

MapReduce并行编程模型

一、课前准备

1. 准备3节点hadoop集群
2. 安装IDEA编程工具
3. 安装maven并配置环境变量

二、课堂主题

1. 围绕MapReduce分布式计算讲解

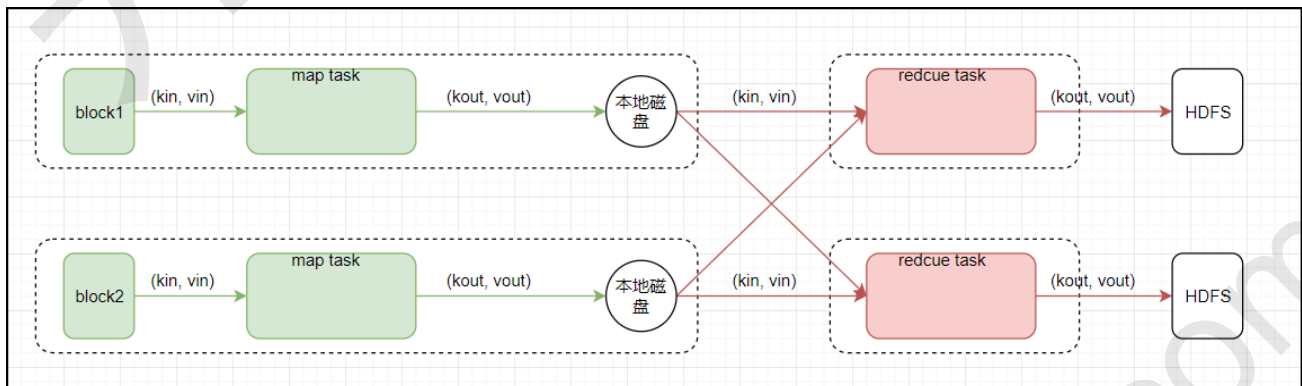
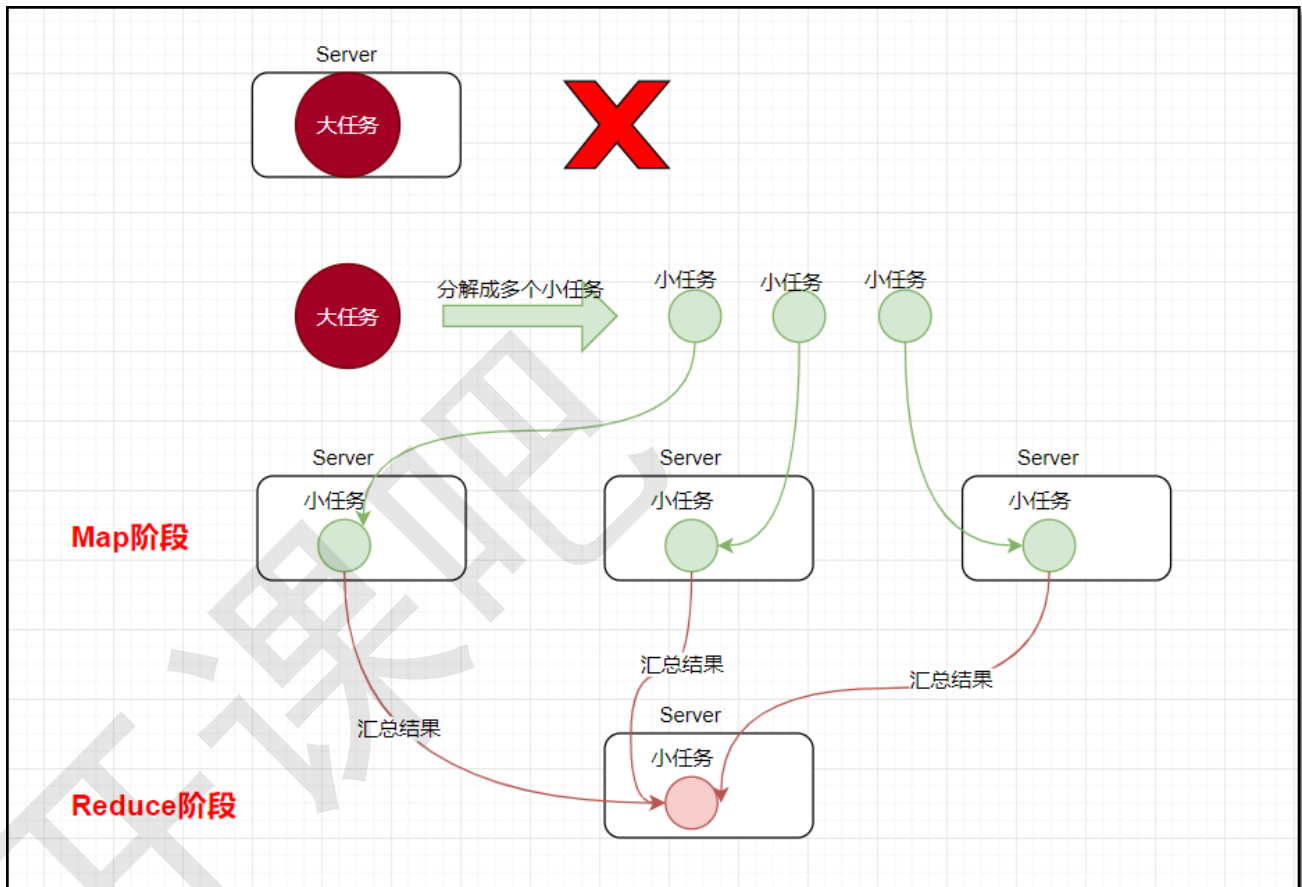
三、课堂目标

1. 理解MapReduce编程模型
2. 独立完成一个MapReduce程序并运行成功
3. 了解MapReduce工程流程
4. 掌握并描述出shuffle全过程（面试）
5. 理解并解决数据倾斜

四、知识要点

1. MapReduce编程模型

- MapReduce是采用一种**分而治之**的思想设计出来的分布式计算框架
- 如一复杂的计算任务，单台服务器无法胜任时，可将此大任务切分成一个个小的任务，小任务分别在不同的服务器上**并行**的执行；最终再汇总每个小任务的结果
- MapReduce由两个阶段组成：Map阶段（切分成一个个小的任务）、Reduce阶段（汇总小任务的结果）。



1.1 Map阶段

- map()函数以kv对作为输入，产生一系列kv对作为中间输出写入本地磁盘。

1.2 Reduce阶段

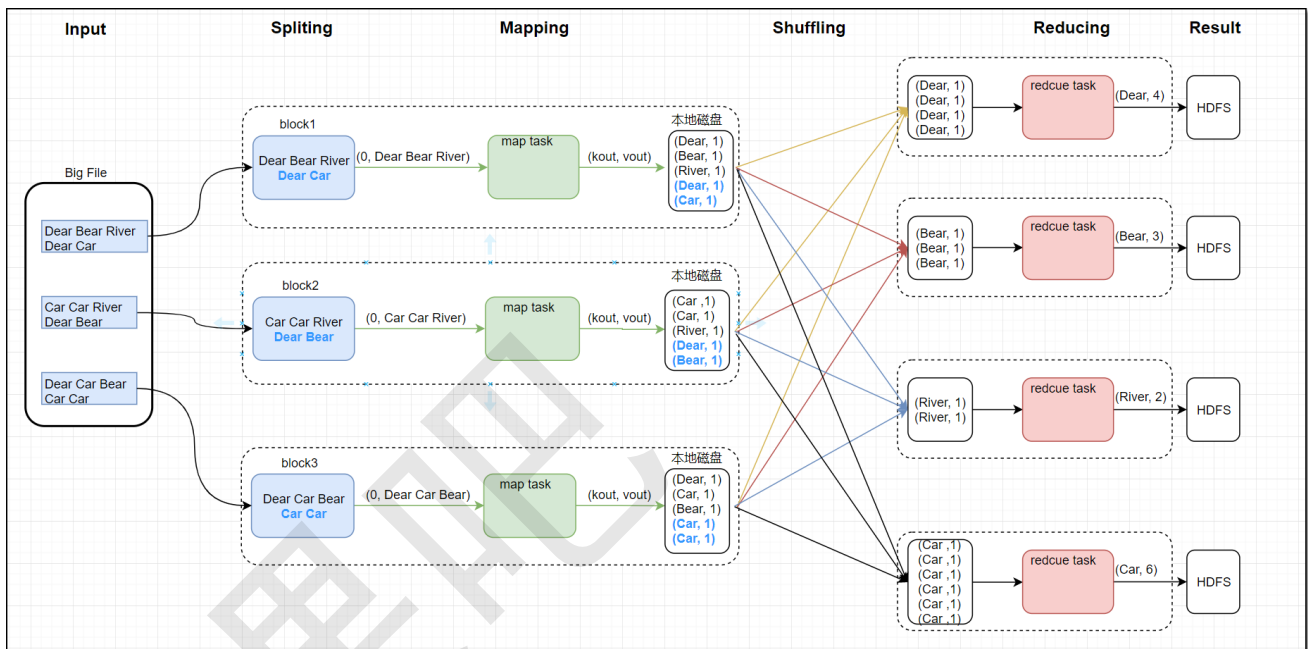
- reduce()函数通过网络将map的输出 (kv对) 作为输入，产生另外一系列kv对作为最终输出写入HDFS

