数据分布式存储

3.1 教学目标

1. 能力目标

(1) 能够根据项目实际,恰当选用 Hadoop、Ceph 和 HBase 产品。
(2) 能够根据工程实际,基于 Hadoop、Ceph 和 HBase 设计存储系统。
(3) 能够基于项目需求,开发访问 Hadoop、Ceph 和 HBase 存储系统的应用程序。
(4) 能够基于现有存储系统的限制,进行最大限度的数据分布式存储方案的改进。

2. 素质目标

(1) 能够准确撰写 Hadoop、Ceph 和 HBase 存储系统的设计文档。
(2) 能够翔实撰写 Hadoop、Ceph 和 HBase 存储系统的搭建文档。

3.2 Hadoop 分布式存储

3.2.1 Hadoop 3.1.1 伪分布式集群环境搭建

1. 设置基础环境

以 root 身份分别在 3 台计算机上,编辑网络配置文件,设置主机名称,执行如下的命令。

```
#vim /etc/sysconfig/network
```

打开文件后,将名字节点的 hostname 改为 master,将 2 个数据节点的 hostname 改为 slave1 和 slave2,将所有节点的 networking 设置为 yes。

然后,在主节点设置主机名,执行如下的命令。

```
#hostnamectl set-hostname  master
```

类似地,在 2 个数据节点设置主机名,分别执行如下的命令。

```
#hostnamectl set-hostname  slave1
#hostnamectl set-hostname  slave2
```

最后，分别在 3 台机器上执行 su 命令，以使主机名生效。

2. 添加全部节点 IP 与主机名的映射

先获取管理员权限（默认后面的命令都已获取该权限），执行如下的命令。

```
#sudo su
```

接着修改主机名映射文件，以设置 IP 地址和机器名称的对应关系，编辑主机名映射文件，执行如下的命令。

```
#vi /etc/hosts
```

打开文件后，在末尾追加 master、slave1 和 slave2 与 IP 地址的对应关系，如下所示。

```
192.168.50.194 master
192.168.50.190 slave1
192.168.50.191 slave2
```

上面的映射表明主节点的主机名为 master，2 个数据节点的主机名分别为 slave1 和 slave2。

3. 在 3 台机器上安装 JDK

Hadoop 3.1.1 需要安装 jdk-8u181-linux-x64.rpm。首先查看版本是否满足需求，如果不满足，则应先卸载，再安装 jdk-8u181-linux-x64.rpm。

（1）检验系统原版本，执行如下的命令。

```
#java -version
```

进一步查看 JDK 信息，执行如下的命令。

```
#rpm -qa | grep java
```

（2）卸载 OpenJDK，执行如下的命令。

```
#rpm -e --nodeps java-1.8.0-openjdk-1.8.0.131-11.b12.el7.x86_64
#rpm -e --nodeps nuxwdog-client-java-1.0.3-5.el7.x86_64
#rpm -e --nodeps javassist-3.26.1-10.el7.noarch
#rpm -e --nodeps pki-base-java-10.4.1-10.el7.noarch
#rpm -e --nodeps tzdata-java-2017b-1.el7.noarch
#rpm -e --nodeps python-javapackages-3.5.1-11.el7.noarch
#rpm -e --nodeps javamail-1.4.6-8.el7.noarch
#rpm -e --nodeps javapackages-tools-3.5.1-11.el7.noarch
#rpm -e --nodeps java-1.8.0-openjdk-headless-1.8.0.131-11.b12.el7.x86_64
```

（3）安装 JDK。

在 Oracle 官网下载 jdk-8u181-linux-x64.rpm 到 Windows 桌面环境中，用 WinSCP 软件将 jdk-8u181-linux-x64.rpm 上传到 3 台虚拟机的/usr/local/目录下，如图 3-1 所示。

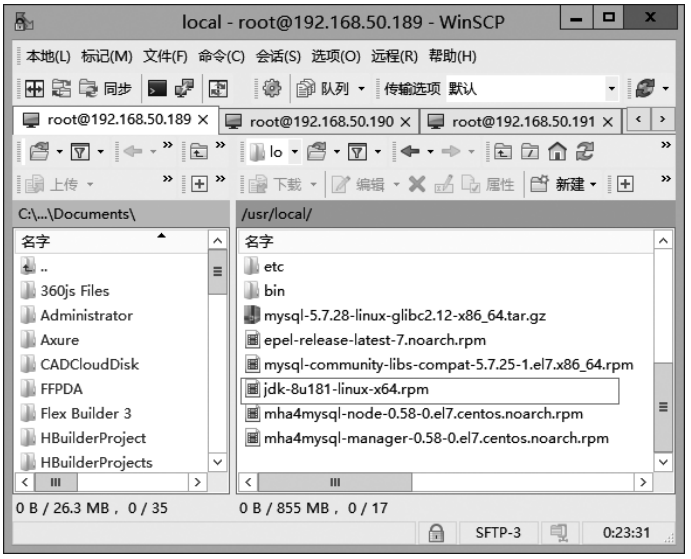

图 3-1 用 WinSCP 把 JDK 包从宿主机复制到虚拟机

然后,分别在 3 台虚拟机中执行以下的命令。

```
#cd /usr/local
#rpm -ivh jdk-8u181-linux-x64.rpm
```

安装成功后,JDK 默认安装在/usr/java 中。查看安装是否成功,执行以下的命令。

```
#java -version
```

(4) 配置环境变量。

Linux 是一个多用户的操作系统。每个用户登录系统后,都有一个专用的运行环境。通常每个用户默认的环境都是相同的,这个默认环境实际上由一组环境变量所定义。用户可以对自己的运行环境进行定制,其方法是修改相应的系统环境变量。常在/etc/profile 文件中修改环境变量,本书中对环境变量的修改对所有用户都起作用。

在 Hadoop 集群的 3 台虚拟机上修改系统环境变量文件,执行如下的命令。

```
#vi /etc/profile
```

打开 profile 文件后,向文件末尾追加以下内容。

```
JAVA_HOME=/usr/java/jdk1.8.0_181-amd64
JRE_HOME=/usr/java/jdk1.8.0_181-amd64/jre
PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin
CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar:$JRE_HOME/lib
```

保存文件后,为使得修改生效,执行如下的命令。

```
#source /etc/profile
```

为了验证输出工作目录是否正确，执行如下的命令。

```
#echo $PATH
```

为每一个运行 bash shell 的用户执行此文件。当 bash shell 被打开时，该文件被读取。修改.bashrc 文件，执行以下的命令。

```
#vim ~/.bashrc
```

