一．建虚拟机

1.建三个虚拟机，装Linux系统，可装完一个复制两个系统（文件复制，或工作站“虚拟机-》克隆”）

2.保证三个虚拟机网络通畅，IP分别为：192.168.1.51/52/53

涉及到的命令：ifconfig、service network restart、ping XXX

修改静态IP：ifconfig eth0 192.168.1.xx

3.修改主机名，三个主机名分别为：gdfwdb01/02/03

修改方法：

a.更改/etc/sysconfig下的network文件，在提示符下输入vi /etc/sysconfig/network，

然后将HOSTNAME后面的值改为想要设置的主机名。

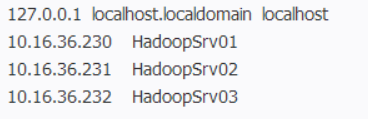
b.更改/etc下的hosts文件，在提示符下输入vi /etc/hosts，加入三个节点对应ip及主机名，保存退出；（i 编辑，esc 后 :wq!保存退出，:q!不保存退出)；

所有节点均：配置hosts文件(目的：解析集群主机名及IP)

192.168.1.51 gdfwdb01

192.168.1.52 gdfwdb02

192.168.1.53 gdfwdb03



c.重启，reboot

5.新建Hadoop用户

# useradd hadp

# passwd hadp (密码：hadoop123)

关防火墙

chkconfig iptables off 关闭

chkconfig iptables --list 查看

禁用selinux:

编辑 /etc/selinux/config文件，设置“SELINUX=disabled”

二．装jdk

6.装jdk

上传jdk安装包，不同节点间复制，scp jdk-6u35-linux-x64.bin hadp@gdfwdb03:/home/hadp

安装：hadp用户，

$ chmod u+x jdk-6u35-linux-x64.bin

$ ./jdk-6u35-linux-x64.bin

装完后，配JAVA\_HOME

$ vi .bash\_profile

加入：

export JAVA\_HOME=/home/hadp/jdk1.6.0\_35

export PATH=$JAVA\_HOME/bin:$PATH

让配置生效，导入全局环境变量

$ source .bash\_profile

三．ssh免密码登录

7.ssh密钥对,保证节点间ssh链接免密钥接入，需在hadp用户底下执行

$ ssh-keygen -t rsa（之后默认回车）

$ cd .ssh

$ ls

$ cp id\_rsa.pub authorized\_keys (意思：将生成的公钥内容复制到authorized\_keys，注意文件名不能错)

$ cat authorized\_keys(查看文件内容)

把三个节点authorized\_keys内容全部拿出来组合成一个大文件，覆盖掉各个节点的authorized\_keys

这样彼此间就免密码ssh连入

$ scp hadp@gdfwdb02:~/.ssh/id\_rsa.pub ./2\_rsa.pub（将2节点的公钥复制过来，命名为2\_rsa.pub）

$ cat 2\_rsa.pub >> authorized\_keys （将2的内容加到authorized\_keys后面）

重复。。。

删除文件 rm -f 文件名

删除文件夹 rm -rf 目录名

完成后，在节点1输入命令$ ssh gdfwdb02，第一次询问输入yes，成功连接到节点2，

输入命令$ exit，退回到节点1，再输一次连接，本次不再询问，成功！

三个节点之间重复执行一下连接工作，包括本身，自己连自己

切换到root用户，修改一个ssh参数，防止后期启动Hadoop报错：

启动hadoop报192.168.1.51: Address 192.168.1.51 maps to gdfwdb01, but this does not map back to the address - POSSIBLE BREAK-IN ATTEMPT!

$ vi /etc/ssh/ssh\_config

GSSAPIAuthentication yse

----改为：----

GSSAPIAuthentication no

据说GSSAPIAuthentication的作用：是否允许使用基于GSSAPI的用户认证。

是否在用户退出登录后自动销毁用户凭证缓存。默认值是”yes”

四．三台服务器时间同步

因为节点与主节点之间有心跳机制监测服务器是否存活，因此彼此之间时间要求一致，需进行服务器时间同步工作，利用linux自带服务ntp(Network Time Protocol,网络时间协议，用来使网络中的各个计算机时间同步的一种协议)服务。我们将master作为时间服务器。

