

# 1.1 MapReduce思想

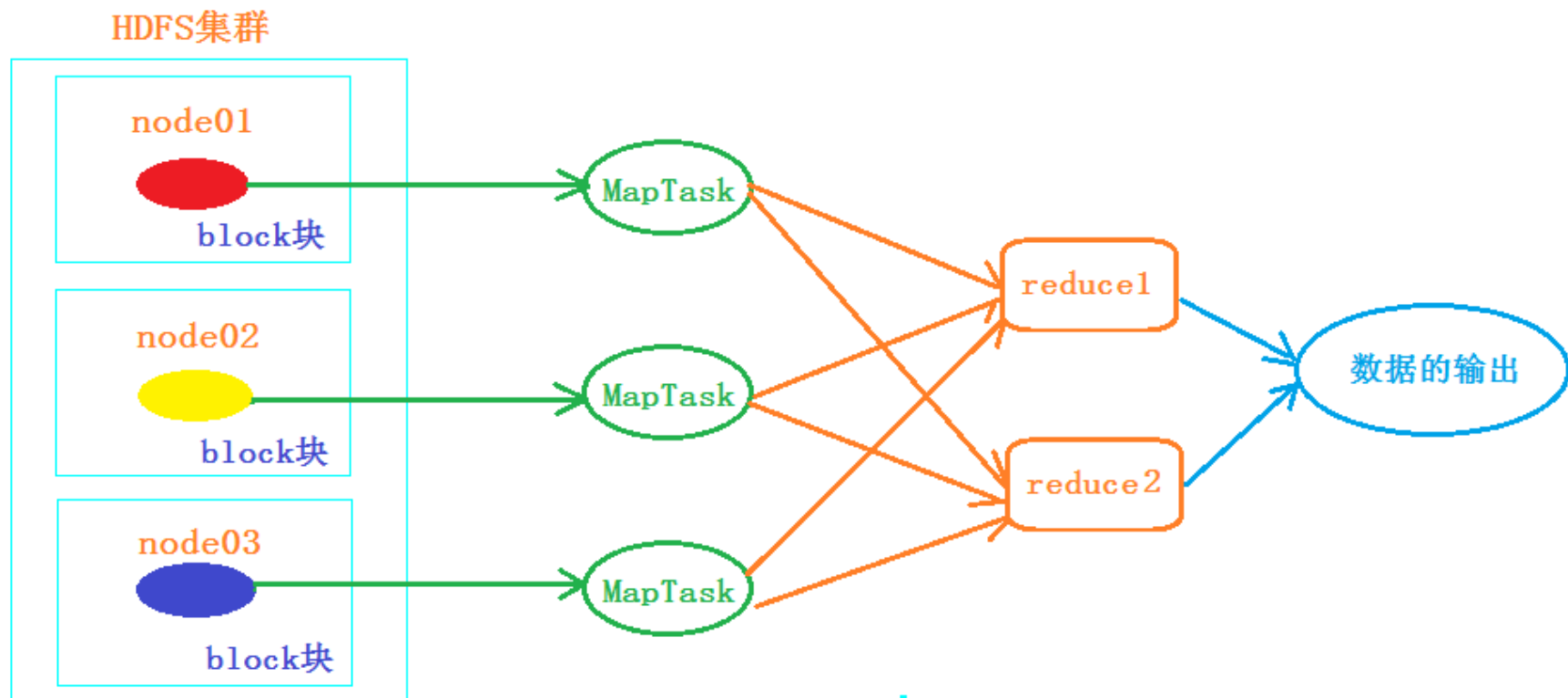
MapReduce在Hadoop 1.x中是一个分布式的计算框架,在Hadoop 2.x中也保留了这种分布式的计算框架,只不过这种计算框架运行在yarn集群上面。

