Hadoop环境安装

1. 下载相关软件

VMware 安装

Centos 安装

Xshell/XFTP 安装

Jdk linux

Hadoop

1. 虚拟机ip配置

vim /etc/hosts

192.168.137.132 Master.Hadoop 5

192.168.137.129 Slave1.Hadoop

192.168.137.130 Slave2.Hadoop

使用以下指令对master主机中进行测试，可使用类似指令在slave主机测试

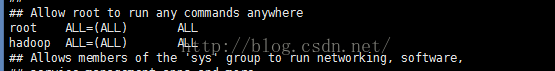
ping Slave1.Hadoop

ping Slave2.Hadoop

1. 配置hadoop用户的权限

为hadoop用户配置root权限，让其可运行root命令

vim /etc/sudoers



cat /etc/passwd 查看用户权限

1. 安装SSH无密码访问

在master机器上执行

* 输入以下指令生成ssh

ssh-keygen // 会生成两个文件，放到默认的/root/.ssh/文件夹中

* 把id\_rsa.pub追加到授权的key里面去

cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys

* 修改文件”authorized\_keys”权限

chmod 600 ~/.ssh/authorized\_keys

* 设置SSH配置

vi /etc/ssh/sshd\_config

// 以下三项修改成以下配置

RSAAuthentication yes # 启用 RSA 认证

PubkeyAuthentication yes # 启用公钥私钥配对认证方式

AuthorizedKeysFile .ssh/authorized\_keys # 公钥文件路径（和上面生成的文件同）

* 重启ssh服务

service sshd restart

把公钥复制所有的Slave机器上

// scp ~/.ssh/id\_rsa.pub 远程用户名@远程服务器IP:~/

scp ~/.ssh/id\_rsa.pub root@192.168.137.129:~/

scp ~/.ssh/id\_rsa.pub root@192.168.137.132:~/

在slave机器上执行

* 在slave主机上创建.ssh文件夹

mkdir ~/.ssh

// 修改权限

chmod 700 ~/.ssh

* 追加到授权文件”authorized\_keys”

cat ~/id\_rsa.pub >> ~/.ssh/authorized\_keys

// 修改权限

chmod 600 ~/.ssh/authorized\_keys

* 删除无用.pub文件

rm –r ~/id\_rsa.pub

在master主机下进行测试

ssh 192.168.137.129

ssh 192.168.137.130

// 如果能够分别无密码登陆slave1, slave2主机，则成功配置

1. 上传并安装jdk到虚拟机

* 在/usr下创建java文件夹
* 使用以下指令从真机传入jdk-8u45-linux-x64.tar.gz文件到虚拟主机中

// 需要在真机中进入文件目录，地址为虚拟主机的ip地址

scp jdk-8u45-linux-x64.tar.gz root@192.168.1.122:/usr/java

scp jdk-8u45-linux-x64.tar.gz root@192.168.1.125:/usr/java

scp jdk-8u45-linux-x64.tar.gz root@192.168.1.124:/usr/java

* 使用以下指令进行加压

tar zxvf jdk-8u45-linux-x64.tar.gz

// 解压后可以删除掉gz文件

rm jdk-8u45-linux-x64.tar.gz

* 配置jdk环境变量

vi /etc/profile

// 将以下数据复制到文件底部

export JAVA\_HOME=/usr/java/jdk1.8.0\_45

export JRE\_HOME=/usr/java/jdk1.8.0\_45/jre

export CLASSPATH=.:$CLASSPATH:$JAVA\_HOME/lib:$JRE\_HOME/lib

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

#java Path

export JAVA\_HOME=/usr/java/jdk1.8.0\_131

export JRE\_HOME=/usr/java/jdk1.8.0\_131/jre

export CLASSPATH=.:$CLASSPATH:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar:$JRE\_HOME/lib/rt.jar

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

* 使配置生效

source /etc/profile

* 验证安装成功

java -version

// 如果出现以下信息，则配置成功

java version "1.8.0\_45"

Java(TM) SE Runtime Environment (build 1.8.0\_45-b14)

Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)

1. 上传并安装hadoop到虚拟机

* 使用以下指令将hadoop-2.7.0.tar.gz文件复制到/usr目录下

scp hadoop-2.7.0.tar.gz [root@192.168.1.122:/usr/](mailto:root@192.168.1.122:/usr/)

* 解压hadoop-2.7.0.tar.gz文件，并重命名

cd /usr

tar zxvf hadoop-2.7.0.tar.gz

mv hadoop-2.7.0 hadoop

// 删除hadoop-2.7.0.tar.gz文件

rm –rf hadoop-2.7.0.tar.gz

* 在”/usr/hadoop”下面创建tmp文件夹

cd /usr/hadoop

mkdir tmp

* 把Hadoop的安装路径添加到”/etc/profile”中

vi /etc/profile

// 将以下数据加入到文件末尾

export HADOOP\_INSTALL=/usr/hadoop

export PATH=${HADOOP\_INSTALL}/bin:${HADOOP\_INSTALL}/sbin${PATH}

export HADOOP\_MAPRED\_HOME=${HADOOP\_INSTALL}

export HADOOP\_COMMON\_HOME=${HADOOP\_INSTALL}

export HADOOP\_HDFS\_HOME=${HADOOP\_INSTALL}

export YARN\_HOME=${HADOOP\_INSTALLL}

export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=${HADOOP\_INSTALL}/lib/natvie

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

* 重启”/etc/profile”

source /etc/profile

配置hadoop（先只在Master主机配置，配置完后传入两个Slave主机）

* 设置hadoop-env.sh和yarn-env.sh中的java环境变量

cd /usr/hadoop/etc/hadoop/

vi hadoop-env.sh

// 修改JAVA\_HOME

export JAVA\_HOME=/usr/java/jdk1.8.0\_45

* 配置core-site.xml文件

vi core-site.xml

// 修改文件内容为以下

<configuration>

<property>

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

<value>/usr/hadoop/tmp</value>

<description>A base for other temporary directories.</description>

</property>

<property>

<name>fs.default.name</name>

<value>hdfs://Master.Hadoop:9000</value>

</property>

</configuration>

* 配置hdfs-site.xml文件

vi hdfs-site.xml

// 修改文件内容为以下

<configuration>

<property>

<name>dfs.namenode.name.dir</name>

<value>file:///usr/hadoop/dfs/name</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>file:///usr/hadoop/dfs/data</value>

</property>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.nameservices</name>

<value>hadoop-cluster1</value>

</property>

<property>

<name>dfs.namenode.secondary.http-address</name>

<value>Master.Hadoop:50090</value>

</property>

<property>

<name>dfs.webhdfs.enabled</name>

<value>true</value>

</property>

</configuration>

* 配置mapred-site.xml文件

vi mapred-site.xml

// 修改文件为以下

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

<final>true</final>

</property>

<property>

<name>mapreduce.jobtracker.http.address</name>

<value>Master.Hadoop:50030</value>

</property>

<property>

<name>mapreduce.jobhistory.address</name>

<value>Master.Hadoop:10020</value>

</property>

<property>

<name>mapreduce.jobhistory.webapp.address</name>

<value>Master.Hadoop:19888</value>

</property>

<property>

<name>mapred.job.tracker</name>

<value>http://Master.Hadoop:9001</value>

</property>

* 配置yarn-site.xml文件

vi yarn-site.xml

// 修改文件内容为以下

<property>

<name>yarn.resourcemanager.hostname</name>

<value>Master.Hadoop</value>

</property>

<property>

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

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.resourcemanager.address</name>

<value>Master.Hadoop:8032</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.address</name>

<value>Master.Hadoop:8030</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address</name>

<value>Master.Hadoop:8031</value>

</property>

<property>

<name>yarn.resourcemanager.admin.address</name>

<value>Master.Hadoop:8033</value>

</property>

<property>

<name>yarn.resourcemanager.webapp.address</name>

<value>Master.Hadoop:8088</value>

</property>

### 配置Hadoop的集群

* 将Master中配置好的hadoop传入两个Slave中

scp -r /usr/hadoop root@192.168.1.125:/usr/

scp -r /usr/hadoop [root@192.168.1.124:/usr/](mailto:root@192.168.1.124:/usr/)

* 修改Master主机上的slaves文件

cd /usr/hadoop/etc/hadoop

vi slaves

// 将文件内容修改为

Slave1.Hadoop

Slave2.Hadoop

* 格式化HDFS文件系统

// 在Master主机上输入以下指令

hadoop namenode –format

* 启动hadoop

// 关闭机器防火墙

service iptables stop

**systemctl   stop   firewalld.service**

cd /usr/hadoop/sbin

./start-all.sh

// 更推荐的运行方式：

cd /usr/hadoop/sbin

./start-dfs.sh

./start-yarn.sh

应该输出以下信息：

Starting namenodes on [Master.Hadoop]

Master.Hadoop: starting namenode, logging to /usr/hadoop/logs/hadoop-root-namenode-localhost.localdomain.out

Slave2.Hadoop: starting datanode, logging to /usr/hadoop/logs/hadoop-root-datanode-Slave2.Hadoop.out

Slave1.Hadoop: starting datanode, logging to /usr/hadoop/logs/hadoop-root-datanode-Slave1.Hadoop.out

starting yarn daemons

starting resourcemanager, logging to /usr/hadoop/logs/yarn-root-resourcemanager-localhost.localdomain.out

Slave1.Hadoop: starting nodemanager, logging to /usr/hadoop/logs/yarn-root-nodemanager-Slave1.Hadoop.out

Slave2.Hadoop: starting nodemanager, logging to /usr/hadoop/logs/yarn-root-nodemanager-Slave2.Hadoop.out

* 验证hadoop

// 1. 直接在Master或Slave输入指令：

jps

// 应该输出以下信息(端口号仅供参考)

Master:

3930 ResourceManager

4506 Jps

3693 NameNode

Slave:

2792 NodeManager

2920 Jps

2701 DataNode

// 2. 输入以下指令

hadoop dfsadmin -report

// 应该输出以下信息：

Configured Capacity: 14382268416 (13.39 GB)

Present Capacity: 10538565632 (9.81 GB)

DFS Remaining: 10538557440 (9.81 GB)

DFS Used: 8192 (8 KB)

DFS Used%: 0.00%

Under replicated blocks: 0

Blocks with corrupt replicas: 0

Missing blocks: 0

Missing blocks (with replication factor 1): 0

-------------------------------------------------

Live datanodes (2):

Name: 192.168.1.124:50010 (Slave2.Hadoop)

Hostname: Slave2.Hadoop

Decommission Status : Normal

Configured Capacity: 7191134208 (6.70 GB)

DFS Used: 4096 (4 KB)

Non DFS Used: 1921933312 (1.79 GB)

DFS Remaining: 5269196800 (4.91 GB)

DFS Used%: 0.00%

DFS Remaining%: 73.27%

Configured Cache Capacity: 0 (0 B)

Cache Used: 0 (0 B)

