**基于Hadoop的数据分析原理和实现**

# 一、概述

Hadoop起源于开源的网络搜索引擎Apache Nutch，Hadoop的出现得益于Google发表的三篇论文，这三篇论文分别得介绍了分布式文件系统GFS、并行计算模型MapReduce、非关系数据存储系统BigTable，这三篇论文第一次提出了针对大数据分布式处理的可重用方案，并提供了行之有效的理论基础。在这三篇论文的基础上，Doug Cutting和Mike Cafarella开发了Hadoop。同时在借鉴和改进Hadoop的基础上，又先后诞生了数十种应用于分布式环境的大数据计算框架。这才有了现在多彩多样的大数据生态系统。在Google论文的启发下。

众所周知，Hadoop是一个分布式系统基础架构，基于Hadoop，用户可以在不需要了解底层复杂的分布式原理的基础下，开发和运行一个分布式程序，从而充分的利用大量廉价的PC来完成一些复杂的计算任务，极大地降低了资源开销，并充分的发挥了集群的优势。

尽管Hadoop本身主要是因为MapReduce计算框架及其分布式文件系统hdfs而出名的，但是Hadoop这个名字也泛指一组相关的项目，这一些相关的项目都是使用的Hadoop这个基础平台进行的分布式计算和海量数据处理。这些项目包括Avro、MapReduce、HDFS、Pig、Hive、Sqoop、HBase等等。

# 二、环境搭建

## 2.1 Hadoop的三种运行方式

Hadoop有三种模式:单机模式、伪分布式模式、完全分布式模式。

### 2.1.1 单机模式

这种模式是Hadoop的默认模式，也是最简单的一种模式，配置此种Hadoop模式时，不需要对配置文件进行修改，Hadoop会以非分布式模式运行的一个独立Java进程存在，这种模式下的Hadoop有以下几个特点：

* 使用的是本地文件系统，而不是分布式文件系统。
* 不会启动NameNode、DataNode、JobTracker、TaskTracker等Hadoop守护进程
* Map任务和Reduce任务是作为同一个进程的不同部分来执行的，而不是单独的进程来执行

这种模式下的Hadoop比较容易配置，主要可以用于对MapReduce程序的逻辑进行调试，确保程序的正确。

### 2.1.2 伪分布式模式

相对于第一种模式，这种模式下的Hadoop主要是用于开发测试Hadoop程序的执行是否正确，此种模式下的Hadoop功能比较完善，基本相当于一个小型的Hadoop集群。这种模式的主要思想是在一台主机模拟多个主机。将一个分布式环境下的多个组件在同一台主机的多个线程中启动，使用多个守护线程模拟不同主机上的各个分布式组件，从而可以在一台主机上，模式分布式环境的操作。这种模式的Hadoop有以下几个特点：

* Hadoop为NameNode、DataNode、JobTracker、TaskTracker等守护进程各自启动一个Java进程，从而使这些守护进程都在同一台机器上运行，并且相互独立
* 使用分布式文件系统
* 各个作业是由JobTraker服务，来管理的独立进程
* 在单机模式之上增加了代码调试功能，允许检查内存使用情况，HDFS输入输出，以及其他的守护进程交互。类似于完全分布式模式

### 2.1.3完全分布式模式

这种模式下的Hadoop一般用于正式的生产环境，它真正的是使用多台机器来搭建分布式集群，各个Hadoop守护进程运行在由多台主机搭建的集群上，从而充分发挥Hadoop的能力。其特点如下：

* 所有的主机上都需安装JDK和Hadoop，组成相互连通的网络
* 主机间需设置SSH免密码登录
* 需指定NameNode和JobTraker等守护进程的位置和端口，设置文件的副本等参数

由于Hadoop主要是运行在Linux环境下，另外考虑到机器的限制，所以下面以Ubuntu14.04为例，演示如何搭建一个伪分布式的Hadoop开发环境。

## 2.2、JDK的安装与配置

(1)、自行前往Oracle官网下载JDK

|  |
| --- |
| vi /etc/profile |

(2)、下载后解压到相关路径，配置环境变量：

|  |
| --- |
| export JAVA\_HOME=/mnt/external/Java/jdk1.8.0\_121  export JAVA\_BIN=$JAVA\_HOME/bin  export JAVA\_LIB=$JAVA\_HOME/lib  export CLASSPATH=.:$JAVA\_LIB/tools.jar:$JAVA\_LIB/dt.jar  export PATH=$PATH:$JAVA\_HOME/bin |