打开文件后，向文件末尾追加如下的内容。

```
export JAVA_HOME=/usr/java/jdk1.8.0_181-amd64
export JRE_HOME=$JAVA_HOME/jre
export CLASSPATH=.:$CLASSPATH:$JAVA_HOME/lib:$JRE_HOME/lib
export PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin
```

保存后退出，为使修改生效，执行如下的命令。

```
#source ~/.bashrc
```

为验证输出工作目录是否正确，执行如下的命令。

```
#echo $JAVA_HOME
```

4. SSH 设置免密登录

设置自身免密登录，执行如下的命令。

```
#yum install openssh-server
```

若上面的安装提示不成功，需要先创建一个目录。

(1) 在 3 个节点依次执行如下的命令。

```
#cd ~/.ssh/
#rm -rf *
#ssh-keygen -t rsa
```

(2) 在本机进行免密登录测试，执行如下的命令。

```
#ssh master      //在 master 上执行
#ssh slave1      //在 slave1 上执行
#ssh slave2      //在 slave2 上执行
```

(3) 传送免密登录密钥到其他节点，使得节点间免密登录。

在 master 节点执行如下的命令。

```
cd ~/.ssh/
mv id_rsa.pub id_rsa_189.pub
scp id_rsa_189.pub slave1:~/.ssh/
scp id_rsa_189.pub slave2:~/.ssh/
```

在 slave1 节点执行如下的命令。

```
cd ~/.ssh/
mv id_rsa.pub id_rsa_190.pub
scp id_rsa_190.pub master:~/.ssh/
scp id_rsa_190.pub slave2:~/.ssh/
```

在 slave2 节点执行如下的命令。

```
cd ~/.ssh/
mv id_rsa.pub id_rsa_191.pub
scp id_rsa_191.pub master:~/.ssh/
scp id_rsa_191.pub slave1:~/.ssh/
```

在 3 个节点执行如下的命令。

```
cat id_rsa_189.pub >>authorized_keys
cat id_rsa_190.pub >>authorized_keys
cat id_rsa_191.pub >>authorized_keys
```

(4) 在 master 节点登录其他 2 个节点,执行如下的命令。

```
#ssh slave1
#exit
#ssh slave2
#exit
```

上面 master 节点分别免密登录 slave1 和 slave2 节点进行测试,执行 exit 指令退出相应的数据节点。

(5) 在 slave1 和 slave2 节点登录 master 节点,执行如下的命令。

```
#ssh master
```

测试成功后执行 exit 指令退出 master 节点登录。

5. 在 3 个节点安装 Hadoop

在 Hadoop 官网下载 Hadoop3.1.1,选择 binary 格式,文件为 hadoop-3.1.1.tar.gz,在 Windows 桌面环境中使用 WinSCP 将其上传到每个虚拟机节点的/usr/local/后,执行如下的命令。

```
#cd /usr/local
#tar -zxvf /usr/local/hadoop-3.1.1.tar.gz -C /usr/local
#cd /usr/local
#mv ./hadoop-3.1.1 ./hadoop          #将文件夹名改为 hadoop
```

在系统环境配置文件中添加 Hadoop 相关环境,即在~/.bashrc 文件的 JAVA_HOME 末尾追加,打开文件,执行如下的命令。

```
#vim ~/.bashrc
```

打开文件后，下载 3-3-1-5-Hadoop 安装 bashrc 文件，更新当前.bashrc 内容。同样需要执行 source ~/.bashrc，以使修改生效，再执行 HDFS 观察是否出现命令帮助提示，执行如下的命令。

```
#source ~/.bashrc
#hdfs
```

6. Hadoop 配置

首先，在 master 节点配置/usr/local/hadoop/etc/hadoop/下的 6 个相关配置文件 hadoop-env.sh、core-site.xml、hdfs-site.xml、yarn-site.xml、mapred-site.xml、workers。

(1) 修改 hadoop-env.sh，配置 Hadoop 运行中使用的变量，执行如下的命令。

```
#cd /usr/local/hadoop/etc/hadoop/
#vim hadoop-env.sh
```

打开文件后，下载 3-3-1-6-Hadoop 配置 env 文件，更新当前 hadoop-env.sh 内容。

(2) 修改 core-site.xml 文件，配置文件系统，执行如下的命令。

```
#vim core-site.xml
```

打开文件后，下载 3-3-1-6-Hadoop 配置 core-site 文件，更新当前 core-site.xml 内容。

(3) 修改 hdfs-site.xml 文件，配置文件系统和相关协议的访问地址，执行如下的命令。

```
#vim hdfs-site.xml
```

打开文件后，下载 3-3-1-6-Hadoop 配置 hdfs-site 文件，更新当前 hdfs-site.xml 内容。

(4) 修改 yarn-site.xml，配置 YARN 资源管理器的有关参数，执行如下的命令。

```
#vim yarn-site.xml
```

打开文件后，下载 3-3-1-6-Hadoop 配置 yarn-site 文件，更新当前 yarn-site.xml 内容。

(5) 修改 mapred-site.xml，配置 MapReduce 的有关参数，执行如下的命令。

```
#vim mapred-site.xml
```

打开文件后，下载 3-3-1-6-Hadoop 配置 mapred-site 文件，更新当前 mapred-site.xml 内容。

(6) 修改 workers，设置数据节点，执行如下的命令。

```
#vim workers
```

打开文件后，向文件末尾追加如下的配置。

```
slave1
slave2
```

(7) 保证 3 个节点配置一致,将 master 节点的配置文件复制到集群其他节点,在 master 机器上执行如下的命令。

```
#scp hadoop-env.sh root@slave1:/usr/local/hadoop/etc/hadoop/
#scp core-site.xml root@slave1:/usr/local/hadoop/etc/hadoop/
#scp hdfs-site.xml root@slave1:/usr/local/hadoop/etc/hadoop/
#scp mapred-site.xml root@slave1:/usr/local/hadoop/etc/hadoop/
#scp yarn-site.xml root@slave1:/usr/local/hadoop/etc/hadoop/
#scp workers root@slave1:/usr/local/hadoop/etc/hadoop/
#scp hadoop-env.sh root@slave2:/usr/local/hadoop/etc/hadoop/
#scp core-site.xml root@slave2:/usr/local/hadoop/etc/hadoop/
#scp hdfs-site.xml root@slave2:/usr/local/hadoop/etc/hadoop/
#scp mapred-site.xml root@slave2:/usr/local/hadoop/etc/hadoop/
#scp yarn-site.xml root@slave2:/usr/local/hadoop/etc/hadoop/
#scp workers root@slave2:/usr/local/hadoop/etc/hadoop/
```

