

**数学与信息学院学生实验报告**

**实验课程名称：** **教师：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验项目名称** | **熟悉常用的HDFS操作** | | | **实验成绩** |  |
| **学生姓名** |  | **学 号** |  | **年级专业**  **班级** |  |
| **小组成员** | **无** | | | **实验日期** | **2019年 10 月** |

# 实验目的和要求

## 实验目的

① 理解HDFS在Hadoop体系结构中的角色，

② 熟练使用HDFS操作常用的Shell命令，

③ 熟悉HDFS操作常用的Java API。

## 实验软硬件环境

① 操作系统：Linux Ubuntu 16.04.6

② Java：jdk1.8.0\_221

③ Apache Hadoop：3.2.0

④ IDE：IntellJ IDEA Ultimate 2019.2.3

## 实验要求

① 编程实现指定的HDFS相关操作，并利用Hadoop提供的Shell命令完成相同任务；

②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“readLi

ne()”，如果读到文件末尾，则返回空，否则返回文件一行的文本。

③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。

# 实验记录

## 通过Shell命令与Java API 实现HDFS相关操作

### 在实现HDFS相关操作之前，首先要建立与hadoop连接，这边我封装了单例配置类供所有程序使用，通过这个单例配置类，可以获取到创建FileSystem必要的配置信息：

import org.apache.hadoop.conf.Configuration;  
  
*/\*\*  
 \** ***@author*** *aabbcczhy  
 \** ***@date*** *2019-10-08 11:45:00  
 \*/*public class MyConfiguration {  
 private static MyConfiguration *instance*;  
 private Configuration conf;  
 public static MyConfiguration getInstance(){  
 if(*instance* == null){  
 synchronized (MyConfiguration.class){  
 if(*instance* == null){  
 *instance* = new MyConfiguration();  
 }  
 }  
 }  
 return *instance*;  
 }  
  
 private MyConfiguration(){  
 conf = new Configuration();  
 conf.set("fs.defaultFS","hdfs://localhost:9000");  
 conf.set("fs.hdfs.impl","org.apache.hadoop.hdfs.DistributedFileSystem");

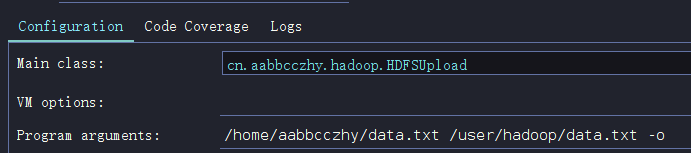
conf.set("dfs.support.append","true");  
 conf.set("dfs.client.block.write.replace-datanode-on-failure.policy", "NEVER");  
 conf.set("dfs.client.block.write.replace-datanode-on-failure.enable", "true");  
 }  
  
 public Configuration getConfiguration(){  
 return conf;  
 }  
}

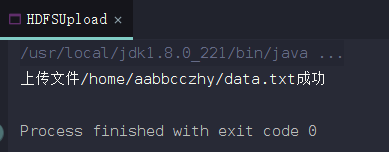
### 向HDFS中上传任意文本文件，如果指定的文件在HDFS中已经存在，则由用户来指定是追加到原有文件末尾还是覆盖原有的文件；

import org.apache.hadoop.fs.FSDataOutputStream;  
import org.apache.hadoop.fs.*FileSystem*;  
import org.apache.hadoop.fs.Path;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
  
*/\*\*  
 \** ***@author*** *aabbcczhy  
 \** ***@date*** *2019-10-13 20:21:59  
 \*/*public class HDFSUpload {  
 private *FileSystem* fs;

