066课时 Hadoop MapReduce框架中的数据类型讲解

Hadoop的自己的需求，Java的数据类型不适合Hadoop框架.如我们在程序中使用的几个Class: Text

public class Text extends BinaryComparable

implements WritableComparable<BinaryComparable>

我们查看WritableComparable是一个数据接口。

public interface WritableComparable<T> extends Writable, Comparable<T>

1. Hadoop中的所有的数据key/value类型必须实现Writable接口，有两个方法：

**void** write(DataOutput out) **throws** IOException;

**void** readFields(DataInput in) **throws** IOException;

分别用于读（反序列化）和写（序列化操作）

关于Wirtable接口的说明：

|  |
| --- |
| **public** **interface** Writable {  /\*\*  \* Serialize the fields of this object to <code>out</code>.  \*  \* **@param** out <code>DataOuput</code> to serialize this object into.  \* **@throws** IOException  \*/  **void** write(DataOutput out) **throws** IOException;  /\*\*  \* Deserialize the fields of this object from <code>in</code>.  \*  \* <p>For efficiency, implementations should attempt to re-use storage in the  \* existing object where possible.</p>  \*  \* **@param** in <code>DataInput</code> to deseriablize this object from.  \* **@throws** IOException  \*/  **void** readFields(DataInput in) **throws** IOException;  } |

1. 所有的key必须实现Comparable接口，在MapReduce过程中，需要对于key/value进行反复的排序,默认情况下，根据key进行排序，所以所有的key类型必须实现Comparable中的compareTo()方法。因此在Hadoop中key的类型需要实现Writable接口，又要实现Comparable接口，在hadoop中有一个公共的接口WritableComparable接口。

public interface WritableComparable<T> extends Writable, Comparable<T>

1. 由于需要进行序列化和反序列化，对于JAVA对象需要重写一些方法
   1. public int hashCode()
   2. public boolean equals(Objexct o)
   3. public String toString()

以IntWritable为例：

|  |
| --- |
| **public** **class** IntWritable **implements** WritableComparable {  **private** **int** value;  **public** IntWritable() {}  **public** IntWritable(**int** value) { set(value); }  /\*\* Set the value of this IntWritable. \*/  **public** **void** set(**int** value) { **this**.value = value; }  /\*\* Return the value of this IntWritable. \*/  **public** **int** get() { **return** value; }  **public** **void** readFields(DataInput in) **throws** IOException {  value = in.readInt();  }  **public** **void** write(DataOutput out) **throws** IOException {  out.writeInt(value);  }  /\*\* Returns true iff <code>o</code> is a IntWritable with the same value. \*/  **public** **boolean** equals(Object o) {  **if** (!(o **instanceof** IntWritable))  **return** **false**;  IntWritable other = (IntWritable)o;  **return** **this**.value == other.value;  }  **public** **int** hashCode() {  **return** value;  }  /\*\* Compares two IntWritables. \*/  **public** **int** compareTo(Object o) {  **int** thisValue = **this**.value;  **int** thatValue = ((IntWritable)o).value;  **return** (thisValue<thatValue ? -1 : (thisValue==thatValue ? 0 : 1));  }  **public** String toString() {  **return** Integer.*toString*(value);  }  } |

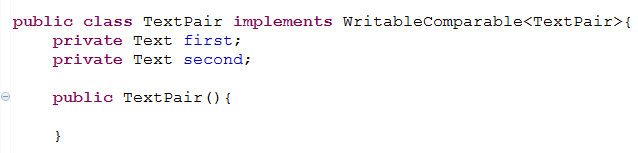
1. Hadoop中的数据类型必须有一个无参数的构造方法，这样为了方便的反射创建对象.序列化（Serialization）是将结构化对象转化成为字节流，以便在网络上传输，或者是写入到磁盘进行永久性的存储。反序列化（deserialization）是指将字节流转换成为结构化的对象。序列化在分布式数据处理的两个领域的重要应用：进程间通信和网络传输。

在Hadoop中，系统中多个节点上进程间的通信是通过：远程过程调用（Remote produce call,RPC）实现的。RPC协议将消息序列化为二进制的数据流发送到远程节点，远程节点将二进制流反序列化为原始的消息。

在Hadoop中有自己的序列化格式Writable，它格式紧凑，速度快，但是难以通过Java之外的语言进行扩展。Writable是Hadoop的核心。MapReduce框架都会使用它。

常见的Hadoop数据类型：BooleanWritable ByteWritable DoubleWritable FloatWritable IntWritable LongWritable Text(使用UTF-8进行存储的文本) NullWritable 空值

1. 在自定义的数据类型中，建议使用Java原生数据类型，最好不要使用hadoop对原生类型封装好的数据类型。即不推荐使用这种Hadoop的封装好的数据类型。在性能方面、反序列化方面存在问题。



067课时 Hadoop MapReduce数据比较器Comparator

当前的问题：当数据写入磁盘中，如果需要对数据进行排序的话，需要将数据从磁盘中的二进制流进行反序列化，成为对象，然后在内存中对于反序列化的对象进行比较。

但是在hadoop中也可以实现对于字节进行比较，不需要进行反序列化的操作。实现上述功能，则在hadoop数据类型需要实现一个RawComparator接口

|  |
| --- |
| **public** **interface** RawComparator<T> **extends** Comparator<T> {  **public** **int** compare(**byte**[] b1, **int** s1, **int** l1, **byte**[] b2, **int** s2, **int** l2);  } |

RawComparator：对于MapReduce来说，类型比较非常重要，因为中间有一个基于键的排序阶段。Hadoop提供了一个优化接口是继承自Java的 Comparator的 RawComparator接口。该接口允许其实现直接比较数据流中的记录，无需把数据流反序列化，这样避免了新建对象的开销。比如我们可以根据IntWritable接口实现的Comparator实现compare()方法，该方法可以从每个字节数组b1&b2中读取给定的起始位置s1&s2以及长度l1&l2的一个整数进行直接的比较。

WritableComparator是对继承WritableComparable类的RawComparator接口的一个通用的实现。(比如TextPair是继承了WritableComparable的类，WritableComparator是对于RawComparator的一个通用的实现类 )他提供了两个主要功能：第一提供对于原始compare()方法的一个默认实现，该方法能够反序列化将在流中的进行比较的对象，并且调用对象的compare()方法；第二它充当了RawComparator实例的工厂（已注册Writable的实现）。以IntWritable为例：为了获得IntWritable的comparator，我们直接调用：

RawComparator<IntWritable> comparator = WritableComparator.get(IntWritable.class);

