实验3 MapReduce编程

本实验的知识地图如图3-1所示（表示重点表示难点）。

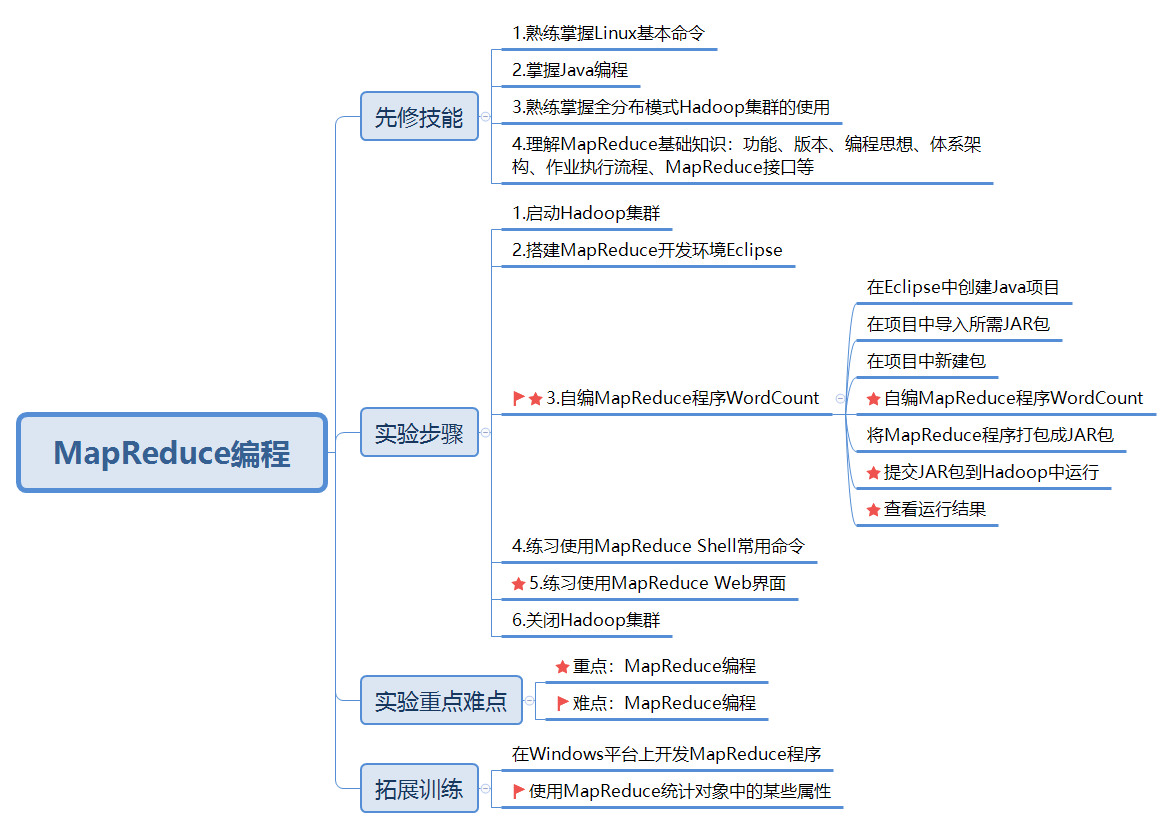


图3-1 实验3MapReduce编程知识地图

一、实验目的

1. 理解MapReduce编程思想。

2. 理解MapReduce作业执行流程。

3. 理解MR-App编写步骤，掌握使用MapReduce Java API进行MapReduce基本编程，熟练掌握如何在Hadoop集群上运行MR-App并查看运行结果。

4. 熟练掌握MapReduce Web界面的使用。

5. 掌握MapReduce Shell常用命令的使用。

二、实验环境

本实验所需的软件环境包括全分布模式Hadoop集群、Eclipse。

三、实验内容

1. 启动全分布模式Hadoop集群，守护进程包括NameNode、DataNode、SecondaryNameNode、ResourceManager、NodeManager和JobHistoryServer。

2. 在Hadoop集群主节点上搭建MapReduce开发环境Eclipse。

3. 查看Hadoop自带的MR-App单词计数源代码WordCount.java，在Eclipse项目MapReduceExample下建立新包com.xijing.mapreduce，模仿内置的WordCount示例，自己编写一个WordCount程序，最后打包成JAR形式并在Hadoop集群上运行该MR-App，查看运行结果。

4 分别在自编MapReduce程序WordCount运行过程中和运行结束后查看MapReduce Web界面。

5. 分别在自编MapReduce程序WordCount运行过程中和运行结束后练习MapReduce Shell常用命令。

6. 关闭Hadoop集群。

四、实验原理

（一）MapReduce编程思想

MapReduce是Hadoop生态中的一款分布式计算框架，它可以让不熟悉分布式计算的人员也能编写出优秀的分布式系统，因此可以让开发人员将精力专注到业务逻辑本身。

MapReduce采用“分而治之”的核心思想，可以先将一个大型任务拆分成若干个简单的子任务，然后将每个子任务交给一个独立的节点去处理。当所有节点的子任务都处理完毕后，再汇总所有子任务的处理结果，从而形成最终的结果。以“单词统计”为例，如果要统计一个拥有海量单词的词库，就可以先将整个词库拆分成若干个小词库，然后将各个小词库发送给不同的节点去计算，当所有节点将分配给自己的小词库中的单词统计完毕后，再将各个节点的统计结果进行汇总，形成最终的统计结果。以上，“拆分”任务的过程称为Map阶段，“汇总”任务的过程称为Reduce阶段，如图3-2所示。

节点3

海量词库

小词库

小词库

小词库

统计部分单词

统计全部单词

Map阶段

Reduce阶段

节点1

节点2

统计部分单词

节点4

节点5

图3-2 MapReduce执行流程

MapReduce在发展史上经过一次重大改变，旧版MapReduce（MapReduce 1.0）采用的是典型的Master/Slave结构，Master表现为JobTracker进程，而Slave表现为TaskTracker，MapReduce 1.0体系架构如图3-3所示。但是这种架构过于简单，例如Master的任务过于集中，并且存在单点故障等问题。因此，MapReduce进行了一次重要的升级，舍弃JobTracker和TaskTracker，而改用了ResourceManager进程负责处理资源，并且使用ApplicationMaster进程管理各个具体的应用，用NodeManager进程对各个节点的工作情况进行监听。升级后的MapReduce称为MapReduce 2.0，MapReduce 2.0体系架构如图3-4所示。

JobTracker

TaskTracker

Client

Client

TaskScheduler

Map Task

Map Task

Reduce Task

TaskTracker

Map Task

Map Task

Reduce Task

TaskTracker

Map Task

Map Task

Reduce Task



图3-3 MapReduce 1.0体系架构

ResourceManager

NameNode

NodeManager

ApplicationMaster

DataNode

NodeManager

ApplicationMaster

DataNode

NodeManager

Container

DataNode

Container

NodeManager

Container

DataNode

NodeManager

Container

DataNode

NodeManager

Container

DataNode

Client

Client

图3-4 MapReduce 2.0执行作业时体系架构

（二）MapReduce作业执行流程

MapReduce作业的执行流程主要包括InputFormat、Map、Shuffle、Reduce、OutputFormat五个阶段，MapReduce作业执行流程如图3-5所示。

图3-5 MapReduce作业执行流程

最终结果

<key, value>

中间结果

<key, List(value)>

中间结果

<key, value>

输入

<key, value>

加载文件

最终结果

<key, value>

中间结果

<key, List(value)>

中间结果

<key, value>

输入

<key, value>

写入文件

分布式文件系统（如HDFS）

InputFormat

Split

Split

Split

RR

RR

RR

Map

Map

Map

Shuffle

Reduce

OutputFormat

节点1

加载文件

写入文件

InputFormat

Split

Split

Split

RR

RR

RR

Map

Map

Map

Shuffle

Reduce

OutputFormat

节点2

分布式文件系统（如HDFS）

关于MapReduce作业各个执行阶段的详细说明，具体如下所示。

（1）InputFormat

InputFormat模块首先对输入数据做预处理，比如验证输入格式是否符合输入定义；然后将输入文件切分为逻辑上的多个InputSplit，InputSplit是MapReduce对文件进行处理和运算的输入单位，并没有对文件进行实际切割；由于InputSplit是逻辑切分而非物理切分，所以还需要通过RecordReader（图4-4中的RR）根据InputSplit中的信息来处理InputSplit中的具体记录，加载数据并转换为适合Map任务读取的键值对<key, valule>，输入给Map任务。