Root用户操作一下步骤：

1. 查询是否安装ntp，若没有安装的话使用yum进行安装。

检查命令：rpm -q ntp

安装命令：yum–y install ntp

1. 修改master服务器上ntp配置文件， vi /etc/ntp.conf

#restrict -6 ::1

restrict 192.168.1.0 mask 255.255.255.0 nomodify notrap #1网段的客户端都可以访问

#server 0.pool.ntp.org ＃不配置Internet网络时间服务器 三个server全注释

1. 启动master ntp服务，service ntpd restart(start/restart/stop/status)

让ntp服务自动启动：chkconfig ntpd on

测试：ntptrace

ntpq –p

查看进程：ps –ef | grep ntpd

1. 修改各子节点配置文件，vi /etc/ntp.conf

#restrict -6 ::1

restrict 192.168.1.51 mask 255.255.255.0 nomodify

server 192.168.1.51 #设置局域网时间服务器地址

#server 0.pool.ntp.org ＃不配置Internet网络时间服务器 三个server全注释

1. 查询时间：date
2. 执行命令，与服务器同步时间

ntpdate 192.168.1.51

1. 再查询时间看看是否已同步
2. 如果执行同步时间命令出现错误：
3. the NTP socket is in use, exiting(ntpd进程已存在，利用上面查进程命令，kill掉ntp进程)
4. No Server suitable for synchronization found(配置好服务器并启动服务器进程后，马上启动客户端进程，就会报错，需大概3-5分钟以后启动客户端进程)

五．Hadoop安装

8.在一个节点操作：解压安装包

$ tar xzvf hadoop-2.5.2.tar.gz

x 从档案文件中释放文件。相当于拆包。

-z ：是否同时具有 gzip 的属性？亦即是否需要用 gzip 压缩或解压？ 一般格式为xx.tar.gz或xx. tgz

-v ：压缩的过程中显示文件

-f ：使用档名，请留意，在 f 之后要立即接档名喔！不要再加其他参数！

$ vi .bash\_profile

加入：

export HADOOP\_PREFIX=/home/hadp/hadoop-2.5.2

export PATH=$JAVA\_HOME/bin:$HADOOP\_PREFIX/bin:$PATH

export HADOOP\_OPTS="-Djava.library.path=$HADOOP\_PREFIX/lib:$HADOOP\_PREFIX/lib/native"

(

Hadoop安装包是在32位机器上编译的，64位机器加载本地库.so文件出错，不影响使用。

解决办法：

1.重新编译源码后将新的lib/native替换到集群中原来的lib/native

2.修改hadoop-env.sh,增加export HADOOP\_OPTS...

使用ant

下载apache-ant新包

ssh上传到虚拟机，解压，file apache-ant-xxx.tar.gz发现是gzip压缩的，先gunzip解压为apache-ant-xx.tar,

然后tar xzvf apache-ant-xx.tar

配置环境变量

export ANT\_HOME=/home/hadp/apache-ant-1.9.5

export PATH=....:$ANT\_HOME/bin:$PATH

$ source .bash\_profile

ant compile-native

)

让配置生效，导入全局环境变量

$ source .bash\_profile

六．Hadoop参数文件配置

新建文件夹，存放数据

$ cd hadoop-2.5.2

$ mkdir name data tmp

9.配置Hadoop环境变量,目录：hadoop-2.5.2/etc/hadoop/

hadoop-env.sh 修改Java\_home地址,HADOOP\_LOG\_DIR (使用默认hadoop内logs文件夹)

yarn-env.sh 修改Java\_home地址

core-site.xml ：

<configuration>

<property>

<name>fs.defaultFS</name> --分布式文件系统url

<value>hdfs://192.168.1.51:9100</value>

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>/home/hadp/hadoop-2.5.2/tmp </value>

</property>

</configuration>

hdfs-site.xml

<configuration>

<property>

<name>dfs.namenode.name.dir</name> --存储名称节点元数据的目录（元数据：整个文件系统信息，哪些文件，由哪些数据块组成，数据块分别位于哪个节点）

<value>file:/home/hadp/hadoop-2.5.2/name</value> --此文件夹为预先创建的

</property>

<property>

<name>dfs.datanode.data.dir</name> --数据节点存储块的目录

<value>file:/home/hadp/hadoop-2.5.2/data</value> --此文件夹为预先创建的

</property>

<property>

<name>dfs.replication</name> --分布式文件系统数据复制多少份

<value>2</value> --有两个节点，复制两份

</property>

<property>

<name>dfs.namenode.secondary.http-address</name> --备份名称节点位置

<value>192.168.1.51:50090</value>

</property>

</configuration>

mapred-site.xml(不用处理此文件，设置为Yarn后，测试wordcount失败，新版本可以忽略此文件)

