# 第3章 HDFS的Shell操作（开发重点）

1．基本语法



bin/hadoop fs +具体命令

2．命令大全

[master@master hadoop-2.7.3]$ bin/hadoop fs

3．常用命令实操

**HDFS ——> HDFS:**

mkdir、cp、rm、ls、du、df、cat

**本地 ——>HDFS:**

put

copyFromLocal

moveFromLocal

appendToFile

**HDFS ——> 本地**：

get

getmerge

copyToLocal

（0）启动Hadoop集群

[master@master hadoop-2.7.3]$ sbin/start-dfs.sh

另外一台机器

[master@hadoop1 hadoop-2.7.3]$ sbin/start-yarn.sh

（1）-help：输出这个命令参数

[master@master hadoop-2.7.3]$ hadoop fs -help rm

（2）**HDFS ——> HDFS:**

mkdir、cp、rm、ls（显示目录信息）:

创建一个文件夹

hadoop fs -mkdir –p /test

拷贝这个文件夹：

hadoop fs -cp /test /test1

删除这个文件夹：

hadoop fs -rm -r /test /yanshi

删除所有则是：

hadoop fs -rm -r hdfs://master:9000/\*

-ls: 在本地显示HDFS目录信息

hadoop fs -ls /

（3）本地到HDFS ：

-put(-copyFromLocal)、-moveFromLocal（剪切），–appendToFile（追加）

在本地创建一个目录

[master@master hadoop-2.7.3]$ mkdir test

[master@master hadoop-2.7.3]$ cd test

创建一个文件并编辑

[master@master test]$ cat <<EOF>>1.txt

[master@master test]$ 111

[master@master test]$ 111

[master@master test]$ 111

[master@master test]$ EOF

[master@master test]$ cat 1.txt

1.put命令：把本地文件上传到HDFS上

[master@master test]$ hadoop fs -put 1.txt /test

2.copyFromLocal和put命令是一样

[master@master test]$hadoop fs -help copyFromLocal

[master@master test]$ hadoop fs -copyFromLocal 1.txt /test

3. moveFromLocal 命令，相当于剪切

[master@master test]$ hadoop fs -moveFromLocal 1.txt /test1

4. appendToFile命令，追加

[master@master test]$ cat <<EOF>>2.txt

[master@master test]$ 222

[master@master test]$ 222

[master@master test]$ 222

[master@master test]$ EOF

[master@master test]$ cat 2.txt

[master@master test]$hadoop fs -appendToFile 2.txt /test/1.txt

5. cat命令，查看文件

[master@master test]$ hadoop fs -cat /test/1.txt

（4）从HDFS拷贝到本地：

1.-get（-copyToLocal：）下载

[master@master test]$hadoop fs -get /test/1.txt ./

下载为自己起的文件名

[master@master test]$hadoop fs -get /test/1.txt ./111.txt

copyToLocal 命令，同get

[master@master test]$hadoop fs -help copyToLocal

2. getmerge合并命令，比如HDFS的目录 /test下有多个文件:1,2,3...