Cache Remaining: 0 (0 B)

Cache Used%: 100.00%

Cache Remaining%: 0.00%

Xceivers: 1

Last contact: Thu Jul 02 10:45:04 CST 2015

Name: 192.168.1.125:50010 (Slave1.Hadoop)

Hostname: Slave1.Hadoop

Decommission Status : Normal

Configured Capacity: 7191134208 (6.70 GB)

DFS Used: 4096 (4 KB)

Non DFS Used: 1921769472 (1.79 GB)

DFS Remaining: 5269360640 (4.91 GB)

DFS Used%: 0.00%

DFS Remaining%: 73.28%

Configured Cache Capacity: 0 (0 B)

Cache Used: 0 (0 B)

Cache Remaining: 0 (0 B)

Cache Used%: 100.00%

Cache Remaining%: 0.00%

Xceivers: 1

* 访问网页
* 操作系统环境：[CentOS](http://www.linuxidc.com/topicnews.aspx?tid=14) Linux release 7.0.1406(Core) 64位  
  CentOS 7.0默认使用的是firewall作为防火墙，这里改为iptables防火墙步骤。
* 1、关闭firewall：  
  systemctl stop firewalld.service #停止firewall  
  systemctl disable firewalld.service #禁止firewall开机启动  
  firewall-cmd --state #查看默认防火墙状态（关闭后显示notrunning，开启后显示running）
* 2、iptables防火墙（这里iptables已经安装，下面进行配置）  
  vi/etc/sysconfig/iptables #编辑防火墙配置文件  
  # sampleconfiguration for iptables service  
  # you can edit thismanually or use system-config-firewall  
  # please do not askus to add additional ports/services to this default configuration  
  \*filter  
  :INPUT ACCEPT [0:0]  
  :FORWARD ACCEPT[0:0]  
  :OUTPUT ACCEPT[0:0]  
  -A INPUT -m state--state RELATED,ESTABLISHED -j ACCEPT  
  -A INPUT -p icmp -jACCEPT  
  -A INPUT -i lo -jACCEPT  
  -A INPUT -p tcp -mstate --state NEW -m tcp --dport 22 -j ACCEPT  
  -A INPUT -p tcp -m state --state NEW -m tcp --dport 80 -jACCEPT  
  -A INPUT -p tcp -m state --state NEW -m tcp --dport 8080-j ACCEPT  
  -A INPUT -j REJECT--reject-with icmp-host-prohibited  
  -A FORWARD -jREJECT --reject-with icmp-host-prohibited  
  COMMIT  
  :wq! #保存退出
* 备注：这里使用80和8080端口为例。\*\*\*部分一般添加到“-A INPUT -p tcp -m state --state NEW -m tcp--dport 22 -j ACCEPT”行的上面或者下面，切记不要添加到最后一行，否则防火墙重启后不生效。  
  systemctlrestart iptables.service #最后重启防火墙使配置生效  
  systemctlenable iptables.service #设置防火墙开机启动

// CentOS7中iptables这个服务的配置文件没有了，采用了新的firewalld

// 输入以下指令后，可以在真机浏览器上访问hadoop网页

systemctl stop firewalld

// 输入以下网页，进入hadoop管理首页

<http://192.168.137.130:50070/dfshealth.html>

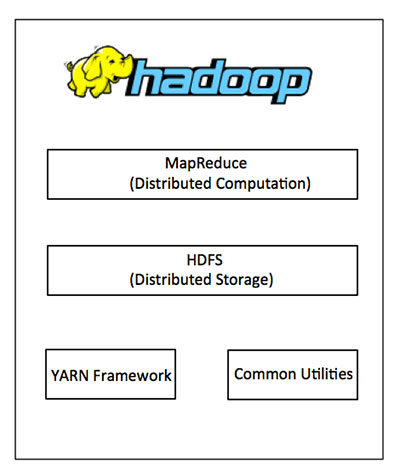
<http://192.168.137.130:8088/cluster>

hadoop jar /usr/hadoop/hadoop-2.8.0/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.0.jar wordcount /test/input/books.txt /test/output

## Hadoop的架构

在其核心，Hadoop主要有两个层次，即：

* 加工/计算层(MapReduce)，以及
* 存储层(Hadoop分布式文件系统)。



### MapReduce

MapReduce是一种并行编程模型，用于编写普通硬件的设计，谷歌对大量数据的高效处理(多TB数据集)的分布式应用在大型集群(数千个节点)以及可靠的容错方式。 MapReduce程序可在Apache的开源框架Hadoop上运行。

### Hadoop分布式文件系统

Hadoop分布式文件系统（HDFS）是基于谷歌文件系统（GFS），并提供了一个设计在普通硬件上运行的分布式文件系统。它与现有的分布式文件系统有许多相似之处。来自其他分布式文件系统的差别是显著。它高度容错并设计成部署在低成本的硬件。提供了高吞吐量的应用数据访问，并且适用于具有大数据集的应用程序。

除了上面提到的两个核心组件，Hadoop的框架还包括以下两个模块：

* Hadoop通用：这是Java库和其他Hadoop组件所需的实用工具。
* Hadoop YARN ：这是作业调度和集群资源管理的框架。

## Hadoop如何工作？

建立重配置，处理大规模处理服务器这是相当昂贵的，但是作为替代，可以联系许多普通电脑采用单CPU在一起，作为一个单一功能的分布式系统，实际上，集群机可以平行读取数据集，并提供一个高得多的吞吐量。此外，这样便宜不到一个高端服务器价格。因此使用Hadoop跨越集群和低成本的机器上运行是一个不错不选择。

Hadoop运行整个计算机集群代码。这个过程包括以下核心任务由 Hadoop 执行：

* 数据最初分为目录和文件。文件分为128M和64M（128M最好）统一大小块。
* 然后这些文件被分布在不同的群集节点，以便进一步处理。
* HDFS，本地文件系统的顶端﹑监管处理。
* 块复制处理硬件故障。
* 检查代码已成功执行。
* 执行发生映射之间，减少阶段的排序。
* 发送排序的数据到某一计算机。
* 为每个作业编写的调试日志。

## Hadoop的优势

* Hadoop框架允许用户快速地编写和测试的分布式系统。有效并在整个机器和反过来自动分配数据和工作，利用CPU内核的基本平行度。
* Hadoop不依赖于硬件，以提供容错和高可用性（FTHA），而Hadoop库本身已被设计在应用层可以检测和处理故障。
* 服务器可以添加或从集群动态删除，Hadoop可继续不中断地运行。
* Hadoop的的另一大优势在于，除了是开源的，因为它是基于Java并兼容所有的平台。

**Hadoop实例**

hadoop jar /usr/hadoop/hadoop-2.8.0/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.0.jar wordcount /user/root/lytao/test/input /user/root/lytao/test/output

Hadoop文件系统使用分布式文件系统设计开发。它是运行在普通硬件。不像其他的分布式系统，HDFS是高度容错以及使用低成本的硬件设计。

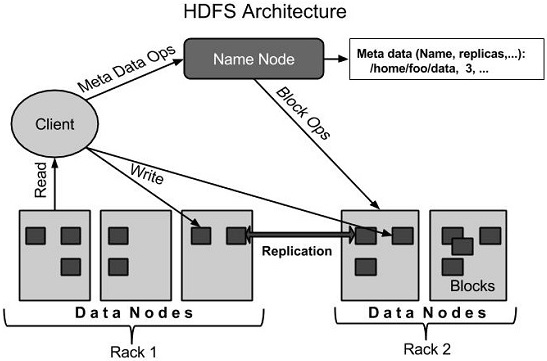
HDFS拥有超大型的数据量，并提供更轻松地访问。为了存储这些庞大的数据，这些文件都存储在多台机器。这些文件都存储以冗余的方式来拯救系统免受可能的数据损失，在发生故障时。 HDFS也使得可用于并行处理的应用程序。

## HDFS的特点

* 它适用于在分布式存储和处理。
* Hadoop提供的命令接口与HDFS进行交互。
* 名称节点和数据节点的帮助用户内置的服务器能够轻松地检查集群的状态。
* 流式访问文件系统数据。
* HDFS提供了文件的权限和验证。

## HDFS架构

下面给出是Hadoop的文件系统的体系结构。



HDFS遵循主从架构，它具有以下元素。

### 名称节点 - Namenode

名称节点是包含GNU/Linux操作系统和软件名称节点的普通硬件。它是一个可以在商品硬件上运行的软件。具有名称节点系统作为主服务器，它执行以下任务：

* 管理文件系统命名空间。
* 规范客户端对文件的访问。
* 它也执行文件系统操作，如重命名，关闭和打开的文件和目录。

### 数据节点 - Datanode

Datanode具有GNU/Linux操作系统和软件Datanode的普通硬件。对于集群中的每个节点(普通硬件/系统)，有一个数据节点。这些节点管理数据存储在它们的系统。

* 数据节点上的文件系统执行的读写操作，根据客户的请求。
* 还根据名称节点的指令执行操作，如块的创建，删除和复制。

### 块

一般用户数据存储在HDFS文件。在一个文件系统中的文件将被划分为一个或多个段和/或存储在个人数据的节点。这些文件段被称为块。换句话说，数据的HDFS可以读取或写入的最小量被称为一个块。缺省的块大小为64MB，但它可以增加按需要在HDFS配置来改变。

## HDFS的目标

* 故障检测和恢复：由于HDFS包括大量的普通硬件，部件故障频繁。因此HDFS应该具有快速和自动故障检测和恢复机制。
* 巨大的数据集：HDFS有数百个集群节点来管理其庞大的数据集的应用程序。
* 数据硬件：请求的任务，当计算发生不久的数据可以高效地完成。涉及巨大的数据集特别是它减少了网络通信量，并增加了吞吐量。

## 启动HDFS

首先，格式化配置HDFS文件系统，打开NameNode(HDFS服务器)，然后执行以下命令。

$ hadoop namenode -format

格式化HDFS后，启动分布式文件系统。以下命令将启动名称节点和数据节点的集群。

$ start-dfs.sh

## HDFS的文件列表

加载服务器信息后，使用'ls' 可以找出文件列表中的目录，文件状态。下面给出的是ls，可以传递一个目录或文件名作为参数的语法。

$ $HADOOP\_HOME/bin/hadoop fs -ls <args>

## 将数据插入到HDFS

假设在本地系统，这是所谓的file.txt文件中的数据,应当保存在HDFS文件系统。按照下面给出插入在Hadoop的文件系统所需要的文件的步骤。

### 第1步

必须创建一个输入目录。

$ $HADOOP\_HOME/bin/hadoop fs -mkdir /user/input

### 第2步

传输并使用本地系统put命令，Hadoop文件系统中存储的数据文件。

$ $HADOOP\_HOME/bin/hadoop fs -put /home/file.txt /user/input

### 第3步

可以使用ls命令验证文件。

$ $HADOOP\_HOME/bin/hadoop fs -ls /user/input

## 从HDFS中检索数据

假设在HDFS文件名为outfile。下面给出的是一个简单的演示用于检索从Hadoop文件系统所需的文件。

### 第1步

最初，使用cat命令来查看来自HDFS的数据。

$ $HADOOP\_HOME/bin/hadoop fs -cat /user/output/outfile

### 第2步

从HDFS得到文件使用get命令在本地文件系统。

$ $HADOOP\_HOME/bin/hadoop fs -get /user/output/ /home/hadoop\_tp/

## 关闭HDFS

可以使用下面的命令关闭HDFS。

$ stop-dfs.sh

在“$HADOOP\_HOME/bin/hadoop fs” 里有更多的命令。./bin/hadoop DFS 列出所有可以使用在FsShell系统上运行的命令。此外，$HADOOP\_HOME/bin/hadoop fs -help 命令名称会显示一个简短的用法。

所有表的操作如下所示。以下是使用参数一般方式：

"<path>" means any file or directory name.