通过comparator可以比较两个IntWritable对象：

IntWritable w1 = new IntWritable(111), w2 =new IntWritable(120);

assertThat(comparator.compare(w1,w2). greaterThan(0));

或者是将其序列化：

byte[] b1 =serilize(w1); byte[] b2 = serilize(w2);

assertThat(comparator.compare(b1,0,b1.length, b2, 0, b2.length));

在hadoop中有一个针对Writable数据类型，进行实现的一个通用的实现类型WritableComparator类。所有类型只需要继承通用了，再去需要具体覆写相应的compare方法。比如IntWritable中

**public** **class** WritableComparator **implements** RawComparator;

|  |
| --- |
| /\*\* A WritableComparator optimized for Text keys. \*/  **public** **static** **class** Comparator **extends** WritableComparator {  **public** Comparator() {  **super**(Text.**class**);  }  **public** **int** compare(**byte**[] b1, **int** s1, **int** l1,  **byte**[] b2, **int** s2, **int** l2) {  **int** n1 = WritableUtils.*decodeVIntSize*(b1[s1]);  **int** n2 = WritableUtils.*decodeVIntSize*(b2[s2]);  **return** *compareBytes*(b1, s1+n1, l1-n1, b2, s2+n2, l2-n2);  }  }  **static** {  // register this comparator  WritableComparator.*define*(Text.**class**, **new** Comparator());  } |

对于自定义的Comparator需要实现以下几步：

1. 推荐在Hadoop类型内部实现静态内部类Comparator，继承WritableComparator。
2. 重写默认无参数的构造方法，方法内部必须调用父类有参数的构造方法。即

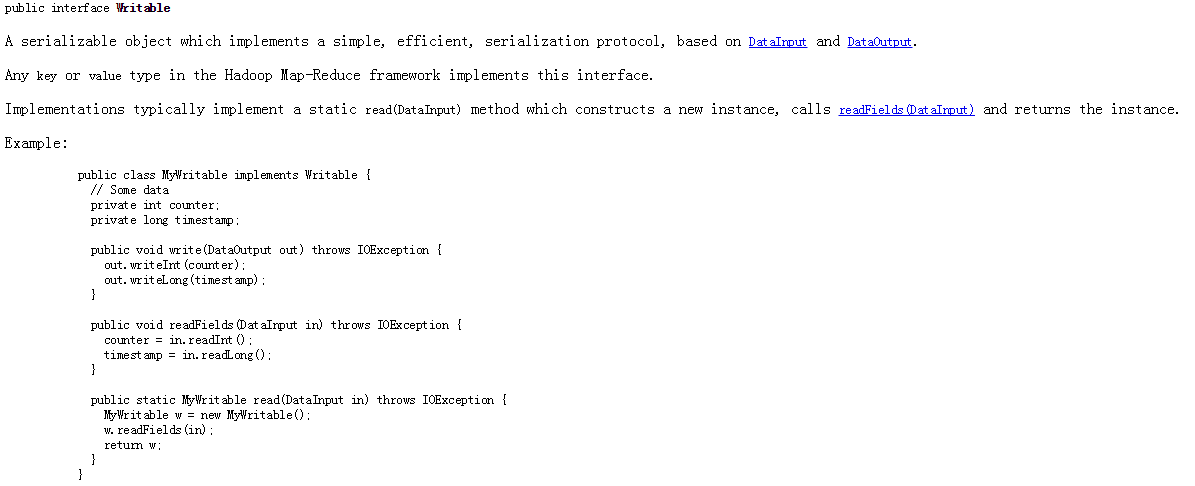
|  |
| --- |
| **protected** Comparator() {  **super**(TextPair.**class**);  } |

1. 重载父类的compare()方法，根据具体实现该方法
2. 向WritableComparator类中注册自定义的Comparator类，如下:

|  |
| --- |
| **static** {  // register this comparator  WritableComparator.*define*(TextPair.**class**, **new** Comparator());  } |

068课时 复习总结如何自定义数据类型

1. Hadoop中的Writable接口中API实现例子

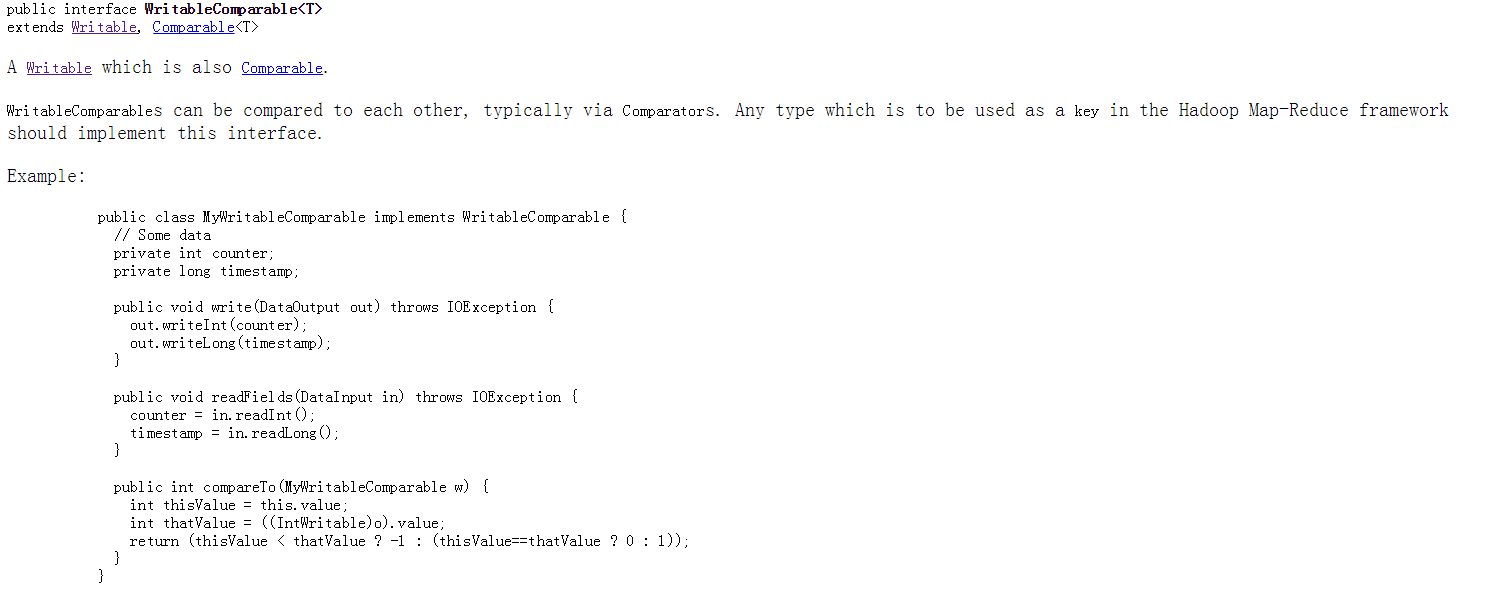


Implements static read(DataInput in) method which constructs a new instance, calls readFields(DataInput in) and returns the instance for serialization.

