

# 《大数据技术及应用》 虚拟平台实验手册

石家庄铁道大学信息学院  
(章鱼互联网学院平台)

马新娜

**2021 年春季学期**

## 目录

实验一：Hadoop 伪分布模式安装.....	3
实验二：Hadoop 开发插件安装.....	14
实验三：Hadoop Shell 基本操作.....	22
实验四：HDFS JAVA API.....	28
实验五：HBase API 操作.....	44
实验六：MAPREDUCE 实例——WordCount.....	61
实验七：MAPREDUCE 求平均值.....	76
实验八：数据统计分析.....	87
附录：实验报告模板.....	91

# 实验一：Hadoop 伪分布模式安装

## 一：任务目标

- 1、了解 Hadoop 的 3 种运行模式
- 2、熟练掌握 Hadoop 伪分布模式安装流程
- 3、培养独立完成 Hadoop 伪分布安装的能力

## 二：相关知识

Hadoop 由 Apache 基金会开发的分布式系统基础架构，是利用集群对大量数据进行分布式处理和存储的软件框架。用户可以轻松地在 Hadoop 集群上开发和运行处理海量数据的应用程序。Hadoop 有高可靠，高扩展，高效性，高容错等优点。Hadoop 框架最核心的设计就是 HDFS 和 MapReduce。HDFS 为海量的数据提供了存储，MapReduce 为海量的数据提供了计算。此外，Hadoop 还包括了 Hive，Hbase，ZooKeeper，Pig，Avro，Sqoop，Flume，Mahout 等项目。

Hadoop 的运行模式分为 3 种：本地运行模式，伪分布运行模式，完全分布运行模式。

### (1) 本地模式 (local mode)

这种运行模式在一台单机上运行，没有 HDFS 分布式文件系统，而是直接读写本地操作系统中的文件系统。在本地运行模式 (local mode) 中不存在守护进程，所有进程都运行在一个 JVM 上。单机模式适用于开发阶段运行 MapReduce 程序，这也是最少使用的一个模式。

### (2) 伪分布模式

这种运行模式是在单台服务器上模拟 Hadoop 的完全分布模式，单机上的分布式并不是真正的分布式，而是使用线程模拟的分布式。在这个模式中，所有守护进程(NameNode, DataNode, ResourceManager, NodeManager, SecondaryNameNode)都在同一台机器上运行。因为伪分布运行模式的 Hadoop 集群只有一个节点，所以 HDFS 中的块复制将限制为单个副本，其 secondary-master 和 slave 也都将运行于本地主机。此种模式除了并非真正意义的分布式之外，其程序执行逻辑完全类似于完全分布式，因此，常用于开发人员测试程序的执行。本次实验就是在台服务器上进行伪分布运行模式的搭建。

### (3) 完全分布模式

这种模式通常被用于生产环境，使用 N 台主机组成一个 Hadoop 集群，Hadoop 守护进程运行在每台主机之上。这里会存在 Namenode 运行的主机，Datanode 运行的主机，以及 SecondaryNameNode 运行的主机。在完全分布式环境下，主节点和从节点会分开。

## 三、系统环境

Linux Ubuntu 16.04

## 四、任务内容

在只安装 Linux 系统的服务器上，安装 Hadoop2.6.0 伪分布模式。

## 五、任务步骤

1.此步为可选项，建议用户创建一个新用户及用户组，后续的操作基本都是在此用户下来操作。但是用户亦可在自己当前非 root 用户下进行操作。 创建一个用户，名为 zhangyu，并为此用户创建 home 目录，此时会默认创建一个与 zhangyu 同名的用户组。

[view plain copy](#)

```
1. sudo useradd -d /home/zhangyu -m zhangyu
```

为 zhangyu 用户设置密码，执行下面的语句

[view plain copy](#)

```
1. sudo passwd zhangyu
```

按提示消息，输入密码以及确认密码即可，此处密码设置为 zhangyu

将 zhangyu 用户的权限，提升到 sudo 超级用户级别

[view plain copy](#)

```
1. sudo usermod -G sudo zhangyu
```

后续操作，我们需要切换到 zhangyu 用户下来进行操作。

[view plain copy](#)

```
1. su - zhangyu
```

2.首先来配置 SSH 免密码登陆

SSH 免密码登陆需要在服务器执行以下命令，生成公钥和私钥对

[view plain copy](#)

```
1. ssh-keygen -t rsa
```

此时会有多处提醒输入在冒号后输入文本，这里主要是要求输入 ssh 密码以及密码的放置位置。在这里，只需要使用默认值，按回车即可。

[view plain copy](#)

```
1. zhangyu@b6b1577cfbc8:/apps$ ssh-keygen -t rsa
```

```
2. Generating public/private rsa key pair.
```

```
3. Enter file in which to save the key (/home/zhangyu/.ssh/id_rsa):
```

```
4. Created directory '/home/zhangyu/.ssh'.
```

```
5. Enter passphrase (empty for no passphrase):
```

```
6. Enter same passphrase again:
```

```
7. Your identification has been saved in /home/zhangyu/.ssh/id_rsa.
```

```
8. Your public key has been saved in /home/zhangyu/.ssh/id_rsa.pub.
```

```
9. The key fingerprint is:
```

```
10. b3:00:c6:75:86:d6:8b:17:45:c6:7d:a1:74:aa:16:a7 zhangyu@b6b1577cfbc8
```

```
11. The key's randomart image is:
```

```
12. +--[ RSA 2048]-----+
```

```
13. |      .oo++.. o. |
```

```
14. |   . .ooo....+. |
```

```
15. |   +. . o. +. |
```

```
16. |   . .. o  =  |
```

```
17. |      ..S E   |
```

```
18. |      . +     |
```

```
19. |      .       |
```

```
20. |
21. |
22. +-----+
23. zhangyu@b6b1577cfbc8:/apps$
```

此时 ssh 公钥和私钥已经生成完毕，且放置在 ~/.ssh 目录下。切换到 ~/.ssh 目录下

```
view plain copy
1. cd ~/.ssh
```

可以看到 ~/.ssh 目录下的文件

```
view plain copy
1. zhangyu@b6b1577cfbc8:~/.ssh$ ll
2. 总用量 16
3. drwx----- 2 zhangyu zhangyu 4096 11 月 1 06:37 ./
4. drwxr-xr-x 51 zhangyu zhangyu 4096 11 月 1 06:37 ../
5. -rw----- 1 zhangyu zhangyu 1675 11 月 1 06:37 id_rsa
6. -rw-r--r-- 1 zhangyu zhangyu 402 11 月 1 06:37 id_rsa.pub
7. zhangyu@b6b1577cfbc8:~/.ssh$
```

下面在 ~/.ssh 目录下，创建一个空文本，名为 authorized\_keys

```
view plain copy
1. touch ~/.ssh/authorized_keys
```

将存储公钥文件的 id\_rsa.pub 里的内容，追加到 authorized\_keys 中

```
view plain copy
1. cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

下面执行 ssh localhost 测试 ssh 配置是否正确

```
view plain copy
1. ssh localhost
```

第一次使用 ssh 访问，会提醒是否继续连接，输入 “yes” 继续进行，执行完以后退出

```
view plain copy
1. zhangyu@b6b1577cfbc8:~/.ssh$ ssh localhost
2. The authenticity of host 'localhost (127.0.0.1)' can't be established.
3. ECDSA key fingerprint is 72:63:26:51:c7:2a:9e:81:24:55:5c:43:b6:7c:14:10.
4. Are you sure you want to continue connecting (yes/no)? yes
5. Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
6. Welcome to Ubuntu 14.04.2 LTS (GNU/Linux 3.16.0-23-generic x86_64)
7. * Documentation:  https://help.ubuntu.com/
8. Last login: Tue Nov 1 06:04:05 2016 from 192.168.1.179
9. zhangyu@b6b1577cfbc8:~$
10. zhangyu@b6b1577cfbc8:~$ exit
11. 注销
12. Connection to localhost closed.
13. zhangyu@b6b1577cfbc8:~/.ssh$
```

后续再执行 ssh localhost 时，就不用输入密码了

3.下面首先来创建两个目录，用于存放安装程序及数据。

```
1. sudo mkdir /apps
```

```
1. sudo mkdir /data
```

并为/apps 和/data 目录切换所属的用户为 zhangyu 及用户组为 zhangyu

```
1. sudo chown -R zhangyu:zhangyu /apps
```

```
1. sudo chown -R zhangyu:zhangyu /data
```

两个目录的作用分别为：/apps 目录用来存放安装的框架，/data 目录用来存放临时数据、HDFS 数据、程序代码或脚本。

切换到根目录下，执行 ls -l 命令

```
1. cd /
2. ls -l
```

可以看到根目录下/apps 和/data 目录所属用户及用户组已切换为 zhangyu:zhangyu

```
1. drwxr-xr-x 171 root    root    4096 11 月  2 01:56 ./
2. drwxr-xr-x 171 root    root    4096 11 月  2 01:56 ../
3. drwxr-xr-x   4 zhangyu zhangyu 4096 11 月  1 02:39 apps/
4. drwxr-xr-x   2 root    root    4096 11 月  1 02:56 bin/
5. drwxr-xr-x   2 root    root    4096  4 月 10 2014 boot/
6. drwxr-xr-x   2 zhangyu zhangyu 4096 11 月  2 01:56 data/
7. -rw-r--r--   1 root    root   193531  8 月 17 10:04 desk.jpg
```

4.配置 HDFS。

创建/data/hadoop1 目录，用来存放相关安装工具，如 jdk 安装包 jdk-7u75-linux-x64.tar.gz 及 hadoop 安装包 hadoop-2.6.0-cdh5.4.5.tar.gz。

```
1. mkdir -p /data/hadoop1
```

切换目录到/data/hadoop1 目录，使用 wget 命令，下载所需的 hadoop 安装包 jdk-7u75-linux-x64.tar.gz 及 hadoop-2.6.0-cdh5.4.5.tar.gz。

```
1. cd /data/hadoop1
2. wget http://192.168.1.100:60000/allfiles/hadoop1/jdk-7u75-linux-x64.tar.gz
3. wget http://192.168.1.100:60000/allfiles/hadoop1/hadoop-2.6.0-cdh5.4.5.tar.gz
```

5.安装 jdk。将/data/hadoop1 目录下 jdk-7u75-linux-x64.tar.gz 解压缩到/apps 目录下。

```
1. tar -xzvf /data/hadoop1/jdk-7u75-linux-x64.tar.gz -C /apps
```

其中，tar -xzvf 对文件进行解压缩，-C 指定解压后，将文件放到/apps 目录下。

切换到/apps 目录下，我们可以看到目录下内容如下：

```
1. cd /apps/  
2. ls -l
```

下面将jdk1.7.0\_75 目录重命名为java，执行：

```
1. mv /apps/jdk1.7.0_75/ /apps/java
```

6.下面来修改环境变量：系统环境变量或用户环境变量。我们在这里修改用户环境变量。

```
1. sudo vim ~/.bashrc
```

输入上面的命令，打开存储环境变量的文件。空几行，将java 的环境变量，追加进用户环境变量中。

```
1. #java  
2. export JAVA_HOME=/apps/java  
3. export PATH=$JAVA_HOME/bin:$PATH
```

输入 Esc, 进入 vim 命令模式，输入 :wq !进行保存。

让环境变量生效。

```
1. source ~/.bashrc
```

执行 source 命令，让java 环境变量生效。执行完毕后，可以输入 java，来测试环境变量是否配置正确。

如果出现下面界面，则正常运行。

```
1. java
```

```
1. zhangyu@6ebe6fe49149:~$ java  
2. 用法: java [-options] class [args...]  
3.           (执行类)  
4. 或 java [-options] -jar jarfile [args...]  
5.           (执行 jar 文件)  
6. 其中选项包括:  
7.     -d32      使用 32 位数据模型 (如果可用)  
8.     -d64      使用 64 位数据模型 (如果可用)  
9.     -server   选择 "server" VM  
10.             默认 VM 是 server,  
11.             因为您是在服务器类计算机上运行。  
12.     -cp <目录和 zip/jar 文件的类搜索路径>  
13.     -classpath <目录和 zip/jar 文件的类搜索路径>  
14.             用 : 分隔的目录, JAR 档案  
15.             和 ZIP 档案列表, 用于搜索类文件。  
16.     -D<名称>=<值>
```

17. 设置系统属性
18. `-verbose:[class|gc|jni]`
19. 启用详细输出
20. `-version` 输出产品版本并退出
21. `-version:<值>`
22. 需要指定的版本才能运行
23. `-showversion` 输出产品版本并继续
24. `-jre-restrict-search | -no-jre-restrict-search`
25. 在版本搜索中包括/排除用户专用 JRE
26. `-? -help` 输出此帮助消息
27. `-X` 输出非标准选项的帮助
28. `-ea[:<packagename>...|:<classname>]`
29. `-enableassertions[:<packagename>...|:<classname>]`
30. 按指定的粒度启用断言
31. `-da[:<packagename>...|:<classname>]`
32. `-disableassertions[:<packagename>...|:<classname>]`
33. 禁用具有指定粒度的断言
34. `-esa | -enablesystemassertions`
35. 启用系统断言
36. `-dsa | -disablesystemassertions`
37. 禁用系统断言
38. `-agentlib:<libname>[=<选项>]`
39. 加载本机代理库 <libname>, 例如 `-agentlib:hprof`
40. 另请参阅 `-agentlib:jdwp=help` 和 `-agentlib:hprof=help`
41. `-agentpath:<pathname>[=<选项>]`
42. 按完整路径名加载本机代理库
43. `-javaagent:<jarpath>[=<选项>]`
44. 加载 Java 编程语言代理, 请参阅 `java.lang.instrument`
45. `-splash:<imagepath>`
46. 使用指定的图像显示启动屏幕
47. 有关详细信息, 请参阅 <http://www.oracle.com/technetwork/java/javase/documentation/index.html>。
48. `zhangyu@6ebe6fe49149:~$`

7.下面安装 hadoop, 切换到/data/hadoop1 目录下, 将 hadoop-2.6.0-cdh5.4.5.tar.gz 解压缩到/apps 目录下。

[view plain copy](#)

1. `cd /data/hadoop1`
2. `tar -xvzf /data/hadoop1/hadoop-2.6.0-cdh5.4.5.tar.gz -C /apps/`

为了便于操作, 我们也将 hadoop-2.6.0-cdh5.4.5 重命名为 hadoop。

[view plain copy](#)

1. `mv /apps/hadoop-2.6.0-cdh5.4.5/ /apps/hadoop`

8.修改用户环境变量, 将 hadoop 的路径添加到 path 中。先打开用户环境变量文件。



[view plain copy](#)

```
1. sudo vim ~/.bashrc
```

将以下内容追加到环境变量~/.bashrc 文件中。

[view plain copy](#)

```
1. #hadoop
2. export HADOOP_HOME=/apps/hadoop
3. export PATH=$HADOOP_HOME/bin:$PATH
```

让环境变量生效。

[view plain copy](#)

```
1. source ~/.bashrc
```

验证 hadoop 环境变量配置是否正常

[view plain copy](#)

```
1. hadoop version
```

```
zhangyu@fdb0cf228a43:/data/hadoop1$ hadoop version
Hadoop 2.6.0-cdh5.4.5
Subversion http://github.com/cloudera/hadoop -r ab14c89fe25e9fb3f9de4fb852c21365b7c5608b
Compiled by jenkins on 2015-08-12T21:08Z
Compiled with protoc 2.5.0
From source with checksum d31cb7e46b8602edaf68d335b785ab
This command was run using /apps/hadoop/share/hadoop/common/hadoop-common-2.6.0-cdh5.4.5.jar
zhangyu@fdb0cf228a43:/data/hadoop1$
```

9.下面来修改 hadoop 本身相关的配置。首先切换到 hadoop 配置目录下。

[view plain copy](#)

```
1. cd /apps/hadoop/etc/hadoop
```

10.输入 vim /apps/hadoop/etc/hadoop/hadoop-env.sh , 打开 hadoop-env.sh 配置文件。

[view plain copy](#)

```
1. vim /apps/hadoop/etc/hadoop/hadoop-env.sh
```

将下面 JAVA\_HOME 追加到 hadoop-env.sh 文件中。

[view plain copy](#)

```
1. export JAVA_HOME=/apps/java
```

11.输入 vim /apps/hadoop/etc/hadoop/core-site.xml , 打开 core-site.xml 配置文件。

[view plain copy](#)

```
1. vim /apps/hadoop/etc/hadoop/core-site.xml
```

添加下面配置到<configuration>与</configuration>标签之间。

[view plain copy](#)

```
1. <property>
2.     <name>hadoop.tmp.dir</name>
3.     <value>/data/tmp/hadoop/tmp</value>
4. </property>
5. <property>
6.     <name>fs.defaultFS</name>
7.     <value>hdfs://localhost:9000</value>
8. </property>
```

这里有两项配置：

一项是 `hadoop.tmp.dir`，配置 `hadoop` 处理过程中，临时文件的存储位置。这里的目录 `/data/tmp/hadoop/tmp` 需要提前创建。

view plain copy

```
1. mkdir -p /data/tmp/hadoop/tmp
```

另一项是 `fs.defaultFS`，配置 `hadoop` HDFS 文件系统的地址。

12. 输入 `vim /apps/hadoop/etc/hadoop/hdfs-site.xml`，打开 `hdfs-site.xml` 配置文件。

view plain copy

```
1. vim /apps/hadoop/etc/hadoop/hdfs-site.xml
```

添加下面配置到 `<configuration>` 与 `</configuration>` 标签之间。

view plain copy

```
1. <property>
2.     <name>dfs.namenode.name.dir</name>
3.     <value>/data/tmp/hadoop/hdfs/name</value>
4. </property>
5. <property>
6.     <name>dfs.datanode.data.dir</name>
7.     <value>/data/tmp/hadoop/hdfs/data</value>
8. </property>
9. <property>
10.    <name>dfs.replication</name>
11.    <value>1</value>
12. </property>
13. <property>
14.    <name>dfs.permissions.enabled</name>
15.    <value>>false</value>
16. </property>
```

配置项说明：

`dfs.namenode.name.dir`，配置元数据信息存储位置；

`dfs.datanode.data.dir`，配置具体数据存储位置；

`dfs.replication`，配置每个数据库备份数，由于目前我们使用 1 台节点，所以，设置为 1，如果设置为 2 的话，运行会报错。

`dfs.permissions.enabled`，配置 `hdfs` 是否启用权限认证

另外 `/data/tmp/hadoop/hdfs` 路径，需要提前创建，所以我们需要执行

view plain copy

```
1. mkdir -p /data/tmp/hadoop/hdfs
```

13. 输入 `vim /apps/hadoop/etc/hadoop/slaves`，打开 `slaves` 配置文件。

view plain copy

```
1. vim /apps/hadoop/etc/hadoop/slaves
```

将集群中 `slave` 角色的节点的主机名，添加进 `slaves` 文件中。目前只有一台节点，所以 `slaves` 文件内容为：

view plain copy

```
1. localhost
```

14.下面格式化 HDFS 文件系统。执行：

view plain copy  
1. `hadoop namenode -format`

15.切换目录到/apps/hadoop/sbin 目录下。

view plain copy  
1. `cd /apps/hadoop/sbin/`

16.启动 hadoop 的 hdfs 相关进程。

view plain copy  
1. `./start-dfs.sh`

这里只会启动 HDFS 相关进程。

17.输入 jps 查看 HDFS 相关进程是否已经启动。

view plain copy  
1. `jps`

```
zhangyu@fdb0cf228a43:/apps/hadoop/sbin$ jps
703 DataNode
600 NameNode
900 SecondaryNameNode
1394 Jps
zhangyu@fdb0cf228a43:/apps/hadoop/sbin$
```

我们可以看到相关进程，都已经启动。

18.下面可以再进一步验证 HDFS 运行状态。先在 HDFS 上创建一个目录。

view plain copy  
1. `hadoop fs -mkdir /myhadoop1`

19.执行下面命令，查看目录是否创建成功。

view plain copy  
1. `hadoop fs -ls -R /`

```
zhangyu@fdb0cf228a43:/apps/hadoop/sbin$ hadoop fs -ls -R /
17/01/11 03:56:17 WARN util.NativeCodeLoader: Unable to load native-hadoop
va classes where applicable
drwxr-xr-x  - zhangyu supergroup          0 2017-01-11 03:56 /myhadoop1
zhangyu@fdb0cf228a43:/apps/hadoop/sbin$
```

以上，便是 HDFS 安装过程。

20.下面来配置 MapReduce 相关配置。再次切换到 hadoop 配置文件目录

view plain copy  
1. `cd /apps/hadoop/etc/hadoop`

21.下面将 mapreduce 的配置文件 mapred-site.xml.template，重命名为 mapred-site.xml。

view plain copy  
1. `mv /apps/hadoop/etc/hadoop/mapred-site.xml.template /apps/hadoop/etc/hadoop/mapred-site.xml`

22.输入 vim /apps/hadoop/etc/hadoop/mapred-site.xml，打开 mapred-site.xml 配置文件。

[view plain copy](#)

```
1. vim /apps/hadoop/etc/hadoop/mapred-site.xml
```

将 mapreduce 相关配置，添加到<configuration>标签之间。

[view plain copy](#)

```
1. <property>
2.     <name>mapreduce.framework.name</name>
3.     <value>yarn</value>
4. </property>
```

这里指定 mapreduce 任务处理所使用的框架。

23.输入 vim /apps/hadoop/etc/hadoop/yarn-site.xml，打开 yarn-site.xml 配置文件。

[view plain copy](#)

```
1. vim /apps/hadoop/etc/hadoop/yarn-site.xml
```

将 yarn 相关配置，添加到<configuration>标签之间。

[view plain copy](#)

```
1. <property>
2.     <name>yarn.nodemanager.aux-services</name>
3.     <value>mapreduce_shuffle</value>
4. </property>
```

这里的配置是指定所用服务，默认为空。

24.下面来启动计算层面相关进程，切换到 hadoop 启动目录。

[view plain copy](#)

```
1. cd /apps/hadoop/sbin/
```

25.执行命令，启动 yarn。

[view plain copy](#)

```
1. ./start-yarn.sh
```

26.输入 jps，查看当前运行的进程。

```
zhangyu@fdb0cf228a43:/apps/hadoop/sbin$ jps
703 DataNode
1547 ResourceManager
600 NameNode
1650 NodeManager
2008 Jps
900 SecondaryNameNode
zhangyu@fdb0cf228a43:/apps/hadoop/sbin$
```

27.执行测试。

切换到/apps/hadoop/share/hadoop/mapreduce 目录下。

[view plain copy](#)

```
1. cd /apps/hadoop/share/hadoop/mapreduce
```

然后，在该目录下跑一个 mapreduce 程序，来检测一下 hadoop 是否能正常运行。

[view plain copy](#)

```
1. hadoop jar hadoop-mapreduce-examples-2.6.0-cdh5.4.5.jar pi 3 3
```

这个程序是计算数学中的 pi 值。当然暂时先不用考虑数据的准确性。当你看到下面流程的时候，表示程序已正常运行，hadoop 环境也是没问题的。

```
17/01/11 04:02:35 INFO mapreduce.Job: Running job: job_1484107180814_0001
17/01/11 04:02:42 INFO mapreduce.Job: Job job_1484107180814_0001 running in uber mode : false
17/01/11 04:02:42 INFO mapreduce.Job: map 0% reduce 0%
17/01/11 04:02:48 INFO mapreduce.Job: map 67% reduce 0%
17/01/11 04:02:49 INFO mapreduce.Job: map 100% reduce 0%
17/01/11 04:02:54 INFO mapreduce.Job: map 100% reduce 100%
17/01/11 04:02:54 INFO mapreduce.Job: Job job_1484107180814_0001 completed successfully
```

至此，Hadoop 伪分布模式已经安装完成!

## 实验二：Hadoop 开发插件安装

### 一、任务目标

- 1.了解 Eclipse 开发环境的使用
- 2.熟练掌握 Hadoop 开发插件安装

### 二、相关知识

Eclipse 是一个开放源代码的、基于 Java 的可扩展开发平台。就其本身而言，它只是一个框架和一组服务，用于通过插件组件构建开发环境。幸运的是 Eclipse 附带了一个标准的插件集，包括 Java 开发工具 (Java Development Tools, JDT)。

Eclipse 的插件机制是轻型软件组件化架构。在客户机平台上，Eclipse 使用插件来提供所有的附加功能，例如支持 Java 以外的其他语言。已有的分离的插件已经能够支持 C/C++ (CDT)、Perl、Ruby、Python、telnet 和数据库开发。插件架构能够支持将任意的扩展加入到现有环境中，例如配置管理，而决不仅仅限于支持各种编程语言。

Eclipse 的设计思想是：一切皆插件。Eclipse 核心很小，其它所有功能都以插件的形式附加于 Eclipse 核心之上。

Eclipse 基本内核包括：图形 API (SWT/Jface)，Java 开发环境插件(JDT)，插件开发环境(PDE)等。Hadoop 是一个强大的并行软件开发框架，它可以让任务在分布式集群上并行处理，从而提高执行效率。但是，它也有一些缺点，如编码、调试 Hadoop 程序的难度较大，这样的缺点直接导致开发人员入门门槛高，开发难度大。

因此，Hadoop 的开发者为了降低 Hadoop 的难度，开发出了 Hadoop Eclipse 插件，它可以直接嵌入到 Hadoop 开发环境中，从而实现了开发环境的图形界面化，降低了编程的难度。

Hadoop Eclipse 是 Hadoop 开发环境的插件，在安装该插件之前需要首先配置 Hadoop 的相关信息。用户在创建 Hadoop 程序时，Eclipse 插件会自动导入 Hadoop 编程接口的 jar 文件，这样用户就可以在 Eclipse 插件的图形界面中进行编码、调试和运行 Hadoop 程序，也能通过 Eclipse 插件查看程序的实时状态、错误信息以及运行结果。除此之外，用户还可以通过 Eclipse 插件对 HDFS 进行管理和查看。总而言之，Hadoop Eclipse 插件不仅安装简单，使用起来也很方便。它的功能强大，特别在 Hadoop 编程方面为开发者降低了很大的难度，是 Hadoop 入门和开发的好帮手！

Eclipse 插件的安装方法大体有以下四种：第一种：直接复制法，第二种：使用 link 文件法，第三种：使用 eclipse 自带图形界面安装，第四种：使用 dropins 安装插件，本实验 Hadoop 开发插件安装使用了 eclipse 自带图形界面安装。

### 三、系统环境

Linux Ubuntu 16.04

jdk-7u75-linux-x64

hadoop-2.6.0-cdh5.4

eclipse-java-juno-SR2-linux-gtk-x86\_64

## 四、任务内容

在已经安装好 Hadoop 和 Eclipse 环境中，将 Hadoop 插件安装到 Eclipse 工具上并对插件进行配置。

## 五、任务步骤

1.Eclipse 开发工具以及 Hadoop 默认已经安装完毕，安装在/apps/目录下。

2.在 Linux 本地创建/data/hadoop3 目录，用于存放所需文件。

```
view plain copy
1. mkdir -p /data/hadoop3
```

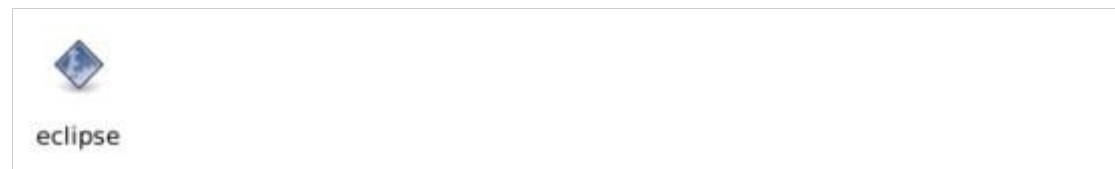
切换目录到 /data/hadoop3 目录下，并使用 wget 命令，下载所需的插件 hadoop-eclipse-plugin-2.6.0.jar。