"<path>..." means one or more file or directory names.

"<file>" means any filename.

"<src>" and "<dest>" are path names in a directed operation.

"<localSrc>" and "<localDest>" are paths as above, but on the local file system.

所有其他文件和路径名是指HDFS内部的对象。

|  |  |
| --- | --- |
| 1. | **ls <path>**  列出路径指定的目录中的内容，示出了名称，权限，拥有者，大小和修改日期的每个条目。 |
| 2. | **lsr <path>**  行为类似于-ls，但递归显示路径的所有子目录项。 |
| 3. | **du <path>**  显示磁盘使用率，以字节为单位，对所有的文件，这些文件匹配的路径;文件名报告使用完整HDFS协议前缀。 |
| 4. | **dus <path>**  类似-du，但打印路径中的所有文件/目录的磁盘使用情况的摘要。 |
| 5. | **mv <src><dest>**  通过移动表示src到dest，在HDFS的文件或目录。 |
| 6. | **cp <src> <dest>**  在HDFS复制确定src中的文件或目录到dest。 |
| 7. | **rm <path>**  删除文件或路径标识的空目录。 |
| 8. | **rmr <path>**  删除路径标识的文件或目录。递归删除所有子条目（例如，文件或路径的子目录）。 |
| 9. | **put <localSrc> <dest>**  从本地localSrc文件系统中的DFS标识文件或目录内复制到dest。 |
| 10. | **copyFromLocal <localSrc> <dest>**  等同于-put |
| 11. | **moveFromLocal <localSrc> <dest>**  从标识 localSrc本地文件系统中的文件或目录中HDFS复制到dest，然后删除本地副本上成功。 |
| 12. | **get [-crc] <src> <localDest>**  拷贝标识 src 来确定localDest本地文件系统路径HDFS文件或目录。 |
| 13. | **getmerge <src> <localDest>**  检索匹配的路径的src HDFS中的所有文件，并将它们复制合并文件到标识localDest本地文件系统中。 |
| 14. | **cat <filen-ame>**  显示在标准输出文件名的内容。 |
| 15. | **copyToLocal <src> <localDest>**  等同于 -get |
| 16. | **moveToLocal <src> <localDest>**  工作方式类似于-get，但删除HDFS复制成功。 |
| 17. | **mkdir <path>**  在创建一个HDFS命名的目录路径。  创建任何父目录的路径丢失（例如，命令mkdir-p在Linux中）。 |
| 18. | **setrep [-R] [-w] rep <path>**  设置标识路径代表文件的目标文件复制因子。 （实际的复制因子会向着随着时间的推移目标移动） |
| 19. | **touchz <path>**  创建在路径包含当前时间作为时间戳的文件。失败如果文件已经存在于路径，除非文件已经大小为0。 |
| 20. | **test -[ezd] <path>**  返回1，如果路径存在;长度为零;或者是一个目录，否则为0。 |
| 21. | **stat [format] <path>**  打印有关的路径信息。格式是接受块文件大小（％b），文件名（％n），块大小（%o），复制（％r）和修改日期（％y，％Y）的字符串。 |
| 22. | **tail [-f] <file2name>**  显示在标准输出文件的最后1KB。 |
| 23. | **chmod [-R] mode,mode,... <path>...**  变化符合路径标识的一个或多个对象关联的文件权限....递归执行变更与R.模式是3位八进制模式，或{augo}+/-{rwxX}。假设如果没有指定范围，则不适用umask。 |
| 24. | **chown [-R] [owner][:[group]] <path>...**  设置拥有用户和/或组标识路径的文件或目录....设置所有者递归，如果指定-R。 |
| 25. | chgrp [-R] group <path>...  设置所属组标识路径的文件或目录....设置组递归，如果指定-R。 |
| 26. | **help <cmd-name>**  返回使用上面列出的命令之一信息。必须省略了'-' 字符在cmd。 |

MapReduce它可以编写应用程序来处理海量数据，并行，大集群的普通硬件，以可靠的方式的框架。

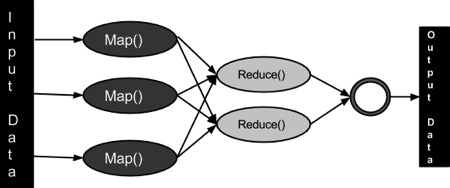
## MapReduce是什么?

MapReduce是一种处理技术和程序模型基于Java的分布式计算。 MapReduce算法包含了两项重要任务，即Map 和 Reduce。Map采用了一组数据，并将其转换成另一组数据，其中，各个元件被分解成元组(键/值对)。其次，减少任务，这需要从Map 作为输入并组合那些数据元组成的一组小的元组输出。作为MapReduce暗示的名称的序列在Map作业之后执行reduce任务。

MapReduce主要优点是，它很容易大规模数据处理在多个计算节点。下面MapReduce模型中，数据处理的原语被称为映射器和减速器。分解数据处理应用到映射器和减速器有时是普通的。但是编写MapReduce形式的应用，扩展应用程序运行在几百，几千，甚至几万机集群中的仅仅是一个配置的更改。这个简单的可扩展性是吸引了众多程序员使用MapReduce模型。

## 算法

* 通常MapReduce范例是基于向发送计算机数据的位置！
* MapReduce计划分三个阶段执行，即映射阶段，shuffle阶段，并减少阶段。
  + 映射阶段：映射或映射器的工作是处理输入数据。一般输入数据是在文件或目录的形式，并且被存储在Hadoop的文件系统（HDFS）。输入文件被传递到由线映射器功能线路。映射器处理该数据，并创建数据的若干小块。
  + 减少阶段：这个阶段是：Shuffle阶段和Reduce阶段的组合。减速器的工作是处理该来自映射器中的数据。处理之后，它产生一组新的输出，这将被存储在HDFS。
* 在一个MapReduce工作，Hadoop的发送Map和Reduce任务到集群的相应服务器。
* 框架管理数据传递例如发出任务的所有节点之间的集群周围的详细信息，验证任务完成，和复制数据。
* 大部分的计算发生在与在本地磁盘上，可以减少网络通信量数据的节点。
* 给定的任务完成后，将群集收集并减少了数据，以形成一个合适的结果，并且将其发送回Hadoop服务器。



## 输入和输出（Java透视图）

MapReduce框架上的<key, value>对操作，也就是框架视图的输入工作作为一组<key, value>对，并产生一组<key, value>对作为作业的输出可以在不同的类型。

键和值类在框架连载的方式，因此，需要实现接口。此外，键类必须实现可写，可比的接口，以方便框架排序。MapReduce工作的输入和输出类型：（输入）<k1, v1> ->映射 - ><k2, v2>-> reduce - ><k3, v3>（输出）。

|  |  |  |
| --- | --- | --- |
|  | **输入** | **输出** |
| **Map** | <k1, v1> | list (<k2, v2>) |
| **Reduce** | <k2, list(v2)> | list (<k3, v3>) |

## 术语

* **PayLoad**- 应用程序实现映射和减少功能，形成工作的核心。
* **Mapper** - 映射器的输入键/值对映射到一组中间键/值对。
* **NamedNode** - 节点管理Hadoop分布式文件系统（HDFS）。
* **DataNode** - 节点数据呈现在任何处理发生之前。
* **MasterNode** - 节点所在JobTracker运行并接受来自客户端作业请求。
* **SlaveNode** - 节点所在Map和Reduce程序运行。
* **JobTracker** - 调度作业并跟踪作业分配给任务跟踪器。
* **Task Tracker**- 跟踪任务和报告状态的JobTracker。
* **Job** -程序在整个数据集映射器和减速的执行。
* **Task** - 一个映射程序的执行或对数据的一个片段的减速器。
* **Task Attempt**- 一种尝试的特定实例在SlaveNode执行任务。

## 示例场景

下面给出是关于一个组织的电消耗量的数据。它包含了每月的用电量，各年的平均。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Jan** | **Feb** | **Mar** | **Apr** | **May** | **Jun** | **Jul** | **Aug** | **Sep** | **Oct** | **Nov** | **Dec** | **Avg** |
| 1979 | 23 | 23 | 2 | 43 | 24 | 25 | 26 | 26 | 26 | 26 | 25 | 26 | 25 |
| 1980 | 26 | 27 | 28 | 28 | 28 | 30 | 31 | 31 | 31 | 30 | 30 | 30 | 29 |
| 1981 | 31 | 32 | 32 | 32 | 33 | 34 | 35 | 36 | 36 | 34 | 34 | 34 | 34 |
| 1984 | 39 | 38 | 39 | 39 | 39 | 41 | 42 | 43 | 40 | 39 | 38 | 38 | 40 |
| 1985 | 38 | 39 | 39 | 39 | 39 | 41 | 41 | 41 | 00 | 40 | 39 | 39 | 45 |

如果上述数据作为输入，我们需要编写应用程序来处理它而产生的结果，如发现最大使用量，最低使用年份，依此类推。这是一个轻松取胜用于记录有限数目的编程器。他们将编写简单地逻辑，以产生所需的输出，并且将数据传递到写入的应用程序。