(8) 在 3 个节点上分别创建 Hadoop 配置对应的目录,执行如下的命令。

```
mkdir /usr/hadoop
mkdir /usr/hadoop/tmp
mkdir /usr/local/hadoop/hdfs
mkdir /usr/local/hadoop/hdfs/name
mkdir /usr/local/hadoop/hdfs/data
```

7. *启动* Hadoop

(1) 格式化 namenode。

第一次启动需在 master 节点进行格式化操作,执行如下的命令。

```
#hdfs namenode -format
```

如果提示信息中出现“/usr/local/hadoop/hdfs/name has been successfully formatted.”,表示格式化成功。

(2) 启动集群服务。

在 master 节点启动集群,执行如下的命令。

```
#cd /usr/local/hadoop
#sbin/start-all.sh
```

验证集群启动是否成功,在 3 个节点分别执行 jps,查看启动服务情况。首先在 master 节点执行 jps 命令。

显示结果如下。

```
13780 NameNode
14443 ResourceManager
14875 Jps
```

```
14175 SecondaryNameNode
```

出现上面 4 个服务进程信息表示 master 节点作为名字节点、资源管理器、备用名字节点启动成功。

在 slave1 节点上执行 jps 命令，显示结果如下。

```
13880 NodeManager
14202 Jps
13755 DataNode
```

出现上面 3 个服务进程信息表示 slave1 节点作为数据节点、节点管理器启动成功。在 slave2 节点上执行 jps，与 slave1 节点类似，不再累述。

8. 用 Web 浏览器查看集群服务

在浏览器中访问 master：50070，Hadoop 集群的概览如图 3-2 所示，Hadoop 集群信息如图 3-3 所示。

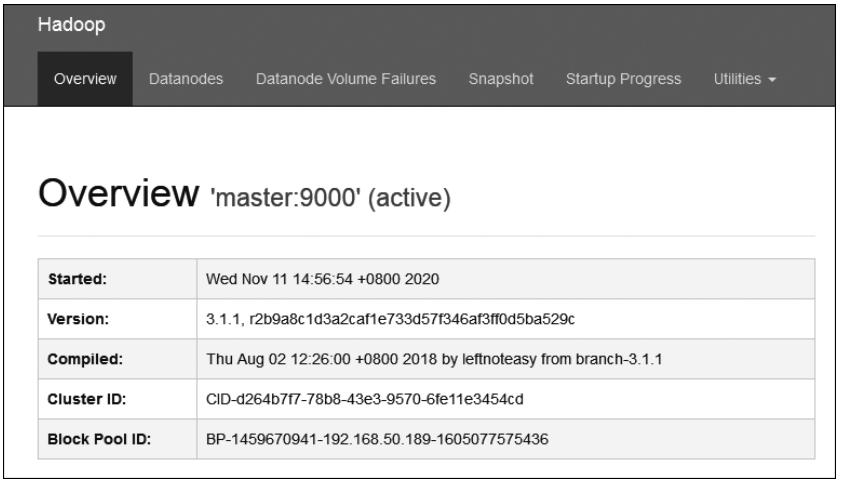

图 3-2　Hadoop 集群的概览

至此，完成整个安装以及配置过程。

9. 关闭集群服务

在 master 节点上执行如下的命令。

```
#sbin/stop-all.sh
```

10. 重新格式化

当启动 Hadoop 失败或者首次格式化失败，需要重新格式化。如果执行格式化后，slave1 或 slave2 节点中的 Datanode 无法启动，可尝试关闭 Hadoop 集群，删除 master 节点中/usr/local/hadoop/hdfs/name/目录以及 slave1 或 slave2 节点中/usr/local/hadoop/

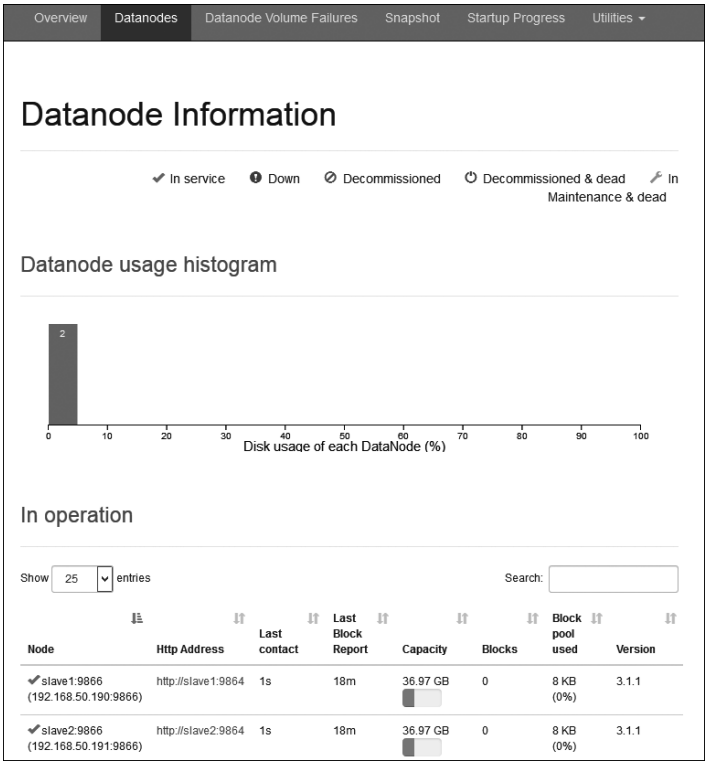

图 3-3 Hadoop 集群数据节点信息

hdfs/data/目录下的所有文件，再到 master 节点执行格式化，使 Namenode 和 Datanode 中的 ClusterID 一致。

11. WordCount 运行详解

(1) MapReduce 编程模型。

MapReduce 采用“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数据节点共同完成，然后通过整合各个数据节点的中间结果，得到最终结果。简而言之，MapReduce 是“任务的分解与结果的汇总”。

在 Hadoop 中用于执行 MapReduce 任务的机器角色有两个，一个是 JobTracker，另一个是 TaskTracker。JobTracker 用于调度工作，TaskTracker 用于执行工作。一个 Hadoop 集群中只有一个 JobTracker。

在分布式计算中，MapReduce 框架负责处理并行编程中分布式存储、工作调度、负载均衡、容错均衡、容错处理以及网络通信等复杂问题，把处理过程高度抽象为 map()和 reduce()两个函数，map()负责把任务分解成多个任务，reduce()负责把分解后多任务处理的结果汇总。