（2）Map

Map模块会根据用户自定义的映射规则，输出一系列的<key, value>作为中间结果。

（3）Shuffle

为了让Reduce可以并行处理Map的结果，需要对Map的输出进行一定的排序、分区、合并、归并等操作，得到<key, List(value)>形式的中间结果，再交给对应的Reduce进行处理，这个过程叫做Shuffle。

（4）Reduce

Reduce以一系列的<key, List(value)>中间结果作为输入，执行用户定义的逻辑，输出<key, valule>形式的结果给OutputFormat。

（5）OutputFormat

OutputFormat模块会验证输出目录是否已经存在以及输出结果类型是否符合配置文件中的配置类型，如果都满足，就输出Reduce的结果到分布式文件系统。

（三）MapReduce Web UI

MapReduce Web UI接口面向管理员。可以在页面上看到已经完成的所有MR-App执行过程中的统计信息，该页面只支持读，不支持写。MapReduce Web UI的默认地址为http://JobHistoryServerIP:19888，可以查看MapReduce的历史运行情况，如图3-6所示。

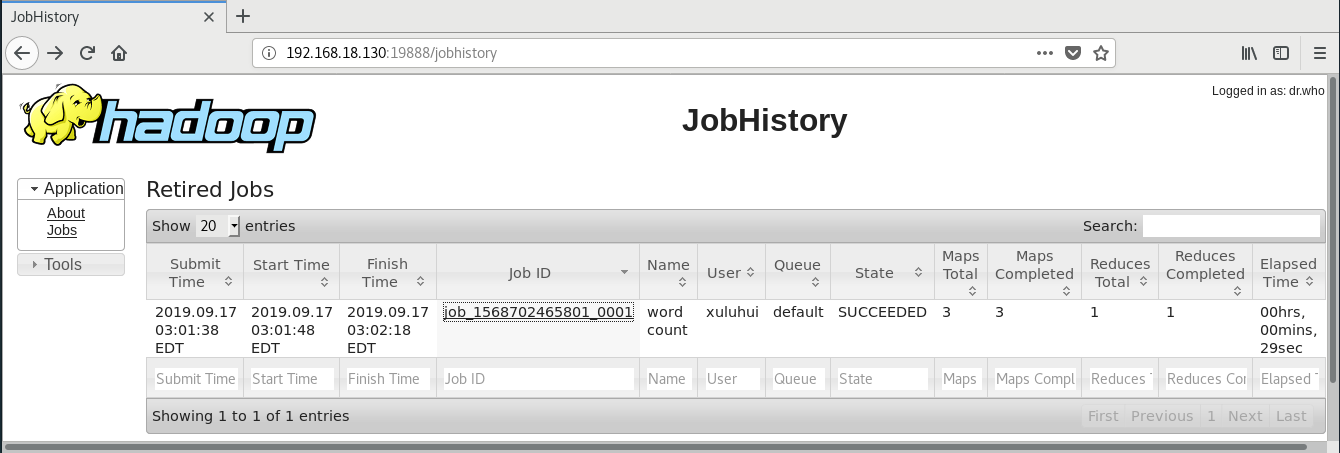


图3-6 MapReduce历史情况

（四）MapReduce Shell

MapReduce Shell接口面向MapReduce程序员。程序员通过Shell接口能够向YARN集群提交MR-App，查看正在运行的MR-App，甚至可以终止正在运行的MR-App。

MapReduce Shell命令统一入口为：mapred，语法格式如下：

mapred [--config confdir] [--loglevel loglevel] COMMAND

读者需要注意的是，若$HADOOP\_HOME/bin未加入到系统环境变量PATH中，则需要切换到Hadoop安装目录下，输入“bin/mapred”。

读者可以使用“mapred -help”查看其帮助，命令“mapred”的具体用法和参数说明如图3-7所示。

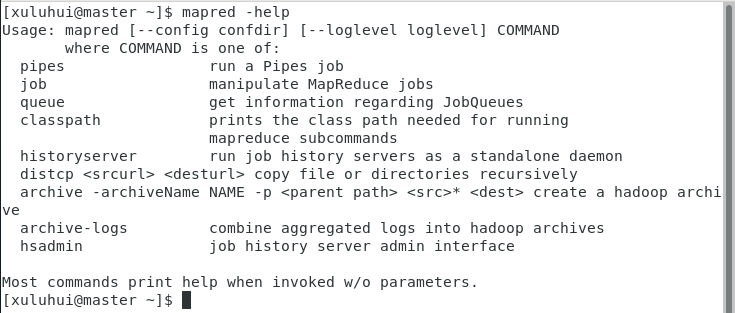


图3-7 命令“mapred”用法

MapReduce Shell命令分为用户命令和管理员命令。本章仅介绍部分命令，关于MapReduce Shell命令的完整说明，读者请参考官方网站https://hadoop.apache.org/docs/r2.9.2/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapredCommands.html。

1. 用户命令

MapReduce Shell用户命令如表3-1所示。

表3-1 MapReduce Shell用户命令

|  |  |
| --- | --- |
| 命令选项 | 功能描述 |
| archive | 创建一个Hadoop档案文件 |
| archive-logs | 将聚合日志合并到Hadoop档案文件中 |
| classpath | 打印运行MapReduce子命令所需的包路径 |
| distcp | 递归拷贝文件或目录 |
| job | 管理MapReduce作业 |
| pipes | 运行Pipes任务，此功能允许用户使用C++语言编写MapReduce程序 |
| queue | 查看Job Queue信息 |

2. 管理员命令

MapReduce Shell管理员命令如表3-2所示。

表3-2 MapReduce Shell用户命令

|  |  |
| --- | --- |
| 命令选项 | 功能描述 |
| historyserver | 启动JobHistoryServer服务 |
| hsadmin | JobHistoryServer管理命令接口 |

其中，命令“mapred historyserver”与启动MapReduce的命令“mr-jobhistory-daemon.sh start historyserver”效果相同。

读者请注意，一般不建议使用命令start-all.sh启动HDFS和YARN，而是建议使用start-dfs.sh和start-yarn.sh命令来分别启动。另外，对于一般计算机而言，在执行start-dfs.sh和start-yarn.sh命令之后最好等待一会儿再操作各种MapReduce命令，防止因为线程未加载完毕而导致的各种初始化问题。

在MapReduce程序运行一段时间后，可能由于各种故障造成HDFS的数据在各个DataNode中的分布不均匀的情况，此时也只需要通过以下shell命令即可重新分布HDFS集群上的各个DataNode。

$HADOOP\_HOME/bin/start-balancer.sh

此外，在启动时可以通过日志看到“Name node in safe mode”提示，这表示系统正在处于安全模式，此时只需要等待一会即可（通常是十几秒）。如果硬件资源较差，也可以通过执行以下命令直接退出安全模式。

$HADOOP\_HOME/bin /hadoop dfsadmin -safemode leave

（五）MapReduce Java API

MapReduce Java API接口面向Java开发工程师。程序员可以通过该接口编写MR-App用户层代码MRApplicationBusinessLogic。基于YARN编写的MR-App和基于MapReduce 1.0编写的MR-App编程步骤相同。

MR-App称为MapReduce应用程序，标准YARN-App包含3部分：MRv2框架中的MRAppMaster、MRClient，加上用户编写的MRApplicationBusinessLogic（Mapper类和Reduce类），合称为MR-App。MR-App编写步骤如下所示：

（1）编写MRApplicationBusinessLogic。自行编写。

（2）编写MRApplicationMaster。无需编写，Hadoop开发人员已编写好MRAppMaster.java。

（3）编写MRApplicationClient。无需编写，Hadoop开发人员已编写好YARNRunner.java。

其中，MRApplicationBusinessLogic编写步骤如下：

（1）确定<key,value>对。

（2）定制输入格式。

（3）Mapper阶段，其业务代码需要继承自org.apache.hadoop.mapreduce.Mapper类。

（4）Reducer阶段，其业务代码需要继承自org.apache.hadoop.mapreduce.Reducer类。

（5）定制输出格式。

编写类后，在main方法里，按下述过程依次指向各类即可：

（1）实例化配置文件类。

（2）实例化Job类。

（3）指向InputFormat类。

（4）指向Mapper类。

（5）指向Partitioner类。

（6）指向Reducer类。

（7）指向OutputFormat类。

（8）提交任务。

实际开发中，MapReduce Java API最常用的类是org.apache.hadoop.mapreduce.Mapper和org.apache.hadoop.mapreduce.Reducer。常用的MapReduce Java类如表3-3所示。

