);  System.out.println("时间: " + date);  }  fs.close();  }    /\*\*  \* 主函数  \*/  public static void main(String[] args) {  Configuration conf = new Configuration();  conf.set("fs.default.name","hdfs://localhost:9000");  String remoteFilePath = "/user/hadoop/text.txt"; // HDFS路径    try {  System.out.println("读取文件信息: " + remoteFilePath);  HDFSApi.ls(conf, remoteFilePath);  System.out.println("\n读取完成");  } catch (Exception e) {  e.printStackTrace();  }  }  } |      1. 给定HDFS中某一个目录，输出该目录下的所有文件的读写权限、大小、创建时间、路径等信息，如果该文件是目录，则递归输出该目录下所有文件相关信息；  * Shell命令  |  | | --- | | $ cd /usr/local/hadoop  $ ./bin/hdfs dfs -ls -R -h /user/hadoop |      * Java代码  |  | | --- | | import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.\*;  import java.io.\*;  import java.text.SimpleDateFormat;  public class HDFSApi {  /\*\*  \* 显示指定文件夹下所有文件的信息（递归）  \*/  public static void lsDir(Configuration conf, String remoteDir) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path dirPath = new Path(remoteDir);  /\* 递归获取目录下的所有文件 \*/  RemoteIterator<LocatedFileStatus> remoteIterator = fs.listFiles(dirPath, true);  /\* 输出每个文件的信息 \*/  while (remoteIterator.hasNext()) {  FileStatus s = remoteIterator.next();  System.out.println("路径: " + s.getPath().toString());  System.out.println("权限: " + s.getPermission().toString());  System.out.println("大小: " + s.getLen());  /\* 返回的是时间戳,转化为时间日期格式 \*/  Long timeStamp = s.getModificationTime();  SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  String date = format.format(timeStamp);  System.out.println("时间: " + date);  System.out.println();  }  fs.close();  }    /\*\*  \* 主函数  \*/  public static void main(String[] args) {  Configuration conf = new Configuration();  conf.set("fs.default.name","hdfs://localhost:9000");  String remoteDir = "/user/hadoop"; // HDFS路径    try {  System.out.println("(递归)读取目录下所有文件的信息: " + remoteDir);  HDFSApi.lsDir(conf, remoteDir);  System.out.println("读取完成");  } catch (Exception e) {  e.printStackTrace();  }  }  } |      1. 提供一个HDFS内的文件的路径，对该文件进行创建和删除操作。如果文件所在目录不存在，则自动创建目录；  * Shell命令  |  | | --- | | $ if $(hdfs dfs -test -d dir1/dir2);  $ then $(hdfs dfs -touchz dir1/dir2/filename);  $ else $(hdfs dfs -mkdir -p dir1/dir2 && hdfs dfs -touchz dir1/dir2/filename);  $ fi  $ hdfs dfs -rm dir1/dir2/filename #删除文件 |      * Java代码  |  | | --- | | import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.\*;  import java.io.\*;  public class HDFSApi {  /\*\*  \* 判断路径是否存在  \*/  public static boolean test(Configuration conf, String path) throws IOException {  FileSystem fs = FileSystem.get(conf);  return fs.exists(new Path(path));  }    /\*\*  \* 创建目录  \*/  public static boolean mkdir(Configuration conf, String remoteDir) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path dirPath = new Path(remoteDir);  boolean result = fs.mkdirs(dirPath);  fs.close();  return result;  }  /\*\*  \* 创建文件  \*/  public static void touchz(Configuration conf, String remoteFilePath) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path remotePath = new Path(remoteFilePath);  FSDataOutputStream outputStream = fs.create(remotePath);  outputStream.close();  fs.close();  }    /\*\*  \* 删除文件  \*/  public static boolean rm(Configuration conf, String remoteFilePath) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path remotePath = new Path(remoteFilePath);  boolean result = fs.delete(remotePath, false);  fs.close();  return result;  }    /\*\*  \* 主函数  \*/  public static void main(String[] args) {  Configuration conf = new Configuration();  conf.set("fs.default.name","hdfs://localhost:9000");  String