[master@master test]$ cat <<EOF>>3.txt

[master@master test]$ 333

[master@master test]$ 333

[master@master test]$ 333

[master@master test]$ EOF

[master@master test]$ hadoop fs -put 3.txt /test

[master@master test]$hadoop fs -getmerge /test/\*

./yiqi.txt

（5）tail命令，查看文件的内容

补充：Linux里面的tail用法

1.1 tail 命令可用于查看文件的内容，有一个常用的参数

[master@master test]$ tail 1.txt

1.2 tail -f filename会把filename文件里的最尾部的内容显示在屏幕上，并且不断刷新，只要 filename 更新就可以看到最新的文件内容

在一个xshell连接的master中,输入

[master@master test]$ tail -f 1.txt

再连接另外一个master

首先进入相同的目录，然后输入。“1”代表追加的内容

[master@master test]$ echo 1 >> 1.txt

结束显示的状态，按键盘上的control + C

HDFS中的tail使用：

[master@master test]$ hadoop fs -tail -f /test/1.txt

再连接另外一个master

首先进入相同的目录，然后输入。“1”代表追加的内容

[master@master test]$ hadoop fs -appendToFile 3.txt /test/1.txt

（6）设置副本数量 setrep

1.设置为大于实际数据节点的数量

[master@master test]$ hadoop fs -setrep 10 /test/1.txt

2.设置少于副本冗余的数量，系统会删除

[master@master test]$ hadoop fs -setrep 1 /test/1.txt

3.设置少于副本冗余的数量，且小于数据节点的数量，系统会增加

[master@master test]$ hadoop fs -setrep 2 /test/1.txt

（7）-du,-df统计文件夹的大小信息: d 相当于 disk 磁盘

补充：本地linux ：

1. **df(disk free)**通过文件系统来快速获取空间大小的信息，当我们删除一个文件的时候，这个文件不是马上就在文件系统当中消失了，而是暂时消失了，当所有程序都不用时，才会根据OS的规则释放掉已经删除的文件，当文件系统也确定删除了该文件后，这时候du与df就一致了。简单的说，检查linux服务器的文件系统的磁盘空间占用情况

方法：

“df -h”以更易读的方式显示目前磁盘空间和使用情况。

2.**du(disk usage)**是通过搜索文件来计算每个文件的大小然后累加，du能看到的文件只是一些当前存在的，没有被删除的。他计算的大小就是当前他认为存在的所有文件大小的累加和。

方法：

在命令行中输入 “du -sh \*”可以查看每个文件夹的大小。此举可以快速定位大文件所存在的位置。

[master@master hadoop-2.7.3]$ df -h

./ 表示当前目录

[master@master hadoop-2.7.3]$ du -h ./

[master@master hadoop-2.7.3]$ du -h /home

[master@master hadoop-2.7.3]$ du -sh /home

在HDFS中：

[master@master hadoop-2.7.3]$ hadoop fs -df -h

[master@master hadoop-2.7.3]$ hadoop fs -du -h /test

（8）判断文件是否存在

$ hadoop fs -e /test/1.txt

$ echo $?

$ hadoop fs -help test

# 第3章 HDFS客户端操作（开发重点）

## 3.1 HDFS客户端环境准备

1．根据自己电脑的操作系统拷贝对应的编译后的hadoop jar包到非中文路径（例如：D:\Develop\hadoop-2.7.2），如图3-4所示。

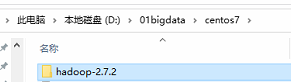


图3-4 编译后的hadoop jar包

2．配置HADOOP\_HOME环境变量，如图3-5所示。

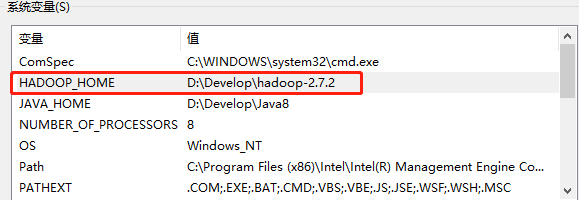


图3-5 配置HADOOP\_HOME环境变量

3. 配置Path环境变量，如图3-6所示。

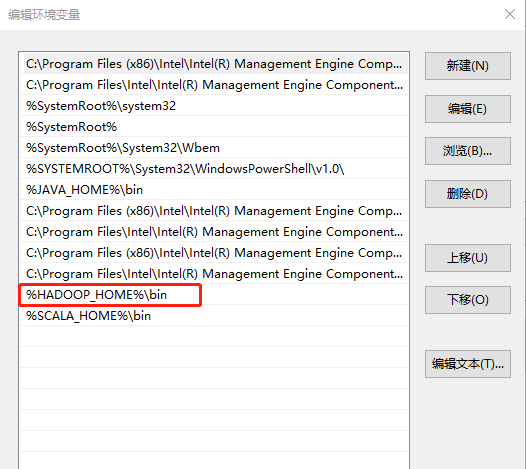


图3-6 配置Path环境变量

4．用idea创建一个Maven工程HdfsClientDemo

5．导入相应的依赖坐标+日志添加

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>RELEASE</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>

注意：如果Eclipse/Idea打印不出日志，在控制台上只显示

1.log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).