表3-3 MapReduce Java API常用类

|  |  |
| --- | --- |
| 类名 | 说明 |
| org.apache.hadoop.mapreduce.Job | MapReduce作业类 |
| org.apache.hadoop.mapreduce.Mapper | Mapper类，泛型类，带有4个参数，分别表示Map阶段输入数据的key类型、输入数据的value类型、输出数据的key类型、输出数据的value类型。其中，输入的key为Object（默认是行），输入的值为Text（Hadoop中的String类型），输出的key为Text（关键字），输出的值为IntWritable（Hadoop中的int类型） |
| org.apache.hadoop.mapreduce.Reducer | Reducer类，泛型类，带有4个参数，分别表示Reduce阶段输入数据的key类型、value类型，输出数据的key类型、value类型 |
| org.apache.hadoop.mapreduce.InputFormat | MapReduce接收输入数据的顶级类 |
| org.apache.hadoop.mapreduce.OutputFormat | MapReduce接收输出数据的顶级类 |

关于MapReduce API的完整说明，读者请参考官方网站https://hadoop.apache.org/docs/r2.9.2/api/index.html。

五、实验步骤

（一）启动Hadoop集群

在主节点上依次执行以下3条命令启动全分布模式Hadoop集群。

start-dfs.sh

start-yarn.sh

mr-jobhistory-daemon.sh start historyserver

start-dfs.sh命令会在主节点上启动NameNode和SecondaryNameNode服务，会在从节点上启动DataNode服务；start-yarn.sh命令会在主节点上启动ResourceManager服务，会在从节点上启动NodeManager服务；mr-jobhistory-daemon.sh命令会在主节点上启动JobHistoryServer服务。

（二）搭建MapReduce开发环境Eclipse

在Hadoop集群主节点上搭建MapReduce开发环境Eclipse，具体过程请读者参考实验项目2，此处不再赘述。

（三）编写并运行MapReduce程序WordCount

查看Hadoop自带的MR-App单词计数源代码WordCount.java，在Eclipse项目MapReduceExample下建立新包com.xijing.mapreduce，模仿内置的WordCount示例，自己编写一个WordCount程序，最后打包成JAR形式并在Hadoop集群上运行该MR-App，查看运行结果。具体过程如下所示。

1. 查看示例WordCount

从$HADOOP\_HOME/share/hadoop/mapreduce/sources/hadoop-mapreduce-examples-2.9.2-sources.jar中找到单词计数源代码文件WordCount.java，打开并查看源代码，完整的源代码如下所示。

package org.apache.hadoop.examples;

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;

package org.apache.hadoop.examples;

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("Usage: wordcount <in> [<in>...] <out>");

System.exit(2);

}

Job job = Job.getInstance(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);

for (int i = 0; i < otherArgs.length - 1; ++i) {

FileInputFormat.addInputPath(job, new Path(otherArgs[i]));

}

FileOutputFormat.setOutputPath(job, new Path(otherArgs[otherArgs.length - 1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

2. 在Eclipse中创建Java项目

进入/usr/local/eclipse中通过可视化桌面打开Eclipse IDE，默认的工作空间为“/home/xuluhui/eclipse-workspace”。选择菜单『File』→『New』→『Java Project』，创建Java项目“MapReduceExample”，如图3-8所示。本书中关于MapReduce编程实例均存放在此项目下。

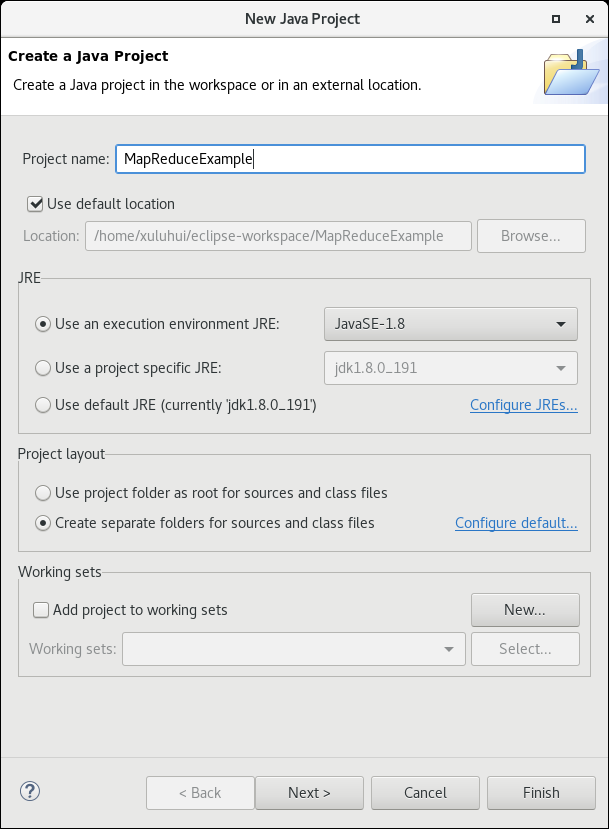


图3-8 创建Java项目“MapReduceExample”

3. 在项目中导入所需JAR包

为了编写关于MapReduce应用程序，需要向Java工程中添加MapReduce核心包hadoop-mapreduce-client-core-2.9.2.jar，该包中包含了可以访问MapReduce的Java API，位于$HADOOP\_HOME/share/hadoop/mapreduce下。另外，由于还需要对HDFS文件进行操作，所以还需要导入JAR包hadoop-common-2.9.2.jar，该包位于$HADOOP\_HOME/share/hadoop/common下。若不导入这两个JAR包，代码将会出现错误。读者可以按以下步骤添加该应用程序编写时所需的JAR包。

（1）右键单击Java项目“MapReduceExample”，从弹出的菜单中选择『Build Path』→『Configure Build Path…』，如图3-9所示。

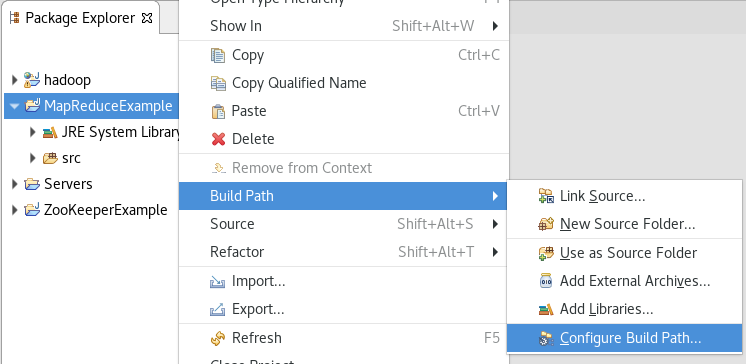


图3-9 进入“MapReduceExample”项目“Java Build Path”

（2）进入窗口【Properties for MapReduceExample】，可以看到添加JAR包的主界面，如图3-10所示。

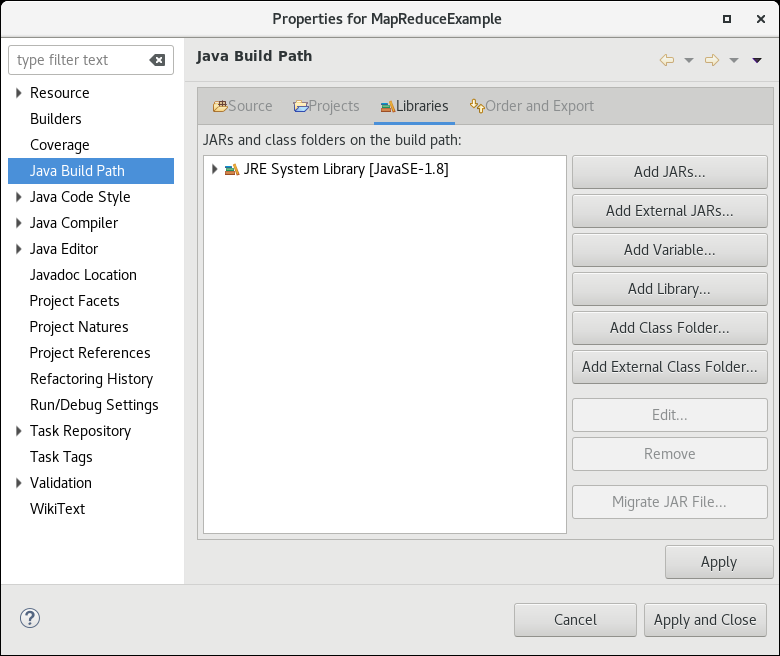


图3-10 添加JAR包主界面

（3）单击图中的按钮Add External JARS，依次添加jar文件$HADOOP\_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.9.2.jar和$HADOOP\_HOME/share/hadoop/common/hadoop-common-2.9.2.jar。其中添加JAR包hadoop-mapreduce-client-core-2.9.2.jar的过程如图3-11所示，找到此JAR包后选中并单击右上角的OK按钮，这样就成功把mapreduce-client-core-2.9.2.jar增加到了当前Java项目中。添加hadoop-common-2.9.2.jar的过程同此，不再赘述。