(3)、刷新配置

|  |
| --- |
| source /etc/profile |

## 2.3配置SSH

### 2.3.1 安装ssh和rsync

|  |
| --- |
| sudo apt-get install ssh  sudo apt-get install rsync |

### 2.3.2 公钥的生成

(1)、用户主目录（cd ~），ls -a查看文件，其中一个为“.ssh”，该文件价是存放密钥的。生成的密钥都会放到这个文件夹中。

(2)、生成公钥和.ssh文件夹

|  |
| --- |
| ssh-keygen -t rsa |

遇到需要确认的就一路回车，不需要输入密码

### 2.3.3 密钥的生成

|  |
| --- |
| ssh-keygen -t rsa -P '' -f ~/.ssh/id\_rsa |
|  |

上面的命令式使用rsa加密方式生成密钥，-P是指示密码，-f则是指定生成的rsa秘钥的存放位置。这条命令会在.ssh目录下生成密钥对：id\_rsa和id\_rsa.pub

输入以上命令后回车，会提示三次输入信息，直接回车即可。

将生成的公钥id\_rsa.pub 内容追加到授权的authorized\_keys中去，执行命令：

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

### 2.3.4 环境验证

首先重启SSH服务：service sshd restart

执行命令：ssh localhost(主机名) 无错误提示则配置成功

因为是单主机伪分布式模式，所以不需要将Master节点的秘钥分发到各个slave节点去。

## 2.4 Hadoop的相关配置

官方网站下载到Hadoop，此处下载的是Hadoop-2.7.3版本的，下载后解压路径为/usr/Hadoop/Hadoop-2.7.3

### 2.4.1 Hadoop-env.sh的配置

进入到Hadoop解压目录下的./ etc/Hadoop，此处是Hadoop的配置文件的存放位置。打开编辑Hadoop-env.sh文件，在该文件的第27行处找到JAVA\_HOME的配置，将注释打开，并设置为：

|  |
| --- |
| JAVA\_HOME=/mnt/external/Java/jdk1.8.0\_121 |

此处配置为和上面的JDK的JAVA\_HOME环境变量一致即可

### 2.4.2 core-site.xml的配置

同样在Hadoop解压目录下的./ etc/Hadoop目录中打开该文件，在其<configuration>标签中加入如下内容：

|  |
| --- |
| <!-- 配置指定NameNode的地址 -->  <property>  <name>**fs.defaultFS**</name>  <value>**hdfs://master:9000**</value>  </property>  <property>  <name>**fs.default.name**</name>  <value>**hdfs://master:9000**</value>  </property>  <!-- 指定Hadoop运行时的临时文件存储目录-->  <property>  <name>**Hadoop.tmp.dir**</name>  <value>**/usr/local/Hadoop-2.7.3/tmp**</value>  </property> |

上面的master是当前主机的HOSTNAME，配置时需要根据自己电脑的HOSTNAME进行对应的修改

### 2.4.3 hdfs-site.xml的配置

Hadoop解压目录下的./ etc/Hadoop目录中打开该文件，在其<configuration>标签中加入如下内容：

|  |
| --- |
| <!-- 指定HDFS副本的数量 -->  <property>  <name>**dfs.replication**</name>  <value>**1**</value>  </property>  <!-- HDFS权限验证 -->  <property>  <name>**dfs.permissions.enabled**</name>  <value>**false**</value>  </property>  <!-- HDFS文件系统超级权限用户 -->  <property>  <name>**dfs.permissions.superusergroup**</name>  <value>**root**</value>  </property>  <!-- NameNode临时文件的存放目录 -->  <property>  <name>**dfs.NameNode.name.dir**</name>  <value>**file:/usr/local/Hadoop-2.7.3/tmp/dfs/name**</value>  </property>  <!-- datanode临时文件的存放目录 -->  <property>  <name>**dfs.datanode.data.dir**</name>  <value>**file:/usr/local/Hadoop-2.7.3/tmp/dfs/data**</value>  </property> |