```
view plain copy
1. cd /data/hadoop3
2. wget http://192.168.1.100:60000/allfiles/hadoop3/hadoop-eclipse-plugin-2.6.0.jar
```

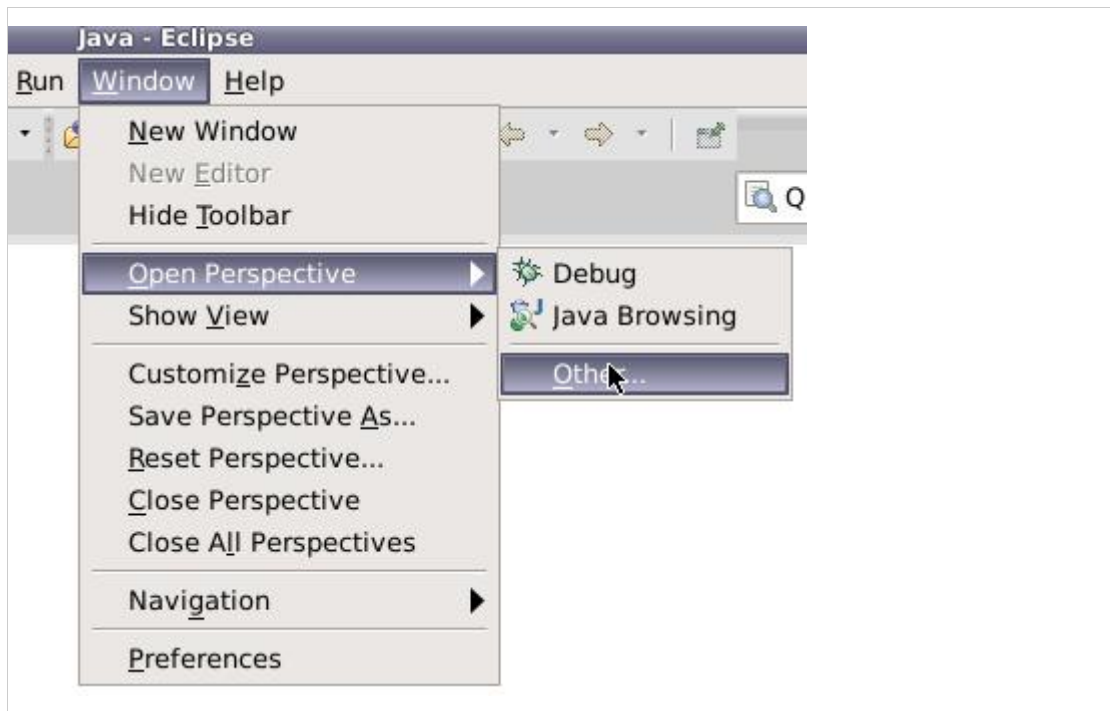
3.将插件 hadoop-eclipse-plugin-2.6.0.jar，从/data/hadoop3 目录下，拷贝到/apps/eclipse/plugins 的插件目录下。

```
view plain copy
1. cp /data/hadoop3/hadoop-eclipse-plugin-2.6.0.jar /apps/eclipse/plugins/
```

4.进入 ubuntu 图形界面，双击 eclipse 图标，启动 eclipse。



5.在 Eclipse 窗口界面，依次点击 Window => Open Perspective => Other。

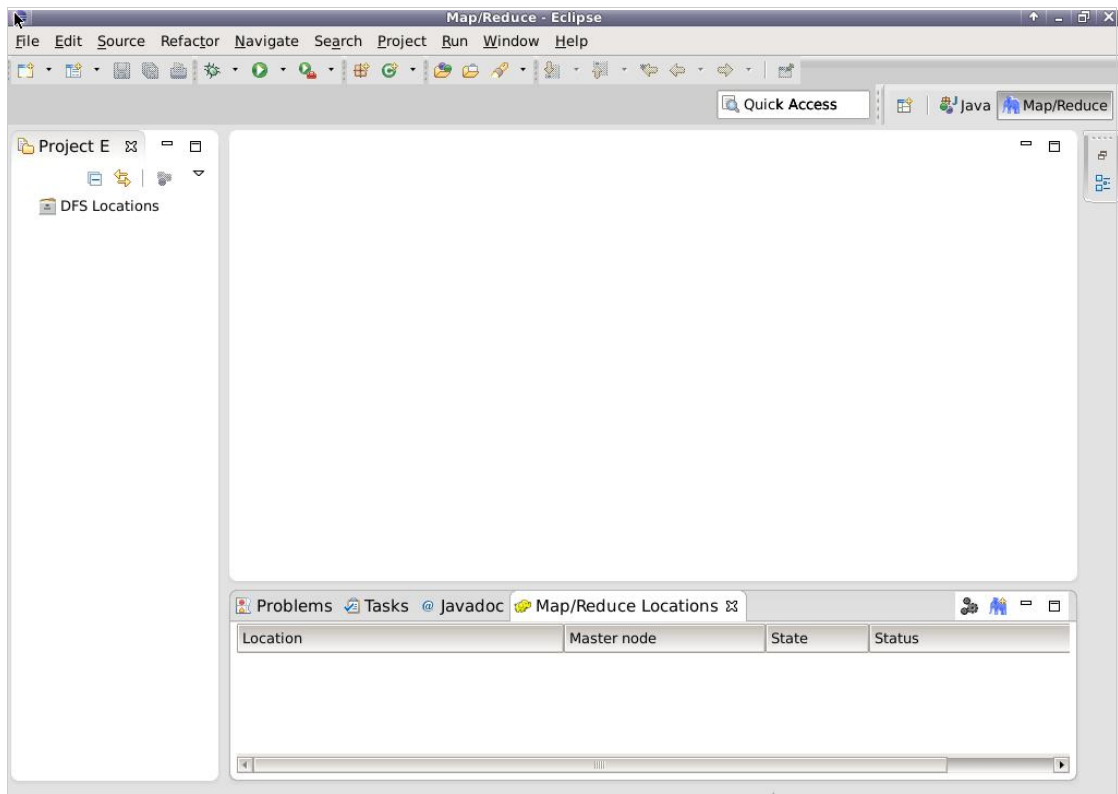


弹出一个窗口。

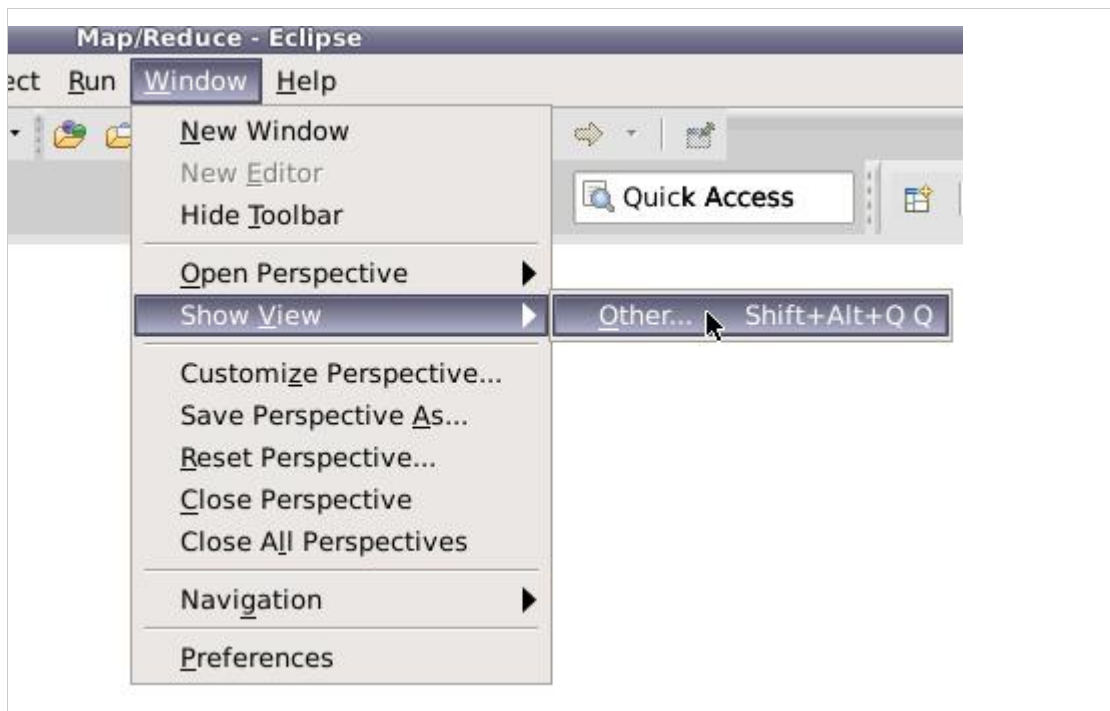


选择 Map/Reduce ,并点击 OK ,可以看到窗口中 ,有三个变化。( 左侧项目浏览器、右上角操作布局切换、面板窗口 )

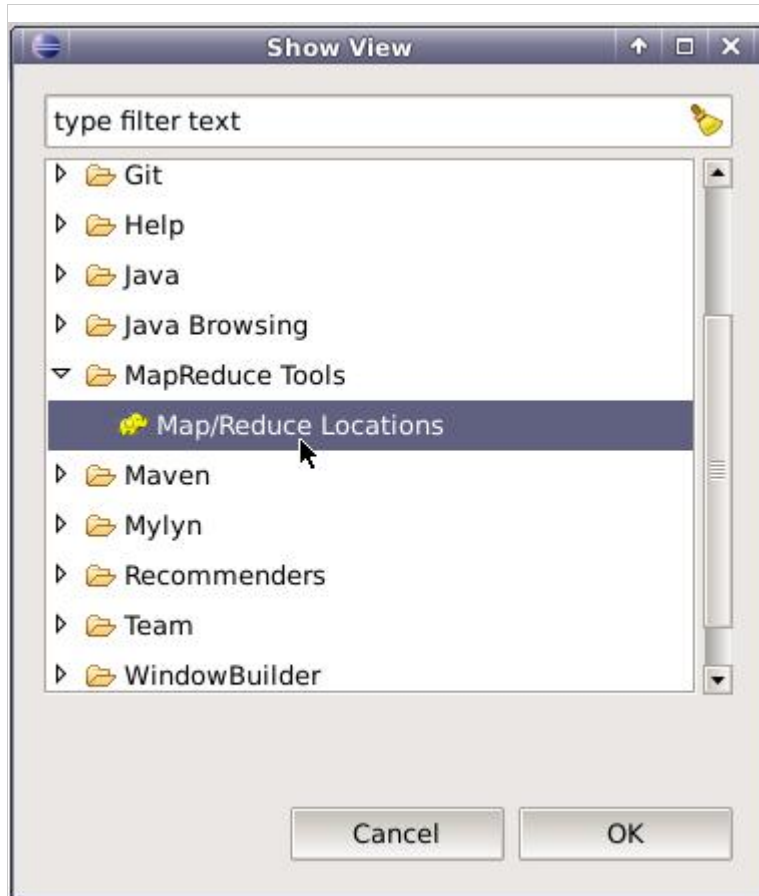




如果在 windows 下 ,则需要手动调出面板窗口 Map/Reduce Locations 面板 ,操作为 ,点击 window => show view => Other.

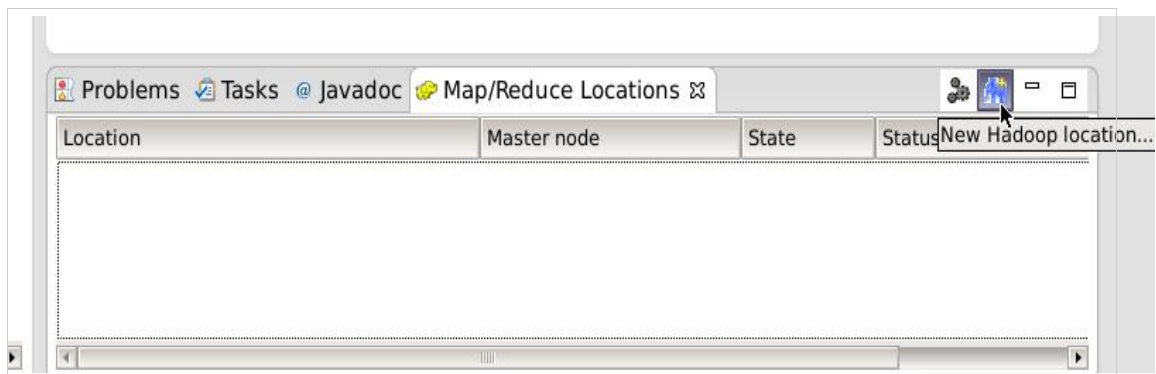


在弹出的窗口中 ,选择 Map/Reduce Locations 选项 ,并点击 OK。



这样便可以调出视图窗口 Map/Reduce Location。

6. 添加 Hadoop 配置，连接 Hadoop 集群。



在这里添加 Hadoop 相关配置。

**New Hadoop location...**

**Define Hadoop location**

Define the location of a Hadoop infrastructure for running MapReduce applications.

**General** **Advanced parameters**

**Location name:** myhadoop

**Map/Reduce(V2) Master**

Host: localhost

Port: 50020

**DFS Master**

☒ Use M/R Master host

Host: localhost

Port: 9000

**User name:** zhangyu

**SOCKS proxy**

☐ Enable SOCKS proxy

Host: host

Port: 1080

**Load from file** **Validate location**

**?** **Cancel** **Finish**

Location name，是为此配置起的一个名字。

DFS Master，是连接 HDFS 的主机名和端口号。

点击 Finish 保存配置。

7.另外还需保证 Hadoop 中的 HDFS 相关进程已经启动。在终端命令行输入 jps 查看进程状态。

[view plain copy](#)

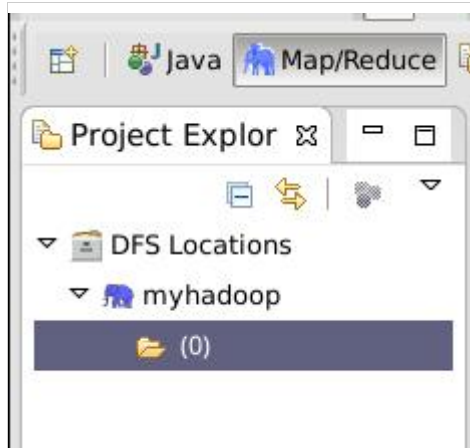
## 1. jps

若不存在 hdfs 相关的进程，如 Namenode、Datanode、secondarynamenode，则需要先切换到 HADOOP\_HOME 下的 sbin 目录，启动 hadoop。

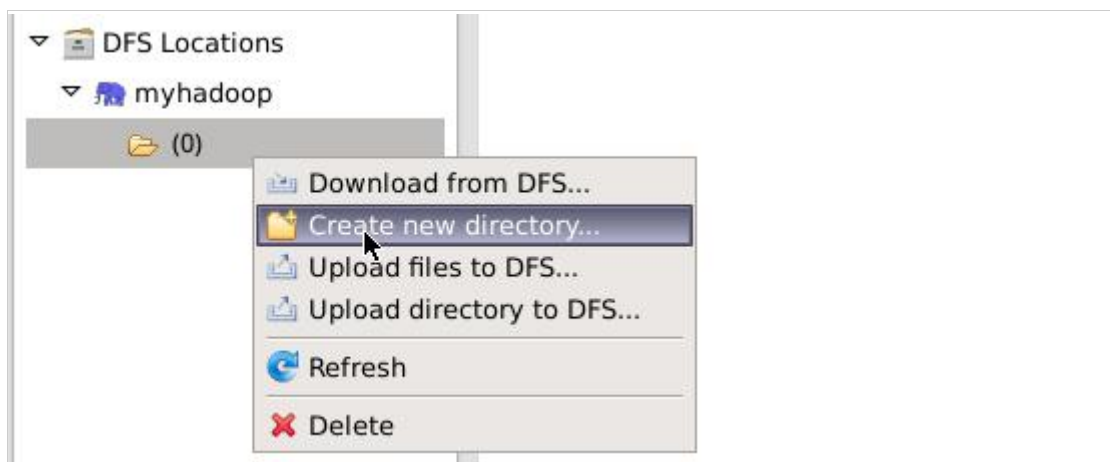
[view plain copy](#)

1. cd /apps/hadoop/sbin
2. ./start-all.sh

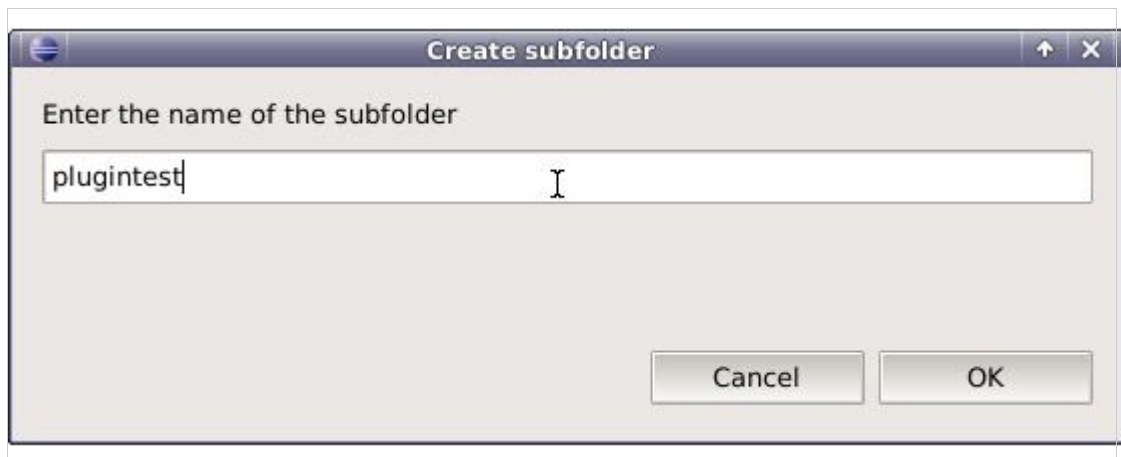
8.展开左侧项目浏览视图，可以看到 HDFS 目录结构。



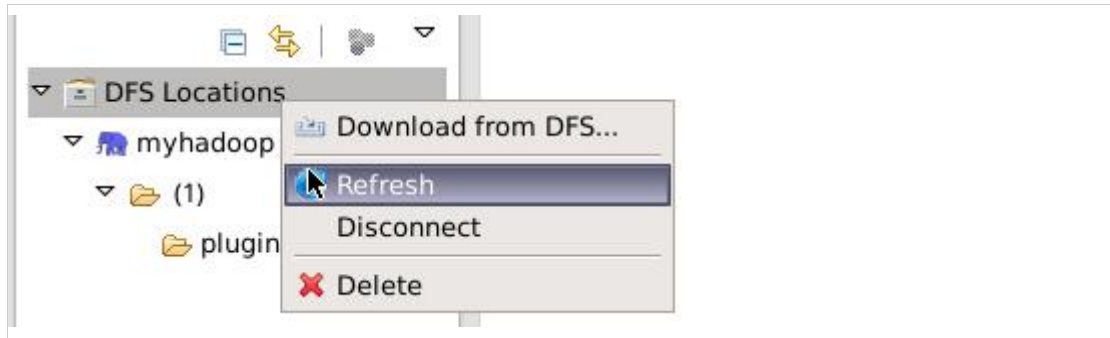
9.上图发现 HDFS 上，没有存放任何目录。那下面来创建一个目录，检测插件是否可用。



右键 myhadoop 下的文件夹，在弹出的菜单中，点击 Create new directory。



输入目录名称，点击 OK 则创建目录成功。



右键文件夹，点击 Refresh，可用刷新 HDFS 目录。（如果刷新不出来，重启 eclipse 即可）。  
到此 Hadoop 开发插件已经安装完毕！

## 实验三：Hadoop Shell 基本操作

### 一、任务目标

1.熟练掌握常用的 hadoop shell 命令

### 二、相关知识

调用文件系统(FS)Shell 命令应使用 `hadoop fs <args>`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`，或者更简单的 `/parent/child`（假设你配置文件中的默认值是 `namenode:namenodeport`）。大多数 FS Shell 命令的行为和对应的 Unix Shell 命令类似，出错信息会输出到 `stderr`，其他信息输出到 `stdout`。

### 三、系统环境

Linux Ubuntu 16.04

hadoop-2.6.0-cdh5.4

### 四、任务内容

- 1.学习在开启、关闭 Hadoop
- 2.学习在 Hadoop 中创建、修改、查看、删除文件夹及文件
- 3.学习改变文件的权限及文件的拥有者
- 4.学习使用 shell 命令提交 job 任务
- 5.Hadoop 安全模式的进入与退出

### 五、任务步骤

- 1.打开终端模拟器，切换到 `/apps/hadoop/sbin` 目录下，启动 Hadoop

```
view plain copy
1. cd /apps/hadoop/sbin
2. ./start-all.sh

zhangyu@a81ff3854876:~$ cd /apps/hadoop/sbin
zhangyu@a81ff3854876:/apps/hadoop/sbin$ ./start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [0.0.0.0]
0.0.0.0: starting namenode, logging to /apps/hadoop/logs/hadoop-zhangyu-namenode-a81ff3854876.out
localhost: starting datanode, logging to /apps/hadoop/logs/hadoop-zhangyu-datanode-a81ff3854876.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /apps/hadoop/logs/hadoop-zhangyu-secondarynamenode-a81ff3854876.out
starting yarn daemons
starting resourcemanager, logging to /apps/hadoop/logs/yarn-zhangyu-resourcemanager-a81ff3854876.out
localhost: starting nodemanager, logging to /apps/hadoop/logs/yarn-zhangyu-nodemanager-a81ff3854876.out
zhangyu@a81ff3854876:/apps/hadoop/sbin$
```

除了直接执行 `start-all.sh` 外，还可以分步启动 `start-dfs.sh` 和 `start-yarn.sh`。

2.执行 jps , 检查一下 Hadoop 相关进程是否启动

[view plain copy](#)

1. jps

```
zhangyu@a81ff3854876:/apps/hadoop/sbin$ jps
791 ResourceManager
444 DataNode
320 NameNode
618 SecondaryNameNode
1231 Jps
894 NodeManager
```

3.在/目录下创建一个 test1 文件夹

[view plain copy](#)

1. hadoop fs -mkdir /test1

4.在 Hadoop 中的 test1 文件夹中创建一个 file.txt 文件

[view plain copy](#)

1. hadoop fs -touchz /test1/file.txt

5.查看根目录下所有文件

[view plain copy](#)

1. hadoop fs -ls /

```
zhangyu@a81ff3854876:/apps/hadoop/sbin$ hadoop fs -ls /
Found 1 items
drwxr-xr-x - zhangyu supergroup          0 2017-08-04 07:35 /test1
```

6.还可以使用 ls -R 的方式递归查看根下所有文件

[view plain copy](#)

1. hadoop fs -ls -R /

```
zhangyu@a81ff3854876:/apps/hadoop/sbin$ hadoop fs -ls -R /
drwxr-xr-x - zhangyu supergroup          0 2017-08-04 07:35 /test1
-rw-r--r-- 1 zhangyu supergroup          0 2017-08-04 07:35 /test1/file.txt
zhangyu@a81ff3854876:/apps/hadoop/sbin$
```

7.将 Hadoop 根下 test1 目录中的 file.txt 文件 , 移动到根下并重命名为 file2.txt

[view plain copy](#)

1. hadoop fs -mv /test1/file.txt /file2.txt

Hadoop 中的 mv 用法同 Linux 中的一样 , 都可以起到移动文件和重命名的作用。

8.将 Hadoop 根下的 file2.txt 文件复制到 test1 目录下

[view plain copy](#)

1. hadoop fs -cp /file2.txt /test1

9.在 Linux 本地/data 目录下 , 创建一个 data.txt 文件 , 并向其中写入 hello hadoop !

[view plain copy](#)

1. cd /data

2. touch data.txt

3. echo hello hadoop! >> data.txt

10.将 Linux 本地/data 目录下的 data.txt 文件，上传到 HDFS 中的/test1 目录下

[view plain copy](#)

```
1. hadoop fs -put /data/data.txt /test1
```

11.查看 Hadoop 中/test1 目录下的 data.txt 文件

[view plain copy](#)

```
1. hadoop fs -cat /test1/data.txt
```

```
zhangyu@a81ff3854876:/data$ hadoop fs -cat /test1/data.txt
hello hadoop!
```

12.除此之外还可以使用 tail 方法

[view plain copy](#)

```
1. hadoop fs -tail /test1/data.txt
```

```
zhangyu@a81ff3854876:/data$ hadoop fs -tail /test1/data.txt
hello hadoop!
```

tail 方法是将文件尾部 1K 字节的内容输出。支持-f 选项，行为和 Unix 中一致。

13.查看 Hadoop 中/test1 目录下的 data.txt 文件大小

[view plain copy](#)

```
1. hadoop fs -du -s /test1/data.txt
```

```
zhangyu@a81ff3854876:/data$ hadoop fs -du -s /test1/data.txt
16 16 /test1/data.txt
```

-du 后面可以不加-s，直接写目录表示查看该目录下所有文件大小

14.text 方法可以将源文件输出为文本格式。允许的格式是 zip 和 TextRecordInputStream。

[view plain copy](#)

```
1. hadoop fs -text /test1/data.txt
```

```
zhangyu@a81ff3854876:/data$ hadoop fs -text /test1/data.txt
hello hadoop!
```

15.stat 方法可以返回指定路径的统计信息，有多个参数可选，当使用-stat 选项但不指定 format 时候，只打印文件创建日期，相当于%y

[view plain copy](#)

```
1. hadoop fs -stat /test1/data.txt
```

```
zhangyu@a81ff3854876:/data$ hadoop fs -stat /test1/data.txt
2017-08-04 07:38:06
```

下面列出了 format 的形式：

%b：打印文件大小（目录为 0）

%n：打印文件名

%o：打印 block size（我们要的值）



%r : 打印备份数

%y : 打印 UTC 日期 yyyy-MM-dd HH:mm:ss

%Y : 打印自 1970 年 1 月 1 日以来的 UTC 微秒数

%F : 目录打印 directory, 文件打印 regular file

16.将 Hadoop 中/test1 目录下的 data.txt 文件 , 下载到 Linux 本地/apps 目录中

```
view plain copy
1. hadoop fs -get /test1/data.txt /apps
```

17.查看一下/apps 目录下是否存在 data.txt 文件

```
view plain copy
1. ls /apps
```

```
zhangyu@f6e317fc6c81:~$ ls /apps
data.txt eclipse flume hadoop hbase hive java kafka scala spark sqoop tomcat6 toolbox zookeeper
```

18.使用 chown 方法 , 改变 Hadoop 中/test1 目录中的 data.txt 文件拥有者为 root , 使用-R 将使改变在目录结构下递归进行。

```
view plain copy
1. hadoop fs -chown root /test1/data.txt
```

19.使用 chmod 方法 , 赋予 Hadoop 中/test1 目录中的 data.txt 文件 777 权限

```
view plain copy
1. hadoop fs -chmod 777 /test1/data.txt
```

20.删除 Hadoop 根下的 file2.txt 文件

```
view plain copy
1. hadoop fs -rm /file2.txt
```

```
zhangyu@a81ff3854876:/data$ hadoop fs -rm /file2.txt
17/08/04 07:43:15 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Empty interval = 0 minutes.
Deleted /file2.txt
```

21.删除 Hadoop 根下的 test1 目录

```
view plain copy
1. hadoop fs -rm -r /test1
```

```
zhangyu@a81ff3854876:/data$ hadoop fs -rm -r /test1
17/08/04 07:43:39 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Empty interval = 0 minutes.
Deleted /test1
```

22.当在 Hadoop 中设置了回收站功能时 , 删除的文件会保留在回收站中 , 可以使用 expunge 方法清空回收站。