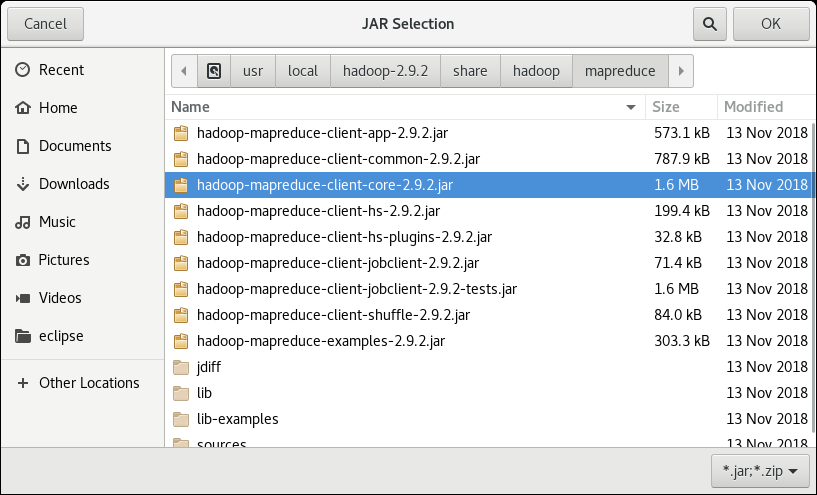


图3-11 添加hadoop-mapreduce-client-core-2.9.2.jar到Java项目中

（4）完成JAR包添加后的界面如图3-12所示，单击按钮Apply and Close。

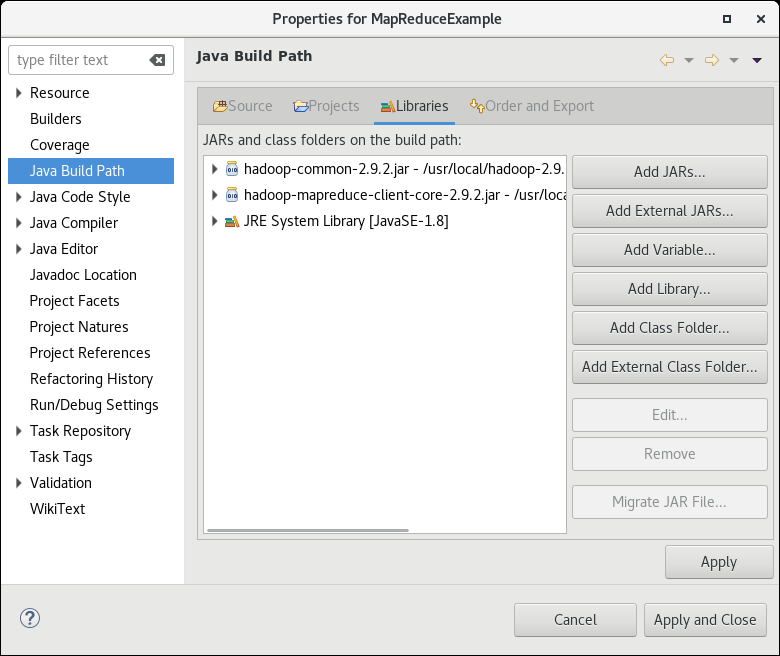


图3-12 完成JAR包添加后的界面

（5）自动返回到Eclipse界面，如图3-13所示，从图中可以看到，项目“MapReduceExample”目录树下多了“Referenced Libraries”，内部有以上步骤添加进来的两个JAR包。

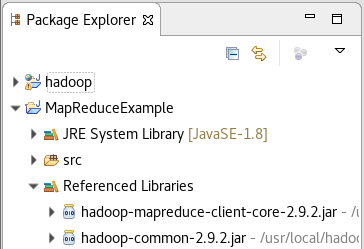


图3-13 添加JAR包后“MapReduceExample”项目目录树变化

4. 在项目中新建包

右键单击项目“MapReduceExample”，从弹出的快捷菜单中选择『New』→『Package』，创建包“com.xijing.mapreduce”，如图3-14所示。

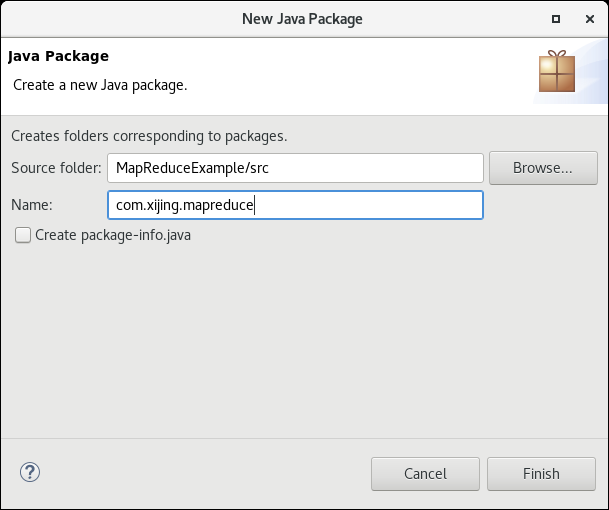


图3-14 创建包“com.xijing.mapreduce”

5. 自编MapReduce程序WordCount

下面模仿示例WordCount自编一个WordCount应用程序，借助MapReduce API，实现对输入文件单词频次的统计。

1）编写Mapper类

（1）右键单击Java项目“MapReduceExample”中目录“src”下的包“com.xijing.mapreduce”，从弹出的菜单中选择『New』→『Class』，如图3-15所示。

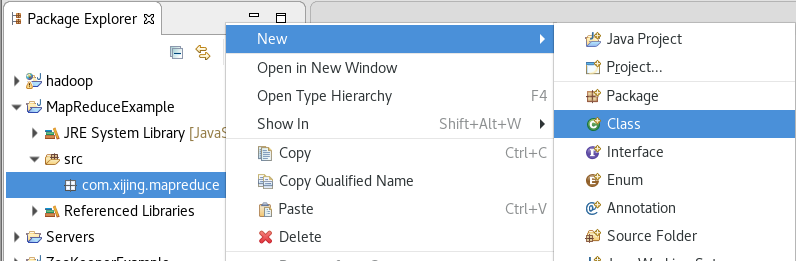


图3-15 进入“com.xijing.mapreduce”包的新建类窗口

（2）进入窗口【New Java Class】。可以看出，由于上步在包“com.xijing.mapreduce”下新建类，故此处不需要选择该类所属包；输入新建类的名字，例如“WordCountMapper”，之所以这样命名，是本类要实现Map阶段业务逻辑，建议读者命名时也要做到见名知意；读者还可以选择是否创建main函数。本实验中新建类“WordCountMapper”的具体输入和选择如图3-16所示。完成后单击Finish按钮。

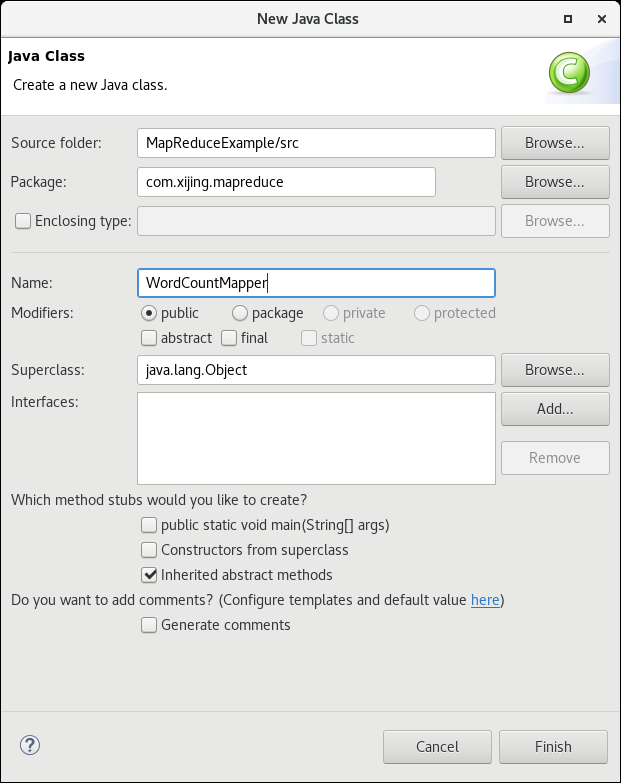


图3-16 新建类“WordCountMapper”