### 2.4.4 mapred-site.xml的配置

进入方法同上，添加如下内容：

|  |
| --- |
| <!-- 指定MapReduce运行在YRAN上 -->  <property>  <name>**MapReduce.framework.name**</name>  <value>**YRAN**</value>  </property>  <!-- 指定任务是否可以跨平台提交 -->  <property>  <name>**MapReduce.app-submission.cross-platform**</name>  <value>**true**</value>  </property>  <!-- MapReduce应用程序的依赖包 -->  <property>  <name>**MapReduce.application.classpath**</name>  <value>  **/usr/local/Hadoop-2.7.3/etc/Hadoop,**  **/usr/local/Hadoop-2.7.3/share/Hadoop/common/\*,**  **/usr/local/Hadoop-2.7.3/share/Hadoop/common/lib/\*,**  **/usr/local/Hadoop-2.7.3/share/Hadoop/hdfs/\*,**  **/usr/local/Hadoop-2.7.3/share/Hadoop/hdfs/lib/\*,**  **/usr/local/Hadoop-2.7.3/share/Hadoop/MapReduce/\*,**  **/usr/local/Hadoop-2.7.3/share/Hadoop/MapReduce/lib/\*,**  **/usr/local/Hadoop-2.7.3/share/Hadoop/YRAN/\*,**  **/usr/local/Hadoop-2.7.3/share/Hadoop/YRAN/lib/\***  </value>  </property> |

### 2.4.5 YRAN-site.xml的配置

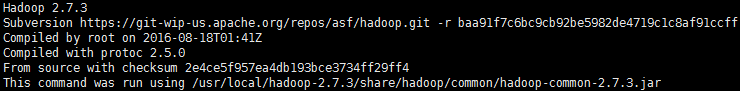
进入方法同上，添加如下内容：

|  |
| --- |
| <!-- 指定YRAN的老大（ResourceManager）的地址 -->  <property>  <name>**YRAN.resourcemanager.hostname**</name>  <value>**master**</value>  </property>  <!-- reducer获取数据的方式 -->  <property>  <name>**YRAN.nodemanager.aux-services**</name>  <value>**MapReduce\_shuffle**</value>  </property>  <!-- YRAN 应用环境 -->  <property>  <name>**YRAN.application.classpath**</name>  <value>  **/usr/local/Hadoop-2.7.3/etc/Hadoop,**  **/usr/local/Hadoop-2.7.3/share/Hadoop/common/\*,**  **/usr/local/Hadoop-2.7.3/share/Hadoop/common/lib/\*,**  **/usr/local/Hadoop-2.7.3/share/Hadoop/hdfs/\*,**  **/usr/local/Hadoop-2.7.3/share/Hadoop/hdfs/lib/\*,**  **/usr/local/Hadoop-2.7.3/share/Hadoop/MapReduce/\*,**  **/usr/local/Hadoop-2.7.3/share/Hadoop/MapReduce/lib/\*,**  **/usr/local/Hadoop-2.7.3/share/Hadoop/YRAN/\*,**  **/usr/local/Hadoop-2.7.3/share/Hadoop/YRAN/lib/\***  </value>  </property> |

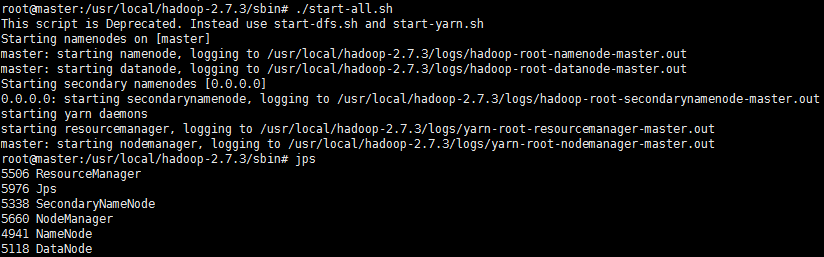
### 2.4.6 环境验证

上述文件全部配置好后，首先需要格式化NameNode，即对NameNode进行初始化。

|  |
| --- |
| hdfs NameNode -format 或 Hadoop NameNode -format |

 确认NameNode初始化成功后，则输入Hadoop version，显示如下

进入到Hadoop解压目录下的sbin目录中，运行start-all.sh脚本，启动NameNode各个守护进程。启动后输入JPS命令，显示如下，则环境正常：

****

### 2.4.7 安装过程中的常见问题

(1)、Hadoop频繁报警

|  |
| --- |
| Unable to load native-Hadoop library for your platform... using builtin-Java classes where applicable |

解决办法：将Hadoop根目录下etc/Hadoop/Hadoop-env.sh中的如下内容注释掉：

|  |
| --- |
| export HADOOP\_OPTS=  "$HADOOP\_OPTS -Djava.net.preferIPv4Stack=true" |