用 MapReduce 来处理的数据集(或任务)必须可以分解成许多小的数据集，而且所有小数据集可以完全并行地进行处理。

(2) MapReduce 处理过程。

如图 3-4 所示,在 Hadoop 中每个 MapReduce 任务都被初始化为一个 Job,每个 Job 分为 map 阶段和 reduce 阶段。这两个阶段分别用 map()函数和 reduce()函数来实现。map()函数接收一个<key,value>形式的输入,然后同样产生一个<key,value>形式的中间输出,reduce()函数接收一个如<key,(list of values)>形式的输入,然后对这个 value 集合进行处理,每个 reduce()产生 0 或 1 个输出,reduce()的输出也是<key,value>形式。

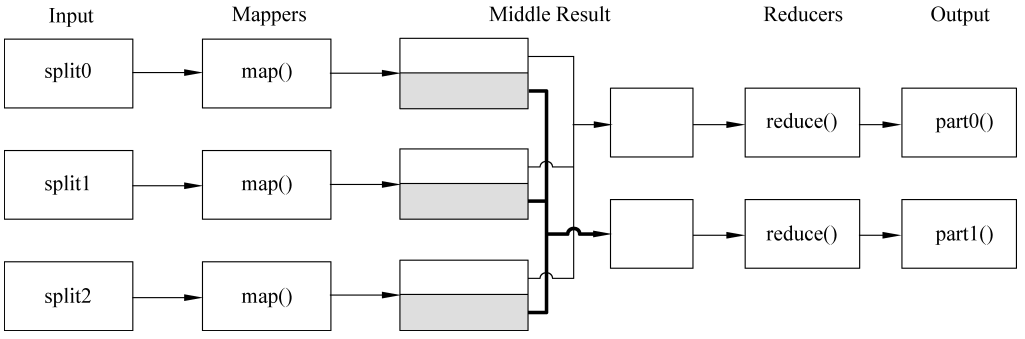

图 3-4 MapReduce 处理大数据的过程

(3) 运行 WordCount 程序。

单词计数以最简单的样例体现 MapReduce 思想,称为 MapReduce 版的 Hello World,该程序的完整代码可以在 Hadoop 安装包的 examples 目录下找到。单词计数的主要功能是统计某文本文件中每个单词出现的次数。

为实现单词计数,在 master 节点执行如下的命令。

```
#hadoop fs -chmod -R 777 /
#hadoop fs -mkdir /input
#hadoop fs -ls /
#hadoop fs -put LICENSE.txt /input
#hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.1.jar
wordcount /input /output
```

查看结果文件和运行结果,执行如下的命令。

```
#hadoop fs -ls /output
```

命令执行结果显示如下。

```
Found 2 items
-rw-r--r--   1 root supergroup          0 2020-11-11 17:02 /output/_SUCCESS
-rw-r--r--   1 root supergroup      34795 2020-11-11 17:02 /output/part-r-00000
```

上面的输出信息表示单词计数成功并存储在 output 目录,查看单词计数统计结果,执行如下的命令。

```
#hadoop fs -cat /output/part-r-00000
```

命令执行结果显示如下。

```
......
'Your'        2
'You'         4
'as           1
'commercial'  3
'control'     2
```

12. WordCount 源码分析

(1) 特别数据类型介绍。

Hadoop 提供的数据类型都实现了 WritableComparable 接口，以便用这些类型定义的数据可以被序列化进行网络传输和文件存储，以及进行大小比较，具体数据类型如下。

BooleanWritable：标准布尔型数值。

ByteWritable：单字节数值。

DoubleWritable：双字节数。

FloatWritable：浮点数。

IntWritable：整型数。

LongWritable：长整型数。

Text：使用 UTF8 格式存储的文本。

NullWritable：当<key,value>中的 key 或 value 为空时使用。

(2) 各阶段说明。

• JobConf 具体配置项如下所列。

setInputFormat：设置 map 的输入格式，默认为 TextInputFormat，key 为 LongWritable，value 为 Text。

setNumMapTasks：设置 map 任务的个数，此设置通常不起作用，map 任务的个数取决于输入的数据所能分成的 input split 的个数。

setMapperClass：设置 Mapper，默认为 IdentityMapper。

setMapRunnerClass：设置 MapRunner，map task 是由 MapRunner 运行的，默认为 MapRunnable，其功能为顺次读取 input split 的全部 record，依次调用 Mapper 的 map() 函数。

setMapOutputKeyClass 和 setMapOutputValueClass：设置 Mapper 输出对 key-value 的格式。

setOutputKeyClass 和 setOutputValueClass：设置 Reducer 输出对 key-value 的格式。

setPartitionerClass 和 setNumReduceTasks：设置 Partitioner，默认为 HashPartitioner，其根据 key 的 Hash 值来决定进入哪个 partition，每个 partition 被一个 Reduce Task 处理，所以 partition 的个数等于 Reduce Task 的个数。

setReducerClass：设置 Reducer，默认为 IdentityReducer。

setOutputFormat：设置任务的输出格式，默认为 TextOutputFormat。

FileInputFormat.addInputPath：设置输入文件的路径（可以是一个文件、一个路径、一个通配符），可以被调用多次添加多个路径。

FileOutputFormat.setOutputPath：设置输出文件的路径，在 Job 运行前此路径不应该存在。

JobConf 对象制定作业执行规范，构造函数的参数为作业所在的类，Hadoop 会通过该类来查找包含该类的 JAR 文件。

构造 JobConf 对象后，指定输入和输出数据的路径。本书通过 FileInputFormat 的静态方法 addInputPath() 来定义输入数据的路径，路径可以是单个文件，也可以是目录（即目录下的所有文件）或符合特定模式的一组文件，可以多次调用 addInputPath()方法。

同理，FileOutputFormat.setOutputPath() 指定输出路径，即写入目录。运行作业前，如果写入目录不应该存在，Hadoop 会拒绝并报错。这样设计主要是防止数据丢失，因为 Hadoop 运行时间长。

FileOutputFormat.setOutputPath() 和 conf.setMapperClass() 指定 map() 和 reduce() 类型。

接着，setOutputKeyClass 和 setOutputValueClass 指定 map()和 reduce()函数的输出类型，这两个函数的输出类型往往相同。如果不同，map()函数的输出类型通过 setMapOutputKeyClass 和 setMapOutputValueClass 指定。

输入的类型用 InputFormat 设置，本例中没有指定，使用默认的 TextInputFormat。最后 JobClient.runJob()会提交作业并等待完成，将结果写到控制台。