Hadoop的MapReduce

MapReduce是一种思想,就是一个分治的思想

- ▶ 主要有两个阶段:
- ▶ Map阶段:Map阶段主要负责"分",分成若干个简单的任务,并行的计算,彼此间几乎没有依赖关系。(你也能干,我也能干,你不依赖我,我不依赖你)
- ▶ Reduce阶段:Reduce阶段主要负责"合",先分在合,合呢就是把Map处理完的结果汇总到一起,形成一个最终的结果,这个就是MapReduce
- ▶ 的核心思想;

## MapReduce的核心思想



# 1.2 Hadoop MapReduce 设计构思

- ▶ 第一个思想：分治；
- ▶ 第二个思想：简单的抽象模型Map和Reduce,一个负责分,一个负责合,先分后合；
- ▶ 第三个思想：隐藏系统细节,就是写的时候,只要重点关注Map和Reduce怎么做,如果其他的一些设置你都可以不用,只关注MapReduce就行了；
- ▶ 这个Map和Reduce也是最重要的两个步骤。

# 1.4 MapReduce编程模型-WordCount实例分析

- ▶ MapReduce编程的8个步骤：
- ▶ Map阶段2个步骤：
  - ▶ 1.第一步：读取文件,解析成key,value对儿；
  - ▶ 这里的key,value对 指代的是k1,v1。
  - ▶ 2.第二步：自定义map逻辑,接收第一步读取的k1,v1,转换成新的k2,v2输出。

- ▶ shuffle阶段4个步骤：
- ▶ 3.第三步：分区(相同key的数据发送到同一个reduce里面去,形成一个集合)。
- ▶ 4.第四步:排序 对我们的数据进行字典顺序的排列
- ▶ 5.第五步:规约 主要是在Map端对数据进行一次聚合,减少我们输出的k2的数据量。
- ▶ 6.第六步:分组 将相同的数据发送到同一组里面去,调用一次reduce逻辑。

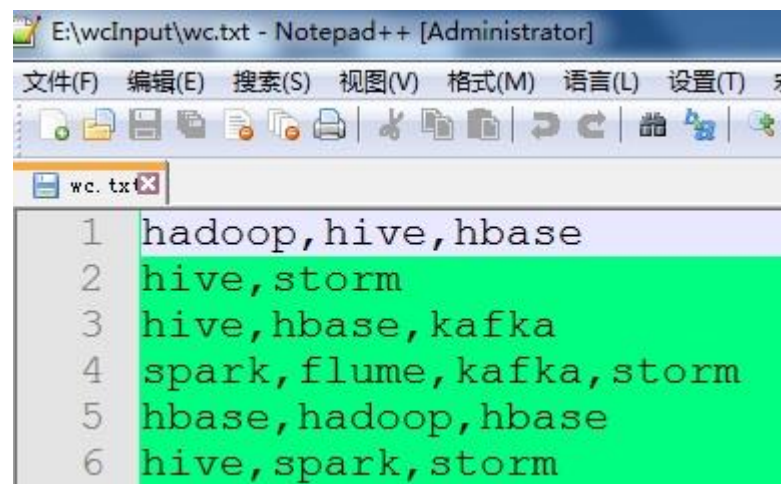
- ▶ reduce阶段的2个步骤:
- ▶ 7.第七步:自定义reduce逻辑
- ▶ 接收k2,v2转换成新的k3,v3 进行输出。
- ▶ 8.第八部:将我们reduce处理完成之后的数据进行输出。

- ▶ 这里面有3个key,value对:
- ▶ 哪里形成k1,v1,哪里形成k2,v2,哪里形成k3,v3。
- ▶ 每一个步骤都是一个class类,将8个步骤的class类组织到一起就是我们的mapreduce程序。



# 1.4 MapReduce编程模型- WordCount实例分析

- ▶ 做一个单词计数的统计,一个个单词用逗号分割的。
- ▶ 我们该怎么做?



## 1.4 MapReduce编程模型- WordCount实例分析

- ▶ 第一步:读取文件,解析成key,value对
- ▶ 读取文件用哪个类呢?