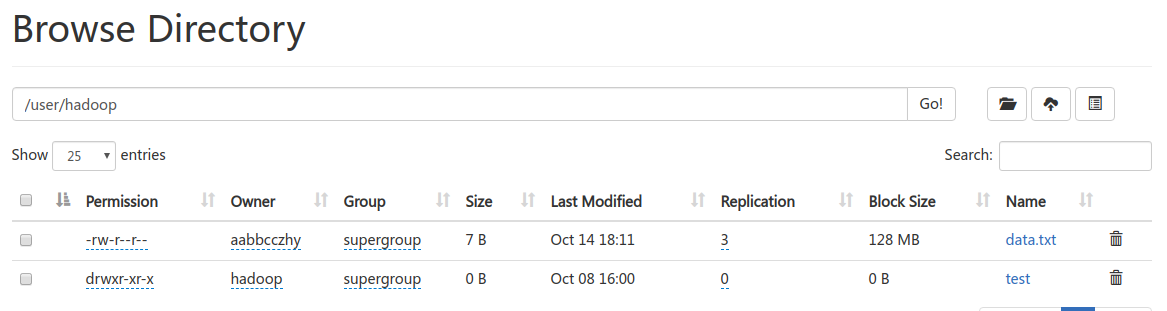
public HDFSUpload(){  
 try {  
 fs = *FileSystem*.*get*(MyConfiguration.*getInstance*().getConfiguration());  
 } catch (IOException *e*) {  
 *e*.printStackTrace();  
 }  
 }  
  
 public void uploadFile(String *localFilePath*, String *remoteFilePath*,boolean *overwrite*) throws IOException {  
 Path localPath = new Path(*localFilePath*);  
 Path remotePath = new Path(*remoteFilePath*);  
 if(*overwrite*){  
 fs.copyFromLocalFile(false,true,localPath,remotePath);  
 }else{  
 FileInputStream inputStream = new FileInputStream(*localFilePath*);  
 FSDataOutputStream outputStream = fs.append(remotePath);  
 byte[] buffer = new byte[1024];  
 int content = -1;  
 while((content = inputStream.read(buffer)) > 0){  
 outputStream.write(buffer,0,content);  
 }  
 outputStream.close();  
 inputStream.close();  
 }  
 }  
  
 public void uploadFile(String *localFilePath*, String *remoteFilePath*) throws IOException {  
 uploadFile(*localFilePath*,*remoteFilePath*,true);  
 }  
  
 */\*\*  
 \* 用于判断文件是否存在  
 \** ***@param*** *filePath  
 \** ***@return*** *\*/* private boolean exist(String *filePath*){  
 try {  
 return fs.exists(new Path(*filePath*));  
 } catch (IOException *e*) {  
 *e*.printStackTrace();  
 return false;  
 }  
 }  
  
 public static void main(String[] *args*) {  
 if(*args*.length < 3){  
 System.*out*.println("Usage: HDFSUpload localPath remotePath [-a,-o]");  
 System.*out*.println("-a: 若文件存在，则追加内容到原有文件末尾");  
 System.*out*.println("-o: 若文件存在，则覆盖原有文件");  
 return;  
 }  
 HDFSUpload hdfsUpload = new HDFSUpload();  
 if (!hdfsUpload.exist(*args*[1]) || "-o".equals(*args*[2])) {  
 try {  
 hdfsUpload.uploadFile(*args*[0],*args*[1]);  
 System.*out*.println("上传文件" + *args*[0] + "成功");  
 } catch (IOException *e*) {  
 *e*.printStackTrace();  
 }  
 }else if("-a".equals(*args*[2])){  
 try {  
 hdfsUpload.uploadFile(*args*[0],*args*[1],false);  
 System.*out*.println("上传文件" + *args*[0] + "成功");  
 } catch (IOException *e*) {  
 *e*.printStackTrace();  
 }  
 }  
 }  
}

