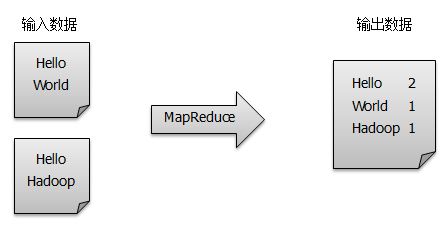
# Hadoop之单词计数

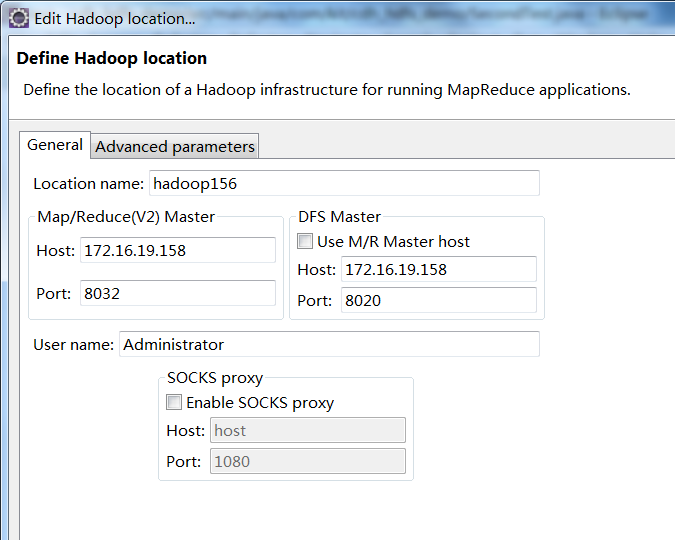
单词计数是最简单也是最能体现MapReduce思想的程序之一，可以称为MapReduce版“Hello World”，该程序的完整代码可以在Hadoop安装包的src/example目录下找到。单词计数主要完成的功能：统计一系列文本文件中每个单词出 现的次数，如下图所示。本blog将通过分析WordCount源码来帮助大家摸清MapReduce程序的基本结构和运行机制。