修改为：

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

(2)、启动NameNode进程，出现IllegalArgumentException异常

修改hdfs-site.xml 文件，将dfs.NameNode.name.dir和dfs.datanode.data.dir两个配置项下的路径改为绝对路径，不要用相对路径

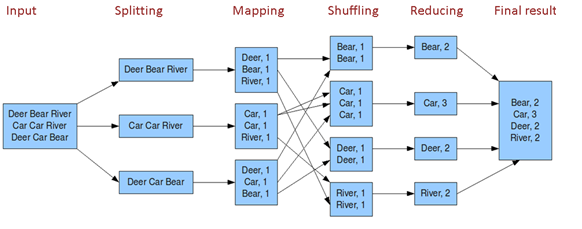
# 三、MapReduce和YRAN

Hadoop发展至今，已经经历了三个重大版本的更新，目前使用最多的是2.X版本的Hadoop

## 3.1 MapReduce的基本概念

在概要中提到，Hadoop的基本原理来自于谷歌公司的三篇论文，其中很重要的一篇就是介绍并行计算模型MapReduce，所谓MapReduce是由两个基本的步骤Map(映射)和Reduce(化简)组成，Map/Reduce由此得名，简单的理解可以认为，Map是从海量的数据源中将需要分析的数据挑选出来，并尽可能的均匀分区，而Reduce则是按照自定义的统计的逻辑，对Map分区的数据进行处理后输出。对应这两个过程，分布式程序编辑人员需要编写两个基本的函数：Map函数和Reduce函数。

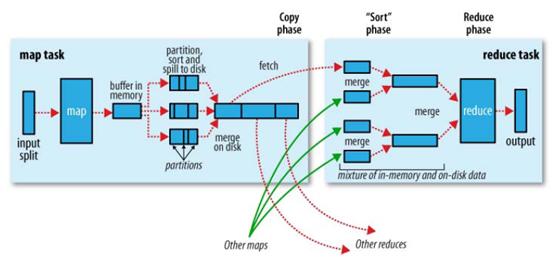
下图是MapReduce处理过程，各阶段数据的一个简单图形。



上图便是一个简单的MapReduce单词计数功能的数据变化过程：Split（分片）、Map（映射）、Shuffle（混选）、Reduce（化简）。从原理上看，

## 3.2 MapReduce的原理

MapReduce的处理过程可参考如下官方文档的流程图



上图中按先后顺序，其处理逻辑如下：

**(1)**、在进行map计算之前，MapReduce会根据输入文件计算输入分片（split），每个输入分片针对一个map任务，输入分片存储的并非数据本身，而是一个分片长度和一个记录数据的位置的数组。

**(2)**、map函数读取输入分片的数据，根据自定义的业务逻辑产生输出，Map任务在执行时会产生溢出（spill）文件，溢出文件首先会写到缓存区，缓冲区写满后会写到磁盘。写磁盘之前，线程会首先根据最终的reduce个数，把数据排划分为相应的分区（partition），并根据数据所属的分区排序（sort），在每个分区内，后台线程按键进行内排序（sort）。Map完成之后，溢出文件会被合并成一个已分区且已排序的输出文件。

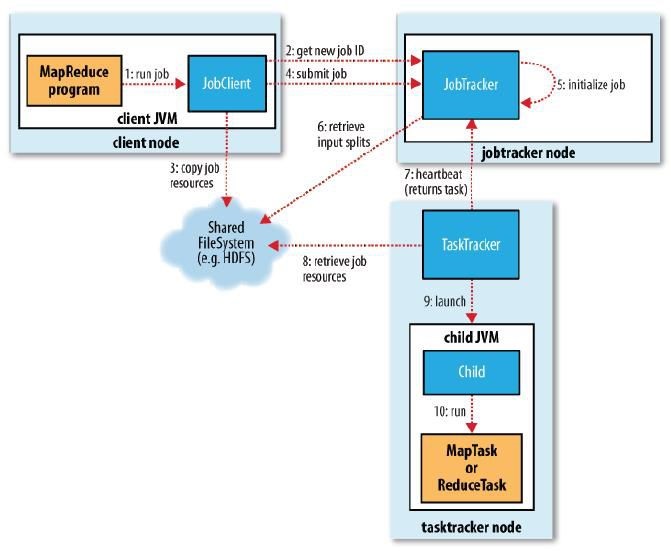
**(3)**、Map完成后，转到Reduce阶段，一般而言，一个reduce任务可能需要多个map任务的输出作为其输入，而每个map任务的完成时间各不相同，所以只要有一个map任务完成，reduce就会开始复制其输出（Copy）,这个交互是通过心跳机制完成的，之后reduce会维持map输出的排序，合并（merge）map输出，并对输出中的每个键调用reduce函数。