但是，代表一个特定状态下所有的大规模产业的电力消耗数据。

当我们编写应用程序来处理这样的大量数据，

* 他们需要大量的时间来执行。
* 将会有一个很大的网络流量，当我们将数据从源到网络服务器等。

为了解决这些问题，使用MapReduce框架。

### 输入数据

上述数据被保存为 sample.txt 并作为输入。输入文件看起来如下所示。

1979 23 23 2 43 24 25 26 26 26 26 25 26 25

1980 26 27 28 28 28 30 31 31 31 30 30 30 29

1981 31 32 32 32 33 34 35 36 36 34 34 34 34

1984 39 38 39 39 39 41 42 43 40 39 38 38 40

1985 38 39 39 39 39 41 41 41 00 40 39 39 45

### 示例程序

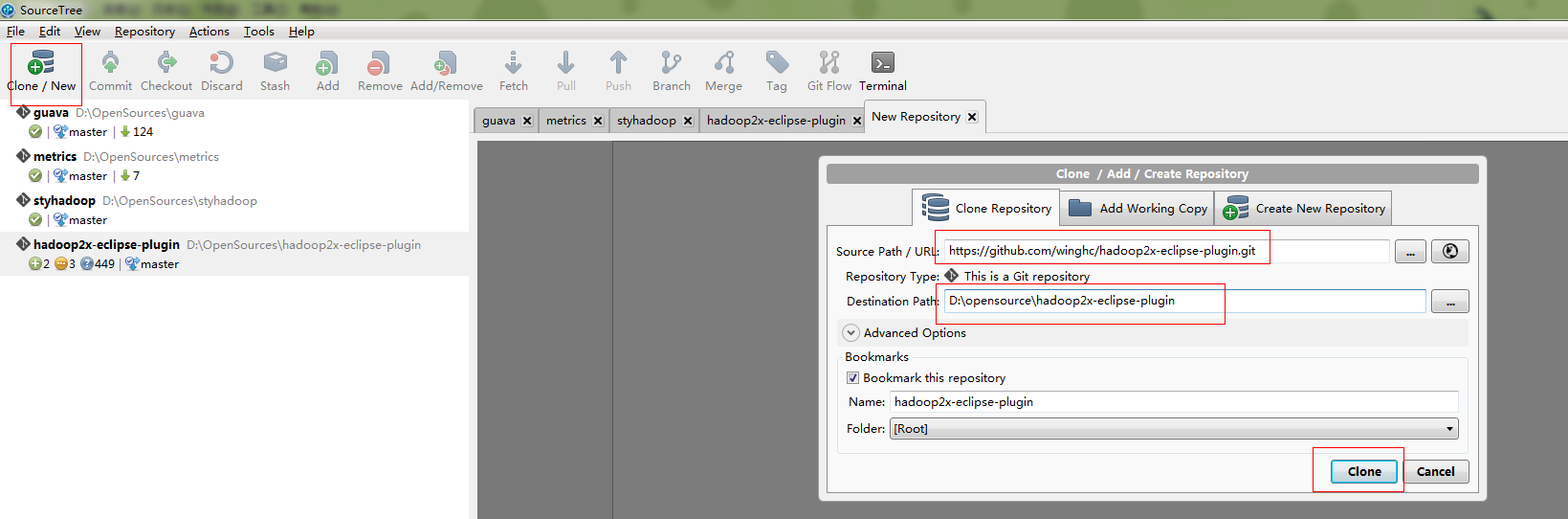
下面给出的是使用MapReduce框架的样本数据的程序。

# 0.[搭建Hadoop2.7.2开发环境1(编译Eclipse中hadoop插件hadoop2x-eclipse-plugin)](http://blog.csdn.net/fxsdbt520/article/details/50865143)

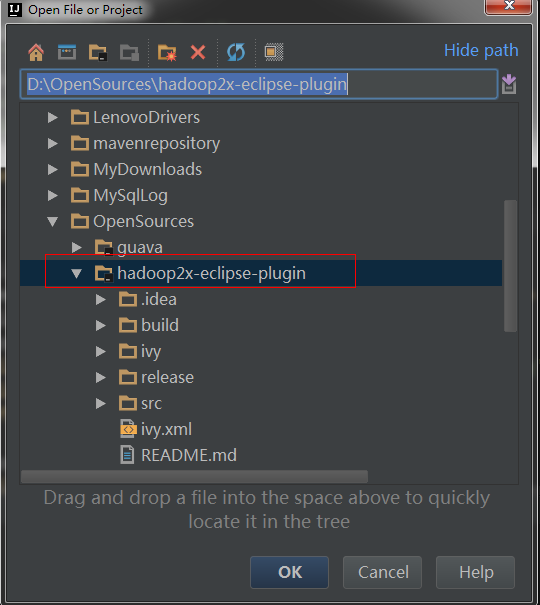
1.首先通过sourcetree获取hadoop2x-eclipse-plugin插件。

插件地址在github上:https://github.com/winghc/hadoop2x-eclipse-plugin.[**Git**](http://lib.csdn.net/base/28)