## 1.4 MapReduce编程模型- WordCount实例分析

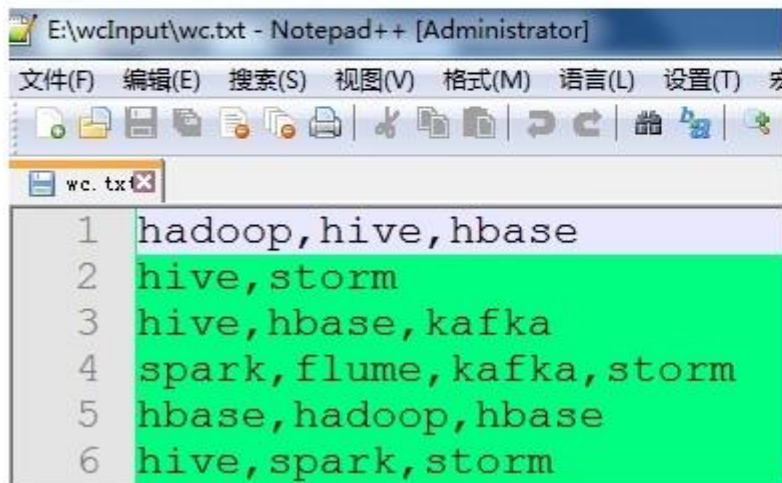
- ▶ 第一步:读取文件,解析成key,value对
- ▶ 读取文件用哪个类呢?

# 1.4 MapReduce编程模型- WordCount实例分析

hadoop的mapreduce的wordcount示例：

读取文件

TextInputFormat解析成一个key, value对  
key是我们这一行的行文本内容的偏移量  
value是我们这个文件的行内容



```
E:\wcInput\wc.txt - Notepad++ [Administrator]
文件(F) 编辑(E) 搜索(S) 视图(V) 格式(M) 语言(L) 设置(T)
wc. txt
1 hadoop, hive, hbase
2 hive, storm
3 hive, hbase, kafka
4 spark, flume, kafka, storm
5 hbase, hadoop, hbase
6 hive, spark, storm
```

第一步：

# 1.4 MapReduce编程模型- WordCount实例分析

- ▶ 行偏移量究竟是什么?

key1	value1
0	hadoop,hive,hbase
17	hive,storm
27	hive,hbase,kafka

# 1.4 MapReduce编程模型- WordCount实例分析

hadoop的mapreduce的wordcount示例:

读取文件

TextInputFormat解析成一个key, value对  
key是我们这一行的行文本内容的偏移量  
value是我们这个文件的行内容

map逻辑

接收key1, value1 转换成新的key2, value2输出  
那这里究竟该怎么转换呢?

第一步:

第二步:





# 1.4 MapReduce编程模型- WordCount实例分析

hadoop的mapreduce的wordcount示例:

读取文件

TextInputFormat解析成一个key, value对  
key是我们这一行的行文本内容的偏移量  
value是我们这个文件的行内容

map逻辑

接收key1, value1 转换成新的key2, value2输出  
那这里究竟该怎么转换呢?

第一步:

key1	value1
0	hadoop, hive, hbase
17	hive, storm
27	hive, hbase, kafka

第二步:

把这一行的文本内容进行切割, 按照逗号进行切开

# 1.4 MapReduce编程模型- WordCount实例分析

hadoop的mapreduce的wordcount示例:

读取文件

TextInputFormat解析成一个key, value对  
key是我们这一行的行文本内容的偏移量  
value是我们这个文件的行内容

map逻辑

接收key1, value1 转换成新的key2, value2输出  
那这里究竟该怎么转换呢?