```
view plain copy
1. hadoop fs -expunge
```

```
zhangyu@a81ff3854876:/data$ hadoop fs -expunge
17/08/04 07:44:04 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Empty interval = 0 minutes.
```

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。

### 23.使用 Shell 命令执行 Hadoop 自带的 WordCount

首先切换到/data 目录下，使用 vim 编辑一个 data.txt 文件，内容为:hello world hello hadoop hello ipieuvre

- ```
view plain copy
```
1. `cd /data`
  2. `vim data.txt`

在 HDFS 的根下创建 in 目录，并将/data 下的 data.txt 文件上传到 HDFS 中的 in 目录

- ```
view plain copy
```
1. `hadoop fs -put /data/data.txt /in`

执行 `hadoop jar` 命令，在 hadoop 的 /apps/hadoop/share/hadoop/mapreduce 路径下存在 hadoop-mapreduce-examples-2.6.0-cdh5.4.5.jar 包，我们执行其中的 wordcount 类，数据来源为 HDFS 的/in 目录，数据输出到 HDFS 的/out 目录

- ```
view plain copy
```
1. `hadoop jar /apps/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.0-cdh5.4.5.jar wordcount /in /out`

查看 HDFS 中的/out 目录

- ```
view plain copy
```
1. `hadoop fs -ls /out`
  2. `hadoop fs -cat /out/*`

```
zhangyu@e2addc16fd8:/data$ hadoop fs -ls /out

Found 2 items
-rw-r--r--  1 zhangyu supergroup          0 2017-08-11 06:49 /out/_SUCCESS
-rw-r--r--  1 zhangyu supergroup        36 2017-08-11 06:49 /out/part-r-00000
zhangyu@e2addc16fd8:/data$ hadoop fs -cat /out/*
hadoop 1
hello  3
ipieuvre      1
world  1
zhangyu@e2addc16fd8:/data$
```

### 24.进入 Hadoop 安全模式

- ```
view plain copy
```
1. `hdfs dfsadmin -safemode enter`

```
zhangyu@a81ff3854876:/data$ hdfs dfsadmin -safemode enter
Safe mode is ON
```

### 25.退出 Hadoop 安全模式

```
view plain copy
```

1. `hdfs dfsadmin -safemode leave`

```
zhangyu@a81ff3854876:/data$ hdfs dfsadmin -safemode leave
Safe mode is OFF
```

26.切换到/apps/hadoop/sbin 目录下，关闭 Hadoop

[view plain copy](#)

1. `cd /apps/hadoop/sbin`
2. `./stop-all.sh`

```
zhangyu@a81ff3854876:/data$ cd /apps/hadoop/sbin
zhangyu@a81ff3854876:/apps/hadoop/sbin$ ./stop-all.sh
This script is Deprecated. Instead use stop-dfs.sh and stop-yarn.sh
Stopping namenodes on [0.0.0.0]
0.0.0.0: stopping namenode
localhost: stopping datanode
Stopping secondary namenodes [0.0.0.0]
0.0.0.0: stopping secondarynamenode
stopping yarn daemons
stopping resourcemanager
localhost: stopping nodemanager
no proxyserver to stop
```

## 实验四：HDFS JAVA API

### 一、任务目标

- 1.掌握 HDFS JAVA API 的
- 2.了解 JAVA API 的执行流程

### 二、相关知识

1.HDFS ( Hadoop Distributed File System ) 是 Hadoop 项目的核心子项目，是分布式计算中数据存储管理的基础篇，为了实现本地与 HDFS 的文件传输，主要借助 Eclipse 开发环境，通过 java 编程实现了远程 HDFS 的文件创建，上传，下载，删除等。

其实对 HDFS 的文件操作主要有两种方式：命令行的方式和 JavaAPI 的方式。命令行的方式简单直接，但是必须要求本地机器也是在 Linux 系统中已经安装了 hadoop，这对习惯用 windows 系统的用户来说不得不安装虚拟机，然后再在虚拟机上安装 Linux 系统，这是一种挑战。同时 windows 系统与虚拟机上安装的 Linux 系统进行文件传输也是要借助一些工具才可以实现。

为了实现以上所遇到诸如系统不一致，手动输入命令等的困扰，我们选择 Java API 的方式，有专门的 API 函数，可以在非 Hadoop 机器上实现访问，同时与系统无关（windows、Linux 甚至 XP 系统也可以）。Hadoop 中关于文件操作类基本上全部是在"org.apache.hadoop.fs"包中，Hadoop 类库中最终面向用户提供的接口类是 FileSystem，该类封装了几乎所有的文件操作，例如 CopyToLocalFile、CopyFromLocalFile、mkdir 及 delete 等。综上基本上可以得出操作文件的程序库框架：

```
operator() {
```

```
    得到 Configuration 对象
```

```
    得到 FileSystem 对象
```

```
    进行文件操作 }
```

2.下面介绍实现上述程序库框架中各个操作的具体步骤

Java 抽象类 org.apache.hadoop.fs.FileSystem 定义了 hadoop 的一个文件系统接口。该类是一个抽象类，通过以下两种静态工厂方法可以过去 FileSystem 实例：

```
public static FileSystem.get(Configuration conf) throws IOException
```

```
public static FileSystem.get(URI uri, Configuration conf) throws IOException
```

HDFS 上的文件创建，上传，下载，删除等操作的具体方法实现：

```
( 1 ) public boolean mkdirs(Path f) throws IOException
```

一次性新建所有目录（包括父目录），f 是完整的目录路径。

```
( 2 ) public FSOutputStream create(Path f) throws IOException
```

创建指定 path 对象的一个文件，返回一个用于写入数据的输出流

create()有多个重载版本，允许我们指定是否强制覆盖已有的文件、文件备份数量、写入文件缓冲区大小、文件块大小以及文件权限。

( 3 ) public boolean copyFromLocal(Path src, Path dst) throws IOException

将本地文件拷贝到文件系统

( 4 ) public boolean exists(Path f) throws IOException

检查文件或目录是否存在

( 5 ) public boolean delete(Path f, Boolean recursive)

永久性删除指定的文件或目录，如果 f 是一个空目录或者文件，那么 recursive 的值就会被忽略。只有 recursive = true 时，一个非空目录及其内容才会被删除。

( 6 ) FileStatus 类封装了文件系统中文件和目录的元数据，包括文件长度、块大小、备份、修改时间、所有者以及权限信息。

### 三、系统环境

Linux Ubuntu 16.04

jdk-7u75-linux-x64

hadoop-2.6.0-cdh5.4.5

hadoop-2.6.0-eclipse-cdh5.4.5.jar

eclipse-java-juno-SR2-linux-gtk-x86\_64

### 四、任务内容

本实验涉及到使用 Java API 对 HDFS 的一些基本操作。

- 1.创建类 MakeDir.class，在 HDFS 的根目录下，创建名为 hdfs-test 的目录。
- 2.创建类 TouchFile.class，在 HDFS 的目录/hdfs-test 下，创建名为 touchfile 的文件。
- 3.创建类 CopyFromLocalFile.class，将 linux 本地文件/data/mydata/sample\_data，上传到 HDFS 文件系统的/hdfs-test 目录下。
- 4.创建类 CopyToLocalFile.class，将 HDFS 文件系统上的文件/hdfs-test/sample\_data，下载到本地 /data/mydata/copytolocal。
- 5.创建类 ListFiles.class，列出 HDFS 文件系统/hdfs-test 目录下，所有的文件，以及文件的权限、用户组、所属用户。
- 6.创建类 IteratorListFiles.class，列出 HDFS 文件系统/根目录下，以及各级子目录下，所有文件以及文件的权限、用户组，所属用户。
- 7.了解 FileSystem 类下的方法，例如：判断文件是否存在、删除文件、重命名文件等。
- 8.创建类 LocateFile.class，查看 HDFS 文件系统上，文件/hdfs-test/sample\_data 的文件块信息。
- 9.创建类 WriteFile.class，在 HDFS 上，创建/hdfs-test/writefile 文件，并在文件中写入内容“hello world hello data!”。
- 10.创建类 PutMerge.class，将 Linux 本地文件夹/data/mydata/下的所有文件，上传到 HDFS 上并合并成一个文件/hdfs-test/mergefile。

## 五、任务步骤

1. 切换目录到/apps/hadoop/sbin 下，启动 hadoop。

view plain copy

1. `cd /apps/hadoop/sbin`
2. `./start-all.sh`

在 Linux 本地创建/data/hadoop4 目录。

view plain copy

1. `mkdir -p /data/hadoop4`

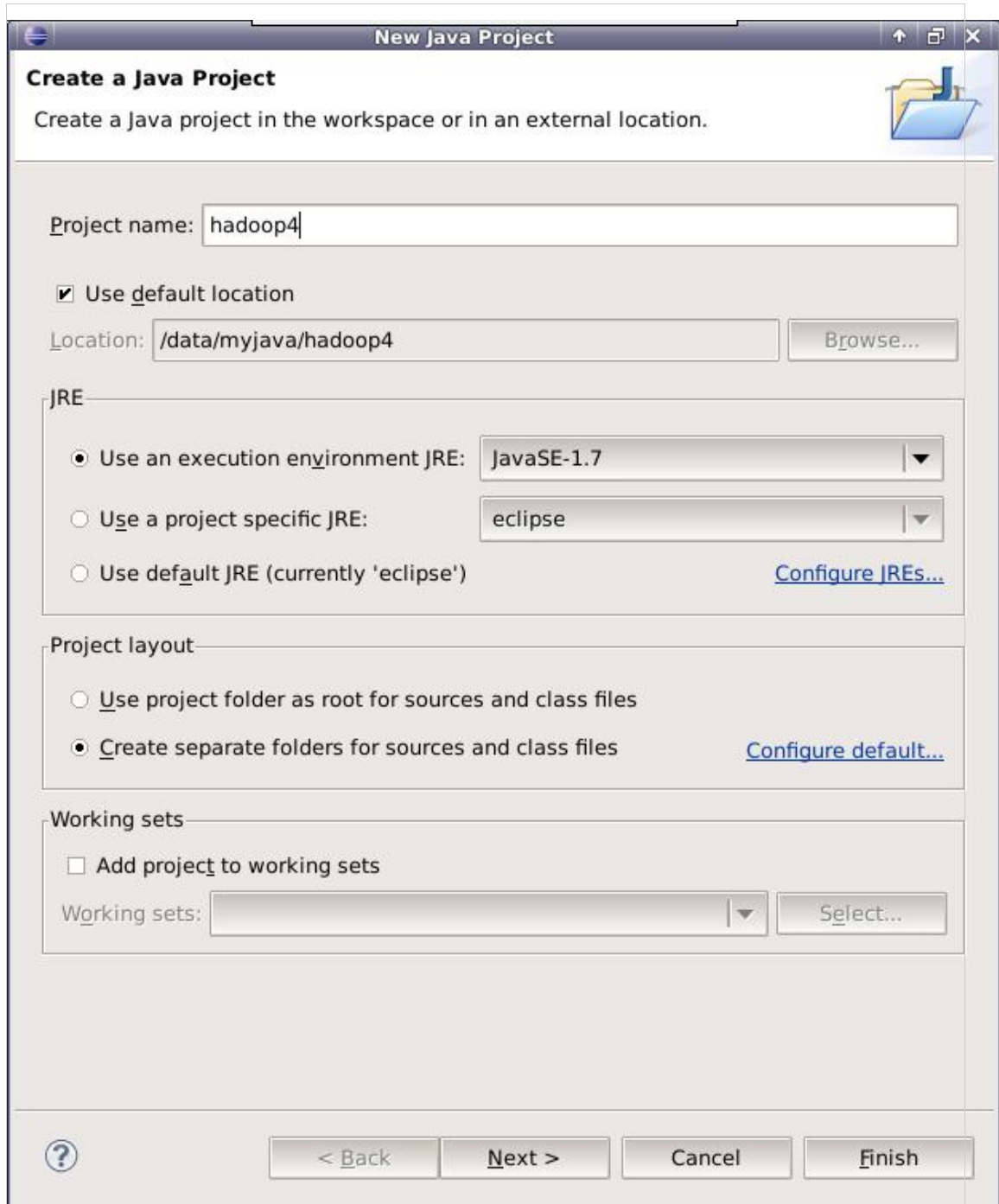
2. 切换到/data/hadoop4 目录，用 wget 命令，从 <http://192.168.1.100:60000/allfiles/hadoop4/> 网址上下载依赖包 hadoop2lib.tar.gz，并解压到当前目录。

view plain copy

1. `cd /data/hadoop4`
2. `wget http://192.168.1.100:60000/allfiles/hadoop4/hadoop2lib.tar.gz`
3. `tar zxvf hadoop2lib.tar.gz`

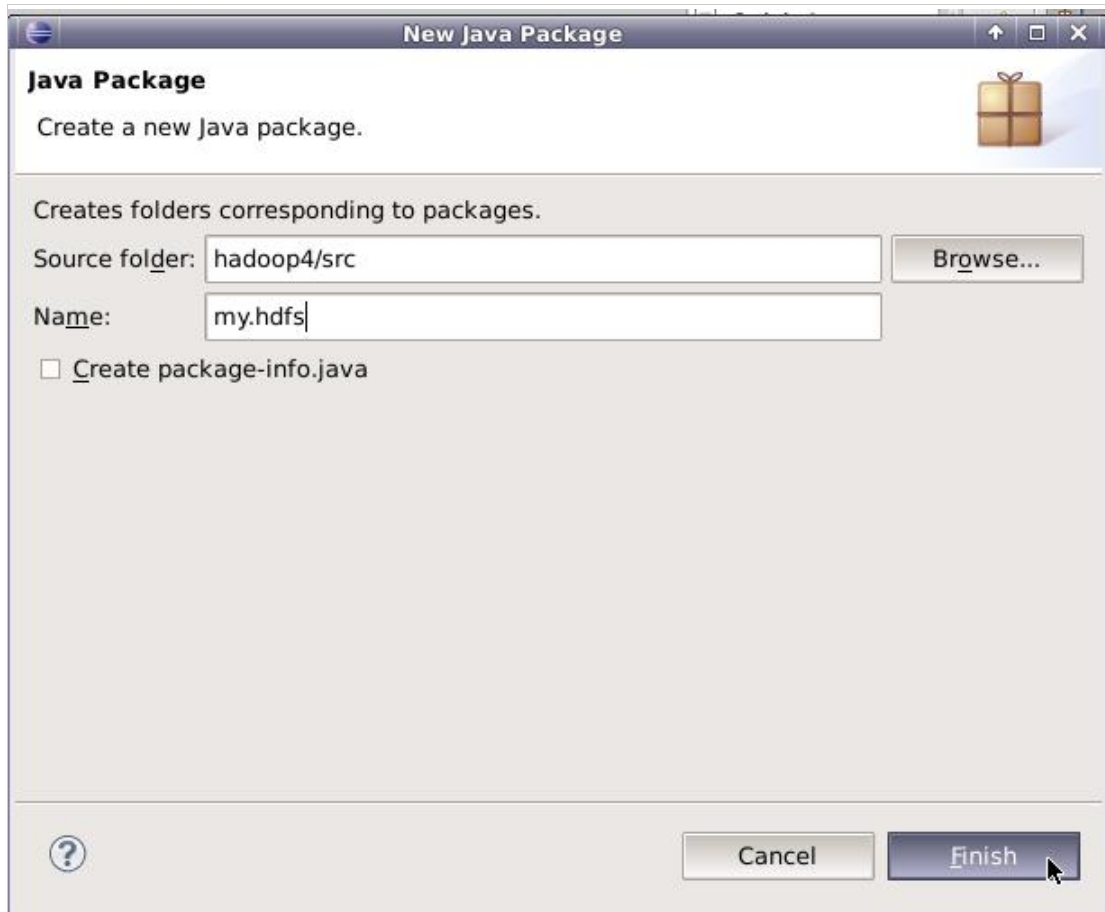
3. 新建 JAVA 项目，名为 hadoop4。



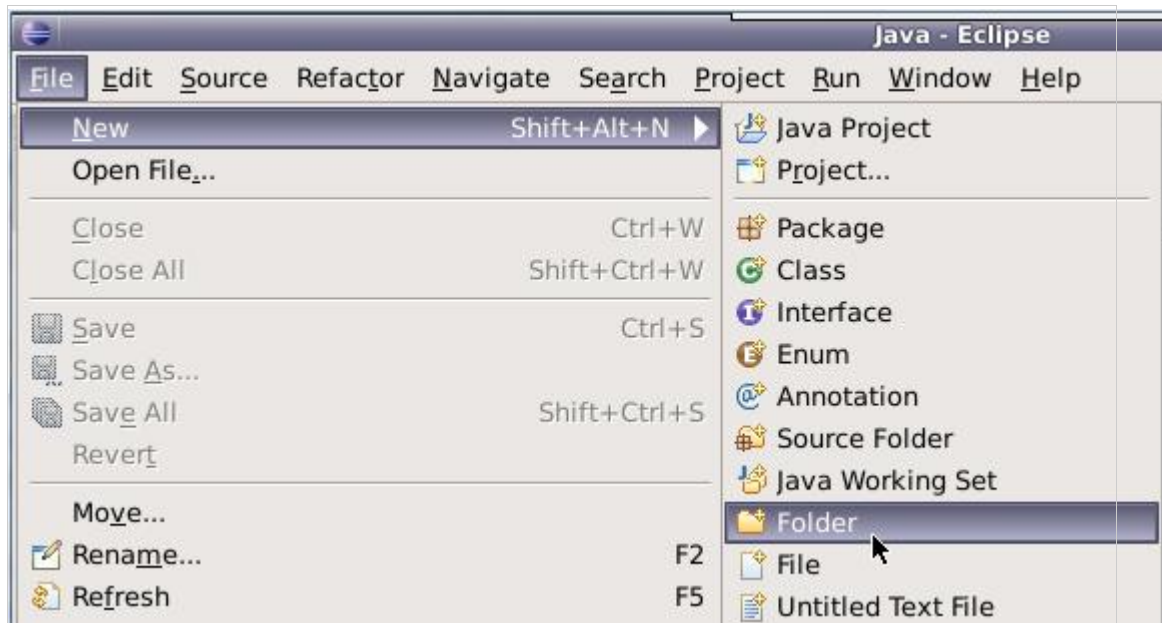


4.在 hadoop4 项目下新建包，名为 my.hdfs。

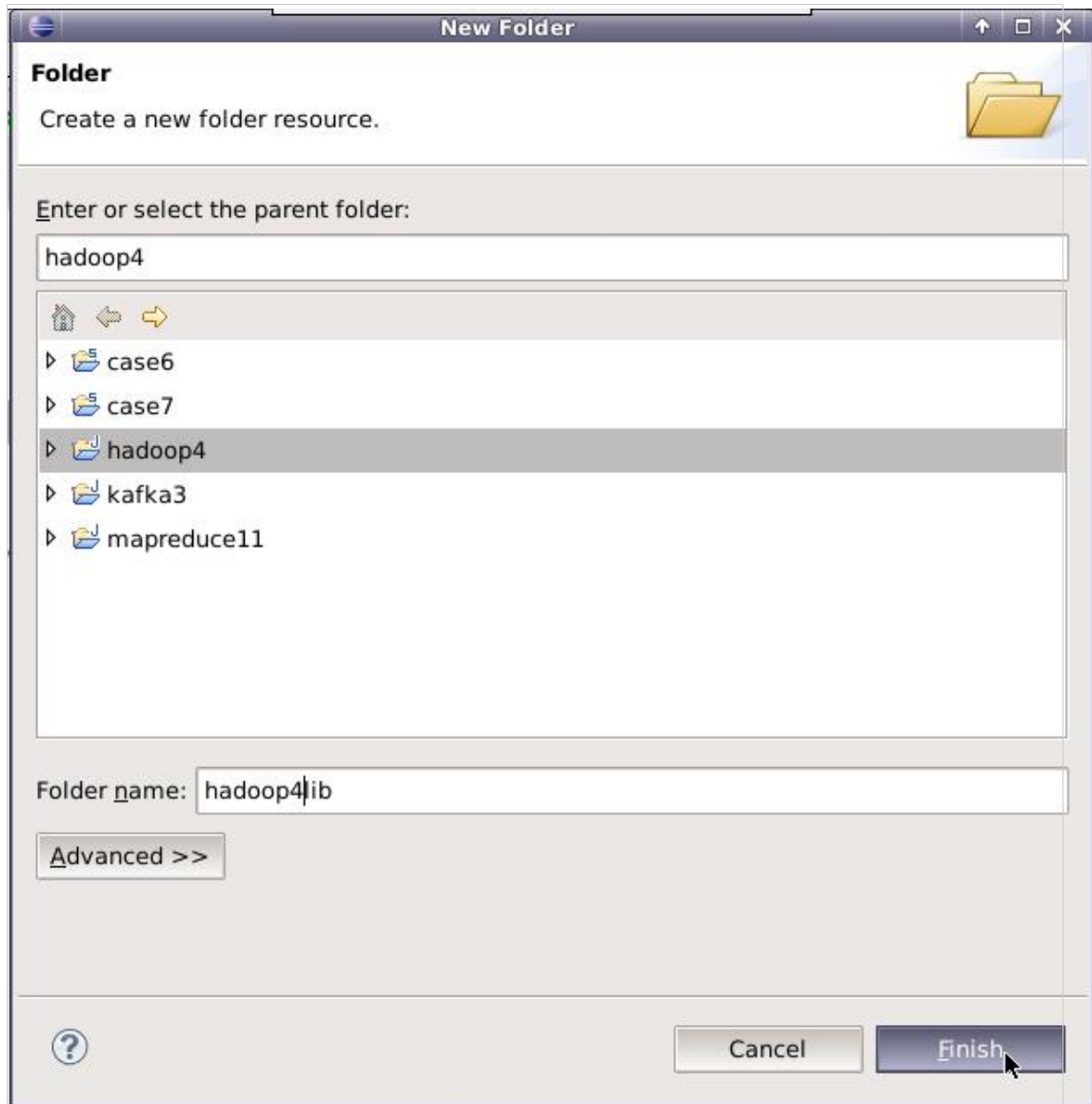




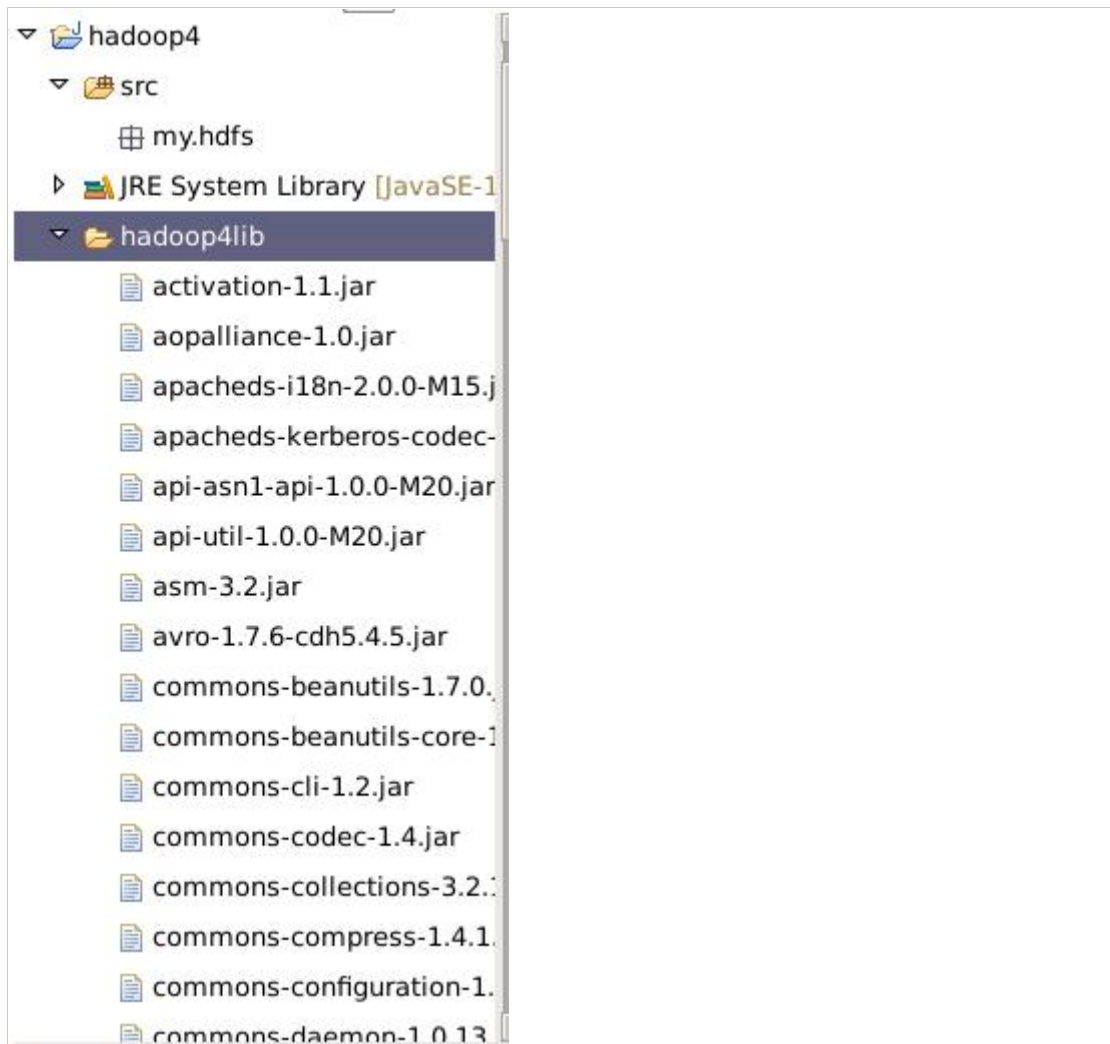
5.在 hadoop4 项目下创建目录，名为 hadoop4lib，用于存放项目所需依赖包。



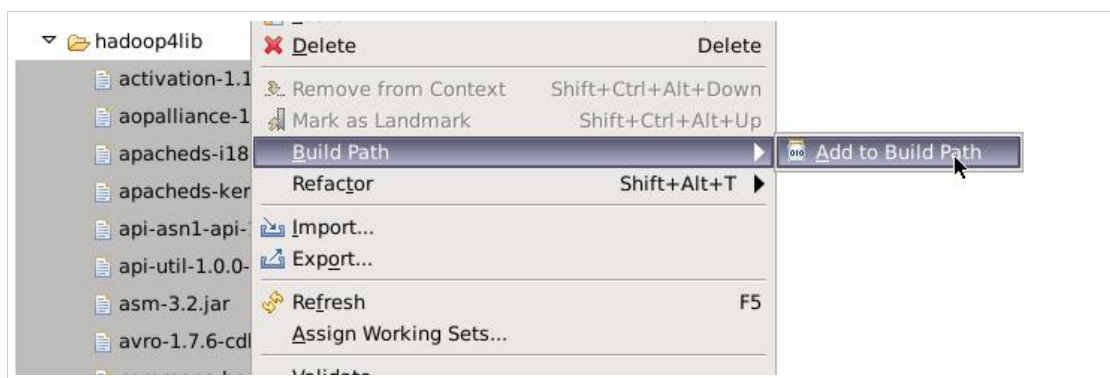




6.从/data/hadoop4/hadoop2lib 目录下拷贝所有 jar 包到项目下的 hadoop4lib 目录。



选中 hadoop4lib 里面的所有 jar 包，右键点击 BuildPath=>Add to Build Path 选项。



这样就将 jar 包加载到项目里面了，然后是进行 Java API 对 HDFS 的各种基本操作。

7.在 my.hdfs 包下，新建类 MakeDir，程序功能是在 HDFS 的根目录下，创建名为 hdfs-test 的目录。

```
view plain copy
1. package my.hdfs;
2. import java.io.IOException;
3. import java.net.URI;
4. import java.net.URISyntaxException;
5. import org.apache.hadoop.conf.Configuration;
```

