数据二教学笔记 1224

目录

01 数据二 - MapReduce 概念及案例运行

1. MapReduce 学习

Hadoop MapReduce 是一个软件框架，用于轻松编写应用程序，以可靠，容错的方式在大型集群（数千个节点）的商用硬件上并行处理大量数据（多 TB 数据集）。

MapReduce 作业通常将输入数据集拆分为独立的块，这些块由 map 任务以完全并行的方式处理。框架对地图的输出进行排序，然后输入到 reduce 任务。通常，作业的输入和输出都存储在文件系统中。该框架负责调度任务，监视它们并重新执行失败的任务。

通常，计算节点和存储节点是相同的，即 MapReduce 框架和 Hadoop 分布式文件系统（请参阅 HDFS 体系结构指南）在同一组节点上运行。此配置允许框架有效地在已存在数据的节点上调度任务，从而在集群中产生非常高的聚合带宽。

MapReduce 框架由单个主 ResourceManager，每个集群节点一个从 NodeManager 和每个应用程序的 MRAppMaster 组成

MapReduce 作业的输入和输出类型：

（输入）<k1 ，v1> - > map - > <k2，v2> - > combine - > <k2，v2> - > reduce - > <k3，v3>（输出）

示例：WordCount v1.0

在我们深入研究细节之前，让我们通过一个示例 MapReduce 应用程序来了解它们的工作方式。

WordCount 是一个简单的应用程序，它计算给定输入集中每个单词的出现次数。

原课堂笔记

(1) MapReducer 分两个阶段 1.map 2.reducer

(2) map-->reducer 会触发按照 map 输出的 key 合并 value

```
context.write(new Text(str),map中第一个参数是key));
```

①

(3) 常用的数据类型(在 mapreducer 中涉及到多机器夸网络、磁盘交互)

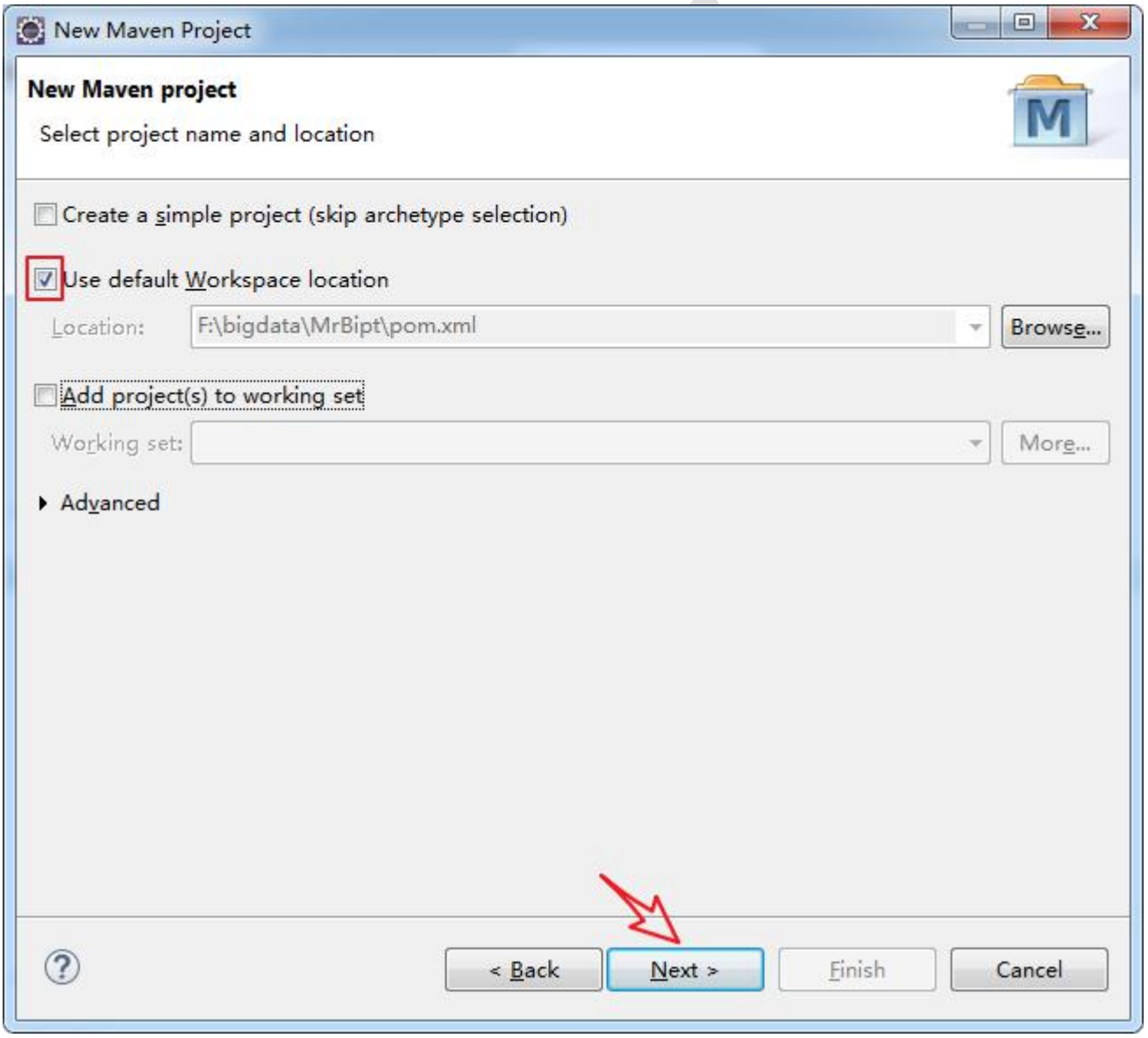
```
/*
 * 常用数据类型
 * LongWritable ---> Long(long)
 * Text------------> String
 * IntWritable ----> int(Integer)
 * NullWritable----> null
 * ShortWritable---> short(Short)
 * DoubleWritable--> double(Double)
```

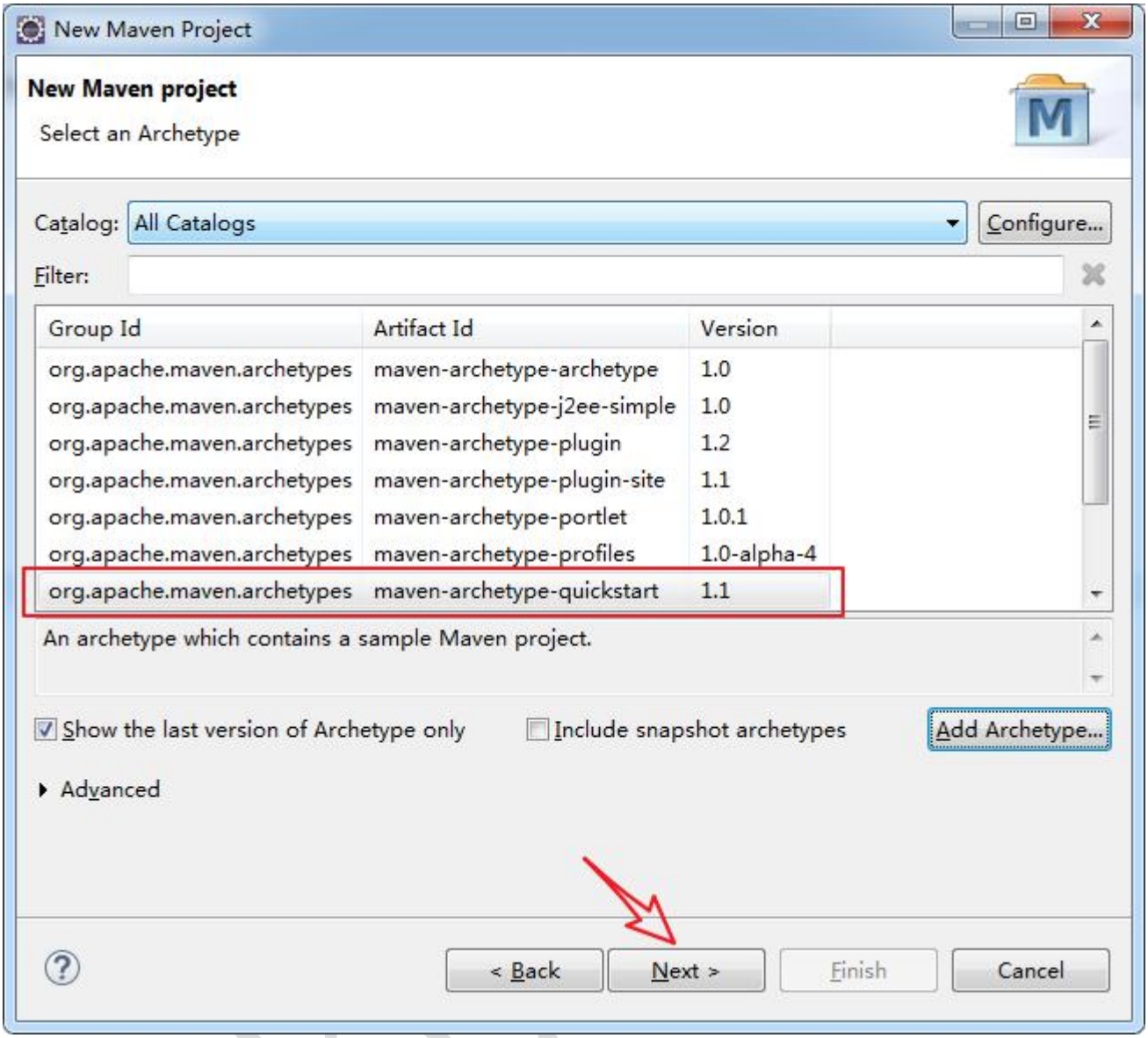
```
* FloatWritable---> float(Float)
* ByteWritable----> byte(Byte)
* BooleanWritable-> boolean(Boolean)
*/
```