第一步:

第二步:

key1	value1
0	hadoop,hive,hbase
17	hive,storm
27	hive,hbase,kafka

key2	value2
hadoop	1
hive	1
hbase	1
hive	1
storm	1
hive	1
hbase	1
kafka	1

把这一行的文本内容进行切割, 按照逗号进行切开



# 1.4 MapReduce编程模型- WordCount实例分析

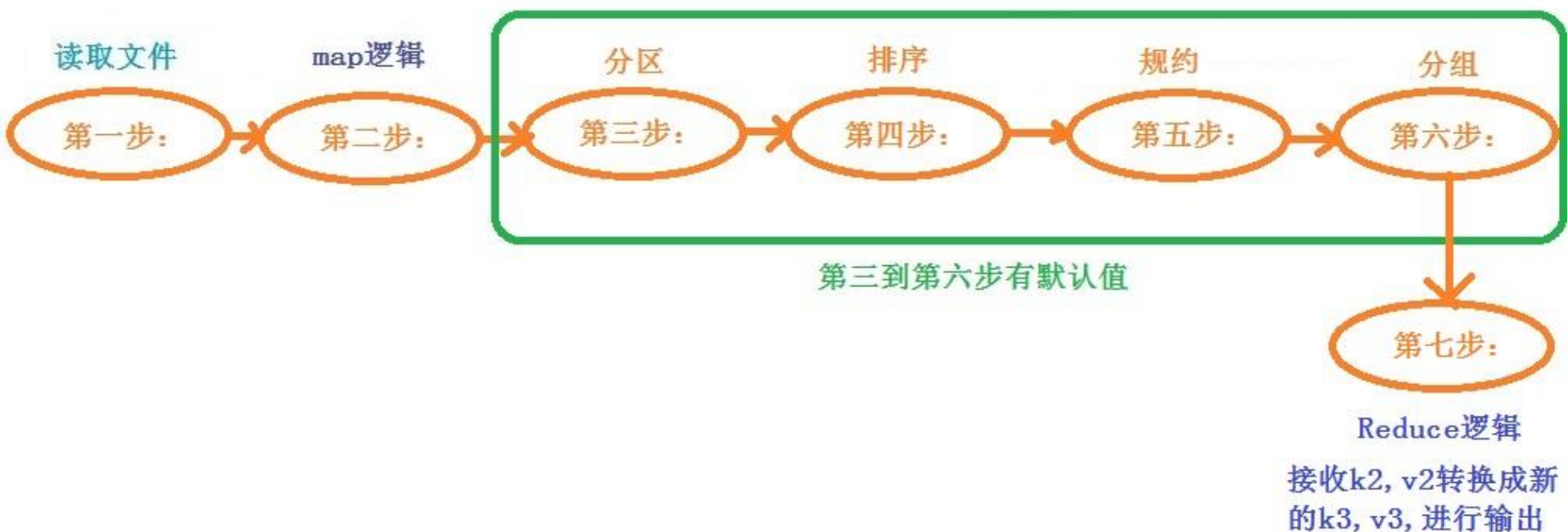
hadoop的mapreduce的wordcount示例：



第三到第六步有默认值

# 1.4 MapReduce编程模型- WordCount实例分析

hadoop的mapreduce的wordcount示例:



# 1.4 MapReduce编程模型- WordCount实例分析

- ▶ 第三步分区里面有一句很重要的话?

## 1.4 MapReduce编程模型- WordCount实例分析

- ▶ 相同key的数据发送到同一个reduce里面去,形成一个集合。

## 1.4 MapReduce编程模型- WordCount实例分析

- ▶ 问这个key指的是key1,还是key2?