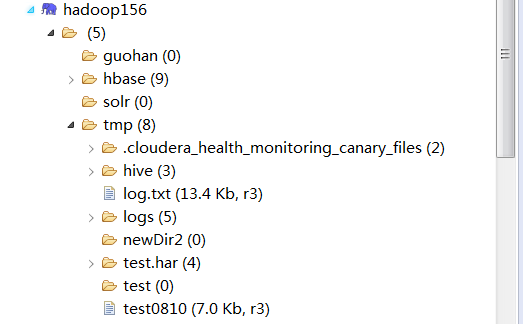


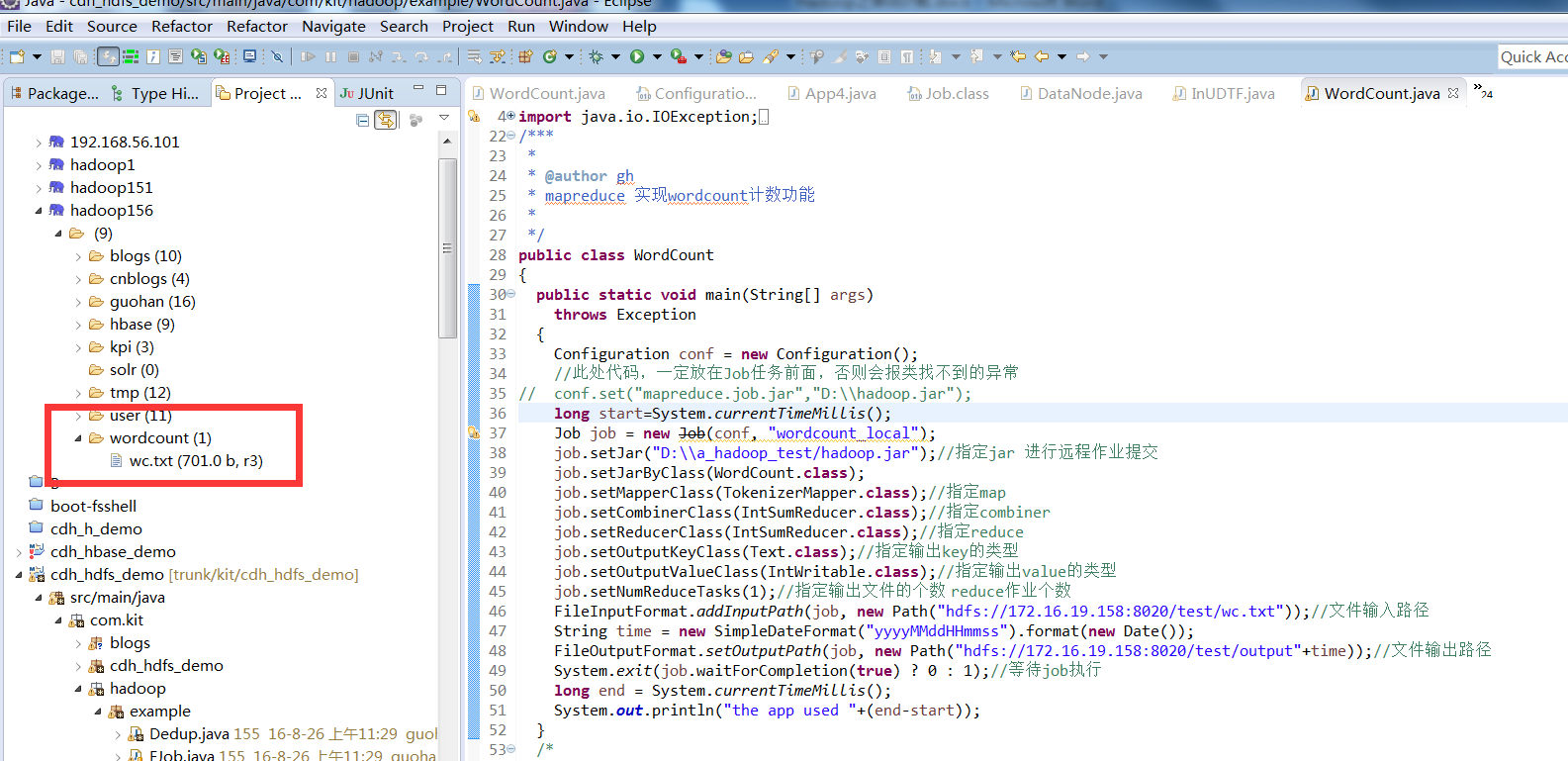
### 开发环境

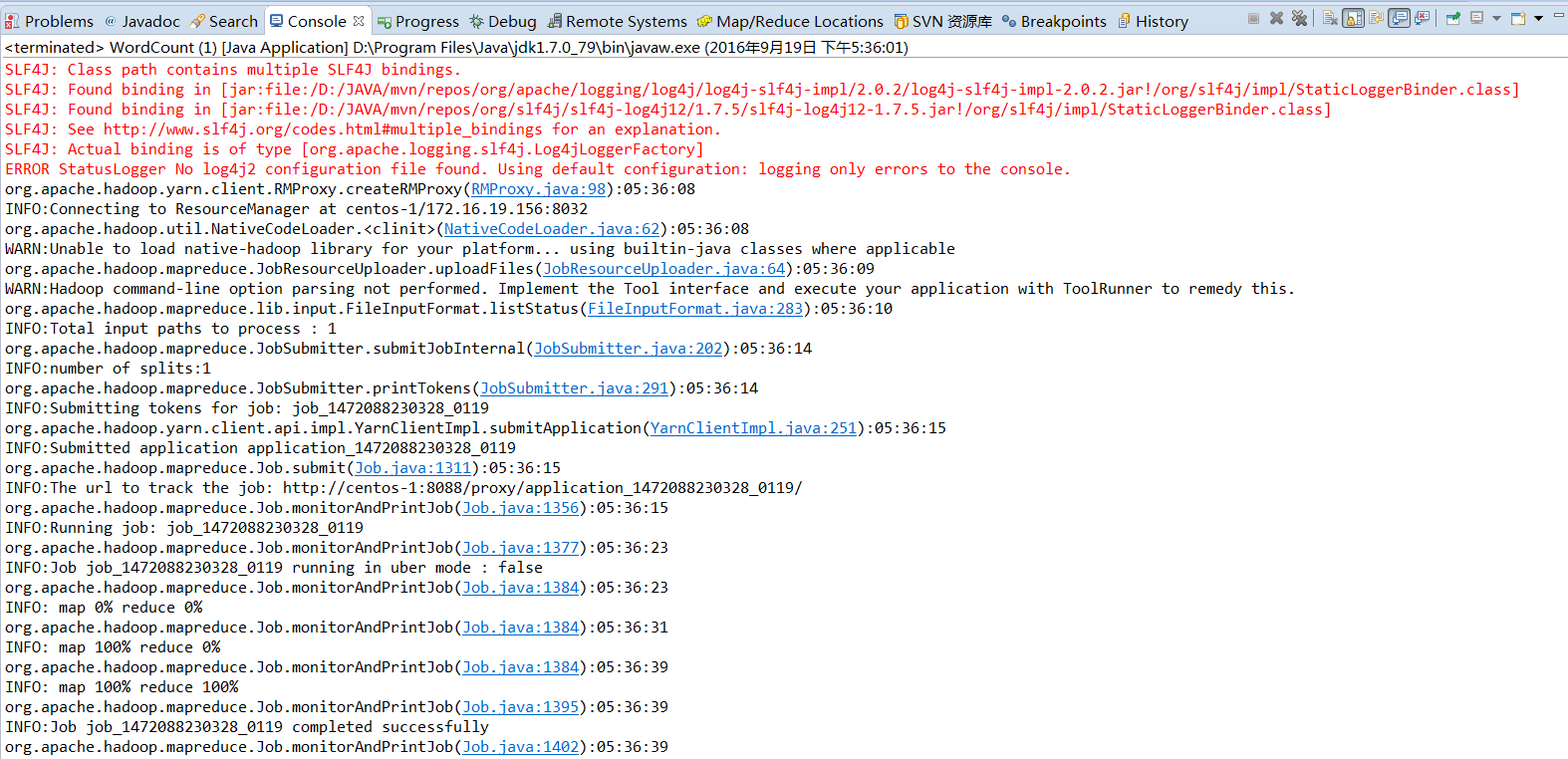
硬件环境：Centos 6.5 服务器4台（一台为Master节点，三台为Slave节点）   
软件环境：Java 1.7.0\_45、hadoop-1.2.1