我们自己实现一个MyWritable:

|  |
| --- |
| **//MyWritable.java**  **public** **class** MyWritable **implements** Writable {    **private** **int** counter;  **private** **long** timestamp;    **public** MyWritable() {  // **TODO** Auto-generated constructor stub  }    **public** MyWritable(**int** counter, **long** timestamp){  **this**.set(counter, timestamp);  }      **public** **static** MyWritable read(DataInput in) **throws** IOException{  MyWritable myWritable = **new** MyWritable();  myWritable.readFields(in);    **return** myWritable;  }  **public** **void** set(**int** counter, **long** timestamp){  **this**.counter = counter;  **this**.timestamp = timestamp;  }      **public** **int** getCounter() {  **return** counter;  }    **public** **long** getTimestamp() {  **return** timestamp;  }  **public** **void** write(DataOutput out) **throws** IOException {  out.writeInt(counter);  out.writeLong(timestamp);  }  **public** **void** readFields(DataInput in) **throws** IOException {  **this**.counter = in.readInt();  **this**.timestamp = in.readLong();  }  @Override  **public** String toString() {  **return** **this**.counter+"\t"+ **this**.timestamp;  }  @Override  **public** **int** hashCode() {  **final** **int** prime = 31;  **int** result = 1;  result = prime \* result + counter;  result = prime \* result + (**int**) (timestamp ^ (timestamp >>> 32));  **return** result;  }  @Override  **public** **boolean** equals(Object obj) {  **if** (**this** == obj)  **return** **true**;  **if** (obj == **null**)  **return** **false**;  **if** (getClass() != obj.getClass())  **return** **false**;  MyWritable other = (MyWritable) obj;  **if** (counter != other.counter)  **return** **false**;  **if** (timestamp != other.timestamp)  **return** **false**;  **return** **true**;  }  } |

1. Hadoop 中 API之 WritableComparable



Hadoop中的 DataType中所有的Key，必须实现WritableComparable接口。

1. 比较器RawComparator官方API

RawComparator接口中的一个方法

int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2);

基于字节的比较，在Hadoop中有一个通用的实现类WritableComparator类, 在定义的类中实现一个内部类Comparator继承WritableComparator类

1. Hadoop数据类型序列化的大小

BooleanWritable 1;ByteWritable 1; IntWritabel 4; VintWritable 1-5; FloatWritable 4; DoubleWritable 8;LongWritable 8; VlongWritable 1-9.

NullWritable Text BytesWritable MD5Hash ObjectWritable GenericWritable

NullWritable是Writable的一个特殊的数据类型，序列化长度为0.不从数据流中读取，也不写入数据流，他是一个不可变的单实例类型，通过调用NullWritable.get()方法可以获取这个实例。

NullWritable类型的实现代码

|  |
| --- |
| **//NullWritable.java 源代码**  **public** **class** NullWritable **implements** WritableComparable {  **private** **static** **final** NullWritable *THIS* = **new** NullWritable();  **private** NullWritable() {} // no public ctor  /\*\* Returns the single instance of this class. \*/  **public** **static** NullWritable get() { **return** *THIS*; }    **public** String toString() {  **return** "(null)";  }  **public** **int** hashCode() { **return** 0; }  **public** **int** compareTo(Object other) {  **if** (!(other **instanceof** NullWritable)) {  **throw** **new** ClassCastException("can't compare " + other.getClass().getName()  + " to NullWritable");  }  **return** 0;  }  **public** **boolean** equals(Object other) { **return** other **instanceof** NullWritable; }  **public** **void** readFields(DataInput in) **throws** IOException {}  **public** **void** write(DataOutput out) **throws** IOException {}  /\*\* A Comparator &quot;optimized&quot; for NullWritable. \*/  **public** **static** **class** Comparator **extends** WritableComparator {  **public** Comparator() {  **super**(NullWritable.**class**);  }  /\*\*  \* Compare the buffers in serialized form.  \*/  **public** **int** compare(**byte**[] b1, **int** s1, **int** l1,  **byte**[] b2, **int** s2, **int** l2) {  **assert** 0 == l1;  **assert** 0 == l2;  **return** 0;  }  }  **static** { // register this comparator  WritableComparator.*define*(NullWritable.**class**, **new** Comparator());  }  } |

1. ObjectWritable & GenericWritable

ObjectWritable是针对Java基本类型或者是有这些基本类型组成的数组的一个通用的封装。在Hadoop RPC中用于对方法的参数和返回值类型进行封装和解封装。当一个字段中包含多个类型的时候，ObjectWritable是非常有用的，比如SequenceFile中的值包含多个类型，就可以将值类型声明为ObjectWritable类型，并且将每一个类型封装在一个ObjectWritable中。作为一个通用的机制，每一次序列化都写封装类型的名称，会非常的浪费空间。如果封装的类型数量比较少，并且能够提前知道，那么可以通过使用静态类型的数组，并使用对序列化后的类型的引用加入位置索引提高性能。这是GenericWritable类采取的方法，并且你可以在继承的子类中指定需要支持的类型。 详细资料自己查找使用方法。

还有一些类：ArrayWritable ArrayPrimitiveWritable TwoDArrayWritable MapWritable SortedMapWritable EnumSetWritable.

069 70课时 MapReduce 应用案例手机流量统计及其实现

根据电信的实际业务，进行手机流量的数据统计

**业务分析**

1. 分析业务需求：用户上网存在上传（发送信息流量）下载数据（接收数据流量）。每一种流量在网络传输过程中有两种形式：包的大小和流量的大小。使用手机上网，以手机号为唯一标识进行记录。记录内容包括很多信息：