2.log4j:WARN Please initialize the log4j system properly.

3.log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.

需要在项目的src/main/resources目录下，新建一个文件，命名为“log4j.properties”，在文件中填入

log4j.rootLogger=INFO, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n

log4j.appender.logfile=org.apache.log4j.FileAppender

log4j.appender.logfile.File=target/spring.log

log4j.appender.logfile.layout=org.apache.log4j.PatternLayout

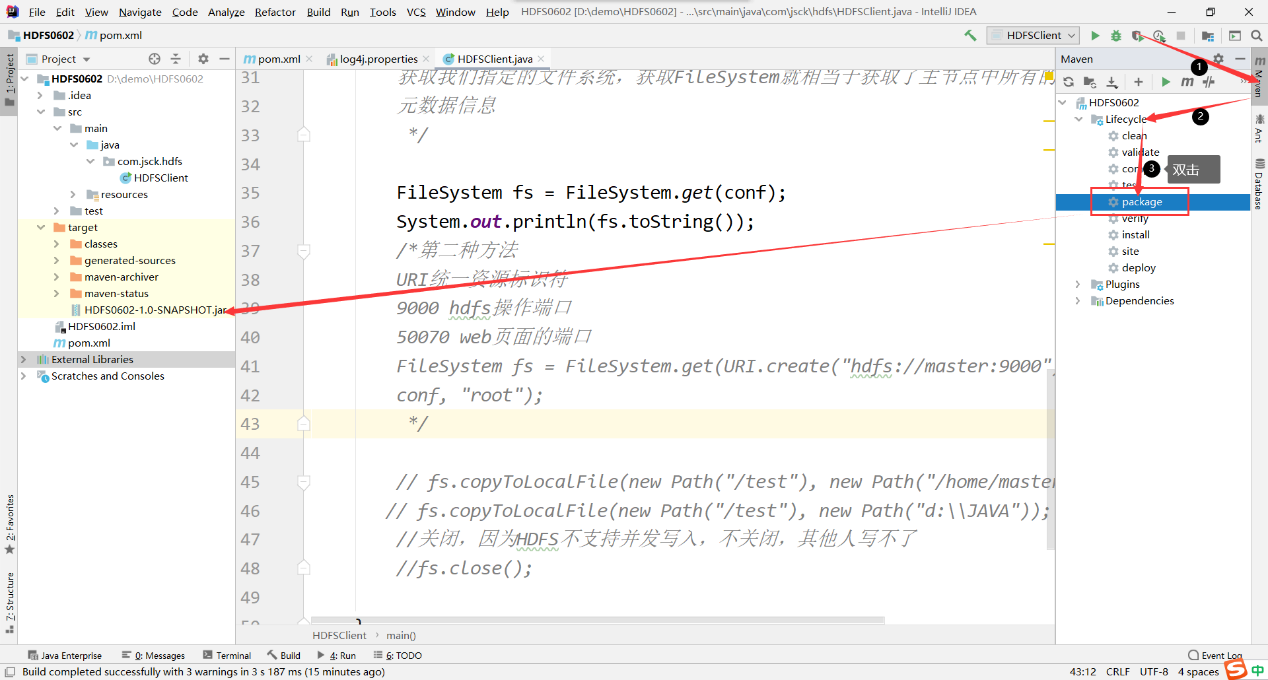
log4j.appender.logfile.layout.ConversionPattern=%d %p [%c] - %m%n

6．创建包名：com.jsck.hdfs

7．创建HdfsClient类

package com.jsck.hdfs;  
  
  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.FileSystem;  
import org.apache.hadoop.fs.Path;  
  
import java.io.IOException;  
import java.net.URI;  
  
public class HdfsClient{  
 */\*写了一个静态方法，通过类名·出来  
 \*如果不是，要实例化才能使用，  
 \* 什么是实例化？就是创建一个对象Configuration conf = new Configuration();  
 \* new后面写什么该类的构造器  
 \*方法名为类名，而且方法没有返回值。并不是返回值为空  
 \*这个类中有几个构造器，那么就有几种方式去创建该类的对象  
 \*/* public static void main(String[] args) throws IOException, InterruptedException {  
 *//该类的对象封装了客户端或者服务器的配置* Configuration conf = new Configuration();  
  