- MapReduce 的处理过程主要涉及以下 4 个部分。
  - 客户端 Client：用于提交 MapReduce 任务 Job。
  - JobTracker：协调整个 Job 的运行。它是一个 Java 进程，其 main class 为 JobTracker。
  - TaskTracker：运行此 Job 的 task，处理 input split。它是一个 Java 进程，其 main class 为 TaskTracker。
  - HDFS：Hadoop 分布式文件系统，在各个进程间共享 Job 相关的文件。JobClient.runJob()创建一个新的 JobClient 实例，调用其 submitJob()函数。JobClient 实例的作用依次为向 JobTracker 请求一个新的 Job ID、检测此 Job 的 output 配置、计算此 Job 的 input splits 和将 Job 运行所需的资源复制到 JobTracker 的文件系统中的文件夹中（包括 Job Jar 文件、job.xml 配置文件和 input splits）、通知 JobTracker 此 Job 已经可以运行。

提交任务后，runJob 每隔一秒钟轮询一次 Job 的进度，将进度返回到命令行，直到任务运行完毕。当 JobTracker 收到 submitJob 调用的时候，将此任务放到一个队列中，Job 调度器将从队列中获取任务并初始化任务。

初始化首先创建一个对象来封装 Job 运行的 tasks、status 以及 progress。在创建 task 之前，Job 调度器首先从共享文件系统中获得 JobClient 计算出的 input splits。其为每个 input split 创建一个 map task。每个 task 被分配一个 ID。

TaskTracker 周期性地向 JobTracker 发送 heartbeat。在 heartbeat 中，TaskTracker 告知 JobTracker 其已经准备运行一个新的 task，JobTracker 将分配给其一个 task。

在 JobTracker 为 TaskTracker 选择一个 task 之前，JobTracker 必须首先按照优先级选择一个 Job，在最高优先级的 Job 中选择一个 task。

TaskTracker 有固定数量的位置来运行 map task 或者 reduce task。默认的调度器对待 map task 优先于 reduce task。当选择 reduce task 的时候，JobTracker 不是在多个 task 之间选择，而是直接取下一个，因为 reduce task 没有数据本地化的概念。

TaskTracker 被分配了一个 task 后便运行此 task。首先，TaskTracker 将此 Job 的 Jar 从共享文件系统中复制到 TaskTracker 的文件系统中。TaskTracker 从 distributed cache 中将 Job 运行所需要的文件复制到本地磁盘。其次，其为每个 task 创建一个本地的工作目录，将 Jar 解压缩到文件目录中。最后，其创建一个 TaskRunner 来运行 task。

TaskRunner 创建一个新的 JVM 来运行 task。被创建的 child JVM 和 TaskTracker 通信来报告运行进度。

- Map 的过程：MapRunnable 从输入 split 中逐个读取记录，然后依次调用 Mapper 的 map()函数，将结果输出。map()的输出不是直接写入硬盘，而是将其写入缓存 memory buffer。当 buffer 中数据到达一定规模，一个背景线程将数据开始写入硬盘。在写入硬盘之前，内存中的数据通过 partitioner 分成多个 partition。在同一个 partition 中，背景线程会将数据按照 key 在内存中排序。每次从内存向硬盘 flush 数据，都生成一个新的 spill 文件。当此 task 结束之前，所有的 spill 文件被合并为一个被分区而且排好序的文件。Reducer 可以通过 HTTP 协议请求 map 的输出文件，tracker.http.threads 可以设置 HTTP 服务线程数。
- Reduce 的过程：当 map task 结束后，其通知 TaskTracker，TaskTracker 通知 JobTracker。对于一个 Job，JobTracker 知道 TaskTracer 和 map 输出的对应关系。Reducer 中一个线程周期性地向 JobTracker 请求 map 输出的位置，直到其取得了所有的 map 输出。reduce task 需要其对应的 partition 的所有的 map 输出。Reduce task 中的复制是当每个 map task 结束时开始复制输出，因为不同的 map task 完成时间不同。Reduce task 中有多个 copy 线程，可以并行复制 map 输出。当很多 map 输出复制到 reduce task 后，一个背景线程将其合并为一个大的排好序的文件。当所有的 map 输出都复制到 Reduce task 后，进入 sort 过程，将所有的 map 输出合并为大的排好序的文件。最后进入 Reduce 过程，调用 Reducer 的 reduce()函数，处理排好序的输出的每个 key，最后的结果写入 HDFS。
- 任务结束：当 JobTracker 获得最后一个 task 的运行成功的报告后，将 Job 的状态改为成功。当 JobClient 从 JobTracker 轮询的时候，发现此 Job 已经成功结束，则向用户打印消息，从 runJob()函数中返回。

(3) 新的 WordCount 分析

下载 3-3-1-14-MapReduce-WordCount 源程序及分析文件，分析 map、reduce 和任务执行的具体实现。

13. WordCount 处理过程

WordCount 详细的执行步骤如下。

(1) 将文件拆分成 splits，并将文件按行分割形成<key，value>对。以文件内容“Hello World Bye World Hello Hadoop Bye Hadoop”为例，如图 3-5 所示。这一步由 MapReduce 框架自动完成，其中偏移量(即 key 值)包括了回车所占的字符数(Windows 和 Linux 环境会不同)。

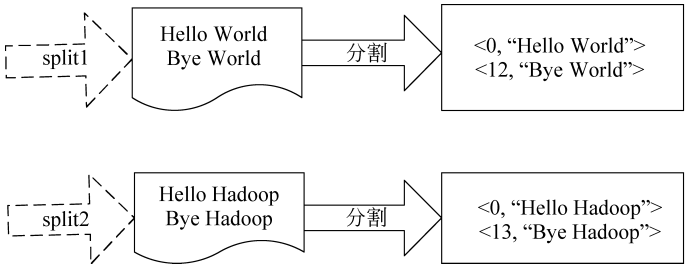

图 3-5 分割过程

(2) 将分割好的<key，value>对交给 map()方法处理，生成新的<key，value>对，如图 3-6 所示。

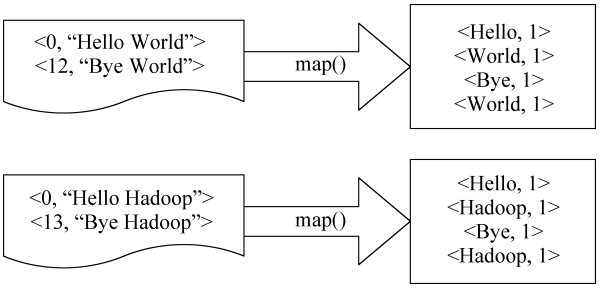

图 3-6 映射过程

(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 3-7 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 reduce()方法进行处理，得到新的<key，value>对，并作为 WordCount 的输出结果，如图 3-8 所示。