（3）编写Mapper类，自编WordCount程序Mapper类的源码如下所示。

package com.xijing.mapreduce;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class WordCountMapper extends Mapper<LongWritable, Text,Text, IntWritable> {

//自定义map方法

@Override

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

String line = value.toString();

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

for (String word:words){

//context.write()将数据交给下一阶段处理shuffle

context.write( new Text( word ), new IntWritable(1) );

}

}

}

2）编写Reducer类

在包“com.xijing.mapreduce”下新建类“WordCountReducer”，方法同上文“WordCountMapper”类。自编WordCount程序Reducer类的源码如下所示。

package com.xijing.mapreduce;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class WordCountReducer extends Reducer<Text, IntWritable,Text,IntWritable> {

//自定义reduce方法

@Override

protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

int sum = 0;

for(IntWritable value:values)

sum += value.get();

context.write(key, new IntWritable(sum));

}

}

3）编写入口Driver类

Mapper类和Reducer类编写完毕后，再通过Driver类将本次Job进行设置。在包“com.xijing.mapreduce”下新建类“WordCountDriver”，方法同上文“WordCountMapper”，入口Driver类的源码如下所示。

package com.xijing.mapreduce;

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.io.compress.BZip2Codec;

import org.apache.hadoop.io.compress.CompressionCodec;

import org.apache.hadoop.mapreduce.Job;

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

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

import java.io.IOException;

public class WordCountDriver {

//args：输入文件路径和输出文件路径

public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {

Configuration conf = new Configuration();

//开启map阶段的压缩

conf.setBoolean("mapreduce.map.output.compress",true);

//指定压缩类型

conf.setClass("mapreduce.map.output.compress.codec",BZip2Codec.class, CompressionCodec.class);

Job job = Job.*getInstance*(conf, "word count diy");

job.setJarByClass(WordCountDriver.class);

job.setMapperClass(WordCountMapper.class);

//使用了自定义Combine

job.setCombinerClass(WordCountReducer.class);

job.setReducerClass(WordCountReducer.class);

//指定map输出数据的类型

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

//指定reduce输出数据的类型

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

//设置输入文件路径

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

//设置输出文件路径

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

//开启reduce阶段的解压缩

FileOutputFormat.*setCompressOutput*(job,true);

//指定解压缩类型（需要与压缩类型保持一致）

FileOutputFormat.*setOutputCompressorClass*(job,BZip2Codec.class);

boolean result = job.waitForCompletion(true);

System.*exit*(result? 0 : 1);

}

}

6. 将MapReduce程序打包成JAR包

为了运行写好的MapReduce程序，需要首先将程序打包成JAR包。可以使用Maven或者Eclipse打JAR包，下面以Eclipse为例进行介绍。

（1）右键单击项目“MapReduceExample”，从弹出的快捷菜单中选择『Export...』，如图3-17所示。

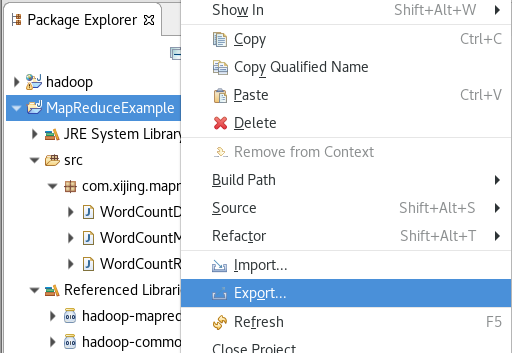


图3-17 进入“MapReduceExample”项目Export窗口

（2）进入窗口【Export】，选择Java→JAR file，单击按钮Next >，如图3-18所示。

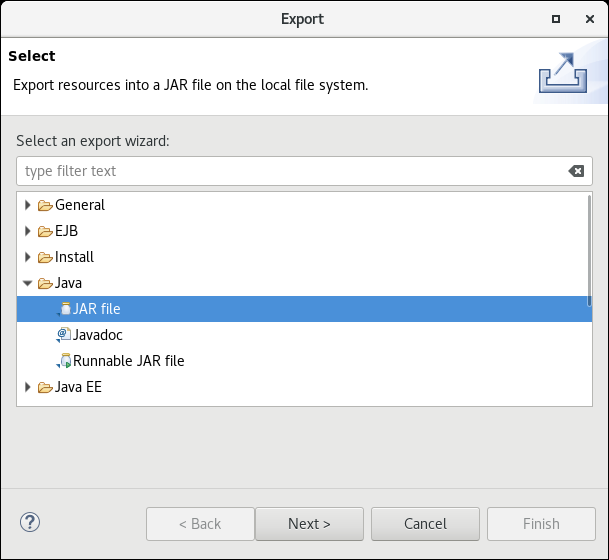


图3-18 在窗口【Export】中选择Java→JAR file

（3）进入窗口【JAR Export】，单击按钮Browse…选择JAR包的导出位置和文件名，此处编者将其保存在/home/xuluhui/eclipse-workspace/MapReduceExample下，命名为WordCountDIY.jar，效果如图3-19所示。

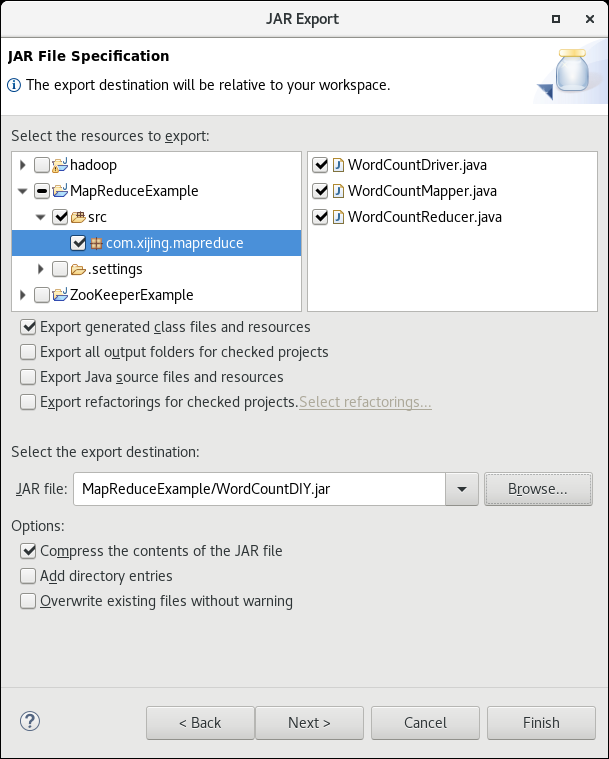


图3-19 选择打包文件及JAR包存放路径及名字

7. 提交JAR包到Hadoop中运行

与运行hadoop-mapreduce-examples-2.9.2.jar中的wordcount程序一样，只需要执行以下命令，就能在Hadoop集群中成功运行自己编写的MapReduce程序了，命令如下所示。

hadoop jar /home/xuluhui/eclipse-workspace/MapReduceExample/WordCountDIY.jar com.xijing.mapreduce.WordCountDriver /InputDataTest /OutputDataTest4

上述命令中，/InputDataTest表示输入目录，/OutputDataTest4表示输出目录。执行该命令前，假设HDFS的目录/InputDataTest下已存在待分析词频的3个文件，而输出目录/OutputDataTest4不存在，在执行过程中会自动创建。部分执行过程如图3-20所示。