```

6. import org.apache.hadoop.fs.FileSystem;
7. import org.apache.hadoop.fs.Path;
8. public class MakeDir {
9.     public static void main(String[] args) throws IOException, URISyn
taxException {
10.         Configuration conf = new Configuration();
11.
12.         String hdfsPath = "hdfs://localhost:9000";
13.         FileSystem hdfs = FileSystem.get(new URI(hdfsPath), conf);
14.
15.         String newDir = "/hdfstest";
16.
17.         boolean result = hdfs.mkdirs(new Path(newDir));
18.         if (result) {
19.             System.out.println("Success!");
20.         } else {
21.             System.out.println("Failed!");
22.         }
23.     }
24. }

```

在 Eclipse 里执行，然后在 HDFS 上查看实验结果。

[view plain copy](#)

```
1. hadoop fs -ls -R /
```

```

zhangyu@add7d6eb763c:~$ hadoop fs -ls -R /
drwxr-xr-x  - zhangyu supergroup          0 2019-06-04 06:55 /hdfstest
zhangyu@add7d6eb763c:~$

```

8. 在 my.hdfs 包下，新建类 TouchFile，程序功能是在 HDFS 的目录/hdfstest 下，创建名为 touchfile 的文件。

[view plain copy](#)

```

1. package my.hdfs;
2. import java.io.IOException;
3. import java.net.URI;
4. import java.net.URISyntaxException;
5. import org.apache.hadoop.conf.Configuration;
6. import org.apache.hadoop.fs.FSDataOutputStream;
7. import org.apache.hadoop.fs.FileSystem;
8. import org.apache.hadoop.fs.Path;
9. public class TouchFile {
10.     public static void main(String[] args) throws IOException, URISyn
taxException {
11.         Configuration configuration = new Configuration();
12.
13.         String hdfsPath = "hdfs://localhost:9000";

```

```

14.         FileSystem hdfs = FileSystem.get(new URI(hdfsPath), configura
            tion);
15.
16.         String filePath = "/hdfstest/touchfile";
17.
18.         FSDataOutputStream create = hdfs.create(new Path(filePath));
19.
20.         System.out.println("Finish!");
21.     }
22. }

```

在 Eclipse 里执行，然后在 hdfs 上查看实验结果。

```

1. hadoop fs -ls -R /

zhangyu@add7d6eb763c:/data/hadoop4$ hadoop fs -ls -R /
drwxr-xr-x  - zhangyu supergroup          0 2019-06-04 07:24 /hdfstest
-rw-r--r--  3 zhangyu supergroup          0 2019-06-04 07:24 /hdfstest/touchfile
zhangyu@add7d6eb763c:/data/hadoop4$

```

9.在/data/hadoop4 下使用 vim 打开 sample\_data 文件，

```

1. cd /data/hadoop4
2. vim sample_data

```

向 sample\_data 文件中写入 hello world。(使用 vim 编辑时，需输入 a，开启输入模式)

```

1. hello world

```

在 my.hdfs 包下，创建类 CopyFromLocalFile.class，程序功能是将本地 linux 操作系统上的文件 /data/hadoop4/sample\_data，上传到 HDFS 文件系统的/hdfstest 目录下。

```

1. package my.hdfs;
2. import java.io.IOException;
3. import java.net.URI;
4. import java.net.URISyntaxException;
5. import org.apache.hadoop.conf.Configuration;
6. import org.apache.hadoop.fs.FileSystem;
7. import org.apache.hadoop.fs.Path;
8. public class CopyFromLocalFile {
9.     public static void main(String[] args) throws IOException, URISyn
        taxException {
10.         Configuration conf = new Configuration();
11.         String hdfsPath = "hdfs://localhost:9000";
12.         FileSystem hdfs = FileSystem.get(new URI(hdfsPath), conf);
13.         String from_Linux = "/data/hadoop4/sample_data";
14.         String to_HDFS = "/hdfstest/";

```

```

15.         hdfs.copyFromLocalFile(new Path(from_Linux), new Path(to_HDFS)
16.         );
17.     }
18. }

```

在 Eclipse 里执行，然后在 HDFS 上查看实验结果。

```

1. hadoop fs -ls -R /

```

```

zhangyu@add7d6eb763c:/data/hadoop4$ hadoop fs -ls -R /
drwxr-xr-x  - zhangyu supergroup          0 2019-06-04 07:26 /hdfstest
-rw-r--r--  3 zhangyu supergroup        12 2019-06-04 07:26 /hdfstest/sample_data
-rw-r--r--  3 zhangyu supergroup          0 2019-06-04 07:24 /hdfstest/touchfile
zhangyu@add7d6eb763c:/data/hadoop4$

```

10.在/data/hadoop4/下创建目录 copytolocal。

```

1. mkdir /data/hadoop4/copytolocal

```

在 my.hdfs 包下，创建类 CopyToLocalFile.class，程序功能是将 HDFS 文件系统上的文件 /hdfstest/sample\_data，下载到本地/data/hadoop4/copytolocal。

```

1. package my.hdfs;
2. import java.io.IOException;
3. import java.net.URI;
4. import java.net.URISyntaxException;
5. import org.apache.hadoop.conf.Configuration;
6. import org.apache.hadoop.fs.FileSystem;
7. import org.apache.hadoop.fs.Path;
8. public class CopyToLocalFile {
9.     public static void main(String[] args) throws IOException, URISyn
taxException {
10.         Configuration conf = new Configuration();
11.
12.         String hdfsPath = "hdfs://localhost:9000";
13.         FileSystem hdfs = FileSystem.get(new URI(hdfsPath), conf);
14.
15.         String from_HDFS = "/hdfstest/sample_data";
16.         String to_Linux = "/data/hadoop4/copytolocal";
17.
18.         hdfs.copyToLocalFile(false, new Path(from_HDFS), new Path(to_
Linux));
19.
20.         System.out.println("Finish!");
21.     }
22. }

```

在 Eclipse 里执行，然后在 Linux 本地/data/hadoop4 上查看实验结果。

view plain copy

1. `cd /data/hadoop4/copytolocal`
2. `ls`

```
zhangyu@36484dcecf90:/data/hadoop4/copytolocal$ ls
sample_data
```

11.在 my.hdfs 包下，新建类 ListFiles，程序功能是列出 HDFS 文件系统/hdfstest 目录下，所有的文件，以及文件的权限、用户组、所属用户。

view plain copy

```
1. package my.hdfs;
2. import java.io.IOException;
3. import java.net.URI;
4. import org.apache.hadoop.conf.Configuration;
5. import org.apache.hadoop.fs.FileStatus;
6. import org.apache.hadoop.fs.FileSystem;
7. import org.apache.hadoop.fs.Path;
8. public class ListFiles {
9.     public static void main(String[] args) throws IOException {
10.         Configuration conf = new Configuration();
11.         String hdfspath = "hdfs://localhost:9000/";
12.         FileSystem hdfs = FileSystem.get(URI.create(hdfspath), conf);
13.
14.         String watchHDFS = "/hdfstest";
15.         FileStatus[] files = hdfs.listStatus(new Path(watchHDFS));
16.         for (FileStatus file : files) {
17.             System.out.println(file.getPermission() + " " + file.getOwner()
18.
19.
20.
21.
22.
23.
24.
25.
26.
27.
28.
29.
30.
31.
32.
33.
34.
35.
36.
37.
38.
39.
40.
41.
42.
43.
44.
45.
46.
47.
48.
49.
50.
51.
52.
53.
54.
55.
56.
57.
58.
59.
60.
61.
62.
63.
64.
65.
66.
67.
68.
69.
70.
71.
72.
73.
74.
75.
76.
77.
78.
79.
80.
81.
82.
83.
84.
85.
86.
87.
88.
89.
90.
91.
92.
93.
94.
95.
96.
97.
98.
99.
100.
101.
102.
103.
104.
105.
106.
107.
108.
109.
110.
111.
112.
113.
114.
115.
116.
117.
118.
119.
120.
121.
122.
123.
124.
125.
126.
127.
128.
129.
130.
131.
132.
133.
134.
135.
136.
137.
138.
139.
140.
141.
142.
143.
144.
145.
146.
147.
148.
149.
150.
151.
152.
153.
154.
155.
156.
157.
158.
159.
160.
161.
162.
163.
164.
165.
166.
167.
168.
169.
170.
171.
172.
173.
174.
175.
176.
177.
178.
179.
180.
181.
182.
183.
184.
185.
186.
187.
188.
189.
190.
191.
192.
193.
194.
195.
196.
197.
198.
199.
200.
201.
202.
203.
204.
205.
206.
207.
208.
209.
210.
211.
212.
213.
214.
215.
216.
217.
218.
219.
220.
221.
222.
223.
224.
225.
226.
227.
228.
229.
230.
231.
232.
233.
234.
235.
236.
237.
238.
239.
240.
241.
242.
243.
244.
245.
246.
247.
248.
249.
250.
251.
252.
253.
254.
255.
256.
257.
258.
259.
260.
261.
262.
263.
264.
265.
266.
267.
268.
269.
270.
271.
272.
273.
274.
275.
276.
277.
278.
279.
280.
281.
282.
283.
284.
285.
286.
287.
288.
289.
290.
291.
292.
293.
294.
295.
296.
297.
298.
299.
300.
301.
302.
303.
304.
305.
306.
307.
308.
309.
310.
311.
312.
313.
314.
315.
316.
317.
318.
319.
320.
321.
322.
323.
324.
325.
326.
327.
328.
329.
330.
331.
332.
333.
334.
335.
336.
337.
338.
339.
340.
341.
342.
343.
344.
345.
346.
347.
348.
349.
350.
351.
352.
353.
354.
355.
356.
357.
358.
359.
360.
361.
362.
363.
364.
365.
366.
367.
368.
369.
370.
371.
372.
373.
374.
375.
376.
377.
378.
379.
380.
381.
382.
383.
384.
385.
386.
387.
388.
389.
390.
391.
392.
393.
394.
395.
396.
397.
398.
399.
400.
401.
402.
403.
404.
405.
406.
407.
408.
409.
410.
411.
412.
413.
414.
415.
416.
417.
418.
419.
420.
421.
422.
423.
424.
425.
426.
427.
428.
429.
430.
431.
432.
433.
434.
435.
436.
437.
438.
439.
440.
441.
442.
443.
444.
445.
446.
447.
448.
449.
450.
451.
452.
453.
454.
455.
456.
457.
458.
459.
460.
461.
462.
463.
464.
465.
466.
467.
468.
469.
470.
471.
472.
473.
474.
475.
476.
477.
478.
479.
480.
481.
482.
483.
484.
485.
486.
487.
488.
489.
490.
491.
492.
493.
494.
495.
496.
497.
498.
499.
500.
501.
502.
503.
504.
505.
506.
507.
508.
509.
510.
511.
512.
513.
514.
515.
516.
517.
518.
519.
520.
521.
522.
523.
524.
525.
526.
527.
528.
529.
530.
531.
532.
533.
534.
535.
536.
537.
538.
539.
540.
541.
542.
543.
544.
545.
546.
547.
548.
549.
550.
551.
552.
553.
554.
555.
556.
557.
558.
559.
560.
561.
562.
563.
564.
565.
566.
567.
568.
569.
570.
571.
572.
573.
574.
575.
576.
577.
578.
579.
580.
581.
582.
583.
584.
585.
586.
587.
588.
589.
590.
591.
592.
593.
594.
595.
596.
597.
598.
599.
600.
601.
602.
603.
604.
605.
606.
607.
608.
609.
610.
611.
612.
613.
614.
615.
616.
617.
618.
619.
620.
621.
622.
623.
624.
625.
626.
627.
628.
629.
630.
631.
632.
633.
634.
635.
636.
637.
638.
639.
640.
641.
642.
643.
644.
645.
646.
647.
648.
649.
650.
651.
652.
653.
654.
655.
656.
657.
658.
659.
660.
661.
662.
663.
664.
665.
666.
667.
668.
669.
670.
671.
672.
673.
674.
675.
676.
677.
678.
679.
680.
681.
682.
683.
684.
685.
686.
687.
688.
689.
690.
691.
692.
693.
694.
695.
696.
697.
698.
699.
700.
701.
702.
703.
704.
705.
706.
707.
708.
709.
710.
711.
712.
713.
714.
715.
716.
717.
718.
719.
720.
721.
722.
723.
724.
725.
726.
727.
728.
729.
730.
731.
732.
733.
734.
735.
736.
737.
738.
739.
740.
741.
742.
743.
744.
745.
746.
747.
748.
749.
750.
751.
752.
753.
754.
755.
756.
757.
758.
759.
760.
761.
762.
763.
764.
765.
766.
767.
768.
769.
770.
771.
772.
773.
774.
775.
776.
777.
778.
779.
780.
781.
782.
783.
784.
785.
786.
787.
788.
789.
790.
791.
792.
793.
794.
795.
796.
797.
798.
799.
800.
801.
802.
803.
804.
805.
806.
807.
808.
809.
810.
811.
812.
813.
814.
815.
816.
817.
818.
819.
820.
821.
822.
823.
824.
825.
826.
827.
828.
829.
830.
831.
832.
833.
834.
835.
836.
837.
838.
839.
840.
841.
842.
843.
844.
845.
846.
847.
848.
849.
850.
851.
852.
853.
854.
855.
856.
857.
858.
859.
860.
861.
862.
863.
864.
865.
866.
867.
868.
869.
870.
871.
872.
873.
874.
875.
876.
877.
878.
879.
880.
881.
882.
883.
884.
885.
886.
887.
888.
889.
890.
891.
892.
893.
894.
895.
896.
897.
898.
899.
900.
901.
902.
903.
904.
905.
906.
907.
908.
909.
910.
911.
912.
913.
914.
915.
916.
917.
918.
919.
920.
921.
922.
923.
924.
925.
926.
927.
928.
929.
930.
931.
932.
933.
934.
935.
936.
937.
938.
939.
940.
941.
942.
943.
944.
945.
946.
947.
948.
949.
950.
951.
952.
953.
954.
955.
956.
957.
958.
959.
960.
961.
962.
963.
964.
965.
966.
967.
968.
969.
970.
971.
972.
973.
974.
975.
976.
977.
978.
979.
980.
981.
982.
983.
984.
985.
986.
987.
988.
989.
990.
991.
992.
993.
994.
995.
996.
997.
998.
999.
1000.
1001.
1002.
1003.
1004.
1005.
1006.
1007.
1008.
1009.
1010.
1011.
1012.
1013.
1014.
1015.
1016.
1017.
1018.
1019.
1020.
1021.
1022.
1023.
1024.
1025.
1026.
1027.
1028.
1029.
1030.
1031.
1032.
1033.
1034.
1035.
1036.
1037.
1038.
1039.
1040.
1041.
1042.
1043.
1044.
1045.
1046.
1047.
1048.
1049.
1050.
1051.
1052.
1053.
1054.
1055.
1056.
1057.
1058.
1059.
1060.
1061.
1062.
1063.
1064.
1065.
1066.
1067.
1068.
1069.
1070.
1071.
1072.
1073.
1074.
1075.
1076.
1077.
1078.
1079.
1080.
1081.
1082.
1083.
1084.
1085.
1086.
1087.
1088.
1089.
1090.
1091.
1092.
1093.
1094.
1095.
1096.
1097.
1098.
1099.
1100.
1101.
1102.
1103.
1104.
1105.
1106.
1107.
1108.
1109.
1110.
1111.
1112.
1113.
1114.
1115.
1116.
1117.
1118.
1119.
1120.
1121.
1122.
1123.
1124.
1125.
1126.
1127.
1128.
1129.
1130.
1131.
1132.
1133.
1134.
1135.
1136.
1137.
1138.
1139.
1140.
1141.
1142.
1143.
1144.
1145.
1146.
1147.
1148.
1149.
1150.
1151.
1152.
1153.
1154.
1155.
1156.
1157.
1158.
1159.
1160.
1161.
1162.
1163.
1164.
1165.
1166.
1167.
1168.
1169.
1170.
1171.
1172.
1173.
1174.
1175.
1176.
1177.
1178.
1179.
1180.
1181.
1182.
1183.
1184.
1185.
1186.
1187.
1188.
1189.
1190.
1191.
1192.
1193.
1194.
1195.
1196.
1197.
1198.
1199.
1200.
1201.
1202.
1203.
1204.
1205.
1206.
1207.
1208.
1209.
1210.
1211.
1212.
1213.
1214.
1215.
1216.
1217.
1218.
1219.
1220.
1221.
1222.
1223.
1224.
1225.
1226.
1227.
1228.
1229.
1230.
1231.
1232.
1233.
1234.
1235.
1236.
1237.
1238.
1239.
1240.
1241.
1242.
1243.
1244.
1245.
1246.
1247.
1248.
1249.
1250.
1251.
1252.
1253.
1254.
1255.
1256.
1257.
1258.
1259.
1260.
1261.
1262.
1263.
1264.
1265.
1266.
1267.
1268.
1269.
1270.
1271.
1272.
1273.
1274.
1275.
1276.
1277.
1278.
1279.
1280.
1281.
1282.
1283.
1284.
1285.
1286.
1287.
1288.
1289.
1290.
1291.
1292.
1293.
1294.
1295.
1296.
1297.
1298.
1299.
1300.
1301.
1302.
1303.
1304.
1305.
1306.
1307.
1308.
1309.
1310.
1311.
1312.
1313.
1314.
1315.
1316.
1317.
1318.
1319.
1320.
1321.
1322.
1323.
1324.
1325.
1326.
1327.
1328.
1329.
1330.
1331.
1332.
1333.
1334.
1335.
1336.
1337.
1338.
1339.
1340.
1341.
1342.
1343.
1344.
1345.
1346.
1347.
1348.
1349.
1350.
1351.
1352.
1353.
1354.
1355.
1356.
1357.
1358.
1359.
1360.
1361.
1362.
1363.
1364.
1365.
1366.
1367.
1368.
1369.
1370.
1371.
1372.
1373.
1374.
1375.
1376.
1377.
1378.
1379.
1380.
1381.
1382.
1383.
1384.
1385.
1386.
1387.
1388.
1389.
1390.
1391.
1392.
1393.
1394.
1395.
1396.
1397.
1398.
1399.
1400.
1401.
1402.
1403.
1404.
1405.
1406.
1407.
1408.
1409.
1410.
1411.
1412.
1413.
1414.
1415.
1416.
1417.
1418.
1419.
1420.
1421.
1422.
1423.
1424.
1425.
1426.
1427.
1428.
1429.
1430.
1431.
1432.
1433.
1434.
1435.
1436.
1437.
1438.
1439.
1440.
1441.
1442.
1443.
1444.
1445.
1446.
1447.
1448.
1449.
1450.
1451.
1452.
1453.
1454.
1455.
1456.
1457.
1458.
1459.
1460.
1461.
1462.
1463.
1464.
1465.
1466.
1467.
1468.
1469.
1470.
1471.
1472.
1473.
1474.
1475.
1476.
1477.
1478.
1479.
1480.
1481.
1482.
1483.
1484.
1485.
1486.
1487.
1488.
1489.
1490.
1491.
1492.
1493.
1494.
1495.
1496.
1497.
1498.
1499.
1500.
1501.
1502.
1503.
1504.
1505.
1506.
1507.
1508.
1509.
1510.
1511.
1512.
1513.
1514.
1515.
1516.
1517.
1518.
1519.
1520.
1521.
1522.
1523.
1524.
1525.
1526.
1527.
1528.
1529.
1530.
1531.
1532.
1533.
1534.
1535.
1536.
1537.
1538.
1539.
1540.
1541.
1542.
1543.
1544.
1545.
1546.
1547.
1548.
1549.
1550.
1551.
1552.
1553.
1554.
1555.
1556.
1557.
1558.
1559.
1560.
1561.
1562.
1563.
1564.
1565.
1566.
1567.
1568.
1569.
1570.
1571.
1572.
1573.
1574.
1575.
1576.
1577.
1578.
1579.
1580.
1581.
1582.
1583.
1584.
1585.
1586.
1587.
1588.
1589.
1590.
1591.
1592.
1593.
1594.
1595.
1596.
1597.
1598.
1599.
1600.
1601.
1602.
1603.
1604.
1605.
1606.
1607.
1608.
1609.
1610.
1611.
1612.
1613.
1614.
1615.
1616.
1617.
1618.
1619.
1620.
1621.
1622.
1623.
1624.
1625.
1626.
1627.
1628.
1629.
1630.
1631.
1632.
1633.
1634.
1635.
1636.
1637.
1638.
1639.
1640.
1641.
1642.
1643.
1644.
1645.
1646.
1647.
1648.
1649.
1650.
1651.
1652.
1653.
1654.
1655.
1656.
1657.
1658.
1659.
1660.
1661.
1662.
1663.
1664.
1665.
1666.
1667.
1668.
1669.
1670.
1671.
1672.
1673.
1674.
1675.
1676.
1677.
1678.
1679.
1680.
1681.
1682.
1683.
1684.
1685.
1686.
1687.
1688.
1689.
1690.
1691.
1692.
1693.
1694.
1695.
1696.
1697.
1698.
1699.
1700.
1701.
1702.
1703.
1704.
1705.
1706.
1707.
1708.
1709.
1710.
1711.
1712.
1713.
1714.
1715.
1716.
1717.
1718.
1719.
1720.
1721.
1722.
1723.
1724.
1725.
1726.
1727.
1728.
1729.
1730.
1731.
1732.
1733.
1734.
1735.
1736.
1737.
1738.
1739.
1740.
1741.
1742.
1743.
1744.
1745.
1746.
1747.
1748.
1749.
1750.
1751.
1752.
1753.
1754.
1755.
1756.
1757.
1758.
1759.
1760.
1761.
1762.
1763.
1764.
1765.
1766.
1767.
1768.
1769.
1770.
1771.
1772.
1773.
1774.
1775.
1776.
1777.
1778.
1779.
1780.
1781.
1782.
1783.
1784.
1785.
1786.
1787.
1788.
1789.
1790.
1791.
1792.
1793.
1794.
1795.
1796.
1797.
1798.
1799.
1800.
1801.
1802.
1803.
1804.
1805.
1806.
1807.
1808.
1809.
1810.
1811.
1812.
1813.
1814.
1815.
1816.
1817.
1818.
1819.
1820.
1821.
1822.
1823.
1824.
1825.
1826.
1827.
1828.
1829.
1830.
1831.
1832.
1833.
1834.
1835.
1836.
1837.
1838.
1839.
1840.
1841.
1842.
1843.
1844.
1845.
1846.
1847.
1848.
1849.
1850.
1851.
1852.
1853.
1854.
1855.
1856.
1857.
1858.
1859.
1860.
1861.
1862.
1863.
1864.
1865.
1866.
1867.
1868.
1869.
1870.
1871.
1872.
1873.
1874.
1875.
1876.
1877.
1878.
1879.
1880.
1881.
1882.
1883.
1884.
1885.
1886.
1887.
1888.
1889.
1890.
1891.
1892.
1893.
1894.
1895.
1896.
1897.
1898.
1899.
1900.
1901.
1902.
1903.
1904.
1905.
1906.
1907.
1908.
1909.
1910.
1911.
1912.
1913.
1914.
1915.
1916.
1917.
1918.
1919.
1920.
1921.
1922.
1923.
1924.
1925.
1926.
1927.
1928.
1929.
1930.
1931.
1932.
1933.
1934.
1935.
1936.
1937.
1938.
1939.
1940.
1941.
1942.
1943.
1944.
1945.
1946.
1947.
1948.
1949.
1950.
1951.
1952.
1953.
1954.
1955.
1956.
1957.
1958.
1959.
1960.
1961.
1962.
1963.
1964.
1965.
1966.
1967.
1968.
1969.
1970.
1971.
1972.
1973.
1974.
1975.
1976.
1977.
1978.
1979.
1980.
1981.
1982.
1983.
1984.
1985.
1986.
1987.
1988.
1989.
1990.
1991.
1992.
1993.
1994.
1995.
1996.
1997.
1998.
1999.
2000.
2001.
2002.
2003.
2004.
2005.
2006.
2007.
2008.
2009.
2010.
2011.
2012.
2013.
2014.
2015.
2016.
2017.
2018.
2019.
2020.
2021.
2022.
2023.
2024.
2025.
2026.
2027.
2028.
2029.
2030.
2031.
2032.
2033.
2034.
2035.
2036.
2037.
2038.
2039.
2040.
2041.
2042.
2043.
2044.
2045.
2046.
2047.
2048.
2049.
2050.
2051.
2052.
2053.
2054.
2055.
2056.
2057.
2058.
2059.
2060.
2061.
2062.
2063.
2064.
2065.
2066.
2067.
2068.
2069.
2070.
2071.
2072.
2073.
2074.
2075.
2076.
2077.
2078.
2079.
2080.
2081.
2082.
2083.
2084.
2085.
2086.
2087.
2088.
2089.
2090.
2091.
2092.
2093.
2094.
2095.
2096.
2097.
2098.
2099.
2100.
2101.
2102.
2103.
2104.
2105.
2106.
2107.
2108.
2109.
2110.
2111.
2112.
2113.
2114.
2115.
2116.
2117.
2118.
2119.
2120.
2121.
2122.
2123.
2124.
2125.
2126.
2127.
2128.
2129.
2130.
2131.
2132.
2133.
2134.
2135.
2136.
2137.
2138.
2139.
2140.
2141.
2142.
2143.
2144.
2145.
2146.
2147.
2148.
2149.
2150.
2151.
2152.
2153.
2154.
2155.
2156.
2157.
2158.
2159.
2160.
2161.
2162.
2163.
2164.
2165.
2166.
2167.
2168.
2169.
2170.
2171.
2172
```

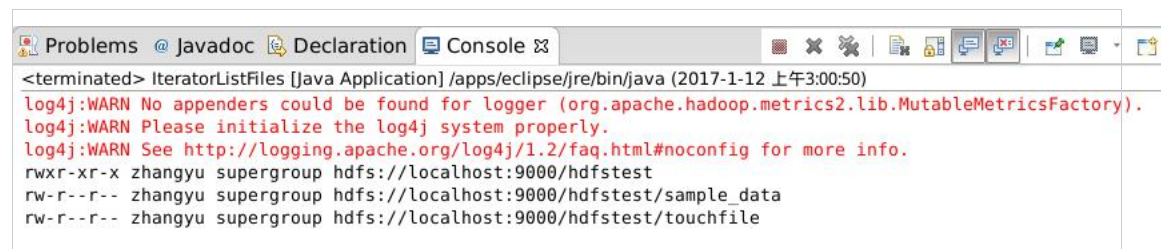
```

1. package my.hdfs;
2. import java.io.FileNotFoundException;
3. import java.io.IOException;
4. import java.net.URI;
5. import org.apache.hadoop.conf.Configuration;
6. import org.apache.hadoop.fs.FileStatus;
7. import org.apache.hadoop.fs.FileSystem;
8. import org.apache.hadoop.fs.Path;
9. public class IteratorListFiles {
10.     public static void main(String[] args) throws IOException {
11.         Configuration conf = new Configuration();
12.         String hdfspath = "hdfs://localhost:9000/";
13.         FileSystem hdfs = FileSystem.get(URI.create(hdfspath), conf);

14.         String watchHDFS = "/";
15.
16.         iteratorListFile(hdfs, new Path(watchHDFS));
17.     }
18.     public static void iteratorListFile(FileSystem hdfs, Path path)
19.         throws FileNotFoundException, IOException {
20.         FileStatus[] files = hdfs.listStatus(path);
21.         for (FileStatus file : files) {
22.             if (file.isDirectory()) {
23.                 System.out.println(file.getPermission() + " " + file.
24.                     getOwner()
25.                         + " " + file.getGroup() + " " + file.getPath()
26.                     );
27.                 iteratorListFile(hdfs, file.getPath());
28.             } else if (file.isFile()) {
29.                 System.out.println(file.getPermission() + " " + file.
30.                     getOwner()
31.                         + " " + file.getGroup() + " " + file.getPath()
32.                     );
33.             }
34.         }
35.     }
36. }

```

在 Eclipse 里执行，然后在 Eclipse 的控制界面 Console 上查看实验结果。