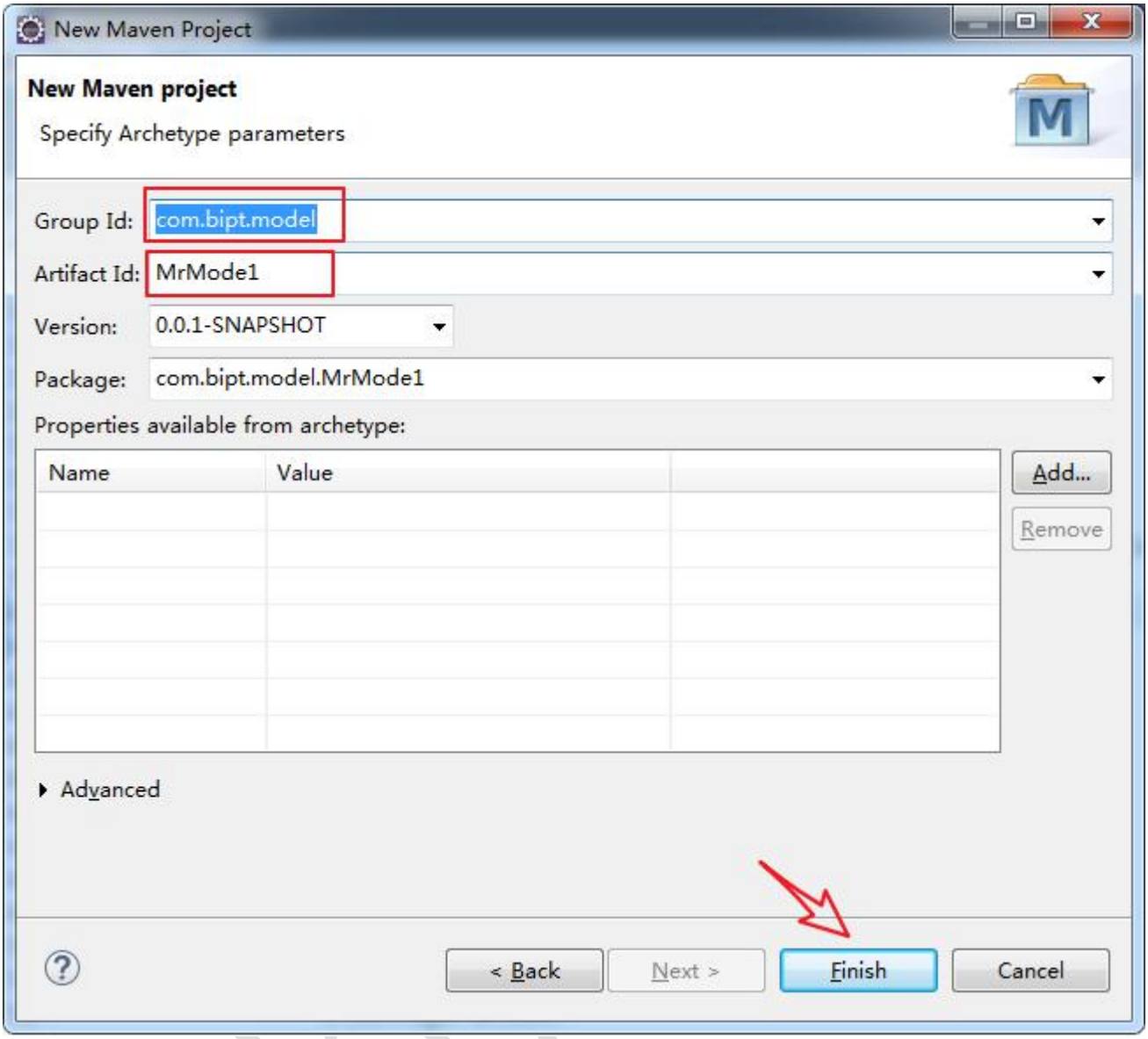
(4) reducer 阶段的输入 key,value 类型必须和 map 阶段输出的 key,value 类型一致

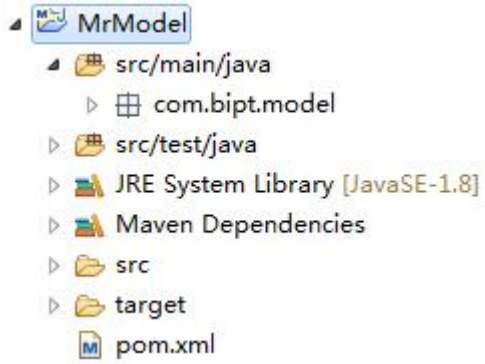
2. 创建 Maven 项目

File --> New --> Other --> Maven --> Maven Project




New Maven Project
New Maven project
Select an Archetype
Catalog: All Catalogs
Configure...
Filter:
Group Id
Artifact Id
Version
org.apache.maven.archetypes
maven-archetype-archetype
1.0
org.apache.maven.archetypes
maven-archetype-j2ee-simple
1.0
org.apache.maven.archetypes
maven-archetype-plugin
1.2
org.apache.maven.archetypes
maven-archetype-plugin-site
1.1
org.apache.maven.archetypes
maven-archetype-portlet
1.0.1
org.apache.maven.archetypes
maven-archetype-profiles
1.0-alpha-4
org.apache.maven.archetypes
maven-archetype-quickstart
1.1
An archetype which contains a sample Maven project.
Show the last version of Archetype only
Include snapshot archetypes
Add Archetype...
Advanced
< Back
Next >
Finish
Cancel


New Maven Project
New Maven project
Specify Archetype parameters
Group Id: com.bipt.model
Artifact Id: MrMode1
Version: 0.0.1-SNAPSHOT
Package: com.bipt.model.MrMode1
Properties available from archetype:
Name
Value
Add...
Remove
Advanced
< Back
Next >
Finish
Cancel


MrModel
src/main/java
com.bipt.model
src/test/java
JRE System Library [JavaSE-1.8]
Maven Dependencies
src
target
pom.xml

(1) 配置 pom.xml 文件

① 3.1 指定 maven 保存路径

Window --> Preferences --> Maven --> User Settings --> User Settings(open file): --> Browe...
选择配置保存路径的 xml 文件（settings.xml）

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
<!--           对应路径修改成，自己电脑本地希望保持 maven 同步文件的已存在路径 -->
<localRepository>F:\mavenStore\hg\repository</localRepository>
</settings>
```

② 3.2 修改 pom.xml 配置文件（Hadoop MapReduce 开发）

Hadoop MapReduce 的基本开发配置如下

```
<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>
    <!-- 项目命名空间基本信息 -->

    <!-- 程序创建生成基本信息、每个项目各不相同，不可能完全拷贝 -->
    <!-- project group id & artifact id -->
    <groupId>com.bipt.model</groupId>
    <artifactId>MrModel</artifactId>
    <!-- maven model version -->
    <version>0.0.1-SNAPSHOT</version>
    <!-- packing type 打包类型 -->
    <packaging>jar</packaging>

    <name>MrModel</name>
    <url>http://maven.apache.org</url>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
```

```
    <!-- 依赖关系。实际上 pom 之间存在好三种关系：继承、依赖、聚合 -->
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>

        <!-- hadoop maven about reduce -->
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-common</artifactId>
            <version>2.7.1</version>
        </dependency>
        <!--          mr core -->
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-mapreduce-client-core</artifactId>
            <version>2.7.1</version>
        </dependency>
        <!--       mr job client -->
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-mapreduce-client-jobclient</artifactId>
            <version>2.7.1</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.3.2</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

(2) 实现 wordcount 功能

① Mapper 实现

```
package com.????;
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
/**
 *  Mapper 输入输出信息
 *  输入的 key    :    LongWritable 类型的偏移量
 *  出入的 value :  Text 文本类型
 *  输出的 key    :  Text 文本类型，可以根据实际需求变更；对应 Reduce 的输入 key
 *  输出的 value :  IntWritable 返回 Integer 的类型，可以根据实际需求变更；对应 Reduce 的输入 value
 */
public class MyMapReduce_01Map extends Mapper<LongWritable, Text, Text, IntWritable>
{
    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        super.map(key, value, context);
        String[] splits = value.toString().split(" ");//文本根据空格切分
        for (String str : splits) {
            //将切分好的文本存入 key，对应每个 value 都是 1 <str : 1>
            context.write(new Text(str), new IntWritable(1));
        }
    }
```

② Reducer 实现

```
package com.????;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
/**
 *  Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT>
 *  Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT>
 *  Mapper 输入输出信息
 *  输入的 key   :   LongWritable 类型的偏移量
 *  出入的 value :  Text 文本类型
 *  输出的 key   :  Text 文本类型，可以根据实际需求变更；对应 Reduce 的输入 key
 *  输出的 value :  IntWritable 返回 Integer 的类型，可以根据实际需求变更；对应 Reduce 的输入
value
 *  每一个 Mapper 会调用一次 Reduce
 */
public class MyMapReduce_01Reducer extends Reducer<Text, IntWritable, Text, IntWritable>
{
    @Override
    protected void reduce(Text key, Iterable<IntWritable> values,
            Context context) throws IOException, InterruptedException {
        int count = 0;
        //将每一个 Mapper 中的文本对应的，标识 1 进行累加，统计文本出现的总次数；
        for(IntWritable v : values){
            count += v.get();
        }
        //输出 key 文本内容；value 统计出现的次数；
        context.write(key, new IntWritable(count));
    }
}
```

③ 启动类

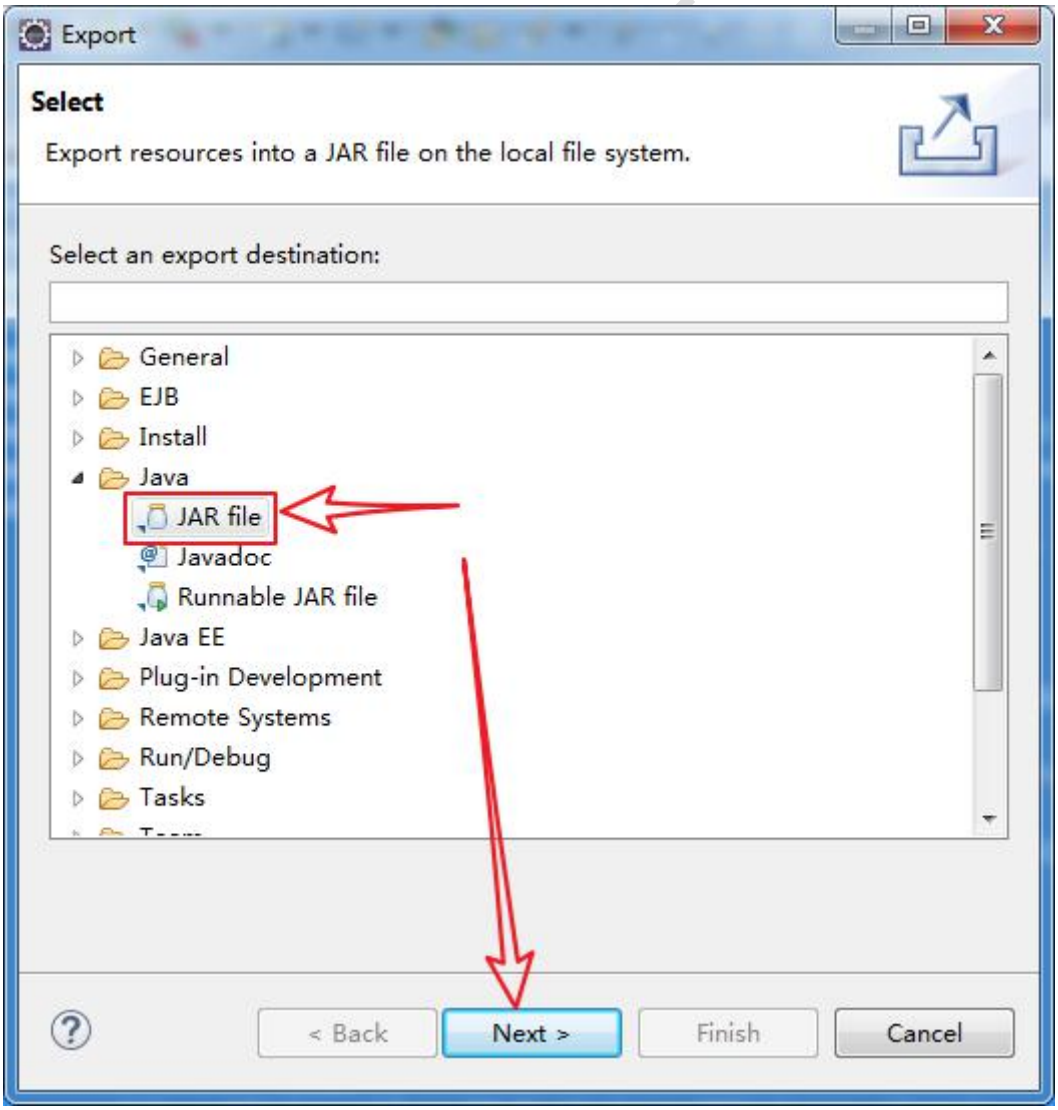
```
package com.????;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class MyMapReduce_01 {
    public static void main(String[] args) throws Exception {
        if (args.length < 2) {// 判断参数是否正常
            System.out.println("usage <in> ... <out>");
            System.err.println("必须有输入和输出路径");
            System.exit(2);// 系统退出标识：只有 0 标识正常，非 0 标识非正常系统退出；
        }
        Configuration conf = new Configuration(); // 注意导包 hadoop
        Job job = Job.getInstance(conf);// 创建计算工作
        job.setJarByClass(MyMapReduce_01.class);// 需要指定工作主类
        job.setJobName("mywordcount");//8088 端口结算结果显示的名字
        //Mapper 的实现类
        job.setMapperClass(MyMapReduce_01Map.class);
        job.setMapOutputKeyClass(Text.class);// 可以省略
        job.setMapOutputValueClass(IntWritable.class);//
        //Reducer 实现类
        job.setReducerClass(MyMapReduce_01Reducer.class);
        job.setOutputKeyClass(Text.class);//
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));//指定输入
        FileOutputFormat.setOutputPath(job, new Path(args[1]));//输出
        System.out.println(job.waitForCompletion(true) ? 0 : 1);//执行；成功 0；失败 1；
    }
}
```