3.2.2 Eclipse 访问 Hadoop

1. 基础准备

(1) 从官网下载 hadoop-eclipse-plugin-2.7.2.jar。

打开网页 https://wiki.apache.org/hadoop/EclipsePlugIn，进入下载页面，如图 3-9 所示。

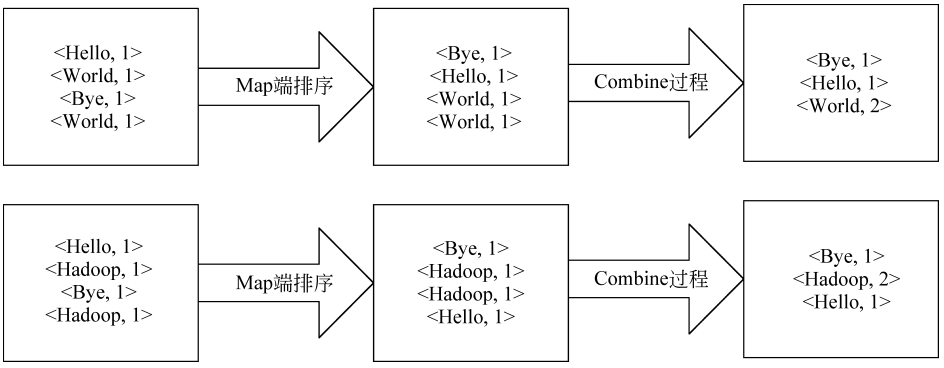

图 3-7 Map 排序与 Combine 过程

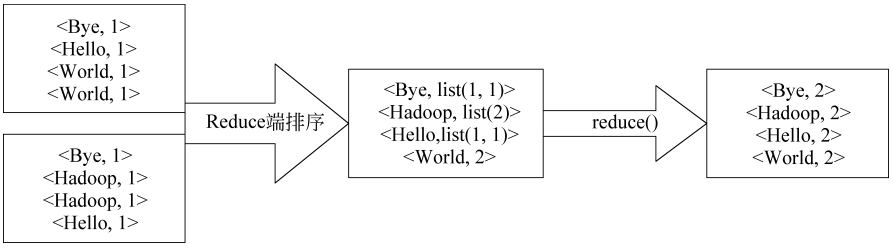

图 3-8 Reduce 排序与输出结果

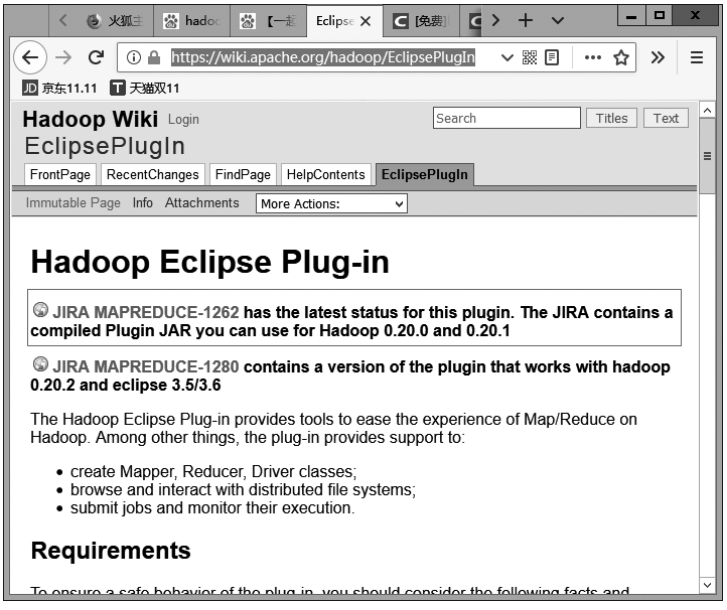

图 3-9 下载 hadoop-eclipse-plugin-2.7.2.jar 页面

(2) 如前面 3.2.1 节所述,启动 Hadoop 集群。

(3) 在名字节点安装 Eclipse,执行如下的命令。

```
#cd /usr/local
```

```
#tar -zxvf eclipse-jee-luna-SR2-linux-gtk-x86_64.tar.gz
#ln -s eclipse/eclipse /usr/bin/eclipse
#vi /usr/share/applications/eclipse.desktop
```

打开文件后，下载 3-3-2-1-eclipse-desktop 文件，作为 eclipse.desktop 文件的内容。

（4）将/usr/share/applications 下的 eclipse.desktop 文件复制到桌面。

（5）启动 Eclipse。

如图 3-10 所示，依次单击应用程序→编程→Eclipse 4.4.2，设置并确认“/home/189/workspace”空间后，单击 OK 按钮后则成功启动 Eclipse，如图 3-11 所示。