新版本找不到此文件，复制重命名

$ cp mapred-site.xml.template mapred-site.xml

<configuration>

<property>

<name>mapreduce.framework.name</name> --配置其使用 Yarn 框架执行 map-reduce 处理程序

<value>Yarn</value>

</property>

</configuration>

MapReduce新框架配置文件yarn-site.xml

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.resourcemanager.address</name> --NodeManager 与 RM 通信的接口地址

<value>192.168.1.51:8032</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.address</name> --NodeManger需要知道RM主机的scheduler调度服务接口地址

<value>192.168.1.51:8030</value>

</property>

<property>

<name>yarn.resourcemanager.webapp.address</name> --各个 task 的资源调度及运行状况通过通过该 web 界面访问

<value>192.168.1.51:8088</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address</name> --NodeManager 需要向 RM 报告任务运行状态供 Resouce 跟踪，因此 NodeManager 节点主机需要知道 RM 主机的 tracker 接口地址

<value>192.168.1.51:8031</value>

</property>

</configuration>

10.masters 记录名称节点和jobtraker存放主机名

slaves 记录数据节点 datanode 和 tasktraker 节点主机名

新版本没有master文件了

$ vi slaves

输入两个数据节点

gdfwdb02

gdfwdb03

11.都配置好后，跨节点复制，将Hadoop文件夹复制到另外两台机器(此处就用到了ssh免密码登录)

$ scp -r(拷贝整个目录) ./hadoop-2.5.2 gdfwdb02:/home/hadp

$ scp -r(拷贝整个目录) ./hadoop-2.5.2 gdfwdb03:/home/hadp

七．启动hadoop集群

12.格式化名称节点，在Hadoop安装目录下（只需在主节点启动）

$ bin/hadoop namenode -format

13.启动（只需在主节点启动）

hadoop-2.5.2/sbin/start（stop）-dfs.sh --分布式文件系统启动

hadoop-2.5.2/sbin/start(stop)-yarn.sh --yarn启动(MapReduce2)

或者

hadoop-2.5.2/sbin/start(stop)-all.sh --启动两个（用这个居然导致子节点NodeManager进程没启动，搞不懂为啥）

验证dfs：http://localhost:50070

14.进程检查

启动后检查进程是否都已启动：进jdk/bin $ jps （观看跟Java有关的进程信息）

主节点：

进程号 ResourceManager (yarn启动后有）

进程号 SecondaryNameNode (dfs 启动后有)

进程号 NameNode (dfs 启动后有)

slaves:

进程号 DataNode (dfs 启动后有)

进程号 NodeManager (yarn启动后有）

子节点启动报错：“Incompatible clusterIDs”的错误原因是在执行“hdfs namenode -format”之前，

没有清空DataNode节点的data目录。

解决办法:

直接删除data目录，再创建data目录

或者

　　1、在hdfs-site.xml配置文件中，配置了dfs.namenode.name.dir，在master中,该配置的目录下有个current文件夹，里面有个VERSION文件,内容如下:

　　#Thu Mar 13 10:51:23 CST 2014

　　namespaceID=1615021223

　　clusterID=CID-8e201022-6faa-440a-b61c-290e4ccfb006

　　cTime=0

　　storageType=NAME\_NODE

　　blockpoolID=BP-1257313099-10.10.208.38-1394679083528

　　layoutVersion=-40

　　2、在core-site.xml配置文件中，配置了hadoop.tmp.dir，在slave中,该配置的目录下有个dfs/data/current目录，里面也有一个VERSION文件，内容

　　#Wed Mar 12 17:23:04 CST 2014

　　storageID=DS-414973036-10.10.208.54-50010-1394616184818

　　clusterID=clustername

　　cTime=0

　　storageType=DATA\_NODE

　　layoutVersion=-40

八．Hdfs操作

15.操作文件系统（在bin目录下）：

#创建目录

./hadoop fs -mkdir /test (从/开始，否则找不到文件或目录)

