实验报告

课程名称： 《Hadoop大数据技术与应用》

实验学期： 2019年至2020年第1学期

专 业： 数学与应用数学

班 级： 数学17-1

姓 名： 杨文俊

学 号： 17124080137

指导教师： 黄志毅

**3.1通过 Shell 命令访问 HDFS**

1.1实验目的

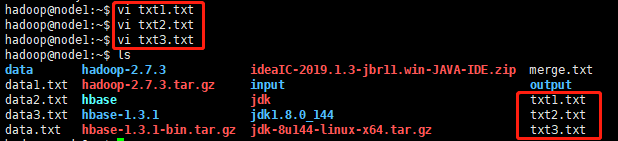
1.2要求

1.3实验原理

1.4实验步骤

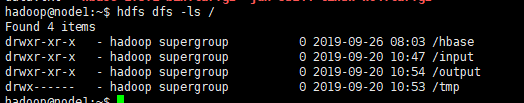
A：掌握常用的HDFS操作命令（hdfs dfs）

1.4.1 在本地创建3个文本文件：txt1.txt，txt2.txt，txt3.txt，并在文本输入有一些内容

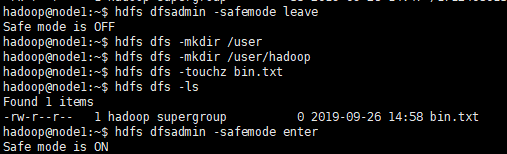


1.4.2 列出根目录 hdfs dfs -ls hdfs dfs -ls /





**注释：当我第一次输入hdfs dfs -ls 是会报错的，于是我上网找了资料找到以下解决方法：**



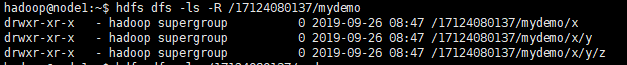
1.4.3 创建目录（-p表示会创建父目录）





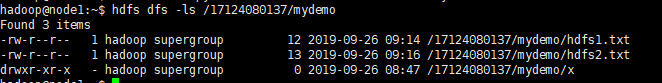
1.4.3 列出文件夹17124080137/mydemo下的文件夹（-R表示列出所有子文件夹）



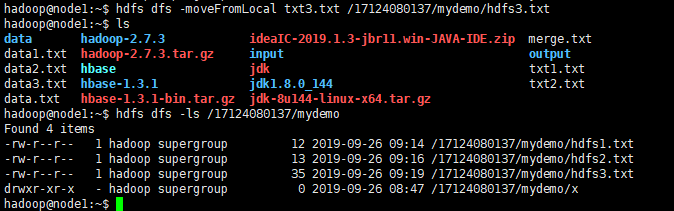


1.4.4 上传文件将本地目录的txt1.txt，txt2.txt，文件上传到HDFS上，并重命名hdfs1.txt,hdfs2.txt



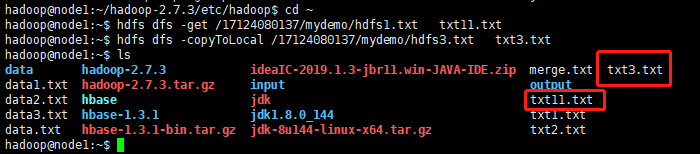


1.4.5 将本地文件移到HDFS（同时本地的文件也被删除掉）



1.4.6下载文件

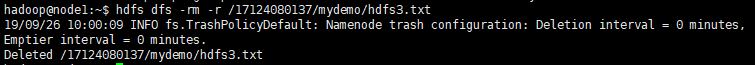
将HDFS 中的文件hdfs1.txt，hdfs3.txt复制到本地系统，并分别命名txt11.txt，txt3.txt



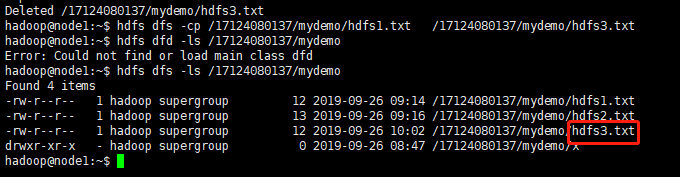
1.4.7查看文件 查看HDFS中/17124080137/mydemo/hdfs2.txt的内容



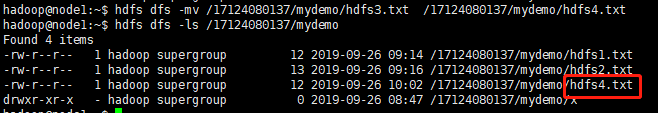
1.4.8 删除文件 删除HDFS中名为hdfs3的文件（参数-r为递归删除）



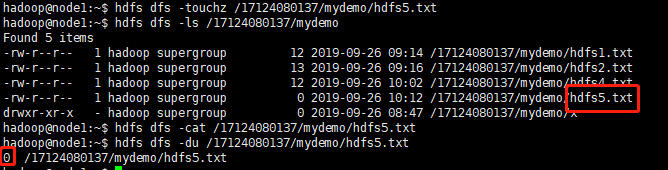
1.4.9 文件或文件夹的复制



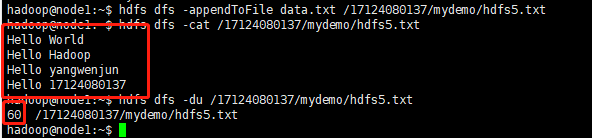
1.4.10 文件或文件夹的移动（或者理解为更名！）



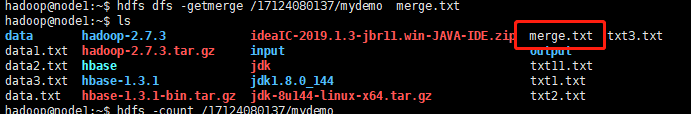
1.4.11 使用touchz命令创建一个空文件file



1.4.12 追加数据到文件末尾的指令



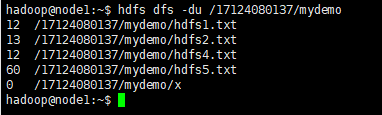
1.4.13 文件的合并再下载（下载到本地）



1.4.14 count统计（显示：目录个数、文件个数、文件总计大小、输入路径）



1.4.15 查看文件的大小



1.4.16 设置拓展属性，获取拓展属性

设置拓展属性：

-n：指定属性的名称（设置属性时设置此属性），属性的名称必须以“user/trusted/security/system/raw”中某一个为前缀

-v：指定属性的值

-x：指定属性的名称（删除属性时使用此参数）

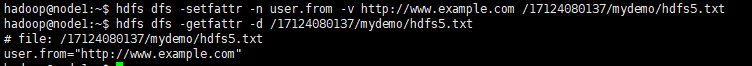
hdfs dfs -setfattr -x name <path> 可以删除属性

获取拓展属性：

-n：同上

-d：指dump，即显示所有属性

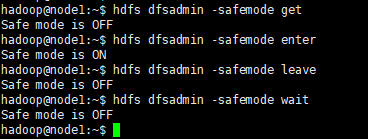
-e：指encoding，包含text、hex、base64等。



B：掌握常用的HDFS管理命令（hdfs dfsadmin）

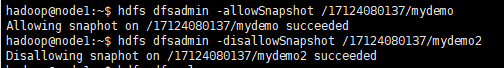
练习常用的HDFS管理命令，执行并查看结果（可以输入hdfs dfsadmin -help，查询命令的做法）