```

<terminated> IteratorListFiles [Java Application] /apps/eclipse/jre/bin/java (2017-1-12 上午3:00:50)
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
rwxr-xr-x zhangyu supergroup hdfs://localhost:9000/hdfs-test
rw-r--r-- zhangyu supergroup hdfs://localhost:9000/hdfs-test/sample_data
rw-r--r-- zhangyu supergroup hdfs://localhost:9000/hdfs-test/touchfile

```



### 13.文件是否存在、删除文件、重命名文件

了解 FileSystem 类下的方法，例如：判断文件是否存在、删除文件、重命名文件等。

FileSystem 的方法 exists、delete、rename。

#### exists

```
String hdfsPath = "hdfs://localhost:9000";
FileSystem hdfs = FileSystem.get(new URI(hdfsPath), conf);
```

|                                       |                                                                                          |
|---------------------------------------|------------------------------------------------------------------------------------------|
| hdfs.ex                               |                                                                                          |
| exists(Path f) : boolean - FileSystem | Check if exists.<br><b>Parameters:</b><br>f source file<br><b>Throws:</b><br>IOException |

#### rename

```
String hdfsPath = "hdfs://localhost:9000";
FileSystem hdfs = FileSystem.get(new URI(hdfsPath), conf);
```

|                                                                                                  |                                                                                                                                                                                                                                                    |
|--------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hdfs.ren                                                                                         |                                                                                                                                                                                                                                                    |
| rename(Path src, Path dst) : boolean - FileSystem                                                | Renames Path src to Path dst. Can take place on local or remote DFS.<br><b>Parameters:</b><br>src path to be renamed<br>dst new path after rename<br><b>Returns:</b><br>true if rename is successful<br><b>Throws:</b><br>IOException - on failure |
| renameSnapshot(Path path, String snapshotOldName, String snapshotNewName) : boolean - FileSystem |                                                                                                                                                                                                                                                    |

#### delete

```
String hdfsPath = "hdfs://localhost:9000";
FileSystem hdfs = FileSystem.get(new URI(hdfsPath), conf);
```

|                                                                    |                                                                                                                                                                                                                                                                                                                                         |
|--------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hdfs.delete                                                        |                                                                                                                                                                                                                                                                                                                                         |
| delete(Path f) : boolean - FileSystem                              | Delete a file.<br><b>Parameters:</b><br>f the path to delete.<br>recursive if path is a directory and set to true, the directory is deleted else throws an exception. In case of file the recursive can be set to either true or false.<br><b>Returns:</b><br>true if delete is successful else false.<br><b>Throws:</b><br>IOException |
| delete(Path f, boolean recursive) : boolean - FileSystem           |                                                                                                                                                                                                                                                                                                                                         |
| deleteOnExit(Path f) : boolean - FileSystem                        |                                                                                                                                                                                                                                                                                                                                         |
| deleteSnapshot(Path path, String snapshotName) : void - FileSystem |                                                                                                                                                                                                                                                                                                                                         |

用于查看文件块的信息。

14.在 my.hdfs 包下 ,新建类 LocateFile ,程序功能是查看 HDFS 文件系统上 ,文件/hdftest/sample\_data 的文件块信息。

[view plain copy](#)

```
1. package my.hdfs;
2. import java.io.IOException;
3. import java.net.URI;
4. import java.net.URISyntaxException;
5. import org.apache.hadoop.conf.Configuration;
6. import org.apache.hadoop.fs.BlockLocation;
7. import org.apache.hadoop.fs.FileStatus;
8. import org.apache.hadoop.fs.FileSystem;
9. import org.apache.hadoop.fs.Path;
```

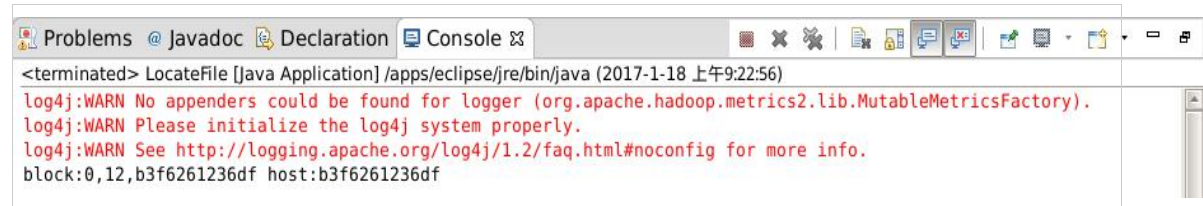


```

10. public class LocateFile {
11.     public static void main(String[] args) throws IOException, URISyn
        taxException {
12.         Configuration conf = new Configuration();
13.         String hdfsPath = "hdfs://localhost:9000";
14.         FileSystem hdfs = FileSystem.get(new URI(hdfsPath), conf);
15.
16.         Path file = new Path("/hdfstest/sample_data");
17.         FileStatus fileStatus = hdfs.getFileStatus(file);
18.
19.         BlockLocation[] location = hdfs.getFileBlockLocations(fileSta
            tus, 0, fileStatus.getLen());
20.         for (BlockLocation block : location) {
21.             String[] hosts = block.getHosts();
22.             for (String host : hosts) {
23.                 System.out.println("block:" + block + " host:" + host);
24.             }
25.         }
26.     }
27. }

```

在 Eclipse 里执行，然后在 Eclipse 的控制界面 Console 上查看实验结果。



```

<terminated> LocateFile [Java Application] /apps/eclipse/jre/bin/java (2017-1-18 上午9:22:56)
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
block:0,12,b3f6261236df host:b3f6261236df

```

15. 在 my.hdfs 包下，新建类 WriteFile，程序功能是在 HDFS 上，创建/hdfstest/writefile 文件并在文件中写入内容“hello world hello data!”。

```

1. package my.hdfs;
2. import java.io.IOException;
3. import java.net.URI;
4. import org.apache.hadoop.conf.Configuration;
5. import org.apache.hadoop.fs.FSDataOutputStream;
6. import org.apache.hadoop.fs.FileSystem;
7. import org.apache.hadoop.fs.Path;
8. public class WriteFile {
9.     public static void main(String[] args) throws IOException {
10.         Configuration conf = new Configuration();
11.
12.         String hdfsPath = "hdfs://localhost:9000";

```

```

13.         FileSystem hdfs = FileSystem.get(URI.create(hdfsPath), conf);
14.
15.         String filePath = "/hdfstest/writefile";
16.
17.         FSDataOutputStream create = hdfs.create(new Path(filePath));
18.
19.         System.out.println("Step 1 Finish!");
20.
21.         String sayHi = "hello world hello data!";
22.         byte[] buff = sayHi.getBytes();
23.         create.write(buff, 0, buff.length);
24.         create.close();
25.         System.out.println("Step 2 Finish!");
26.     }
27. }

```

在 Eclipse 里执行，然后在 HDFS 上查看实验结果。

```

view plain copy
1. hadoop fs -ls -R /hdfstest
2. hadoop fs -cat /hdfstest/writefile

```

```

zhangyu@add7d6eb763c:/data/hadoop4$ hadoop fs -ls -R /hdfstest
-rw-r--r--  3 zhangyu supergroup      12 2019-06-04 07:26 /hdfstest/sample_data
-rw-r--r--  3 zhangyu supergroup       0 2019-06-04 07:24 /hdfstest/touchfile
-rw-r--r--  3 zhangyu supergroup      23 2019-06-04 07:29 /hdfstest/writefile
zhangyu@add7d6eb763c:/data/hadoop4$ hadoop fs -cat /hdfstest/writefile
hello world hello data!zhangyu@add7d6eb763c:/data/hadoop4$

```

16.首先切换到/data/hadoop4 目录下，将该目录下的所有文件删除（此时要求/data/hadoop4 中必须全是文件，不能有目录）。

```

view plain copy
1. cd /data/hadoop4
2. rm -r /data/hadoop4/*

```

然后在该目录下新建两文件，分别命名为 file1 ,file2。

```

view plain copy
1. touch file1
2. touch file2

```

向 file1 和 file2 中，分别输入内容如下

```

view plain copy
1. echo "hello file1" > file1
2. echo "hello file2" > file2

```

在 my.hdfs 包下，新建类 PutMerge，程序功能是将 Linux 本地文件夹/data/hadoop4/下的所有文件，上传到 HDFS 上并合并成一个文件/hdfstest/mergefile。

view plain copy

```

1. package my.hdfs;
2. import java.io.IOException;
3. import java.net.URI;
4. import java.net.URISyntaxException;
5. import org.apache.hadoop.conf.Configuration;
6. import org.apache.hadoop.fs.FSDataInputStream;
7. import org.apache.hadoop.fs.FSDataOutputStream;
8. import org.apache.hadoop.fs.FileStatus;
9. import org.apache.hadoop.fs.FileSystem;
10. import org.apache.hadoop.fs.Path;
11. public class PutMerge {
12.     public static void main(String[] args) throws IOException, URISyn
        taxException {
13.         Configuration conf = new Configuration();
14.         String hdfsPath = "hdfs://localhost:9000";
15.         FileSystem hdfs = FileSystem.get(new URI(hdfsPath), conf);
16.         FileSystem local = FileSystem.getLocal(conf);
17.         String from_LinuxDir = "/data/hadoop4/";
18.         String to_HDFS = "/hdfstest/mergefile";
19.         FileStatus[] inputFiles = local.listStatus(new Path(from_Linu
            xDir));
20.         FSDataOutputStream out = hdfs.create(new Path(to_HDFS));
21.
22.         for (FileStatus file : inputFiles) {
23.             FSDataInputStream in = local.open(file.getPath());
24.             byte[] buffer = new byte[256];
25.             int bytesRead = 0;
26.             while ( (bytesRead = in.read(buffer) ) > 0) {
27.                 out.write(buffer, 0, bytesRead);
28.             }
29.             in.close();
30.         }
31.         System.out.println("Finish!");
32.     }
33. }

```

在 Eclipse 里执行，然后在 HDFS 上查看实验结果。

[view plain copy](#)

```
1. hadoop fs -ls /hdfstest
```

```

zhangyu@bc17474dbdd1:/data/hadoop4$ hadoop fs -ls /hdfstest
Found 4 items
-rw-r--r--  3 zhangyu supergroup      242 2017-01-12 03:42 /hdfstest/mergefile
-rw-r--r--  3 zhangyu supergroup     121 2017-01-12 02:46 /hdfstest/sample_data
-rw-r--r--  3 zhangyu supergroup      0 2017-01-12 02:37 /hdfstest/touchfile
-rw-r--r--  3 zhangyu supergroup      23 2017-01-12 03:08 /hdfstest/writefile
zhangyu@bc17474dbdd1:/data/hadoop4$

```

## 实验五：HBase API 操作

### 一、任务目标

- 1.了解 HBase 语言的基本语法
- 2.了解 HBase 开发的原理
- 3.了解 HBase Java API 的使用

### 二、相关知识

HBase 与 Hadoop 一样，都是用 Java 编写的，所以 HBase 对 Java 支持是必须的，HBase Java API 核心类介绍如下：

#### 1.HBaseConfiguration 类

HBaseConfiguration 是每一个 HBase Client 都会使用到的对象，它代表 HBase 配置信息，有两种构造方式：

- ①public HBaseConfiguration()
- ②public HBaseConfiguration(final Configuration c)

#### 2.创建表

创建表通过 HBaseAdmin 对象操作。HBaseAdmin 负责 META 表信息的处理。

HBaseAdmin 提供了 createTable 方法。

```
public void createTable(HTableDescriptor desc)
```

HTableDescriptor 表示表的 Schema，提供的常用方法有以下两个：

- ①setMaxFileSize：指定最大的 Region 大小。
- ②setMemStoreFlushSize：指定 MemStore Flush 到 HDFS 的文件大小。

#### 3.删除表

删除表也是通过 HBaseAdmin 来操作，删除表之前首先要 disable 表。这是一个非常耗时的操作，所以不建议频繁删除表。

disable Table 和 deleteTable 分别用来执行 disable 和 delete 操作。

#### 4.插入数据

HTable 通过 put 方法插入数据。可以传递单个 put 对象或 List put 对象分别实现单条插入和批量插入。

- ①public void put(final Put put) throws IOException
- ②public void put(final List<Put> puts) throws IOException

Put 提供三种构造方式。

- ①public Put (byte [] row)
- ②public Put (byte [] row,RowLock rowLock)
- ③public Put(Put putToCopy)

#### 5.查询数据

查询分为单条随机查询和批量查询。单条查询通过 Row Key 在 Table 中查询某一行的数据，HTable 提供了 get 方法完成单条查询。批量查询通过制定一段 Row Key 的范围来查询，HTable 提供了 getScanner 方法完成批量查询。

### 三、系统环境

Linux Ubuntu 16.04

jdk-7u75-linux-x64

hbase-1.0.0-cdh5.4.5

hadoop-2.6.0-cdh5.4.5

hadoop-2.6.0-eclipse-cdh5.4.5.jar

eclipse-java-juno-SR2-linux-gtk-x86\_64

### 四、任务内容

使用 Java API 对 HBase 表的基本操作，主要包含以下四个部分：

- 1.编写 Java 代码，实现创建 HBase 表的操作。
- 2.编写 Java 代码，实现删除 HBase 表的操作。
- 3.编写 Java 代码，实现写数据到 HBase 表中的操作。
- 4.编写 Java 代码，实现读取 HBase 表中数据的操作。

### 五、任务步骤

1.首先检查 Hadoop 相关进程，是否已经启动。若未启动，切换到/apps/hadoop/sbin 目录下，启动 Hadoop。

view plain copy

1. jps
2. cd /apps/hadoop/sbin
3. ./start-all.sh

当 Hadoop 相关进程启动后，进入 HBase 的 bin 目录下，启动 HBase 服务。

view plain copy

1. cd /apps/hbase/bin/
2. ./start-hbase.sh

2.切换到/data/hbase2 目录下，如不存在需提前创建 hbase2 文件夹。

view plain copy

1. mkdir -p /data/hbase2
2. cd /data/hbase2

3.使用 wget 命令，下载 http://192.168.1.100:60000/allfiles/hbase2 中的文件。

view plain copy

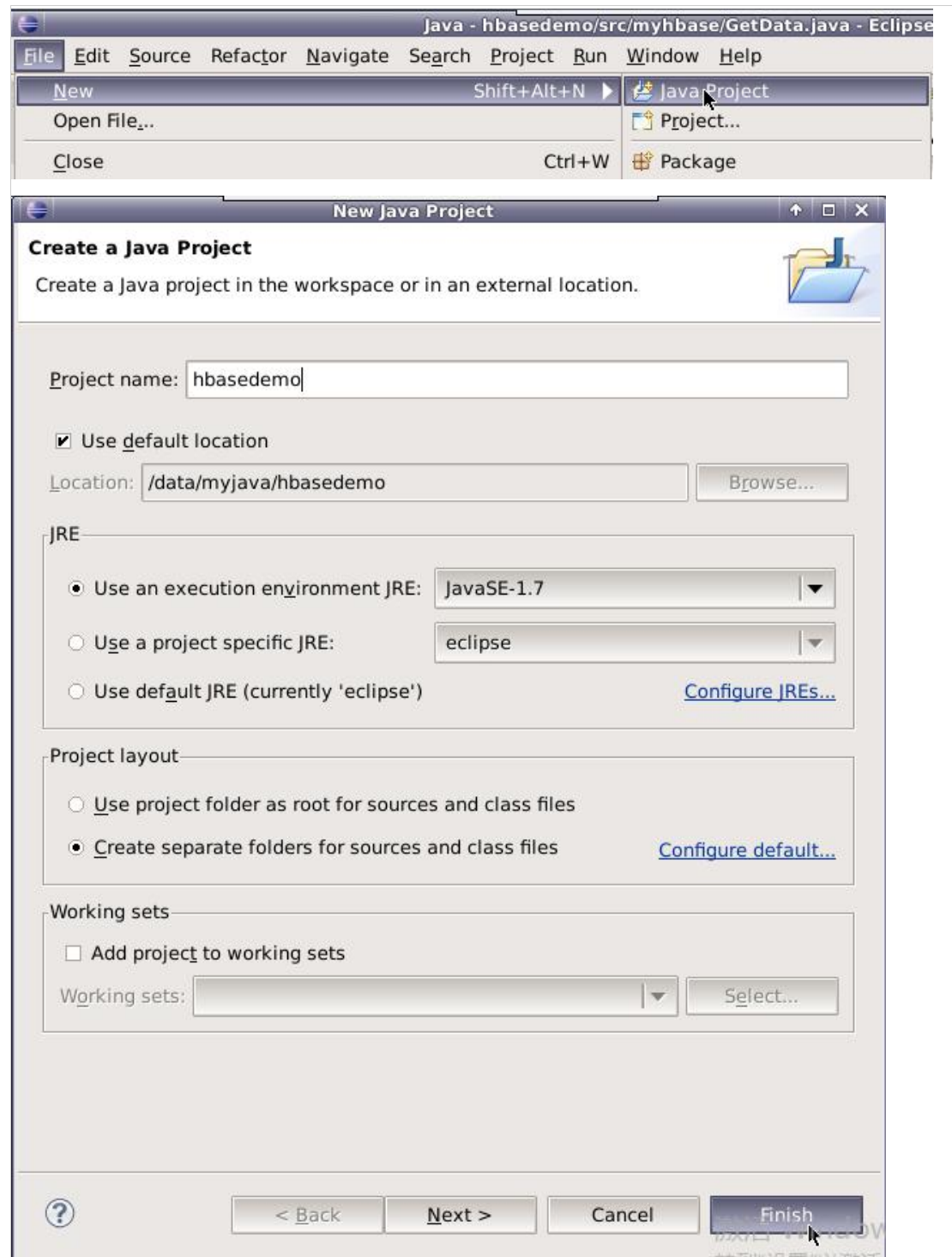
1. wget http://192.168.1.100:60000/allfiles/hbase2/hbasedemolib.tar.gz
2. wget http://192.168.1.100:60000/allfiles/hbase2/hbasedemo.tar.gz
3. wget http://192.168.1.100:60000/allfiles/hbase2/CreateMyTable.java
4. wget http://192.168.1.100:60000/allfiles/hbase2/DeleteMyTable.java
5. wget http://192.168.1.100:60000/allfiles/hbase2/GetData.java

6. `wget http://192.168.1.100:60000/allfiles/hbase2/PutData.java`

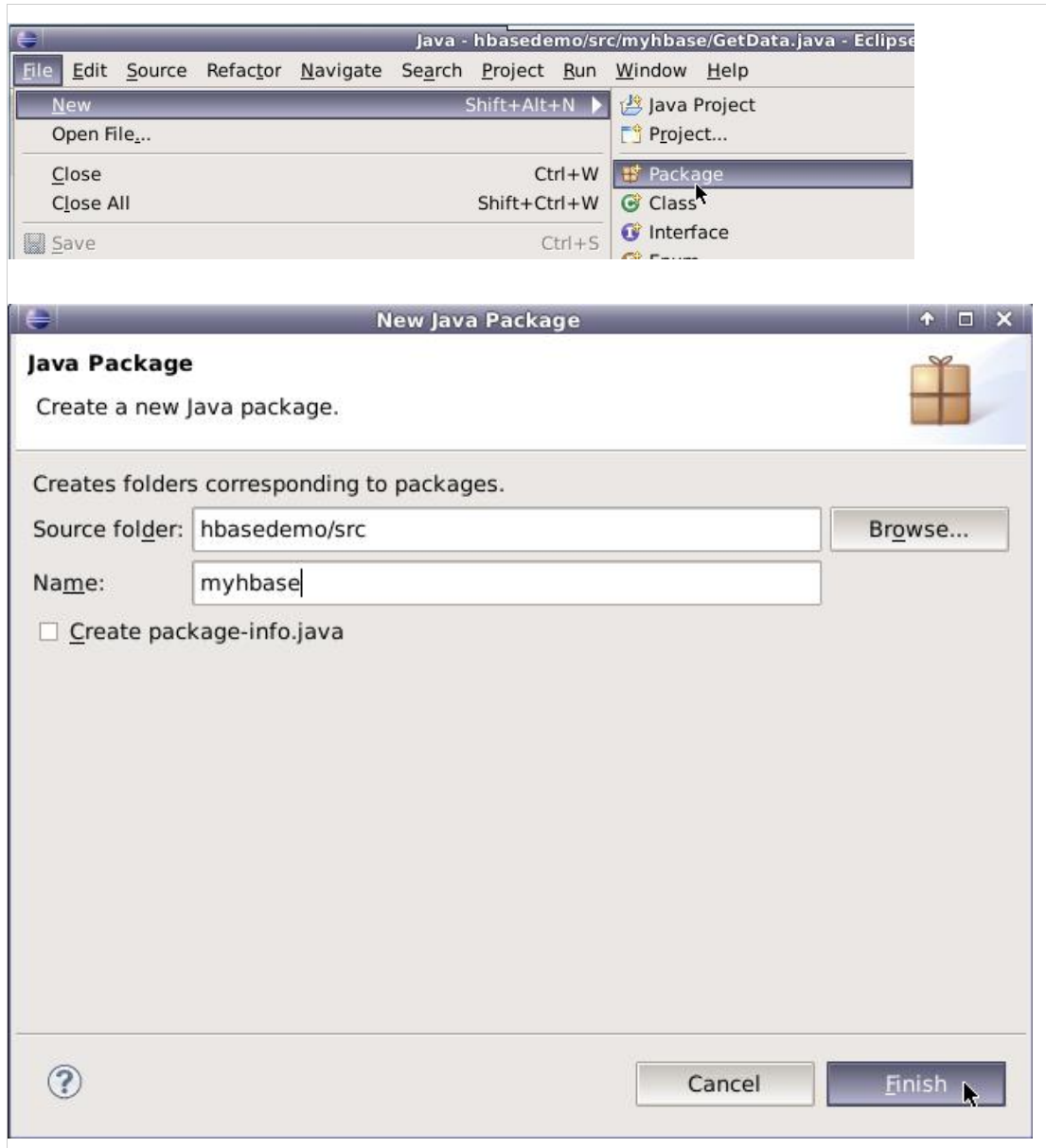
4. 解压/data/hbase2 中的 hbasedemolib.tar.gz 包到/data/hbase2 中。

1. `tar zxvf hbasedemolib.tar.gz`

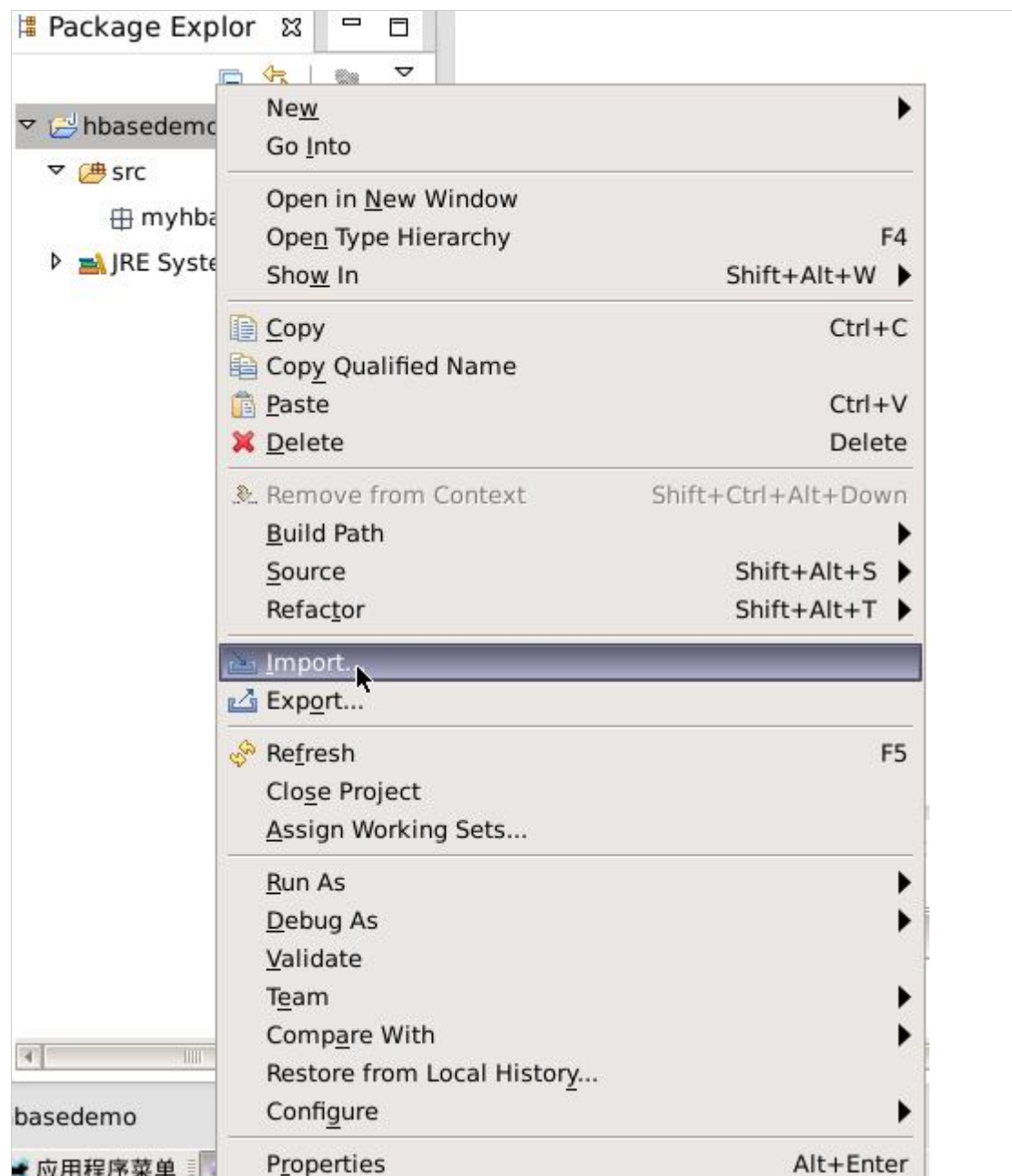
5. 打开 Eclipse，创建 java 项目，名为 hbasedemo。



在 hbasedemo 项目下，创建包，包名为 myhbase。

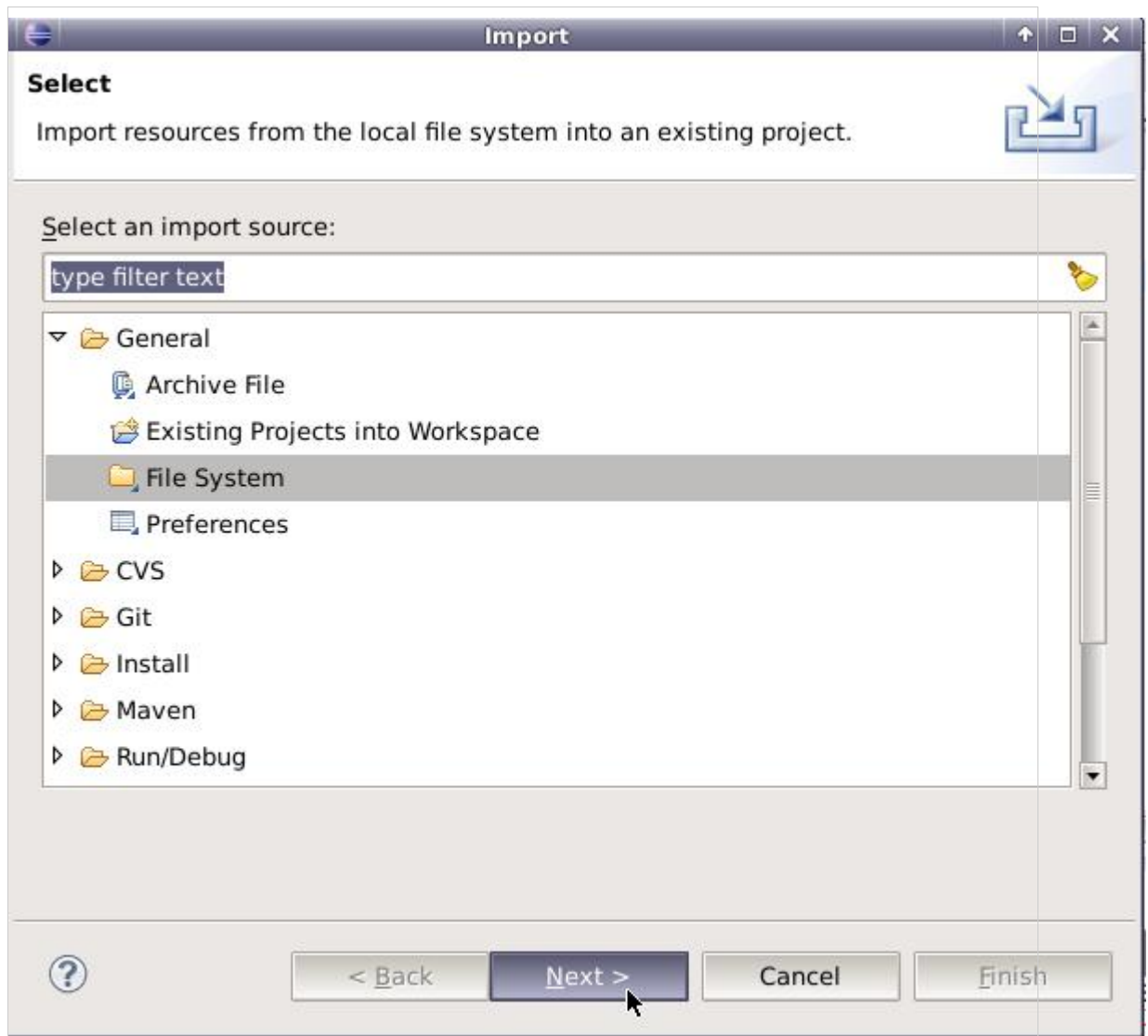


添加项目依赖的jar包，右击 hbasedemo，选择 import。

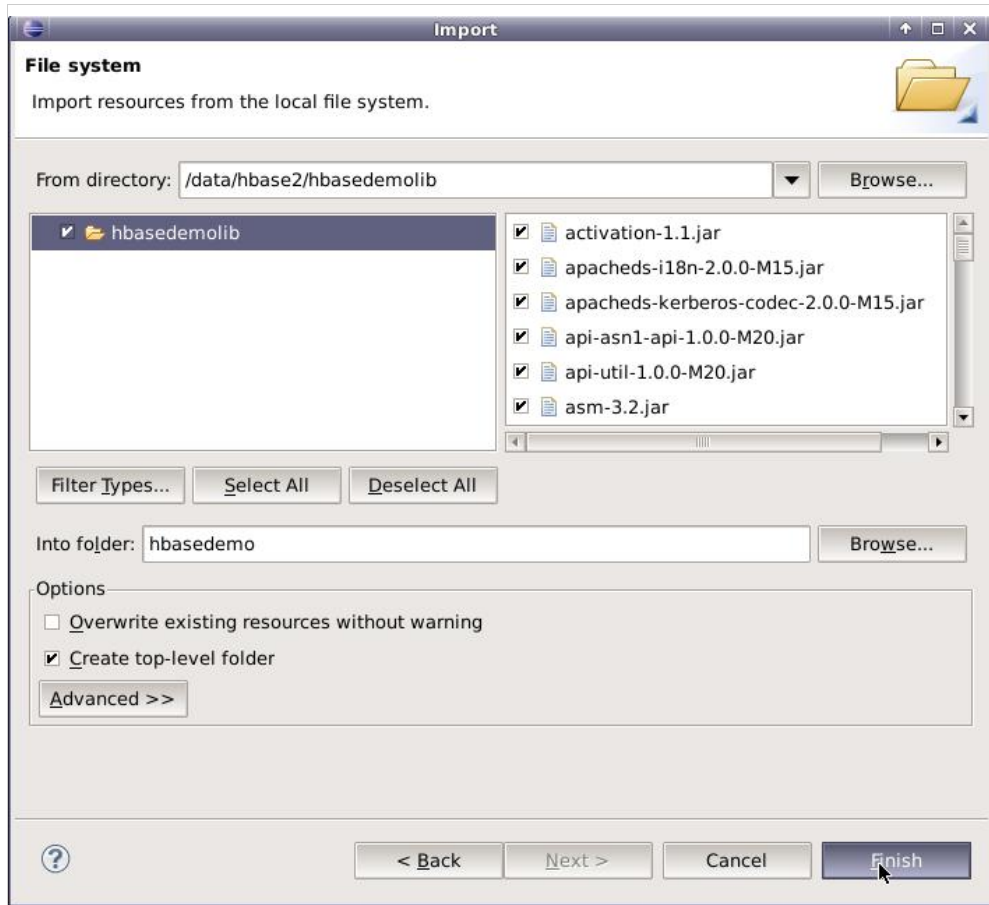


进入下面界面，选择 General 中的 File System，点击 Next。

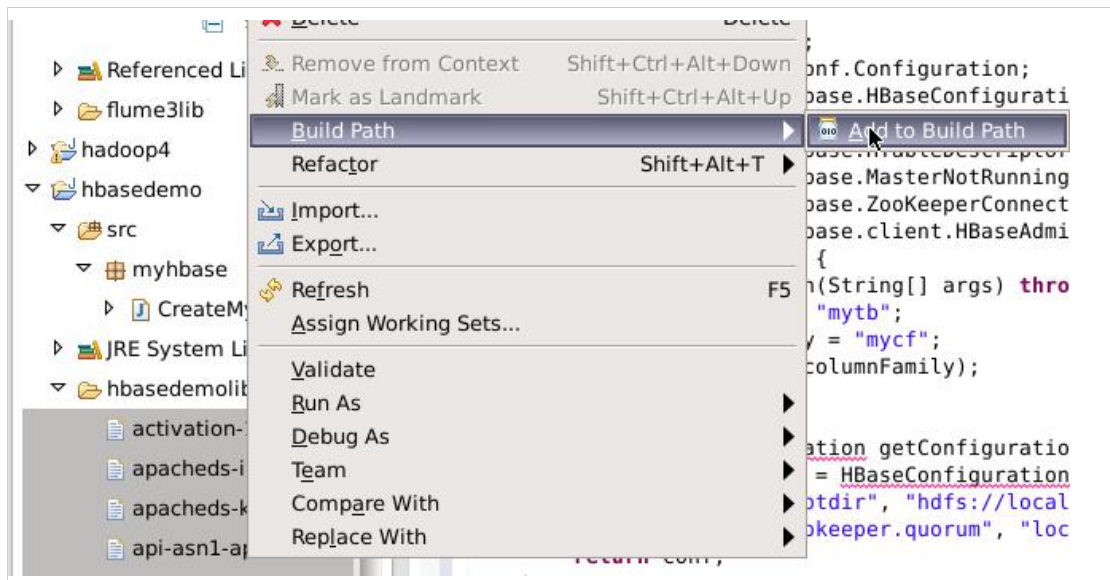




进入以下界面，选择/data/hbase2 中的 hbasedemolib 文件夹，并勾选 Create top-level folder，点击 Finish。



然后，选中 hbasedemolib 里面的所有文件，单击右键 Build Path=>Add to Build Path 选项，就将所有 jar 包加载到项目里面了。



## 6.创建表的 API

创建类，名为 CreateMyTable，功能为在 HBase 中创建名为 mytb，列族为 mycf 的表。

**New Java Class**

**Java Class**  
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

---

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

#### 程序代码

[view plain copy](#)