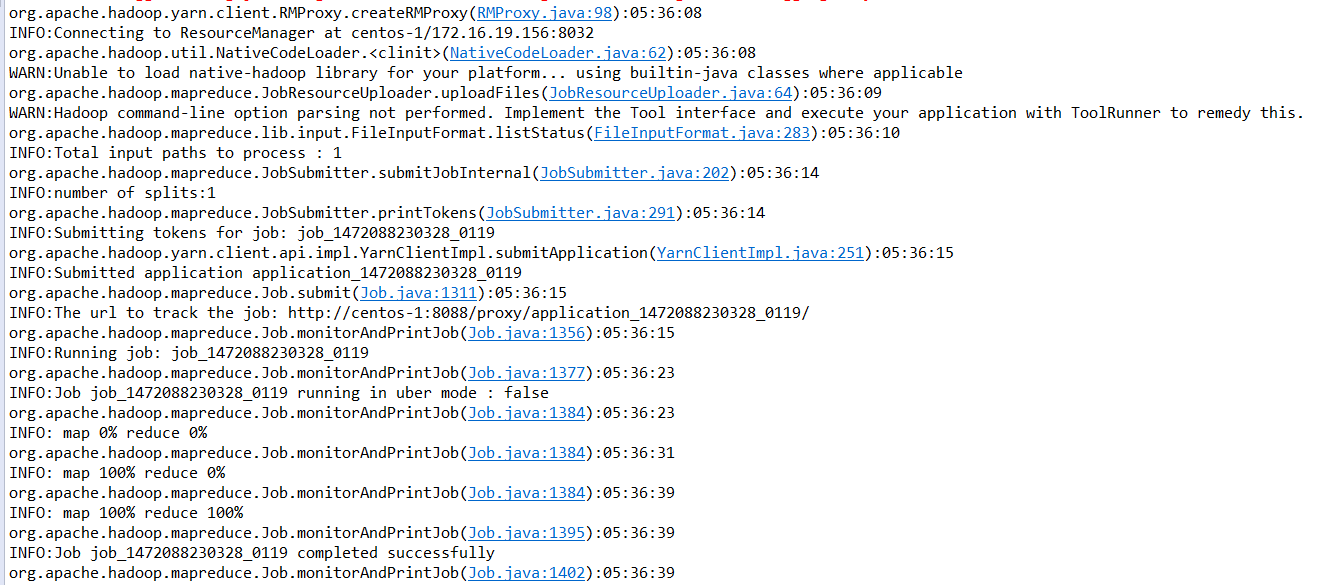
eclipse开发环境配置，不作重点讲解，成功安装插件后：

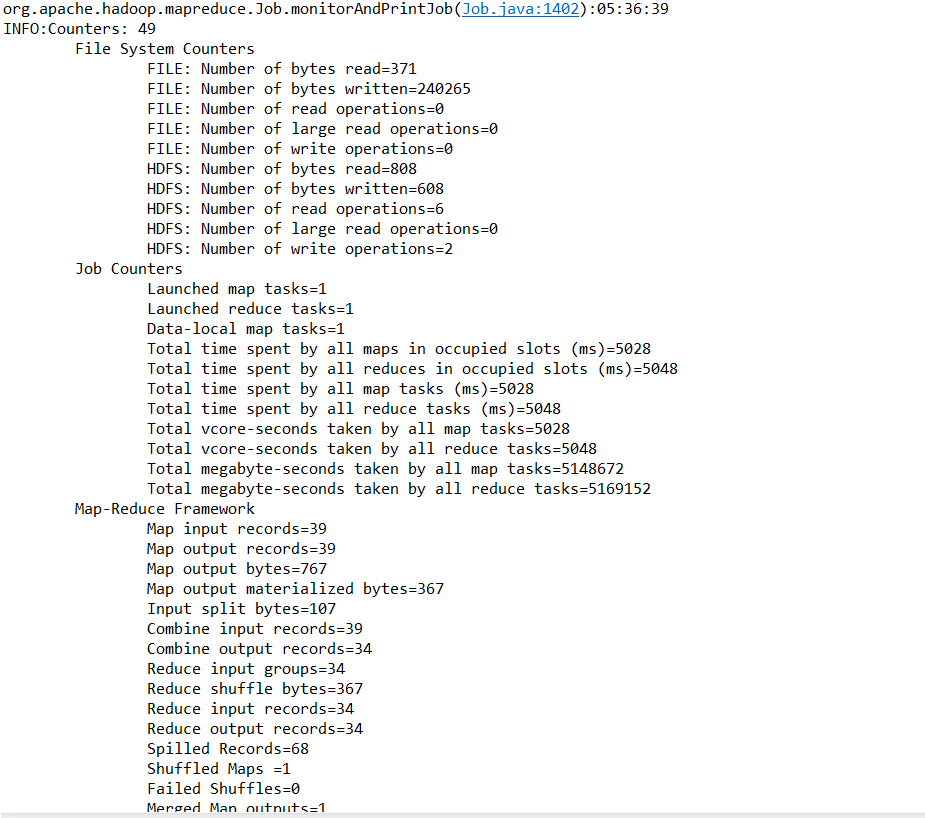












### 1、 WordCount的Map过程

Map过程需要继承org.apache.hadoop.mapreduce包中Mapper类，并重写其map方法。Map方法中的value 值存储的是文本文件中的一行记录（以回车符为结束标记），而key值为该行的首字符相对于文本文件的首地址的偏移量。然后StringTokenizer 类将每一行拆分成一个个的单词，并将

### 2、 WordCount的Reduce过程

Reduce过程需要继承org.apache.hadoop.mapreduce包中的Reduce类，并重写其reduce方法。 Reduce方法的输入参数key为单个单词，而values是由各Mapper上对应单词的计数值所组成的列表，所以只要遍历values并求和，即可 得到某个单词出现的总次数。   
IntSumReducer类的实现代码如下，详细源码请参考：



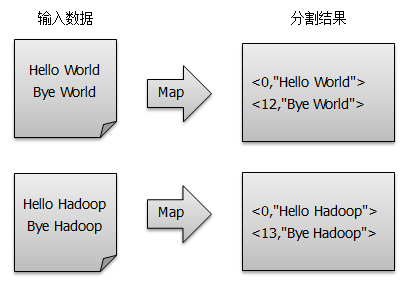
### 3、 WordCount的驱动执行过程

在MapReduce中，由Job对象负责管理和运行一个计算任务，并通过Job的一些方法对任务的参数进行相关的设置。此处设置了使用 TokenizerMapper完成Map过程和使用IntSumReducer完成Combine和Reduce过程。