#查看到目录：

./hadoop fs -ls /

Found 1 items

drwxr-xr-x - hadoop supergroup 0 2013-04-27 02:32 /test

#查看所有目录

./hadoop fs -ls -R /

查看test下所有文件

./hadoop fs -ls /test/\*\

#删除文件（删除后，block空间被释放）

./hadoop fs -rm /test/input/test1.txt

#删除目录

./hadoop fs -rm -r /test/input

#将文件拷贝出来 源 目标

./hadoop fs -copyToLocal /test/input/test1.txt test1.copy.txt

或者

./hadoop fs -get /test/input/test1.txt test1.copy.txt

#查看hdfs基本统计信息

./hadoop dfsadmin -report

总空间，用了多少，使用百分百等等

各个节点情况等

#进入和退出安全模式

./hadoop dfsadmin -safemode enter

./hadoop dfsadmin -safemode leave

#准备文件，在~下创建目录，新建两个文本

$ mkdir input

$ cd input

$ echo "hello world" > test1.txt (echo输出，>输出定向符，将要输出在屏幕上的词输出在test1.txt)

$ echo "hello piter" > test2.txt

$ cat test1.txt

hello world

#回到Hadoop安装目录下bin里面，将input里面的文本拷贝至hdfs，存为in文件夹

(放置文件前必须关闭所有节点防火墙，service iptables stop 或 chkconfig iptables on/off)：

$ ./hadoop fs -put ../../input /in

$ ./hadoop fs -ls /in

-rw-r--r-- 2 hadp supergroup 12 2015-07-30 14:58 /in/test1.txt

-rw-r--r-- 2 hadp supergroup 12 2015-07-30 14:58 /in/test2.txt

16.hello world，调用mapreduce测试程序wordcount，统计in目录下文件，统计结果输出至out

$ ./hadoop jar ../share/hadoop/mapreduce/hadoop-mapreduce-examples-2.5.2.jar wordcount /in /out

(Hadoop自带wordcount例子程序，路径在Hadoop安装目录下share/hadoop/mapreduce/hadoop-mapreduce-examples-2.5.2.jar)