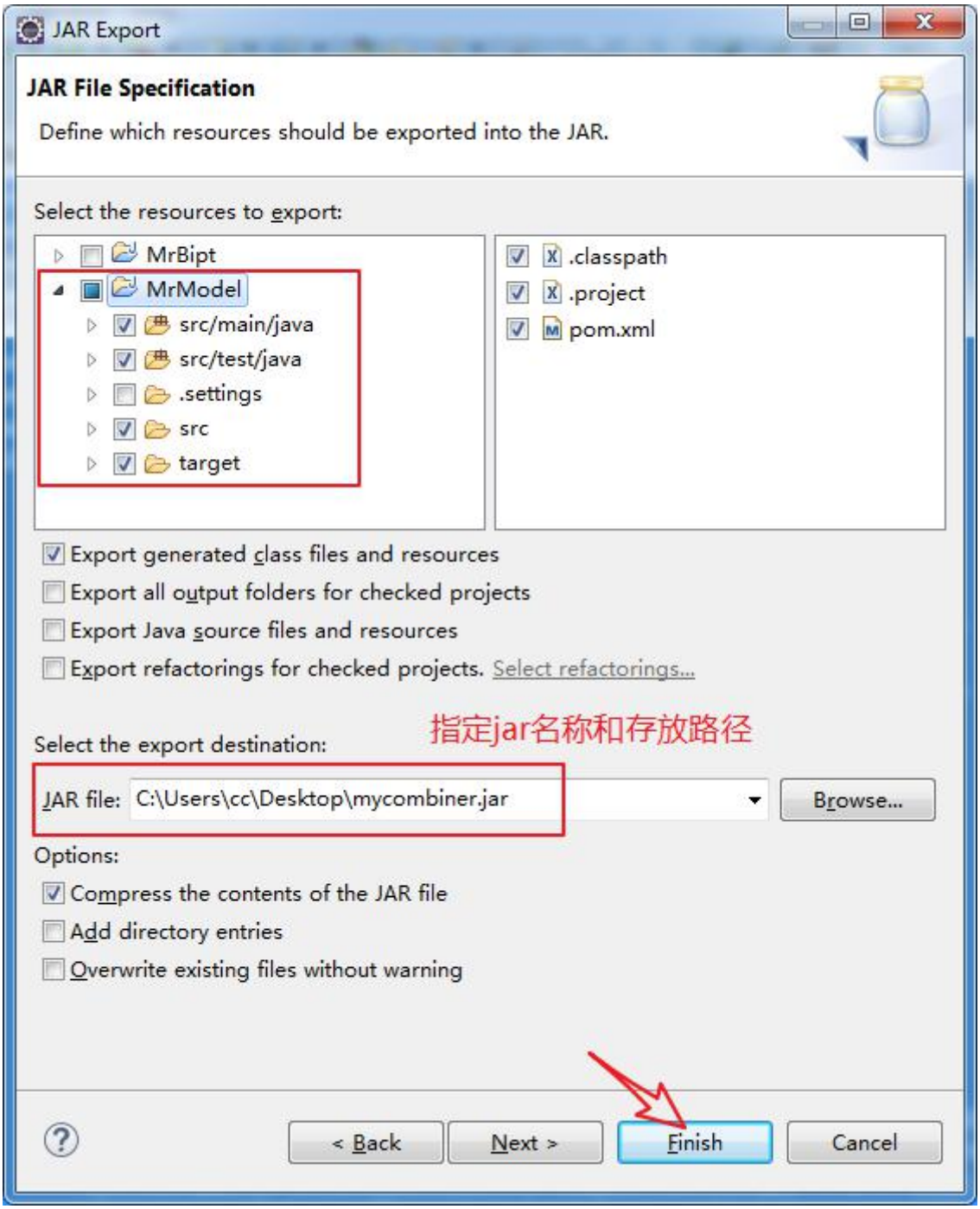
3. 导出项目 jar 运行步骤

(1) 导出 jar

Export... --> Jar file--> export Browse... --> -->


MrMode
src/m
cc
src/te
JRE S
Mave
src
targe
pom.
zkJavaAp
Show In Alt+Shift+W
项目上右键
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Refactor Alt+Shift+T
Import...
Export...


Export
Select
Export resources into a JAR file on the local file system.
Select an export destination:
General
EJB
Install
Java
JAR file
Javadoc
Runnable JAR file
Java EE
Plug-in Development
Remote Systems
Run/Debug
Tasks
< Back
Next >
Finish
Cancel



(2) 执行 jar 进行计算

① mapreduce 程序的命令

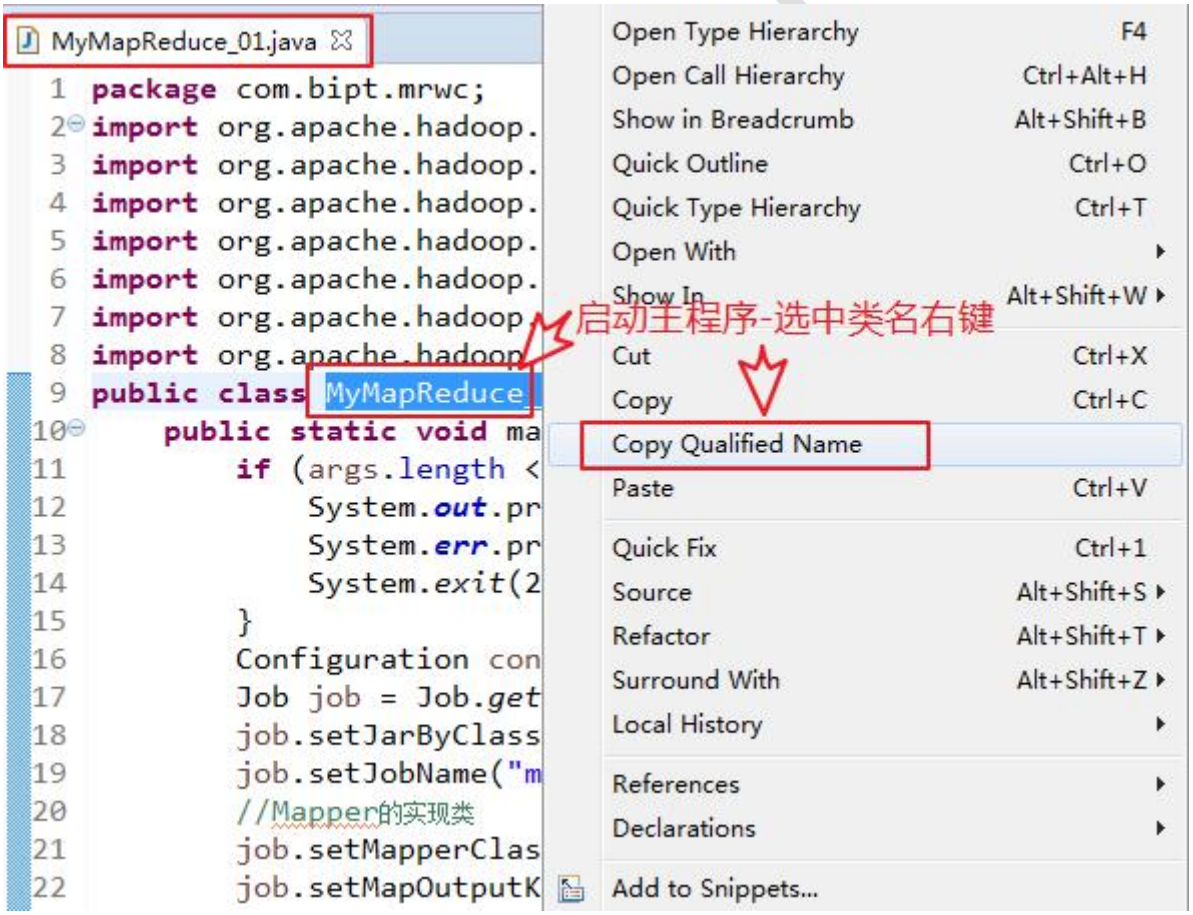
yarn jar hadoop-mapreduce-examples-2.7.1.jar wordcount /LICENSE.txt /output01

hadoop-mapreduce-examples-2.7.1.jar : MapReduce 的 jar
wordcount : 要运行的程序的名字
/LICENSE.txt 要计算的文件(是 HDFS 上的文件)
/output01 : 计算结果目录 (保证运算之前不存在)

② 自定义 jar 的运行

yarn jar mycombiner.jar com.bipt.mrwc.MyMapReduce_01 /LICENSE.txt /output01

mycombiner.jar ： 自定义实现的 wordcount 导出的 jar
com.bipt.mrwc.MyMapReduce_01 : 自定义主类的完成地址和类型、获得方式如下



02 数据二 - MapReduce 单文件实现

1. 单文件实现 wordcount

```
package com.?????;
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.LongWritable;
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;
/**
 * 自定义实现 MapReduce 中 wordcount 功能：统计词频
 */
public class MyMapReduce_02 {
    public static class MyMapReduce_02Map
        extends Mapper<LongWritable, Text, Text, IntWritable> {
        @Override
        protected void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            String[] splits = value.toString().split(" ");
            for (String s : splits) {
                // 1,拿出字符串；  2,标记 1; 每个单词都单独拿出进行标记
                // (hello, 1)
                context.write(new Text(s), new IntWritable(1));
            }
        }
    }
    public static class MyMapReduce_02Reducer
        extends Reducer<Text, IntWritable, Text, IntWritable> {
        @Override
        protected void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException {
            int count = 0;
```

```
            for (IntWritable v : values) {
                count += v.get();
            }
            context.write(key, new IntWritable(count));
        }
    }
    public static void main(String[] args) throws Exception {
        if (args.length < 2) {
            System.out.println("usage...<in>...<out>");
            System.err.println("缺少参数");
            System.exit(2);// 非领异常退出
        } else {
            Configuration conf = new Configuration();
            // 打成 jar 执行
            Job job = Job.getInstance(conf);
            job.setJarByClass(MyMapReduce_02.class);
            job.setJobName("MyMapReduce_02");// 8088 页面显示的名字
            // map 输出的数据类型是什么？
            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(IntWritable.class);
            // reduce 输出的数据类型是什么？
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(IntWritable.class);
            // 数据在哪里？
            FileInputFormat.addInputPath(job, new Path(args[0]));
            // 数据输出到哪里？
            FileOutputFormat.setOutputPath(job, new Path(args[1]));
            // 交给 yarn 去执行，直到执行结束才退出本程序
            System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
    }
}
```