1.4.17 （1）安全模式的相关练习



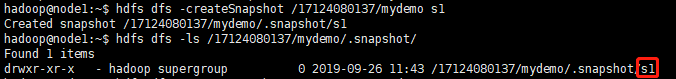
1.4.17（2）快照相关练习

开启或禁用快照的功能的命令如下：

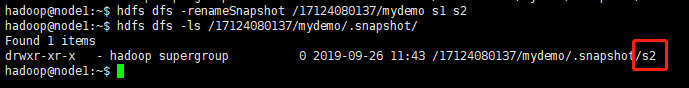


创建快照、删除快照、重命名快照的命令如下

创建快照并查看是否创建成功



重命名快照并验证



删除快照并验证



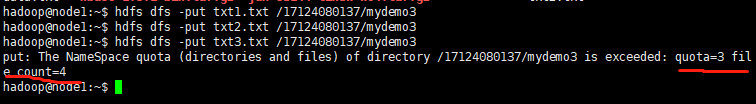
1.4.17（3）配额相关

配额命令如下：

设置mydemo3下的文件目录数和目录数之和不超过3



尝试向mydemo3传送三个文件，发现报错



究其原因，因为目录“.”已经占用一个，所以此例子做多只能传送2个文件。

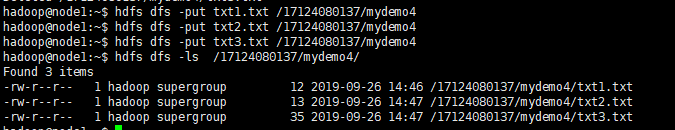
1.4.17（4）空间配额相关

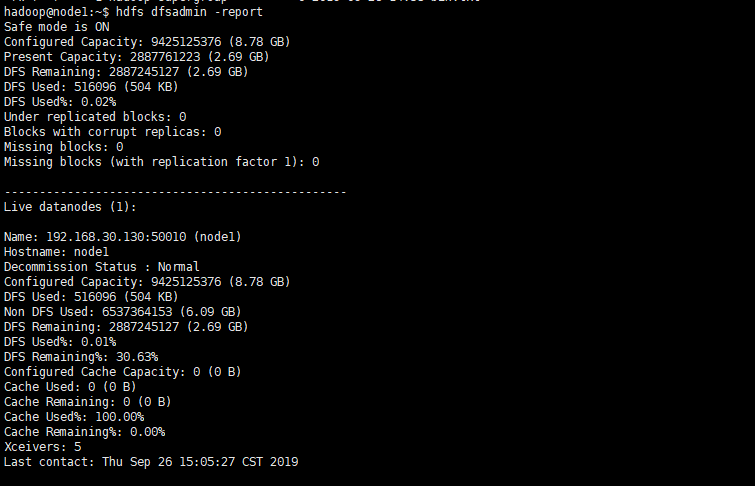
eg.



setSpaceQuota命令用于设置HDFS 中某个目录可用存储空间大小，单位为byte。此例为创建256MB的空间！

试着向mydemo4上传3个文件。



1.4.17（5）报告文件系统的基本信息和统计信息

1.4.17（6）查看拓扑



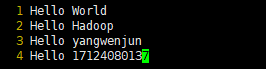
**1.4.18 命令的综合运用**

在本地创建一个文件file1，查看file1是否创建成功，往file1内写入一些内容。在HDFS上创建一个文件夹folder1，把file1上传到folder1中。查看是否成功，成功后查看file1的内容。把file1下载到本地，查看file1的内容。把folder1删除，并查看是否删除成功。最后把本地的file1删除。

步骤如下：

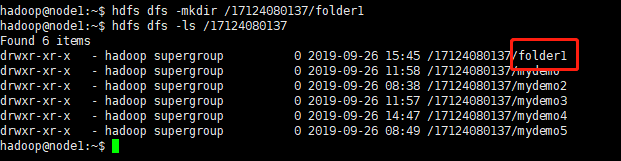
1. 创建文件并查看是否创建成功，并编辑内容



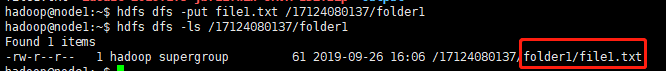




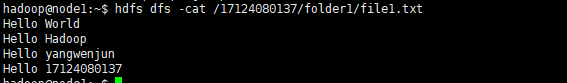
1. 在hdfs创建一个folder1文件夹，并查看是否创建成功。



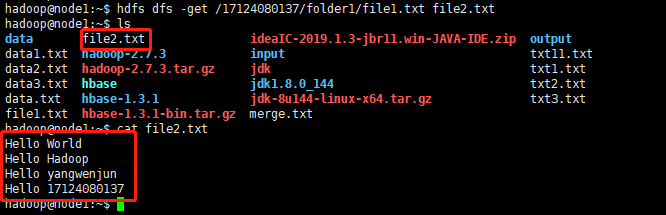
1. 把file1文件上传到folder1中，并查看是否上传成功。



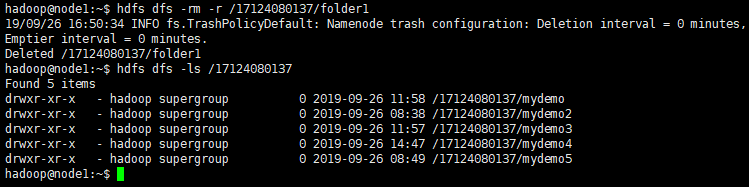
1. 查看文件信息



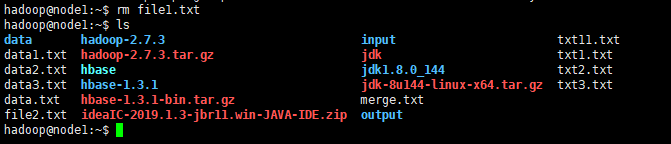
1. 把hdfs下的file1.txt下载到本地并命名为file2.txt ，并查看内容



1. 把hdfs的folder1删除，并查看是否成功



1. 把本地的file1.txt删除



**实验3.2：熟悉基于 IDEA+Maven 的 Java 开发环境**

**【实验名称】 熟悉基于 IDEA+Maven 的 Java 开发环境**

**【实验目的】**

（1）“工欲善其事，必先利其器”。IDEA+Maven（或 Eclipse+ Maven）是大多

数从事 Java

开发的企业和工程师优先选择的开发工具。学生通过本次实验可以懂得如何配置和使

用

IDEA+Maven 开发环境。

（2）了解如何使用 IDEA 创建 Maven 工程、运行 Maven 工程。

（3）了解 Maven 的一些基本命令（如打包命令），为后续的程序开发做准备。

**【实验原理】**

使用IDEA+Maven环境，开发一个Hello World程序，并运行

**【实验环境】**

操作系统：64 位 Windows。

JDK：JDK 1.8 安装包。

Maven：Maven 安装包。

IDEA：IDEA 安装包。

注意：

（1）此实验以 Windows 环境为例。如果采用 Linux 环境，使用的安装包和配置环 境变量的 方式不一样，但配置和使用 IDEA 的方式是相似的。

（2）如果采用 Linux 环境，且多用户同时使用，为了避免本地仓库的权限问题，可将 Maven 安装目录下的 conf/settings.xml 复制到~/.m2 下，并 改“~/.m2/settings.xml”中的 localRepository为当前用户有权限的目录。1 <localRepository>local\_repo</localRepository>