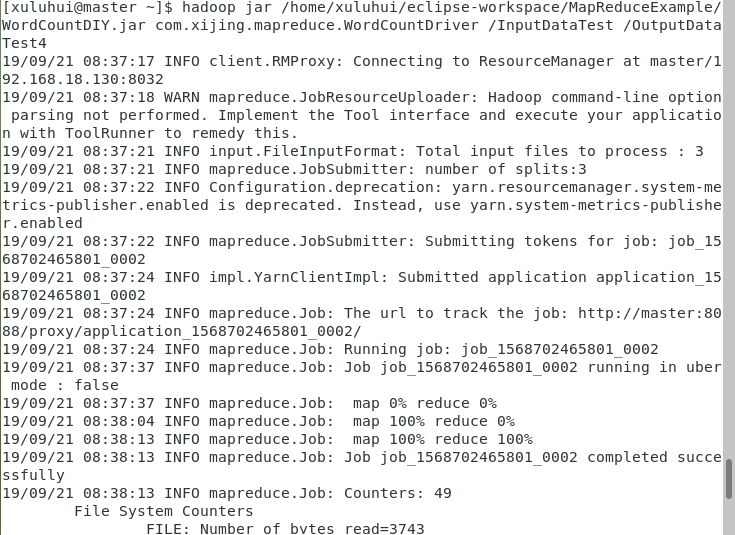


图3-20 向Hadoop集群提交并运行自编WordCount的执行过程（部分）

8. 查看运行结果

如图3-21所示，上述程序执行完毕后，会将结果输出到/OutputDataTest4目录中，可以使用命令“hdfs dfs -ls /OutputDataTest4”来查看。图3-20中/OutputDataTest4目录下有2个文件，其中/OutputDataTest4/\_SUCCESS表示Hadoop程序已执行成功，这个文件大小为0，文件名就告知了Hadoop程序的执行状态；第二个文件/OutputDataTest4/part-r-00000.bz2才是Hadoop程序的运行结果。由于输出结果进行了压缩，所以无法使用命令“hdfs dfs -cat /OutputDataTest4/part-r-00000.bz2”直接查看Hadoop程序的运行结果，查看效果如图3-21所示。

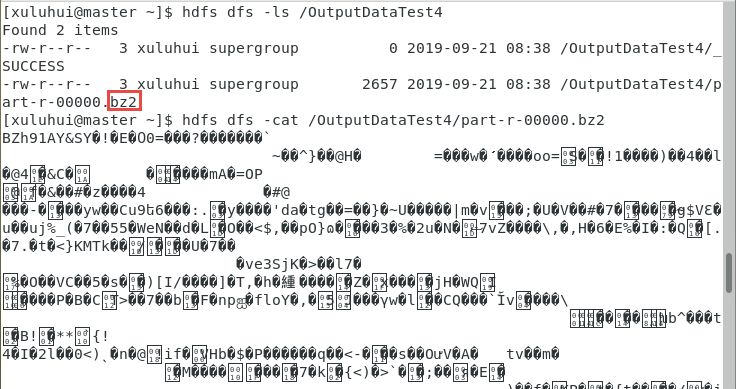


图3-21 无法使用-cat选项直接查看输出文件为.bz2的结果

若想查看输出文件扩展名为.bz2的文件，读者可以首先使用命令“hdfs dfs -get”将HDFS上的文件/OutputDataTest4/part-r-00000.bz2下载到本地操作系统，然后使用命令“bzcat”查看.bz2文件的结果，使用命令及运行结果如图3-22所示。

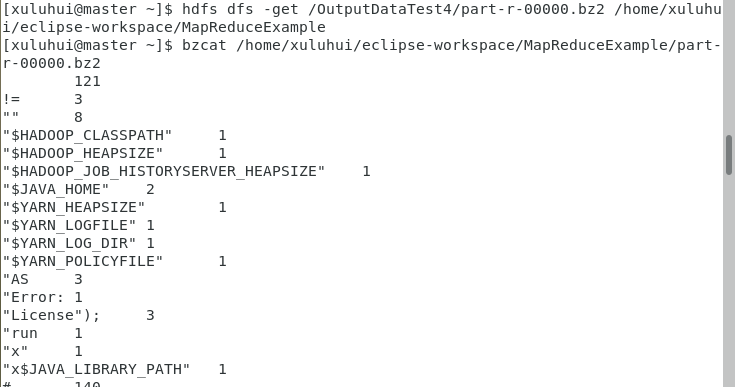


图3-22 下载.bz2文件到本地并使用bzcat查看运行结果

（四）练习使用MapReduce Shell命令

分别在自编MapReduce程序WordCount运行过程中和运行结束后练习MapReduce Shell常用命令。

例如，使用如下命令查看MapReduce作业的状态信息。

mapred job -status <job-id>

如图3-23所示，当前MapReduce作业“job\_1568702465801\_0002”正处于运行（RUNNING）状态。

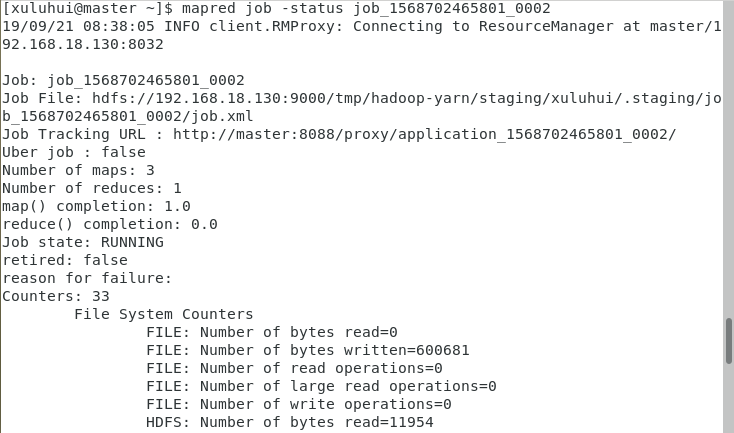


图3-23 通过命令“mapred job -status”查看该MapReduce作业状态

（五）练习使用MapReduce Web界面

分别在自编MapReduce程序WordCount运行过程中和运行结束后查看MapReduce Web界面。

例如，如图3-24所示，当前MapReduce作业“job\_1568702465801\_0002”已运行结束，其State为成功（SUCCEEDED）状态。

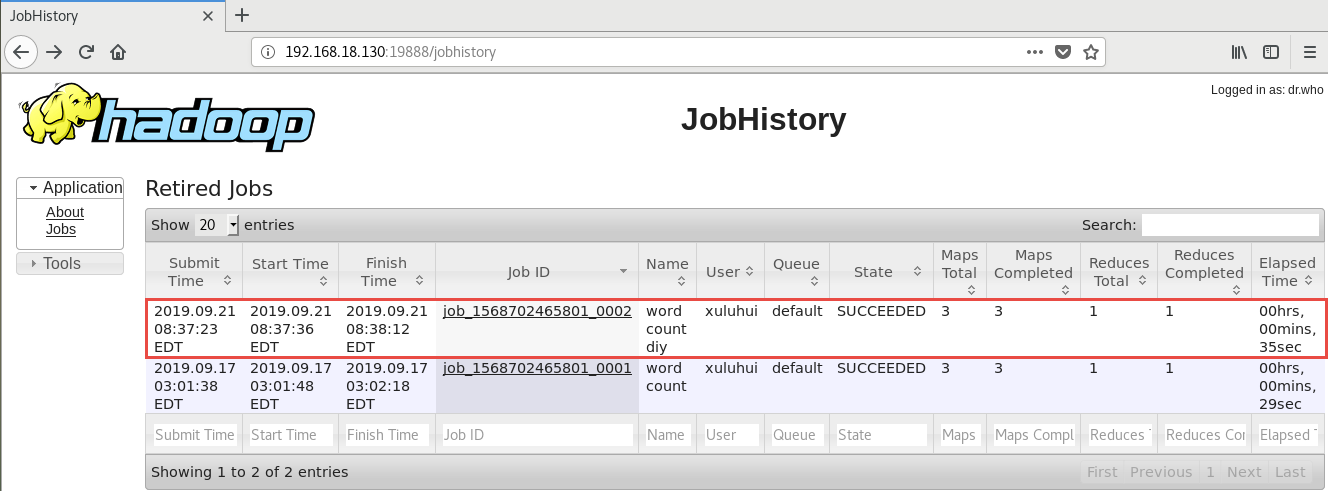


图3-24 通过MapReduce Web查看该MapReduce作业信息

（六）关闭Hadoop集群

关闭全分布模式Hadoop集群的命令与启动命令次序相反，只需在主节点master上依次执行以下3条命令即可关闭Hadoop。

mr-jobhistory-daemon.sh stop historyserver

stop-yarn.sh

stop-dfs.sh

执行mr-jobhistory-daemon.sh stop historyserver时，其\*historyserver.pid文件消失；执行stop-yarn.sh时，\*resourcemanager.pid和\*nodemanager.pid文件依次消失；stop-dfs.sh，\*namenode.pid、\*datanode.pid、\*secondarynamenode.pid文件依次消失。