```

1. package myhbase;
2. import java.io.IOException;
3. import org.apache.hadoop.conf.Configuration;
4. import org.apache.hadoop.hbase.HBaseConfiguration;
5. import org.apache.hadoop.hbase.HColumnDescriptor;
6. import org.apache.hadoop.hbase.HTableDescriptor;
7. import org.apache.hadoop.hbase.MasterNotRunningException;
8. import org.apache.hadoop.hbase.ZooKeeperConnectionException;
9. import org.apache.hadoop.hbase.client.HBaseAdmin;
10. public class CreateMyTable {
11.     public static void main(String[] args) throws MasterNotRunningExc
        eption, ZooKeeperConnectionException, IOException {
12.         String tableName = "mytb";

```

```

13.     String columnFamily = "mycf";
14.     create(tableName, columnFamily);
15. }
16.
17.     public static Configuration getConfiguration() {
18.         Configuration conf = HBaseConfiguration.create();
19.         conf.set("hbase.rootdir", "hdfs://localhost:9000/hbase");
20.         conf.set("hbase.zookeeper.quorum", "localhost");
21.         return conf;
22.     }
23.     public static void create(String tableName, String columnFamily)
24.         throws MasterNotRunningException, ZooKeeperConnectionExce
25.             ption,
26.                 IOException {
27.         HBaseAdmin hBaseAdmin = new HBaseAdmin(getConfiguration());
28.         if (hBaseAdmin.tableExists(tableName)) {
29.             System.err.println("Table exists!");
30.         } else {
31.             HTableDescriptor tableDesc = new HTableDescriptor(tableName);
32.             tableDesc.addFamily(new HColumnDescriptor(columnFamily));
33.             hBaseAdmin.createTable(tableDesc);
34.             System.err.println("Create Table SUCCESS!");
35.         }
36. }

```

在 Eclipse 中执行程序代码 ,在 CreateMyTable 类文件中 ,单击右键=>Run As=>Run on Hadoop 选项 ,将任务提交到 Hadoop 中。



然后查看 HBase 中新创建的 mytb 表 ,先启动 hbase shell 命令行模式。

[view plain copy](#)

## 1. hbase shell

执行 list ,列出当前 HBase 中的表。

[view plain copy](#)

## 1. list

```
hbase(main):001:0> list
TABLE
mytb
1 row(s) in 0.2150 seconds

=> ["mytb"]
hbase(main):002:0>
```

执行以下命令，查看创建的表结构。

[view plain copy](#)

### 1. describe 'mytb'

```
hbase(main):006:0> describe 'mytb'
Table mytb is ENABLED
mytb
COLUMN FAMILIES DESCRIPTION
{NAME => 'mycf', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
1 row(s) in 0.0330 seconds

hbase(main):007:0>
```

## 7.删除表的 API

创建类，命名为 DeleteMyTable，功能为将 HBase 中表 mytb 删除。

**New Java Class**

Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

---

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

#### 程序代码

[view plain copy](#)

```

1. package myhbase;
2. import java.io.IOException;
3. import org.apache.hadoop.conf.Configuration;
4. import org.apache.hadoop.hbase.HBaseConfiguration;
5. import org.apache.hadoop.hbase.client.HBaseAdmin;
6. public class DeleteMyTable {
7.     public static void main(String[] args) throws IOException {
8.         String tableName = "mytb";
9.         delete(tableName);
10.    }
11.
12.    public static Configuration getConfiguration() {
13.        Configuration conf = HBaseConfiguration.create();
14.        conf.set("hbase.rootdir", "hdfs://localhost:9000/hbase");
15.        conf.set("hbase.zookeeper.quorum", "localhost");

```

```

16.         return conf;
17.     }
18.
19.     public static void delete(String tableName) throws IOException {
20.         HBaseAdmin hAdmin = new HBaseAdmin(getConfiguration());
21.         if(hAdmin.tableExists(tableName)){
22.             try {
23.                 hAdmin.disableTable(tableName);
24.                 hAdmin.deleteTable(tableName);
25.                 System.err.println("Delete table Success");
26.             } catch (IOException e) {
27.                 System.err.println("Delete table Failed ");
28.             }
29.         }else{
30.             System.err.println("table not exists");
31.         }
32.     }
33. }

```

在 DeleteMyTable 类文件中，单击右键=>Run As=>Run on Hadoop 选项，将任务提交到 Hadoop 中。



在 Eclipse 中执行完成，然后在 hbase 中查看结果，查看 mytb 表是否被删除。

[view plain copy](#)

## 1. list

```

hbase(main):004:0> list
TABLE
0 row(s) in 0.0110 seconds

=> []
hbase(main):005:0>

```

## 8.写入数据的 API

某电商网站，后台有买家信息表 buyer，每注册一名新用户网站后台会产生一条日志，并写入 HBase 中。数据格式为：用户 ID ( buyer\_id )，注册日期 ( reg\_date )，注册 IP ( reg\_ip )，卖家状态 ( buyer\_status，0 表示冻结，1 表示正常)，以 “\t” 分割，数据内容如下：

[view plain copy](#)

1. 用户 ID    注册日期    注册 IP    卖家状态
2. 20385,2010-05-04,124.64.242.30,1
3. 20386,2010-05-05,117.136.0.172,1
4. 20387,2010-05-06 ,114.94.44.230,1

将数据以 buyer\_id 作为行键写入到 HBase 的 buyer 表中,插入之前,需确保 buyer 表已存在,若不存在,提前创建。

[view plain copy](#)

1. create 'buyer','reg\_date'

```
hbase(main):008:0> create 'buyer','reg_date'
0 row(s) in 0.4740 seconds

=> Hbase::Table - buyer
```

创建类, 名为 PutData, 功能为将以上三条数据写入到 buyer 表中。

The screenshot shows the 'New Java Class' dialog box. The 'Name' field is filled with 'PutData'. The 'Package' field is filled with 'myhbase'. The 'Source folder' field is filled with 'hbasedemo/src'. The 'Superclass' field is filled with 'java.lang.Object'. The 'Enclosing type' checkbox is unchecked. The 'Modifiers' section has 'public' selected. The 'Interfaces' section is empty. The 'Which method stubs would you like to create?' section has 'Inherited abstract methods' selected. The 'Do you want to add comments?' section has 'Generate comments' selected. The 'Finish' button is highlighted with a mouse cursor.

程序代码如下：

[view plain copy](#)

```
1. package myhbase;
2. import java.io.IOException;
3. import org.apache.hadoop.conf.Configuration;
4. import org.apache.hadoop.hbase.HBaseConfiguration;
5. import org.apache.hadoop.hbase.MasterNotRunningException;
6. import org.apache.hadoop.hbase.ZooKeeperConnectionException;
7. import org.apache.hadoop.hbase.client.HTable;
```



```

8. import org.apache.hadoop.hbase.client.Put;
9. import org.apache.hadoop.hbase.util.Bytes;
10. public class PutData {
11.     public static void main(String[] args) throws MasterNotRunningExc
        eption,
12.         ZooKeeperConnectionException, IOException {
13.         String tableName = "buyer";
14.         String columnFamily = "reg_date";
15.         put(tableName, "20385", columnFamily, "2010-05-04:reg_ip", "1
            24.64.242.30");
16.         put(tableName, "20385", columnFamily, "2010-05-04:buyer_statu
            s", "1");
17.
18.         put(tableName, "20386", columnFamily, "2010-05-05:reg_ip", "1
            17.136.0.172");
19.         put(tableName, "20386", columnFamily, "2010-05-05:buyer_statu
            s", "1");
20.
21.         put(tableName, "20387", columnFamily, "2010-05-06:reg_ip", "1
            14.94.44.230");
22.         put(tableName, "20387", columnFamily, "2010-05-06:buyer_statu
            s", "1");
23.
24.     }
25.     public static Configuration getConfiguration() {
26.         Configuration conf = HBaseConfiguration.create();
27.         conf.set("hbase.rootdir", "hdfs://localhost:9000/hbase");
28.         conf.set("hbase.zookeeper.quorum", "localhost");
29.         return conf;
30.     }
31.     public static void put(String tableName, String row, String colum
        nFamily,
32.         String column, String data) throws IOException {
33.         HTable table = new HTable(getConfiguration(), tableName);
34.         Put put = new Put(Bytes.toBytes(row));
35.         put.add(Bytes.toBytes(columnFamily),
36.             Bytes.toBytes(column),
37.             Bytes.toBytes(data));
38.         table.put(put);
39.         System.err.println("SUCCESS");
40.     }
41. }

```

在 Eclipse 中执行程序代码，在 PutData 类文件中，右键并点击=>Run As=>Run on Hadoop 选项，将任务提交到 Hadoop 中。



执行完成后，进入 HBase 中查看 buyer 表结果。

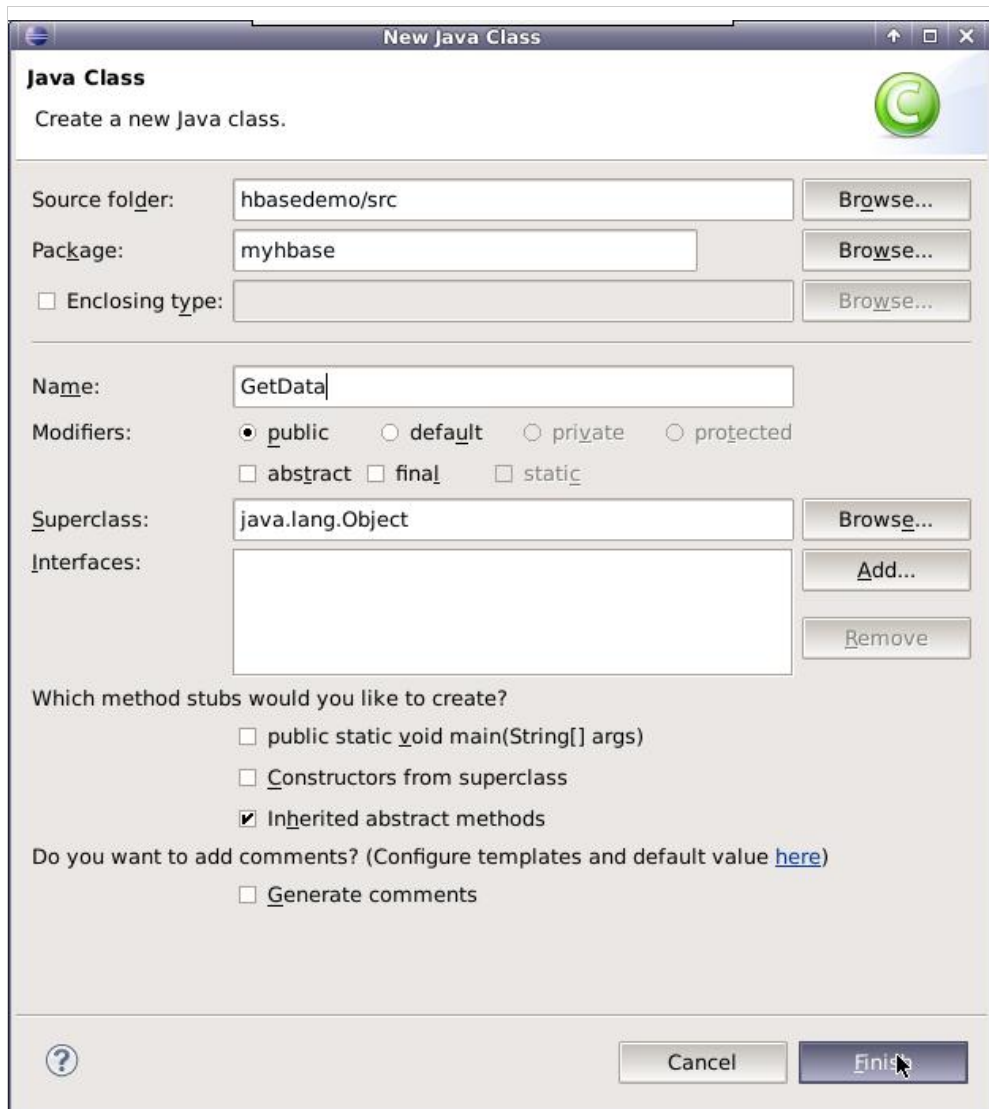
[view plain copy](#)

## 1. scan 'buyer'

```
hbase(main):004:0> scan 'buyer'
ROW                                COLUMN+CELL
20385                             column=reg_date:2010-05-04:buyer_status, timestamp=1484615495845, value=1
20385                             column=reg_date:2010-05-04:reg_ip, timestamp=1484615495801, value=124.64.242.30
20386                             column=reg_date:2010-05-05:buyer_status, timestamp=1484615495897, value=1
20386                             column=reg_date:2010-05-05:reg_ip, timestamp=1484615495873, value=117.136.0.172
20387                             column=reg_date:2010-05-06:buyer_status, timestamp=1484615495944, value=1
20387                             column=reg_date:2010-05-06:reg_ip, timestamp=1484615495920, value=114.94.44.230
3 row(s) in 0.0820 seconds
```

## 9.查询数据的 API

创建类 GetData，功能为查询 HBase 的 buyer 表中 rowkey 为 20386 的数据。



程序代码如下：

[view plain copy](#)

```
1. package myhbase;
2. import java.io.IOException;
3. import org.apache.hadoop.conf.Configuration;
4. import org.apache.hadoop.hbase.HBaseConfiguration;
5. import org.apache.hadoop.hbase.client.Get;
6. import org.apache.hadoop.hbase.client.HTable;
7. import org.apache.hadoop.hbase.client.Result;
8. import org.apache.hadoop.hbase.util.Bytes;
9. public class GetData {
10.     public static void main(String[] args) throws IOException {
11.         String tableName = "buyer";
12.         get(tableName, "20386");
13.     }
14.     public static Configuration getConfiguration() {
15.         Configuration conf = HBaseConfiguration.create();
16.         conf.set("hbase.rootdir", "hdfs://localhost:9000/hbase");
17.         conf.set("hbase.zookeeper.quorum", "localhost");
18.         return conf;
19.     }
20.     public static void get(String tableName, String rowkey) throws IO
        Exception {
21.         HTable table = new HTable(getConfiguration(), tableName);
22.         Get get = new Get(Bytes.toBytes(rowkey));
23.         Result result = table.get(get);
24.         byte[] value1 = result.getValue("reg_date".getBytes(), "2010-
            05-05:reg_ip".getBytes());
25.         byte[] value2 = result.getValue("reg_date".getBytes(), "2010-
            05-05:buyer_status".getBytes());
26.         System.err.println("line1:SUCCESS");
27.         System.err.println("line2:"
28.             + new String(value1) + "\t"
29.             + new String(value2));
30.     }
31. }
```

在 Eclipse 中执行程序代码，在 GetData 类文件中，单击右键=>Run As=>Run on Hadoop 选项，将任务提交到 Hadoop 中。



执行完成后，可以在 Eclipse 中的 console 界面查看到执行结果为：