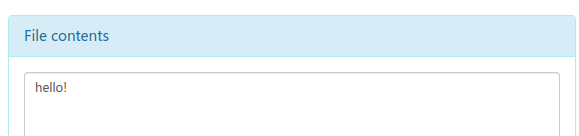


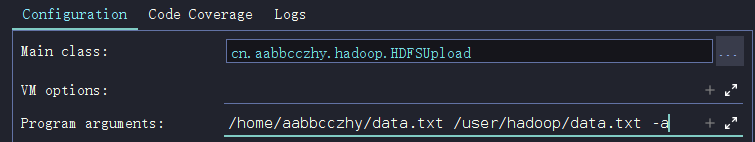




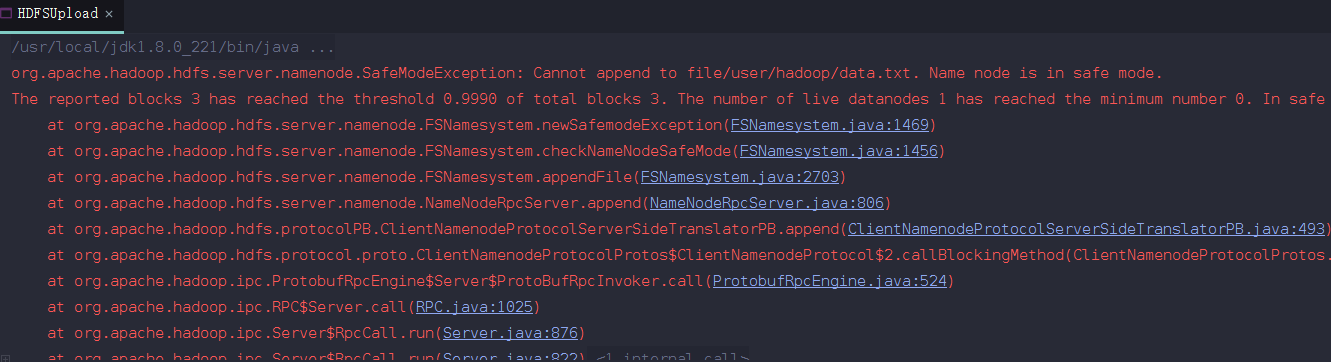
上传文件成功！







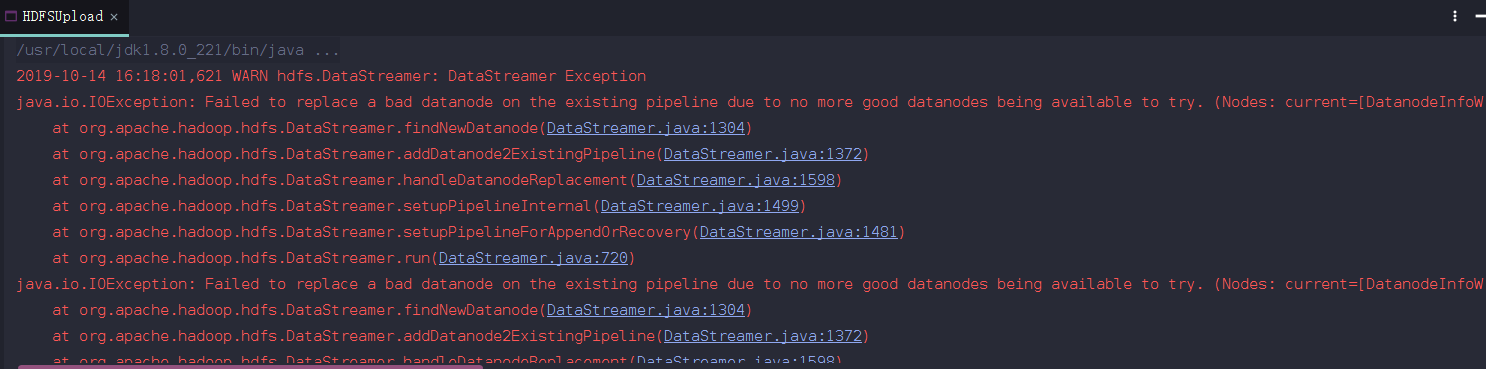
在使用append的时候出现了以下异常：



很明显，告诉我们，名称节点处于安全模式，不能append，因此我们需要关闭掉hadoop的安全模式，终端运行如下：

> hdfs -dfsadmin -safemode leave

提示Safe Mode is OFF,关闭安全模式成功，再次运行程序，然而并非一帆风顺的：



从意思来看，是说没有更好的数据节点能替换已经存在的数据节点，这我就不懂了，百度了一下，给出的解释是：

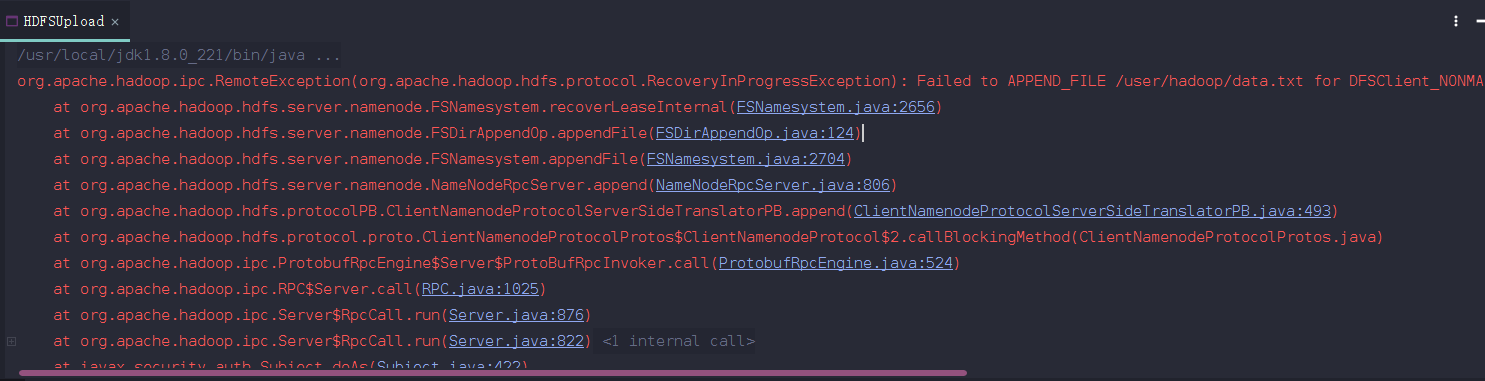
在进行pipeline写数据时，如果DN或者网络故障，客户端将尝试移除失败的DN，然后写到剩下的DN。这样的结果是pipeline中的DN减少了。该属性用于配置这种情况下是否添加新的DN到pipeline。这是一个站点范围的选项。当集群规模非常小时，例如3个或者更小，集群管理者可能想在默认配置文件中设置遇到故障时使用的策略为NEVER，或者禁止掉此属性。否则，客户端将会因为找不到新的DN来代替而遇到反常的高概率的写入失败。

解决方案：

在配置类中加入两条属性：

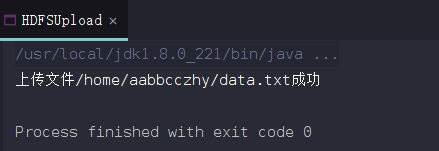
conf.set("dfs.client.block.write.replace-datanode-on-failure.policy", "NEVER");  