## 1.4 MapReduce编程模型- WordCount实例分析

key2	value2
hadoop	1
hive	1
hbase	1
hive	1
storm	1
hive	1
hbase	1
kafka	1

# 1.4 MapReduce编程模型- WordCount实例分析

key2	value2
hadoop	1
hive	1
hbase	1
hive	1
storm	1
hive	1
hbase	1
kafka	1

key2	value2
hadoop	<1>
hive	<1,1,1>
hbase	<1,1>
storm	<1>
kafka	<1>



# 1.4 MapReduce编程模型- WordCount实例分析

hadoop的mapreduce的wordcount示例:





# 1.4 MapReduce编程模型- WordCount实例分析

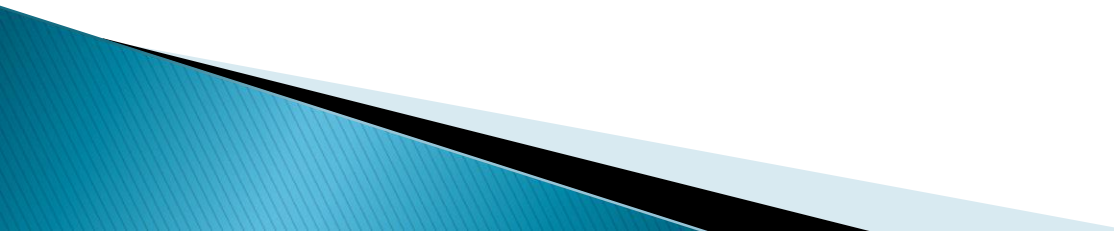
key2	value2
hadoop	<1>
hive	<1,1,1>
hbase	<1,1>
storm	<1>
kafka	<1>

key3	value3
hadoop	1
hive	3
hbase	2
storm	1
kafka	1

## 1.4 MapReduce编程模型- WordCount实例分析

- ▶ 接下来我们看总结一下怎么实现的：

# 1.4 MapReduce编程模型- WordCount实例分析

- ▶ 接下来我们看总结一下怎么实现的：
  - ▶ map阶段2个步骤；
  - ▶ shuffle阶段4个步骤；
  - ▶ reduce阶段的2个步骤；
- 

## 1.4 MapReduce编程模型- WordCount实例分析

- ▶ 接下来我们看总结一下怎么实现的：
- ▶ 第三步：分区；
- ▶ 相同key的value发送到同一个reduce里面去,形成一个集合；

## 1.4 MapReduce编程模型- WordCount实例分析

- ▶ 能不能理解reduce里面为何是这样的数据,就理解这句话,就是他决定了,我们输入到reduce里面
- ▶ 形成这样的结果:

key2	value2
hadoop	<1>
hive	<1,1,1>
hbase	<1,1>
storm	<1>
kafka	<1>

# 1.4 MapReduce编程模型- WordCount实例分析

► 原先是：

key2	value2
hadoop	1
hive	1
hbase	1
hive	1
storm	1
hive	1
hbase	1
kafka	1

现在是：

key2	value2
hadoop	<1>
hive	<1,1,1>
hbase	<1,1>
storm	<1>
kafka	<1>

# 1.4 MapReduce编程模型- WordCount实例分析

▶ 原先是：

key2	value2
hadoop	1
hive	1
hbase	1
hive	1
storm	1
hive	1
hbase	1
kafka	1

现在是：

key2	value2
hadoop	<1>
hive	<1,1,1>
hbase	<1,1>
storm	<1>
kafka	<1>

▶ 相同的key都合并了，原先8个key,现在5个key。

## 1.4 MapReduce编程模型- WordCount实例分析

- ▶ reduce阶段的2个步骤
- ▶ 第七步：对多个map的任务进行合并，排序，写reduce函数自己的逻辑，
- ▶ 对输入的key，value对进行处理，转换成新的key，value对进行输出
- ▶ 第八步：设置outputformat将输出的key，value对数据进行保存到文件中