六、实验报告要求

实验报告以电子版和打印版双重形式提交。

实验报告主要内容包括实验名称、实验类型、实验地点、学时、实验环境、实验原理、实验步骤、实验结果、总结与思考等。实验报告格式如表1-9所示。

七、拓展训练

（一）在Windows平台上开发MapReduce程序

在学习阶段，我们也可以直接在Windows平台上开发并运行MapReduce程序。

【案例3-1】在Windows平台上开发并运行MapReduce程序。

具体实现过程如下所示。

（1）将编译后的Windows版本的Hadoop解压到本地，并将解压后的路径设置为环境变量，如图3-25所示。

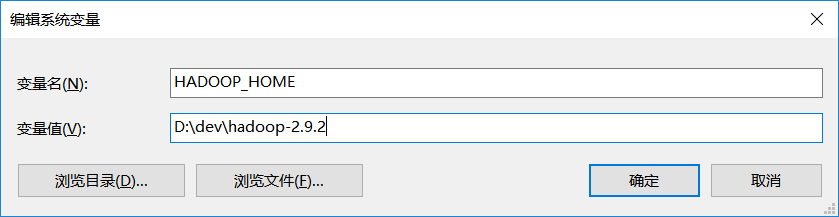


图3-25 配置HADOOP\_HOME系统变量

（2）将Hadoop中可执行命令的目录\bin和\sbin添加到环境变量PATH中，如图3-26所示。

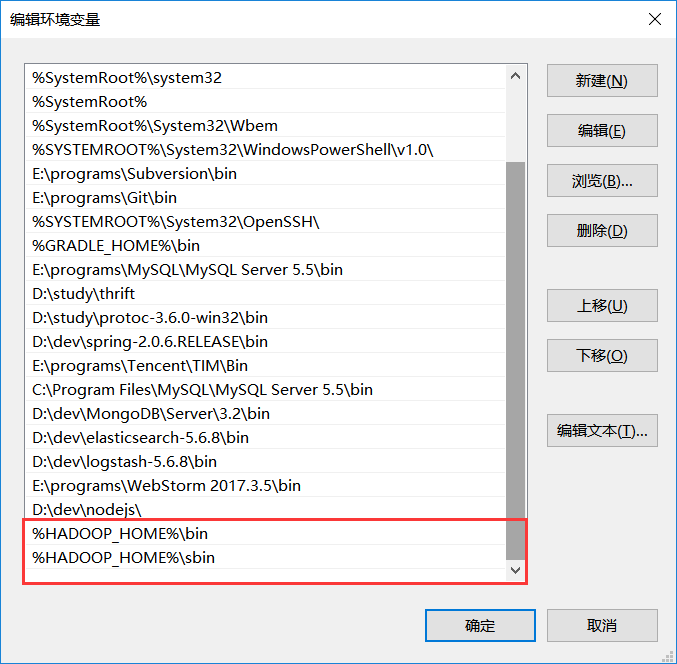


图3-26 配置Hadoop环境变量

（3）将刚刚解压后的MapReduce中的相关jar文件引入工程，或者使用Maven引入需要的JAR包，pom.xml如下所示。

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>test</groupId>

<artifactId>mp</artifactId>

<version>1.0-SNAPSHOT</version>

<!-- 统一Hadoop版本号-->

<properties>

<hadoop.version>2.9.2</hadoop.version>

</properties>

<dependencies>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-common</artifactId>

<version>${hadoop.version}</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-hdfs</artifactId>

<version>${hadoop.version}</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-mapreduce-client-core</artifactId>

<version>${hadoop.version}</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-mapreduce-client-jobclient</artifactId>

<version>${hadoop.version}</version>

</dependency>

<dependency>

<groupId>commons-cli</groupId>

<artifactId>commons-cli</artifactId>

<version>1.3.1</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-client</artifactId>

<version>${hadoop.version}</version>

</dependency>

</dependencies>

</project>

（4）为了在运行时可以在Eclipse控制台观察到MapReduce的运行时日志，可以在项目中引入Log4j，并将log4j.properties存放在项目的CLASSPATH下，log4j.properties的内容如下所示。

log4j.rootLogger=DEBUG, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n

（5）在运行时，由于权限限制，还需要通过运行参数设置访问Hadoop的用户是master。具体方法是首先在Eclipse中单击右键，从弹出快捷菜单中选择『Run As』→『Run Configurations... 』，如图3-27所示。