屏幕会输出一堆INFO

查看统计后生成的文件

$ ./hadoop fs -ls /out

Found 2 items

-rw-r--r-- 2 hadp supergroup 0 2015-07-30 15:11 /out/\_SUCCESS

-rw-r--r-- 2 hadp supergroup 24 2015-07-30 15:11 /out/part-r-00000

查看统计结果

$ ./hadoop fs -cat /out/\*

hello 2

piter 1

world 1

web中查看文件系统

http://192.168.1.51:50070

Utilities / Browse the filesystem

在输入框里输/in ，点击go，会列出文件

All Applications

http://192.168.1.51:8088

在Linux里找到我们放入的文件，在配置文件中，有一项配置数据文件目录，hdfs-site.xml dfs.datanode.data.dir

到datanode里，02或03均可以看到（dfs.replication副本数，设置为2）

[hadp@gdfwdb02 data]$ ls -lR（列出所有文件，current，blk\_xxxx记录写数据，blk\_xxxx.meta记录元数据）

HDFS设计基础：冗余

文件一经写入，不允许修改，只能删掉重写（不会存在修改冲突问题）

Namenode管理文件系统命名空间，记录每个文件数据块在各个datanode上的位置和副本信息，

使用事务日志记录HDFS元数据的变化。使用映像文件存储文件系统的命名空间，包括文件映射，文件属性等。元数据配置路径。

Fsimage映像文件

DataNode负责所在物理节点的存储管理。典型的块大小是64MB

hdfs数据保护措施：

机架感知：本机架存放一个副本，在其他机架再存放别的副本，防止机架失效时丢失数据，也可以提高宽带利用率

心跳机制：周期性从datanode接受心跳信号和块报告，根据报告验证元数据，没有按时发送的datanode被标记为宕机，

不再给它请求；datanode失效造成副本数量下降，并且低于预先设置的阈值，namenode会检测出这些数据块，并在

合适的时间进行重新复制。引发复制的原因还包括：副本本身损坏、磁盘错误、复制因子增大。

安全模式：namenode启动时会先经过安全模式阶段，收集各个datanode的报告，数据块达到最小副本数以上时，任务安全

校验和：文件创立时，每个数据块都产生校验和(dncp\_block\_verification.log.curr)，根据读取数据是否与校验和一致，

判断数据块是否损坏，转而读取其他副本

回收站：删除文件时，其实是放入回收站，可以还原，可以设置一个时间阈值，当文件在回收站里存放时间超过这个阈值，

就会彻底删除，释放占用的数据块

元数据保护：可以配置为拥有多个副本，会降低处理速度，但增加安全性

快照：数据重返时间点状态

文件操作命令：

没有当前目录概念，没有cd命令

17.如何新增节点

新节点装好Hadoop，配置文件拷贝好，修改slaves文件（每个节点），ssh免密码配置，

单独启动该节点任务bin/xxx.sh，想把之前的数据搬到新节点，可运行start-balancer.sh进行负载均衡（重新散部花时间）

hdfs api

《Java就业培训教程》 张孝祥 --两周

《Java程序设计教程》 雍俊海 --半年

九．Java程序操作Hdfs文件

18.利用java上传本地文件到hdfs

UploadFile.java

import java.io.BufferedInputStream;

import java.io.FileInputStream;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.URI;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IOUtils;

public class UploadFile {

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// TODO Auto-generated method stub

try {

String localStr = args[0]; //第一个参数，本地文件路径

String dst = args[1];//第二个参数，hadoop文件系统路径

//in对应的是本地文件系统的目录

InputStream in = new BufferedInputStream(new FileInputStream(localStr));

Configuration conf = new Configuration();

//获得hadoop系统的连接

FileSystem fs = FileSystem.get(URI.create(dst),conf);

//out对应Hadoop文件系统输出目录

OutputStream out = fs.create(new Path(dst));

IOUtils.copyBytes(in, out, 4096,true);

System.out.println("success");

} catch (Exception e) {

System.out.println(e.toString());

}

}

}

编译java代码UploadFile.class,注意用到的jar包(java文件放到hadp/input内)

[hadp@gdfwdb01 input]$ javac -classpath ../hadoop-2.5.2/share/hadoop/common/hadoop-common-2.5.2.jar UploadFile.java

出现警告信息：无法找到类型为org.apache.hadoop.classification.InterfaceAudience.LimitedPrivate 的注释方法value

需添加另一个jar包

[hadp@gdfwdb01 input]$ javac -classpath ../hadoop-2.5.2/share/hadoop/common/hadoop-common-2.5.2.jar:../hadoop-2.5.2/share/hadoop/common/lib/hadoop-annotations-2.5.2.jar UploadFile.java

配置hadoop-env.sh（hadoop-2.5.2/etc/hadoop），设置本地class文件路径，新加

export HADOOP\_CLASSPATH=/home/hadp/input

利用UploadFile往hdfs转移文件

[hadp@gdfwdb01 input]$ hadoop UploadFile my.txt /in/my.txt

命令ls|wc -l 可统计当前目录下有多少个文件

zcat xxx.gz |less 查看压缩文件里的内容 |less 翻屏

zcat \*.gz > a 把所有压缩文件输出到a文件内

十．Mapreduce程序示例

mapreduce "分而治之"

Mapper负责"分"，将复杂任务分解为若干个"简单的任务"执行

Reducer 对map阶段的结果进行汇总 。数目由mapred-site.xml配置项mapred.reduce.tasks决定，缺省值为1，可以覆盖为其他值

Shuffler Mapper和Reducer中间步骤（可选），把mapper输出的数据进行压缩，对数据进行预处理，使Reducer更快

计数器

enum Counter{

LINESKIP //出错的行

}

extends Mapper <LongWritable,Text,Text,IntWritable> 参数意义：输入key格式，输入value格式，输出key格式，输出value格式

对应java格式：LongWritable-long,Text-String,IntWritable-int，NullWritable-null

方法map(LongWritable key, Text value, 与Mapper前两个参数保持一致（key:行号，value:此行数据内容）

Context context)

context.write(NullWritable.get(),out);默认输出是key \t value，为了去掉制表符，只返回我们设置的格式，将key写作null

context.getCounter(Counter.LINESKIP).increment(1);//出错令计数器+1

方法run

extends Reducer<Text,Text,Text,Text> （输入key，输入value 匹配Mapper的输出格式）

## 示例程序1：Count.java

package first;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class Count extends Configured implements Tool {

enum Counter{

LINESKIP

}

/\*\*

\*

\* @author 皮

\* 模拟数据 按被拨打号码归类

\* 10086 18901034234|13798803423|

\* 18901034234 10086

\* 13798803423 10086

\* 13520908343 110

\*/

public static class MyMap extends Mapper<LongWritable,Text,Text,Text>{

public void map(LongWritable key,Text value,Context cont)

throws IOException,InterruptedException {

String line=value.toString();

try{

String[] sz=line.split(" ");