至于shuffle阶段，其实就是系统将map输出作为输入传递给reduce的过程，它将数据打乱再整理，这样出来的数据就是干净有序的了。

## 3.3 经典的MapReduce作业原理

### 3.3.1 MapReduce1作业原理图

以下是官方文档中的V1.0版本的Hadoop作业处理流程图

****

### 3.3.2 MapReduce1各组件功能

在上面的官方流程图中，主要包含四个重要的组件：

(1)、JobClient：提交MapReduce作业的客户端

(2)、HDFS：分布式文件系统，存储了输入、输出分片、各种中间临时文件以及各种共享作业文件等

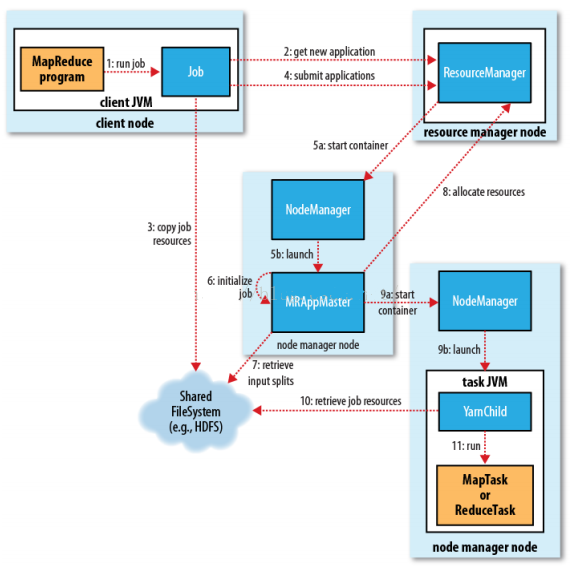
(3)、JobTracker：接收JobClient提交的分布式计算请求，向TaskTracker分发计算任务，管理和协调各种计算资源

(4)、TaskTracker：向下负责具体的Map、Reduce任务的执行，并通过心跳机制与JobTracker汇报各个子任务的进度和状态

## 3.4 YRAN

### 3.4.1 MapReduce2作业原理图

以下是官方文档中的V2.0版本的Hadoop作业处理流程图

****

### 3.4.2 MapReduce2各组件功能

(1)、ResourceManager(RM)：负责管理所有应用程序计算资源的分配

(2)、ApplicationMaster(AM)：每一个应用程序的AM负责相应的任务调度、协调及任务生命周期的管理，它和MapReduce任务一起在Container中运行，这些Container由RM分配，并由NM进行管理

(3)、NodeManager(NM)：管理各个子节点，代替以前的TaskTracker,不过功能基本类似，只不过添加了每个节点的的自管理功能

(4)、Client：提交MapReduce作业的客户端

(5)、HDFS：分布式文件系统

(6)、Container：YRAN为将来的资源隔离而提出的框架，每一个任务都对应一个容器，且任务只能在容器中执行，每个容器都有特定的资源限制

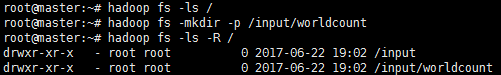
相对于MapReduce1，YRAN将JobTracker的功能拆分为两个独立的守护进程：RM和AM，分别负责相应的职责，从而解决了TaskTracker负担过重的问题，避免了JobTracker可能的单点故障问题。

# 四、代码实现

## 4.1 输入文件准备

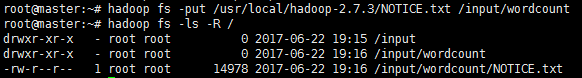
编写MapReduce任务之前，需要先准备数据，此处上传一个文件到分布式文件系统中，编写一个简单的单词数目统计的程序