remoteFilePath = "/user/hadoop/input/text.txt"; // HDFS路径  String remoteDir = "/user/hadoop/input"; // HDFS路径对应的目录    try {  /\* 判断路径是否存在，存在则删除，否则进行创建 \*/  if ( HDFSApi.test(conf, remoteFilePath) ) {  HDFSApi.rm(conf, remoteFilePath); // 删除  System.out.println("删除路径: " + remoteFilePath);  } else {  if ( !HDFSApi.test(conf, remoteDir) ) { // 若目录不存在，则进行创建  HDFSApi.mkdir(conf, remoteDir);  System.out.println("创建文件夹: " + remoteDir);  }  HDFSApi.touchz(conf, remoteFilePath);  System.out.println("创建路径: " + remoteFilePath);  }  } catch (Exception e) {  e.printStackTrace();  }  }  } |      1. 提供一个HDFS的目录的路径，对该目录进行创建和删除操作。创建目录时，如果目录文件所在目录不存在，则自动创建相应目录；删除目录时，由用户指定当该目录不为空时是否还删除该目录；  * Shell命令   创建目录的命令如下：   |  | | --- | | $ hdfs dfs -mkdir -p dir1/dir2 |   删除目录的命令如下：   |  | | --- | | $ hdfs dfs -rmdir dir1/dir2 |   上述命令执行以后，如果目录非空，则会提示not empty，删除操作不会执行。如果要强制删除目录，可以使用如下命令：   |  | | --- | | $ hdfs dfs -rm -R dir1/dir2 |      * Java代码  |  | | --- | | import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.\*;  import java.io.\*;  public class HDFSApi {  /\*\*  \* 判断路径是否存在  \*/  public static boolean test(Configuration conf, String path) throws IOException {  FileSystem fs = FileSystem.get(conf);  return fs.exists(new Path(path));  }  /\*\*  \* 判断目录是否为空  \* true: 空，false: 非空  \*/  public static boolean isDirEmpty(Configuration conf, String remoteDir) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path dirPath = new Path(remoteDir);  RemoteIterator<LocatedFileStatus> remoteIterator = fs.listFiles(dirPath, true);  return !remoteIterator.hasNext();  }    /\*\*  \* 创建目录  \*/  public static boolean mkdir(Configuration conf, String remoteDir) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path dirPath = new Path(remoteDir);  boolean result = fs.mkdirs(dirPath);  fs.close();  return result;  }    /\*\*  \* 删除目录  \*/  public static boolean rmDir(Configuration conf, String remoteDir) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path dirPath = new Path(remoteDir);  /\* 第二个参数表示是否递归删除所有文件 \*/  boolean result = fs.delete(dirPath, true);  fs.close();  return result;  }    /\*\*  \* 主函数  \*/  public static void main(String[] args) {  Configuration conf = new Configuration();  conf.set("fs.default.name","hdfs://localhost:9000");  String remoteDir = "/user/hadoop/input"; // HDFS目录  Boolean forceDelete = false; // 是否强制删除    try {  /\* 判断目录是否存在，不存在则创建，存在则删除 \*/  if ( !HDFSApi.test(conf, remoteDir) ) {  HDFSApi.mkdir(conf, remoteDir); // 创建目录  System.out.println("创建目录: " + remoteDir);  } else {  if ( HDFSApi.isDirEmpty(conf, remoteDir) || forceDelete ) { // 目录为空或强制删除  HDFSApi.rmDir(conf, remoteDir);  System.out.println("删除目录: " + remoteDir);  } else { // 目录不为空  System.out.println("目录不为空，不删除: " + remoteDir);  }  }  } catch (Exception e) {  e.printStackTrace();  }  }  } |      1. 向HDFS中指定的文件追加内容，由用户指定内容追加到原有文件的开头或结尾；  * Shell命令   追加到原文件末尾的命令如下：   |  | | --- | | $ hdfs dfs -appendToFile local.txt text.txt |   追加到原文件的开头，在HDFS中不存在与这种操作对应的命令，因此，无法使用一条命令来完成。可以先移动到本地进行操作，再进行上传覆盖，具体命令如下：   |  | | --- | | $ hdfs dfs -get text.txt  $ cat text.txt >> local.txt  $ hdfs dfs -copyFromLocal -f text.txt text.txt |      * Java代码  |  | | --- | | import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.\*;  import java.io.