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。任务的输入和输出路径则由命令行参数指定，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。   
驱动函数实现代码如下，详细源码请参考：WordCount\src\WordCount.java。

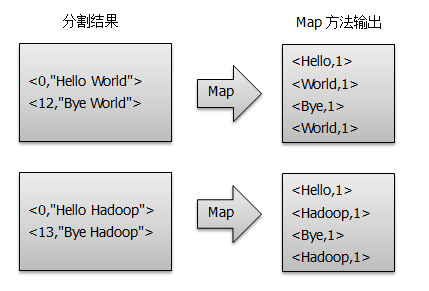


### 4、 WordCount的处理过程

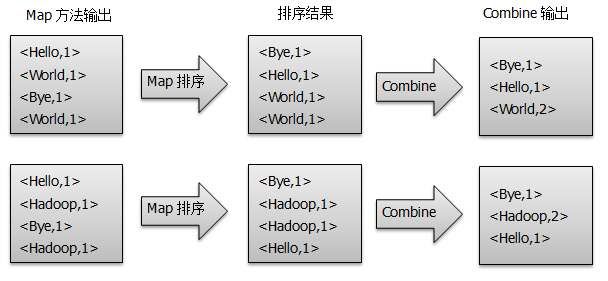
如上所述给出了WordCount的设计思路及源码分析过程，但很多细节都未被提及，本节将根据MapReduce的处理工程，对WordCount进行更详细的讲解。详细的执行步骤如下：   
1）将文件拆分成splits，由于测试用的文件较小，所以每个文件为一个split，并将文件按行分割形成< key,value >对，如图所示。这一步由MapReduce框架自动完成，其中偏移量（即key值）包括了回车符所占的字符数（Windows和Linux环境下 会不同）。



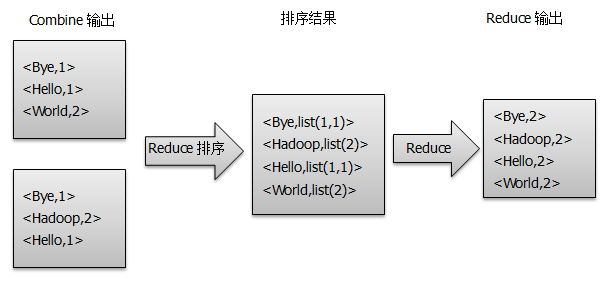
2）将分割好的< key,value>对交给用户定义的map方法进行处理，生成新的< key,value >对，如图所示：



3）得到map方法输出的< key,value>对后，Mapper会将它们按照key值进行排序，并执行Combine过程，将key值相同的value值累加，得到Mapper的最终输出结果，如图所示：