1.3 Main程序入口

2. MapReduce编程示例

- 以词频统计为例

2.1 MapReduce原理图



2.2 MR参考代码

2.2.1 Mapper代码

```
package com.kaikeba.hadoop.wordcount;

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;

public class WordMap extends Mapper<LongWritable, Text, Text, IntWritable> {
    protected void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        String[] words = value.toString().split(" ");
        for (String word : words) {
            // 每个单词出现1次，作为中间结果输出
            context.write(new Text(word), new IntWritable(1));
        }
    }
}
```

2.2.2 Reducer代码

```
package com.kaikeba.hadoop.wordcount;

import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
```

```
public class WordReduce extends Reducer<Text, IntWritable, Text, IntWritable> {
    /*
        key: hello
        value: List(1, 1, ...)
    */
    protected void reduce(Text key, Iterable<IntWritable> values,
                          Context context) throws IOException, InterruptedException {
        int sum = 0;

        for (IntWritable count : values) {
            sum = sum + count.get();
        }
        context.write(key, new IntWritable(sum)); // 输出最终结果
    }
}
```

2.2.3 Main程序入口

```
package com.kaikaba.hadoop.wordcount;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.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;

public class WordMain {
    public static void main(String[] args) throws IOException,
        ClassNotFoundException, InterruptedException {
        if (args.length != 2 || args == null) {
            System.out.println("please input Path!");
            System.exit(0);
        }

        Configuration configuration = new Configuration();

        Job job = Job.getInstance(configuration, WordMain.class.getSimpleName());

        // 打jar包
        job.setJarByClass(WordMain.class);

        // 通过job设置输入/输出格式
        //job.setInputFormatClass(TextInputFormat.class);
        //job.setOutputFormatClass(TextOutputFormat.class);

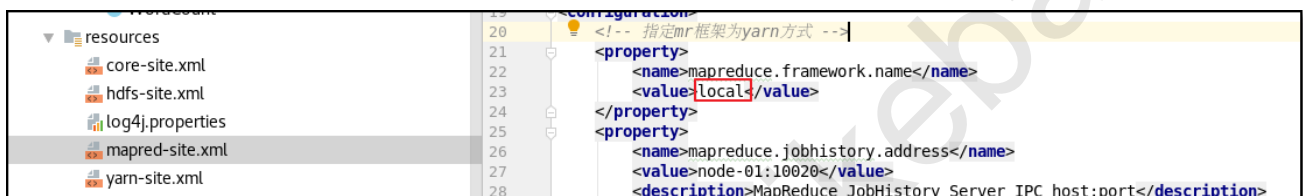
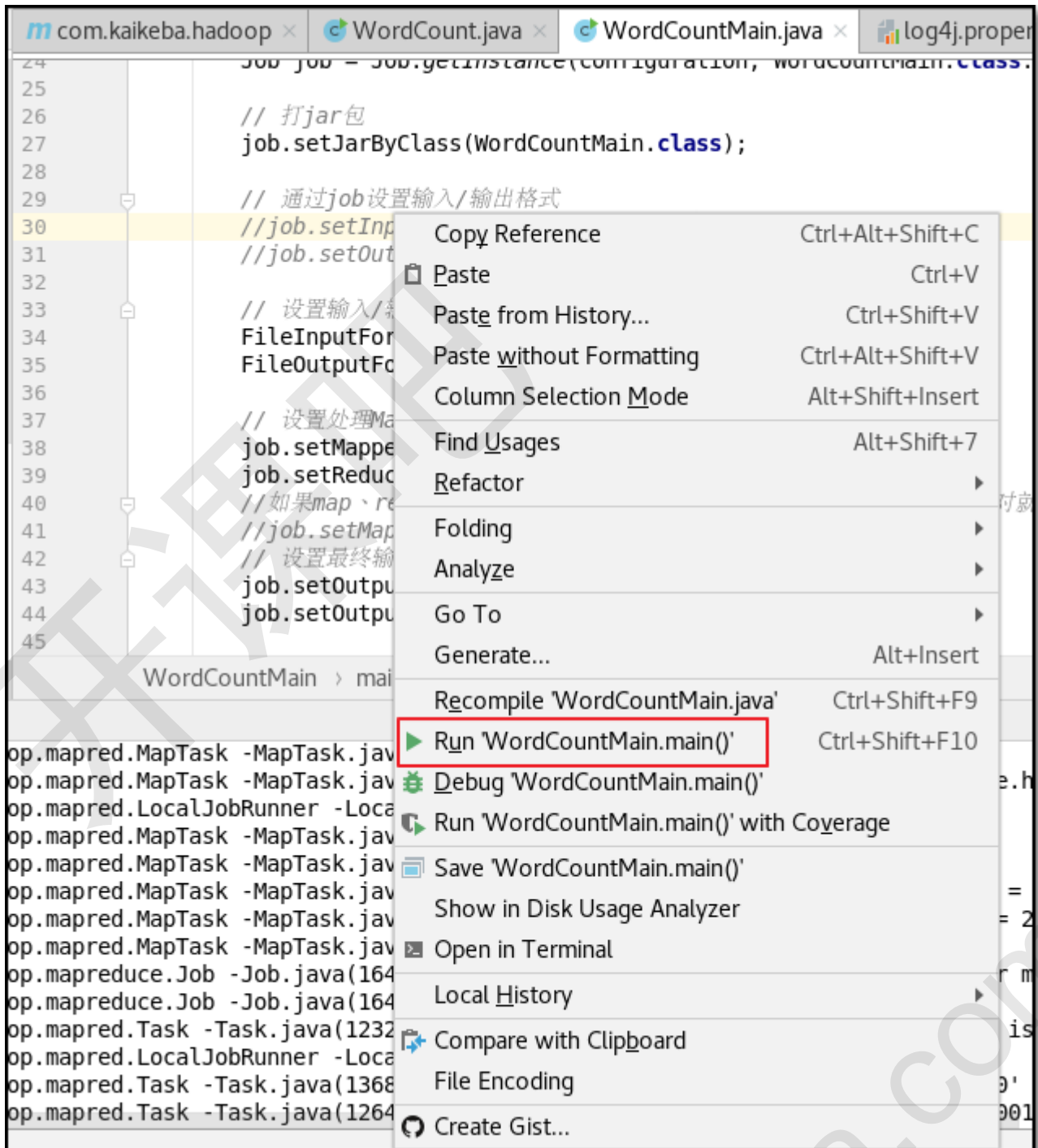
        // 设置输入/输出路径
    }
}
```

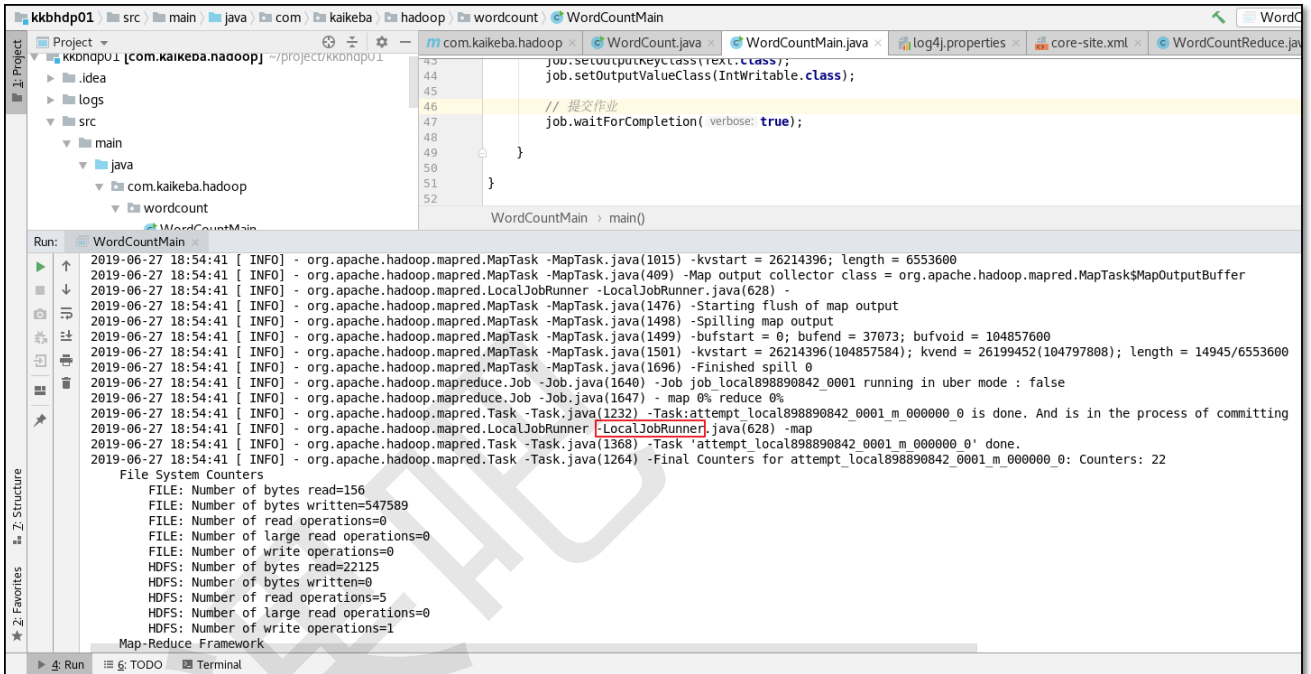
```
FileInputFormat.setInputPaths(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));

// 设置处理Map/Reduce阶段的类
job.setMapperClass(WordMap.class);
job.setReducerClass(WordReduce.class);
//如果map、reduce的输出的kv对类型一致，直接设置reduce的输出的kv对就行；如果不一样，需要分别设置
map, reduce的输出的kv类型
//job.setMapOutputKeyClass(.class)
// 设置最终输出key/value的类型m
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

// 提交作业
job.waitForCompletion(true);
}
}
```