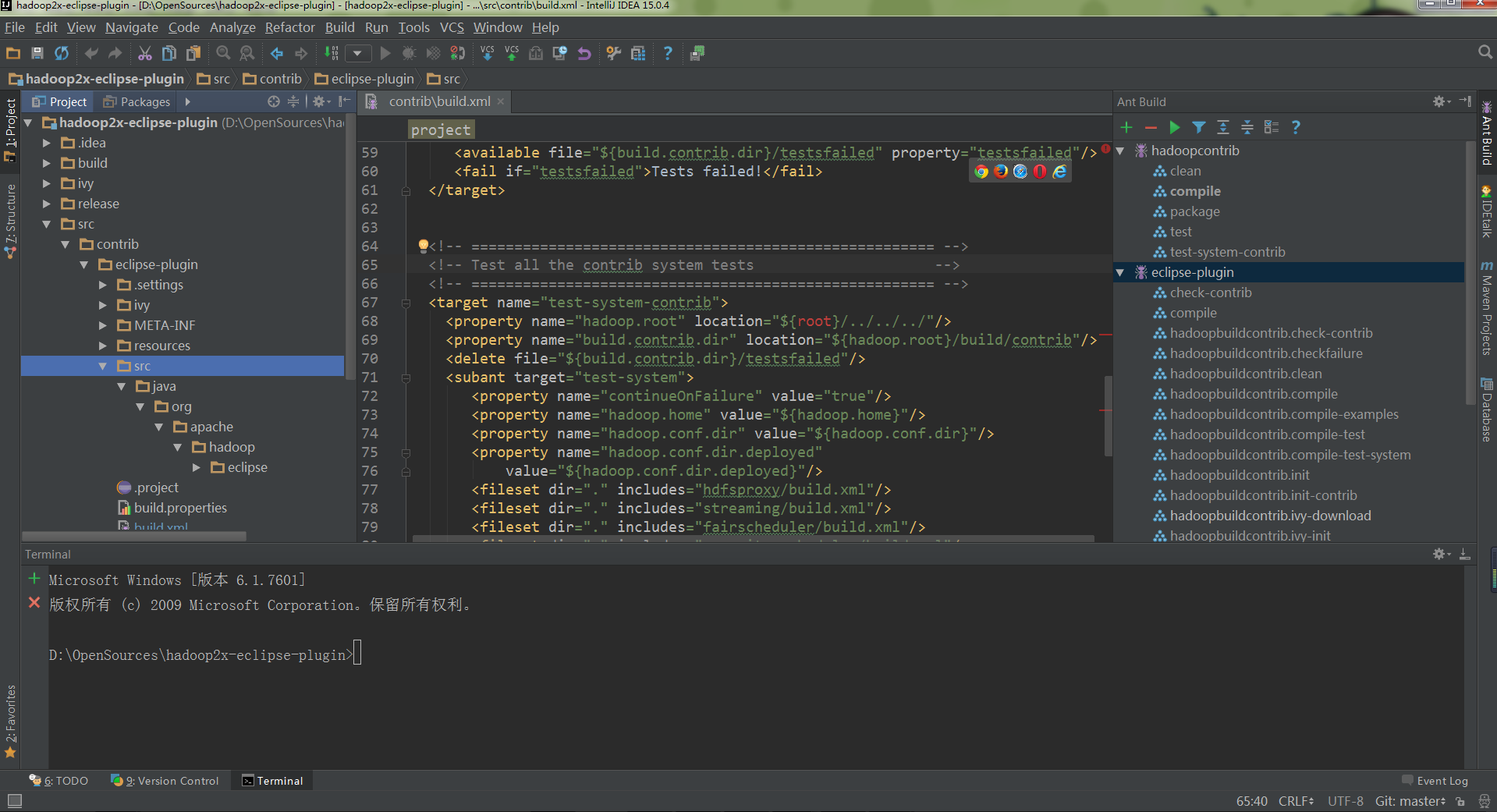
如下图：克隆上面地址到本地目录即可



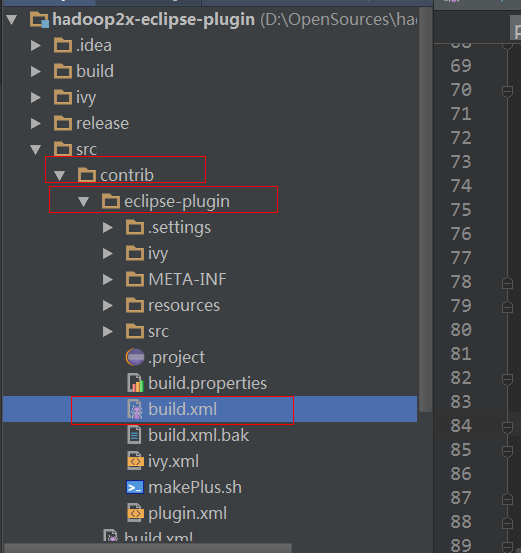
2.将克隆后的文件导入到idea中，如下图：



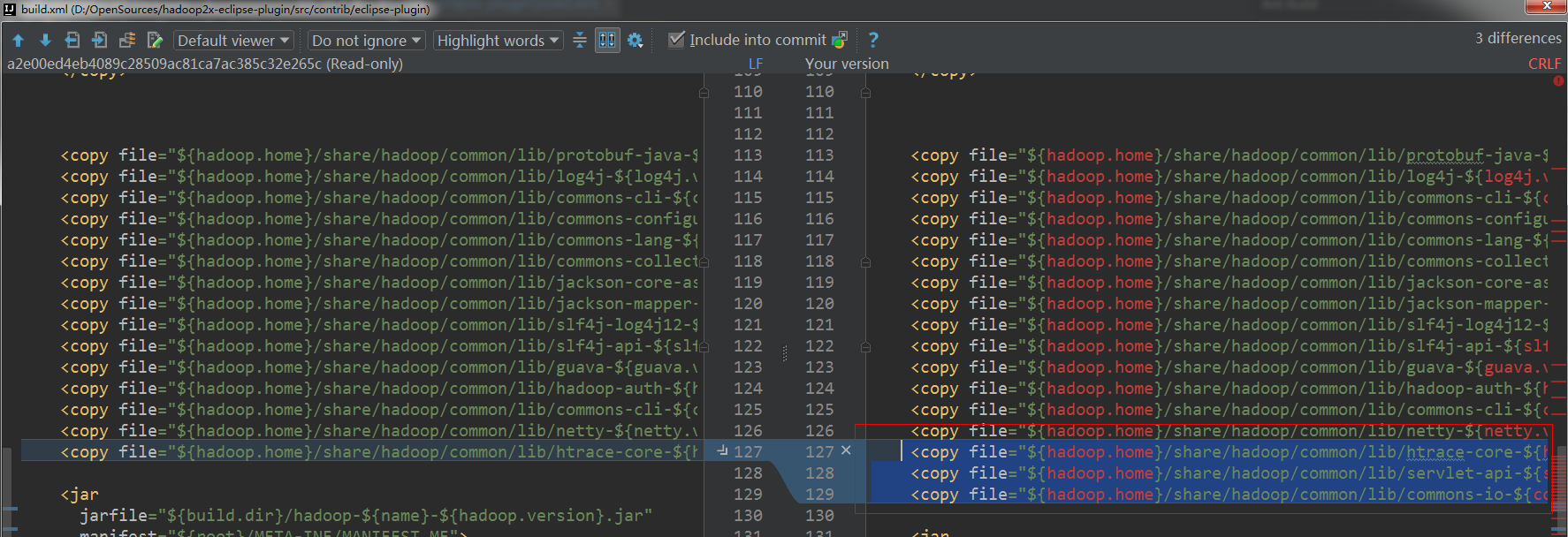
然后将idea中的ant配置好。



接着修改\hadoop2x-eclipse-plugin\src\contrib\eclipse-plugin\build.xml文件



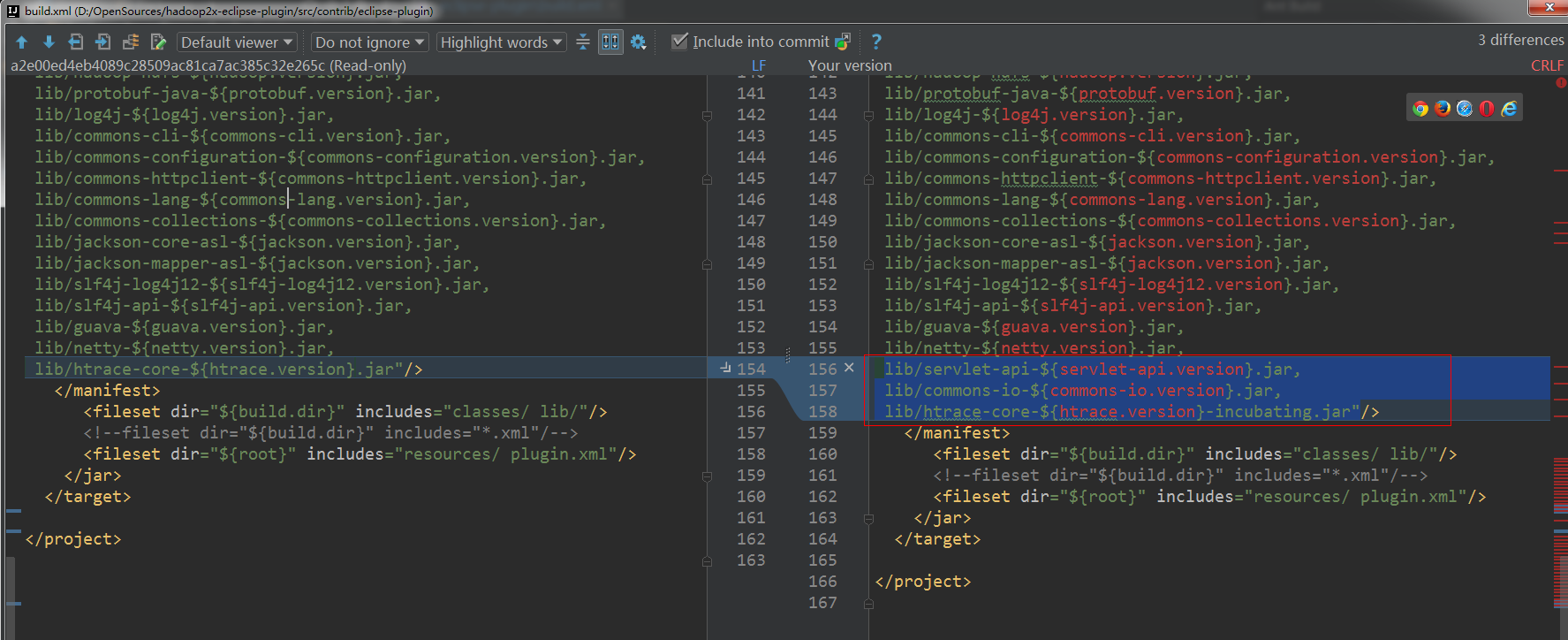
  找到<target name="jar" depends="compile" unless="skip.contrib">标签，然后添加以下内容

 <copy file="${hadoop.home}/share/hadoop/common/lib/htrace-core-${htrace.version}-incubating.jar"  todir="${build.dir}/lib" verbose="true"/>  
    <copy file="${hadoop.home}/share/hadoop/common/lib/servlet-api-${servlet-api.version}.jar"  todir="${build.dir}/lib" verbose="true"/>  
    <copy file="${hadoop.home}/share/hadoop/common/lib/commons-io-${commons-io.version}.jar"  todir="${build.dir}/lib" verbose="true"/>  
  


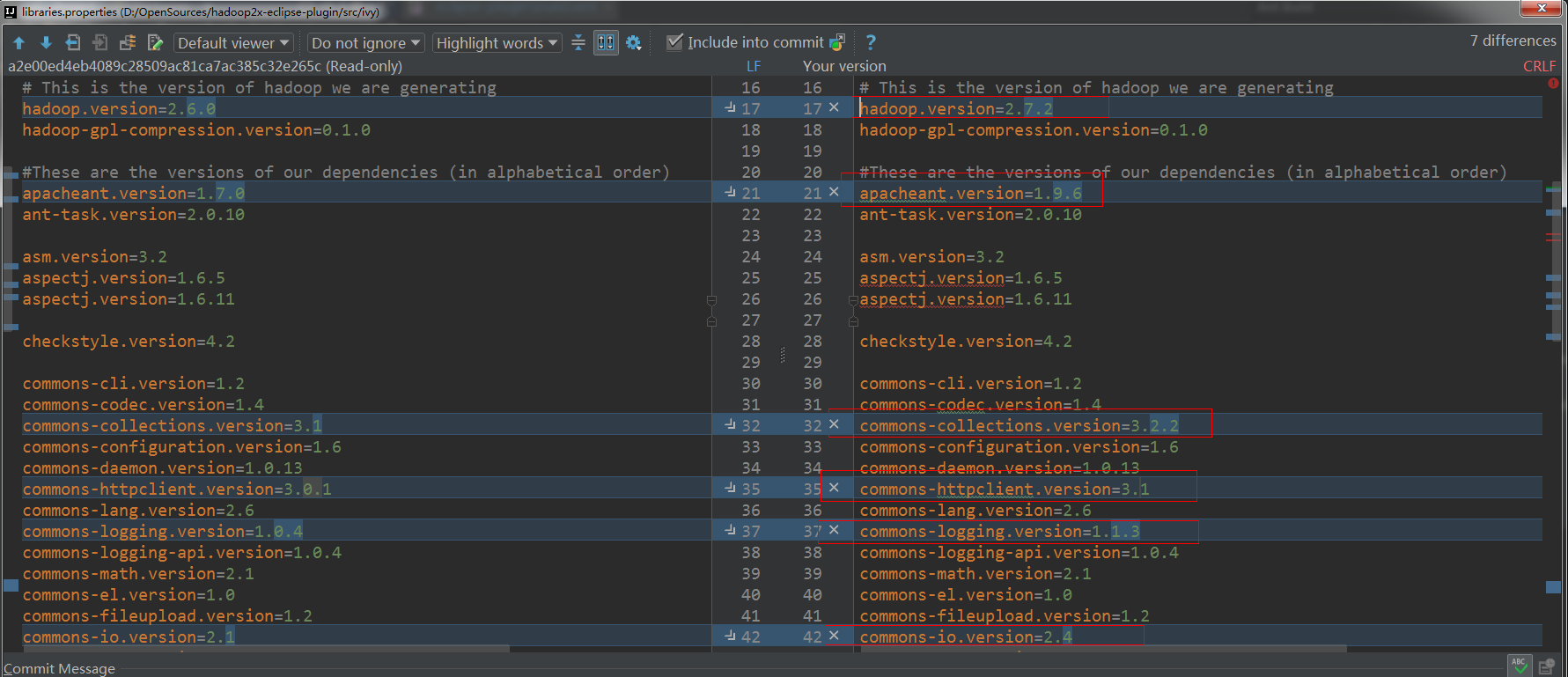
以上这些jar包在编译hadoop2.7.2  eclipse插件的时候需要用到，如果不添加就会报错，所以，我们在ant编译之前先添加进来。

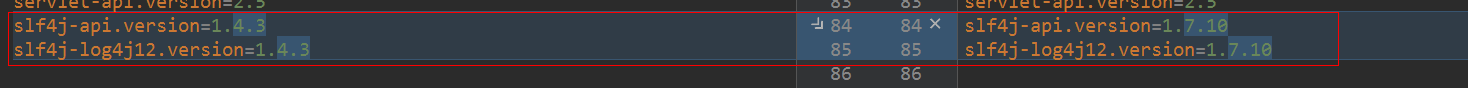
然后再找到<jar arfile="${build.dir}/hadoop-${name}-${hadoop.version}.jar" manifest="${root}/META-INF/MANIFEST.MF">标签，把刚刚添加的包，在ant构建的时候写到mainfest.mf文件的Bundle-ClassPath中：

lib/servlet-api-${servlet-api.version}.jar,  
 lib/commons-io-${commons-io.version}.jar,  
 lib/htrace-core-${htrace.version}-incubating.jar



再修改\hadoop2x-eclipse-plugin\src\ivy\libraries.properties文件，这个文件配置了ant构建需要用到各个jar包的版本，以及构建hadoop的版本，由于这个插件工程师编译hadoop2.6.0的，所以我们需要修改以下配置，其实在ant构建的时候，会选择本地hadoop2.7.2目录中的jar包版本(\hadoop-2.7.2\share\hadoop\common)，所以只要将版本号改成对应的版本号即可如下图：





最后修改\hadoop2x-eclipse-plugin\ivy\libraries.properties文件，文件的的版本如上图版本修改一样，但是还有一个版本需要修改的就是

htrace.version的版本要改成3.1.0，htrace.version=3.1.0

然后cd到D:\OpenSources\hadoop2x-eclipse-plugin\src\contrib\eclipse-plugin目录

执行以下命令：

ant jar -Dversion=2.7.2 -Declipse.home=E:\eclipse-jee-mars-2-win32-x86\_64\eclipse -Dhadoop.home=E:\hadoop-2.7.2

解释下这个命令：-Dversion是指这个插件的版本，Declipse.home是指eclipse的安装目录，-Dhadoop.home指本地文件中hadoop-2.7.2的安装目录。

命令执行成功之后就可以在\hadoop2x-eclipse-plugin\build\contrib\eclipse-plugin目录下面找到

hadoop-eclipse-plugin-2.7.2.jar 包，这个包就是编译好的eclipse hadoop2.7.2插件，把这个插件放到eclipse安装目录的plugins目录下，我们就可以进入eclipse然后找到一个叫mapreduce的视图，就可以开始尝试编写mapreduce程序了