BeginTime(采集第一个数据包的时间) EndTime（采集最后一个数据包的时间）

MSISDN（用户手机号） SourceIP（用户IP） SourcePort（用户端口号）

APIP（ AP IP） APMAC （AP mac）

ACIP（AC ip） ACMAC（AC mac）

UpPackNum 上行数据包数目，单位个；UpPayLoad 上行总流量，注意单位换算 byte

DownPackNum 下行数据包，DownPayLoad 下行总流量

HttpStatus HTTP的 Response 状态

我们这里需要的数据类型：MSISDN、UpPackNum UpPayLoad DownPackNum DownPayLoad

1. 自定义数据类型 DataWritable 实现WritableComparable接口
2. 编写MapReduce类

Map：从文件中获取数据，抽取五个字段，输出key为手机号，输出value是数据量的类型DataWritable

Reduce：将相同的手机号的Value值进行相加合并，得出数据的总流量（数据包和数据流量），输出到文件中，以制表符分开。

**业务处理过程：**

首先对于原始数据（存储在文件或者是关系型数据库中）进行多次的数据的清理和筛选，使数据符合我们的要求，将脏数据删除掉。

Sqoop框架， 将关系型数据和Hbase、Hive以及HDFS中导入导出数据，但是在实际的业务中，数据都是脏数据，需要自己编写程序对于数据进行清洗。

//DataWritable.java

|  |
| --- |
| **import** java.io.DataInput;  **import** java.io.DataOutput;  **import** java.io.IOException;  **import** org.apache.hadoop.io.Writable;  **public** **class** DataWritable **implements** Writable {  // up load  **private** **int** upPackNum;  **private** **int** upPayLoad;  // down load  **private** **int** downPackNum;  **private** **int** downPayLoad;  **public** DataWritable() {  }  **public** DataWritable(**int** upPackNum, **int** upPayLoad, **int** downPackNum,  **int** downPayLoad) {  **this**.set(upPackNum, upPayLoad, downPackNum, downPayLoad);  }  **public** **void** set(**int** upPackNum, **int** upPayLoad, **int** downPackNum,  **int** downPayLoad) {  **this**.upPackNum = upPackNum;  **this**.upPayLoad = upPayLoad;  **this**.downPackNum = downPackNum;  **this**.downPayLoad = downPayLoad;  }  **public** **int** getUpPackNum() {  **return** upPackNum;  }  **public** **int** getUpPayLoad() {  **return** upPayLoad;  }  **public** **int** getDownPackNum() {  **return** downPackNum;  }  **public** **int** getDownPayLoad() {  **return** downPayLoad;  }  **public** **void** write(DataOutput out) **throws** IOException {  out.writeInt(upPackNum);  out.writeInt(upPayLoad);  out.writeInt(downPackNum);  out.writeInt(downPayLoad);  }  **public** **void** readFields(DataInput in) **throws** IOException {  **this**.upPackNum = in.readInt();  **this**.upPayLoad = in.readInt();  **this**.downPackNum = in.readInt();  **this**.downPayLoad = in.readInt();  }  @Override  **public** String toString() {  **return** upPackNum + "\t" + upPayLoad + "\t" + downPackNum + "\t"  + downPayLoad;  }  } |

// DataTotalMapReduce.java

|  |
| --- |
| **import** java.io.IOException;  **import** org.apache.hadoop.conf.Configuration;  **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;  /\*  \* 手机流量分析  \*/  **public** **class** DataTotalMapReduce {  // Map class  **static** **class** DataTotalMapper **extends**  Mapper<LongWritable, Text, Text, DataWritable> {  **private** DataWritable dataWritable = **new** DataWritable();  **private** Text mapOutputKey = **new** Text();    @Override  **public** **void** map(LongWritable key, Text value, Context context)  **throws** IOException, InterruptedException {  String lineValue = value.toString();    //split  String[] strs = lineValue.split("\t");    //get data  String phoneNum = strs[1];  **int** upPackNum = Integer.*valueOf*(strs[6]);  **int** upPayLoad = Integer.*valueOf*(strs[7]);  **int** downPackNum = Integer.*valueOf*(strs[8]);  **int** downPayLoad = Integer.*valueOf*(strs[9]);    //set map output  mapOutputKey.set(phoneNum);  dataWritable.set(upPackNum, upPayLoad, downPackNum, downPayLoad);    context.write(mapOutputKey, dataWritable);  }  }  // Reduce class  **static** **class** DataTotalReducer **extends**  Reducer<Text, DataWritable, Text, DataWritable> {  **private** DataWritable dataWritable = **new** DataWritable();    @Override  **public** **void** reduce(Text key, Iterable<DataWritable> values,  Context context) **throws** IOException, InterruptedException {  **int** upPackNum = 0;  **int** upPayLoad = 0;  **int** downPackNum = 0;  **int** downPayLoad =0;    **for**(DataWritable data: values){  upPackNum += data.getUpPackNum();  upPayLoad += data.getUpPayLoad();  downPackNum += data.getDownPackNum();  downPayLoad += data.getDownPayLoad();  }    //set datawritable  dataWritable.set(upPackNum, upPayLoad, downPackNum, downPayLoad);    //set reduce/job output  context.write(key, dataWritable);  }  }  // dirver code    **public** **int** run(String[] args) **throws** Exception{  //get conf  Configuration conf = **new** Configuration();    //create job  Job job = **new** Job(conf,DataTotalMapReduce.**class**.getSimpleName());    //set job  job.setJarByClass(DataTotalMapReduce.**class**);  //job input  Path inputDirPath = **new** Path(args[0]);  FileInputFormat.*addInputPath*(job, inputDirPath);  //job map class  job.setMapperClass(DataTotalMapper.**class**);  job.setMapOutputKeyClass(Text.**class**);  job.setMapOutputValueClass(DataWritable.**class**);    //job reduce class  job.setReducerClass(DataTotalReducer.**class**);  job.setOutputKeyClass(Text.**class**);  job.setOutputValueClass(DataWritable.**class**);      //job output  Path outputDir = **new** Path(args[1]);  FileOutputFormat.*setOutputPath*(job, outputDir);        //submit job  **boolean** isSuccess = job.waitForCompletion(**true**);      **return** isSuccess?0:1;    }  **public** **static** **void** main(String[] args) **throws** Exception {  args = **new** String[]{  "hdfs://hadoop-master:9000/opt/data/wc/input",  "hdfs://hadoop-master:9000/opt/data/wc/output"  };    **int** status = **new** DataTotalMapReduce().run(args);  System.*exit*(status);  }  } |

071课时 Mapper基类的讲解

Mapper源代码和API文档了解到

1. InputSplit输入分片，InputFormat
2. 对于Mapper输出结果进行Sorted & Group
3. 对于Mapper输出结果依据Reduce个数进行Partition
4. 对Mapper输出结果进行combiner

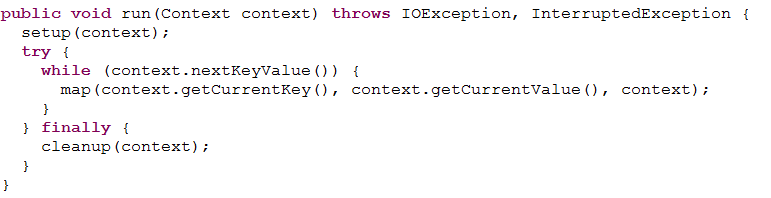
Mapper方法定义如下：

第一类：protected类型，用户根据实际情况进行覆写

1. setup：每一个任务执行前调用一次，对于MapTask进行预处理
2. map:每一个key-value对调用一次
3. cleanup：每一个任务执行结束调用一次，对MapTask进行清理工作