（3）环境必须连接互联网（可以通过连接百度网页查看是否能正常打开来确认）。 如果不能连接互联网，此实验将无法进行。

**【实验步骤】**

参考教材第二章 HDFS ——《实验2 熟悉基于 IDEA+Maven 的 Java 开发环境》

需要注意一下几点：

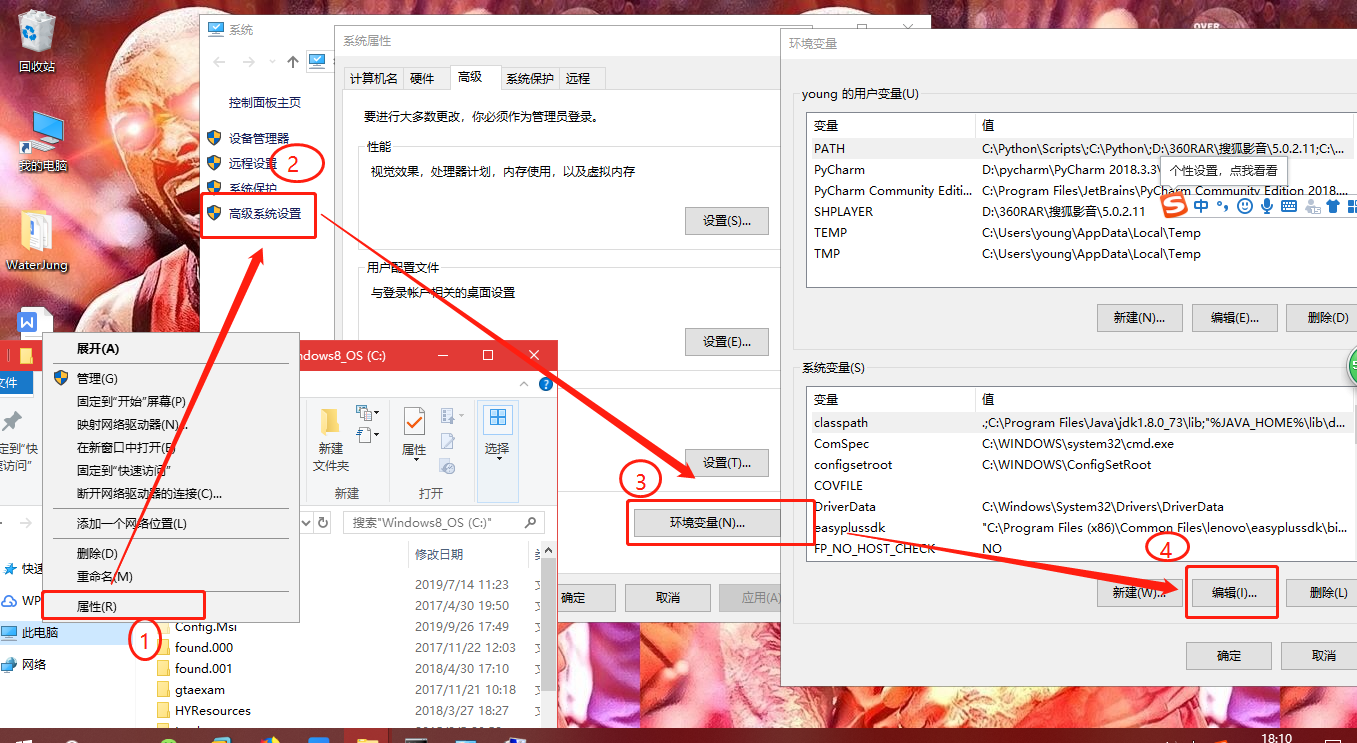
1. 安装相关软件工具

jdk-8u221-windows-x64.exe

ideaIC-2019.1.3-jbr11.win-JAVA-IDE.zip

apache-maven-3.6.2-bin.zip

1. 环境变量编辑的入口



Win+R cmd 进入终端输入‘java -version’测试结果如下：



2. Maven 的镜像配置

说明：这里镜像配置是为了让 Maven 下载 jar 包可以使用阿里云的源，避免直接从国外服务器下载，速度太慢

1 <mirror>

2 <id>nexus‐aliyun</id>

3 <mirrorOf>\*</mirrorOf>

4 <name>Nexus aliyun</name>

5 <url>http://maven.aliyun.com/nexus/content/groups/public</url>

6 </mirror>

1. POM.xml 的 mainClass 配置

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven‐shade‐plugin</artifactId>

<version>2.4.3</version>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>shade</goal>

</goals>

<configuration>

<transformers>

<transformer implementation="org.apache.maven.plugins.shade. resource.ManifestResourceTransformer"

<!‐‐ main() 所在的类，注意修改 ‐‐>

<mainClass>com.mystudy.App</mainClass>

</transformer>

</transformers>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

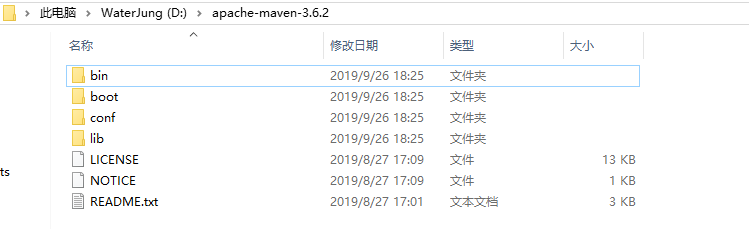
**【实验要求】**

1. 把 Maven 构建 jar 包成功的截图写入实验报告

2. 把 运行 jar 包的结果截图写入实验报告

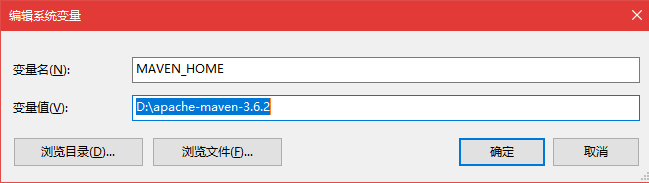
**步骤：**

1. 解压Maven安装包到某一个目录，如：D:\apache-maven-3.6.2



1. **配置环境变量**

1）配置MAVEN\_HOME,值为：D:\apache-maven-3.6.2



2）Path添加：%MAVEN\_HOME%\bin



3）输入mvn -v 测试结果如下：

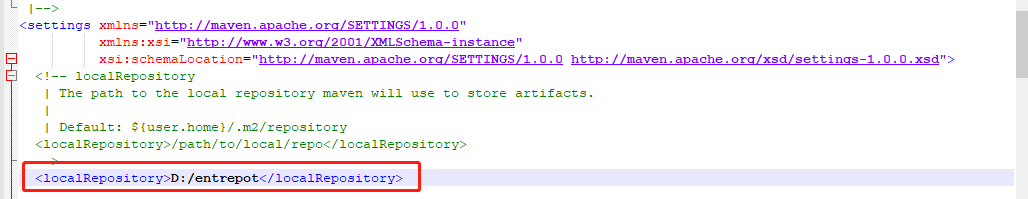


1. 配置maven。

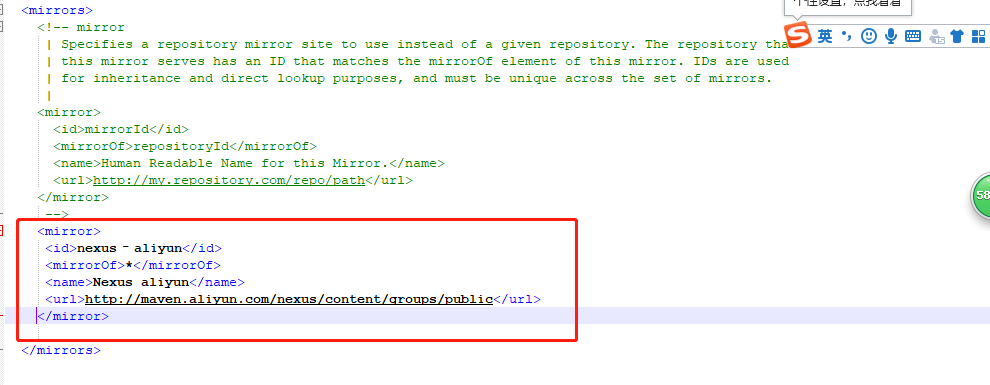
打开maven安装目录下conf目录，D:\apache-maven-3.6.2\conf，找到setting.xml,修改里面的内容。

①配置localRepository，明确本地仓库的位置。

搜索localRepository，找到相关的配置如下图。增加”<localRepository>D:/entrepot</localRepository>”.