2. 配置 Eclipse --> Run MapReduce 计算

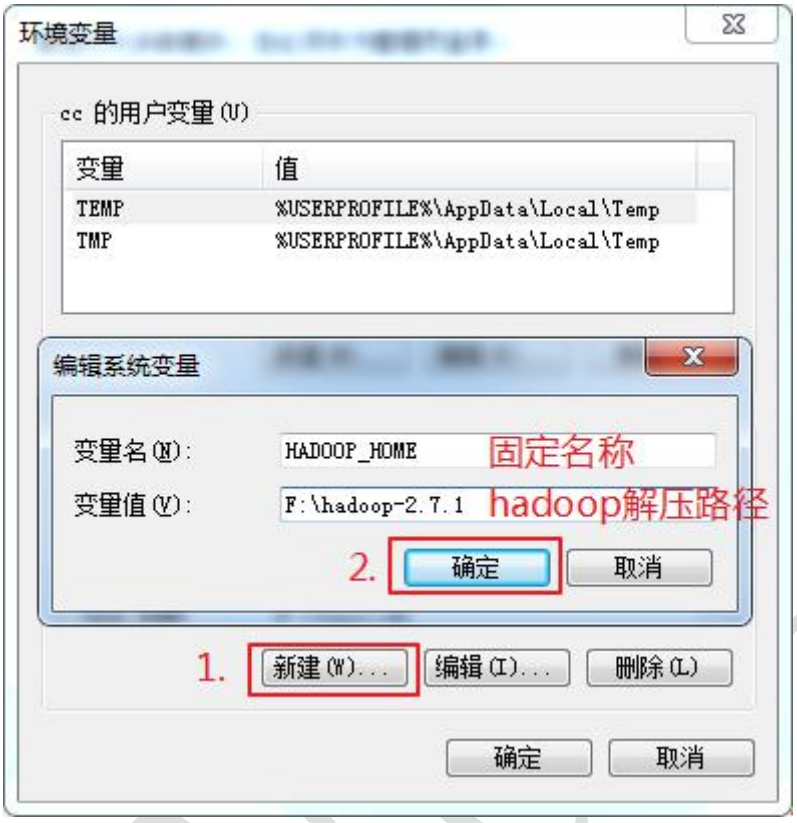
（1）本地解压 hadoop-2.7.1.tar.gz

拟定解压路径

F:\hadoop-2.7.1

（2）配置环境变量

① 添加 **HADOOP_HOME** 值：F:\hadoop-2.7.1

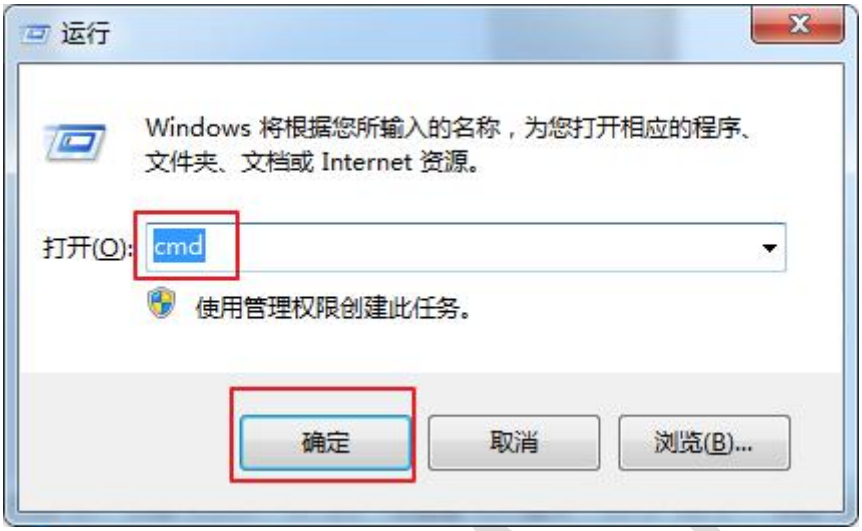


② Path 中添加 HADOOP_HOME

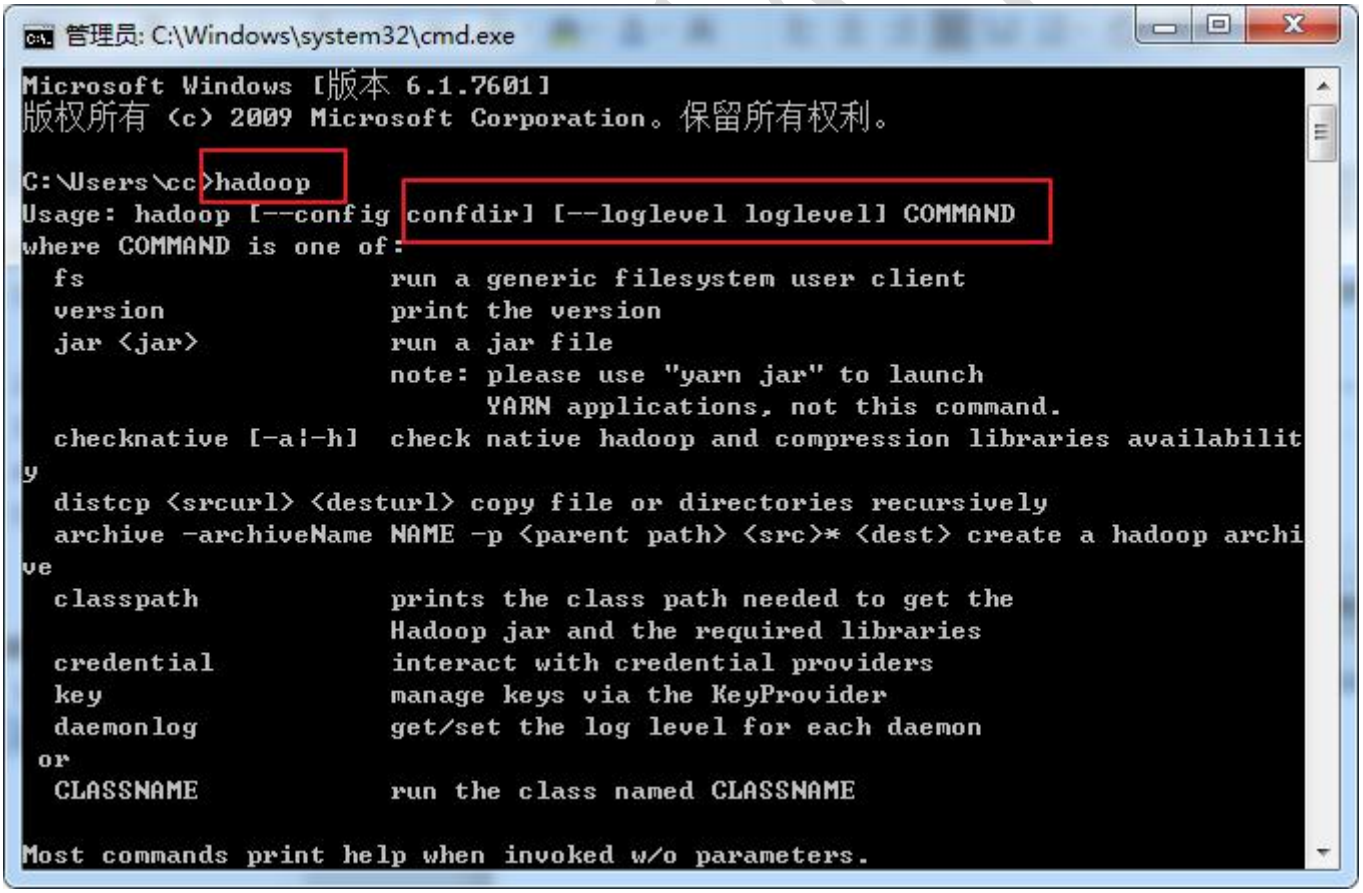
%HADOOP_HOME%\bin;%HADOOP_HOME%\sbin;