第二类：运行的方法（模板）

run()，是Mapper类的入口，方法内部调用了setup() map() cleanup() 方法，对每一个context.nextKeyValue()获得的key-value调用map方法进行处理，提供了上面三个函数的执行模板，是MapTask的驱动程序。



API文档中英文文档描述如下：

The Hadoop Map-Reduce framework spawns one map task for each [InputSplit](eclipse-javadoc:%E2%98%82=MapReduce-WordCount/C:%5C/Users%5C/yang%5C/Desktop%5C/hadoop-1.2.1%5C/hadoop-core-1.2.1.jar%3Corg.apache.hadoop.mapreduce(Mapper.class%E2%98%83Mapper%E2%98%82InputSplit) generated by the [InputFormat](eclipse-javadoc:%E2%98%82=MapReduce-WordCount/C:%5C/Users%5C/yang%5C/Desktop%5C/hadoop-1.2.1%5C/hadoop-core-1.2.1.jar%3Corg.apache.hadoop.mapreduce(Mapper.class%E2%98%83Mapper%E2%98%82InputFormat) for the job. Mapper implementations can access the [Configuration](eclipse-javadoc:%E2%98%82=MapReduce-WordCount/C:%5C/Users%5C/yang%5C/Desktop%5C/hadoop-1.2.1%5C/hadoop-core-1.2.1.jar%3Corg.apache.hadoop.mapreduce(Mapper.class%E2%98%83Mapper%E2%98%82Configuration) for the job via the [JobContext.getConfiguration()](eclipse-javadoc:%E2%98%82=MapReduce-WordCount/C:%5C/Users%5C/yang%5C/Desktop%5C/hadoop-1.2.1%5C/hadoop-core-1.2.1.jar%3Corg.apache.hadoop.mapreduce(Mapper.class%E2%98%83Mapper%E2%98%82JobContext%E2%98%82getConfiguration%E2%98%82).

The framework first calls [setup(org.apache.hadoop.mapreduce.Mapper.Context)](eclipse-javadoc:%E2%98%82=MapReduce-WordCount/C:%5C/Users%5C/yang%5C/Desktop%5C/hadoop-1.2.1%5C/hadoop-core-1.2.1.jar%3Corg.apache.hadoop.mapreduce(Mapper.class%E2%98%83Mapper%E2%98%82%E2%98%82setup%E2%98%82org.apache.hadoop.mapreduce.Mapper.Context), followed by [map(Object, Object, Context)](eclipse-javadoc:%E2%98%82=MapReduce-WordCount/C:%5C/Users%5C/yang%5C/Desktop%5C/hadoop-1.2.1%5C/hadoop-core-1.2.1.jar%3Corg.apache.hadoop.mapreduce(Mapper.class%E2%98%83Mapper%E2%98%82%E2%98%82map%E2%98%82Object%E2%98%82Object%E2%98%82Context) for each key/value pair in the InputSplit. Finally [cleanup(Context)](eclipse-javadoc:%E2%98%82=MapReduce-WordCount/C:%5C/Users%5C/yang%5C/Desktop%5C/hadoop-1.2.1%5C/hadoop-core-1.2.1.jar%3Corg.apache.hadoop.mapreduce(Mapper.class%E2%98%83Mapper%E2%98%82%E2%98%82cleanup%E2%98%82Context) is called.

All intermediate values associated with a given output key are subsequently grouped by the framework, and passed to a [Reducer](eclipse-javadoc:%E2%98%82=MapReduce-WordCount/C:%5C/Users%5C/yang%5C/Desktop%5C/hadoop-1.2.1%5C/hadoop-core-1.2.1.jar%3Corg.apache.hadoop.mapreduce(Mapper.class%E2%98%83Mapper%E2%98%82Reducer) to determine the final output. Users can control the sorting and grouping by specifying two key [RawComparator](eclipse-javadoc:%E2%98%82=MapReduce-WordCount/C:%5C/Users%5C/yang%5C/Desktop%5C/hadoop-1.2.1%5C/hadoop-core-1.2.1.jar%3Corg.apache.hadoop.mapreduce(Mapper.class%E2%98%83Mapper%E2%98%82RawComparator) classes.

The Mapper outputs are partitioned per Reducer. Users can control which keys (and hence records) go to which Reducer by implementing a custom [Partitioner](eclipse-javadoc:%E2%98%82=MapReduce-WordCount/C:%5C/Users%5C/yang%5C/Desktop%5C/hadoop-1.2.1%5C/hadoop-core-1.2.1.jar%3Corg.apache.hadoop.mapreduce(Mapper.class%E2%98%83Mapper%E2%98%82Partitioner).

Users can optionally specify a combiner, via [Job.setCombinerClass(Class)](eclipse-javadoc:%E2%98%82=MapReduce-WordCount/C:%5C/Users%5C/yang%5C/Desktop%5C/hadoop-1.2.1%5C/hadoop-core-1.2.1.jar%3Corg.apache.hadoop.mapreduce(Mapper.class%E2%98%83Mapper%E2%98%82Job%E2%98%82setCombinerClass%E2%98%82Class), to perform local aggregation of the intermediate outputs, which helps to cut down the amount of data transferred from the Mapper to the Reducer.

Applications can specify if and how the intermediate outputs are to be compressed and which [CompressionCodec](eclipse-javadoc:%E2%98%82=MapReduce-WordCount/C:%5C/Users%5C/yang%5C/Desktop%5C/hadoop-1.2.1%5C/hadoop-core-1.2.1.jar%3Corg.apache.hadoop.mapreduce(Mapper.class%E2%98%83Mapper%E2%98%82CompressionCodec)s are to be used via the Configuration.