String aNum=sz[0];

String bNum=sz[1];

cont.write(new Text(bNum), new Text(aNum));

}catch(ArrayIndexOutOfBoundsException ex){

cont.getCounter(Counter.LINESKIP).increment(1);//错误计数器加1

}

}

}

public static class MyReduce extends Reducer<Text,Text,Text,Text >{

public void reduce(Text key,Iterable<Text> values //迭代器

,Context context)

throws IOException,InterruptedException {

String valueStr="";

for(Text value:values){

valueStr+=value.toString()+"|";

}

context.write(key, new Text(valueStr));

}

}

public int run(String[] arg0) throws Exception {

// TODO Auto-generated method stub

Configuration conf=getConf();

Job job=new Job(conf);

job.setJobName("Count");

job.setJarByClass(Count.class);

FileInputFormat.addInputPath(job, new Path(arg0[0])); //输入路径

FileOutputFormat.setOutputPath(job, new Path(arg0[1]));//输出路径

job.setMapperClass(MyMap.class);

// job.setCombinerClass(LongSumReducer.class);

job.setReducerClass(MyReduce.class);

// job.setMapOutputKeyClass(Text.class);//有Reducer不用设置map输出key类型

// job.setMapOutputValueClass(Text.class);//map输出value类型

job.setOutputKeyClass(Text.class); //reduce输出key类型

job.setOutputValueClass(Text.class);//reduce输出value类型

job.waitForCompletion(true);

return job.isSuccessful()?0:1;

}

/\*\*

\* @param args

\* @throws Exception

\*/

public static void main(String[] args) throws Exception {

// TODO Auto-generated method stub

int ext = ToolRunner.run(new Configuration(),new Count(), args);

System.exit(ext);//System.exit(0)是正常退出程序，非0表示非正常退出程序

}

}

## 示例程序2；WordCount.java

**package** first;

**import** java.io.IOException;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.conf.Configured;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.io.IntWritable;

**import** org.apache.hadoop.io.LongWritable;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.mapreduce.Job;

**import** org.apache.hadoop.mapreduce.Mapper;

**import** org.apache.hadoop.mapreduce.Reducer;

**import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

**import** org.apache.hadoop.util.Tool;

**import** org.apache.hadoop.util.ToolRunner;

**public** **class** WordCount **extends** Configured **implements** Tool{

/\*\*

\*

\* **@author** 皮

\* 模拟数据,统计每个单词的个数，wordCount

\* text mont key text ok mont

\*/

**public** **static** **class** MyCountMap **extends** Mapper<LongWritable,Text,Text,IntWritable>{

**public** **void** map(LongWritable key,Text value,Context cont) **throws** IOException, InterruptedException{

String[] itr = **new** String(value.toString()).split("\\s+"); // \\s表示 空格,回车,换行等空白符,+号表示一个或多个的意思

Text word = **new** Text();

IntWritable one = **new** IntWritable(1);

**for**(String w:itr){

cont.write(**new** Text(w), one);

}

}

}

**public** **static** **class** MyCountReduce **extends** Reducer<Text,IntWritable,Text,IntWritable>{

**public** **void** reduce(Text key,Iterable<IntWritable> values,Context cont) **throws** IOException, InterruptedException{

**int** sum=0;

**for**(IntWritable val:values){

sum+=val.get();

}

cont.write(key, **new** IntWritable(sum));

}

}

@Override

**public** **int** run(String[] args) **throws** Exception {

Configuration conf = **new** Configuration();

**if** (args.length < 2) {

System.*err*.println("Usage: wordcount <in> [<in>...] <out>");

System.*exit*(2);

}

Job job = **new** ~~Job~~(conf, "word count");

job.setJarByClass(WordCount.**class**);

job.setMapperClass(MyCountMap.**class**);

job.setReducerClass(MyCountReduce.**class**);

job.setOutputKeyClass(Text.**class**);

job.setOutputValueClass(IntWritable.**class**);

FileInputFormat.*addInputPath*(job, **new** Path(args[0]));

FileOutputFormat.*setOutputPath*(job,**new** Path(args[1]));