②搜索”mirrors”,找到内容如下，增加框内内容。

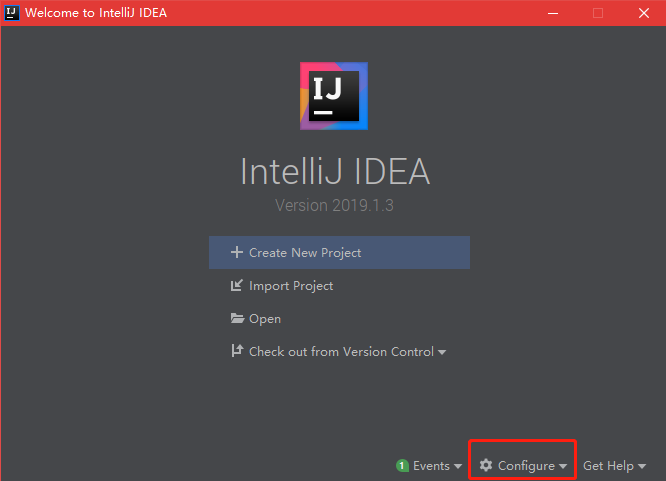


1. **安装IDEA**

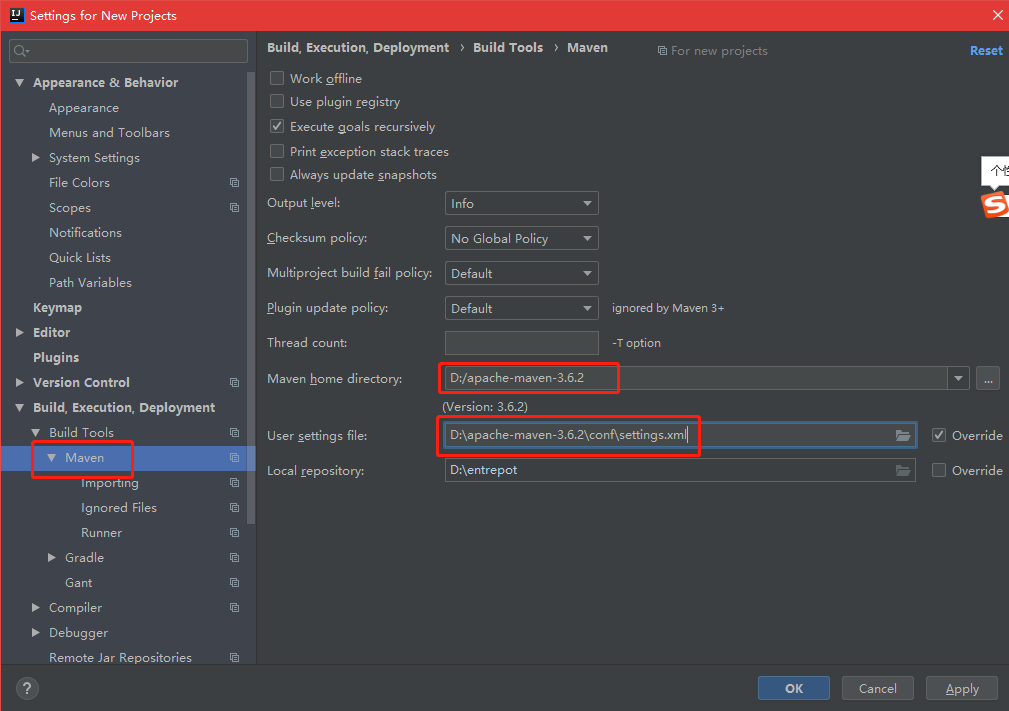
1.安装

2.配置Maven

首次启动IDEA,需要同意许可协议，选择IDEA等风格。然后出现下面的界面，单击Configure→Setting。

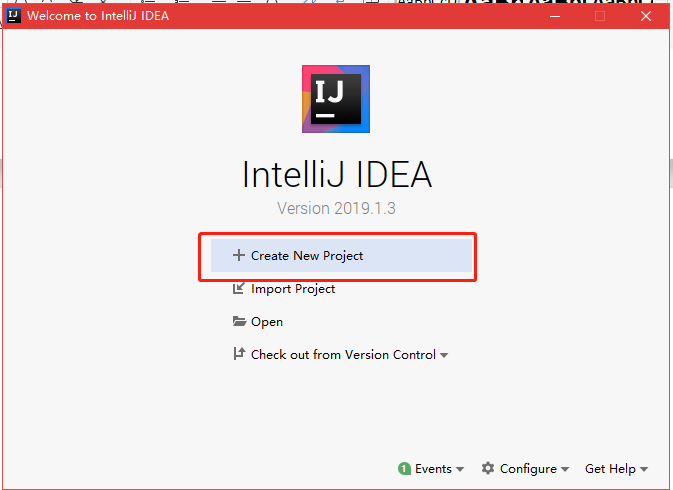


按下图的步骤，设置Maven和setting.xml的位置。

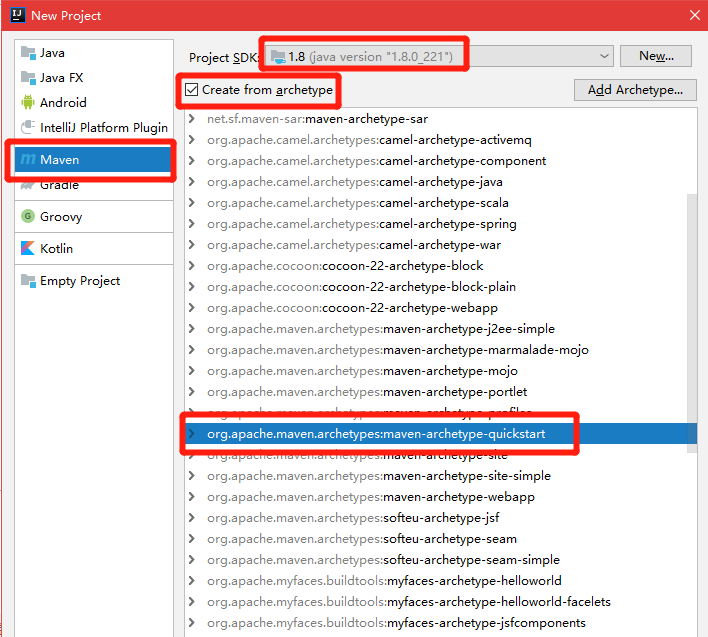


**5.创建工程**

1）在IDEA界面中，单击【Create New Project】，界面如下



2）



1. 设置项目的“坐标”如下：



1. 项目刚开始会自动联网下载一些文件，很快就可以看到项目的空间结构如下：

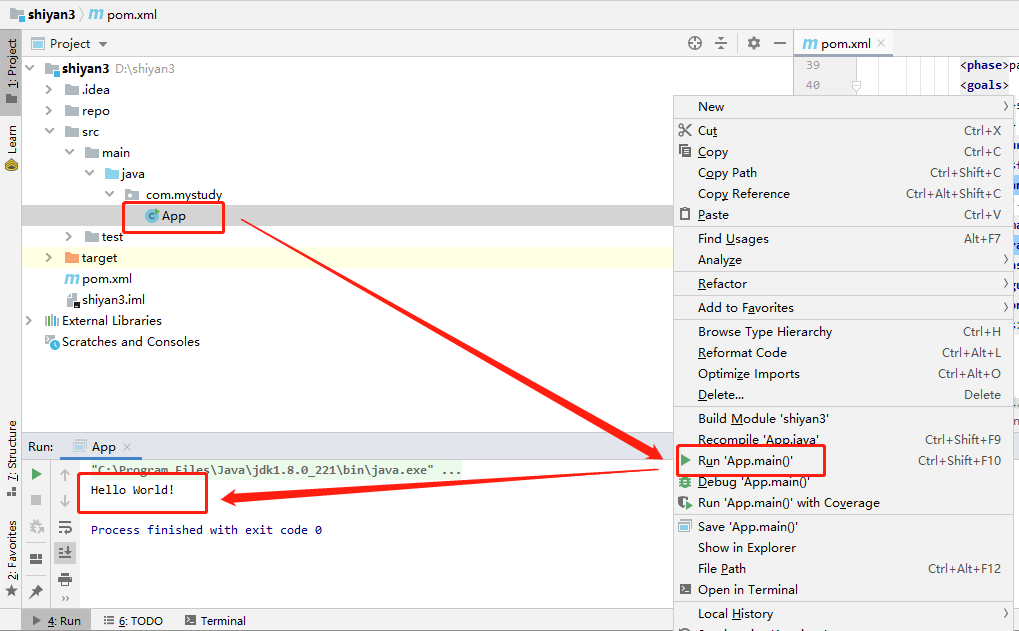


1. 修改pom.xml,指定工程的主类。

选择界面左边的pom.xml,并编辑，如下图所示。



1. 在IDEA中运行工程

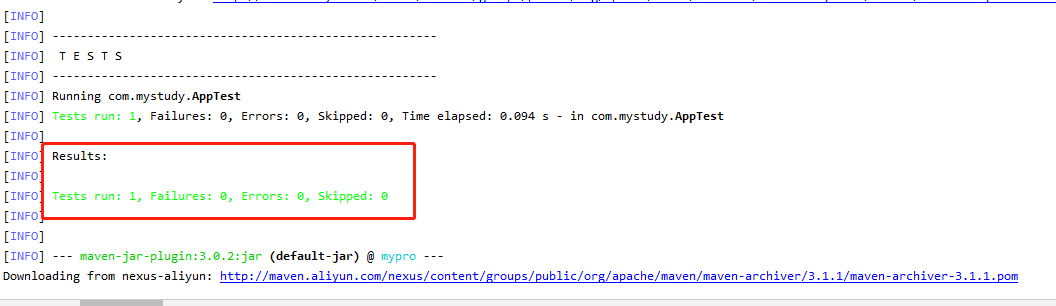


1. 单击Terminal,此时默认已在项目的目录。输入下面的命令进行打包。

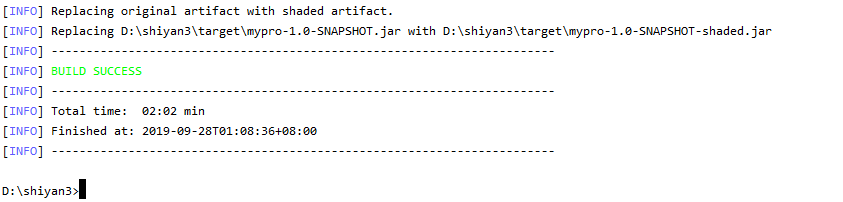
mvn clean package

可以看到如下结果：

**测试成功↓**



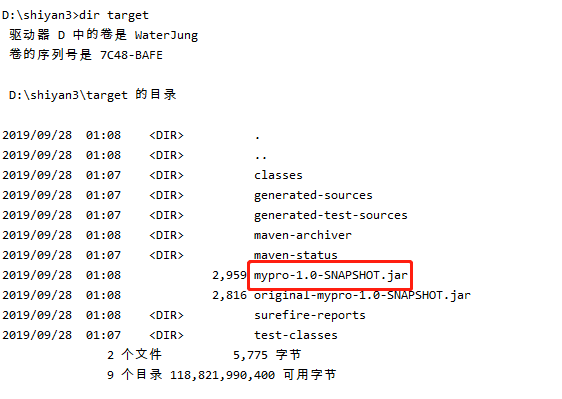
**构建成功↓**



1. 在命令行中运行

查看生成的jar包，命令如下：

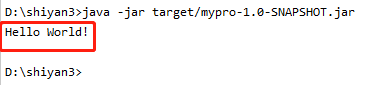
dir target



运行jar包：

java –jar target /mypro-1.0-SNAPSHOT.jar

得到预期的结果如下图：



**实验3.3：通过 API 访问 HDFS（必做）**

**【实验名称】 通过 API 访问 HDFS**

**【实验目的】**

（1）理解 HDFS 在 Hadoop 体系结构中的作用。

（2）熟悉通过 Java API 访问 HDFS。

**【实验原理】**

文件拷贝的原理和生活中的下面示例很像：

桶A可以获取输入流，桶B获取输出流，勺子作为缓冲器。勺子从A取水，倒到桶B，循环进行，就能实现将桶A的水“移动”到桶B。不同的是文件拷贝是原文件还在而已。实验采用JDK原生的接口InputStream、OutputStream、FileInputStream、 FileOutputStream实现。不能采用第三方封装的接口。源文件、目标文件可以通过参 数传入。后面的HDFS的上传、下载从根本上说，与文件拷贝实现基本一样的。所以，理解这个很有必要。

**【实验环境】**

操作系统：Linux。

Hadoop 版本：2.7.3 或以上版本。

JDK 版本：1.8 或以上版本。

Java IDE：IDEA/Eclipse。

**【实验步骤】**

1. **参考实验3.2，使用 IDEA 重新创建一个Maven项目。 配置pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>hadoop.ch03.v17124080137</groupId>

<artifactId>pro1</artifactId>

<version>1.0-SNAPSHOT</version>

<name>pro1</name>

<!-- FIXME change it to the project's website -->

<url>http://www.example.com</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>1.7</maven.compiler.source>

<maven.compiler.target>1.7</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-common</artifactId>