 */\*第一种方法  
 设置Configuration对象，  
 设置的目的是来指定我们要操作的文件系统  
 \*/* conf.set("fs.defaultFS","hdfs://master:9000");  
 */\*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，通过FileSystem的静态方法get获得该对象  
 \*get方法从conf中的一个参数fs.defaultFS的配置判断具体是什么类型的文件系统  
 \*/  
 /\*  
 获取我们指定的文件系统，获取FileSystem就相当于获取了主节点中所有的  
 元数据信息  
 \*/* FileSystem fs = FileSystem.*get*(URI.*create*("hdfs://master:9000"),conf, "root");  
 System.*out*.println(fs.toString());  
 Path src=new Path("D:/win\_hadoop/hadoop/test.txt");  
 Path dst=new Path("/");  
 fs.copyFromLocalFile(src,dst);  
 fs.close();  
 */\*第二种方法  
 URI统一资源标识符  
 9000 hdfs操作端口  
 50070 web页面的端口  
 FileSystem fs = FileSystem.get(URI.create("hdfs://master:9000"),  
 conf, "root");  
 \*/  
  
 // fs.copyToLocalFile(new Path("/test"), new Path("/home/master/apps/a"));  
 // fs.copyToLocalFile(new Path("/test"), new Path("d:\\JAVA"));  
 //关闭，因为HDFS不支持并发写入，不关闭，其他人写不了  
 //fs.close();* }  
}

**打jar包**

1.生成jar包

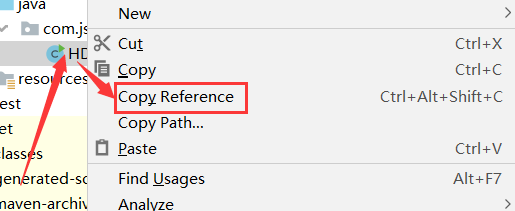
2．右键复制jar包，拷贝到电脑上，并重命名，1.jar

3. 拷贝到xshell上

4.在xshell上输入命令：

$ hadoop jar 1.jar com.jsck.hdfs.HDFSClient

com.jsck.hdfs.HDFSClient 为主程序的类，右键复制。

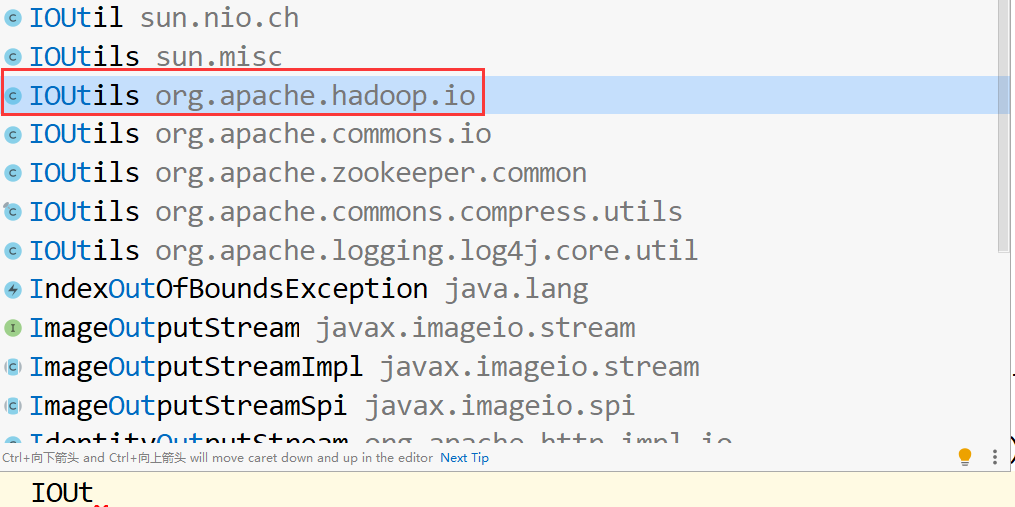


2.上传，创建文件，重命名，删除

**package** com.hwua.com;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.FileSystem;  
**import** org.apache.hadoop.fs.Path;  
**import** org.junit.After;  
**import** org.junit.Before;  
**import** org.junit.Test;  
**import** java.io.IOException;  
**import** java.net.URI;  
  