2.3 本地运行





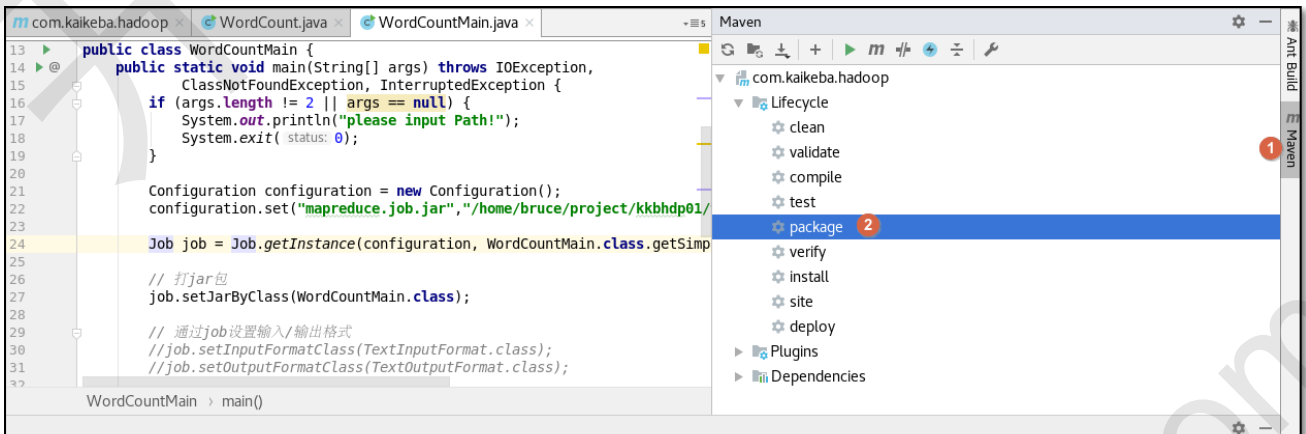
```

2019-06-27 18:54:41 [INFO] - org.apache.hadoop.mapred.MapTask -MapTask.java(1015) -kvstart = 26214396; length = 6553600
2019-06-27 18:54:41 [INFO] - org.apache.hadoop.mapred.MapTask -MapTask.java(409) -Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
2019-06-27 18:54:41 [INFO] - org.apache.hadoop.mapred.LocalJobRunner -LocalJobRunner.java(628) -
2019-06-27 18:54:41 [INFO] - org.apache.hadoop.mapred.MapTask -MapTask.java(1476) -Starting flush of map output
2019-06-27 18:54:41 [INFO] - org.apache.hadoop.mapred.MapTask -MapTask.java(1498) -Spilling map output
2019-06-27 18:54:41 [INFO] - org.apache.hadoop.mapred.MapTask -MapTask.java(1499) -bufstart = 0; bufend = 37073; bufvoid = 104857600
2019-06-27 18:54:41 [INFO] - org.apache.hadoop.mapred.MapTask -MapTask.java(1501) -kvstart = 26214396(104857584); kvend = 26199452(104797808); Length = 14945/6553600
2019-06-27 18:54:41 [INFO] - org.apache.hadoop.mapred.MapTask -MapTask.java(1696) -Finished spill 0
2019-06-27 18:54:41 [INFO] - org.apache.hadoop.mapreduce.Job -Job.java(1640) -Job job local898890842_0001 running in uber mode : false
2019-06-27 18:54:41 [INFO] - org.apache.hadoop.mapreduce.Job -Job.java(1647) - map 0% reduce 0%
2019-06-27 18:54:41 [INFO] - org.apache.hadoop.mapred.Task -Task.java(1232) -Task:attempt local898890842_0001_m_000000_0 is done. And is in the process of committing
2019-06-27 18:54:41 [INFO] - org.apache.hadoop.mapred.LocalJobRunner -LocalJobRunner.java(628) -map
2019-06-27 18:54:41 [INFO] - org.apache.hadoop.mapred.Task -Task.java(1368) -Task 'attempt local898890842_0001_m_000000_0' done.
2019-06-27 18:54:41 [INFO] - org.apache.hadoop.mapred.Task -Task.java(1264) -Final Counters for attempt_local898890842_0001_m_000000_0: Counters: 22
File System Counters
FILE: Number of bytes read=156
FILE: Number of bytes written=547589
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=22125
HDFS: Number of bytes written=0
HDFS: Number of read operations=5
HDFS: Number of large read operations=0
HDFS: Number of write operations=1
Map-Reduce Framework