072课时 Reduce 基类的讲解

Reducer 源代码和Mapper差不多，查看Reducer中的API文档

该类的功能：获取map()方法输出的中间结果；将中间结果中的value按照key或分为group，而group按照key排序，形成<key, collection values>的结构，此时key是唯一的；处理group中所有的value，相同的key的value组合，最终形成key对应的value唯一。

过程：

1. shuffle
2. Sort & Secondary Sort
3. Reduce过程

073 74课时 Mapper&Reducer中的Context内部类的讲解

Mapper & Reducer类中都有一个Context内部类,分别继承MapContext & ReduceContext类。

同时通过学习Mapper & Reducer类，我们了解到，我们还可以对于执行map&reduce函数之前的setup()和执行之后的cleanup()进行覆写，以实现我们自己定义的功能。

075 76 77课时 最值讲解值某个文件中的最值和TopKey值

1. 某个文件中某列数据的最值：

对每一列的数据值进行依次比较，保存最大的值进行输出，算法思想类似于排序（快速或者是冒泡排序）

1. 某个文件中某列数据的Top Key值 V2 V3

使用TreeSet保存前几个值

1. 文件中某列数据的Top Key值（多个文件中） V4

按照传统方法进行比较，在进行reduce方法进行处理，会变得十分的复杂，1&2传统的方式进行reduce就会有问题。

078课时 单词统计和TopKey值应用的讲解

数据格式：

语言类别 **歌曲名称** 收藏次数 **播放次数** 歌手名称

需求：

统计播放次数前10的歌曲名称和播放次数

使用我们上面的程序既可以实现这个功能

079 80 81课时 MapReduce最小驱动配置 跟踪源代码查看MapReduce默认配置

读取输入文件中的内容，输出到指定的目录下的，此时文件中的内容key就是原输入文件的起始位置，value就是输入文件中的每一行内容。

|  |
| --- |
| **package** com.yang.hadoop.mr.min;  **import** java.io.IOException;  **import** org.apache.hadoop.conf.Configuration;  **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.input.TextInputFormat;  **import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  **import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  **import** org.apache.hadoop.mapreduce.lib.partition.HashPartitioner;  **public** **class** DefaultMapReduce {    **public** **static** **void** main(String[] args)  **throws** IOException, ClassNotFoundException, InterruptedException{  args = **new** String[] {  "hdfs://hadoop-master:9000/opt/data/topk/input",  "hdfs://hadoop-master:9000/opt/data/topk/output" };  //Step 1 get conf  Configuration conf = **new** Configuration();    //Step 2 create job  Job job = **new** Job(conf, DefaultMapReduce.**class**.getSimpleName());    //Step 3 set job    //1. set run jar class  job.setJarByClass(DefaultMapReduce.**class**);    //2. set inputfromat  job.setInputFormatClass(TextInputFormat.**class**);    //3. set input path  FileInputFormat.*addInputPath*(job, **new** Path(args[0]));    //4. set mapper  job.setMapperClass(Mapper.**class**);    //5. set map output key value class  job.setMapOutputKeyClass(LongWritable.**class**);  job.setMapOutputValueClass(Text.**class**);    //6. set partitioner class  job.setPartitionerClass(HashPartitioner.**class**);    //7. set reduce number  job.setNumReduceTasks(1);    //8. set sort comparator class  job.setSortComparatorClass(LongWritable.Comparator.**class**);    //9. set group comparator class  job.setGroupingComparatorClass(LongWritable.Comparator.**class**);    //10. set combiner class  // job.setCombinerClass(null);    //11.set reduce class  job.setReducerClass(Reducer.**class**);    //12. set output Format  job.setOutputFormatClass(TextOutputFormat.**class**);    //13. set job output key value class  job.setOutputKeyClass(LongWritable.**class**);  job.setOutputValueClass(Text.**class**);    //14. set job output path  FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));    //setp 4 submit job  **boolean** isSuccess = job.waitForCompletion(**true**);    //step 5 exit program  System.*exit*(isSuccess?0:1);  }  } |

082课时 MapReduce模板类的编写

|  |
| --- |
| **package** com.yang.hadoop.mr.module;  **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.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.input.TextInputFormat;  **import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  **import** org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  **import** org.apache.hadoop.mapreduce.lib.partition.HashPartitioner;  **import** org.apache.hadoop.util.Tool;  **import** org.apache.hadoop.util.ToolRunner;  **public** **class** ModuleMapReduce **extends** Configured **implements** Tool {  /\*  \* Mapper class  \*/  **public** **static** **class** ModuleMapper **extends**  Mapper<LongWritable, Text, LongWritable, Text> {  @Override  **protected** **void** cleanup(Context context) **throws** IOException,  InterruptedException {  // **TODO** Auto-generated method stub  **super**.cleanup(context);  }  @Override  **protected** **void** map(LongWritable key, Text value, Context context)  **throws** IOException, InterruptedException {  // **TODO** Auto-generated method stub  **super**.map(key, value, context);  }  @Override  **protected** **void** setup(Context context) **throws** IOException,  InterruptedException {  // **TODO** Auto-generated method stub  **super**.setup(context);  }  }  /\*  \* Reducer class  \*/  **public** **static** **class** ModuleReducer **extends**  Reducer<LongWritable, Text, LongWritable, Text> {  @Override  **protected** **void** cleanup(Context context) **throws** IOException,  InterruptedException {  // **TODO** Auto-generated method stub  **super**.cleanup(context);  }  @Override  **protected** **void** reduce(LongWritable key, Iterable<Text> values,  Context context) **throws** IOException, InterruptedException {  // **TODO** Auto-generated method stub  **super**.reduce(key, values, context);  }  @Override  **protected** **void** setup(Context context) **throws** IOException,  InterruptedException {  // **TODO** Auto-generated method stub  **super**.setup(context);  }  }  /\*  \* Drive code  \*/  **public** Job parseInputAndOutput(Tool tool, Configuration conf, String[] args)  **throws** IOException {  // validate  **if** (args.length != 2) {  System.*err*.printf(  "Useage: %s [generic options] <input> <output>\n", tool  .getClass().getSimpleName());  ToolRunner.*printGenericCommandUsage*(System.*err*);  **return** **null**;  }  // Step 2 create job  Job job = **new** Job(conf, tool.getClass().getSimpleName());  // 3. set input path  FileInputFormat.*addInputPath*(job, **new** Path(args[0]));  // 14. set job output path  FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  **return** job;  }  **public** **int** run(String[] args) **throws** Exception {  // Step 1 get conf  Configuration conf = **new** Configuration();  Job job = parseInputAndOutput(**this**, conf, args);  **if** (**null** == job) {  **return** -1;  }  // 1. set run jar class  job.setJarByClass(ModuleMapReduce.**class**);  // 2. set inputfromat  job.setInputFormatClass(TextInputFormat.**class**);  // 4. set mapper  job.setMapperClass(ModuleMapper.**class**);  // 5. set map output key value class  job.setMapOutputKeyClass(LongWritable.**class**);  job.setMapOutputValueClass(Text.**class**);  // 6. set partitioner class  job.setPartitionerClass(HashPartitioner.**class**);  // 7. set reduce number  job.setNumReduceTasks(1);  // 8. set sort comparator class  // job.setSortComparatorClass(LongWritable.Comparator.class);  // 9. set group comparator class  // job.setGroupingComparatorClass(LongWritable.Comparator.class);  // 10. set combiner class  // job.setCombinerClass(null);  // 11.set reduce class  job.setReducerClass(ModuleReducer.**class**);  // 12. set output Format  job.setOutputFormatClass(TextOutputFormat.**class**);  // 13. set job output key value class  job.setOutputKeyClass(LongWritable.**class**);  job.setOutputValueClass(Text.**class**);  // 14. set job output path  FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));  // setp 4 submit job  **boolean** isSuccess = job.waitForCompletion(**true**);  **return** isSuccess ? 0 : 1;  }  **public** **static** **void** main(String[] args) **throws** Exception {  args = **new** String[] {  "hdfs://hadoop-master:9000/opt/data/topk/input",  "hdfs://hadoop-master:9000/opt/data/topk/output" };  **int** status = ToolRunner.*run*(**new** ModuleMapReduce(), args);  System.*exit*(status);  }  } |