4） Reducer先对从Mapper接收的数据进行排序，再交由用户自定义的reducer方法进行处理，得到新的< key,value>对，并作为WordCount的输出结果，如图所示：



### 5、部署运行

#### 1）部署源码

#设置工作环境

[hadoop@K-Master ~]$ mkdir -p /usr/hadoop/workspace/MapReduce

#部署源码

将WordCount 文件夹拷贝到/usr/hadoop/workspace/MapReduce/ 路径下；

… 你可以直接 [下载 WordCount](http://pan.baidu.com/s/18kOaU)

#### 2）编译文件

在使用javac编译命令时，我们用到了两个参数：-classpath指定编译该类所需要的核心包，-d指定编译后生成的class文件的存放路径；最后的WordCount.java表示编译的对象是当前文件夹下的WordCount.java类。

[hadoop@K-Master ~]$ cd /usr/hadoop/workspace/MapReduce/WordCount

[hadoop@K-Master WordCount]$ javac -classpath /usr/hadoop/hadoop-core-1.2.1.jar:/usr/hadoop/lib/commons-cli-1.2.jar -d bin/ src/WordCount.java

#查看编译结果

[hadoop@K-Master WordCount]$ ls bin/ -la

总用量 12

drwxrwxr-x 2 hadoop hadoop 102 9月 15 11:08 .

drwxrwxr-x 4 hadoop hadoop 69 9月 15 10:55 ..

-rw-rw-r-- 1 hadoop hadoop 1830 9月 15 11:08 WordCount.class

-rw-rw-r-- 1 hadoop hadoop 1739 9月 15 11:08 WordCount$IntSumReducer.class

-rw-rw-r-- 1 hadoop hadoop 1736 9月 15 11:08 WordCount$TokenizerMapper.class

#### 3）打包jar文件

在使用jar命令进行打包class文件时，我们用到了两个参数：-cvf表示打包class文件并显示详细的打包信息，-C指定打包的对象；命令最后的“.”表示将打包生成的文件保存在当前目录下。

[hadoop@K-Master WordCount]$ jar -cvf WordCount.jar -C bin/ .

已添加清单

正在添加: WordCount$TokenizerMapper.class(输入 = 1736) (输出 = 754)(压缩了 56%)