（3）CMD 测试

WIN+R



hadoop 测试环境变量



（4）Eclipse 执行方法

hadoop --> bin 下添加两个文件

F:\hadoop-2.7.1\bin

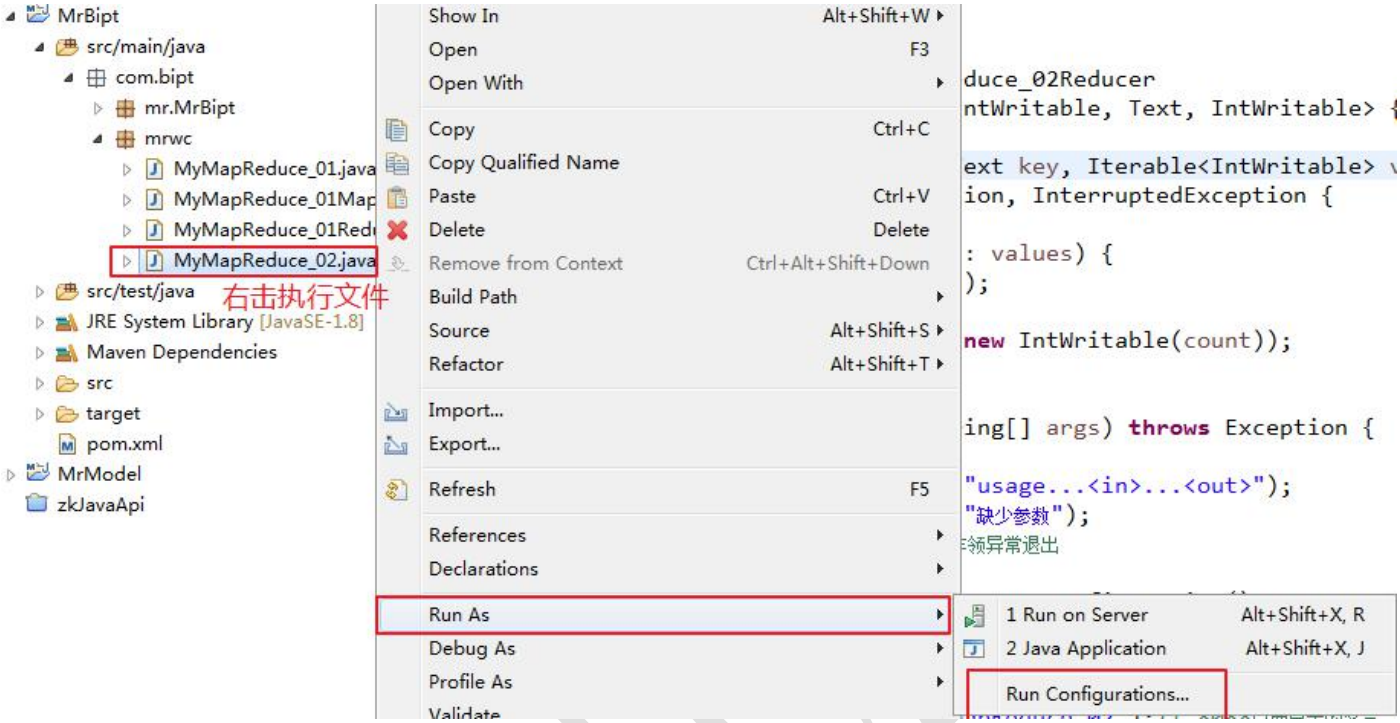
hadoop.dll

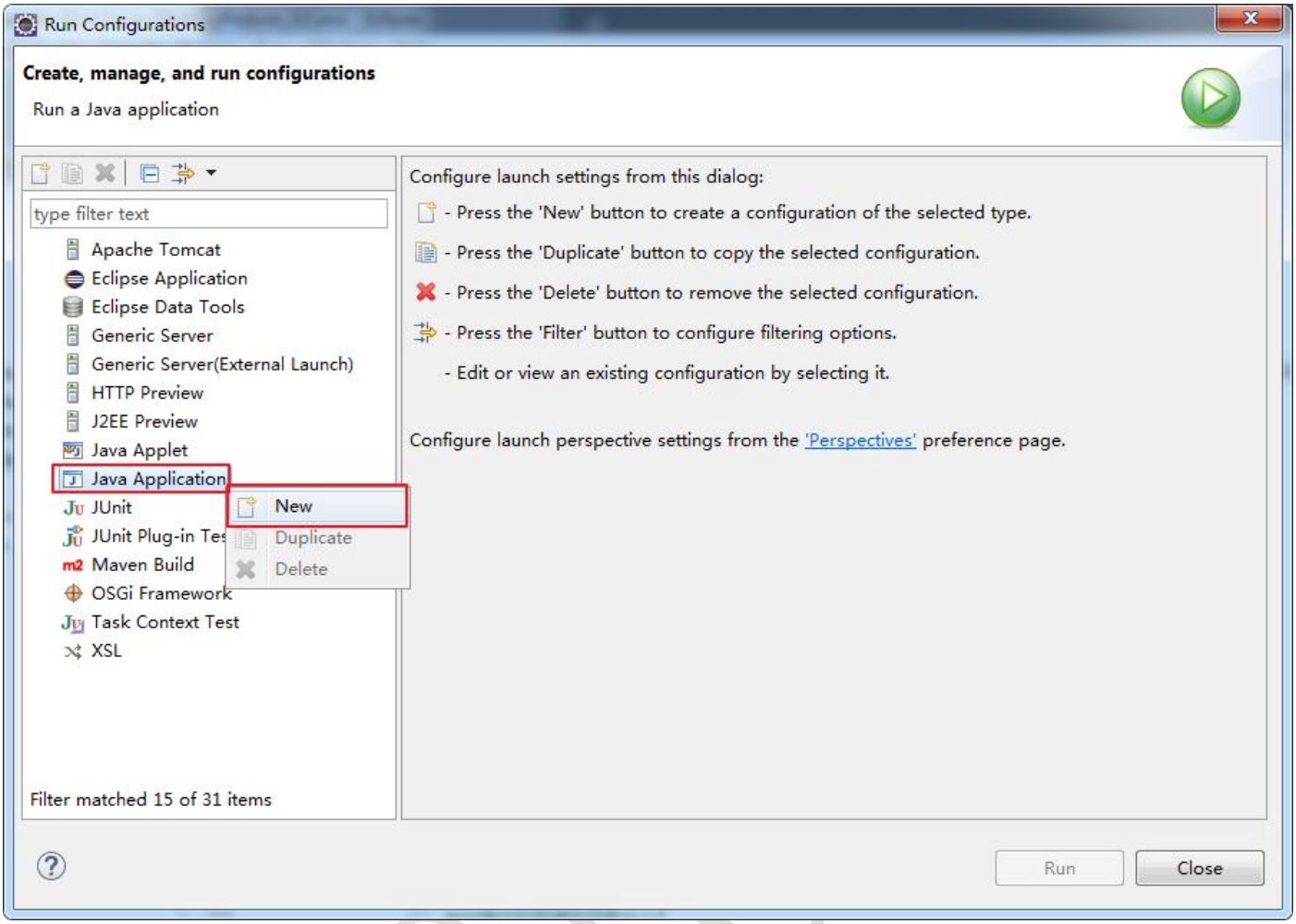
winutils.exe

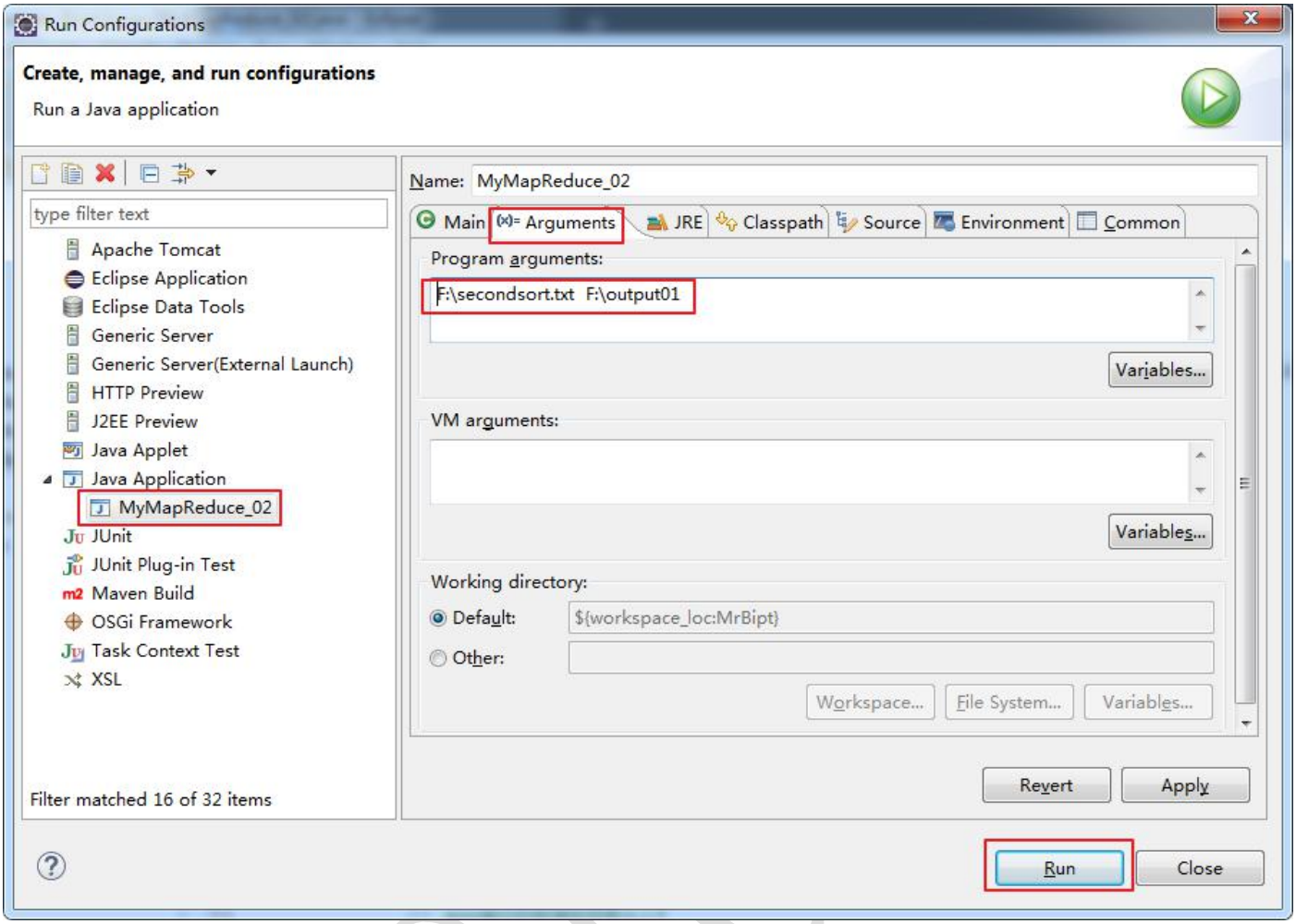
执行

执行文件右键

New 创建




Run Configurations
Create, manage, and run configurations
Run a Java application
type filter text
Apache Tomcat
Eclipse Application
Eclipse Data Tools
Generic Server
Generic Server(External Launch)
HTTP Preview
J2EE Preview
Java Applet
Java Application
JUnit
JUnit Plug-in Tes
Maven Build
OSGi Framework
Task Context Test
XSL
New
Duplicate
Delete
Filter matched 15 of 31 items
Configure launch settings from this dialog:
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.
Configure launch perspective settings from the 'Perspectives' preference page.
Run
Close



路径

F:\secondsort.txt F:\output01

secondsort.txt 需要计算的输入本地文件

output01 计算结果输出的路径（不存在的路径）

03 数据二 - MapReduce 模板和三个实例

1. 原课堂笔记

(1) task :
  ① MapTask : 每一台机器处理的每一个计算单元。一般会和 splits(输入单位)对应。
  ② ReducerTask : 每一个 reducer 端的计算单元,由程序指定(默认一个)。
(2) splits : 输入单位(一般会和 block 对应)。
(3) 如何获取文件名称 :
(4) FileSplit filesplit = (FileSplit)context.getInputSplit();
(5) String fileName = filesplit.getPath().getName();
(6) Mapper 类中 setup 方法在 map 方法调用之前调用一次
(7) Mapper 类中 cleanup 方法在 map 方法调用结束调用一次
(8) Reducer 类中的 setup 和 cleanup 同理
(9) 自定义 Combiner(输入的 k,v 必须和输出的 k,v 一致)
(10) 每一个 map 都可能会产生大量的本地输出，Combiner 的作用就是对 map 端的输出先做一次合并，以减少在 map 和 reduce 节点之间的数据传输量，以提高网络 IO 性能，是 MapReduce 的一种优化手段之一，其具体的作用如下所述。
  ① Combiner 最基本是实现本地 key 的聚合，对 map 输出的 key 排序，value 进行迭代。
  ② Combiner 还有本地 reduce 功能（其本质上就是一个 reduce），例如 Hadoop 自带的 wordcount 的例子和找出 value 的最大值的程序，combiner 和 reduce 完全一致。
(11) 自定义类型实现 Writable 接口 (条件不能放在有 map 和 reducer 两个阶段的 map 的输出 key 上)
  ① 如果想要使用 map 的 key 上的自定义类型并且有 reducer 阶段那么必须有比较规则(由于 map 到 reducer 端中间会触发一个默认的合并排序功能,这个功能是由 map 的 key 定义),必须要有 Comparable 功能。
(12) 如何在本地运行 MapRedcer
  ① 解压 hadoop 的安装目录(不能有中文)
  ② 添加环境变量 ;%HADOOP_HOME%\bin;%HADOOP_HOME%\sbin
  ③ 导入 bin 下 hadoop.dll winutils.exe
  ④ Run Configurations 设置参数

2. MapReduce 模板

（1）Base 模板 MapReduceModel

```
package com.????;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
```

```java
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
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.Tool;
import org.apache.hadoop.util.ToolRunner;
/**
 * MapRudecer 模板
 */
public class MrModel extends Configured implements Tool {
    public static class ModelMap
    extends Mapper<LongWritable, Text, Text, Text>{
        @Override
        protected void map(LongWritable key, Text value,
                Context context)
                throws IOException, InterruptedException {
        }
    }
    public static class ModelReduce
    extends Reducer<Text, Text, Text, Text>{
        @Override
        protected void reduce(Text key, Iterable<Text> values,
                Context context)
                throws IOException, InterruptedException {
        }
    }
    @Override
    public int run(String[] args) throws Exception {
        if(args.length < 2){//判断参数问题
            System.out.println("error >>> <in> ....<out>");
            System.out.println("参数有问题");
            System.exit(2);
        }
        //初始化信息
        Configuration conf = getConf();
        Job job = Job.getInstance(conf);
        job.setJobName("MrModel");//设置工作的名字
        job.setJarByClass(MrModel.class);
        //设置 mapper 有关
        job.setMapperClass(ModelMap.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);
```

```
        FileInputFormat.addInputPath(job, new Path(args[0]));
        //设置 reduce 有关
        job.setReducerClass(ModelReduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        //启动事件
        return job.waitForCompletion(true)?0:1;
    }
    public static void main(String[] args) throws Exception {
        System.exit(ToolRunner.run(new MrModel(), args));
    }
}
```

