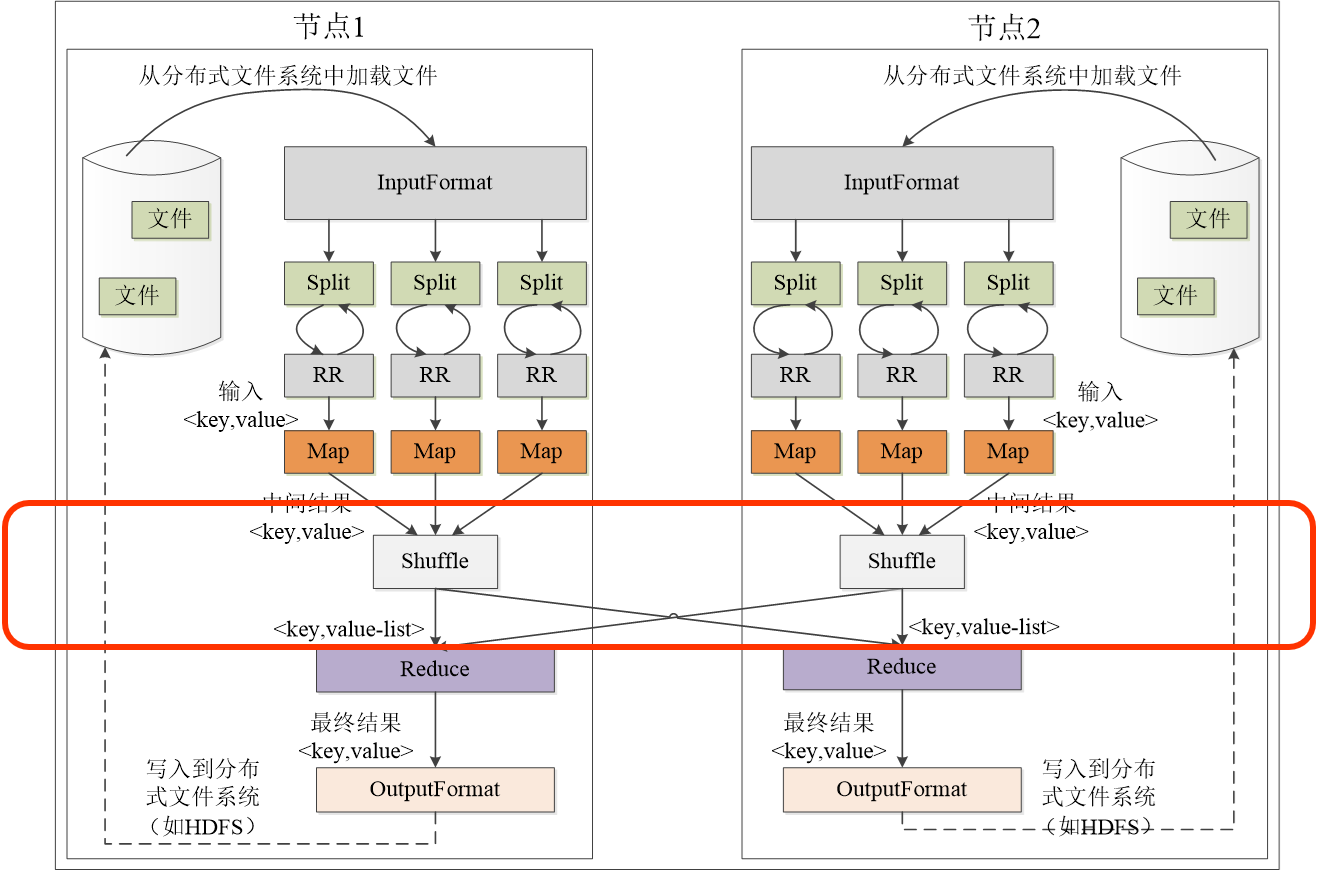
实验五　MapReduce的编程实践

# 实验目的

1. 通过实验掌握基本的MapReduce编程方法；
2. 掌握用MapReduce解决一些常见的数据处理问题，包括数据去重、数据排序和数据挖掘等。

# 实验原理

Map函数+Reduce函数+Shuffle过程



# 实验平台

1. Ubuntu 20.0.1
2. Hadoop 3.3.1（至少完成伪分布模式）
3. JDK 1.8.0\_301
4. Eclipse 2019-12(R)

# 实验步骤

**1、修改配置文件，使Hadoop集群可以运行MapReduce模型**

**1.1修改配置文件：mapred-site.xml**

cd /opt/module/hadoop/etc/hadoop

vi mapred-site.xml

添加一下内容：

#/opt/module/hadoop/是我的Hadoop的安装目录，改成你的

<property>

<name>yarn.app.mapreduce.am.env</name>

<value>HADOOP\_MAPRED\_HOME=/opt/module/hadoop/</value>

</property>

<property>

<name>mapreduce.map.env</name>