<version>2.7.3</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-hdfs</artifactId>

<version>2.7.3</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-mapreduce-client-core</artifactId>

<version>2.7.3</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-mapreduce-client-jobclient</artifactId>

<version>2.7.3</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-client</artifactId>

<version>2.7.3</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-shade-plugin</artifactId>

<version>3.1.0</version>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>shade</goal>

</goals>

<configuration>

<transformers>

<transformer implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">

<!-- main()所在的类，注意修改 -->

<mainClass>hadoop.ch03.v17124080137.App</mainClass>

</transformer>

</transformers>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

<pluginManagement>

<plugins>

<!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#clean\_Lifecycle -->

<plugin>

<artifactId>maven-clean-plugin</artifactId>

<version>3.1.0</version>

</plugin>

<!-- default lifecycle, jar packaging: see https://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_jar\_packaging -->

<plugin>

<artifactId>maven-resources-plugin</artifactId>

<version>3.0.2</version>

</plugin>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.0</version>

</plugin>

<plugin>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.22.1</version>

</plugin>

<plugin>

<artifactId>maven-jar-plugin</artifactId>

<version>3.0.2</version>

</plugin>

<plugin>

<artifactId>maven-install-plugin</artifactId>

<version>2.5.2</version>

</plugin>

<plugin>

<artifactId>maven-deploy-plugin</artifactId>

<version>2.8.2</version>

</plugin>

<!-- site lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#site\_Lifecycle -->