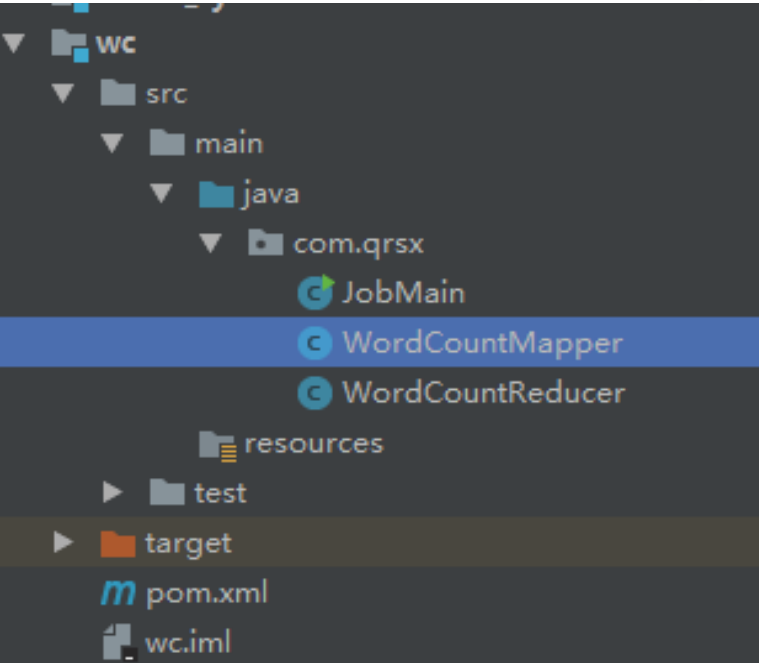
# 1.4 MapReduce编程模型- WordCount实例分析

- ▶ 接下来就看一下wordcount究竟该怎么写？

## 1.4 MapReduce编程模型- WordCount实例分析

- ▶ 接下来就看一下wordcount究竟该怎么写？
- ▶ 定义了map类；
- ▶ 定义了reduce类；
- ▶ 然后组装一下,然后让他提交；

# 1.4 MapReduce编程模型- WordCount实例分析



Search Everywhere Double Shift

Go to File Ctrl+Shift+N

Recent Files Ctrl+E

Navigation Bar Alt+Home

Drop files here to open

## 1.4 MapReduce编程模型- WordCount实例分析

```
+ import ...
```

//程序入口, 首先继承Configured, 再实现Tool

```
public class JobMain extends Configured implements Tool {
```

```
+  public int run(String[] args) throws Exception {...}
```

```
+ public static void main(String[] args) throws Exception {...}  
}
```

# 1.4 MapReduce编程模型- WordCount实例分析

```
public class JobMain extends Configured implements Tool {
```



```
public int run(String[] args) throws Exception {...}
```

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

```
    Configuration configuration = new Configuration();
```

```
    //提交job任务, mapreduce运行完成后, 程序返回一个值, 一个状态码值, 0成功
```

```
    //第二个参数是个tool, 当前这个类JobMain就实现了Tool接口
```

```
    int run = ToolRunner.run(configuration, new JobMain(), args);
```

```
    System.exit(run);
```

```
}
```

```
}
```

```
public int run(String[] args) throws Exception {  
    //获取一个job对象,用于任务组织,通过job对象将八个步骤组织到一起,提交给ya  
    Job job = Job.getInstance(super.getConf(), "xxx");  
    //第一步,读取文件解析成key,value对儿  
    job.setInputFormatClass(TextInputFormat.class);  
    //路径  
    TextInputFormat.addInputPath(job, new Path("file:///E:\\wcInput"));  
    //第二部,自定义map逻辑,接收k1,v1,转化为k2,v2  
    job.setMapperClass(WordCountMapper.class);  
    //设置k2类型  
    job.setMapOutputKeyClass(Text.class);  
    job.setMapOutputValueClass(IntWritable.class);  
    job.setReducerClass(WordCountReducer.class);  
    job.setOutputKeyClass(Text.class);  
    job.setOutputValueClass(IntWritable.class);  
    TextOutputFormat.setOutputPath(job, new Path("file:///E:\\wcOutput"));  
    //提交任务  
    boolean b = job.waitForCompletion(true);  
    return b?0:1;  
}
```