conf.set("dfs.client.block.write.replace-datanode-on-failure.enable", "true");

但是，还是报错了，这次错误变成了：

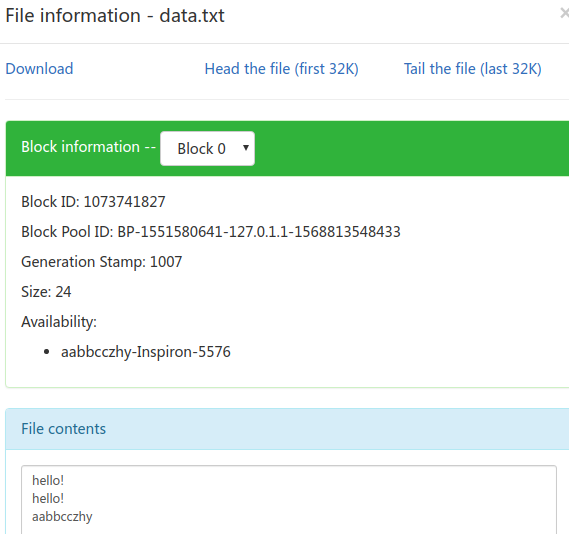


Append失败了，最后的解决方案也很简单，只要在配置类里添加如下属性即可：

conf.set("dfs.support.append","true");



总算成功了！通过浏览器看看文件的改变情况：



下面使用Shell命令来实现上传文本文件：

$ hadoop fs -put /home/aabbcczhy/data.txt /user/hadoop/test/

$ hadoop fs -appendToFile /home/aabbcczhy/data.txt /user/hadoop/test/data.txt

### 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；

if $(hadoop fs -test -e /home/hadoop/data.txt);

then

hadoop fs -copyToLocal /user/hadoop/data.txt ~/data.txt;

else

hadoop fs -copyToLocal /user/hadoop/data.txt /home/hadoop/data(1).txt;

fi

### 将HDFS中指定文件的内容输出到终端中；

$ hadoop fs -cat /user/hadoop/data.txt

hello!

hello!