```
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
line1:SUCCESS
line2:117.136.0.172    1
```

## 实验六：MAPREDUCE 实例——WordCount

### 一、任务目标

- 1.准确理解 Mapreduce 的设计原理
- 2.熟练掌握 WordCount 程序代码编写
- 3.学会自己编写 WordCount 程序进行词频统计

### 二、相关知识

MapReduce 采用的是“分而治之”的思想，把对大规模数据集的操作，分发给一个主节点管理下的各个从节点共同完成，然后通过整合各个节点的中间结果，得到最终结果。简单来说，MapReduce 就是“任务的分解与结果的汇总”。

#### 1.MapReduce 的工作原理

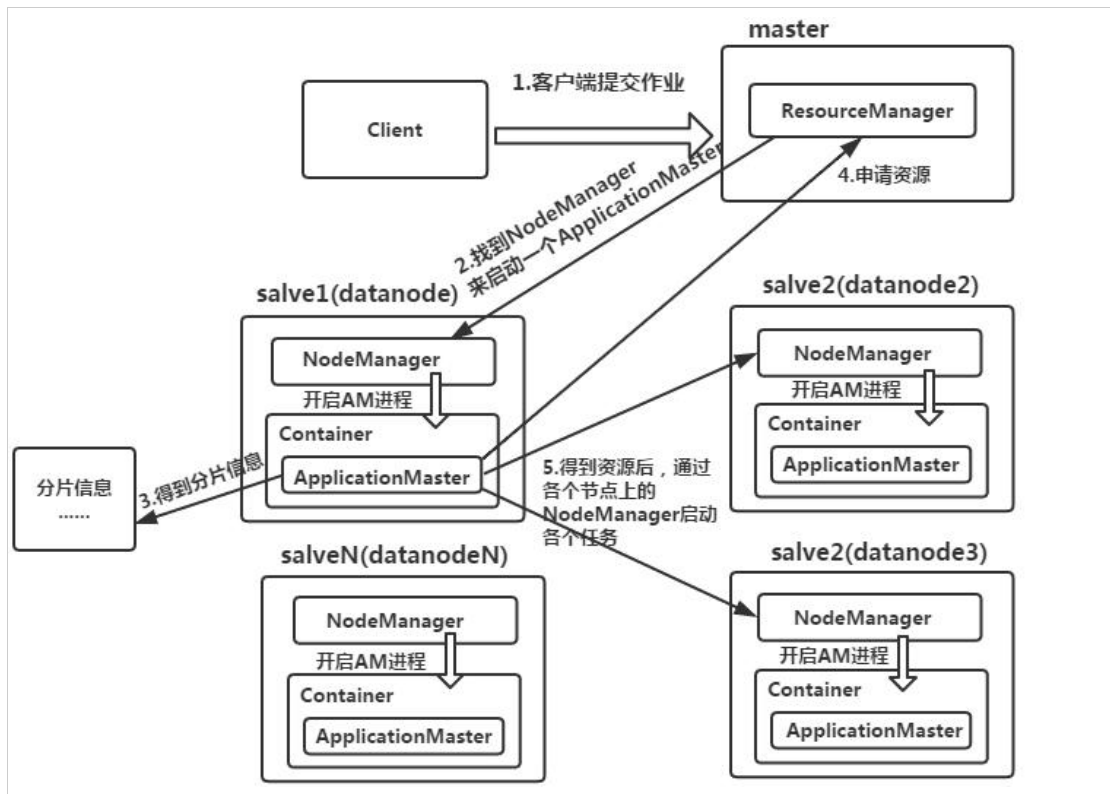
在分布式计算中，MapReduce 框架负责处理了并行编程里分布式存储、工作调度，负载均衡、容错处理以及网络通信等复杂问题，现在我们把处理过程高度抽象为 Map 与 Reduce 两个部分来进行阐述，其中 Map 部分负责把任务分解成多个子任务，Reduce 部分负责把分解后多个子任务的处理结果汇总起来，具体设计思路如下。

(1) Map 过程需要继承 org.apache.hadoop.mapreduce 包中 Mapper 类，并重写其 map 方法。通过在 map 方法中添加两句把 key 值和 value 值输出到控制台的代码，可以发现 map 方法中输入的 value 值存储的是文本文件中的一行(以回车符为行结束标记)，而输入的 key 值存储的是该行的首字母相对于文本文件的首地址的偏移量。然后用 StringTokenizer 类将每一行拆分成为一个个的字段，把截取需要的字段(本实验为买家 id 字段)设置为 key，并将其作为 map 方法的结果输出。

(2) Reduce 过程需要继承 org.apache.hadoop.mapreduce 包中 Reducer 类，并重写其 reduce 方法。Map 过程输出的 <key,value> 键值对先经过 shuffle 过程把 key 值相同的所有 value 值聚集起来形成 values，此时 values 是对应 key 字段的计数值所组成的列表，然后将 <key,values> 输入到 reduce 方法中，reduce 方法只要遍历 values 并求和，即可得到某个单词的总次数。

在 main()主函数中新建一个 Job 对象，由 Job 对象负责管理和运行 MapReduce 的一个计算任务，并通过 Job 的一些方法对任务的参数进行相关的设置。本实验是设置使用将继承 Mapper 的 doMapper 类完成 Map 过程中的处理和使用 doReducer 类完成 Reduce 过程中的处理。还设置了 Map 过程和 Reduce 过程的输出类型：key 的类型为 Text，value 的类型为 IntWritable。任务的输出和输入路径则由字符串指定，并由 FileInputFormat 和 FileOutputFormat 分别设定。完成相应任务的参数设定后，即可调用 job.waitForCompletion()方法执行任务，其余的工作都交由 MapReduce 框架处理。

#### 2.MapReduce 框架的作业运行流程



(1) ResourceManager：是 YARN 资源控制框架的中心模块，负责集群中所有资源的统一管理和分配。它接收来自 NM(NodeManager)的汇报，建立 AM，并将资源派送给 AM(ApplicationMaster)。

(2) NodeManager：简称 NM，NodeManager 是 ResourceManager 在每台机器上的代理，负责容器管理，并监控他们的资源使用情况（cpu、内存、磁盘及网络等），以及向 ResourceManager 提供这些资源使用报告。

(3) ApplicationMaster：以下简称 AM。YARN 中每个应用都会启动一个 AM，负责向 RM 申请资源，请求 NM 启动 Container，并告诉 Container 做什么事情。

(4) Container：资源容器。YARN 中所有的应用都是在 Container 之上运行的。AM 也是在 Container 上运行的，不过 AM 的 Container 是 RM 申请的。Container 是 YARN 中资源的抽象，它封装了某个节点上一定量的资源（CPU 和内存两类资源）。Container 由 ApplicationMaster 向 ResourceManager 申请的，由 ResouceManager 中的资源调度器异步分配给 ApplicationMaster。Container 的运行是由 ApplicationMaster 向资源所在的 NodeManager 发起的，Container 运行时需提供内部执行的任务命令（可以是任何命令，比如 java、Python、C++进程启动命令均可）以及该命令执行所需的环境变量和外部资源（比如词典文件、可执行文件、jar 包等）。

另外，一个应用程序所需的 Container 分为两大类，如下：

①运行 ApplicationMaster 的 Container：这是由 ResourceManager（向内部的资源调度器）申请和启动的，用户提交应用程序时，可指定唯一的 ApplicationMaster 所需的资源。

②运行各类任务的 Container：这是由 ApplicationMaster 向 ResourceManager 申请的，并为了 ApplicationMaster 与 NodeManager 通信以启动的。

以上两类 Container 可能在任意节点上，它们的位置通常而言是随机的，即 ApplicationMaster 可能与它管理的任务运行在一个节点上。

### 三、系统环境

Linux Ubuntu 16.0

jdk-7u75-linux-x64

hadoop-2.6.0-cdh5.4.5

hadoop-2.6.0-eclipse-cdh5.4.5.jar

eclipse-java-juno-SR2-linux-gtk-x86\_64

### 四、任务内容

现有某电商网站用户对商品的收藏数据，记录了用户收藏的商品 id 以及收藏日期，名为 buyer\_favorite1。

buyer\_favorite1 包含：买家 id，商品 id，收藏日期这三个字段，数据以 “\t” 分割，样本数据及格式如下：

|                                 |       |         |                     |
|---------------------------------|-------|---------|---------------------|
| <a href="#">view plain copy</a> |       |         |                     |
| 1.                              | 买家 id | 商品 id   | 收藏日期                |
| 2.                              | 10181 | 1000481 | 2010-04-04 16:54:31 |
| 3.                              | 20001 | 1001597 | 2010-04-07 15:07:52 |
| 4.                              | 20001 | 1001560 | 2010-04-07 15:08:27 |
| 5.                              | 20042 | 1001368 | 2010-04-08 08:20:30 |
| 6.                              | 20067 | 1002061 | 2010-04-08 16:45:33 |
| 7.                              | 20056 | 1003289 | 2010-04-12 10:50:55 |
| 8.                              | 20056 | 1003290 | 2010-04-12 11:57:35 |
| 9.                              | 20056 | 1003292 | 2010-04-12 12:05:29 |
| 10.                             | 20054 | 1002420 | 2010-04-14 15:24:12 |
| 11.                             | 20055 | 1001679 | 2010-04-14 19:46:04 |
| 12.                             | 20054 | 1010675 | 2010-04-14 15:23:53 |
| 13.                             | 20054 | 1002429 | 2010-04-14 17:52:45 |
| 14.                             | 20076 | 1002427 | 2010-04-14 19:35:39 |
| 15.                             | 20054 | 1003326 | 2010-04-20 12:54:44 |
| 16.                             | 20056 | 1002420 | 2010-04-15 11:24:49 |
| 17.                             | 20064 | 1002422 | 2010-04-15 11:35:54 |
| 18.                             | 20056 | 1003066 | 2010-04-15 11:43:01 |
| 19.                             | 20056 | 1003055 | 2010-04-15 11:43:06 |
| 20.                             | 20056 | 1010183 | 2010-04-15 11:45:24 |
| 21.                             | 20056 | 1002422 | 2010-04-15 11:45:49 |
| 22.                             | 20056 | 1003100 | 2010-04-15 11:45:54 |
| 23.                             | 20056 | 1003094 | 2010-04-15 11:45:57 |
| 24.                             | 20056 | 1003064 | 2010-04-15 11:46:04 |
| 25.                             | 20056 | 1010178 | 2010-04-15 16:15:20 |
| 26.                             | 20076 | 1003101 | 2010-04-15 16:37:27 |
| 27.                             | 20076 | 1003103 | 2010-04-15 16:37:05 |
| 28.                             | 20076 | 1003100 | 2010-04-15 16:37:18 |

```
29. 20076    1003066    2010-04-15 16:37:31
30. 20054    1003103    2010-04-15 16:40:14
31. 20054    1003100    2010-04-15 16:40:16
```

要求编写 MapReduce 程序，统计每个买家收藏商品数量。

统计结果数据如下：

```
view plain copy
1. 买家 id 商品数量
2. 10181    1
3. 20001    2
4. 20042    1
5. 20054    6
6. 20055    1
7. 20056    12
8. 20064    1
9. 20067    1
10. 20076   5
```

## 五、任务步骤

1. 切换目录到/apps/hadoop/sbin 下，启动 hadoop。

```
view plain copy
1. cd /apps/hadoop/sbin
2. ./start-all.sh
```

2. 在 linux 上，创建一个目录/data/mapreduce1。

```
view plain copy
1. mkdir -p /data/mapreduce1
```

3. 切换到 /data/mapreduce1 目录下，使用 wget 命令从网址 [http://192.168.1.100:60000/allfiles/mapreduce1/buyer\\_favorite1](http://192.168.1.100:60000/allfiles/mapreduce1/buyer_favorite1)，下载文本文件 buyer\_favorite1。

```
view plain copy
1. cd /data/mapreduce1
2. wget http://192.168.1.100:60000/allfiles/mapreduce1/buyer_favorite1
```

依然在/data/mapreduce1 目录下，使用 wget 命令，从

<http://192.168.1.100:60000/allfiles/mapreduce1/hadoop2lib.tar.gz>，下载项目用到的依赖包。

```
view plain copy
1. wget http://192.168.1.100:60000/allfiles/mapreduce1/hadoop2lib.tar.gz
```

将 hadoop2lib.tar.gz 解压到当前目录下。

```
view plain copy
1. tar -xzvf hadoop2lib.tar.gz
```

4. 将 linux 本地/data/mapreduce1/buyer\_favorite1，上传到 HDFS 上的/mymapreduce1/in 目录下。

若 HDFS 目录不存在，需提前创建。



1. `hadoop fs -mkdir -p /mymapreduce1/in`
2. `hadoop fs -put /data/mapreduce1/buyer_favorite1 /mymapreduce1/in`

5. 打开 Eclipse , 新建 Java Project 项目。



并将项目名称设置为 mapreduce1。

**New Java Project**

**Create a Java Project**

Create a Java project in the workspace or in an external location.

Project name:

☒ Use default location

Location:  [Browse...](#)

**JRE**

☒ Use an execution environment JRE:  ▼

☐ Use a project specific JRE:  ▼

☐ Use default JRE (currently 'eclipse') [Configure JREs...](#)

**Project layout**

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

**Working sets**

☐ Add project to working sets

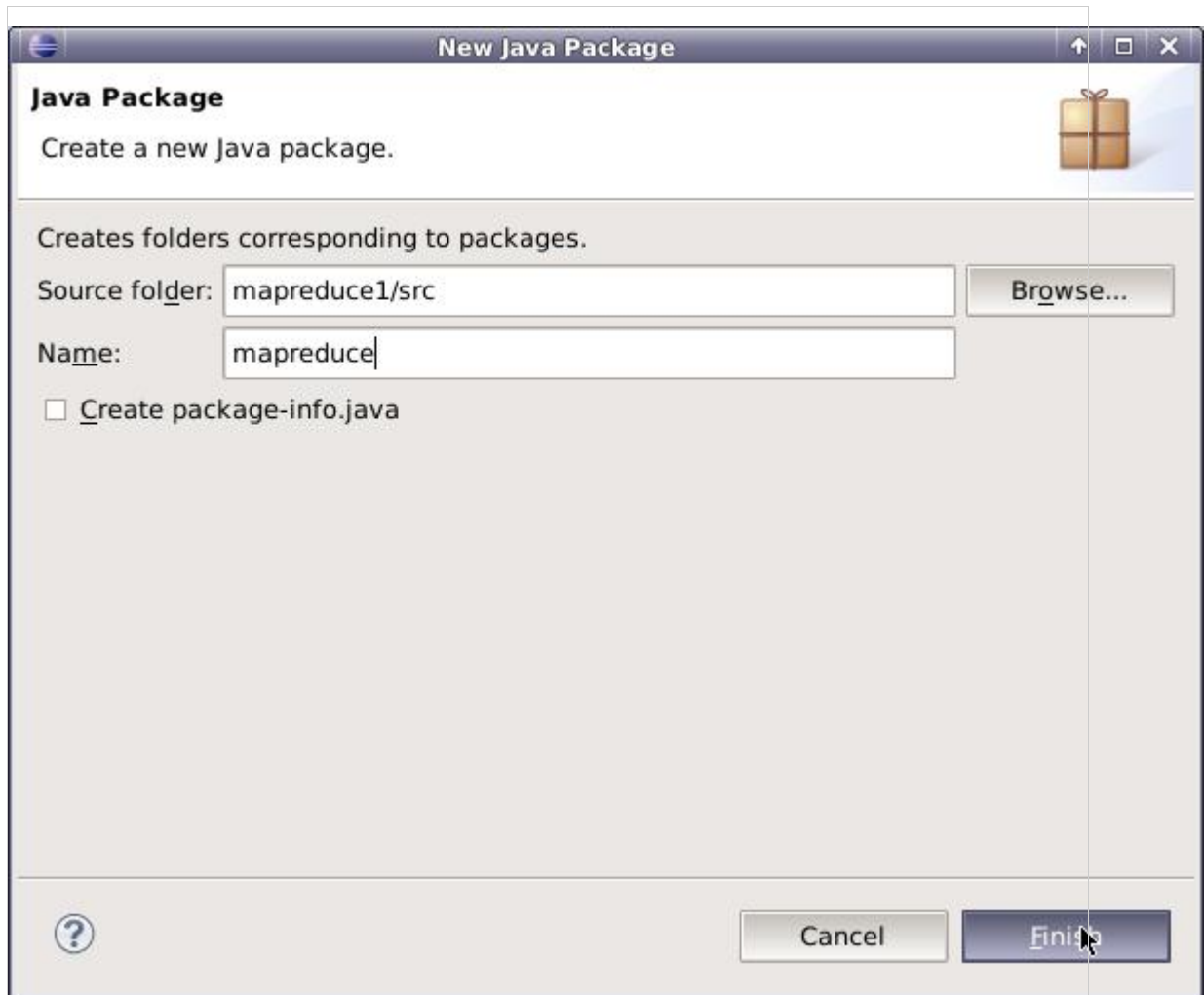
Working sets:  ▼ [Select...](#)

[?](#) [< Back](#) [Next >](#) [Cancel](#) [Finish](#)

6.在项目名称 mapreduce1 下，新建 package 包。



并将包命名为 mapreduce 。



7.在创建的包 mapreduce 下，新建类。



并将类命名为 WordCount。

**New Java Class**

**Java Class**  
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

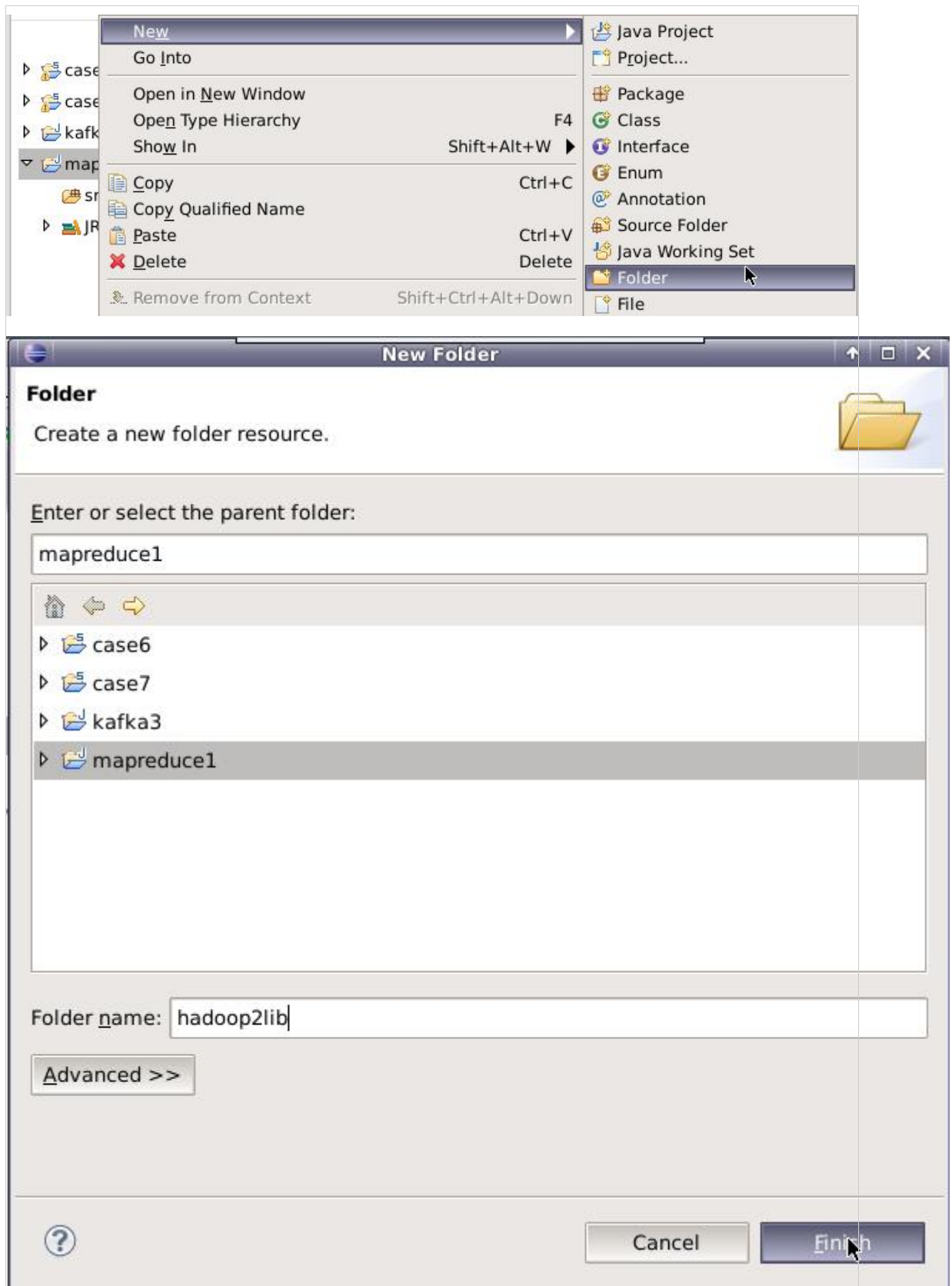
☐ Constructors from superclass

☒ Inherited abstract methods

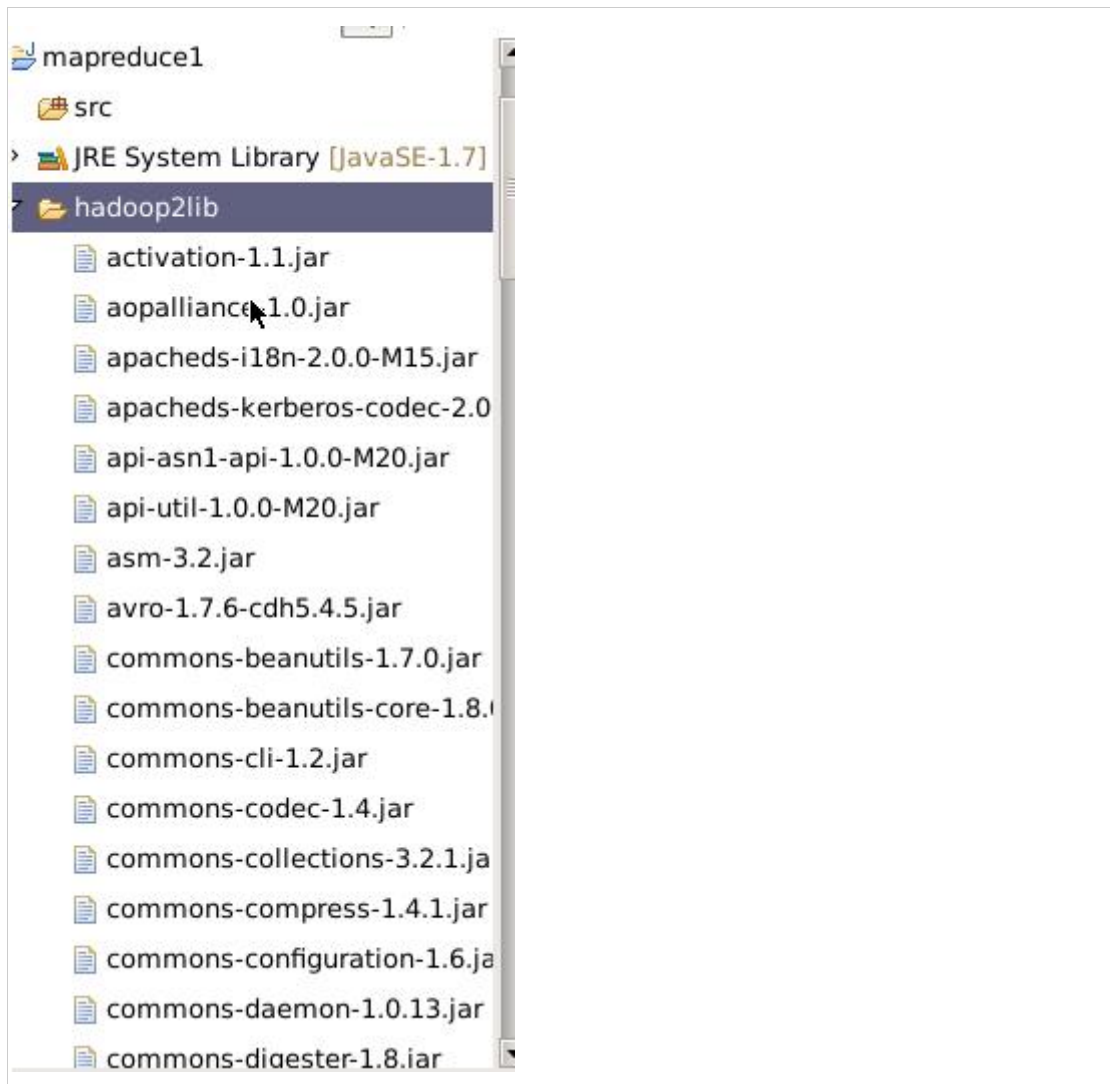
Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

8. 添加项目所需依赖的 jar 包，右键单击项目名，新建一个目录 hadoop2lib，用于存放项目所需的 jar 包。



将 linux 上/data/mapreduce1 目录下 ,hadoop2lib 目录中的 jar 包 ,全部拷贝到 eclipse 中 ,mapreduce1 项目的 hadoop2lib 目录下。

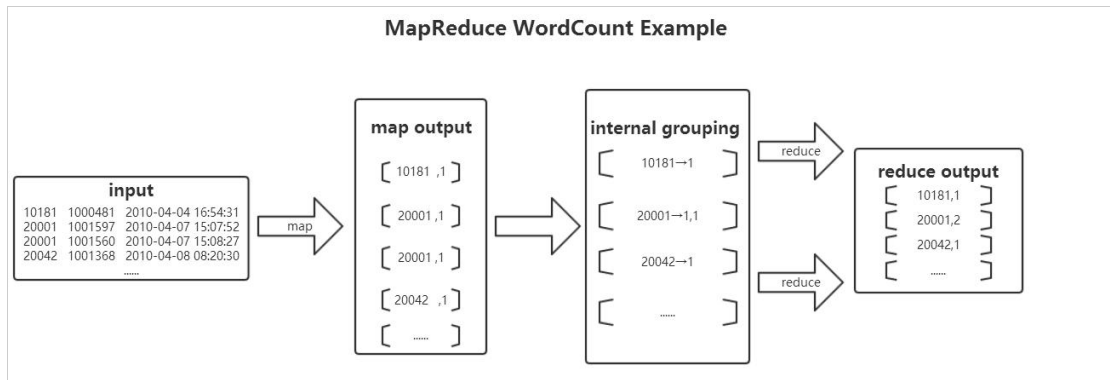


选中 hadoop2lib 目录下所有的 jar 包，单击右键，选择 Build Path=>Add to Build Path



9.编写 Java 代码，并描述其设计思路。

下图描述了该 mapreduce 的执行过程



大致思路是将 hdfs 上的文本作为输入，MapReduce 通过 InputFormat 会将文本进行切片处理，并将每行的首字母相对于文本文件的首地址的偏移量作为输入键值对的 key，文本内容作为输入键值对的 value，经过在 map 函数处理，输出中间结果<word,1>的形式，并在 reduce 函数中完成对每个单词的词频统计。整个程序代码主要包括两部分：Mapper 部分和 Reducer 部分。

Mapper 代码

```
1. public static class doMapper extends Mapper<Object, Text, Text, IntWritable>{
2.     //第一个 Object 表示输入 key 的类型；第二个 Text 表示输入 value 的类型；第三个 Text 表示输出键的类型；第四个 IntWritable 表示输出值的类型
3.     public static final IntWritable one = new IntWritable(1);
4.     public static Text word = new Text();
5.     @Override
6.     protected void map(Object key, Text value, Context context)
7.         throws IOException, InterruptedException
8.         //抛出异常
9.     {
10.         StringTokenizer tokenizer = new StringTokenizer(value.toString(), "\t");
11.         //StringTokenizer 是 Java 工具包中的一个类，用于将字符串进行拆分
12.
13.         word.set(tokenizer.nextToken());
14.         //返回当前位置到下一个分隔符之间的字符串
15.         context.write(word, one);
16.         //将 word 存到容器中，记一个数
17.     }
```

在 map 函数里有三个参数 前面两个 Object key,Text value 就是输入的 key 和 value 第三个参数 Context context 是可以记录输入的 key 和 value。例如 context.write(word,one)；此外 context 还会记录 map 运算的状态。map 阶段采用 Hadoop 的默认的作业输入方式，把输入的 value 用 StringTokenizer()方法 截取出的买家 id 字段设置为 key，设置 value 为 1，然后直接输出<key,value>。



## Reducer 代码

[view plain copy](#)

```
1. public static class doReducer extends Reducer<Text, IntWritable, Text,
   IntWritable>{
2. //参数同 Map 一样, 依次表示是输入键类型, 输入值类型, 输出键类型, 输出值类型
3. private IntWritable result = new IntWritable();
4.     @Override
5.     protected void reduce(Text key, Iterable<IntWritable> values,
   Context context)
6.     throws IOException, InterruptedException {
7.         int sum = 0;
8.         for (IntWritable value : values) {
9.             sum += value.get();
10.        }
11.        //for 循环遍历, 将得到的 values 值累加
12.        result.set(sum);
13.        context.write(key, result);
14.    }
15. }
```

map 输出的<key,value>先要经过 shuffle 过程把相同 key 值的所有 value 聚集起来形成<key,values>后交给 reduce 端。reduce 端接收到<key,values>之后, 将输入的 key 直接复制给输出的 key,用 for 循环遍历 values 并求和 求和结果就是 key 值代表的单词出现的总次 将其设置为 value 直接输出<key,value>。

## 完整代码

[view plain copy](#)

```
1. package mapreduce;
2. import java.io.IOException;
3. import java.util.StringTokenizer;
4. import org.apache.hadoop.fs.Path;
5. import org.apache.hadoop.io.IntWritable;
6. import org.apache.hadoop.io.Text;
7. import org.apache.hadoop.mapreduce.Job;
8. import org.apache.hadoop.mapreduce.Mapper;
9. import org.apache.hadoop.mapreduce.Reducer;
10. import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
12. public class WordCount {
13.     public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {
14.         Job job = Job.getInstance();
15.         job.setJobName("WordCount");
16.         job.setJarByClass(WordCount.class);
17.         job.setMapperClass(doMapper.class);
18.         job.setReducerClass(doReducer.class);
```

```

19.         job.setOutputKeyClass(Text.class);
20.         job.setOutputValueClass(IntWritable.class);
21.         Path in = new Path("hdfs://localhost:9000/mymapreduce1/in/buyer_favorite1");
22.         Path out = new Path("hdfs://localhost:9000/mymapreduce1/out");

23.         FileInputFormat.addInputPath(job, in);
24.         FileOutputFormat.setOutputPath(job, out);
25.         System.exit(job.waitForCompletion(true) ? 0 : 1);
26.     }

27.     public static class doMapper extends Mapper<Object, Text, Text, IntWritable>{
28.         public static final IntWritable one = new IntWritable(1);
29.         public static Text word = new Text();
30.         @Override
31.         protected void map(Object key, Text value, Context context)
32.             throws IOException, InterruptedException {
33.             StringTokenizer tokenizer = new StringTokenizer(value.toString(), "\\t");
34.             word.set(tokenizer.nextToken());
35.             context.write(word, one);
36.         }
37.     }

38.     public static class doReducer extends Reducer<Text, IntWritable, Text, IntWritable>{
39.         private IntWritable result = new IntWritable();
40.         @Override
41.         protected void reduce(Text key, Iterable<IntWritable> values, Context context)
42.             throws IOException, InterruptedException {
43.             int sum = 0;
44.             for (IntWritable value : values) {
45.                 sum += value.get();
46.             }
47.             result.set(sum);
48.             context.write(key, result);
49.         }
50.     }
51. }

```

10.在 WordCount 类文件中，单击右键=>Run As=>Run on Hadoop 选项，将 MapReduce 任务提交到 Hadoop 中。



11.待执行完毕后，打开终端或使用 hadoop eclipse 插件，查看 hdfs 上，程序输出的实验结果。

[view plain copy](#)

1. `hadoop fs -ls /mymapreduce1/out`

2. `hadoop fs -cat /mymapreduce1/out/part-r-00000`