```

2.4 集群运行

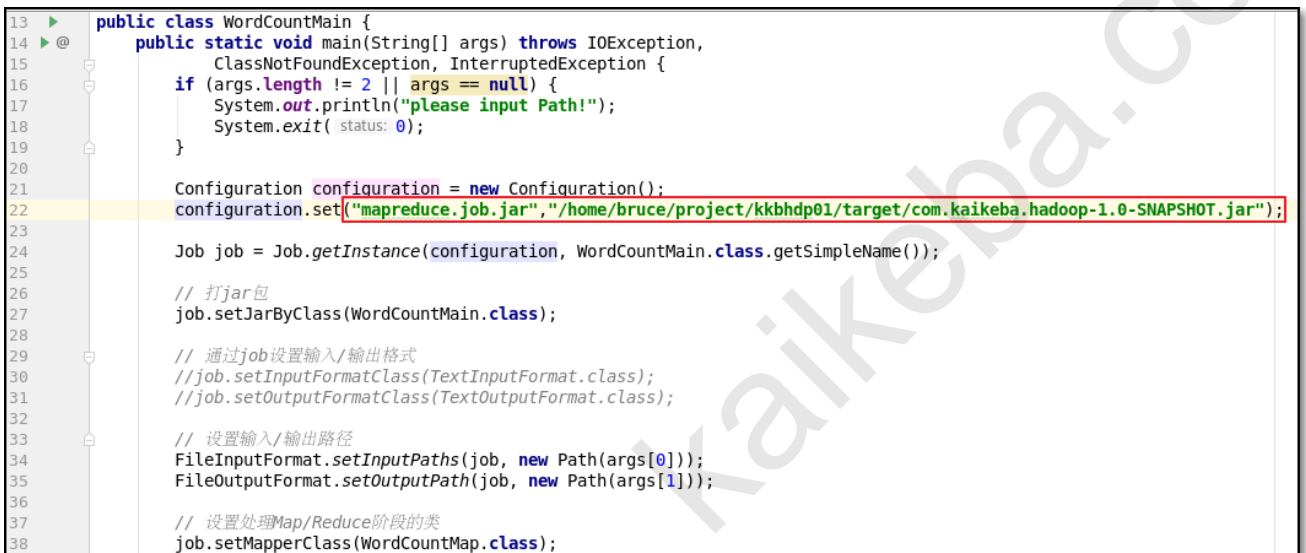
2.4.1 方式一



```

13 public class WordCountMain {
14     public static void main(String[] args) throws IOException,
15         ClassNotFoundException, InterruptedException {
16         if (args.length != 2 || args == null) {
17             System.out.println("please input Path!");
18             System.exit( status: 0);
19         }
20
21         Configuration configuration = new Configuration();
22         configuration.set("mapreduce.job.jar", "/home/bruce/project/kkbhdp01/
23
24         Job job = Job.getInstance(configuration, WordCountMain.class.getSimpleName());
25
26         // 打jar包
27         job.setJarByClass(WordCountMain.class);
28
29         // 通过job设置输入/输出格式
30         //job.setInputFormatClass(TextInputFormat.class);
31         //job.setOutputFormatClass(TextOutputFormat.class);
32
33         WordCountMain > main()

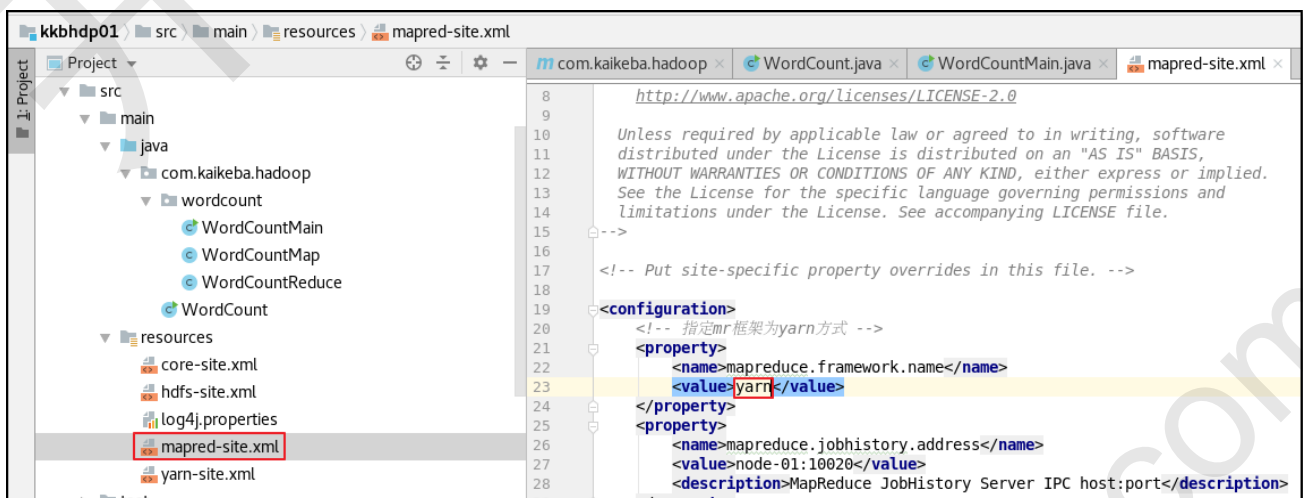
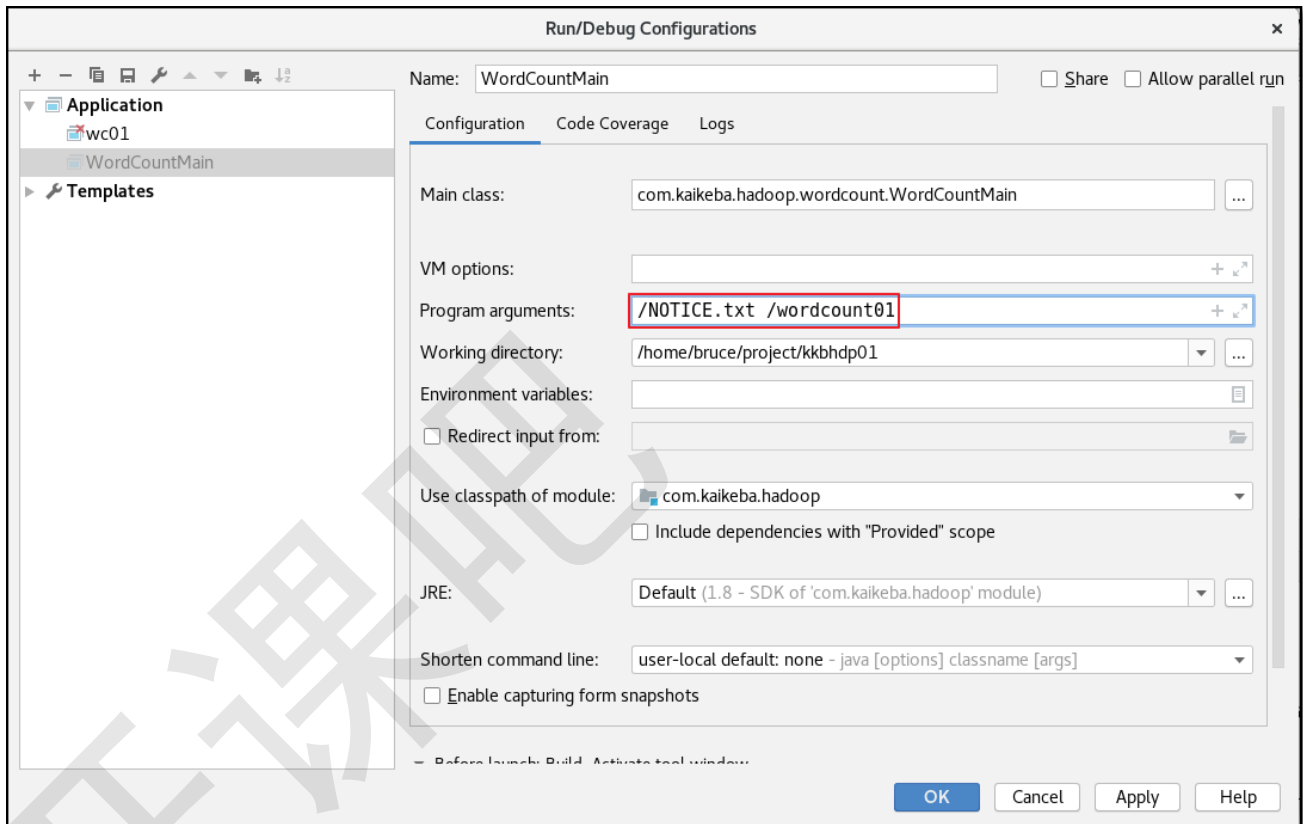
```

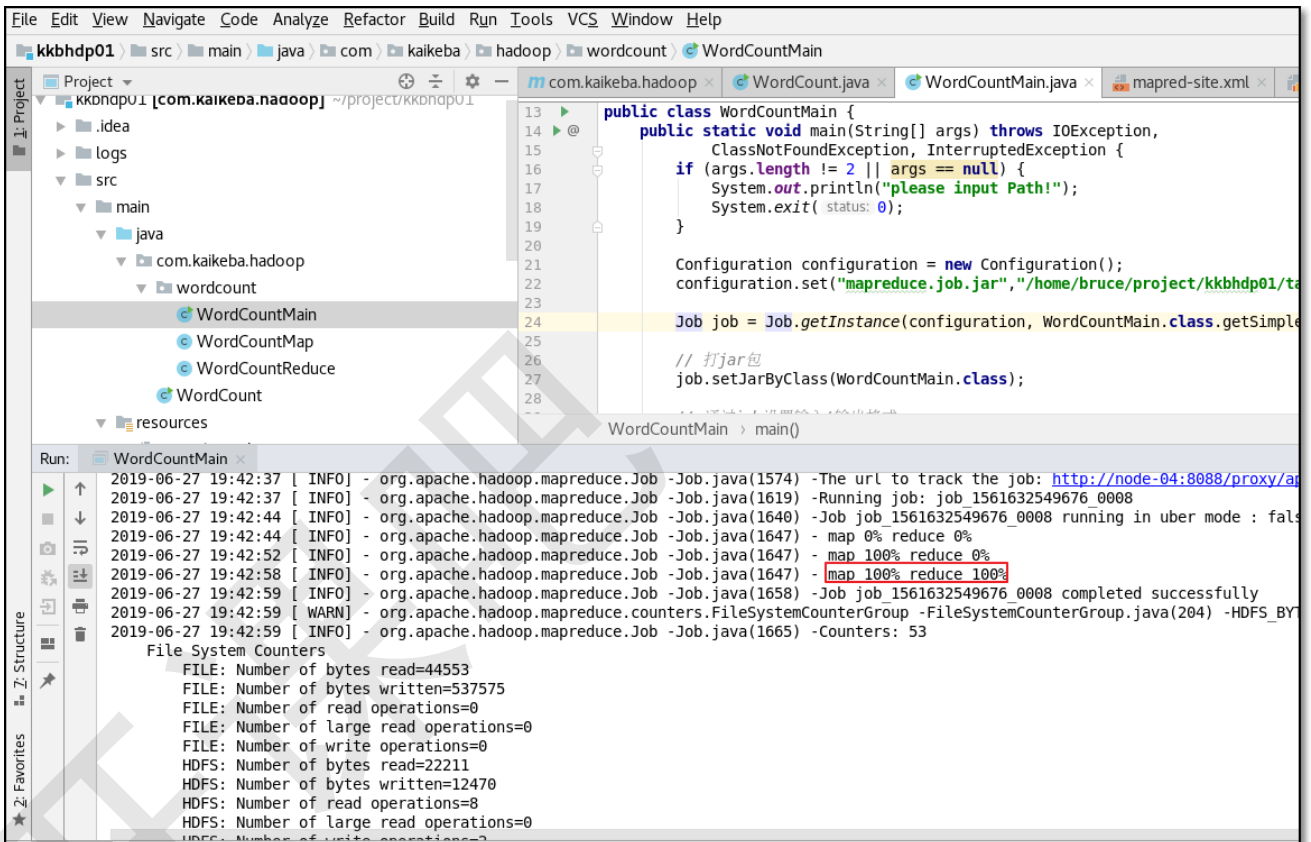


```

13 public class WordCountMain {
14     @ public static void main(String[] args) throws IOException,
15         ClassNotFoundException, InterruptedException {
16         if (args.length != 2 || args == null) {
17             System.out.println("please input Path!");
18             System.exit( status: 0);
19         }
20
21         Configuration configuration = new Configuration();
22         configuration.set("mapreduce.job.jar", "/home/bruce/project/kkbhdp01/target/com.kaikeba.hadoop-1.0-SNAPSHOT.jar");
23
24         Job job = Job.getInstance(configuration, WordCountMain.class.getSimpleName());
25
26         // 打jar包
27         job.setJarByClass(WordCountMain.class);
28
29         // 通过job设置输入/输出格式
30         //job.setInputFormatClass(TextInputFormat.class);
31         //job.setOutputFormatClass(TextOutputFormat.class);
32
33         // 设置输入/输出路径
34         FileInputFormat.setInputPaths(job, new Path(args[0]));
35         FileOutputFormat.setOutputPath(job, new Path(args[1]));
36
37         // 设置处理Map/Reduce阶段的类
38         job.setMapperClass(WordCountMap.class);