<value>HADOOP\_MAPRED\_HOME=/opt/module/hadoop/</value>

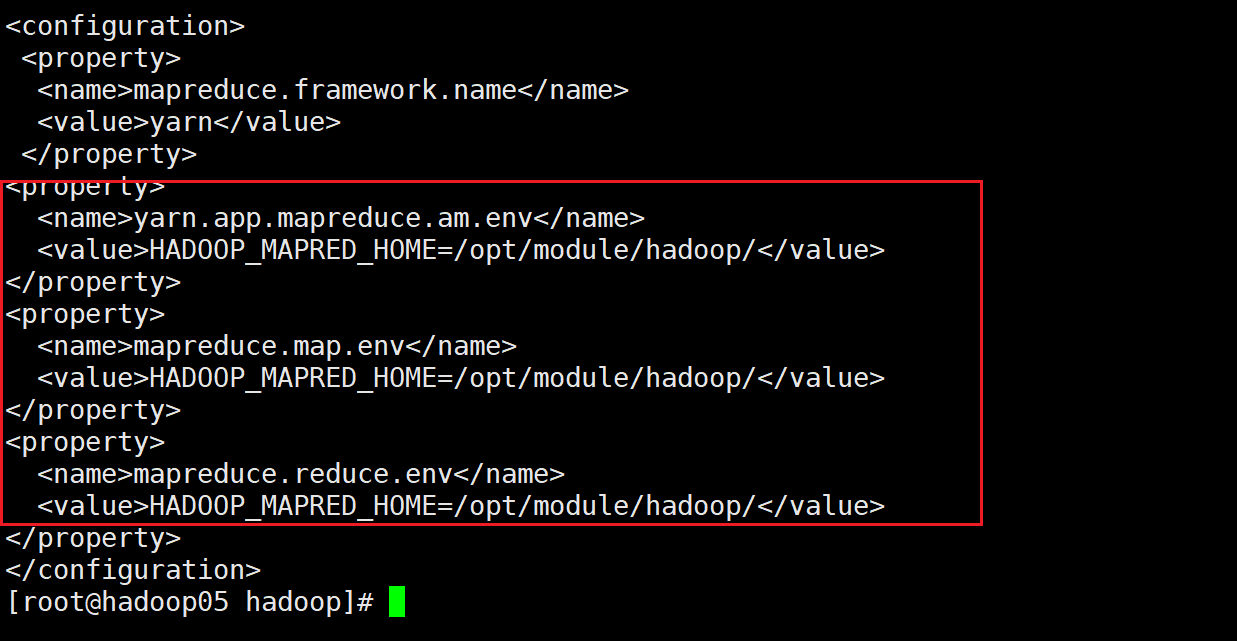
</property>

<property>

<name>mapreduce.reduce.env</name>

<value>HADOOP\_MAPRED\_HOME=/opt/module/hadoop/</value>

</property>



**1.2修改配置文件：yarn-site.xml**

cd /opt/module/hadoop/etc/hadoop

vi yarn-site.xml

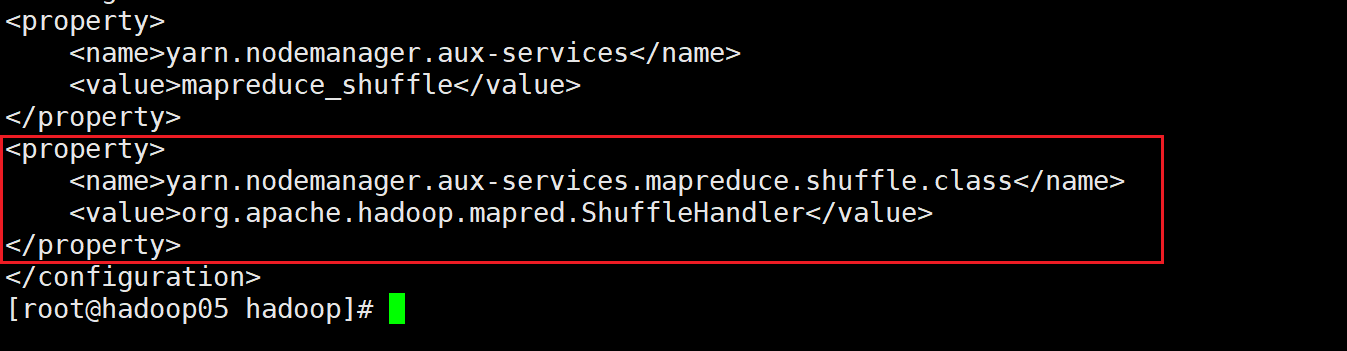
添加内容：

<property>

<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>



**1.3修改配置文件：hdfs-site.xml**

cd /opt/module/hadoop/etc/hadoop

改hdfs-site.xml，去掉权限管理，方便用过webUI操作HDFS

vi hdfs-site.xml

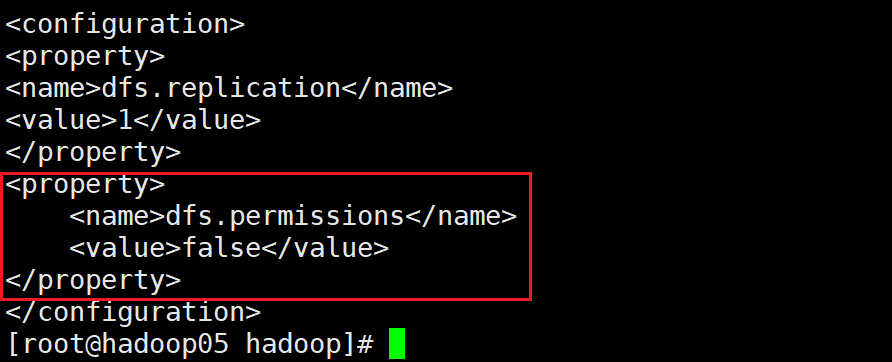
添加内容：

<property>

<name>dfs.permissions</name>

<value>false</value>

</property>



**2、重启Hadoop集群**

关闭Hadoop集群

cd /opt/module/hadoop/sbin

./stop-all.sh

启动关闭Hadoop集群

cd /opt/module/hadoop/sbin

./start-all.sh

jps检查集群状态

## Java API编程实现WordCount

创建文件wordcount.txt

内容为：

hadoop flink java

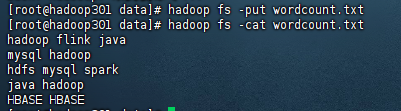
mysql hadoop

hdfs mysql spark

java hadoop

HBASE HBASE

在本地创建文件再通过hadoop fs -put方法上传文件



1.1通过java语言，按照mapreduce思想编写了一段统计文件单词量的代码，编译成 hadoop-mapreduce-examples-3.3.4.jar，进入/opt/module/hadoop/share/hadoop/mapreduce

目录下执行：

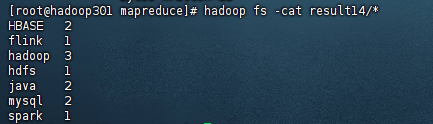
cd /opt/module/hadoop/share/hadoop/mapreduce