<plugin>

<artifactId>maven-site-plugin</artifactId>

<version>3.7.1</version>

</plugin>

<plugin>

<artifactId>maven-project-info-reports-plugin</artifactId>

<version>3.0.0</version>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

1. **编写 Java 代码，利用Java API 在 HDFS 上创建文件 /17124080137/test2.txt 。**

项目路径： src/main/java/

包名： hadoop.ch03.v17124080137

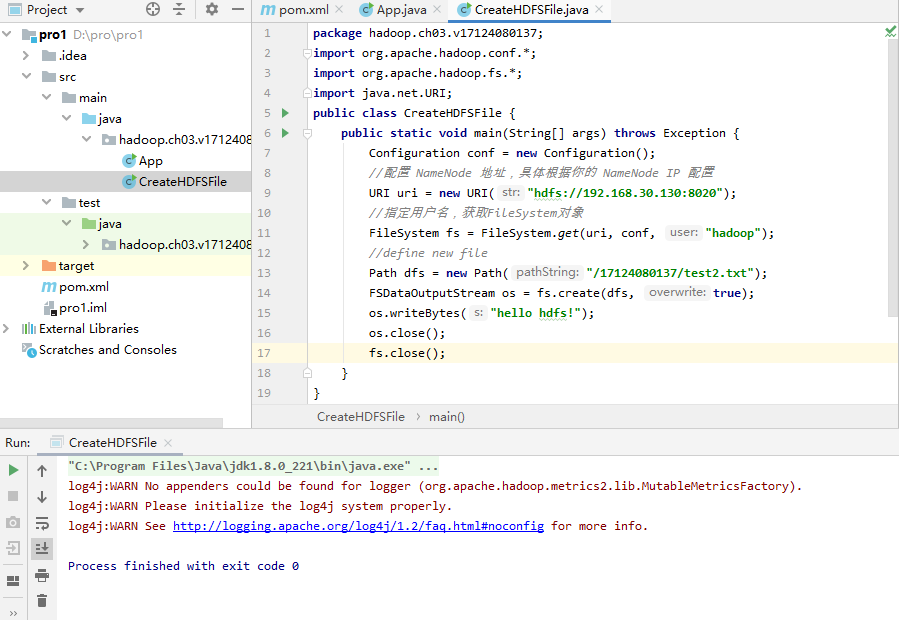
Java 文件名：CreateHDFSFile

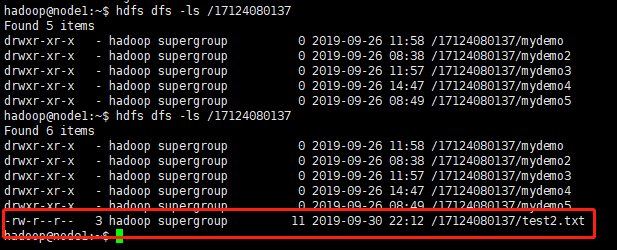
需要将实验过程写入实验报告。

以下为参考代码：

**package** hadoop.ch03.v17124080137;  
**import** org.apache.hadoop.conf.\*;  
**import** org.apache.hadoop.fs.\*;  
**import** java.net.URI;  
**public class** CreateHDFSFile {  
 **public static void** main(String[] args) **throws** Exception {  
 Configuration conf = **new** Configuration();  
 *//配置 NameNode 地址，具体根据你的 NameNode IP 配置* URI uri = **new** URI(**"hdfs://192.168.30.130:8020"**);  
 *//指定用户名，获取FileSystem对象* FileSystem fs = FileSystem.*get*(uri, conf, **"hadoop"**);  
 *//define new file* Path dfs = **new** Path(**"/17124080137/test2.txt"**);  
 FSDataOutputStream os = fs.create(dfs, **true**);  
 os.writeBytes(**"hello hdfs!"**);  
 os.close();  
 fs.close();  
 }  
}

实验截图如下：





1. **编写 Java 代码，利用Java API 在 HDFS 上删除文件 /17124080137/test2.txt 。**

项目路径： src/main/java/

包名： hadoop.ch03.v17124080137

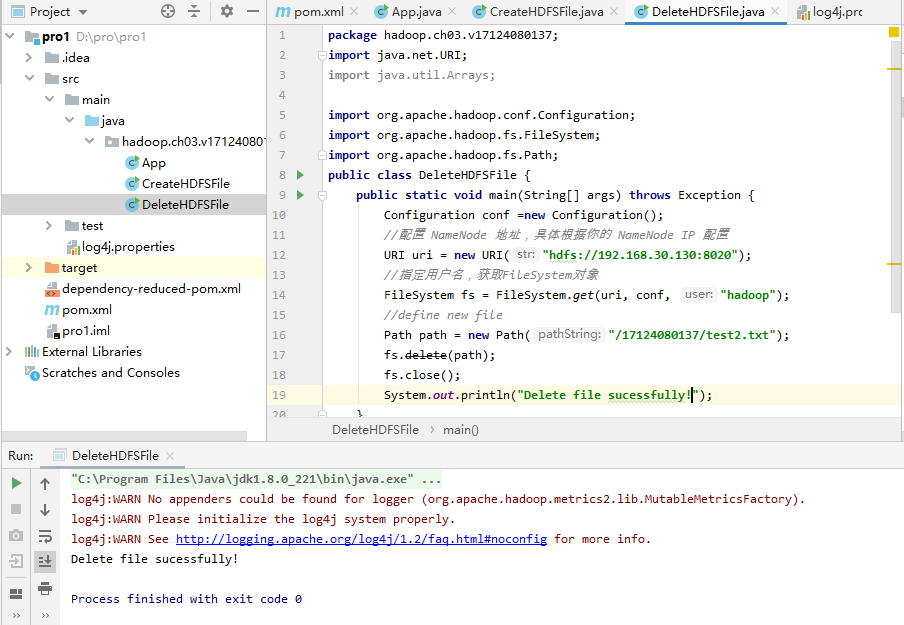
Java 文件名：DeleteHDFSFile

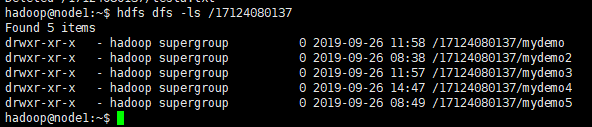
需要将实验过程写入实验报告。

以下为参考代码：

**package** hadoop.ch03.v17124080137;  
**import** java.net.URI;  
**import** java.util.Arrays;  
  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.FileSystem;  
**import** org.apache.hadoop.fs.Path;  
**public class** DeleteHDFSFile {  
 **public static void** main(String[] args) **throws** Exception {  
 Configuration conf =**new** Configuration();  
 *//配置 NameNode 地址，具体根据你的 NameNode IP 配置* URI uri = **new** URI(**"hdfs://192.168.30.130:8020"**);  
 *//指定用户名，获取FileSystem对象* FileSystem fs = FileSystem.*get*(uri, conf, **"hadoop"**);  
 *//define new file* Path path = **new** Path(**"/17124080137/test2.txt"**);  
 fs.delete(path);  
 fs.close();  
 System.***out***.println(**"Delete file sucessfully!"**);  
 }  
}

实验截图如下：





1. **编写 Java 代码，利用Java API 把本地文件test4.txt上传到HDFS路径/17124080137/test4.txt 。**

项目路径： src/main/java/

包名： hadoop.ch03.v17124080137

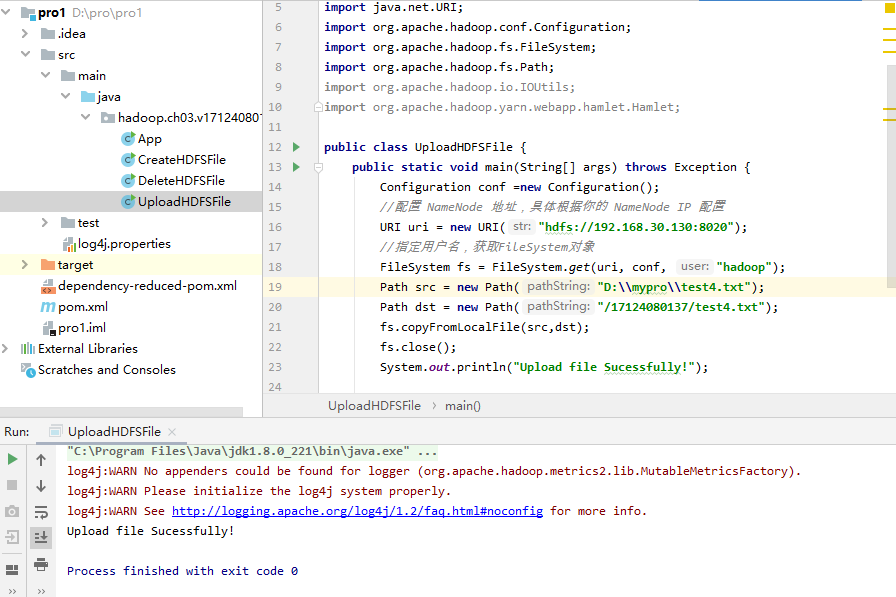
Java 文件名：UploadHDFSFile

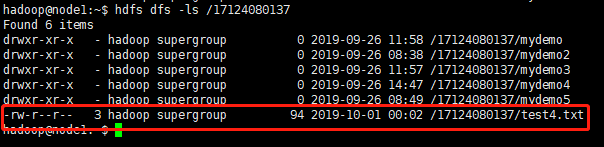
需要将实验过程写入实验报告。

以下为参考代码：

**package** hadoop.ch03.v17124080137;  
**import** java.io.FileInputStream;  
**import** java.io.InputStream;  
**import** java.io.OutputStream;  
**import** java.net.URI;  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.FileSystem;  
**import** org.apache.hadoop.fs.Path;  
**import** org.apache.hadoop.io.IOUtils;  
**import** org.apache.hadoop.yarn.webapp.hamlet.Hamlet;  
  