```





2.4.2 方式二

用maven将项目打包

用命令行运行mr程序

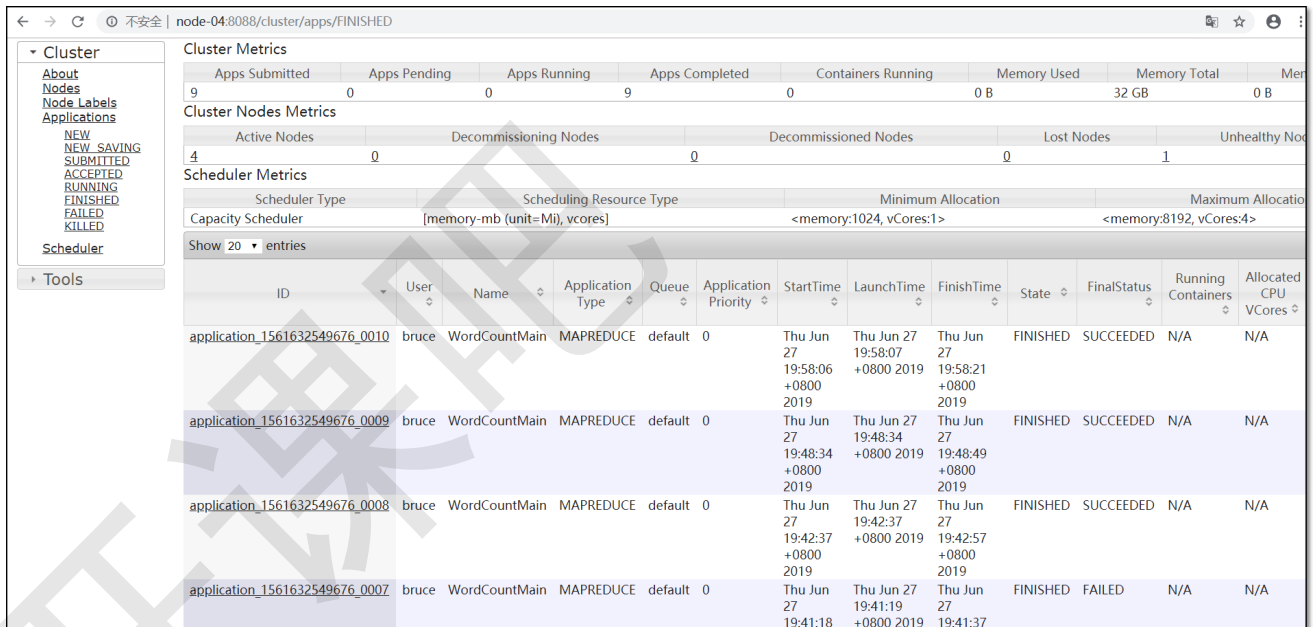
```
[bruce@node-01 Desktop]$ hadoop jar com.kaikeba.hadoop-1.0-SNAPSHOT.jar  
com.kaikeba.hadoop.wordcount.WordCountMain /NOTICE.txt /wordcount01
```

[illegible]

3. WEB UI查看结果

3.1 Yarn

浏览器url地址：rm节点IP:8088



The screenshot shows the Yarn Web UI for a cluster named 'node-04:8088/cluster/apps/FINISHED'. The interface includes a sidebar with navigation links like 'Cluster', 'About', 'Nodes', 'Node Labels', 'Applications', and 'Tools'. The main content area displays various metrics and a table of application details.

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Free
9	0	0	9	0	0 B	32 GB	0 B

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes
4	0	0	0	1

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[memory-mb (unit=Mib), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>

Applications Table

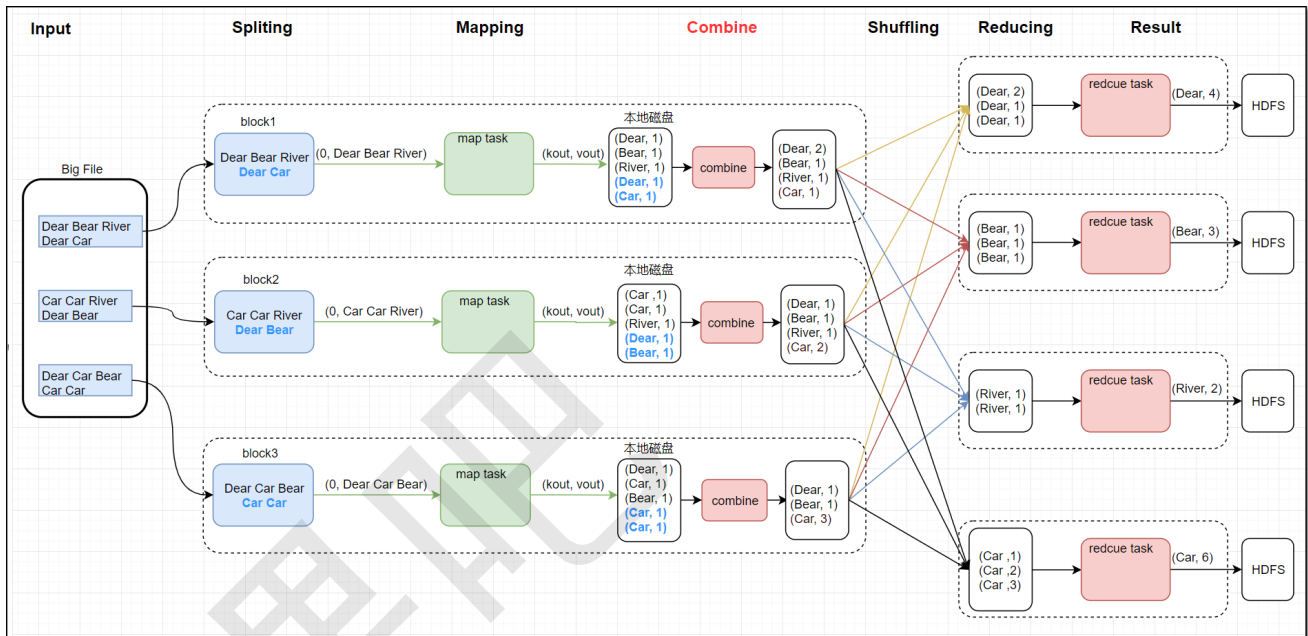
ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU Vcores
application_1561632549676_0010	bruce	WordCountMain	MAPREDUCE	default	0	Thu Jun 27 19:58:06 +0800 2019	Thu Jun 27 19:58:07 +0800 2019	Thu Jun 27 19:58:21 +0800 2019	FINISHED	SUCCEEDED	N/A	N/A
application_1561632549676_0009	bruce	WordCountMain	MAPREDUCE	default	0	Thu Jun 27 19:48:34 +0800 2019	Thu Jun 27 19:48:34 +0800 2019	Thu Jun 27 19:48:49 +0800 2019	FINISHED	SUCCEEDED	N/A	N/A
application_1561632549676_0008	bruce	WordCountMain	MAPREDUCE	default	0	Thu Jun 27 19:42:37 +0800 2019	Thu Jun 27 19:42:37 +0800 2019	Thu Jun 27 19:42:57 +0800 2019	FINISHED	SUCCEEDED	N/A	N/A
application_1561632549676_0007	bruce	WordCountMain	MAPREDUCE	default	0	Thu Jun 27 19:41:19 +0800 2019	Thu Jun 27 19:41:19 +0800 2019	Thu Jun 27 19:41:37 +0800 2019	FINISHED	FAILED	N/A	N/A

3.2 HDFS结果

```
[bruce@node-01 Desktop]$ hadoop fs -ls /wordcount01
Found 2 items
-rw-r--r--  3 bruce supergroup          0 2019-06-27 19:58 /wordcount01/_SUCCESS
-rw-r--r--  3 bruce supergroup    12470 2019-06-27 19:58 /wordcount01/part-r-00000
[bruce@node-01 Desktop]$
```