然后执行：

hadoop jar hadoop-mapreduce-examples-3.3.4.jar wordcount wordcount.txt result14



结果：

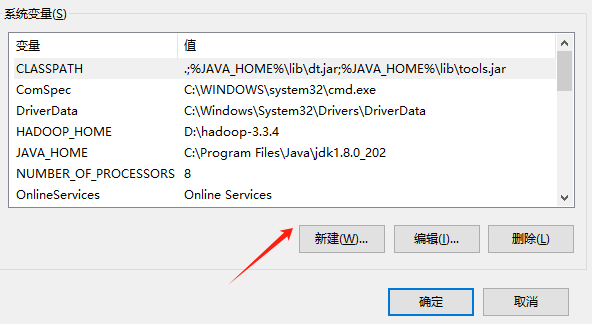


1.2IntelliJ IDEA编程

准备：

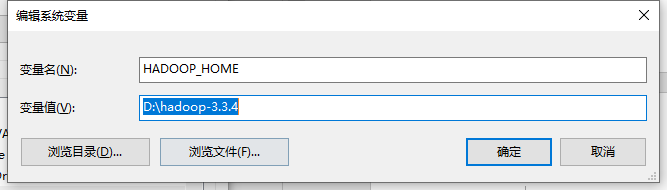


环境变量配置HADOOP\_HOME



HADOOP\_HOME

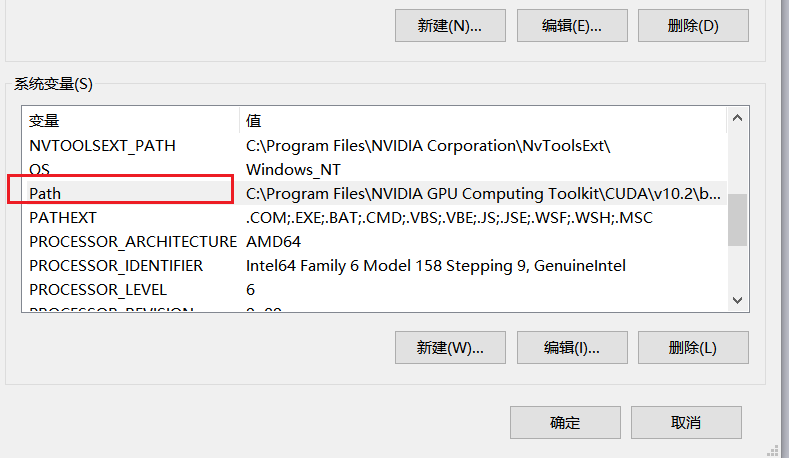
D:\hadoop-3.3.4 以实际位置修改

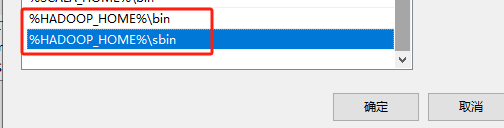


Path配置

%HADOOP\_HOME%\bin

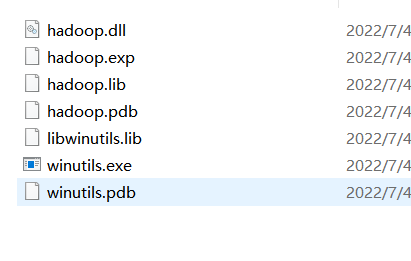
%HADOOP\_HOME%\sbin



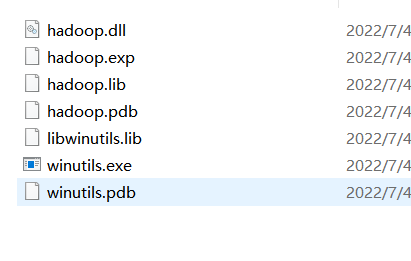


winutils文件复制

解压winutils-master.zip后，进入bin目录：



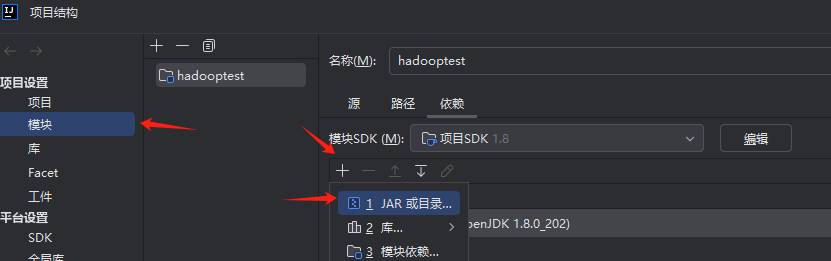
然后把里面的文件复制到目录：D:\hadoop-3.3.4\bin



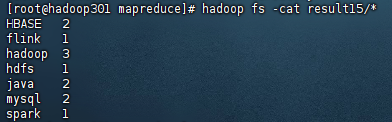
把hadoop.dll文件复制到目录：

C:\Windows\System32

导入相关jar包：lib2



import java.io.IOException;  
import java.util.Iterator;  
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 WordCount() {  
 }  
 public static void main(String[] args) throws Exception {  
 Configuration conf = new Configuration();  
 Job job = Job.*getInstance*(conf, "word count");  
 job.setJarByClass(WordCount.class);  
 job.setMapperClass(WordCount.TokenizerMapper.class);  
 job.setCombinerClass(WordCount.IntSumReducer.class);  
 job.setReducerClass(WordCount.IntSumReducer.class);  
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(IntWritable.class);  
 FileInputFormat.*addInputPath*(job, new Path("hdfs://192.168.182.100:9000/user/root/wordcount.txt"));  
 FileOutputFormat.*setOutputPath*(job, new Path("hdfs://192.168.182.100:9000/user/root/result15"));  
 System.*exit*(job.waitForCompletion(true)?0:1);  
 }  
 public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {  
 private static final IntWritable *one* = new IntWritable(1);  
 private Text word = new Text();  
 public TokenizerMapper() {  
 }  
 public void map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.Context context) throws IOException, InterruptedException {  
 StringTokenizer itr = new StringTokenizer(value.toString());  
 while(itr.hasMoreTokens()) {  
 this.word.set(itr.nextToken());  
 context.write(this.word, *one*);  
 }  
 }  
 }  
 public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {  
 private IntWritable result = new IntWritable();  
 public IntSumReducer() {  
 }  
 public void reduce(Text key, Iterable<IntWritable> values, Reducer<Text, IntWritable, Text, IntWritable>.Context context) throws IOException, InterruptedException {  
 int sum = 0;  
 IntWritable val;  
 for(Iterator i$ = values.iterator(); i$.hasNext(); sum += val.get()) {  
 val = (IntWritable)i$.next();  
 }  
 this.result.set(sum);  
 context.write(key, this.result);  
 }  
 }  
}