```
zhangyu@a6a57a8dc3bf:/data/mapreduce1$ hadoop fs -ls /mymapreduce1/out
Found 2 items
-rw-r--r--   3 zhangyu supergroup          0 2017-01-05 07:35 /mymapreduce1/out/_SUCCESS
-rw-r--r--   3 zhangyu supergroup       73 2017-01-05 07:35 /mymapreduce1/out/part-r-00000
zhangyu@a6a57a8dc3bf:/data/mapreduce1$ hadoop fs -cat /mymapreduce1/out/part-r-00000
10181    1
20001    2
20042    1
20054    6
20055    1
20056   12
20064    1
20067    1
20076    5
```

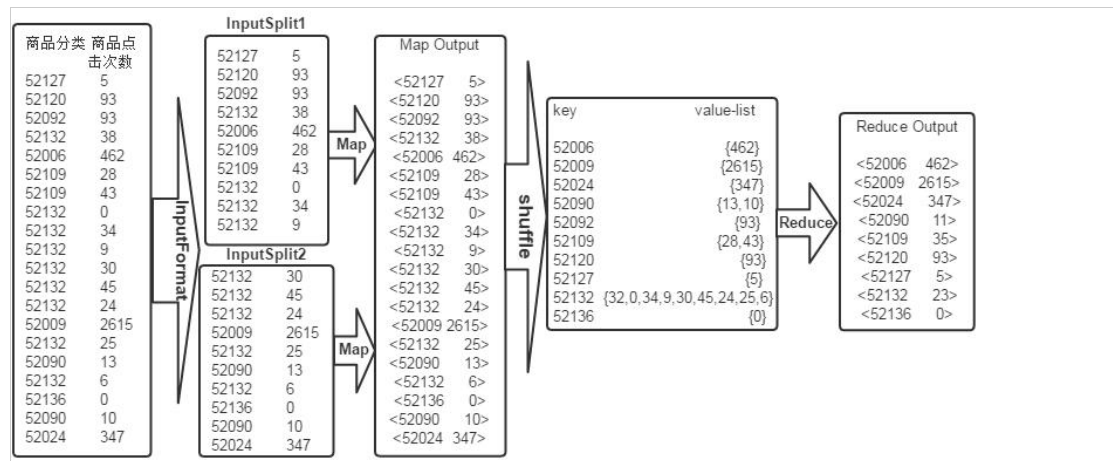
# 实验七：MAPREDUCE 求平均值

## 一、任务目标

- 1.准确理解 Mapreduce 求平均值的设计原理
- 2.熟练掌握 Mapreduce 求平均值程序的编写
- 3.学会编写 Mapreduce 求平均值程序代码解决问题

## 二、相关知识

求平均数是 MapReduce 比较常见的算法，求平均数的算法也比较简单，一种思路是 Map 端读取数据，在数据输入到 Reduce 之前先经过 shuffle，将 map 函数输出的 key 值相同的所有的 value 值形成一个集合 value-list，然后将<key, value-list style="margin: 0px; padding: 0px; box-sizing: border-box;">输入到 Reduce 端，Reduce 端汇总并且统计记录数，然后作商即可。具体原理如下图所示：



## 三、系统环境

Linux Ubuntu 16.04

jdk-7u75-linux-x64

hadoop-2.6.0-cdh5.4.5

hadoop-2.6.0-eclipse-cdh5.4.5.jar

eclipse-java-juno-SR2-linux-gtk-x86\_64

## 四、任务内容

现有某电商关于商品点击情况的数据文件，表名为 goods\_click，包含两个字段（商品分类，商品点击次数），分隔符“\t”，由于数据很大，所以为了方便统计我们只截取它的一部分数据，内容如下：

|                 |             |
|-----------------|-------------|
| view plain copy |             |
| 1.              | 商品分类 商品点击次数 |
| 2.              | 52127 5     |
| 3.              | 52120 93    |
| 4.              | 52092 93    |
| 5.              | 52132 38    |
| 6.              | 52006 462   |
| 7.              | 52109 28    |

```
8. 52109 43
9. 52132 0
10. 52132 34
11. 52132 9
12. 52132 30
13. 52132 45
14. 52132 24
15. 52009 2615
16. 52132 25
17. 52090 13
18. 52132 6
19. 52136 0
20. 52090 10
21. 52024 347
```

要求使用 mapreduce 统计出每类商品的平均点击次数。

结果数据如下：

```
view plain copy
1. 商品分类 商品平均点击次数
2. 52006 462
3. 52009 2615
4. 52024 347
5. 52090 11
6. 52092 93
7. 52109 35
8. 52120 93
9. 52127 5
10. 52132 23
11. 52136 0
```

## 五、任务步骤

1.切换到/apps/hadoop/sbin 目录下，开启 Hadoop。

```
view plain copy
1. cd /apps/hadoop/sbin
2. ./start-all.sh
```

2.在 Linux 本地新建/data/mapreduce4 目录。

```
view plain copy
1. mkdir -p /data/mapreduce4
```

3. 在 Linux 中 切 换 到 /data/mapreduce4 目 录 下 ， 用 wget 命 令 从 [http://192.168.1.100:60000/allfiles/mapreduce4/goods\\_click](http://192.168.1.100:60000/allfiles/mapreduce4/goods_click) 网址上下载文本文件 goods\_click。

```
view plain copy
1. cd /data/mapreduce4
2. wget http://192.168.1.100:60000/allfiles/mapreduce4/goods_click
```

然后在当前目录下用 `wget` 命令从 <http://192.168.1.100:60000/allfiles/mapreduce4/hadoop2lib.tar.gz> 网址上下载项目用到的依赖包。

[view plain copy](#)

```
1. wget http://192.168.1.100:60000/allfiles/mapreduce4/hadoop2lib.tar.gz
```

将 `hadoop2lib.tar.gz` 解压到当前目录下。

[view plain copy](#)

```
1. tar zxvf hadoop2lib.tar.gz
```

4.首先在 HDFS 上新建 `/mymapreduce4/in` 目录，然后将 Linux 本地 `/data/mapreduce4` 目录下的 `goods_click` 文件导入到 HDFS 的 `/mymapreduce4/in` 目录中。

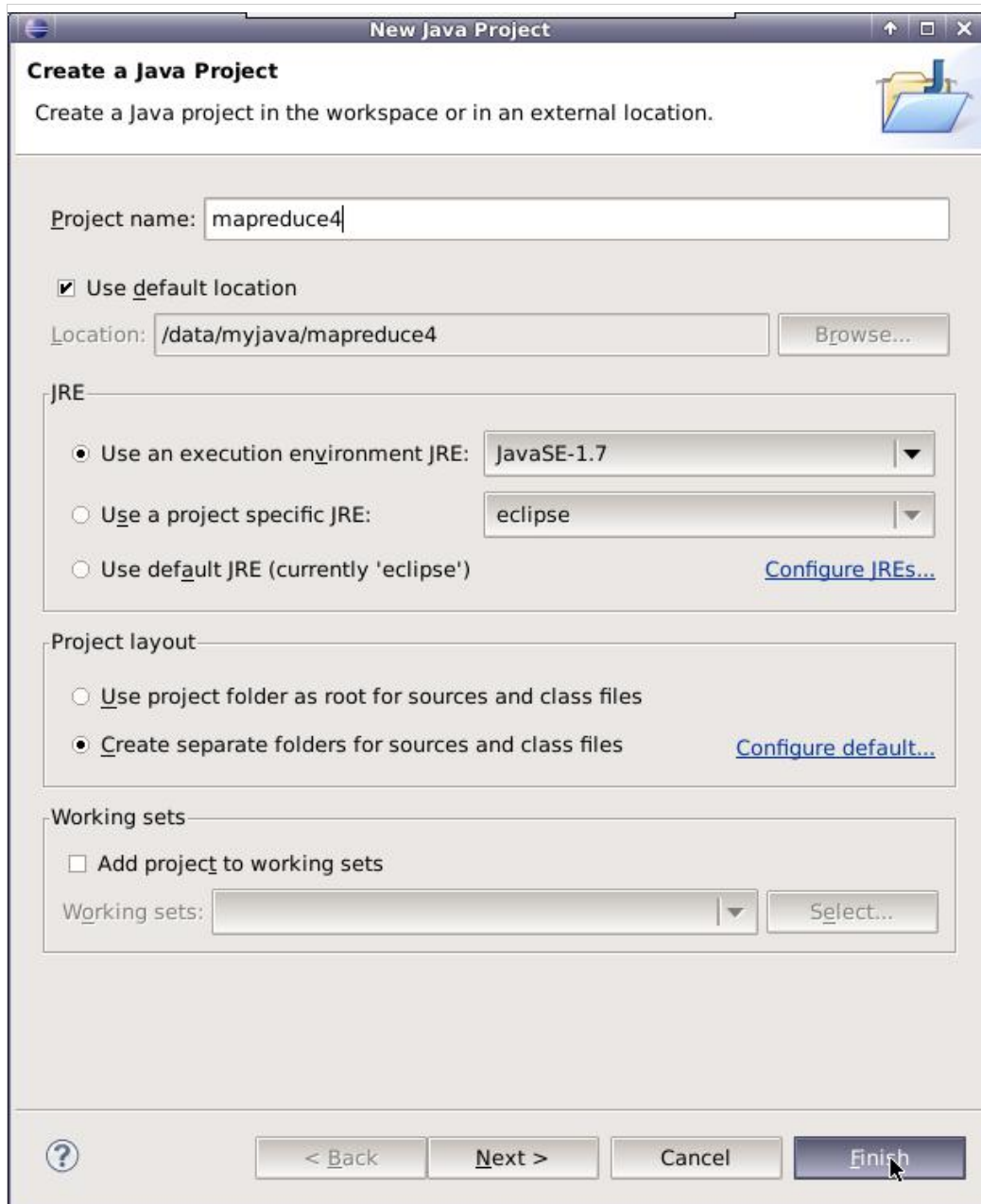
[view plain copy](#)

```
1. hadoop fs -mkdir -p /mymapreduce4/in
```

```
2. hadoop fs -put /data/mapreduce4/goods_click /mymapreduce4/in
```

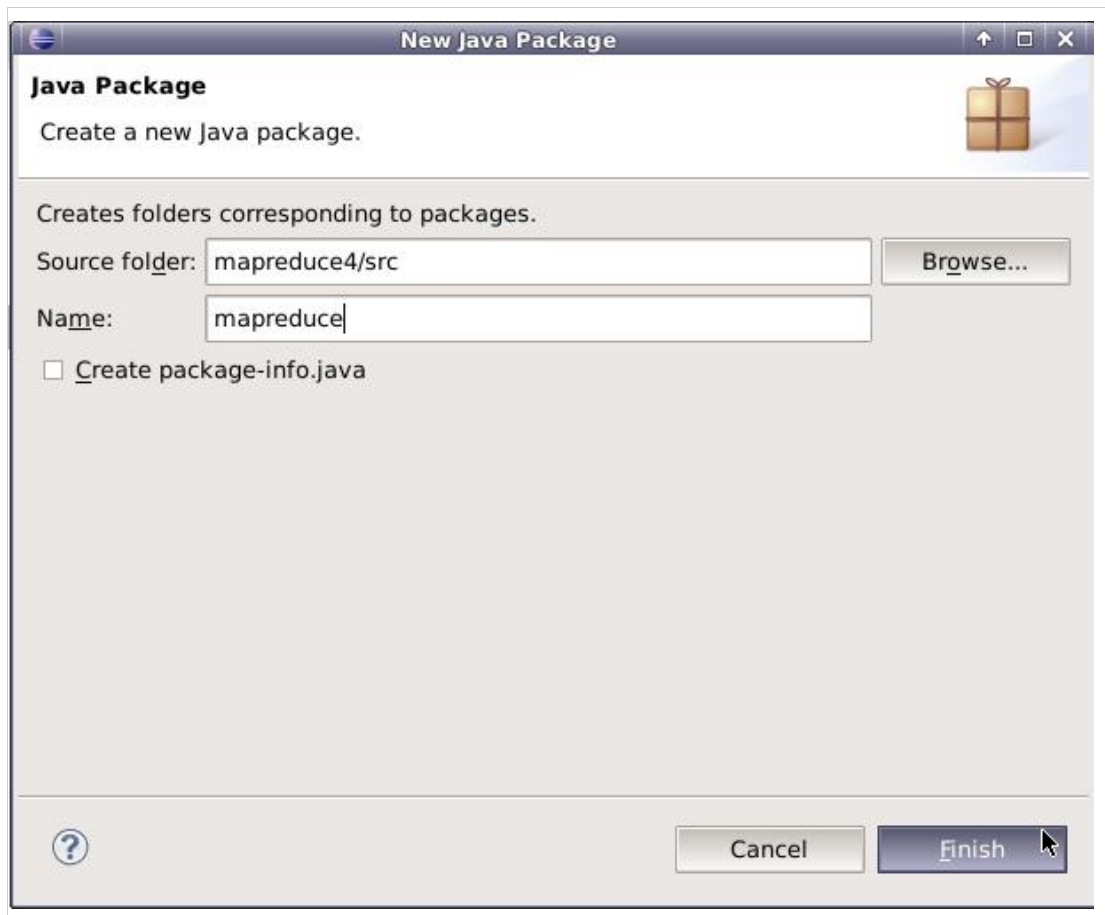
5.新建 Java Project 项目，项目名为 `mapreduce4`。





在 mapreduce4 项目下新建包，包名为 mapreduce。





在 mapreduce 包下新建类，类名为 MyAverage。





**New Java Class**

**Java Class**  
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

---

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

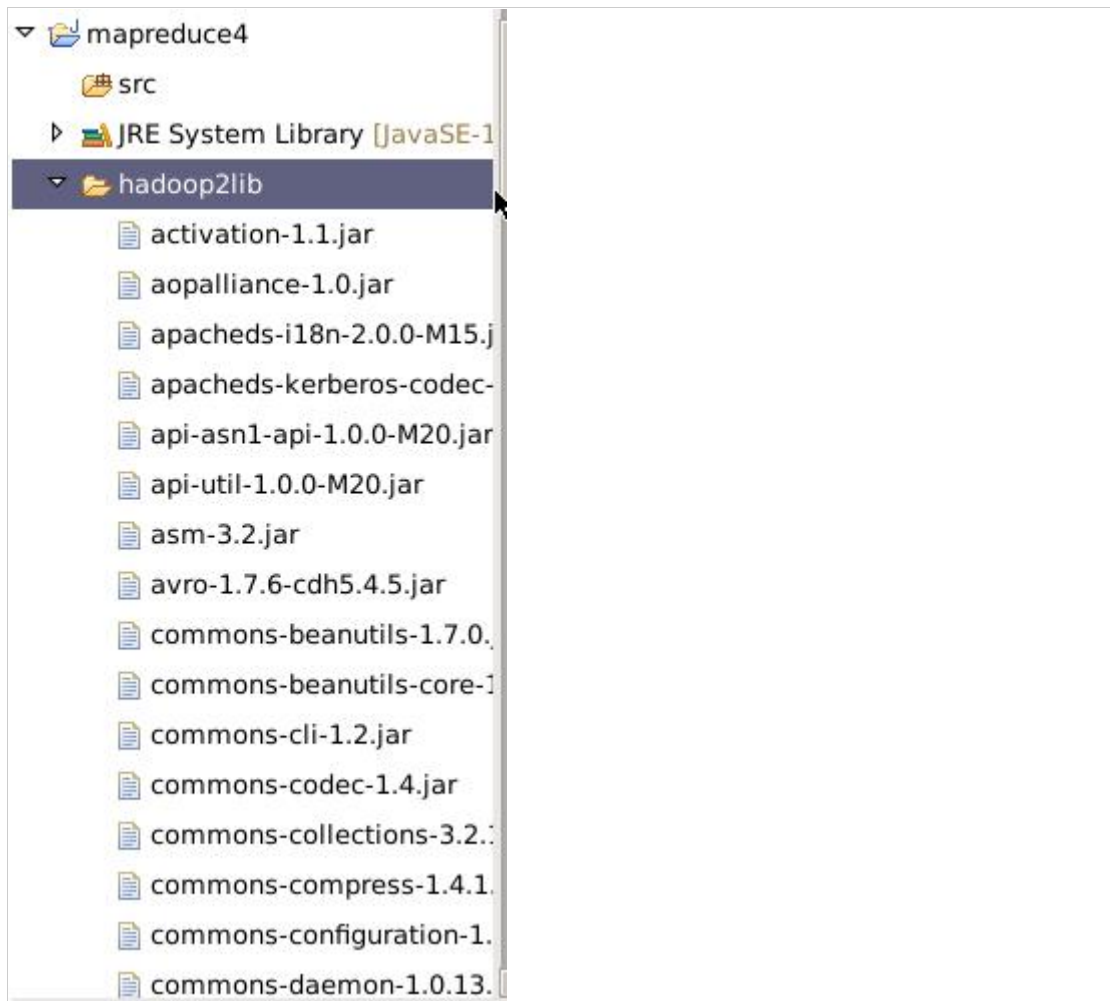
Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

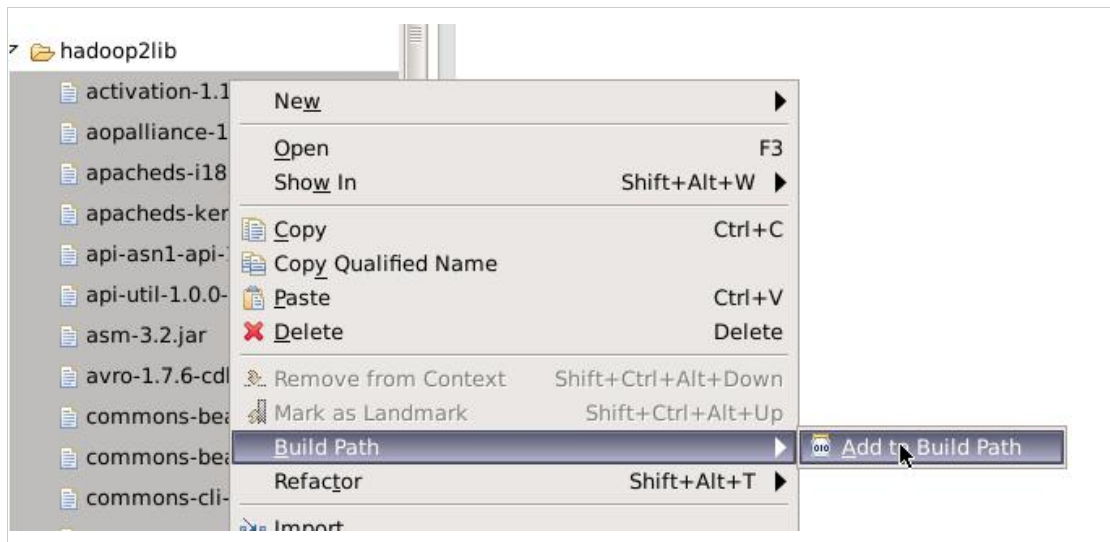
6. 添加项目所需依赖的 jar 包，右键点击 mapreduce4，新建一个文件夹，名为 hadoop2lib，用于存放项目所需的 jar 包。



将/data/mapreduce4 目录下，hadoop2lib 目录中的 jar 包，拷贝到 eclipse 中 mapreduce4 项目的 hadoop2lib 目录下。



选中 hadoop2lib 目录下所有 jar 包，并添加到 Build Path 中。



7. 编写 Java 代码并描述其设计思路。

Mapper 代码

```
view plain copy
1. public static class Map extends Mapper<Object , Text , Text , IntWritable>{
```

```

2.     private static Text newKey=new Text();
3.     //实现 map 函数
4.     public void map(Object key,Text value,Context context) throws IOException, InterruptedException{
5.         // 将输入的纯文本文件的数据转化成 String
6.         String line=value.toString();
7.         System.out.println(line);
8.         String arr[]=line.split("\t");
9.         newKey.set(arr[0]);
10.        int click=Integer.parseInt(arr[1]);
11.        context.write(newKey, new IntWritable(click));
12.    }
13. }

```

map 端在采用 Hadoop 的默认输入方式之后，将输入的 value 值通过 split()方法截取出来，我们把截取的商品点击次数字段转化为 IntWritable 类型并将其设置为 value，把商品分类字段设置为 key,然后直接输出 key/value 的值。

Reducer 代码

```

1. public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>{
2.     //实现 reduce 函数
3.     public void reduce(Text key,Iterable<IntWritable> values,Context context) throws IOException, InterruptedException{
4.         int num=0;
5.         int count=0;
6.         for(IntWritable val:values){
7.             num+=val.get(); //每个元素求和 num
8.             count++;        //统计元素的次数 count
9.         }
10.        int avg=num/count; //计算平均数
11.
12.        context.write(key,new IntWritable(avg));
13.    }
14. }

```

map 的输出<key,value>经过 shuffle 过程集成<key,values>键值对，然后将<key,values>键值对交给 reduce。reduce 端接收到 values 之后，将输入的 key 直接复制给输出的 key，将 values 通过 for 循环把里面的每个元素求和 num 并统计元素的次数 count，然后用 num 除以 count 得到平均值 avg，将 avg 设置为 value，最后直接输出<key,value>就可以了。

完整代码

```

1. package mapreduce;
2. import java.io.IOException;

```

```
3. import org.apache.hadoop.conf.Configuration;
4. import org.apache.hadoop.fs.Path;
5. import org.apache.hadoop.io.IntWritable;
6. import org.apache.hadoop.io.NullWritable;
7. import org.apache.hadoop.io.Text;
8. import org.apache.hadoop.mapreduce.Job;
9. import org.apache.hadoop.mapreduce.Mapper;
10. import org.apache.hadoop.mapreduce.Reducer;
11. import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
12. import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
13. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
14. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
15. public class MyAverage{
16.     public static class Map extends Mapper<Object , Text , Text , Int
        Writable>{
17.         private static Text newKey=new Text();
18.         public void map(Object key,Text value,Context context) throws IOE
        xception, InterruptedException{
19.             String line=value.toString();
20.             System.out.println(line);
21.             String arr[]=line.split("\t");
22.             newKey.set(arr[0]);
23.             int click=Integer.parseInt(arr[1]);
24.             context.write(newKey, new IntWritable(click));
25.         }
26.     }
27.     public static class Reduce extends Reducer<Text, IntWritable, Tex
        t, IntWritable>{
28.         public void reduce(Text key,Iterable<IntWritable> values,Context
        context) throws IOException, InterruptedException{
29.             int num=0;
30.             int count=0;
31.             for(IntWritable val:values){
32.                 num+=val.get();
33.                 count++;
34.             }
35.             int avg=num/count;
36.             context.write(key,new IntWritable(avg));
37.         }
38.     }
39.     public static void main(String[] args) throws IOException, Cl
        assNotFoundException, InterruptedException{
40.         Configuration conf=new Configuration();
41.         System.out.println("start");
```

```

42.      Job job =new Job(conf, "MyAverage");
43.      job.setJarByClass(MyAverage.class);
44.      job.setMapperClass(Map.class);
45.      job.setReducerClass(Reduce.class);
46.      job.setOutputKeyClass(Text.class);
47.      job.setOutputValueClass(IntWritable.class);
48.      job.setInputFormatClass(TextInputFormat.class);
49.      job.setOutputFormatClass(TextOutputFormat.class);
50.      Path in=new Path("hdfs://localhost:9000/mymapreduce4/in/goods
    _click");
51.      Path out=new Path("hdfs://localhost:9000/mymapreduce4/out");

52.      FileInputFormat.addInputPath(job,in);
53.      FileOutputFormat.setOutputPath(job,out);
54.      System.exit(job.waitForCompletion(true) ? 0 : 1);
55.
56.  }
57.  }

```

8.在 MyAverage 类文件中，右键并点击=>Run As=>Run on Hadoop 选项，将 MapReduce 任务提交到 Hadoop 中。



9.待执行完毕后，进入命令模式下，在 HDFS 上/mymapreduce4/out 中查看实验结果。

[view plain copy](#)

1. `hadoop fs -ls /mymapreduce4/out`
2. `hadoop fs -cat /mymapreduce4/out/part-r-00000`

```

zhangyu@a6a57a8dc3bf:/data/mapreduce4$ hadoop fs -ls /mymapreduce4/out
Found 2 items
-rw-r--r--  3 zhangyu supergroup      0 2017-01-06 02:23 /mymapreduce4/out/_SUCCESS
-rw-r--r--  3 zhangyu supergroup    92 2017-01-06 02:23 /mymapreduce4/out/part-r-00000
zhangyu@a6a57a8dc3bf:/data/mapreduce4$ hadoop fs -cat /mymapreduce4/out/part-r-00000
52006 462
52009 2615
52024 347
52090 11
52092 93
52109 35
52120 93
52127 5
52132 23
52136 0

```

# 实验八：数据统计分析

## 一、实验目的：

编写 MapReduce 数据分析统计程序

## 二、实验内容：

### 1、数据分析排序程序：

现在有多个输入文件，每个文件中的每行内容均为一个整数。要求读取所有文件中的整数，进行升序排序后，输出到一个新的文件中，输出的数据格式为每行两个整数，第一个数字为第二个整数的排序位次，第二个整数为原待排列的整数。下面是输入文件和输出文件的一个样例供参考，要求完成的是随机的整数，不能雷同，不能和样例相同。

输入文件 1 的样例如下：

```
33
37
12
40
```

输入文件 2 的样例如下：

```
4
16
39
5
```

输入文件 3 的样例如下：

```
1
45
25
```

根据输入文件 1、2 和 3 得到的输出文件如下：

```
1  1
2  4
3  5
4 12
5 16
6 25
7 33
8 37
9 39
10 40
11 45
```

## 2、编程实现文件合并和去重操作

对于两个输入文件，即文件 A 和文件 B，请编写 MapReduce 程序，对两个文件进行合并，并剔除其中重复的内容，得到一个新的输出文件 C。下面是输入文件和输出文件的一个样例供参考。要求完成的是随机的符号串，不能雷同，不能和样例相同。

输入文件 A 的样例如下：

```
20150101    x
20150102    y
20150103    x
20150104    y
20150105    z
20150106    x
```

输入文件 B 的样例如下：



|          |   |
|----------|---|
| 20150101 | y |
| 20150102 | y |
| 20150103 | x |
| 20150104 | z |
| 20150105 | y |

根据输入文件 A 和 B 合并得到的输出文件 C 的样例如下：

|          |   |
|----------|---|
| 20150101 | x |
| 20150101 | y |
| 20150102 | y |
| 20150103 | x |
| 20150104 | y |
| 20150104 | z |
| 20150105 | y |
| 20150105 | z |
| 20150106 | x |

3、对给定的表格进行信息挖掘（拓展题，根据情况选作，正确加分，错误不扣分）

下面给出一个 child-parent 的表格，要求挖掘其中的父子辈关系，给出祖孙辈关系的表格。下面是输入文件和输出文件的一个样例供参考。要求完成的是随机的 child 和 parent 的姓名，不能雷同，不能和样例相同。

输入文件内容如下：

| child  | parent |
|--------|--------|
| Steven | Lucy   |
| Steven | Jack   |
| Jone   | Lucy   |
| Jone   | Jack   |
| Lucy   | Mary   |

|        |       |
|--------|-------|
| Lucy   | Frank |
| Jack   | Alice |
| Jack   | Jesse |
| David  | Alice |
| David  | Jesse |
| Philip | David |
| Philip | Alma  |
| Mark   | David |
| Mark   | Alma  |

输出文件内容如下：

| <b>grandchild</b> | <b>grandparent</b> |
|-------------------|--------------------|
| Steven            | Alice              |
| Steven            | Jesse              |
| Jone              | Alice              |
| Jone              | Jesse              |
| Steven            | Mary               |
| Steven            | Frank              |
| Jone              | Mary               |
| Jone              | Frank              |
| Philip            | Alice              |
| Philip            | Jesse              |
| Mark              | Alice              |
| Mark              | Jesse              |

## 附录：实验报告模板

封面格式：

# 《大数据技术与应用》 实验报告

学号：XXXXX  
班级：XXXXX  
姓名：XXXXX

石家庄铁道大学信息学院  
(章鱼互联网学院平台)

**2020 年春季学期**

## 实验 X: XXXXXX

- 一、任务目标（简单描述）
- 二、系统环境（简单描述）
- 三、任务内容（简单描述）
- 四、任务步骤

实验过程要对重要的步骤进行截图或文字说明

比如：

运行实例，执行情况如下：（截图带时间和注册姓名（实名注册））

[illegible]

## 五、实验总结