**public class** HdfsClient {  
 **private** FileSystem **fs**;  
 @Before  
 **public void** before() **throws** IOException, InterruptedException {  
 //当给别人写方法抛异常，如果用别人写的方法就处理

//一个函数要么实现业务逻辑，要么实现功能性的，功能性的一般是给别人写的，此时碰到异常不知道该如何处理，就抛出去。当用逻辑写的时候，需要处理根据需求处理异常

Configuration configuration = **new** Configuration();  
 **fs** = FileSystem.*get*(URI.*create*(**"hdfs://master:9000"**),  
 configuration,  
 **"root"**);  
 System.***out***.println(**"执行了before"**);  
 }  
  
 @Test  
 *//下载* **public void** get() **throws** IOException, InterruptedException {  
 *//1.获取一个HDFS的抽象封装对象* Configuration configuration = **new** Configuration();  
 FileSystem fs = FileSystem.*get*(URI.*create*(**"hdfs://master:9000"**),  
 configuration,  
 **"root"**);  
 *//2.用这个对象操作文件系统，下载* fs.copyToLocalFile(**new** Path(**"/test"**), **new** Path(**"d:\\Java"**));  
 *//关闭，因为HDFS不支持并发写入，不关闭，其他人写不了* fs.close();  
 }  
 @Test  
 *//创建* **public void** mk() **throws** IOException {  
 **fs**.mkdirs(**new** Path(**"/test1"**));  
 }  
 @Test  
 *//上传* **public void** put() **throws** IOException {  
 **fs**.copyFromLocalFile(**new** Path(**"D:\\Java\\test\\123.txt"**),**new** Path(**"/test"**));  
 }  
 @Test  
 *//重命名* **public void** rename() **throws** IOException {  
 **fs**.rename(**new** Path(**"/test1"**),**new** Path(**"/test2"**));  
 }  
 @Test  
 *//删除* **public void** delete() **throws** IOException {  
 *//这里的true，相当于-recursive 递归删除* **boolean** delete = **fs**.delete(**new** Path(**"/test2"**), **true**);  
 **if**(delete){  
 System.***out***.println(**"删除成功"**);  
 }**else**{  
 System.***out***.println(**"删除失败"**);  
 }  
 }

@Test  
*//追加***public void** append() **throws** IOException {  
 *//创建一个文件读入流* FileInputStream in = **new** FileInputStream(**"c:\\1.txt"**);  
 *//创建一个文件输出流，输出的内容将追加到文件末尾* FSDataOutputStream append = **fs**.append(**new** Path(**"/test2/1.txt"**), 1024);  
 *//读写文件内容*IOUtils.*copyBytes*(in,append,1024,**true**);  
}  
  
@Test  
*//查看文件和文件夹***public void** ls() **throws** IOException {  
 *//查看根目录* FileStatus[] fileStatuses = **fs**.listStatus(**new** Path(**"/"**));  
 *//遍历，iter* **for** (FileStatus fileStatus : fileStatuses) {  
 **if** (fileStatus.isFile()){  
 System.***out***.println(**"以下信息是一个文件的信息"**);  
 *//fileStatus.getPath().sout 可以这么打* System.***out***.println(fileStatus.getPath());  
 */\*获取信息，AccessTime 访问，存取(计算机文件)时间  
 Permission 权限信息  
 ModificationTime 修改时间  
 Replication 拷贝;重复 副本的数量  
 Symlink 符号链接  
 \* \*/* }**else** {  
 System.***out***.println(**"这是一个文件夹"**);  
 System.***out***.println(fileStatus.getPath());  
 }  
 }  
}  
此时，在XSHELL上操作Linux，进到hadoop路径：