4. Combiner

- map端本地聚合；不论运行多少次Combine操作，都不会影响最终的结果
- 并非所有的mr都适合combine操作，比如求平均值



- WordCountMap与WordCountReduce代码不变
- WordCountMain中，增加`job.setCombinerClass(WordCountReduce.class);`
- 详见工程代码

```
package com.kaikeba.hadoop.wordcount;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCountMain {
    public static void main(String[] args) throws IOException,
        ClassNotFoundException, InterruptedException {
        if (args.length != 2 || args == null) {
            System.out.println("please input Path!");
            System.exit(0);
        }

        Configuration configuration = new Configuration();

        //configuration.set("mapreduce.job.jar", "/home/bruce/project/kkbhdp01/target/com.kaikeba.hadoop
        -1.0-SNAPSHOT.jar");

        Job job = Job.getInstance(configuration, WordCountMain.class.getSimpleName());

        // 打jar包
        job.setJarByClass(WordCountMain.class);
```

```
// 通过job设置输入/输出格式
//job.setInputFormatClass(TextInputFormat.class);
//job.setOutputFormatClass(TextOutputFormat.class);

// 设置输入/输出路径
FileInputFormat.setInputPaths(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));

// 设置处理Map/Reduce阶段的类
job.setMapperClass(WordCountMap.class);
job.setCombinerClass(WordCountReduce.class);
job.setReducerClass(WordCountReduce.class);
//如果map、reduce的输出的kv对类型一致，直接设置reduce的输出的kv对就行；如果不一样，需要分别设置
map, reduce的输出的kv类型
//job.setMapOutputKeyClass(.class)
// 设置最终输出key/value的类型m
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

// 提交作业
job.waitForCompletion(true);
}
}
```

```
public class WordCountMain {
    public static void main(String[] args) throws IOException,
        ClassNotFoundException, InterruptedException {
        if (args.length != 2 || args == null) {
            System.out.println("please input Path!");
            System.exit(0);
        }

        Configuration configuration = new Configuration();
        configuration.set("mapreduce.job.jar", "/home/bruce/project/kkbhdp01/target/com.kaikeba.hadoop-1.0-SNAPSHOT.jar");

        Job job = Job.getInstance(configuration, WordCountMain.class.getSimpleName());

        // 打jar包
        job.setJarByClass(WordCountMain.class);

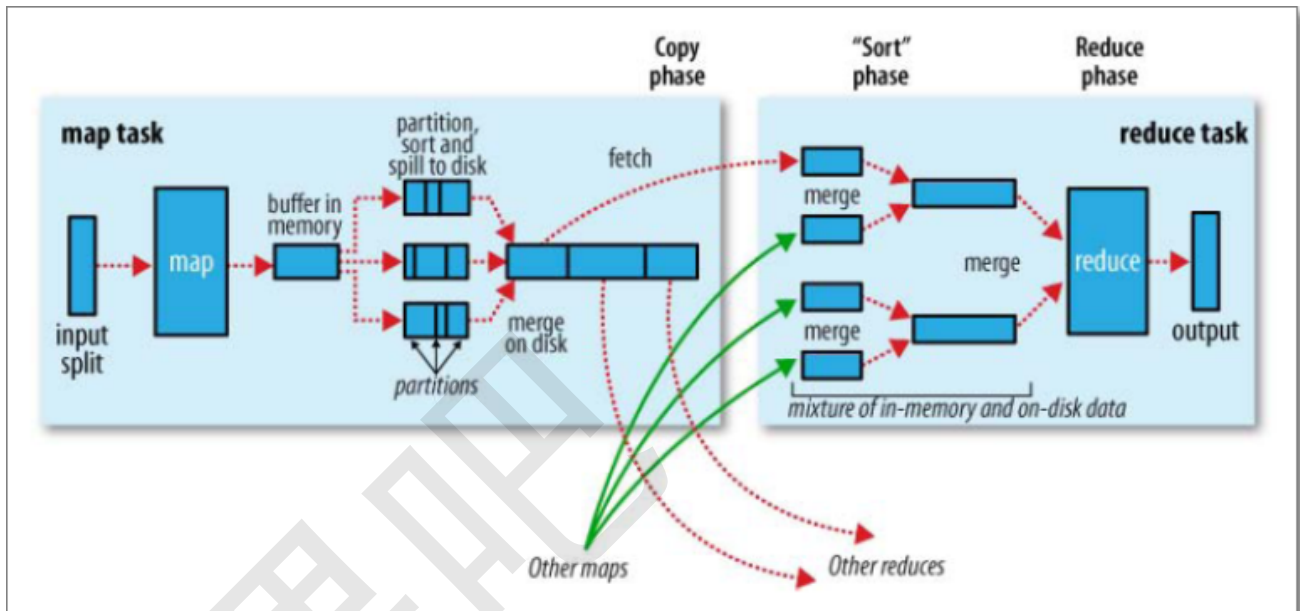
        // 通过job设置输入/输出格式
        //job.setInputFormatClass(TextInputFormat.class);
        //job.setOutputFormatClass(TextOutputFormat.class);

        // 设置输入/输出路径
        FileInputFormat.setInputPaths(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // 设置处理Map/Reduce阶段的类
        job.setMapperClass(WordCountMap.class);
        job.setCombinerClass(WordCountReduce.class);
        job.setReducerClass(WordCountReduce.class);
        //如果map、reduce的输出的kv对类型一致，直接设置reduce的输出的kv对就行；如果不一样，需要分别设置map, reduce的输出的kv类型
        //job.setMapOutputKeyClass(.class)
        // 设置最终输出key/value的类型m
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        // 提交作业
        job.waitForCompletion(verbose: true);
    }
}
```

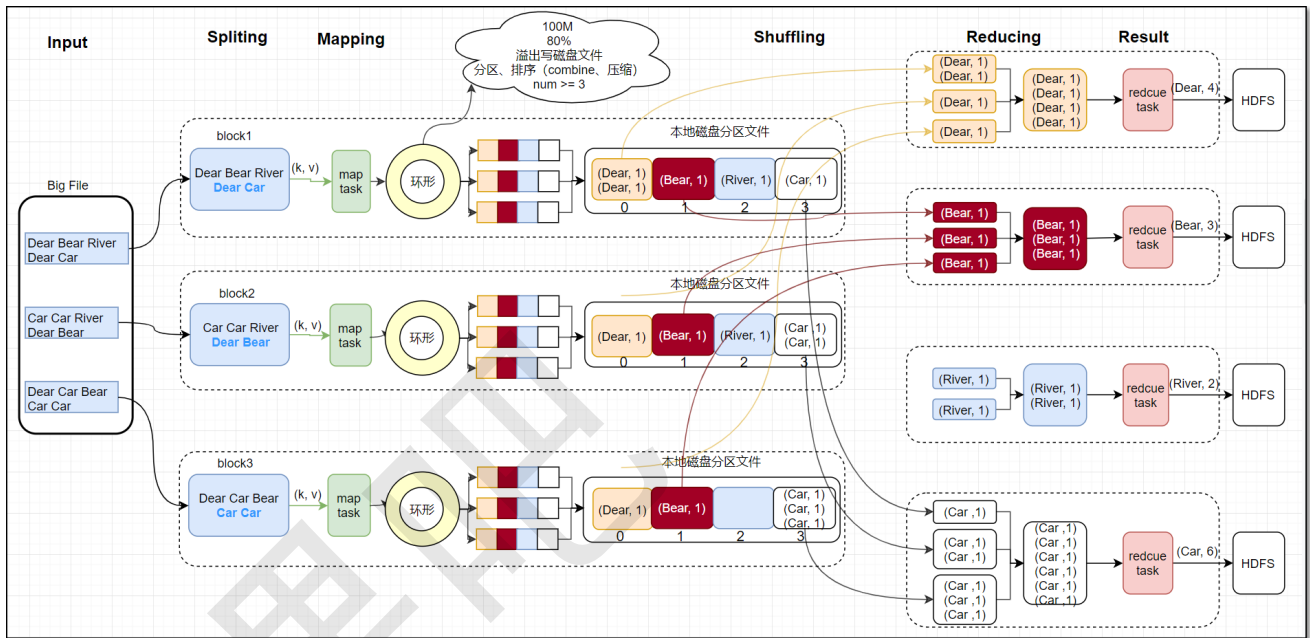
5. Shuffle



```

29  @InterfaceAudience.Public
30  @InterfaceStability.Stable
31  public class HashPartitioner<K2, V2> implements Partitioner<K2, V2> {
32
33      public void configure(JobConf job) {}
34
35      /** Use {@link Object#hashCode()} to partition. */
36      @ public int getPartition(K2 key, V2 value,
37                              int numReduceTasks) {
38          return (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;
39      }
40
41  }

```



6. 自定义分区Partition

- MapReduce自带的分区器是HashPartitioner
- 原理：先对map输出的key求hash值，再模上reduce task个数，根据结果，决定此输出kv对，被匹配的reduce取走