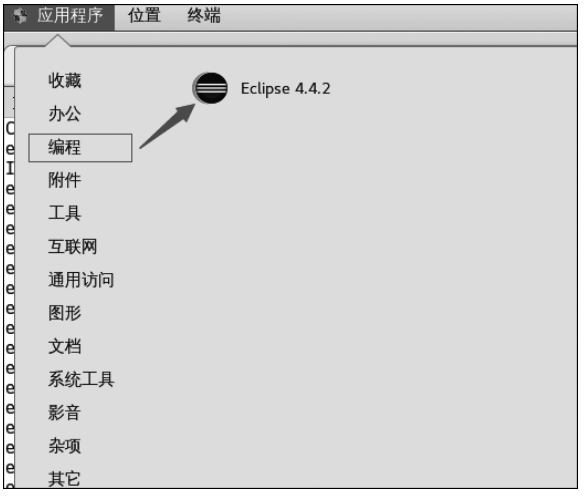

图 3-10　Eclipse 启动图标

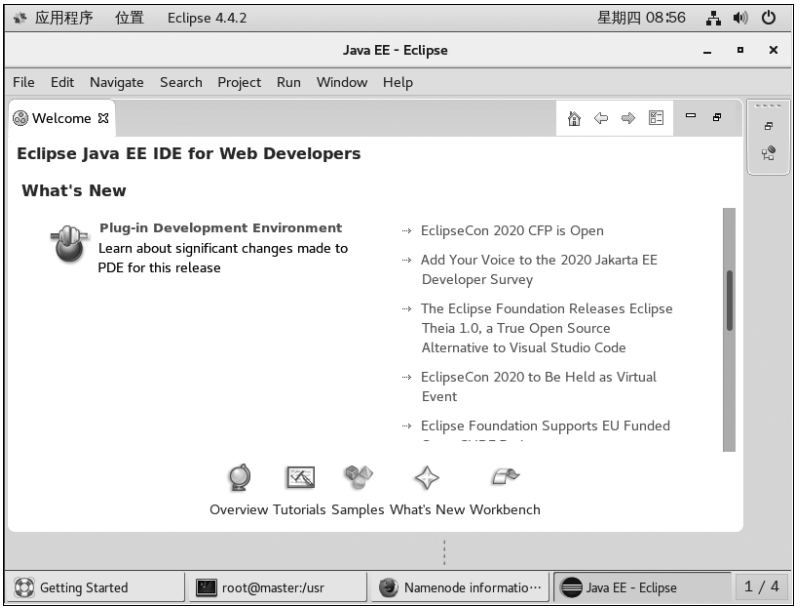

图 3-11　Eclipse 4.4.2 开发环境界面

2. 安装 Hadoop 插件

如图 3-12 所示，Windows 桌面环境下借助 WinSCP 将 hadoop-eclipse-plugin-2.7.2.jar 复制到虚拟机 Master 主机的 eclipse/dropins 目录中。

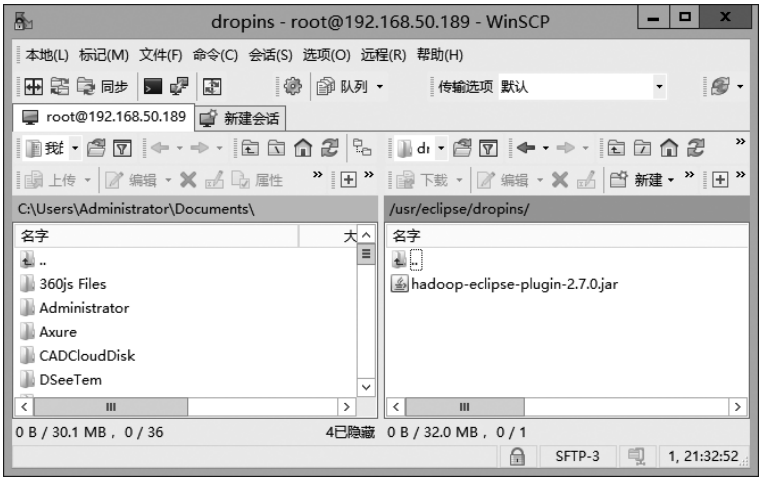

图 3-12 用 WinSCP 将 hadoop-eclipse-plugin-2.7.2.jar 复制到/usr/eclipse/dropins 目录的界面

(1) 配置 DFS Location。重启 Eclipse，依次选择 Window→show view→other→MapReduce Tools/Map/Reduce Locations，弹出如图 3-13 所示的对话框，单击图 3-13 右下角箭头所指图标，新建位置，进入如图 3-14 所示的对话框，在 Map/Reduce Master 组的 Host 和 Port 编辑框中分别输入 192.168.50.189 和 9001，在 DFS Master 组的 Port 中输入 9000，单击 Finish 按钮后出现如图 3-15 所示的 Map/Reduce Locations 记录。

图 3-13 Map/Reduce Locations 操作界面

(2) 新建 Map/Reduce 项目。在 Eclipse 中依次选择 File→New→Other...→Map/Reduce Project→Next，输入项目名 TestWordCount 后，浏览并选择 Hadoop 路径/usr/local/hadoop，单击 Finish 按钮，则项目 TestWordCount 创建成功，此时项目浏览界面如图 3-16 所示。