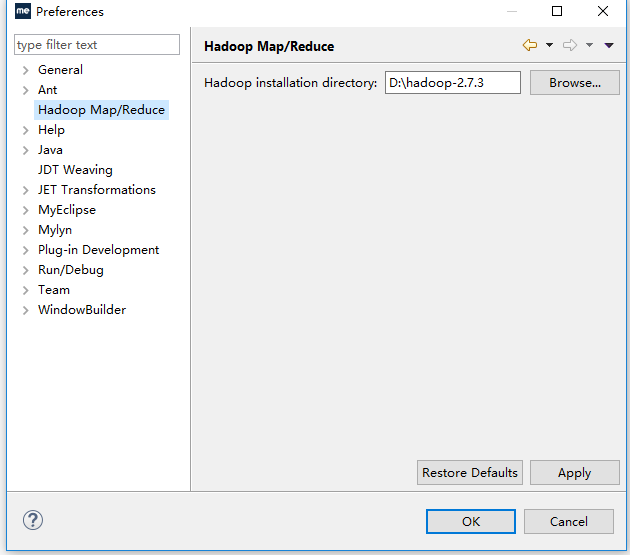
# 1.Hadoop eclipse开发环境搭建

需要下载的文件：链接：http://pan.baidu.com/s/1i5yRyuh 密码：ms91

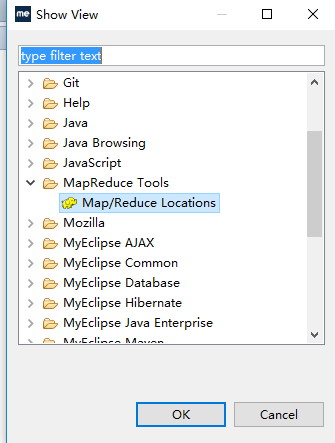
一  下载并编译  hadoop-eclipse-plugin-2.7.3.jar

二  将hadoop-eclipse-plugin-2.7.3.jar放到myeclipse的安装目录下的plugins目录下，并重启myeclipse

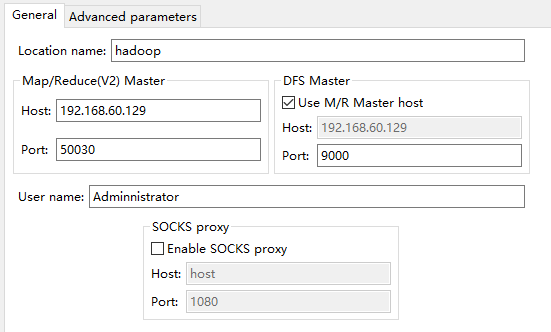
　　在windows->preferences下可看见hadoop Map/Reduce界面，路径选择你WINDOWS下的hadoop解压后的路径。



三 选择Windows->show view->others下的MapReduce Locations



四  新建一个配置 配置如下



host为你的远程hadoop待连接的主机IP地址

Port:50030 对应mapred-site.xml下的jobtracher地址，如下

http://images2015.cnblogs.com/blog/1061762/201611/1061762-20161112160545186-891177377.png

Port:9000对应core-site.xml下的fs.default.name的端口

http://images2015.cnblogs.com/blog/1061762/201611/1061762-20161112160716983-1266366228.png

user name 填你windows的用户名；

修改Advanced parameters下的参数

http://images2015.cnblogs.com/blog/1061762/201611/1061762-20161112161004233-958237557.png

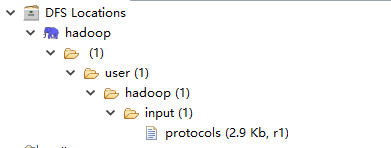
值对应 core-site.xml下的hadoop.tmp.dir参数

http://images2015.cnblogs.com/blog/1061762/201611/1061762-20161112161130499-636649362.png

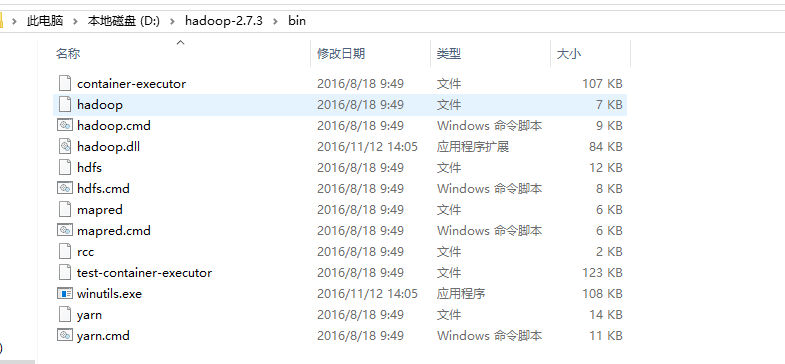
修改hdfs-site.xml下的dfs.permissions参数，允许连接

http://images2015.cnblogs.com/blog/1061762/201611/1061762-20161112161303905-529513455.png

四 保存配置参数并重启myeclipse,可以看见如下的文件结构说明配置连接成功。



五 下载hadoop.ll和winutils.exe 到windows的hadoop/bin目录下

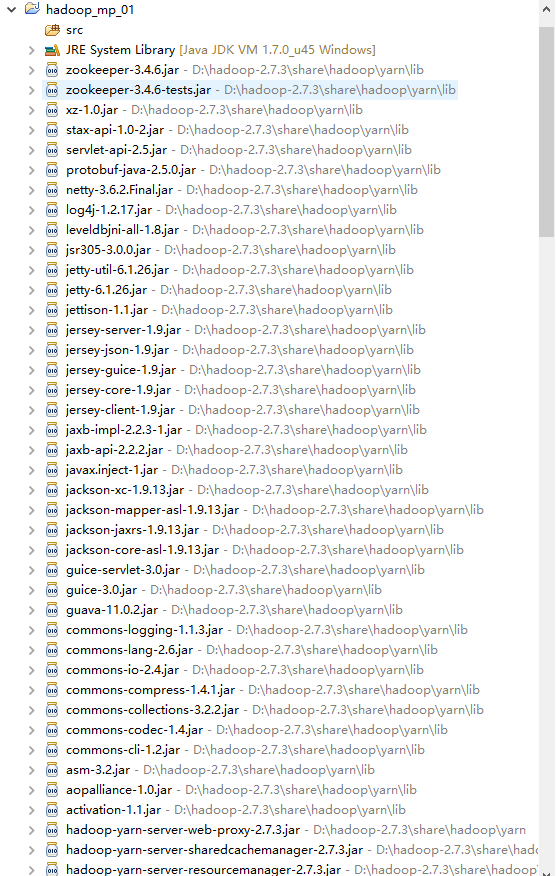


并将hadoop.dll添加到windows->system32目录下

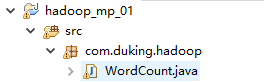
五 环境测试

　　新建项目：File-->New-->Other-->Map/Reduce Project ,项目名可以随便取

　　它会自动添加依赖包，如下：



 新建如下文件：



编写实现代码，与官方例子为例

package com.duking.hadoop;

import java.io.IOException;  
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
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.GenericOptionsParser;

public class WordCount {

public static class TokenizerMapper   
extends Mapper<Object, Text, Text, IntWritable>{

private final static IntWritable one = new IntWritable(1);  
private Text word = new Text();

public void map(Object key, Text value, Context context  
) throws IOException, InterruptedException {  
StringTokenizer itr = new StringTokenizer(value.toString());  
while (itr.hasMoreTokens()) {  
word.set(itr.nextToken());  
context.write(word, one);  
}  
}  
}

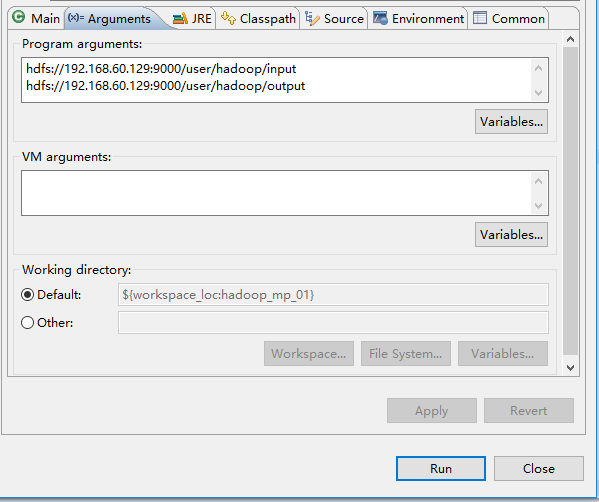
public static class IntSumReducer   
extends Reducer<Text,IntWritable,Text,IntWritable> {  
private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values,   
Context context  
) throws IOException, InterruptedException {  
int sum = 0;  
for (IntWritable val : values) {  
sum += val.get();  
}  
result.set(sum);  
context.write(key, result);  
}  
}

public static void main(String[] args) throws Exception {  
Configuration conf = new Configuration();

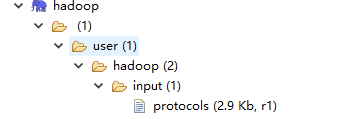
String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();  
if (otherArgs.length != 2) {  
System.err.println(otherArgs.length);  
System.err.println("Usage: wordcount <in> <out>");  
System.exit(2);  
}  
Job job = new Job(conf, "word count");  
job.setJarByClass(WordCount.class);  
job.setMapperClass(TokenizerMapper.class);  
job.setCombinerClass(IntSumReducer.class);  
job.setReducerClass(IntSumReducer.class);  
job.setOutputKeyClass(Text.class);  
job.setOutputValueClass(IntWritable.class);  
FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  
FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  
System.exit(job.waitForCompletion(true) ? 0 : 1);  
}  
}

右击wordcount，选择run as - run configurations



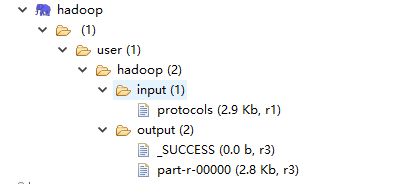
右击wordcount-run as -run on hadoop

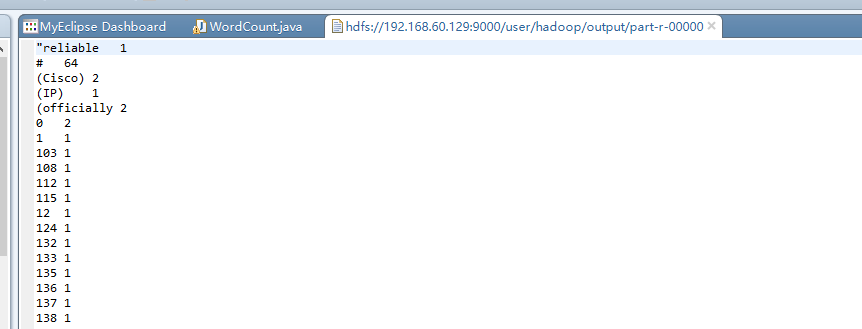
注意：HDFS的目录结构应如下：



protocols为输入待计算的数据。

查看运行结果





至此环境搭建成功！！！！！！！！！！

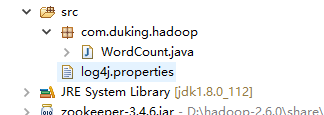
问题总结：环境搭建好后运行mapreduce程序发现output目录下为空，但把程序打包为jar到hadoop环境下运行是有数据输出的。