（2）自定义 IntWritable 使用相同

```
package com.????;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.Writable;
public class MyIntWritable implements Writable {
    int num = 0;  //用于计数
    public MyIntWritable(int num) {
        this.num = num;
    }
    public MyIntWritable() {
        super();
    }
    public int get(){
        return num;
    }
    @Override
    public void write(DataOutput out) throws IOException {
        // 写入数据 ， 顺序决定了读取的顺序
        out.writeInt(num);
    }
    @Override
    public void readFields(DataInput in) throws IOException {
        // 根据写入顺序读取数据
        num = in.readInt();
    }
    @Override
    public String toString() {
```

```
        return "MyIntWritable [num=" + num + "]";
    }
}
```

（3）Combiner

在 MapReduce 编程模型中，在 Mapper 和 Reducer 之间有一个非常重要的组件，它解决了上述的性能瓶颈问题，它就是 Combiner。

每一个 map 都可能会产生大量的**本地输出**，**Combiner** 的作用就是对 map 端的输出**先做一次合并**，以**减少在 map 和 reduce 节点之间的数据传输量**，以**提高网络 IO 性能**，是 MapReduce 的一种**优化手段之一**，其具体的作用如下所述。

（1）Combiner 最基本是实现**本地 key 的聚合，对 map 输出的 key 排序，value 进行迭代**。如下所示：

map: (K1, V1) → list(K2, V2)
combine: (K2, list(V2)) → list(K2, V2)
reduce: (K2, list(V2)) → list(K3, V3)

（2）Combiner 还有**本地 reduce** 功能（其本质上就是一个 reduce），例如 Hadoop 自带的 wordcount 的例子和找出 value 的最大值的程序，combiner 和 reduce 完全一致，如下所示：

map: (K1, V1) → list(K2, V2)
combine: (K2, list(V2)) → list(K3, V3)
reduce: (K3, list(V3)) → list(K4, V4)

（4）Combiner 编码实现

```
package com.????;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
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.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.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WordCountCombiner extends Configured implements Tool {
    public static class WordCountCombinerMap
        extends Mapper<LongWritable, Text, Text, IntWritable> {
```

```
        @Override
        protected void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            // \s 匹配空格 + 表示多个
            String[] splits = value.toString().split("\\s+");
            for (String string : splits) {
                context.write(new Text(string), new IntWritable(1));
                // 为了显示效果而输出 Mapper 的输出键值对信息
                System.out.println("Mapper 输出<" + string + "," + 1 + ">");
            }

        }
    }
    //combiner 用途优化 IO
    //自定义 Combiner 由于此处业务逻辑同 reducer，所以此处可以复用
    //业务逻辑不同时，自定义 combiner
    //combiner 是在 map 端运行的，在数据传输间触发
    public static class WordCountCombinerCombiner extends
        Reducer<Text, IntWritable, Text, IntWritable>{
        @Override
        protected void reduce(Text key, Iterable<IntWritable> values,
                Context context) throws IOException, InterruptedException {
            int count = 0;
            for (IntWritable value : values) {
                count += value.get();
            }
            context.write(key, new IntWritable(count));
             // 显示次数表示输出的 k2,v2 的键值对数量
            System.out.println("Combiner 输出键值对<" + key.toString() + "," + count
                    + ">");
        }
    }
    public static class WordCountCombinerReduce
            // reducer === combiner
            extends Reducer<Text, IntWritable, Text, IntWritable> {
        @Override
        protected void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException {
            int count = 0;// 计数
            for (IntWritable i : values) {
                count += i.get();
            }
            context.write(key, new IntWritable(count));
        }
    }
```

```
    @Override
    public int run(String[] args) throws Exception {
        if (args.length < 2) {    // 判断
            System.out.println("error >>> <in> ....<out>");
            System.out.println("参数有问题");
            System.exit(2);
        }
        // 初始化信息
        Configuration conf = getConf();
        Job job = Job.getInstance(conf);
        job.setJobName("WordCountCombiner");// 设置工作的名字
        job.setJarByClass(WordCountCombiner.class);
        // 设置 mapper 有关
        job.setMapperClass(WordCountCombinerMap.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        // 设置 reduce 有关
        job.setReducerClass(WordCountCombinerReduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        //由于 reducer 中的方法就是我在 combiner 中想实现的功能，所以直接（复用）引入 reducer 类可以
//      job.setCombinerClass(WordCountCombinerReduce.class);//引入 combiner 类
        job.setCombinerClass(WordCountCombinerCombiner.class);//引入 combiner 类

        // 启动事件
        return job.waitForCompletion(true) ? 0 : 1;
    }
    public static void main(String[] args) throws Exception {
        System.exit(ToolRunner.run(new WordCountCombiner(), args));
    }
}
```

3. 二次排序实例

（1）二笔排序原课堂笔记

① 二次排序 ： 使用的就是 map()-->key 上默认触发的排序功能。
② 赋值调用优化 ： 以后尽量不要在输出的位置 new 类型　推荐 set 重新赋值,复用内存。
③ MapReducer 中可以(直接)使用 zip 压缩的文件
④ 多文件链接,要有时刻 map-->key 自动合并的概念。

====数据源比如====

原数据		排序结果（1、2 列都是升序）	
3	12	1	23
1	23	2	34
2	34	3	12
32	2	12	12
12	12	32	2
32	3	32	3

（2）实现编码 Comparable

① 自定义比较类 WritableComparable

```
package com.????;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.WritableComparable;
/**
 * MapReduce 的二次排序算法
 * 自定义 IntPair 类，将示例数据中的 key/value 封装成一个整体作为 Key，
 * 同时实现 WritableComparable 接口并重写其方法。
 */
public class IntPair implements WritableComparable<IntPair> {
    int first = 0;// 第一个成员变量
    int second = 0;// 第二个成员变量
    public IntPair(int first, int second) {
        this.first = first;
        this.second = second;
    }
    public void set(int left, int right) {
        this.first = left;
        this.second = right;
    }
    public IntPair() {
    }
    @Override
    // 反序列化，从流中的二进制转换成 IntPair
    public void readFields(DataInput in) throws IOException {
        first = in.readInt();
```

```
        second = in.readInt();
    }
    @Override
    // 序列化，将 IntPair 转化成使用流传送的二进制
    public void write(DataOutput out) throws IOException {
        out.writeInt(first);
        out.writeInt(second);
    }
    /*
     * Comparable 接口的 comparaTo 方法和上面 Comparator 接口的 compare 方法类似，
     * 这里的 this 即上面的 o1，o 即上面的 o2
     * //this - o 表示升序； o-this 倒叙
     */
    @Override
    public int compareTo(IntPair o) {
        //this - o 表示升序； o-this 倒叙
        int first = this.first-o.first;
        if(first == 0){//第一排相同时候，比较第二排
            int second = this.second - o.second;
            return second;
        }
        return first;
    }
    @Override
    public String toString() {
        //决定了最后结果输出的形式和内容
//      return "IntPair [first=" + first + ", second=" + second + "]";
        return  first + " " + second;
    }
    @Override
    public int hashCode() {
        return first * 157 + second;
    }
    @Override
    public boolean equals(Object right) {
        if (right == null)
            return false;
        if (this == right)
            return true;
        if (right instanceof IntPair) {
            IntPair r = (IntPair) right;
            return r.first == first && r.second == second;
        } else {
            return false;
        }
    }
```

}

② SecondSort 实现类

```
package com.????;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
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.Tool;
import org.apache.hadoop.util.ToolRunner;

/**
 * 二次排序，对两列数据进行排序
 * 封装 WritableComparable 进行比较
 * 先比较第一列；在比较第二列
 */
public class SecondSort extends Configured implements Tool {
    public static class SecondSortMap
        extends Mapper<LongWritable, Text, IntPair, NullWritable>{
        IntPair mapk = new IntPair();
        @Override
        protected void map(LongWritable key, Text value,
              Context context)
                    throws IOException, InterruptedException {
            String[] splits = value.toString().split("\\s+");
            int first = Integer.parseInt(splits[0]);
            int second = Integer.parseInt(splits[1]);
            mapk.set(first, second);
            //NullWritable 是 Writable 的一个特殊类,实现方法为空实现,不从数据流中读数据,
            //也不写入数据，只充当占位符，如在 MapReduce 中，如果你不需要使用键或值,
            //你就可以将键或值声明为 NullWritable,NullWritable 是一个不可变的单实例类
型。
            //获取空值只能 NullWritable.get()来获取
            context.write(mapk, NullWritable.get());     }
    }
```

```
    public static class SecondSortReducer
        extends Reducer<IntPair, NullWritable, IntPair, NullWritable> {
        @Override
        protected void reduce(IntPair key, Iterable<NullWritable> values,
                Context context)
                        throws IOException, InterruptedException {
            context.write(key, NullWritable.get());
        }
    }
    @Override
    public int run(String[] args) throws Exception {
        if(args.length < 2){
            System.out.println("usage ... <in>...<out>");
            System.err.println("你的输入有误...");
            System.exit(2);
        }
        Configuration conf = getConf();
        Job job = Job.getInstance(conf);
        job.setJobName("SecondSort");
        job.setJarByClass(SecondSort.class);
        job.setMapOutputKeyClass(IntPair.class);
        job.setMapOutputValueClass(NullWritable.class);
        job.setOutputKeyClass(IntPair.class);
        job.setOutputValueClass(NullWritable.class);
        job.setMapperClass(SecondSortMap.class);
        job.setReducerClass(SecondSortReducer.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        return job.waitForCompletion(true) ? 0 : 1 ;
    }
    public static void main(String[] args) throws Exception {
        System.exit(ToolRunner.run(new SecondSort(), args));
    }
}
```


4. 天气截取实例

根据天气数据原文件，分析数据并进行截取计算（数据清洗）
天气源数据支持 *.gz 格式

```
// 数据源中：88-92 位置  表示温度
// 零度以下，需要带负号  87 是符号位置
// 空气质量是 数据源中的位置 92-93; 质量需要时 0/1/4/5/9 中的一种才属于正常情况
```