**public class** UploadHDFSFile {  
 **public static void** main(String[] args) **throws** Exception {  
 Configuration conf =**new** Configuration();  
 *//配置 NameNode 地址，具体根据你的 NameNode IP 配置* URI uri = **new** URI(**"hdfs://192.168.30.130:8020"**);  
 *//指定用户名，获取FileSystem对象* FileSystem fs = FileSystem.*get*(uri, conf, **"hadoop"**);  
 Path src = **new** Path(**"D:\\mypro\\test4.txt"**);  
 Path dst = **new** Path(**"/17124080137/test4.txt"**);  
 fs.copyFromLocalFile(src,dst);  
 fs.close();  
 System.***out***.println(**"Upload file Sucessfully!"**);  
  
 }  
}

实验截图如下：





1. **编写 Java 代码，利用Java API 在HDFS创建路径/17124080137/test5.txt ,内容为Hello World,并下载到本地。**

项目路径： src/main/java/

包名： hadoop.ch03.v17124080137

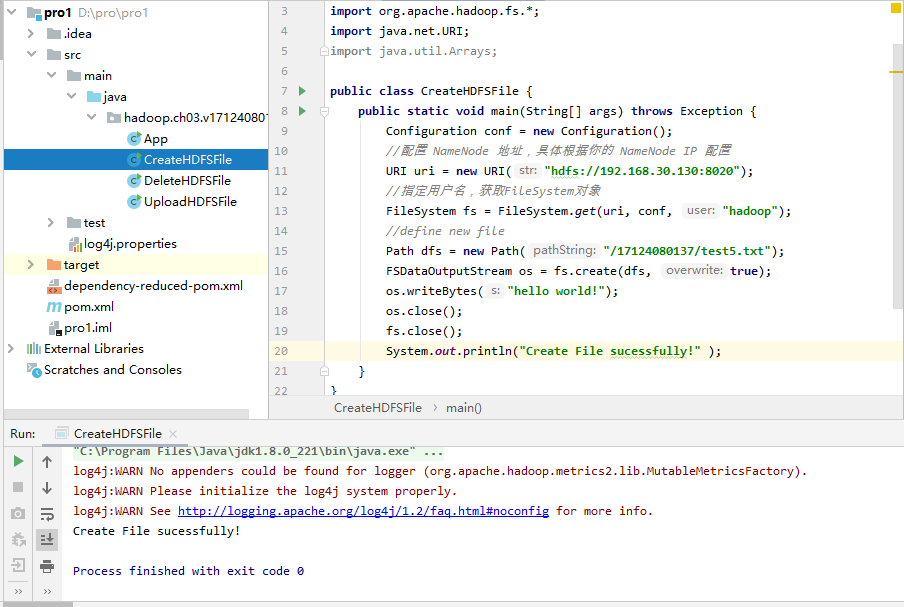
Java 文件名：DownloadHDFSFile

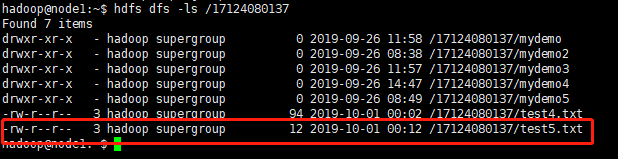
需要将实验过程写入实验报告。

以下为参考创建文件代码：

**package** hadoop.ch03.v17124080137;  
**import** org.apache.hadoop.conf.\*;  
**import** org.apache.hadoop.fs.\*;  
**import** java.net.URI;  
**import** java.util.Arrays;  
  
**public class** CreateHDFSFile {  
 **public static void** main(String[] args) **throws** Exception {  
 Configuration conf = **new** Configuration();  
 *//配置 NameNode 地址，具体根据你的 NameNode IP 配置* URI uri = **new** URI(**"hdfs://192.168.30.130:8020"**);  
 *//指定用户名，获取FileSystem对象* FileSystem fs = FileSystem.*get*(uri, conf, **"hadoop"**);  
 *//define new file* Path dfs = **new** Path(**"/17124080137/test5.txt"**);  
 FSDataOutputStream os = fs.create(dfs, **true**);  
 os.writeBytes(**"hello world!"**);  
 os.close();  
 fs.close();  
 System.***out***.println(**"Create File sucessfully!"** );  
 }  
}

**实验截图如下：**

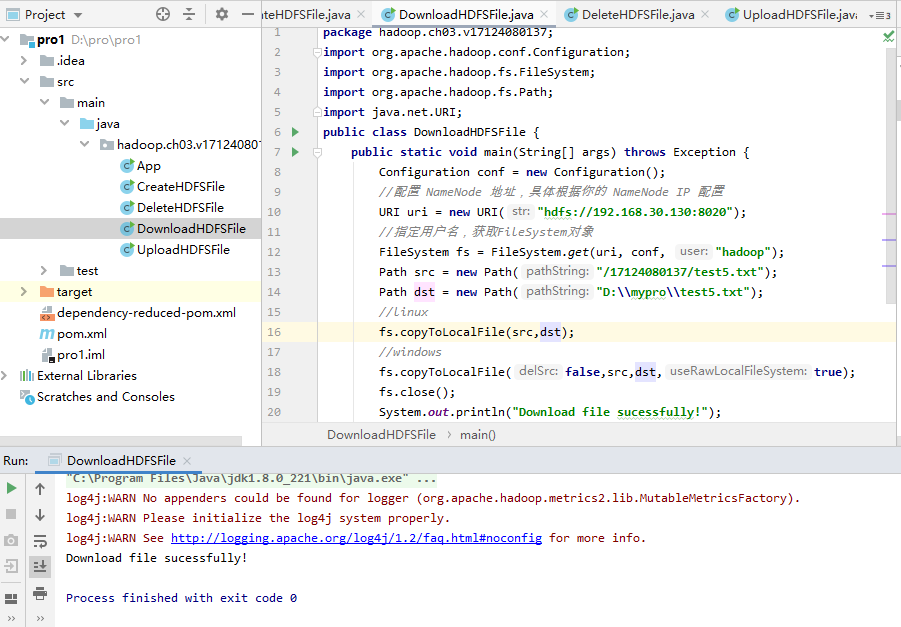


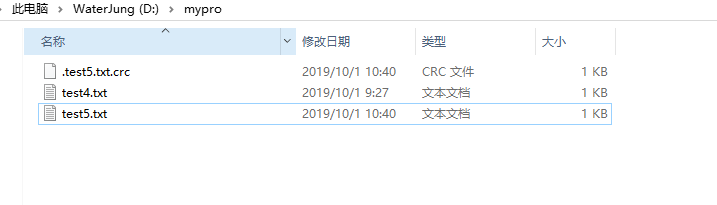


下载代码：

**package** hadoop.ch03.v17124080137;  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.FileSystem;  
**import** org.apache.hadoop.fs.Path;  
**import** java.net.URI;  
**public class** DownloadHDFSFile {  
 **public static void** main(String[] args) **throws** Exception {  
 Configuration conf = **new** Configuration();  
 *//配置 NameNode 地址，具体根据你的 NameNode IP 配置* URI uri = **new** URI(**"hdfs://192.168.30.130:8020"**);  
 *//指定用户名，获取FileSystem对象* FileSystem fs = FileSystem.*get*(uri, conf, **"hadoop"**);  
 Path src = **new** Path(**"/17124080137/test5.txt"**);  
 Path dst = **new** Path(**"D:\\mypro\\test5.txt"**);  
 *//linux* fs.copyToLocalFile(src,dst);  
 *//windows* fs.copyToLocalFile(**false**,src,dst,**true**);  
 fs.close();  
 System.***out***.println(**"Download file sucessfully!"**);  
  