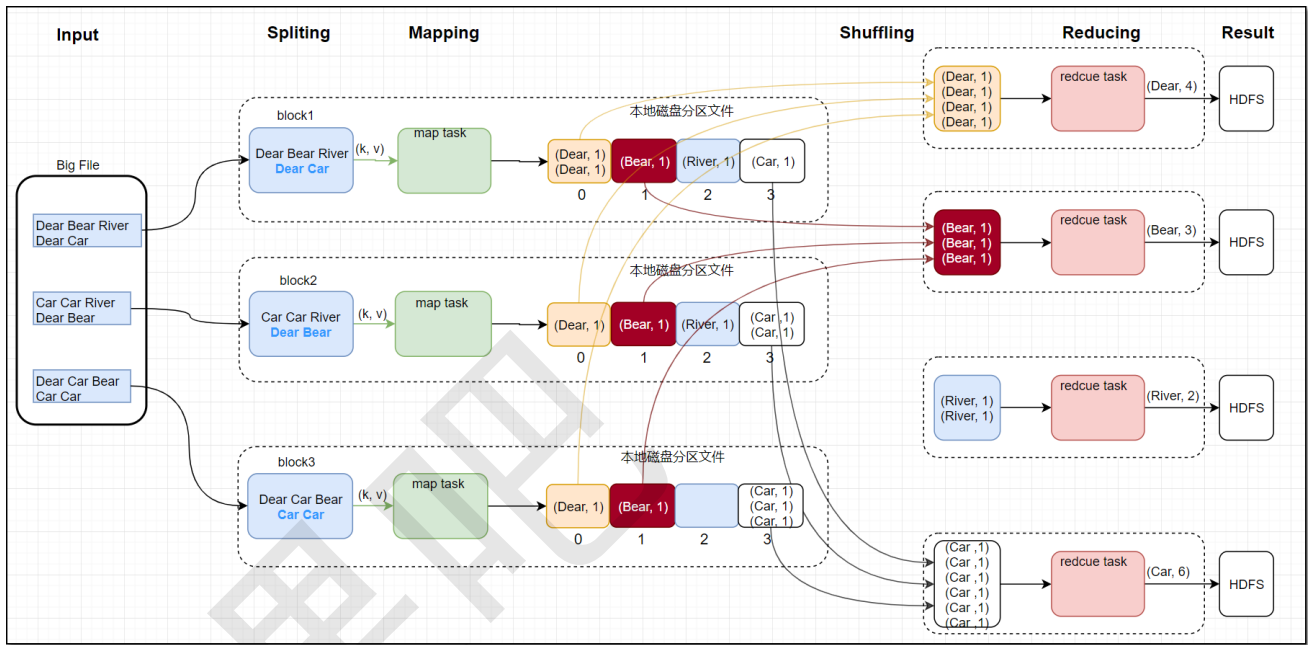
```

29  @InterfaceAudience.Public
30  @InterfaceStability.Stable
31  public class HashPartitioner<K2, V2> implements Partitioner<K2, V2> {
32
33      public void configure(JobConf job) {}
34
35      /** Use {@link Object#hashCode()} to partition. */
36      @ public int getPartition(K2 key, V2 value,
37                              int numReduceTasks) {
38          return (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;
39      }
40
41  }

```

分区值 = $\text{key.hashCode()} \% \text{numReduceTasks}$

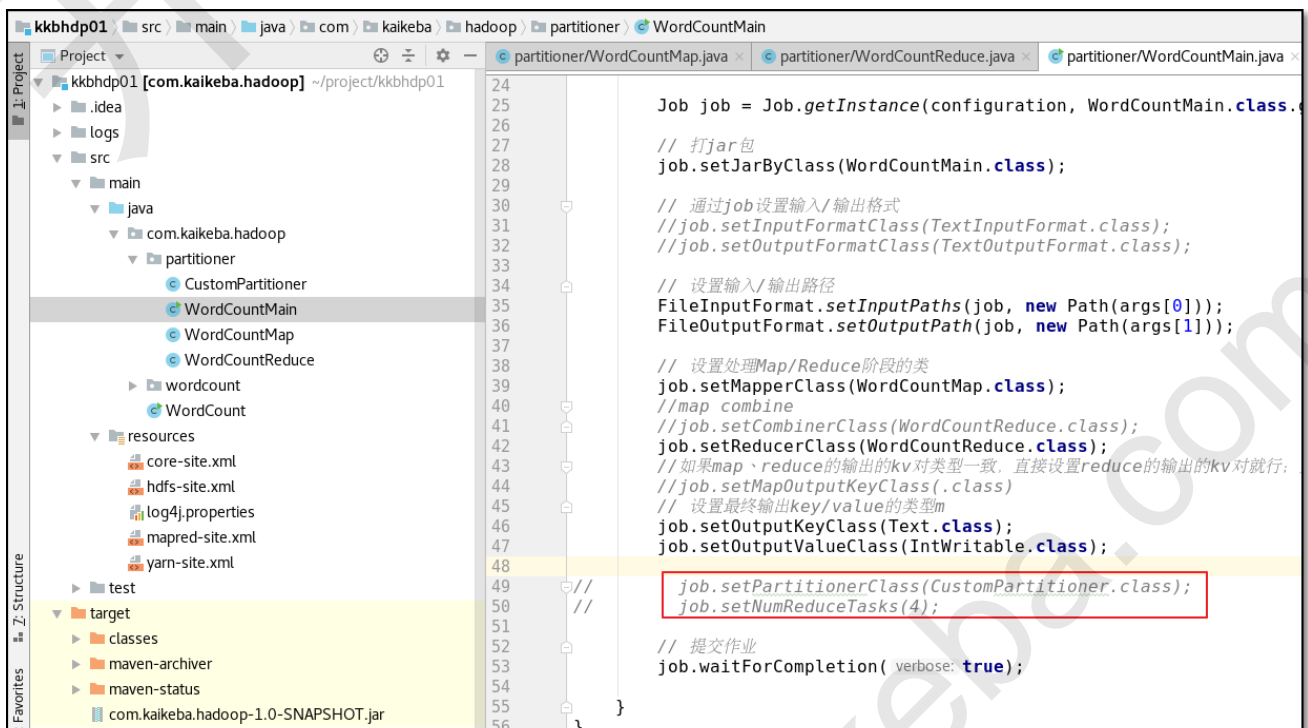
Dear 哈希值 $\% 4 = 0$ → 0号分区, 即reduce1
 Bear 哈希值 $\% 4 = 1$ → 1号分区, 即reduce2
 River 哈希值 $\% 4 = 2$ → 2号分区, 即reduce3
 Car 哈希值 $\% 4 = 3$ → 3号分区, 即reduce4



- 根据业务逻辑，设计自定义分区，比如实现图上的功能

6.1 默认分区

- 默认HashPartitioner分区时，查看结果（看代码）



```

24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56

Job job = Job.getInstance(configuration, WordCountMain.class);

// 打jar包
job.setJarByClass(WordCountMain.class);

// 通过job设置输入/输出格式
//job.setInputFormatClass(TextInputFormat.class);
//job.setOutputFormatClass(TextOutputFormat.class);

// 设置输入/输出路径
FileInputFormat.setInputPaths(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));

// 设置处理Map/Reduce阶段的类
//map combine
job.setMapperClass(WordCountMap.class);
job.setReducerClass(WordCountReduce.class);
// 如果map、reduce的输出的kv对类型一致，直接设置reduce的输出的kv对就行
//job.setMapOutputKeyClass(.class)
// 设置最终输出key/value的类型m
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

// 提交作业
job.waitForCompletion(verbose: true);
    
```

结果如下：

```
[bruce@node-01 Desktop]$ hadoop fs -ls -R /wordcount02
-rw-r--r--  3 bruce supergroup      0 2019-06-27 22:09 /wordcount02/ SUCCESS
-rw-r--r--  3 bruce supergroup    28 2019-06-27 22:09 /wordcount02/part-r-00000
[bruce@node-01 Desktop]$ hadoop fs -cat /wordcount02/part-r-00000  一个结果文件
Bear      3
Car       6
Dear      4
River     2
[bruce@node-01 Desktop]$
```

6.2 自定义分区

- 现开户自定义分区功能，并设定reduce个数为4
- 详见工程代码

```
package com.kaikaba.hadoop.partitionner;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Partitioner;

import java.util.HashMap;

public class CustomPartitioner extends Partitioner<Text, IntWritable> {
    public static HashMap<String, Integer> dict = new HashMap<String, Integer>();

    static{
        dict.put("Dear", 0);
        dict.put("Bear", 1);
        dict.put("River", 2);
        dict.put("Car", 3);
    }

    public int getPartition(Text text, IntWritable intWritable, int i) {
        //
        int partitionIndex = dict.get(text.toString());
        return partitionIndex;
    }
}
```



```

25 Job job = Job.getInstance(configuration, WordCountMain.class.getSimpleName());
26
27 // 打jar包
28 job.setJarByClass(WordCountMain.class);
29
30 // 通过job设置输入/输出格式
31 //job.setInputFormatClass(TextInputFormat.class);
32 //job.setOutputFormatClass(TextOutputFormat.class);
33
34 // 设置输入/输出路径
35 FileInputFormat.setInputPaths(job, new Path(args[0]));
36 FileOutputFormat.setOutputPath(job, new Path(args[1]));
37
38 // 设置处理Map/Reduce阶段的类
39 job.setMapperClass(WordCountMap.class);
40 //map combine
41 //job.setCombinerClass(WordCountReduce.class);
42 job.setReducerClass(WordCountReduce.class);
43 // 如果map、reduce的输出kv对类型一致，直接设置reduce的输出kv类型
44 //job.setMapOutputKeyClass(.class)
45 // 设置最终输出key/value的类型
46 job.setOutputKeyClass(Text.class);
47 job.setOutputValueClass(IntWritable.class);
48
49 job.setPartitionerClass(CustomPartitioner.class);
50 job.setNumReduceTasks(4);
51
52 // 提交作业
53 job.waitForCompletion(verbose: true);