### 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；

$ hadoop fs -ls /user/hadoop/data.txt

-rw-r--r-- 3 aabbcczhy supergroup 14 2019-10-14 18:13 /user/hadoop/data.txt

### 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；

$ hadoop fs -ls -R /user/hadoop

-rw-r--r-- 3 aabbcczhy supergroup 14 2019-10-14 18:13 /user/hadoop/data.txt

drwxr-xr-x - hadoop supergroup 0 2019-10-14 18:26 /user/hadoop/test

-rw-r--r-- 1 hadoop supergroup 4151 2019-09-18 21:48 /user/hadoop/test/.bashrc

-rw-r--r-- 1 hadoop supergroup 14 2019-10-14 18:26 /user/hadoop/test/data.txt

-rw-r--r-- 3 aabbcczhy supergroup 11 2019-10-08 16:00 /user/hadoop/test/hello.txt

### 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；

由于hadoop没有提供自动创建目录的功能，所以我们需要写一个shell程序来实现，实现内容如下：

if $(hadoop fs -test -d /usr/local/hadoop/test);

then

hadoop fs -touch /usr/local/hadoop/test/hello.txt;

else

hadoop fs -mkdir -p /usr/local/hadoop/test

hadoop fs -touch /usr/local/hadoop/test/hello.txt;

fi

### 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；

①创建目录

$ hadoop fs -mkdir -p /user/hadoop/lab2/test

②删除目录

$ hadoop dfs -rm -r /user/hadoop/test

### 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；

$ hadoop fs -appendToFile /home/aabbcczhy/data.txt /user/hadoop/test/data.txt

### 删除HDFS中指定的文件；

$ hadoop fs -rm /user/hadoop/test/hello.txt

Deleted /user/hadoop/test/hello.txt

### 在HDFS中，将文件从源路径移动到目的路径。

$ hadoop fs -mv /user/hadoop/test/data.txt /user/hadoop/lab2

$ hadoop fs -ls /user/hadoop/lab2

Found 2 items

-rw-r--r-- 1 hadoop supergroup 14 2019-10-14 18:26 /user/hadoop/lab2/data.txt

drwxr-xr-x - hadoop supergroup 0 2019-10-20 21:35 /user/hadoop/lab2/test

## 编程实现MyFSDataInputStream

我们的目标是要继承FsDataInputStream来实现读取单行文本文件的功能。先说一下我的思路，因为要实现readLine()自然而然就想到了BufferedReader这个类里面有一个readLine()方法。于是我先去看了一下BufferedReader的源码实现：

String readLine(boolean *ignoreLF*) throws IOException {  
 StringBuffer s = null;  
 int startChar;  
 synchronized (lock) {  
 ensureOpen();  
 boolean omitLF = *ignoreLF* || skipLF;  
 bufferLoop:  
 for (;;) {  
 if (nextChar >= nChars)  
 fill();  
 if (nextChar >= nChars) { */\* EOF \*/* if (s != null && s.length() > 0)  
 return s.toString();  
 else  
 return null;  
 }  
 boolean eol = false;  
 char c = 0;  
 int i;  
 */\* Skip a leftover '\n', if necessary \*/* if (omitLF && (cb[nextChar] == '\n'))  
 nextChar++;  
 skipLF = false;  
 omitLF = false;  
 charLoop:  
 for (i = nextChar; i < nChars; i++) {  
 c = cb[i];  
 if ((c == '\n') || (c == '\r')) {  
 eol = true;  
 break charLoop;  
 }  
 }  
 startChar = nextChar;  
 nextChar = i;  
 if (eol) {  
 String str;  
 if (s == null) {  
 str = new String(cb, startChar, i - startChar);  
 } else {  
 s.append(cb, startChar, i - startChar);  
 str = s.toString();  
 }  
 nextChar++;  
 if (c == '\r') {  
 skipLF = true;  
 }  
 return str;  
 }  
 if (s == null)  
 s = new StringBuffer(*defaultExpectedLineLength*);  
 s.append(cb, startChar, i - startChar);  
 }  
 }  
}