正在添加: WordCount$IntSumReducer.class(输入 = 1739) (输出 = 74

**特别注意：打包命令最后一个字符为“.”，表示将打包生成的文件WordCount.jar保存到当前文件夹下，输入命令时特别留心。**

#### 4）启动Hadoop集群

如果HDFS已经启动，则不需要执行以下命令，可通过jps命令查看HDFS是否已经启动

[hadoop@K-Master WordCount]$ start-dfs.sh #启动HDFS文件系统

[hadoop@K-Master WordCount]$ start-mapred.sh #启动MapReducer服务

[hadoop@K-Master WordCount]$ jps

5082 JobTracker

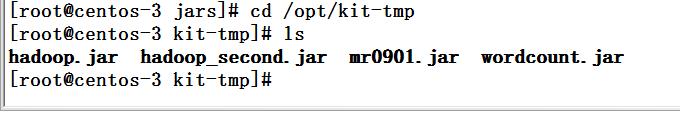
4899 SecondaryNameNode

9048 Jps

4735 NameNode

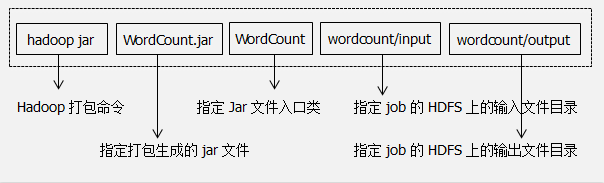
#### 5）上传输入文件到HDFS

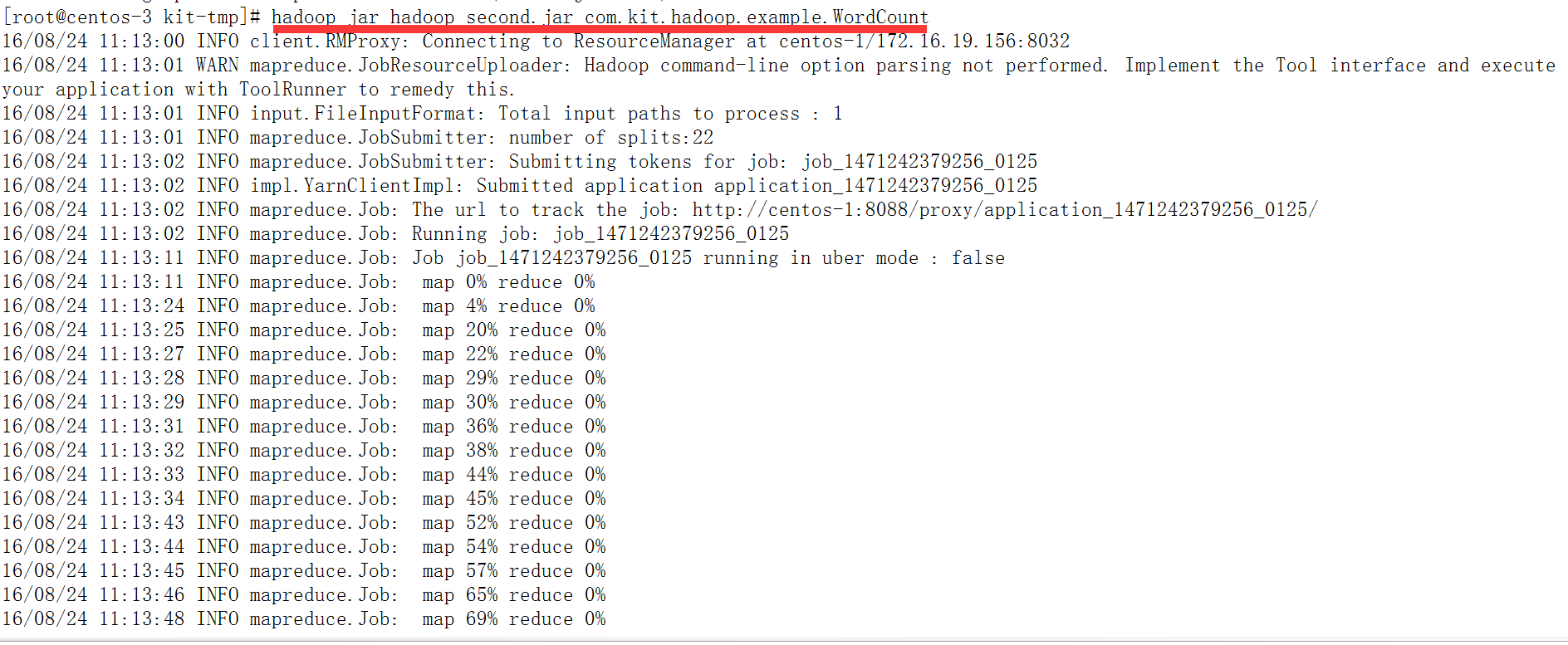
在MapReduce中，一个准备提交执行的应用程序称为“作业（Job）”，Master节点将对该Job划分成多个task运行于各计算节点 上（Slave节点），而task任务输入输出的数据均是基于HDFS分布式文件管理系统，故需要将输入数据上传到HDFS分布式文件管理系统之上，如下 所示。



#### 6）运行Jar文件

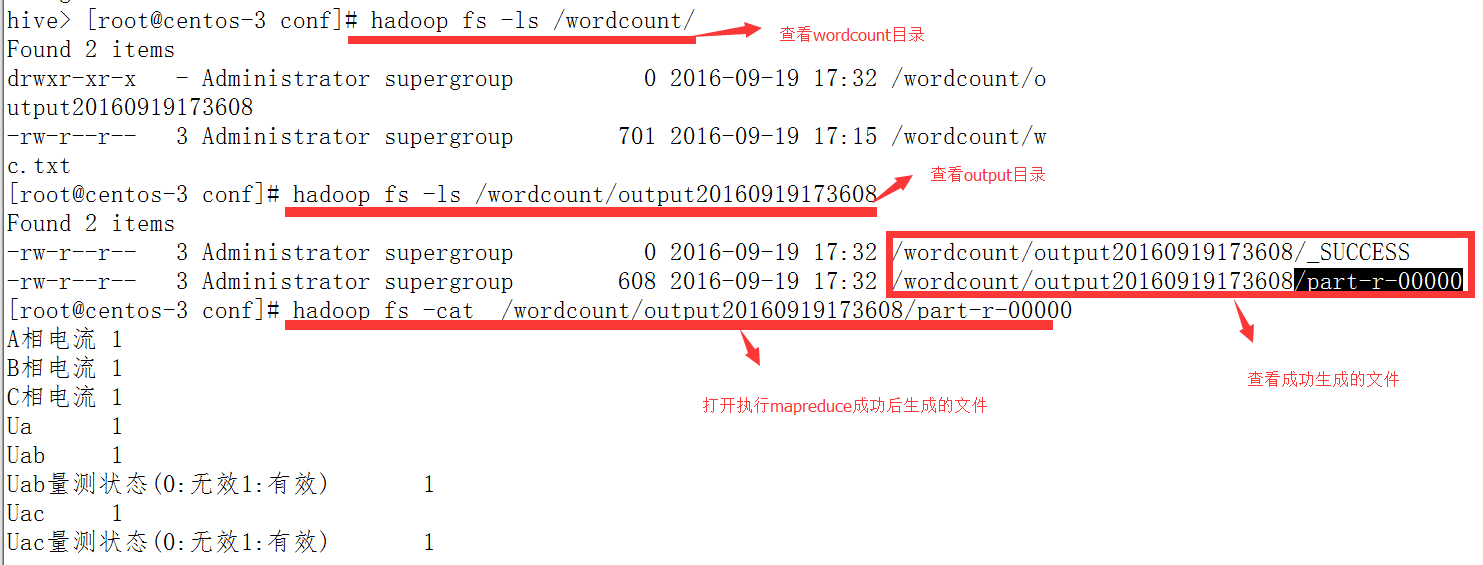
我们通过hadoop jar命令运行一个job任务，关于该命令各个参数的含义如下图所示：