（1）编码实现 MrTemperature

```
package com.????;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
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.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.Tool;
import org.apache.hadoop.util.ToolRunner;

/**
 * 1.根据天气的原始数据，通过截取的方式，找出需要的天气
 * 2.截取指定字符串的天气
```

```
 * 判断所有数据中最高气温，最低气温；
 * 判断所有温度平均值
 */
public class MrTemperature extends Configured implements Tool {

    public static class ModelMap extends Mapper<LongWritable, Text, Text, IntWritable> {
        int MESSING = 9999;// 没有采集到天气的状态值
        Text mapkey = new Text();
        IntWritable mapv = new IntWritable();
        @Override
        protected void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            System.out.println("========map=========");
            String line = value.toString();
            String year = line.substring(15, 19);
            int airTemperature = 0;
            if (line.charAt(87) == '+') {// 数据结构中+、-代表的温度的正负
                // 数据源中：88-92 位置  表示温度
                airTemperature = Integer.parseInt(line.substring(88, 92));
            } else {
                // 零度以下，需要带负号  87 是符号位置
                airTemperature = Integer.parseInt(line.substring(87, 92));
            }
            // 空气质量是 数据源中的位置 92-93; 质量需要时 0/1/4/5/9 中的一种才属于正常情况
            String quality = line.substring(92, 93);
            if (airTemperature != MESSING && quality.matches("[01459]")) {
                //满足这个固定条件，才属于采集到的正常的温度
                mapkey.set(year);
                mapv.set(airTemperature);
                context.write(mapkey, mapv);//存入 key 年；value 温度
                System.out.println("map <"+year+":"+airTemperature+">");
            }
        }
    }

    public static class ModelReduce extends Reducer<Text, IntWritable, Text, IntWritable> {
        Text reducK = new Text();
        IntWritable reducV = new IntWritable();
        @Override
        protected void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException {
            System.out.println("========reduce=========");
            int sumT = 0;// 累加总温度
            int index = 0;
            int maxT = Integer.MIN_VALUE;//初始最小，用于保存出现的最大值
            int minT = Integer.MAX_VALUE;
```

```
            for (IntWritable v : values) {
                int c = v.get();
                if (c > maxT) {maxT = c;}
                if (c < minT) {minT = c;}
                sumT += c;// 计算总温度
                index++;
            }
            String year = key.toString();
            reducK.set(year + "maxTemp:");
            reducV.set(maxT);
            context.write(reducK, reducV);
            reducK.set(year + "minTemp:");
            reducV.set(minT);
            context.write(reducK, reducV);
            if (index != 0) {
                reducK.set(year + "avg :");
                reducV.set(sumT / index);
                context.write(reducK, reducV);
            }
            System.out.println("reduce <"+year+":"+maxT+":"+minT+">");
        }
    }

    @Override
    public int run(String[] args) throws Exception {
        // 判断
        if (args.length < 2) {
            System.out.println("error >>> <in> ....<out>");
            System.out.println("参数有问题");
            System.exit(2);
        }
        // 初始化信息
        Configuration conf = getConf();
        Job job = Job.getInstance(conf);
        job.setJobName("MrTemp");// 设置工作的名字
        job.setJarByClass(MrTemperature.class);
        // 设置 mapper 有关
        job.setMapperClass(ModelMap.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        // 设置 reduce 有关
        job.setReducerClass(ModelReduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

```
        // 启动事件
        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        System.exit(ToolRunner.run(new MrTemperature(), args));
    }
}
```

（2）两文件合并实例(外链接)

① 源数据 01-address.txt

1 Beijing
2 Guangzhou
3 Shenzhen
4 xi'an

② 源数据 02-factory.txt

factoryname addressId
BeiJing Red Star 1
ShenZhen Thunder 3
Guangzhou Honda 2
Beijing Rising 1
Guangzhou Development Bank 2
Tencent 3
Bank of Beijing 1

③ 希望结果

Bank of Beijing Beijing
Beijing Rising Beijing
BeiJing Red Star Beijing
Guangzhou Development Bank Guangzhou
Guangzhou Honda Guangzhou
Tencent Shenzhen
ShenZhen Thunder Shenzhen

④ 外链接实现

```
package com.????;

import java.io.IOException;
import java.util.ArrayList;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
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.Tool;
import org.apache.hadoop.util.ToolRunner;

/**
 * https://blog.csdn.net/ccorg/article/details/85325493
 * 外链接
 */
public class MrCityId extends Configured implements Tool {
    public static class ModelMap extends Mapper<LongWritable, Text, Text, Text> {
        Text mapk = new Text();
        Text mapv = new Text();
        @Override
        protected void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            String line = value.toString();
            if (!line.contains("address")) {// 非第一行无效的标题行可以进入
                String[] split = line.split("\\t");// tab 分割
                if ((split[0].charAt(0) + "").matches("\\d{1}")) {//判断首字母是否包含数字
                    mapk.set(split[0]);
                    mapv.set("a" + split[1]);
                } else {
                    mapk.set(split[1]);
                    mapv.set("f" + split[0]);
                }
                context.write(mapk, mapv);
            }
        }
    }

    public static class ModelReduce extends Reducer<Text, Text, Text, Text> {
        Text rek = new Text();
        Text rev = new Text();
        @Override
        protected void reduce(Text key, Iterable<Text> values, Context context)
                throws IOException, InterruptedException {
            System.out.println("=====reduce=======");
            ArrayList<String> add = new ArrayList<>();
            ArrayList<String> factory = new ArrayList<>();
            for (Text v : values) {
                String line = v.toString();
                char c = line.charAt(0);//取出第一位首字符
```

```
                String str = line.substring(1);//取出后部分使用字段
                if(c == 'a'){//地址
                    add.add(str);
                }else{//公司
                    factory.add(str);
                }
            }
            for (int i = 0; i < factory.size(); i++) {
                for (int j = 0; j < add.size(); j++) {
                    rek.set(factory.get(i));
                    rev.set(add.get(j));
                    context.write(rek, rev);
                }
            }
        }
    }

    @Override
    public int run(String[] args) throws Exception {
        if (args.length < 2) {// 判断
            System.out.println("error >>> <in> ....<out>");
            System.out.println("参数有问题");
            System.exit(2);
        }
        // 初始化信息
        Configuration conf = getConf();
        Job job = Job.getInstance(conf);
        job.setJobName("MrCityId");// 设置工作的名字
        job.setJarByClass(MrCityId.class);
        // 设置 mapper 有关
        job.setMapperClass(ModelMap.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        // 设置 reduce 有关
        job.setReducerClass(ModelReduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        // 启动事件
        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        System.exit(ToolRunner.run(new MrCityId(), args));
    }
```

```
}
```

04 数据二 - MapReduce 单排序

1. 排序合并关系（自连接）

(1) 数据源及 要求结果（列出所有孩子和祖父母的对应关系）

child	parent
Tom	Lucy
Tom	Jack
Jone	Lucy
Jone	Jack
Lucy	Mary
Lucy	Ben
Jack	Alice
Jack	Jesse
Sonn	Jam
Jam	Catt

Tom	Jesse
Jone	Jesse
Tom	Alice
Jone	Alice
Sonn	Catt
Jone	Ben
Tom	Ben
Jone	Mary
Tom	Mary

(2) 编码实现

```
package com.????;

import java.io.IOException;
import java.util.ArrayList;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
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.Tool;
import org.apache.hadoop.util.ToolRunner;
```

```
/**
 * 自连接数据清洗；截取所有孩子和祖父母的对应关系
 */
public class MrSingle extends Configured implements Tool {
    public static class ModelMap extends Mapper<LongWritable, Text, Text, Text> {
        Text mapk = new Text();
        Text mapv = new Text();
        //想写切分执行过程，见笔记
        @Override
        protected void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            System.out.println("--------------map-----------------");
            String line = value.toString();
            if (!line.contains("child")) {
                String[] splits = line.split("\\t+");
                // 右侧保存 父亲
                mapk.set(splits[0]);
                mapv.set("p" + splits[1]);
                context.write(mapk, mapv);
                // 右侧保存孩子
                mapk.set(splits[1]);
                mapv.set("c" + splits[0]);
                context.write(mapk, mapv);
            }
        }
    }

    public static class ModelReduce extends Reducer<Text, Text, Text, Text> {
        Text rek = new Text();
        Text rev = new Text();
        @Override
        protected void reduce(Text key, Iterable<Text> values, Context context)
                throws IOException, InterruptedException {
            System.out.println("--------------reduce-----------------");
            ArrayList<String> gpList = new ArrayList<>();
            ArrayList<String> cList = new ArrayList<>();
            for (Text v : values) {
                String line = v.toString();
                char c = line.charAt(0);
                String str = line.substring(1);
                if (c == 'p') {
                    gpList.add(str);
                    System.out.println("par:"+str);
                } else {
                    cList.add(str);
```

```
                System.out.println("son:"+str);
            }
        }
        String gs = "";//拼接所有祖父母信息
        for (String string : gpList) {
            gs += string+" ";
        }
        String cs = "";//拼接所有孩子信息
        for (String string : cList) {
            cs += string+" ";
        }
        System.out.println("glist= "+gs);
        System.out.println("clist= "+cs);
        for (int i = 0; i < gpList.size(); i++) {
            for (int j = 0; j < cList.size(); j++) {
                rev.set(gpList.get(i));
                rek.set(cList.get(j));
                context.write(rek, rev);
            }
        }
    }
}

@Override
public int run(String[] args) throws Exception {
    if (args.length < 2) {// 判断
        System.out.println("error >>> <in> ....<out>");
        System.out.println("参数有问题");
        System.exit(2);
    }
    // 初始化信息
    Configuration conf = getConf();
    Job job = Job.getInstance(conf);
    job.setJobName("MrSingle");// 设置工作的名字
    job.setJarByClass(MrSingle.class);
    // 设置 mapper 有关
    job.setMapperClass(ModelMap.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    // 设置 reduce 有关
    job.setReducerClass(ModelReduce.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    // 启动事件
```

```
        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        System.exit(ToolRunner.run(new MrSingle(), args));
    }

}
```

(3) 过程分析

① key 孩子 ： value 父
② key 父 ： value 子

child	parent
Tom	p+Lucy
Tom	p+Jack
Jone	p+Lucy
Jone	p+Jack
Lucy	p+Mary
Lucy	p+Ben
Jack	p+Alice
Jack	p+Jesse
Sonn	p+Jam
Jam	p+Catt

parent	child
Lucy	c+Tom
Jack	c+Tom
Lucy	c+Jone
Jack	c+Jone
Mary	c+Lucy
Ben	c+Lucy
Alice	c+Jack
Jesse	c+Jack
Jam	c+Sonn
Catt	c+Jam

③ 合并 key ： {values 集合} （每一行都是一个 task 执行一次 reduce）

key	合并
Tom	{p+Lucy p+Jack}
Jone	{p+Lucy p+Jack}
Lucy	{p+Mary p+Ben Tom Jone}
Jack	{p+Alice p+Jesse Tom Jone}
Mary	{Lucy}
Ben	{Lucy}
Alice	{Jack}
Jesse	{Jack}
Sonn	{p+Jam}
Jam	{p+Catt Sonn}

④ 依次放入列表（没有孩子，和没有祖的 无输出）

```
Lucy    {p+Mary p+Ben Tom Jone}
Jack    {p+Alice p+Jesse Tom Jone}
Jam     {p+Catt Sonn}
```

⑤ 输出结果

Tom	Jesse
Jone	Jesse
Tom	Alice
Jone	Alice
Sonn	Catt
Jone	Ben
Tom	Ben
Jone	Mary
Tom	Mary

2. 自定义分区

(1) 原课堂笔记：

① 自定义分区:把 reducer 的运算结果按照指定逻辑划分区域
② 2.reducer 的运行默认是不触发分区功能的。
③ 如果 reducer 的运行数量大于 1 ,才会触发分区。
④ reducer 的数量如果大于 1,那么可能不在一个节点(机器)上运行。
⑤ MapReducer 中默认的分区类是 HashPartitioner。
⑥ 一般情况下分区分数和 reducerTask 的个数应该是对应的。
⑦ reducerTask 设置为 0,则没有 reducer 阶段。只有 mapper 阶段。中间的过程也不会触发。

(2) 编码实现

① 自定义分区 **Partitioner**

```
package com.????;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Partitioner;
/**
 * 自定义分区 Partitioner
 * <IntWritable, NullWritable>对应 reduce 的最终输出的 key，v
 */
public class MyPartition extends Partitioner<IntWritable, NullWritable>{
    /*
     * @parm : IntWritable reduce 输出的 key
     * @parm :  NullWritable reduce 输出的 value
```