083课时 使用模板方法编写Wordcount 程序

1）该名称

MapReduce Mapper Reduce 类的名称

2）依据实际业务类的需要，修改输入输出类型

3）修改驱动Driver部分的Job参数的设置：Map Reduce 输出类型的修改

4）在Mapper Reducer编写实际的业务逻辑（setup map cleanup）

5）检查模板中的run方法中的参数设置

6）设置输入输出路径，进行MR的测试

084课时 复习MapReduce最小驱动和默认驱动以及MapReduce模板类的编写

略

085课时 MapReduce Job运行打印信息讲解和计数器讲解

运行程序统计日志信息学习

|  |
| --- |
| [hadoop@hadoop-master hadoop]$ hadoop jar hadoop-examples-1.2.1.jar wordcount /opt/data/wc/input /opt/data/wc/output13  #告知输入路径下的文件数目需要进行处理  16/04/24 22:05:01 INFO input.FileInputFormat: Total input paths to process : 7  #hadoop中的本地库￥HADOOP\_HOME/c++/Linux-amd64/lib，加载hadoop本地库  16/04/24 22:05:01 INFO util.NativeCodeLoader: Loaded the native-hadoop library  # 加载本地的snappy压缩算法的库，默认情况下是没有对应的snappy库，需要用户自行配置  16/04/24 22:05:01 WARN snappy.LoadSnappy: Snappy native library not loaded  #运行Job的相关进度信息  #运行Job的名称ID  16/04/24 22:05:02 INFO mapred.JobClient: Running job: job\_201604242159\_0001  #Job运行时 MapTask & ReduceTask运行的进度  16/04/24 22:05:03 INFO mapred.JobClient: map 0% reduce 0%  16/04/24 22:05:08 INFO mapred.JobClient: map 28% reduce 0%  16/04/24 22:05:12 INFO mapred.JobClient: map 57% reduce 0%  16/04/24 22:05:14 INFO mapred.JobClient: map 85% reduce 0%  16/04/24 22:05:15 INFO mapred.JobClient: map 100% reduce 0%  16/04/24 22:05:17 INFO mapred.JobClient: map 100% reduce 33%  16/04/24 22:05:19 INFO mapred.JobClient: map 100% reduce 100%  #job运行完成  16/04/24 22:05:19 INFO mapred.JobClient: Job complete: job\_201604242159\_0001  #显示Job整个运行过程中，计数器Counter的值  #总共有29种计数器，五大类  16/04/24 22:05:19 INFO mapred.JobClient: Counters: 29  16/04/24 22:05:19 INFO mapred.JobClient: **Map-Reduce Framework** #计数器个数  16/04/24 22:05:19 INFO mapred.JobClient: Spilled Records=1200  16/04/24 22:05:19 INFO mapred.JobClient: Map output materialized bytes=10715  16/04/24 22:05:19 INFO mapred.JobClient: Reduce input records=600  16/04/24 22:05:19 INFO mapred.JobClient: Virtual memory (bytes) snapshot=15698837504  16/04/24 22:05:19 INFO mapred.JobClient: Map input records=392  16/04/24 22:05:19 INFO mapred.JobClient: SPLIT\_RAW\_BYTES=882  16/04/24 22:05:19 INFO mapred.JobClient: Map output bytes=21400  16/04/24 22:05:19 INFO mapred.JobClient: Reduce shuffle bytes=10715  16/04/24 22:05:19 INFO mapred.JobClient: Physical memory (bytes) snapshot=1385676800  16/04/24 22:05:19 INFO mapred.JobClient: Reduce input groups=429  16/04/24 22:05:19 INFO mapred.JobClient: Combine output records=600  16/04/24 22:05:19 INFO mapred.JobClient: Reduce output records=429  16/04/24 22:05:19 INFO mapred.JobClient: Map output records=1765  16/04/24 22:05:19 INFO mapred.JobClient: Combine input records=1765  16/04/24 22:05:19 INFO mapred.JobClient: CPU time spent (ms)=3330  16/04/24 22:05:19 INFO mapred.JobClient: Total committed heap usage (bytes)=1095237632  16/04/24 22:05:19 INFO mapred.JobClient:  **File Input Format Counters**  16/04/24 22:05:19 INFO mapred.JobClient: Bytes Read=15518  16/04/24 22:05:19 INFO mapred.JobClient: **FileSystemCounters**  16/04/24 22:05:19 INFO mapred.JobClient: HDFS\_BYTES\_READ=16400  16/04/24 22:05:19 INFO mapred.JobClient: FILE\_BYTES\_WRITTEN=461196  16/04/24 22:05:19 INFO mapred.JobClient: FILE\_BYTES\_READ=10679  16/04/24 22:05:19 INFO mapred.JobClient: HDFS\_BYTES\_WRITTEN=6620  16/04/24 22:05:19 INFO mapred.JobClient: **Job Counters**  16/04/24 22:05:19 INFO mapred.JobClient: Launched map tasks=7  16/04/24 22:05:19 INFO mapred.JobClient: Launched reduce tasks=1  16/04/24 22:05:19 INFO mapred.JobClient: SLOTS\_MILLIS\_REDUCES=10332  16/04/24 22:05:19 INFO mapred.JobClient: Total time spent by all reduces waiting after reserving slots (ms)=0  16/04/24 22:05:19 INFO mapred.JobClient: SLOTS\_MILLIS\_MAPS=19470  16/04/24 22:05:19 INFO mapred.JobClient: Total time spent by all maps waiting after reserving slots (ms)=0  16/04/24 22:05:19 INFO mapred.JobClient: Data-local map tasks=7  16/04/24 22:05:19 INFO mapred.JobClient:  **File Output Format Counters**  16/04/24 22:05:19 INFO mapred.JobClient: Bytes Written=6620 |