\*;  public class HDFSApi {  /\*\*  \* 判断路径是否存在  \*/  public static boolean test(Configuration conf, String path) throws IOException {  FileSystem fs = FileSystem.get(conf);  return fs.exists(new Path(path));  }  /\*\*  \* 追加文本内容  \*/  public static void appendContentToFile(Configuration conf, String content, String remoteFilePath) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path remotePath = new Path(remoteFilePath);  /\* 创建一个文件输出流，输出的内容将追加到文件末尾 \*/  FSDataOutputStream out = fs.append(remotePath);  out.write(content.getBytes());  out.close();  fs.close();  }  /\*\*  \* 追加文件内容  \*/  public static void appendToFile(Configuration conf, String localFilePath, String remoteFilePath) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path remotePath = new Path(remoteFilePath);  /\* 创建一个文件读入流 \*/  FileInputStream in = new FileInputStream(localFilePath);  /\* 创建一个文件输出流，输出的内容将追加到文件末尾 \*/  FSDataOutputStream out = fs.append(remotePath);  /\* 读写文件内容 \*/  byte[] data = new byte[1024];  int read = -1;  while ( (read = in.read(data)) > 0 ) {  out.write(data, 0, read);  }  out.close();  in.close();  fs.close();  }  /\*\*  \* 移动文件到本地  \* 移动后，删除源文件  \*/  public static void moveToLocalFile(Configuration conf, String remoteFilePath, String localFilePath) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path remotePath = new Path(remoteFilePath);  Path localPath = new Path(localFilePath);  fs.moveToLocalFile(remotePath, localPath);  }    /\*\*  \* 创建文件  \*/  public static void touchz(Configuration conf, String remoteFilePath) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path remotePath = new Path(remoteFilePath);  FSDataOutputStream outputStream = fs.create(remotePath);  outputStream.close();  fs.close();  }    /\*\*  \* 主函数  \*/  public static void main(String[] args) {  Configuration conf = new Configuration();  conf.set("fs.default.name","hdfs://localhost:9000");  String remoteFilePath = "/user/hadoop/text.txt"; // HDFS文件  String content = "新追加的内容\n";  String choice = "after"; //追加到文件末尾  // String choice = "before"; // 追加到文件开头    try {  /\* 判断文件是否存在 \*/  if ( !HDFSApi.test(conf, remoteFilePath) ) {  System.out.println("文件不存在: " + remoteFilePath);  } else {  if ( choice.equals("after") ) { // 追加在文件末尾  HDFSApi.appendContentToFile(conf, content, remoteFilePath);  System.out.println("已追加内容到文件末尾" + remoteFilePath);  } else if ( choice.equals("before") ) { // 追加到文件开头  /\* 没有相应的api可以直接操作，因此先把文件移动到本地\*/  /\*创建一个新的HDFS，再按顺序追加内容 \*/  String localTmpPath = "/user/hadoop/tmp.txt";  // 移动到本地  HDFSApi.moveToLocalFile(conf, remoteFilePath, localTmpPath);  // 创建一个新文件  HDFSApi.touchz(conf, remoteFilePath);  // 先写入新内容  HDFSApi.appendContentToFile(conf, content, remoteFilePath);  // 再写入原来内容  HDFSApi.appendToFile(conf, localTmpPath, remoteFilePath);  System.out.println("已追加内容到文件开头: " + remoteFilePath);  }  }  } catch (Exception e) {  e.printStackTrace();  }  }  } |      1. 删除HDFS中指定的文件；  * Shell命令  |  | | --- | | $ hdfs dfs -rm text.txt |      * Java代码  |  | | --- | | import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.\*;  import java.io.\*;  public class HDFSApi {  /\*\*  \* 删除文件  \*/  public static boolean rm(Configuration conf, String remoteFilePath) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path remotePath = new Path(remoteFilePath);  boolean result = fs.delete(remotePath, false);  fs.close();  return result;  }    /\*\*  \* 主函数  \*/  public static void main(String[] args) {  Configuration conf = new Configuration();  conf.set("fs.default.name","hdfs://localhost:9000");  String remoteFilePath = "/user/hadoop/text.txt"; // HDFS文件    try {  if ( HDFSApi.rm(conf, remoteFilePath) ) {  System.out.println("文件删除: " + remoteFilePath);  } else {  System.out.println("操作失败（文件不存在或删除失败）");  }  } catch (Exception e) {  e.printStackTrace();  }  }  } |      1. 在HDFS中，将文件从源路径移动到目的路径。  * Shell命令  |  | | --- | | $ hdfs dfs -mv text.txt text2.txt |  * Java代码  |  | | --- | | import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.\*;  import java.io.\*;  public class HDFSApi {  /\*\*  \* 移动文件  \*/  public static boolean mv(Configuration conf, String remoteFilePath, String remoteToFilePath) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path srcPath = new Path(remoteFilePath);  Path dstPath = new Path(remoteToFilePath);  boolean result = fs.rename(srcPath, dstPath);  fs.close();  return result;  }    /\*\*  \* 主函数  \*/  public static void main(String[] args) {  Configuration conf = new Configuration();  conf.set("fs.default.name","hdfs://localhost:9000");  String remoteFilePath = "hdfs:///user/hadoop/text.txt"; // 源文件HDFS路径  String remoteToFilePath = "hdfs:///user/hadoop/new.txt"; // 目的HDFS路径    try {  if ( HDFSApi.mv(conf, remoteFilePath, remoteToFilePath) ) {  System.out.println("将文件 " + remoteFilePath + " 移动到 " + remoteToFilePath);  } else {  System.out.println("操作失败(源文件不存在或移动失败)");  }  } catch (Exception e) {  e.printStackTrace();  }  }  } |     （二）编程实现一个类“MyFSDataInputStream”，该类继承“org.apache.hadoop.fs.FSDataInputStream”，要求如下：实现按行读取HDFS中指定文件的方法“readLine()”，如果读到文件末尾，则返回空，否则返回文件一行的文本。   |  | | --- | | import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.FSDataInputStream;  import org.apache.hadoop.fs.FileSystem;  import org.apache.hadoop.fs.Path;  import java.io.\*;  public class MyFSDataInputStream extends FSDataInputStream {  public MyFSDataInputStream(InputStream in) {  super(in);  }    /\*\*  \* 实现按行读取  \* 每次读入一个字符，遇到"\n"结束，返回一行内容  \*/  public static String readline(BufferedReader br) throws IOException {  char[] data = new char[1024];  int read = -1;  int off = 0;  // 循环执行时，br 每次会从上一次读取结束的位置继续读取  //因此该函数里，off 每次都从0开始  while ( (read = br.read(data, off, 1)) != -1 ) {  if (String.valueOf(data[off]).equals("\n") ) {  off += 1;  break;  }  off += 1;  }    if (off > 0) {  return String.valueOf(data);  } else {  return null;  }  }  /\*\*  \* 读取文件内容  \*/  public static void cat(Configuration conf, String remoteFilePath) throws IOException {  FileSystem fs = FileSystem.get(conf);  Path remotePath = new Path(remoteFilePath);  FSDataInputStream in = fs.open(remotePath);  BufferedReader br = new BufferedReader(new InputStreamReader(in));  String line = null;  while ( (line = MyFSDataInputStream.readline(br)) != null ) {  System.out.println(line);  }  br.close();  in.close();  fs.close();  }    /\*\*  \* 主函数  \*/  public static void main(String[] args) {  Configuration conf = new Configuration();  conf.set("fs.default.name","hdfs://localhost:9000");  String remoteFilePath = "/user/hadoop/text.txt"; // HDFS路径  try {  MyFSDataInputStream.cat(conf, remoteFilePath);  } catch (Exception e) {  e.printStackTrace();  }  }  } |     （三）查看Java帮助手册或其它资料，用“java.net.URL”和“org.apache.hadoop.fs.FsURLStreamHandlerFactory”编程完成输出HDFS中指定文件的文本到终端中。   |  | | --- | | import org.apache.hadoop.fs.\*;  import org.apache.hadoop.io.IOUtils;  import java.io.\*;  import java.net.URL;  public class HDFSApi {  static{  URL.setURLStreamHandlerFactory(new FsUrlStreamHandlerFactory());  }    /\*\*  \* 主函数  \*/  public static void main(String[] args) throws Exception {  String remoteFilePath = "hdfs:///user/hadoop/text.txt"; // HDFS文件  InputStream in = null;  try{  /\* 通过URL对象打开数据流，从中读取数据 \*/  in = new URL(remoteFilePath).openStream();  IOUtils.copyBytes(in,System.out,4096,false);  } finally{  IOUtils.closeStream(in);  }  }  } | | | | | |
| 出现的问题：  1． 直接用hdfs://localhost:9000如下报错： | | | | |
| 解决方案（列出遇到的问题和解决办法，列出没有解决的问题）：  2.对于在设备上虚拟机或者虚拟服务比较多的，直接将localhost修改为具体的IP地址或主机地址。 | | | | |