最后查资料解决方法如下：首先把

http://images2015.cnblogs.com/blog/1061762/201611/1061762-20161113213606311-1101087048.png

这个文件加入工程目录，注意解压的hadoop目录下有两个这个文件，不要加错了。

最后工程目录如下

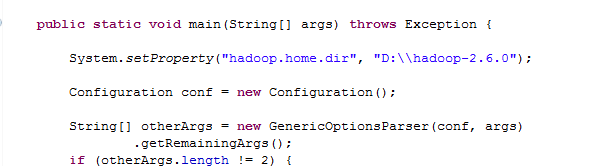


然后运行程序发现报错了，错误提示为：Could not locate executable null



查阅资料后发现是没有添加HADOOP\_HOME环境变量，添加即可。

如果不想重启电脑可以在代码下加如下代码



注意路径改为自己的windows  hadoop路径

# 2.Eclipse 开发Hadoop程序常见错误及解决方法

Windows下的 Eclipse上调试Hadoop2代码，所以我们在windows下的Eclipse配置[**Hadoop**](http://lib.csdn.net/base/20)-eclipse-plugin-2.6.0.jar插件，并在运行Hadoop代码时出现了一系列的问题，搞了好几天终于能运行起代码。接下来我们来看看问题并怎么解决，提供给跟我同样遇到的问题作为参考。

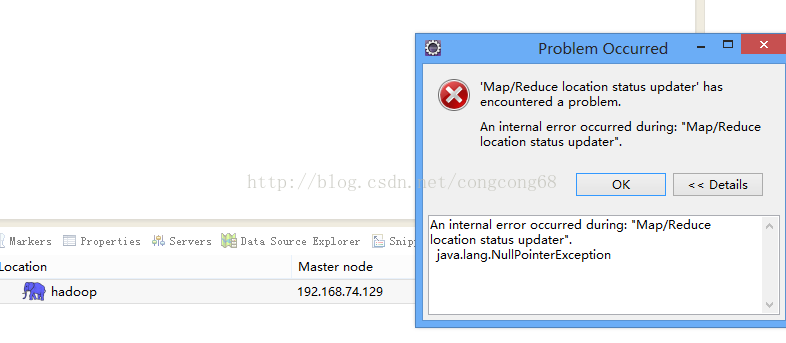
  Hadoop2的WordCount.[**Java**](http://lib.csdn.net/base/17)统计代码如下：

**[java]** [view plain](http://blog.csdn.net/congcong68/article/details/42043093) [copy](http://blog.csdn.net/congcong68/article/details/42043093)

1. **import** java.io.IOException;
2. **import** java.util.StringTokenizer;
4. **import** org.apache.hadoop.conf.Configuration;
5. **import** org.apache.hadoop.fs.Path;
6. **import** org.apache.hadoop.io.IntWritable;
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.output.FileOutputFormat;
14. **public** **class** WordCount {
16. **public** **static** **class** TokenizerMapper
17. **extends** Mapper<Object, Text, Text, IntWritable>{
19. **private** **final** **static** IntWritable one = **new** IntWritable(1);
20. **private** Text word = **new** Text();
22. **public** **void** map(Object key, Text value, Context context
23. ) **throws** IOException, InterruptedException {
24. StringTokenizer itr = **new** StringTokenizer(value.toString());
25. **while** (itr.hasMoreTokens()) {
26. word.set(itr.nextToken());
27. context.write(word, one);
28. }
29. }
30. }
32. **public** **static** **class** IntSumReducer
33. **extends** Reducer<Text,IntWritable,Text,IntWritable> {
34. **private** IntWritable result = **new** IntWritable();
36. **public** **void** reduce(Text key, Iterable<IntWritable> values,
37. Context context
38. ) **throws** IOException, InterruptedException {
39. **int** sum = 0;
40. **for** (IntWritable val : values) {
41. sum += val.get();
42. }
43. result.set(sum);
44. context.write(key, result);
45. }
46. }
48. **public** **static** **void** main(String[] args) **throws** Exception {
49. Configuration conf = **new** Configuration();
50. Job job = Job.getInstance(conf, "word count");
51. job.setJarByClass(WordCount.**class**);
52. job.setMapperClass(TokenizerMapper.**class**);
53. job.setCombinerClass(IntSumReducer.**class**);
54. job.setReducerClass(IntSumReducer.**class**);
55. job.setOutputKeyClass(Text.**class**);
56. job.setOutputValueClass(IntWritable.**class**);
57. FileInputFormat.addInputPath(job, **new** Path(args[0]));
58. FileOutputFormat.setOutputPath(job, **new** Path(args[1]));
59. System.exit(job.waitForCompletion(**true**) ? 0 : 1);
60. }
61. }

## **问题一.An internal error occurred during: "Map/Reducelocation status updater".java.lang.NullPointerException**

   我们hadoop-eclipse-plugin-2.6.0.jar放到Eclipse的plugins目录下，我们的Eclipse目录是F:\tool\eclipse-jee-juno-SR2\eclipse-jee-juno-SR2\plugins，重启一下Eclipse，然后，打开Window-->Preferens，可以看到Hadoop Map/Reduc选项，然后点击出现了An internal error occurredduring: "Map/Reduce location status updater".java.lang.NullPointerException，如图所示：



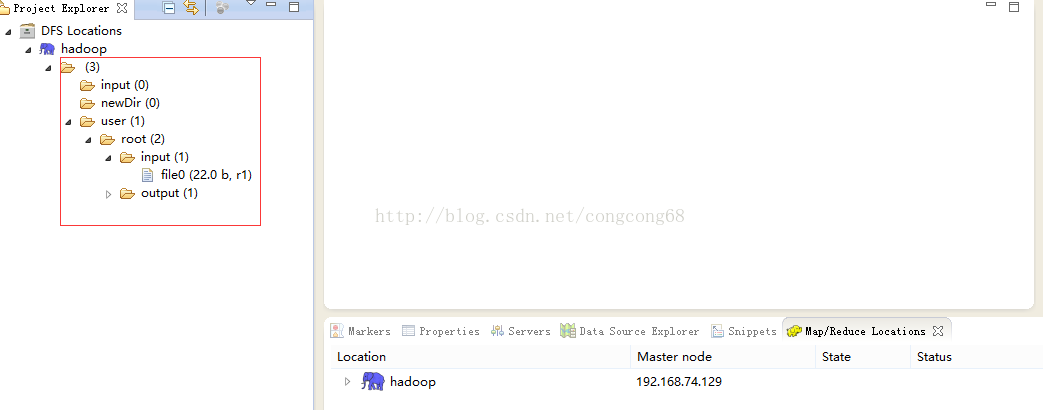
  解决：

   我们发现刚配置部署的Hadoop2还没创建输入和输出目录，先在hdfs上建个文件夹 。

   #bin/hdfs dfs -mkdir –p /user/root/input

   #bin/hdfs dfs -mkdir -p  /user/root/output

 我们在Eclipse的DFS Locations目录下看到我们这两个目录，如图所示：



## **问题二.Exception in thread "main" java.lang.NullPointerException atjava.lang.ProcessBuilder.start(Unknown Source)**

运行Hadoop2的WordCount.java代码时出现了这样错误，

**[html]** [view plain](http://blog.csdn.net/congcong68/article/details/42043093) [copy](http://blog.csdn.net/congcong68/article/details/42043093)

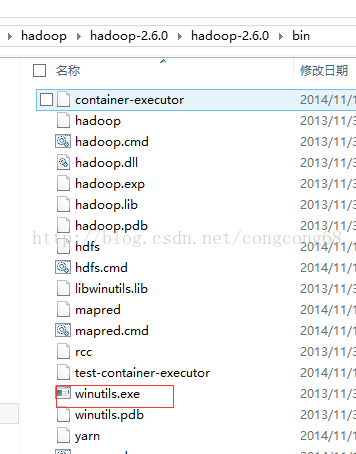
1. log4j:WARNPlease initialize the log4j system properly.
2. log4j:WARN Seehttp://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
3. Exception in thread "main" java.lang.NullPointerException
4. atjava.lang.ProcessBuilder.start(Unknown Source)
5. atorg.apache.hadoop.util.Shell.runCommand(Shell.java:482)
6. atorg.apache.hadoop.util.Shell.run(Shell.java:455)
7. atorg.apache.hadoop.util.Shell$ShellCommandExecutor.execute(Shell.java:715)
8. atorg.apache.hadoop.util.Shell.execCommand(Shell.java:808)
9. atorg.apache.hadoop.util.Shell.execCommand(Shell.java:791)
10. at

分析：

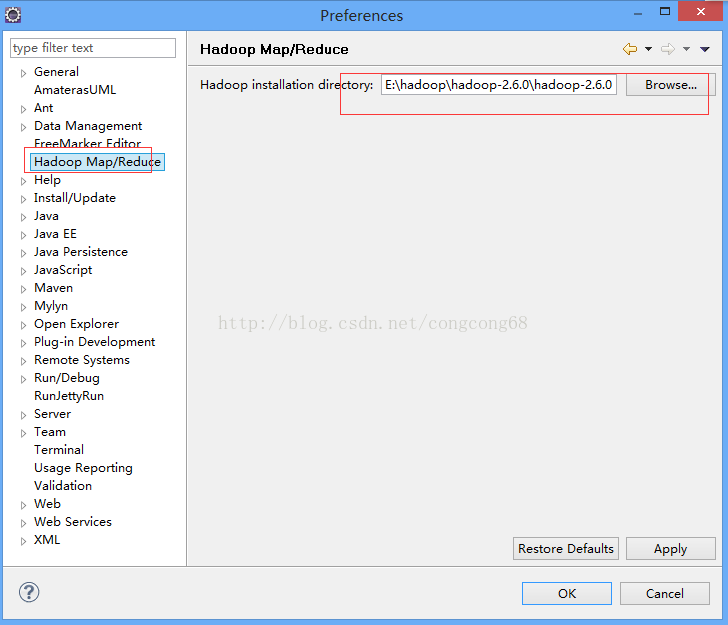
  下载Hadoop2以上版本时，在Hadoop2的bin目录下没有winutils.exe