## Java API编程实现对输入文件的排序

现在有多个输入文件，每个文件中的每行内容均为一个整数。要求读取所有文件中的整数，进行升序排序后，输出到一个新的文件中，输出的数据格式为每行两个整数，第一个数字为第二个整数的排序位次，第二个整数为原待排列的整数。

下面是输入文件和输出文件的一个样例供参考。

输入文件1的样例如下：

33

37

12

40

输入文件2的样例如下：

4

16

39

5

输入文件3的样例如下：

1

45

25

根据输入文件1、2和3得到的输出文件如下：

1 1

2 4

3 5

4 12

5 16

6 25

7 33

8 37

9 39

10 40

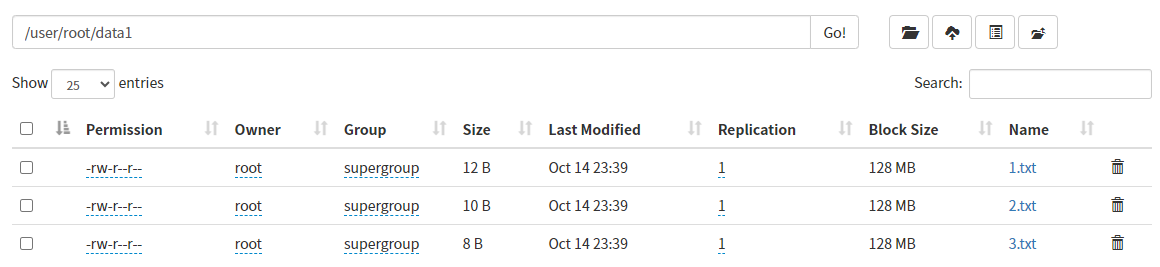
11 45

### 输入

多个输入文件，每个文件中的每行内容均为一个整数；而且测试的功能是随机数整合排序，所以可以自己创建，也可以写个小程序自动生成，或其他方法。



在hdfs上新建个data1文件夹，将多个文件上传到data1。



### 处理

import java.io.IOException;

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.Partitioner;

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 MergeSort {

// map函数读取输入中的value，将其转化成IntWritable类型，最后作为输出key

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

private static IntWritable data = new IntWritable();

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String text = value.toString();

data.set(Integer.parseInt(text)); // 如果txt的行不是整数，解析字符串会报错

context.write(data, new IntWritable(1));

}

}

// reduce函数将map输入的key复制到输出的value上，然后根据输入的value-list中元素的个数决定key的输出次数,定义一个全局变量line\_num来代表key的位次

public static class Reduce extends Reducer<IntWritable, IntWritable, IntWritable, IntWritable> {

private static IntWritable line\_num = new IntWritable(1);

public void reduce(IntWritable key, Iterable<IntWritable> values, Context context)

throws IOException, InterruptedException {

for (IntWritable val : values) {

context.write(line\_num, key);

line\_num = new IntWritable(line\_num.get() + 1);

}

}

}

// 自定义Partition函数，此函数根据输入数据的最大值和MapReduce框架中Partition的数量获取将输入数据按照大小分块的边界，然后根据输入数值和边界的关系返回对应的Partiton

// ID

public static class Partition extends Partitioner<IntWritable, IntWritable> {

public int getPartition(IntWritable key, IntWritable value, int num\_Partition) {

int Maxnumber = 65223;// int型的最大数值

int bound = Maxnumber / num\_Partition + 1;

int keynumber = key.get();

for (int i = 0; i < num\_Partition; i++) {

if (keynumber < bound \* (i + 1) && keynumber >= bound \* i) {

return i;

}

}

return -1;

}

}

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><out>");

// System.exit(2);

// }

Job job = Job.getInstance(conf, "Merge and sort");

job.setJarByClass(MergeSort.class);

job.setMapperClass(Map.class);

job.setReducerClass(Reduce.class);

job.setPartitionerClass(Partition.class);

job.setOutputKeyClass(IntWritable.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path("hdfs://192.168.182.100:9000/user/root/data1"));

FileOutputFormat.setOutputPath(job, new Path("hdfs://192.168.182.100:9000/user/root/data1\_result"));

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

}

}

### 输出

hadoop fs -cat data1\_result/\*