把hdfs-site.xml 文件下载下来

输入命令： sz hdfs-site.xml

或者是 电脑里面配置的hadoop目录去拷贝，

然后拷贝到idea项目里，在resources里  
@Test  
*//自定义上传副本数量***public void** newput() **throws** IOException, InterruptedException {  
 *//设置配置文件* Configuration configuration = **new** Configuration();  
 *//键值对，name，值这里写了，就会使用这里的，不写使用配置文件的，* configuration.setInt(**"dfs.replication"**,1);  
  
 **fs** = FileSystem.*get*(URI.*create*(**"hdfs://master:9000"**),  
 configuration,  
 **"root"**);  
 **fs**.copyFromLocalFile(**new** Path(**"D:\\test\\123.txt"**),**new** Path(**"/test"**));  
}

@After  
 **public void** after() **throws** IOException {  
 System.***out***.println(**"执行了after"**);  
 **fs**.close();  
 }  
}

2.判断文件是否存在

**package** com.hwua.com;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.FileSystem;  
**import** org.apache.hadoop.fs.Path;  
  
**import** java.io.IOException;  
  
**public class** HdfsExist {  
 **public static void** main(String[] args) **throws** IOException {  
 *//判断文件是否存在  
 //filename后面输入的是文件的名字* String filename =**"/test/1.txt"**;  
 Configuration configuration = **new** Configuration();  
 configuration.set(**"fs.defaultFS"**,**"hdfs://master:9000"**);  
 configuration.set(**"fs.hdfs.impl"**,**"org.apache.hadoop.hdfs.DistributedFileSystem"**);

System.*setProperty*(**"HADOOP\_USER\_NAME"**,**"root"**);

FileSystem fs = FileSystem.*get*(configuration);  
 **if** (fs.exists(**new** Path(filename))){  
 System.***out***.println(**"文件存在"**);  
 }**else** {  
 System.***out***.println(**"文件不存在"**);  
 }  
 fs.close();  
 }  
}

3.写入文件

**package** com.hwua.com;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.FSDataOutputStream;  
**import** org.apache.hadoop.fs.FileSystem;  
**import** org.apache.hadoop.fs.Path;  
  
**import** java.io.IOException;  
**import** java.net.URI;  
  
**public class** HdfsWrite {  
 **public static void** main(String[] args) {  
 **try** {  
 Configuration configuration = **new** Configuration();  
*// configuration.set("fs.defaultFS","hdfs://master:9000");  
// configuration.set("fs.hdfs.impl","org.apache.hadoop.hdfs.DistributedFileSystem");  
// System.setProperty(****"HADOOP\_USER\_NAME"****,****"root"****);*

*// FileSystem fs = FileSystem.get(configuration);* FileSystem fs = FileSystem.*get*(URI.*create*(**"hdfs://master:9000"**),  
 configuration, **"root"**);  
 *//要写入的内容* **byte**[] buff = **"Hello World1"**.getBytes();  
 *//要写的文件名* String filename = **"/test/1.txt"**;  
 FSDataOutputStream os = fs.create(**new** Path(filename));  
 os.write(buff,0,buff.**length**);  
 System.***out***.println(**"Create:"**+filename);  
 os.close();  
 fs.close();  
 } **catch** (IOException | InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
}

4.读文件

**package** com.hwua.com;  
  
**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.BufferedReader;  
**import** java.io.IOException;  
**import** java.io.InputStreamReader;  
  
**public class** HdfsReader {  
 **public static void** main(String[] args) **throws** IOException {  
 Configuration configuration = **new** Configuration();  
 configuration.set(**"fs.defaultFS"**,**"hdfs://master:9000"**);  
 configuration.set(**"fs.hdfs.impl"**,**"org.apache.hadoop.hdfs.DistributedFileSystem"**);

System.*setProperty*(**"HADOOP\_USER\_NAME"**,**"root"**);

FileSystem fs = FileSystem.*get*(configuration);  
 Path file = **new** Path(**"/test/1.txt"**);  
 FSDataInputStream getIt = fs.open(file);  
 BufferedReader d = **new** BufferedReader(**new** InputStreamReader(getIt));  
 *//读取文件一行* String content = d.readLine();  
 System.***out***.println(content);  
 d.close();  
 fs.close();  
 }  
}