 }  
}

实验截图如下：





1. **编写 Java 代码，利用Java API 在HDFS文件/17124080137/test5.txt写入一个属性user.id，值是个人的学号，并再读取此属性，利用System.out.println输出到java控制台。**

项目路径： src/main/java/

包名： hadoop.ch03.v17124080137

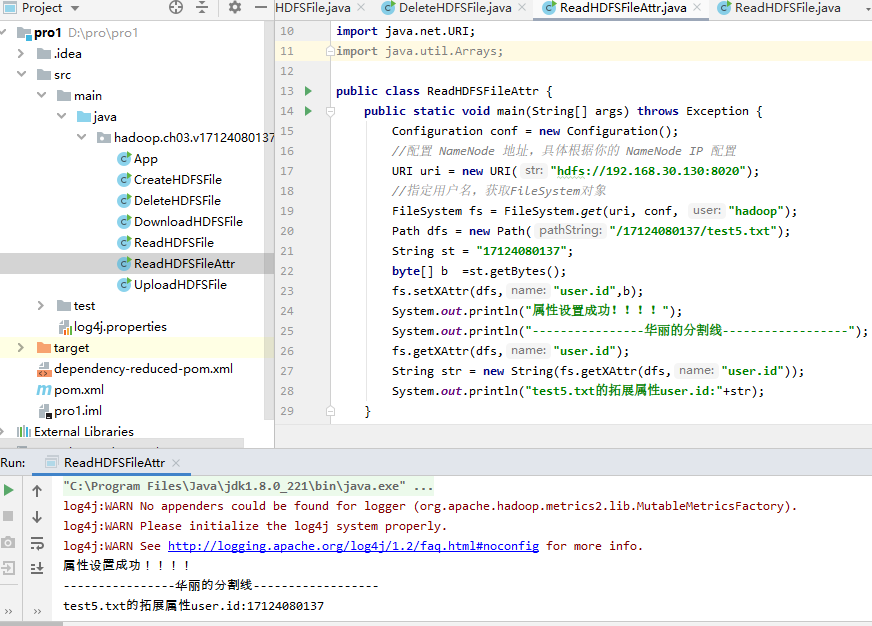
Java 文件名：ReadHDFSFileAttr

需要将实验过程写入实验报告。

代码如下：

**package** hadoop.ch03.v17124080137;  
  
**import** com.google.inject.internal.util.$Lists;  
**import** com.google.inject.internal.util.$ToStringBuilder;  
**import** org.apache.hadoop.conf.Configuration;  
**import** org.apache.hadoop.fs.\*;  
**import** org.apache.xerces.impl.dv.util.Base64;  
  
**import** java.lang.reflect.Type;  
**import** java.net.URI;  
**import** java.util.Arrays;  
  
**public class** ReadHDFSFileAttr {  
 **public static void** main(String[] args) **throws** Exception {  
 Configuration conf = **new** Configuration();  
 *//配置 NameNode 地址，具体根据你的 NameNode IP 配置* URI uri = **new** URI(**"hdfs://192.168.30.130:8020"**);  
 *//指定用户名，获取FileSystem对象* FileSystem fs = FileSystem.*get*(uri, conf, **"hadoop"**);  
 Path dfs = **new** Path(**"/17124080137/test5.txt"**);  
 String st = **"17124080137"**;  
 **byte**[] b =st.getBytes();  
 fs.setXAttr(dfs,**"user.id"**,b);  
 System.***out***.println(**"属性设置成功！！！！"**);  
 System.***out***.println(**"----------------华丽的分割线------------------"**);  
 fs.getXAttr(dfs,**"user.id"**);  
 String str = **new** String(fs.getXAttr(dfs,**"user.id"**));  
 System.***out***.println(**"test5.txt的拓展属性user.id:"**+str);  
 }  
}

实验截图如下：



**7. 编写 Java 代码，利用Java API 在HDFS 上读取文件 /17124080137/test5.txt 的内容，并利用System.out.println 输出到 Java 控制台.**

项目路径： src/main/java/

包名： hadoop.ch03.v[替换为你的完整学号]

Java 文件名：ReadHDFSFile

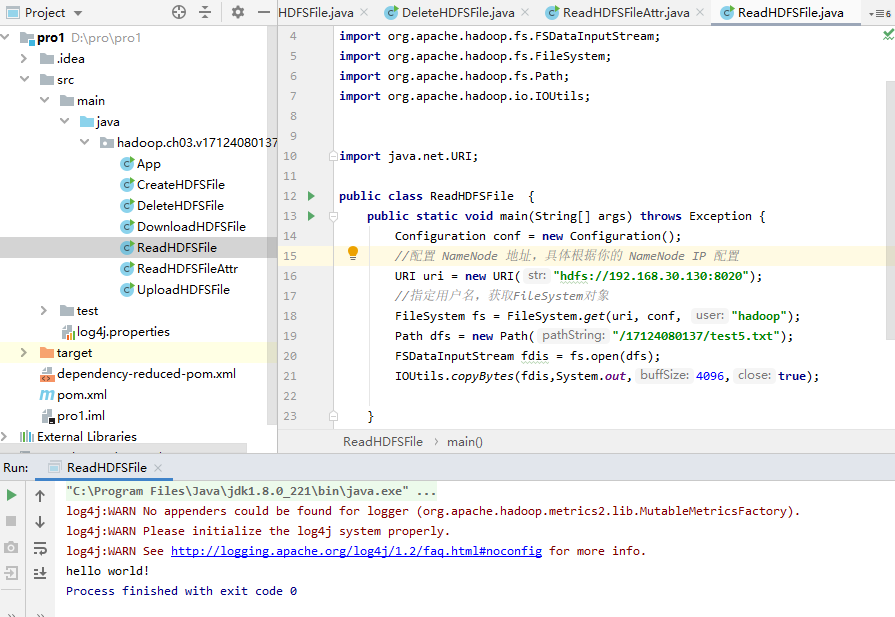
代码自行实现，可以参考实验3【实验原理】的相关API。

需要将实验过程写入实验报告。

代码如下：

**package** hadoop.ch03.v17124080137;  
  
**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** org.apache.hadoop.io.IOUtils;  
  
  
**import** java.net.URI;  
  
**public class** ReadHDFSFile {  
 **public static void** main(String[] args) **throws** Exception {  
 Configuration conf = **new** Configuration();  
 *//配置 NameNode 地址，具体根据你的 NameNode IP 配置* URI uri = **new** URI(**"hdfs://192.168.30.130:8020"**);  
 *//指定用户名，获取FileSystem对象* FileSystem fs = FileSystem.*get*(uri, conf, **"hadoop"**);  
 Path dfs = **new** Path(**"/17124080137/test5.txt"**);  
 FSDataInputStream fdis = fs.open(dfs);  
 IOUtils.*copyBytes*(fdis,System.***out***,4096,**true**);  
  
 }  
}

实验截图如下：



**上传到Github**

实验报告链接：<https://github.com/WaterJung/HadoopExp>

仓库截图：