```

• 运行结果

```

[bruce@node-01 Desktop]$ hadoop fs -ls -R /wordcount03
-rw-r--r--  3 bruce supergroup      0 2019-06-27 22:16 /wordcount03/_SUCCESS
-rw-r--r--  3 bruce supergroup      7 2019-06-27 22:16 /wordcount03/part-r-00000
-rw-r--r--  3 bruce supergroup      7 2019-06-27 22:16 /wordcount03/part-r-00001
-rw-r--r--  3 bruce supergroup      8 2019-06-27 22:16 /wordcount03/part-r-00002
-rw-r--r--  3 bruce supergroup      6 2019-06-27 22:16 /wordcount03/part-r-00003
[bruce@node-01 Desktop]$ hadoop fs -cat /wordcount03/part-r-00000
Dear 4 ①
[bruce@node-01 Desktop]$ hadoop fs -cat /wordcount03/part-r-00001
Bear 3 ②
[bruce@node-01 Desktop]$ hadoop fs -cat /wordcount03/part-r-00002
River 2 ③
[bruce@node-01 Desktop]$ hadoop fs -cat /wordcount03/part-r-00003
Car 6 ④
[bruce@node-01 Desktop]$

```

4个reduce任务，分别对应一个结果文件

7. 二次排序

- MapReduce中，根据key进行分区、排序、分组
- MapReduce会按照基本类型对应的key进行排序，如int类型的IntWritable，默认升序排序
- 为什么要自定义排序规则？
- 现有需求，需要自定义key类型，并自定义key的排序规则，如按照人的salary降序排序，若相同，则再按age升序排序
- 详见工程代码

```

package com.kaikeba.hadoop.secondarysort;

import org.apache.hadoop.io.WritableComparable;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class Person implements WritableComparable<Person> {
    private String name;

```

```
private int age;
private int salary;

public Person() {
}

public Person(String name, int age, int salary) {
    //super();
    this.name = name;
    this.age = age;
    this.salary = salary;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public int getAge() {
    return age;
}

public void setAge(int age) {
    this.age = age;
}

public int getSalary() {
    return salary;
}

public void setSalary(int salary) {
    this.salary = salary;
}

@Override
public String toString() {
    return this.salary + " " + this.age + " " + this.name;
}

//先比较salary, 高的排序在前; 若相同, age小的在前
public int compareTo(Person o) {
    int compareResult1= this.salary - o.salary;
    if(compareResult1 != 0) {
        return -compareResult1;
    } else {
        return this.age - o.age;
    }
}

//序列化, 将NewKey转化成使用流传送的二进制
```

```
public void write(DataOutput dataOutput) throws IOException {
    dataOutput.writeUTF(name);
    dataOutput.writeInt(age);
    dataOutput.writeInt(salary);
}

//使用in读字段的顺序，要与write方法中写的顺序保持一致
public void readFields(DataInput dataInput) throws IOException {
    //read string
    this.name = dataInput.readUTF();
    this.age = dataInput.readInt();
    this.salary = dataInput.readInt();
}
}
```

8. MapReduce分区倾斜

数据倾斜是数据中的常见情况。数据中不可避免地会出现离群值（outlier），并导致数据倾斜。这些离群值会显著地拖慢MapReduce的执行。常见的数据倾斜有以下几类：

1. 数据频率倾斜——某一个区域的数据量要远远大于其他区域。
2. 数据大小倾斜——部分记录的大小远远大于平均值。

在map端和reduce端都有可能发生数据倾斜。在map端的数据倾斜会让多样化的数据集的处理效率更低。在reduce端的数据倾斜常常来源于MapReduce的默认分区器。

数据倾斜会导致map和reduce的任务执行时间大为延长，也会让需要缓存数据集的操作消耗更多的内存资源。

8.1 如何诊断是否存在数据倾斜

1. 关注由map的输出数据中的数据频率倾斜的问题。
2. 如何诊断map输出中哪些键存在数据倾斜？
 - 在reduce方法中加入记录map输出键的详细情况的功能
 - 在发现了倾斜数据的存在之后，就很有必要诊断造成数据倾斜的那些键。有一个简便方法就是在代码里实现追踪每个键的最大值。为了减少追踪量，可以设置数据量阈值，只追踪那些数据量大于阈值的键，并输出到日志中。实现代码如下

```
public static final String MAX_VALUES = "skew.maxvalues";
private int maxValueThreshold;

@Override
public void configure(JobConf job) {
    maxValueThreshold = job.getInt(MAX_VALUES, 100);
}

@Override
public void reduce(Text key, Iterator<Text> values,
    OutputCollector<Text, Text> output,
    Reporter reporter) throws IOException {
```

```
int i = 0;

while (values.hasNext()) {
    values.next();
    i++;
}

if (++i > maxValueThreshold) {
    log.info("Received " + i + " values for key " + key);
}
}
```

- 运行作业后就可以从日志中判断发生倾斜的键以及倾斜程度
- 跟踪倾斜数据是了解数据的重要一步，也是设计MapReduce作业的重要基础

8.2 减缓Reduce端数据倾斜

- Reduce数据倾斜一般是指map的输出数据中存在数据频率倾斜的状况，也就是部分输出键的数据量远远大于其它的输出键
- 如何减小reduce端数据倾斜的性能损失？
 - 用一系列的方法减小数据倾斜的风险，例如使用自定义的分区器，使用map端连接等
 - 在这个方案中将讨论多个减轻reduce数据倾斜的性能损失的方法
 - 1. 抽样和范围分区

Hadoop默认的分区器是基于map输出键的哈希值分区。这仅在数据分布比较均匀时比较好。在有数据倾斜时就很有问题。

使用分区器需要首先了解数据的特性。**TotalOrderPartitioner**中，可以通过对原始数据进行抽样得到的结果集来预设分区边界值。TotalOrderPartitioner中的范围分区器可以通过预设的分区边界值进行分区。因此它也可以很好地用在矫正数据中的部分键的数据倾斜问题。

2. 自定义分区

另一个抽样和范围分区的替代方案是基于输出键的背景知识进行自定义分区。例如，如果map输出键的单词来源于一本书。其中大部分必然是省略词（stopword）。那么就可以将自定义分区将这部分省略词发送给固定的一部分reduce实例。而将其他的都发送给剩余的reduce实例。

3. Combine

使用Combine可以大量地减小数据频率倾斜和数据大小倾斜。在可能的情况下，combine的目的就是聚合并精简数据。在技术48种介绍了combine。

4. Map端连接和半连接

如果连接的数据集太大而不能在map端的连接中使用。那么可以考虑第4章和第7章中介绍的超大数据集的连接优化方案。

5. 数据大小倾斜的自定义策略

在map端或reduce端的数据大小倾斜都会对缓存造成较大的影响，乃至导致OutOfMemoryError异常。处理这种情况并不容易。可以参考以下方法。

- 设置mapred.linerecordreader.maxlength来限制RecordReader读取的最大长度。RecordReader在TextInputFormat和KeyValueTextInputFormat类中使用。默认长度没有上限。

- 通过org.apache.hadoop.contrib.utils.join设置缓存的数据集的记录数上限。在reduce中默认的缓存记录数上限是100条。
- 考虑使用有损数据结构压缩数据，如Bloom过滤器。

9. MR调优

10. 常见MapReduce编程

4. 全排序，防止数据倾斜

五、拓展点、未来计划、行业趋势（5分钟）

1. 利用SequenceFile文件的MR编程，解决HDFS小文件存储问题
2. MR过程中压缩算法的使用

六、总结（5分钟）

七、作业

可从下面的题库任意选择。

八、互动问答

九、题库 - 本堂课知识点

1. 描述MR的shuffle全流程（面试）
2. 搭建MAVEN工程，统计词频，并提交集群运行，查看结果
3. 利用搜狗数据，找出所有独立的uid并写入HDFS
4. 利用搜狗数据，找出所有独立的uid并写入HDFS，并要求使用Map端的Combine操作
5. 谈谈什么是数据倾斜，什么情况会造成数据倾斜？（面试）
6. 对MR数据倾斜，如何解决？（面试）