# 1.4 MapReduce编程模型-WordCount实例分析

//定义4个泛型

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

```
    Text text = new Text();
```

```
    IntWritable intWritable = new IntWritable();
```

```
    //覆写父类的map方法, 处理逻辑都写在map方法里面
```

```
    //map方法反复会被调用, 每一行都会调
```

```
    /*
```

💡 \* key k1 行偏移量, 没用, 丢掉

```
    * value v1 行文本内容, 需切割, 转成新的k2, v2输出
```

```
    * context 上下文对象, 承接上文, 将数据发给下文, 桥梁
```

```
    */
```

```
    @Override
```

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

```
        // 1. 切割文本内容, 生成新的k2, v2
```

# 1.4 MapReduce编程模型- WordCount实例分析

@Override

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

```
    String[] split = value.toString().split(",");
```

```
    //遍历切割出的单词
```

```
    //往下发,全部用到context
```

```
    for (String word : split) {
```

```
        text.set(word);
```

```
        intWritable.set(1);
```

```
        //写出去的k2, v2, 这里类型与k2, v2保持一致
```

```
        context.write(text, intWritable);
```

```
    }
```

```
}
```



# 1.4 MapReduce编程模型- WordCount实例分析

```
import ...
```

```
//定义4个泛型
```

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

```
    Text text = new Text();
```

```
    IntWritable intWritable = new IntWritable();
```

```
@Override
```

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

```
    String[] split = value.toString().split(",");
```

```
    for (String word : split) {
```

```
        text.set(word);
```

```
        intWritable.set(1);
```

```
        //写出去的k2, v2, 这里类型与k2, v2保持一致
```

```
        context.write(text, intWritable);
```

```
    }
```

```
}
```

```
}
```

# 1.4 MapReduce编程模型- WordCount实例分析

```
import ...  
  
/*  
 * 自定义class类  
 * k2, v2, k3, v3  
 * */  
public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable>{  
  
    /**  
     * 覆写reduce方法,  
     * @param key 接收的key 是我们的K2  
     * @param values 接收到value是一个集合 集合里面的数据类型是 v2 类型  
     * @param context 上下文对象, 将我们的数据往外写  
     * @throws IOException  
     * @throws InterruptedException  
     */  
}
```

# 1.4 MapReduce编程模型- WordCount实例分析

```
@Override
protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
    int a = 0;
    for (IntWritable value : values) {
        int i = value.get();
        a += i;
    }
    //将我们的数据写出去
    context.write(key, new IntWritable(a));
}
```

# 1.4 MapReduce编程模型- WordCount实例分析

```
public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable>{  
    @Override  
    protected void reduce(Text k2, Iterable<IntWritable> v2, Context context) throws IOException, InterruptedException {  
        int a = 0;  
        for (IntWritable value : v2) {  
            int i = value.get();  
            a += i;  
        }  
        //将我们的数据写出去  
        context.write(key, new IntWritable(a));  
    }  
}
```

JobMain.java × WordCountMapper.java × WordCountReducer.java ×

```
35     job.setReducerClass(WordCountReducer.class);
36     job.setOutputKeyClass(Text.class);
37     job.setOutputValueClass(IntWritable.class);
38     job.setOutputFormatClass(TextOutputFormat.class);
39     TextOutputFormat.setOutputPath(job, new Path("file:///E:\\wcOutput"));
40     //提交任务
41     boolean b = job.waitForCompletion(true);
42     return b?0:1;
```

}

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

Run 'JobMain.main()' Ctrl+Shift+F10

Debug 'JobMain.main()'

Run 'JobMain.main()' with Coverage

```
46         configuration = new Configuration();
47         //reduce运行完成后,程序返回一个值,一个状态码值,0成功
```

```
48         //第二个参数是个tool,当前这个类JobMain就实现了Tool接口
```

```
49         int run = ToolRunner.run(configuration, new JobMain(), args);
```

```
50         System.exit(run);
```

}

}