(1)、创建输入文件夹



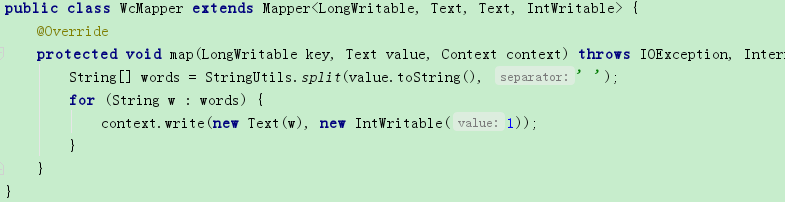
(2)、上传待分析的数据源

此处以Hadoop解压目录下的NOTICE.txt为数据源进行分析



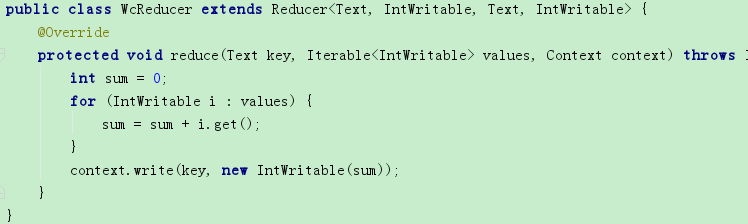
可以看到，该文件已经成功上传到HDFS分布式文件系统中，该文件大小为14976字节

## 4.2 Map函数



Map函数会读取输入分片中的数据，此处由于未设置，所以Hadoop默认会一行一行的解析，即此处的key是偏移量，value是一行一行的值，此处代码中，将一行的值按空格分隔，每个单词都记为一次，然后进入到shuffle阶段。Shuffle会将相同的单词放到同一个分区中，以待reduce函数读取数据。

## 4.3 reduce函数

****

Reduce读取分区中的数据，对于此程序来说，此处的key为map函数写入的某个单词，value为map函数为每个单词写入的值，此处为IntWritable(1)，此对象为Hadoop封装的可序列化的整型对象。这段代码相当于将每一个单词对应的IntWritable(1)相加后写入到输出目录中，最后得到的就是每个单词对应的出现的次数。

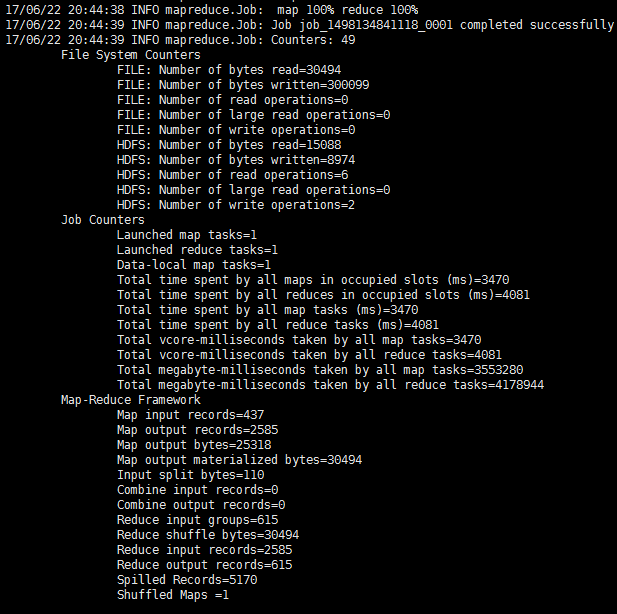
## 4.4 主函数

****

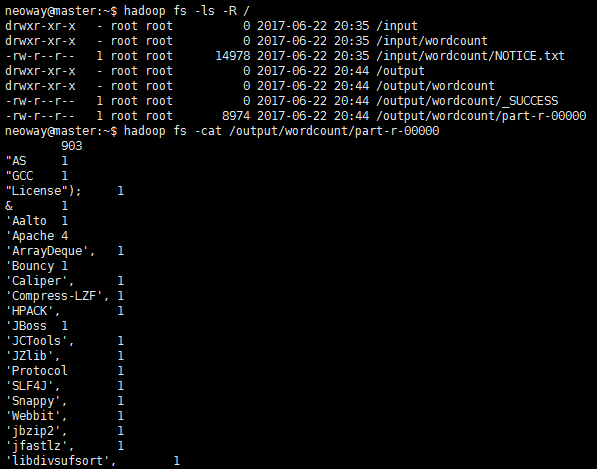
上面的主调度函数，主要是设置任务名称、提交任务的用户、输入输出文件的位置、Map和Reduce函数、map和reduce函数的输入输出数据类型等基本参数。

## 4.5 结果查看

运行日志：



运行结果：

****