解决：

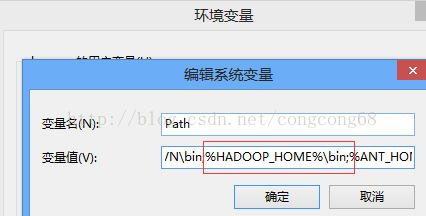
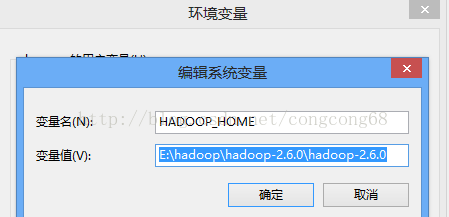
  1.下载<https://codeload.github.com/srccodes/hadoop-common-2.2.0-bin/zip/master>下载hadoop-common-2.2.0-bin-master.zip，然后解压后，把hadoop-common-2.2.0-bin-master下的bin全部复制放到我们下载的Hadoop2的binHadoop2/bin目录下。如图所示：



  2.Eclipse-》window-》Preferences 下的Hadoop Map/Peduce 把下载放在我们的磁盘的Hadoop目录引进来，如图所示：



  3.Hadoop2配置变量环境HADOOP\_HOME 和path，如图所示：



## **问题三.Exception in thread "main"java.lang.UnsatisfiedLinkError:org.apache.hadoop.io.nativeio.NativeIO$Windows.access0(Ljava/lang/String;I)Z**

  当我们解决了问题三时，在运行WordCount.java代码时，出现这样的问题

**[html]** [view plain](http://blog.csdn.net/congcong68/article/details/42043093) [copy](http://blog.csdn.net/congcong68/article/details/42043093)

1. log4j:WARN No appenders could be found forlogger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
2. log4j:WARN Please initialize the log4jsystem properly.
3. log4j:WARN Seehttp://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
4. Exception in thread "main"java.lang.UnsatisfiedLinkError:org.apache.hadoop.io.nativeio.NativeIO$Windows.access0(Ljava/lang/String;I)Z
5. atorg.apache.hadoop.io.nativeio.NativeIO$Windows.access0(Native Method)
6. atorg.apache.hadoop.io.nativeio.NativeIO$Windows.access(NativeIO.java:557)
7. atorg.apache.hadoop.fs.FileUtil.canRead(FileUtil.java:977)
8. atorg.apache.hadoop.util.DiskChecker.checkAccessByFileMethods(DiskChecker.java:187)
9. atorg.apache.hadoop.util.DiskChecker.checkDirAccess(DiskChecker.java:174)
10. atorg.apache.hadoop.util.DiskChecker.checkDir(DiskChecker.java:108)
11. atorg.apache.hadoop.fs.LocalDirAllocator$AllocatorPerContext.confChanged(LocalDirAllocator.java:285)
12. atorg.apache.hadoop.fs.LocalDirAllocator$AllocatorPerContext.getLocalPathForWrite(LocalDirAllocator.java:344)
13. atorg.apache.hadoop.fs.LocalDirAllocator.getLocalPathForWrite(LocalDirAllocator.java:150)
14. atorg.apache.hadoop.fs.LocalDirAllocator.getLocalPathForWrite(LocalDirAllocator.java:131)
15. atorg.apache.hadoop.fs.LocalDirAllocator.getLocalPathForWrite(LocalDirAllocator.java:115)
16. atorg.apache.hadoop.mapred.LocalDistributedCacheManager.setup(LocalDistributedCacheManager.java:131)

 分析：

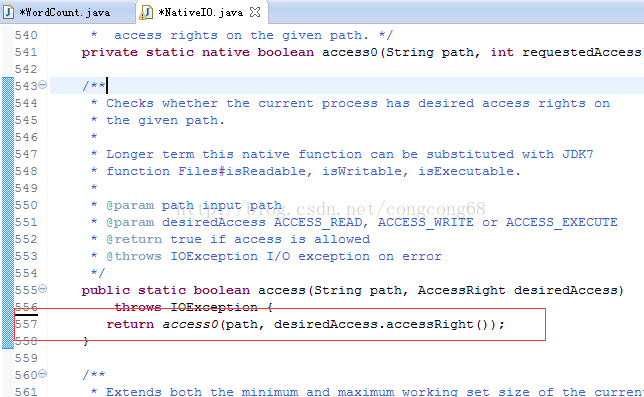
    C:\Windows\System32下缺少hadoop.dll,把这个文件拷贝到C:\Windows\System32下面即可。

 解决：

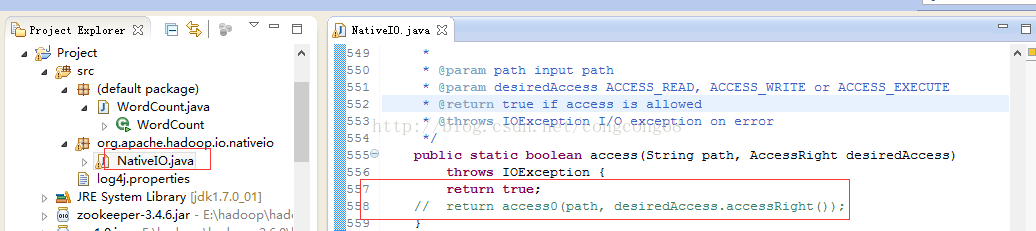
    hadoop-common-2.2.0-bin-master下的bin的hadoop.dll放到C:\Windows\System32下，然后重启电脑，也许还没那么简单，还是出现这样的问题。

  我们在继续分析：

    我们在出现错误的的atorg.apache.hadoop.io.nativeio.NativeIO$Windows.access(NativeIO.java:557)我们来看这个类NativeIO的557行，如图所示：



   Windows的唯一方法用于检查当前进程的请求，在给定的路径的访问权限，所以我们先给以能进行访问，我们自己先修改源代码，return true 时允许访问。我们下载对应hadoop源代码，hadoop-2.6.0-src.tar.gz解压，hadoop-2.6.0-src\hadoop-common-project\hadoop-common\src\main\java\org\apache\hadoop\io\nativeio下NativeIO.java 复制到对应的Eclipse的project，然后修改557行为return true如图所示：



**问题四：org.apache.hadoop.security.AccessControlException: Permissiondenied: user=zhengcy, access=WRITE,inode="/user/root/output":root:supergroup:drwxr-xr-x**

  我们在执行运行WordCount.java代码时，出现这样的问题

**[html]** [view plain](http://blog.csdn.net/congcong68/article/details/42043093) [copy](http://blog.csdn.net/congcong68/article/details/42043093)

1. 2014-12-18 16:03:24,092  WARN (org.apache.hadoop.mapred.LocalJobRunner:560) - job\_local374172562\_0001
2. org.apache.hadoop.security.AccessControlException: Permission denied: user=zhengcy, access=WRITE, inode="/user/root/output":root:supergroup:drwxr-xr-x
3. at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.checkFsPermission(FSPermissionChecker.java:271)
4. at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.check(FSPermissionChecker.java:257)
5. at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.check(FSPermissionChecker.java:238)
6. at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.checkPermission(FSPermissionChecker.java:179)
7. at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.checkPermission(FSNamesystem.java:6512)
8. at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.checkPermission(FSNamesystem.java:6494)
9. at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.checkAncestorAccess(FSNamesystem.java:6446)
10. at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.mkdirsInternal(FSNamesystem.java:4248)
11. at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.mkdirsInt(FSNamesystem.java:4218)
12. at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.mkdirs(FSNamesystem.java:4191)
13. at org.apache.hadoop.hdfs.server.namenode.NameNodeRpcServer.mkdirs(NameNodeRpcServer.java:813)

 分析：

  我们没权限访问output目录。

解决：

    我们 在设置hdfs配置的目录是在hdfs-site.xml配置hdfs文件存放的地方，我在hadoop伪分布式部署那边有介绍过，我们在这边在复习一下，如图所示：

我们在这个etc/hadoop下的hdfs-site.xml添加

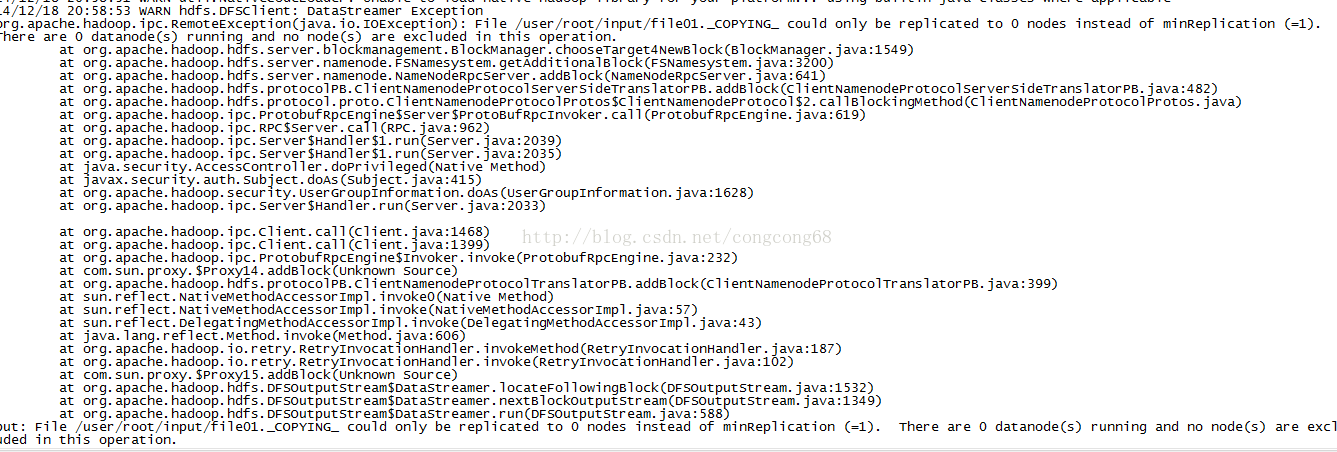
  <property>

     <name>dfs.permissions</name>   
     <value>false</value>   
  </property>

设置没有权限，不过我们在正式的 服务器上不能这样设置。

**问题五：File/usr/root/input/file01.\_COPYING\_ could only be replicated to 0 nodes instead ofminRepLication (=1) There are 0 datanode(s) running and no node(s) are excludedin this operation**

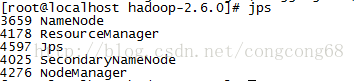
     如图所示：



  分析：

  我们在第一次执行#hadoop namenode –format 完然后在执行#sbin/start-all.sh

在执行#jps，能看到Datanode，在执行#hadoop namenode –format然后执行#jps这时看不到Datanode ,如图所示：



   然后我们想把文本放到输入目录执行bin/hdfs dfs -put/usr/local/hadoop/hadoop-2.6.0/test/\* /user/root/input  把/test/\*文件上传到hdfs的/user/root/input中,出现这样的问题，

 解决：

  是我们执行太多次了hadoopnamenode –format，在创建了多个，我们对应的hdfs目录删除hdfs-site.xml配置的保存datanode和namenode目录。

重新安装需要删除配置目录下的所有文件，然后hadoop namenode –format。不然会出现put文件失败。