计数器种类:提供一个窗口，用于观察程序运行过程中的，分类如下：

文件系统的计数器：FileSystemCounters

HDFS\_BYTES\_READ: map() read data

FILE\_BYTES\_WRITE: map() write data

FILE\_BYTES\_READ: reduce() read data

HDFS\_BYTES\_WRITE:reduce() write data

输入格式化计数器：File Input Format

输出格式化计数器：File Output Format

086课时MapReduce Framework和Job Counter计数器(视频不可用暂时不学习了)

Job Counters

Map-Reduce Framework

087课时 自定义计数器

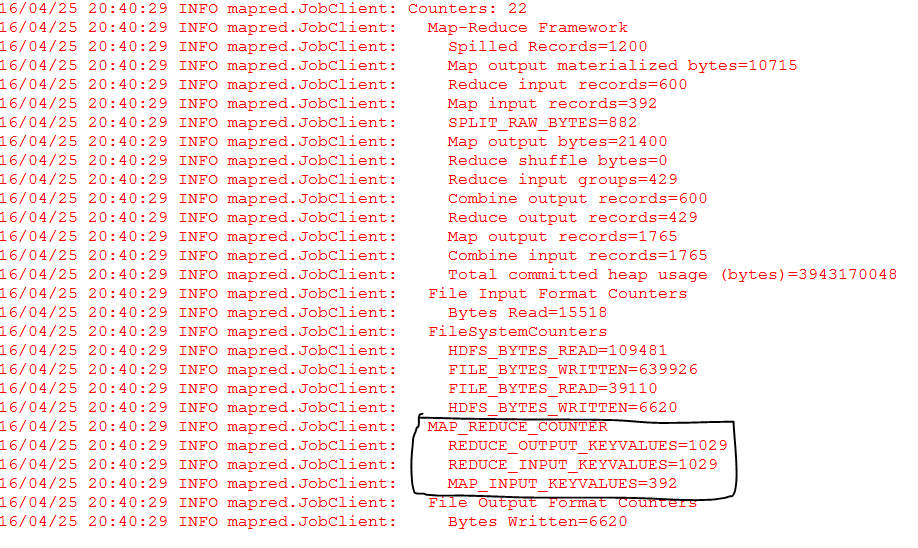
在map & reduce中使用*org.apache.hadoop.mapreduce.Reducer;*的计数器，使用

Counter counter = context.getCounter("MAP\_REDUCE\_COUNTER", "MAP\_INPUT\_KEYVALUES");

获取自定义的MAP\_REDUCE\_COUNTER组下面的MAP\_INPUT\_KEYVALUES名称的计数器，每执行一次加一，

在reduce函数中同样可以定义REUDCE\_INPUT\_KEY\_VALUES名称的计数器，在MAP\_REDUCE\_COUNTER组下面。

运行程序之后，结果如下：



088课时 自定义计数器值使用枚举方式

1.使用context.getCounter(String groupName, String counterName); 方式



2.使用context.getCounter(enum);

在程序类中

public static enum Counter{ MAP\_INPUT\_KEYVALUES,MAP\_OUTPUT\_KEYVALUES,REDUCE\_INPUT\_KEYVALUES,REDUCE\_OUTPUT\_KEYVALUES

}

然后使用上面的API方式进行计算counter

但是我们在Eclipse中没有Job的计数器，因为我们的程序是本地的程序，而不是运行在JobTask上面的程序，没有JobCounter的数据。

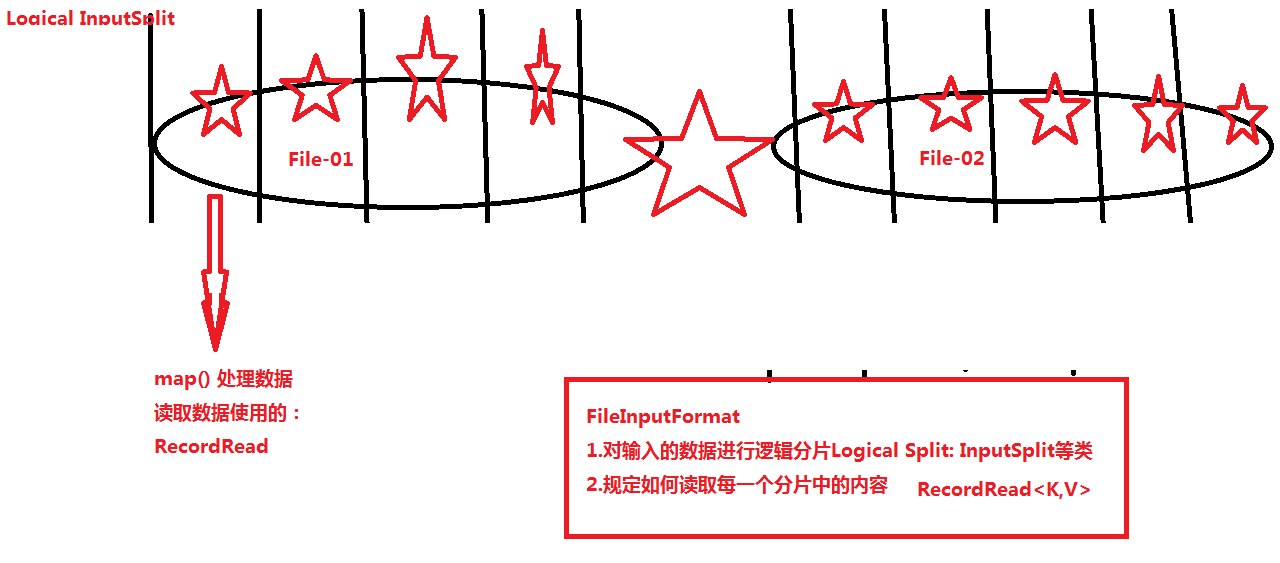
089课时 源码分析InputFormat类

关于Job的14个设置讲解

1. 设置JAR CLASS 不讲解
2. 设置setInputFormatClass

InputFormat describes the input-specification for a Map-Reduce job

1. Validate the input-specification of the job
2. Split-up the input files into logical inputsplits, each of which is then assigned to an individual Mapper
3. Provide the RecordReader implementation to be used to glean input records from the logical InputSplit for processing by the mapper



090课时 MapReduce数据输入格式InputFormat类

针对上面的InputFormat的复习

至此Hadoop1.x 两大块知识 HDFS & MapReduce 初步学习完成