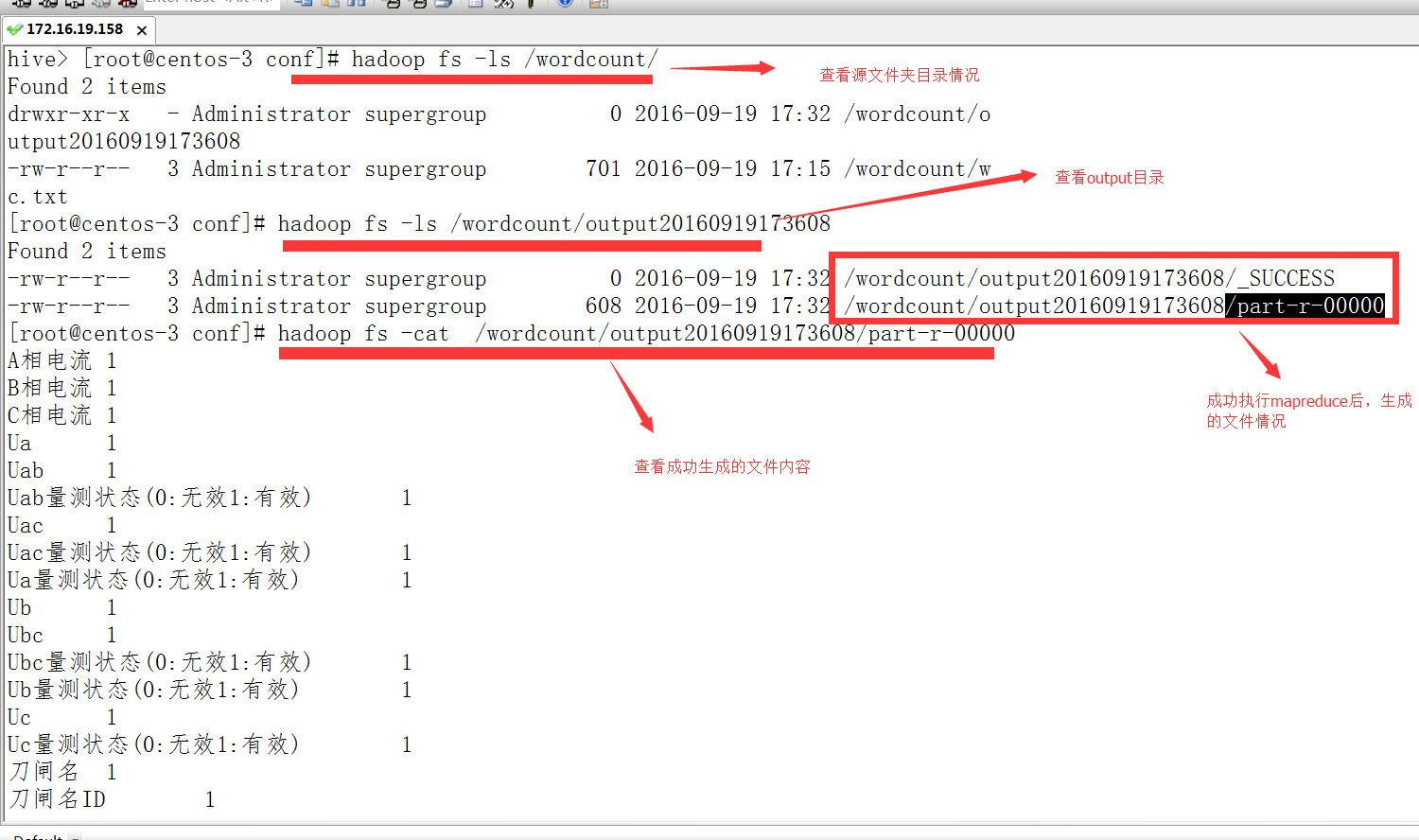


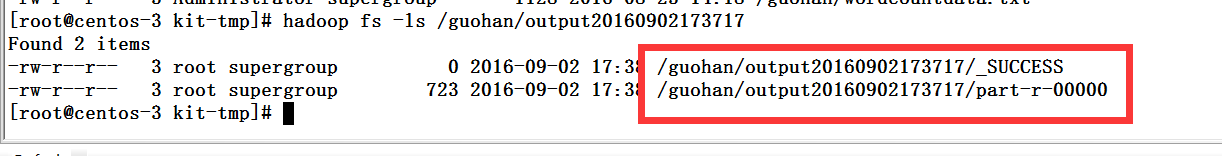


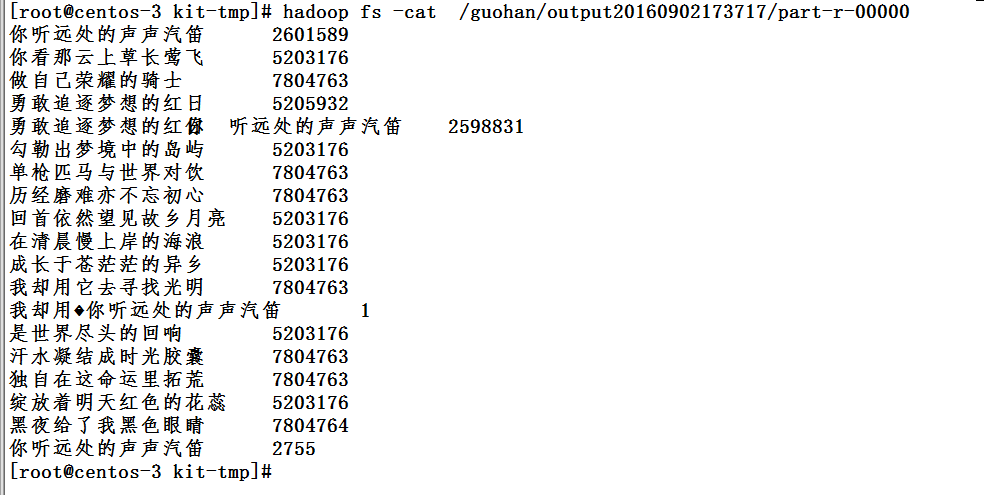
#### 7）查看运行结果

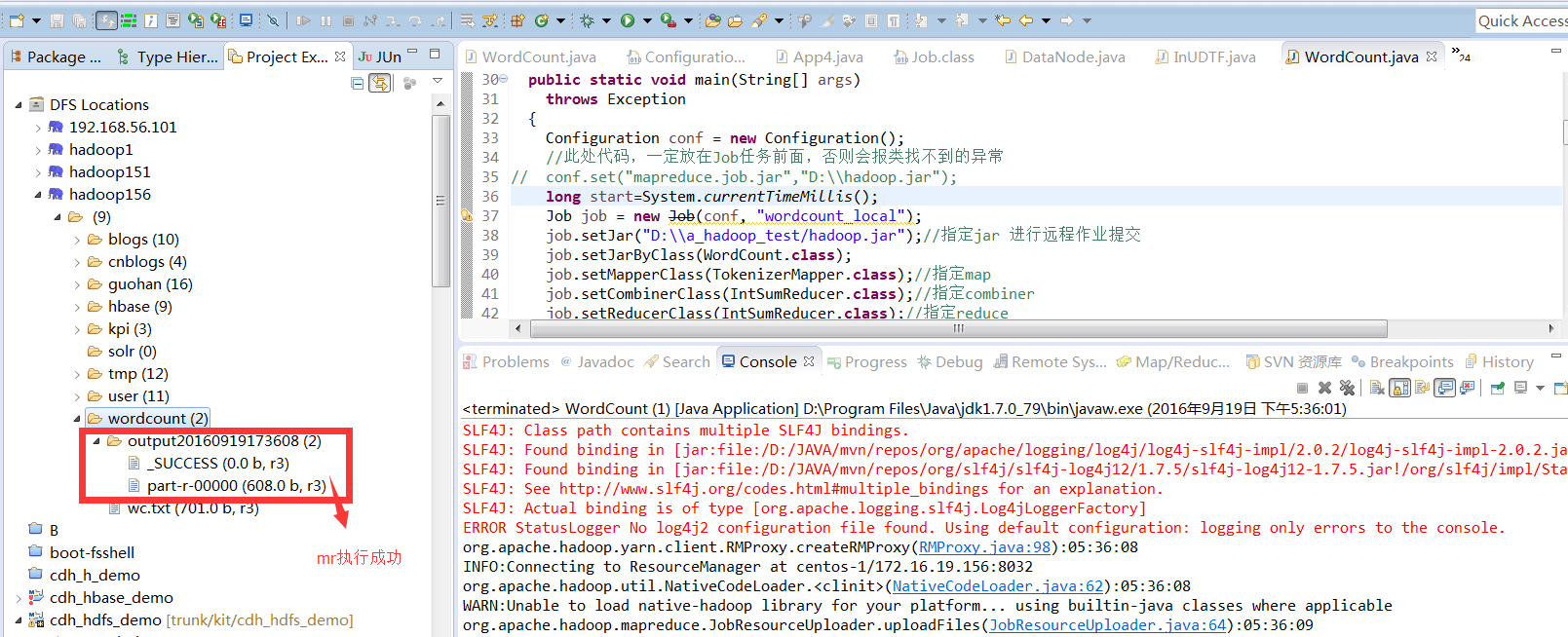
结果文件一般由三部分组成：   
1) \_SUCCESS文件：表示MapReduce运行成功。   
2) \_logs文件夹：存放运行MapReduce的日志。   
3) Part-r-00000文件：存放结果，也是默认生成的结果文件。   
使用hadoop fs -ls wordcount/output命令查看输出结果目录，如下所示











到这里，整个MapReduce的快速入门就结束了