**return** job.waitForCompletion(**true**) ? 0 : 1;

}

**public** **static** **void** main(String[] args) **throws** Exception {

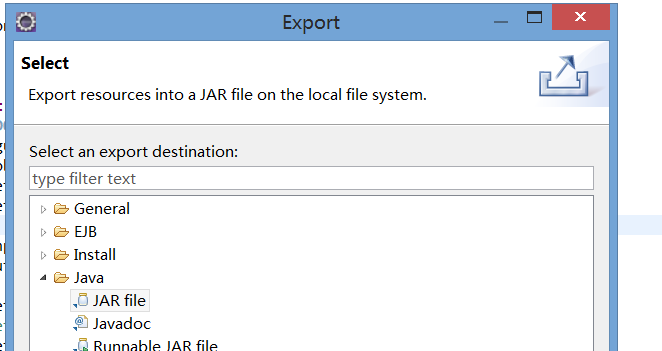
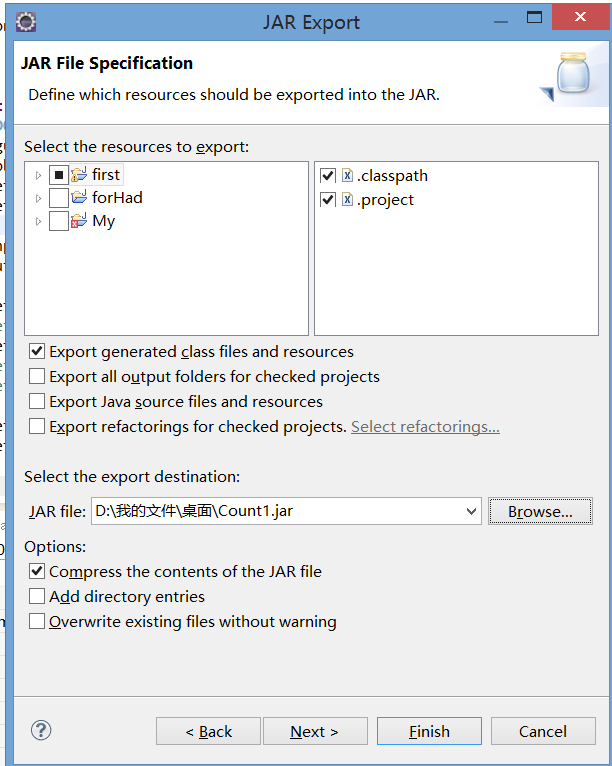
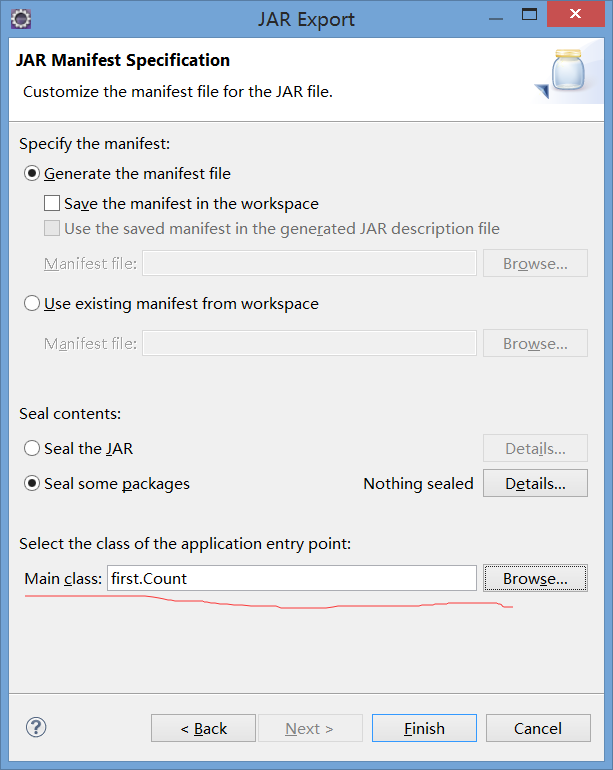
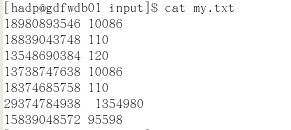
**int** ext=ToolRunner.*run*(**new** Configuration(), **new** WordCount(), args);

System.*exit*(ext);

}

}

## 运行

1. 导出成jar包
2. 
3. 
4. 
5. Ssh上传到linux，准备数据
6. 
7. 放入hdfs，
8. 调用jar包计算，将结果保存到out目录
9. 
10. 处理完成
11. 