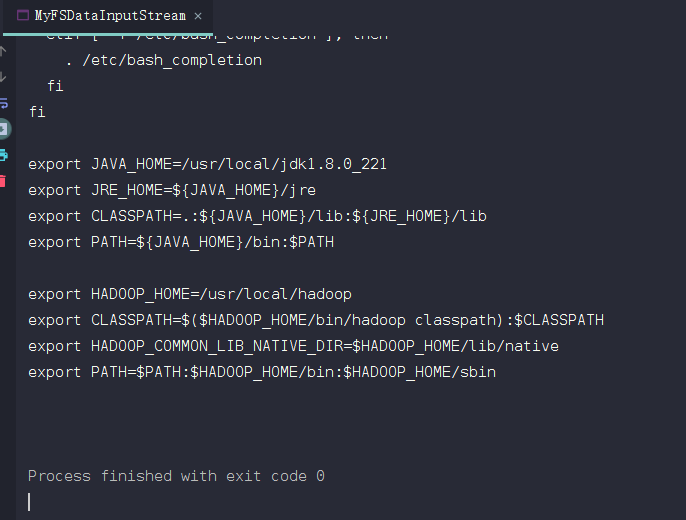
大概看了一下，他是用一个成员变量来记录当前读取字符的位置，然后通过两个Loop来读取一行文本最后通过StringBuffer的append加上该行文本，其中还加了一个锁来保证线程安全。

不过按照源码来实现仿佛是不大可能了，于是就想，既然BufferedReader有这个readLine()方法，我们就直接拿来用就好了：

因为FsDataInputStream需要传入一个InputStream的参数，因此我们的MyFsDataInputStream也自然要实现父类的构造方法，既然取得了这个InputStream，我们就可以利用它来实例化一个BufferedReader，然后实现一个readline()方法即可：

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.\*;  
  
*/\*\*  
 \** ***@author*** *aabbcczhy  
 \** ***@date*** *2019-10-12 15:21:51  
 \*/*public class MyFSDataInputStream extends FSDataInputStream {  
 private BufferedReader bufferedReader;  
 public MyFSDataInputStream(*InputStream in*) {  
 *super*(*in*);  
 bufferedReader = new BufferedReader(new InputStreamReader(*this*));  
 }  
 public String readline() throws IOException {  
 return bufferedReader.readLine();  
 }  
 @Override  
 public void close() throws IOException {  
 bufferedReader.close();  
 *super*.close();  
 }  
 public static void main(String[] *args*) throws IOException {  
 Configuration configuration = MyConfiguration.*getInstance*().getConfiguration();  
 *FileSystem* fs = *FileSystem*.*get*(configuration);  
 MyFSDataInputStream fsDataInputStream = new MyFSDataInputStream(fs.open(new Path("/user/hadoop/test/.bashrc")));  
 String content = null;  
 while((content=fsDataInputStream.readline())!=null){  
 System.*out*.println(content);  
 }  
 }  
}

写了个Main方法测试了一下，输入.bashrc成功：



## 使用URL与FsURLStreamHandlerFactory编程

使用URL和FsURLStreamHandlerFactory比较简单，实现思路大概就是先初始化URL的“流处理工厂”，然后通过调用API很轻松就能连接到Hadoop集群进而读取到文本内容。

代码实现如下：

import org.apache.hadoop.fs.FsUrlStreamHandlerFactory;  
import org.apache.hadoop.io.IOUtils;  
  
import java.io.*InputStream*;  
import java.net.URL;  
  
*/\*\*  
 \** ***@author*** *aabbcczhy  
 \** ***@date*** *2019-10-12 16:05:59  
 \*/*public class FsByURL {  
  
 static{  
 URL.*setURLStreamHandlerFactory*(new FsUrlStreamHandlerFactory());  
 }  
  
 public static void read(String *filePath*){  
 try {  
 *InputStream* inputStream = new URL("hdfs","localhost",9000,*filePath*).openStream();  
 IOUtils.*copyBytes*(inputStream,System.*out*,4096,false);  
 IOUtils.*closeStream*(inputStream);  
 } catch (Exception *e*) {  
 *e*.printStackTrace();  
 }  
 }  
  
 public static void main(String[] *args*) {  
 FsByURL.*read*("/user/hadoop/test/.bashrc");  
 }  
}

# 实验总结

通过本次实验，我熟悉了HDFS一些常用Shell命令的用法，也学会使用Java API来操作HDFS。

本次实验遇到的问题在于Configuration类，由于配置不正确的问题，Java程序会经常性的报错；为了解决这个问题，并且让代码能复用，于是将Configuration以单例模式来实现，取得了不错的效果。