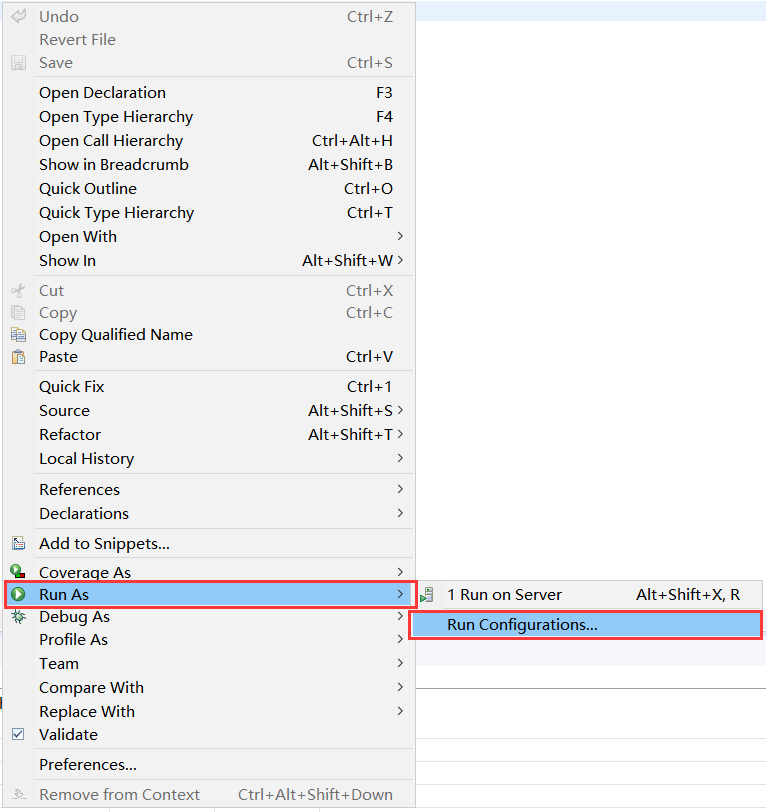


图3-27 进入配置运行参数窗口

然后，在虚拟机参数中，通过语句“-DHADOOP\_USER\_NAME=master”指定执行的用户是master，如图3-28所示。

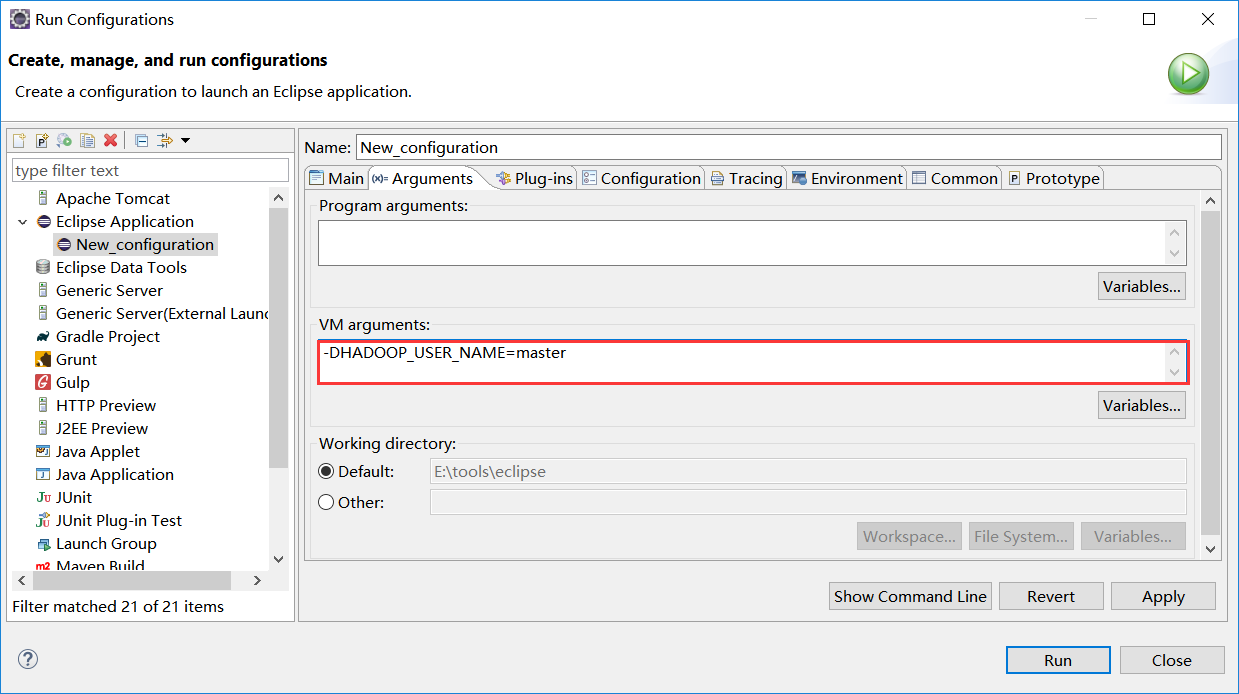


图3-28 设置VM参数

（6）此时，便可以在本地通过main()方法直接运行MapReduce程序了。

（二）MapReduce编程实践：使用MapReduce统计对象中的某些属性

之前使用MapReduce统计的是单词数量，而单词本身属于字面值，是比较容易计算的。本案例将会讲解如何使用MapReduce统计对象中的某些属性。

【案例3-2】以下是某个超市的结算记录，从左往右各字段的含义依次是会员编号、结算时间、消费金额和用户身份，请计算会员和非会员的平均消费金额。

242315 2019-10-15.18:20:10 32 会员

984518 2019-10-15.18:21:02 167 会员

226335 2019-10-15.18:21:54 233 非会员

341665 2019-10-15.18:22:11 5 非会员

273367 2019-10-15.18:23:07 361 非会员

296223 2019-10-15.18:25:12 19 会员

193363 2019-10-15.18:25:55 268 会员

671512 2019-10-15.18:26:04 76 非会员

596233 2019-10-15.18:27:42 82 非会员

323444 2019-10-15.18:28:02 219 会员

345672 2019-10-15.18:28:48 482 会员

...

本案例的实现思路是：先计算会员和非会员的总消费金额，然后除以会员或非会员的数量。具体实现过程如下所示。

（1）编写实体类

编写封装每个消费者记录的实体类，每个消费者至少包含了编号、消费金额和是否为会员等属性，源代码如下所示。

package com.xijing.mapreduce;

import org.apache.hadoop.io.Writable;

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

public class Customer implements Writable {

//会员编号

private String id;

// 消费金额

private int money;

// 0：非会员 1：会员

private int vip;

public Customer() {

}

public Customer( String id,int money, int vip) {

this.id = id;

this.money = money;

this.vip = vip;

}

public int getMoney() {

return money;

}

public void setMoney(int money) {

this.money = money;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public int getVip() {

return vip;

}

public void setVip(int vip) {

this.vip = vip;

}

//序列化

public void write(DataOutput dataOutput) throws IOException {

dataOutput.writeUTF(id);

dataOutput.writeInt(money);

dataOutput.writeInt(vip);

}

//反序列化（注意：各属性的顺序要和序列化保持一致）

public void readFields(DataInput dataInput) throws IOException {

this.id = dataInput.readUTF();

this.money = dataInput.readInt();

this.vip = dataInput.readInt() ;

}

@Override

public String toString() {

return this.id + "\t" + this.money + "\t" + this.vip;

}

}

由于本次统计的Customer对象需要在Hadoop集群中的多个节点之间传递数据，因此需要将Customer对象通过write（DataOutput dataOutput）方法进行序列化操作，并通过readFields（DataInput dataInput）进行反序列化操作。

（2）编写Mapper类

在Map阶段读取文本中的消费者记录信息，并将消费者的各个属性字段拆分读取，然后根据会员情况，将消费者的消费金额输出到MapReduce的下一个处理阶段（即Shuffle），源代码如下所示。

package com.xijing.mapreduce;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class CustomerMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

@Override

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

//将一行内容转成string

String line = value.toString();

//获取各个顾客的消费数据

String[] fields = line.split("\t");

//获取消费金额

int money = Integer.parseInt(fields[2]);

//获取会员情况

String vip = fields[3];

/\*

输出

Key：会员情况，value：消费金额

例如：

会员 32

会员 167

非会员 233

非会员 5

\*/

context.write(new Text(vip), new IntWritable(money));

}

}

（3）编写Reducer类

Map阶段的输出数据在经过shuffle阶段混洗以后，就会传递给Reduce阶段。Reduce拿到的数据形式是“会员（或非会员）,[消费金额1,消费金额2,消费金额3,...]”。因此，与WordCount类似，只需要在Reduce阶段累加会员或非会员的总消费金额就能完成本次任务，源代码如下所示。

package com.xijing.mapreduce;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class CustomerReducer extends Reducer<Text, IntWritable, Text, LongWritable> {

@Override

protected void reduce(Text key, Iterable<IntWritable>values, Context context) throws IOException, InterruptedException {

//统计会员（或非会员）的个数

int vipCount = 0 ;

//总消费金额

long sumMoney = 0;

for(IntWritable money: values){

vipCount++ ;

sumMoney += money.get() ;

}

//会员（或非会员）的平均消费金额

long avgMoney = sumMoney/vipCount ;

context.write(key, new LongWritable(avgMoney));

}

}

（4）编写MapReduce程序的驱动类

在编写MapReduce程序时，程序的驱动类基本是相同的。因此，可以仿照之前的驱动类，编写本次的MapReduce驱动类，源代码如下所示。

package com.xijing.mapreduce;

import org.apache.hadoop.conf.Configuration;

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.lib.input.FileInputFormat;

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

public class CustomerDriver {

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf);

job.setJarByClass(CustomerDriver.class);

job.setMapperClass(CustomerMapper.class);

job.setReducerClass(CustomerReducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(LongWritable.class);

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

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

boolean res = job.waitForCompletion(true);

System.exit(res?0:1);

}

}

最后，使用与WordCount程序相同的方法，将本程序打包成JAR包后就可以提交到Hadoop集群中运行了。执行结果就是会员与非会员的平均消费金额。

在教材中，我们介绍了一些优化MapReduce的具体方法，读者可以尝试使用一些优化策略对本案例进行优化。但在优化时一定要注意权衡利弊，认真思考每一个优化手段是否适合本题目的需求。例如在使用Combine组件前，需要先思考Combine的使用是否会影响程序的最终执行效果。

通过本实验大家可以发现，MapReduce将大数据的运算高度抽象为了Map和Reduce两个阶段，对于大部分的数据计算题目，我们只需要编写map()和reduce()方法，就能实现需要的计算功能。也正因为如此，MapReduce这款分布式计算框架对大数据计算领域有着里程碑的意义，认真学习好MapReduce对后续学习其他计算框架有着非常重要的意义和指导作用。

思考与练习题

1. 查阅资料，了解如何使用IntelliJ IDEA在单机环境下，运行MapReduce程序。

2. 请思考在Windows平台和Linux平台下开发MapReduce有何差异？应该从哪些方面消除这些差异。

3. 在实际工作中，一般都会在集群环境下运行MapReduce程序。但集群的部署需要多台物理机支撑，一般情况下，我们都是使用VMWare等虚拟机软件模拟多台物理机，但这种做法对物理机的性能要求较高。为此，请查阅资料，尝试在云平台上部署Hadoop集群，并成功运行MapReduce程序。

4. MapReduce是一款分布式计算框架，可以实现分布式环境下的并行计算。请思考，如果由我们自己来设计一套小型的分布式并行计算框架，需要使用到哪些技术，各个技术的作用是什么？

参考文献

[1] 董西成. Hadoop技术内幕：深入解析MapReduce架构设计与实现原理[M]. 北京:机械工业出版社,2013.

[2] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters[C]// Communications of the ACM - 50th anniversary issue: 1958 - 2008, 2008,51(1):107-113.

[3] Apache Software Foundation. Apache Hadoop 2.9.2-MapReduce Tutorial[EB/OL]. [2018-11-13]. https://hadoop.apache.org/docs/r2.9.2/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html.

[4] Apache Software Foundation. Apache Hadoop 2.9.2-MapReduce Commands Guide[EB/OL]. [2018-11-13]. https://hadoop.apache.org/docs/r2.9.2/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapredCommands.html.

[5] Apache Software Foundation. Apache Hadoop 2.9.2-Apache Hadoop Main 2.9.2 API[EB/OL]. [2018-11-13]. https://hadoop.apache.org/docs/r2.9.2/api/index.html.