(3) 添加并编写 WordCount.java 源文件，其项目浏览界面如图 3-17 所示。

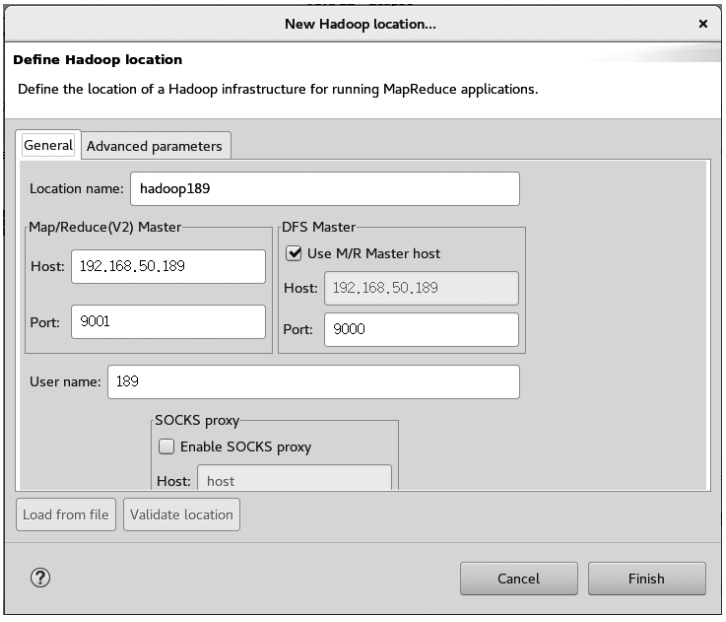

图 3-14　New Hadoop Location 对话框

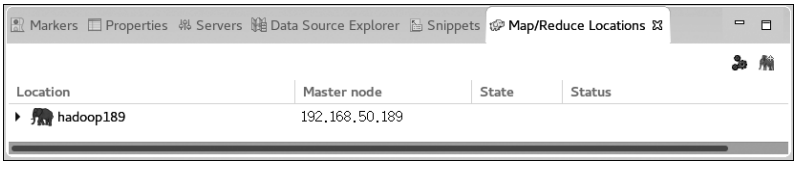

图 3-15　Map/Reduce Locations 界面

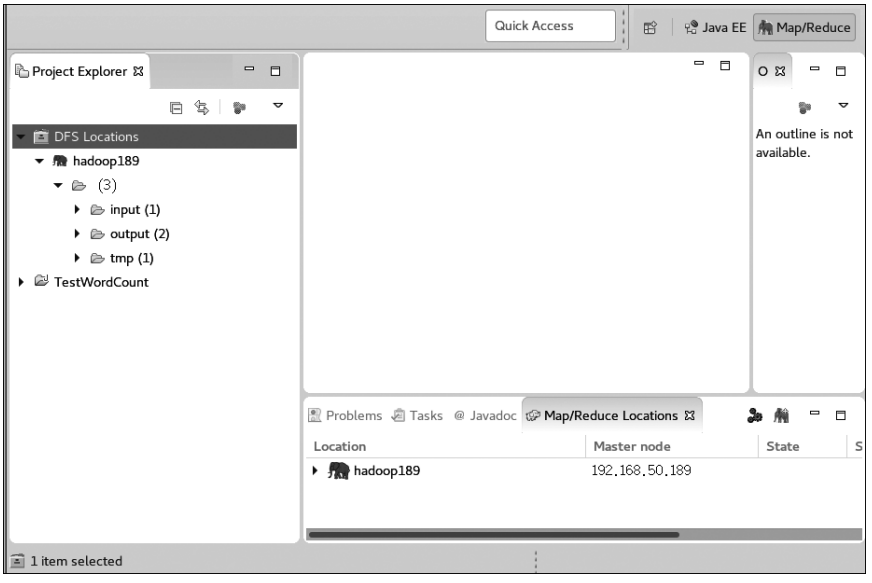

图 3-16　新建 Map/Reduce 项目成功后的项目浏览界面

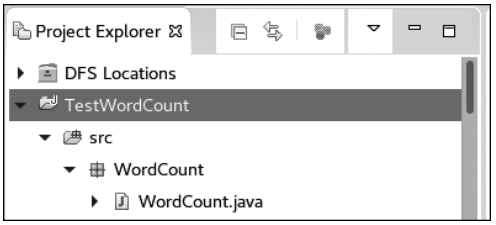

图 3-17 添加 WordCount 源文件后的项目浏览界面

下载 3-3-2-2-eclipse-WordCount 源程序文件，作为 WordCount.java 的内容。

(4) 配置运行参数。在 master 节点中新建一个 HDFS 格式的 tmp 目录，执行如下的命令。

```
#hadoop fs -mkdir /tmp
```

修改目录权限，执行如下的命令。

```
#hadoop fs -chmod -R 777 /tmp
```

(5) 在 Eclipse 本地开发环境中新建一个文件 input01，其文件内容如下。

```
hello world
hello china
hello jiangsu
hello suzhou
```

(6) 配置 Run Configurations。在 Eclipse 中依次单击 Run→Run Configurations，以配置输入/输出参数，如图 3-18 所示。

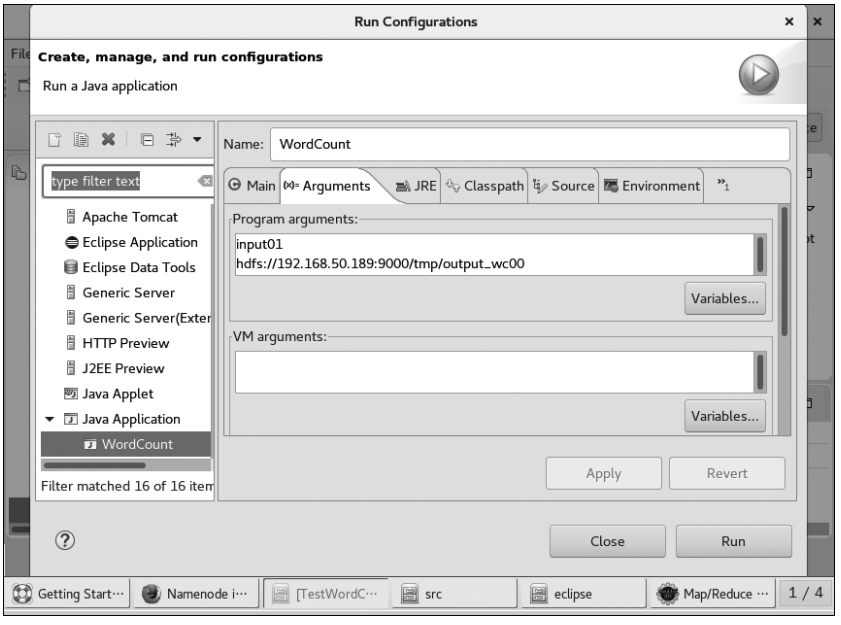

图 3-18 运行配置对话框

(7) 单击 Run 按钮,启动应用程序。

配置参数完成后,单击图 3-18 的 Run 按钮,启动应用程序,运行结果如图 3-19 所示。

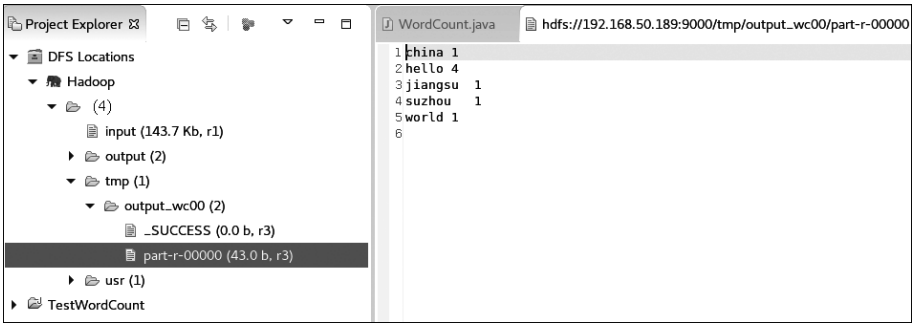

图 3-19 运行结果界面

3.2.3 自训任务和案例实践思考

1. 自训任务

在虚拟机上部署 Hadoop 3.1.1 伪分布式集群,要求如下。

(1) 3 个节点,即 1 个名字节点和 2 个数据节点。

(2) 主机名由 2 部分构成,名字节点和数据节点的主机名前半部分分别为 master 或 slave,后半部分为个人学号的后 3 位。

(3) 部署完成后打开网页 http://masterXXX:50070,XXX 为个人学号的后 3 位,观察集群状态。

(4) 运行 Hadoop 3.1.1 自带的 WordCount 单词个数统计样例包。

(5) 新建 Eclipse Map/Reduce 项目,求解 n 个数的最大值。

(6) 以 Word 文档提交完整的部署过程文档。运行结果截屏为证,体现个人主机特征。

(7) 图示给出求解最大值的具体过程。

2. 案例实践思考

根据水务云平台的解决方案,给出基于 Hadoop 生态环境的存储系统架构设计方案。

3.3 Ceph 分布式存储

3.3.1 Ceph 整体架构

1. Ceph 整体架构

Ceph 架构基于去中心化和高可靠性、高度自动化、高可扩展性的设计思路,其整体架构如图 3-20 所示。