```
     * @Parma :  numPartitions 手动指定的 reduce task 的数量
     */
    @Override
    public int getPartition(IntWritable key, NullWritable value, int numPartitions) {
        int num = key.get();//得到 reduce 输出的 key
        return num % numPartitions;//数字除 3，取余的三种情况分 3 个取
    }
}
```

② 实现分区计算

```
package com.????;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
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.Tool;
import org.apache.hadoop.util.ToolRunner;
/**
 * MapRudecer 模板
 * 测试分区； 源数据是一列单纯的数据:
 * 如把每个数据除 3 取余，有三种情况，通过 3 个分区，分别保存不同情况的数据结果
 */
public class UseMyPartition extends Configured implements Tool {
    public static class ModelMap
    extends Mapper<LongWritable, Text, IntWritable, NullWritable>{
        IntWritable mapk = new IntWritable();
        @Override
        protected void map(LongWritable key, Text value,
                Context context)
                throws IOException, InterruptedException {
            System.out.println("========map========");
            mapk.set(Integer.parseInt(value.toString()));
            context.write(mapk, NullWritable.get());
        }
    }
```

```
public static class ModelReduce
extends Reducer<IntWritable, NullWritable, IntWritable, NullWritable>{
    @Override
    protected void reduce(IntWritable key, Iterable<NullWritable> values,
            Context context)
            throws IOException, InterruptedException {
        System.out.println("========reduce========");
        context.write(key, NullWritable.get());
    }
}
@Override
public int run(String[] args) throws Exception {
    if(args.length < 2){//判断
        System.out.println("error >>> <in> ....<out>");
        System.out.println("参数有问题");
        System.exit(2);
    }
    //初始化信息
    Configuration conf = getConf();
    Job job = Job.getInstance(conf);
    job.setJobName("UseMyPartition");//设置工作的名字
    //========修改当前类=========
    job.setJarByClass(UseMyPartition.class);
    //设置 mapper 有关
    job.setMapperClass(ModelMap.class);
    job.setMapOutputKeyClass(IntWritable.class);
    job.setMapOutputValueClass(NullWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    //设置 reduce 有关
    job.setReducerClass(ModelReduce.class);
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(NullWritable.class);
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    //启动事件
    job.setPartitionerClass(MyPartition.class);//引入自定义的分区类
    /*
     * HashPartitioner 默认的调用分区类 1
     * 数值大于 1 才执行自定义的 分区类
     * 如果设置 0，只有 map 阶段，没有 reduce 阶段，中间的 shuff 过程也不触发
     */
    job.setNumReduceTasks(3);//指定分区数量
    return job.waitForCompletion(true)?0:1;
}

public static void main(String[] args) throws Exception {
    //========修改当前类=========
```

```
        System.exit(ToolRunner.run(new UseMyPartition(), args));
    }

// 三个结果三个文件，可能在不同的机器上运行
// ============文件 01
//  3
//  45
//  345
//  654
//  987
// ===========文件 02
//  4
//  76
//  667
//  ============文件 03
//  2
//  5
//  23
//  44
//  122
//  5789
}
```

3. 分布式缓存

(1) 原课堂笔记

① 分布式缓存
② 1.把一些小文件放到一个可公共访问的内存中,目的是用到这个文件时不需重新加和复制。

(2) 编码实现 - 分布式缓存

① 借用 hdfs 平台，实现文件共享访问
1) 上传 blackLIst 黑名单文件到 hdfs

wangwu lisi zhagnsan

2) 上传需要计算的名单文件到 hdfs 服务器；最终显示除黑名单以外的名字

wangwu
lisi
zhaoliu
zhagnsan
chen

② 编码实现 并导出 jar

```
package com.????;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.net.URI;
import java.util.ArrayList;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
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.Tool;
import org.apache.hadoop.util.ToolRunner;

/**
 * 3. 分布式缓存；人名展示，屏蔽黑名单中的姓名
 * 使用优化资源的分布式缓存实现
 * 1)文件放到分布式缓存
 */
public class ShowName extends Configured implements Tool {
    public static class ModelMap
    extends Mapper<LongWritable, Text, Text, NullWritable>{
        Text mapk = new Text();
        @Override
        protected void map(LongWritable key, Text value,
                Context context)
                throws IOException, InterruptedException {
            ArrayList<String> blackList = new ArrayList<>();
//          File file = new File("blackList");//指定文件名称， 黑名单文件
            File file = new File("nickname");//指定文件,别名使用 #别名
            BufferedReader br = new BufferedReader(new FileReader(file));
```

```
            String str = null;
            while((str = br.readLine())!=null){
                System.out.println("======"+str);
                blackList.add(str);
            }
            String name = value.toString();
            if(!blackList.contains(name)){
                mapk.set(name);
                System.out.println("name  = "+name);
                context.write(mapk, NullWritable.get());
            }
        }
    }
    public static class ModelReduce
    extends Reducer<Text, NullWritable, Text, NullWritable>{
        @Override
        protected void reduce(Text key, Iterable<NullWritable> values,
                Context context)
                throws IOException, InterruptedException {
            context.write(key, NullWritable.get());
        }
    }
    @Override
    public int run(String[] args) throws Exception {
        //判断
        if(args.length < 2){
            System.out.println("error >>> <in> ....<out>");
            System.out.println("参数有问题");
            System.exit(2);
        }
        //初始化信息
        Configuration conf = getConf();
        Job job = Job.getInstance(conf);
        job.setJobName("ShowName");//设置工作的名字
        //========修改当前类=========
        job.setJarByClass(ShowName.class);
        //设置 mapper 有关
        job.setMapperClass(ModelMap.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(NullWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        //设置 reduce 有关
        job.setReducerClass(ModelReduce.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(NullWritable.class);
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

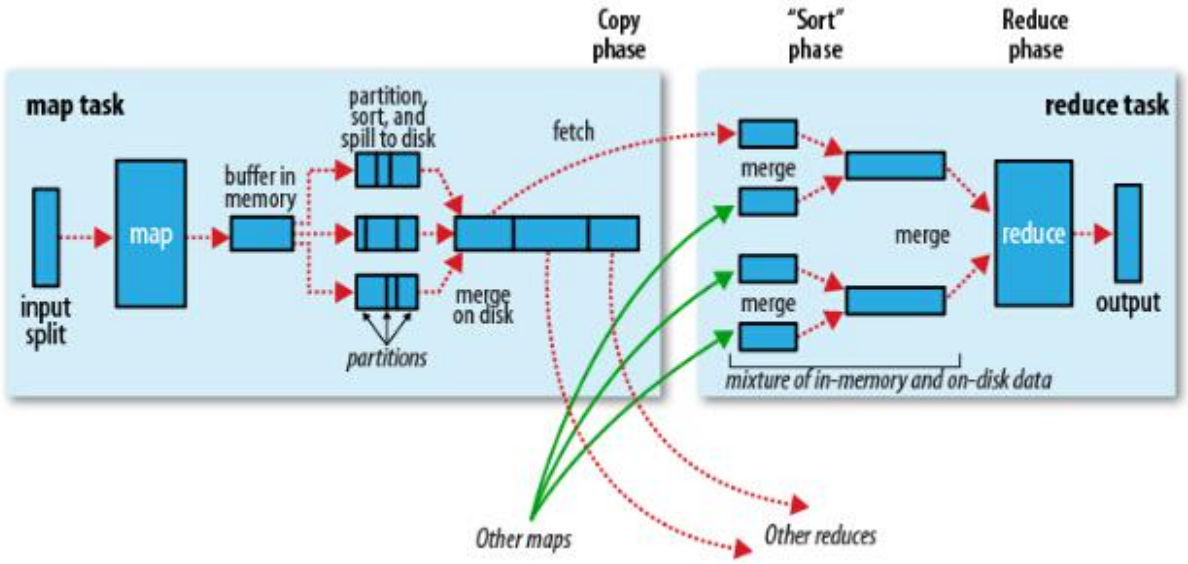
```
        /*
         * 文件放入分布式缓存：可以被所有节点公共访问的路径,会送这个节点上把文件加载到缓存中去
         * 这里暂时使用 hdfs 服务器
         */
//      job.addCacheFile(new URI("hdfs://hadoop-1:9000/blackList"));
        //路径是： hdfs 上的对应的文件 ， backList 可以起别名通过后边加 #别名，使用的时候只能用
nickname
        job.addCacheFile(new URI("hdfs://hadoop-1:9000/hdfsTest1/backList#nickname"));
        //启动事件
        return job.waitForCompletion(true)?0:1;
    }

    public static void main(String[] args) throws Exception {
        //========修改当前类=========
        System.exit(ToolRunner.run(new ShowName(), args));
    }

}
```

4. mapreduce 原理全剖析

(1) map+shuffle+reducer 全部过程

wordcount 为例

Map --> mapShuffler --> reduceShuffler --> Reduce



Reducer shuffle 会拉取 mapshffle 端处理完成的数据

----------mapper 的工作----------

1. mapper 开始运行，调用 InputFormat 组件读取文件逻辑切片（逻辑切片不是 block 块，切片大小默认和 block 块大小相同 可以是 block 大小的 110%）
2. 经过 inputformat 组件处理后，文件以<k,v>的形式进入我们自定义的 mapper 逻辑,经过 map 逻辑处理。

----------mapper shuffle------start----------

3. mapper 逻辑中输出结果会调用 OutPutCollector 组件写入环形缓冲区。
4. 环形缓冲区的存储达到默认阀值会调用 Splliller 组件将内容分区且排序（快排算法，外部排序算法）后溢写到磁盘文件中，mapper 最后结果不满环形缓冲区也会溢写到磁盘。
5. mapper 结束后磁盘中的结果小文件会合并（merge），产生大文件（分区且排序，归并算法）。

----------mapper shuffle---end----------

----------reducer shuffle---start----------

6. reducer shuffle 启动后会到不同的 map 结果文件中拉取相同区号的结果文件，再合并这些来自不同 map 的结果文件，再将这些文件合并（归并算法），产生的大文件是分区且排序且分好组了的，分组调用默认的 GroupingComparator 组件。
7. reducer 把拉取的所有 map 输出文件合并完成之后就会开始读取文件，将读入的内容以<k,v>的形式输入到我们用户自定义的 reducer 处理逻辑中。

----------reducer shuffle---end----------

----------reducer--阶段开始----------

8. 自定义的 reducer 进行处理。
9. 用户逻辑完成之后以<k,v>的形式调用 OutPutFormat 组件输出到 hdfs 文件系统中去保存。

总结 : map shuffle 环形缓冲区(默认大小 100M 阈值是 80%) 合并 排序 分区 对 map 处理之后的数据再处理 combiner 中。

combiner ： 在 map 端计算之后,子节点上还可以进行计算,这个计算叫 combiner 。

Reducer shffule 在缓冲中合并排序 缓冲没有固定值 处理完成会提供给 reducer。

splits ： map 端的输入单位,对应文件的 block,如果最后一个 block 的大小小于 blockSize 的 10%,把最后一个 block 合并到上一个,形成一个 split。

map task : map 端的计算是并行的,每一个并行任务交一个 maptask。maptask 的数量由 split 的数量决定。所以 map 的输出对应 Task 的数量。

reduce task : reduce 端的 task 是以 partition(分区)的个数决定。

【只有在指定分区个数 ＞1 的情况下会触发分区】,如果默认或分区个数是 1,reduceTask 的个数是 1,如果分区大于 1,也就是 reduceTask 的数量大于 1,输出的文件个数会大于 1。reduceTask 的个数如果大于 1,则和 patition 的个数相同,reduce 端的输出文件个数也和 reduceTask 的个数相同。

mapreduce : 默认提供了一个分区类,HashPartitioner。