

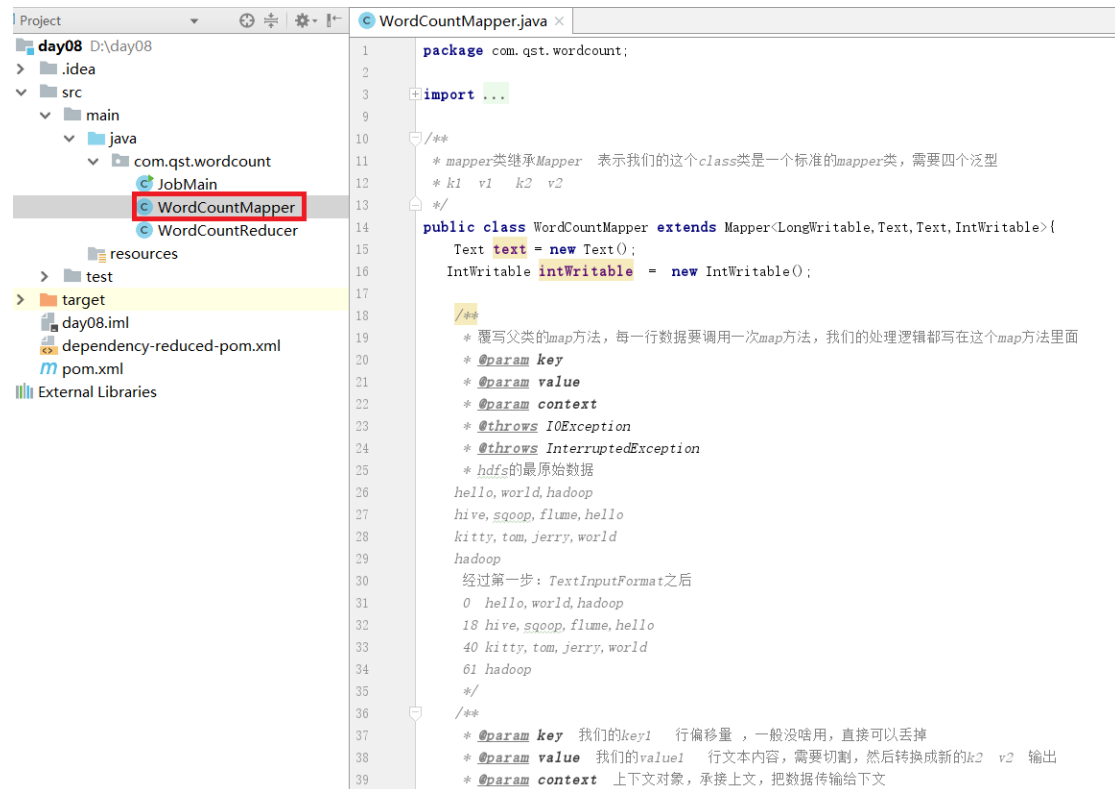
WordCount 示例编写

需求：在一堆给定的文本文件中统计输出每一个单词出现的总次数

数据格式准备如下：

```
hadoop,hive,hbase
hive,storm
hive,hbase,kafka
spark,flume,kafka,storm
hbase,hadoop,hbase
hive,spark,storm
```

定义一个 mapper 类



```
1 package com.qst.wordcount;
2
3 import ...
4
5 /**
6  * mapper类继承Mapper 表示我们的这个class类是一个标准的mapper类，需要四个泛型
7  * k1 v1 k2 v2
8  */
9
10 public class WordCountMapper extends Mapper<LongWritable,Text,Text,IntWritable>{
11     Text text = new Text();
12     IntWritable intWritable = new IntWritable();
13
14     /**
15      * 覆写父类的map方法，每一行数据要调用一次map方法，我们的处理逻辑都写在这个map方法里面
16      * @param key
17      * @param value
18      * @param context
19      * @throws IOException
20      * @throws InterruptedException
21      * hdfs的最原始数据
22      * hello,world,hadoop
23      * hive,sqoop,flume,hello
24      * kitty,tom,jerry,world
25      * hadoop
26      * 经过第一步：TextInputFormat之后
27      * 0 hello,world,hadoop
28      * 18 hive,sqoop,flume,hello
29      * 40 kitty,tom,jerry,world
30      * 61 hadoop
31      */
32     /**
33      * @param key 我们的key1 行偏移量，一般没啥用，直接可以丢掉
34      * @param value 我们的value1 行文本内容，需要切割，然后转换成新的k2 v2 输出
35      * @param context 上下文对象，承接上文，把数据传输给下文
```

```
package com.qst.wordcount;
```

```
import org.apache.hadoop.io.IntWritable;
```

```

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

/**
 * mapper 类继承 Mapper 表示我们的这个 class 类是一个标准的 mapper 类,
 * 需要四个泛型
 * k1  v1  k2  v2
 */

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, 每一行数据要调用一次 map 方法, 我们的处理逻辑都写在这个 map 方法里面
     * @param key
     * @param value
     * @param context
     * @throws IOException
     * @throws InterruptedException
     * hdfs 的最原始数据
     */
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        Text text = new Text();
        text.set(value);
        context.write(text, new IntWritable(1));
    }
}

hadoop,hive,hbase
hive,storm
hive,hbase,kafka
spark,flume,kafka,storm
hbase,hadoop,hbase
hive,spark,storm

经过第一步: TextInputFormat 之后

```

```

0  hadoop,hive,hbase
17 hive,storm
27 hive,hbase,kafka
*/
/**
 * @param key 我们的 key1 行偏移量 , 一般没啥用, 直接可以丢掉
 * @param value 我们的 value1 行文本内容, 需要切割, 然后转换成
新的 k2 v2 输出
 * @param context 上下文对象, 承接上文, 把数据传输给下文
 * @throws IOException
 * @throws InterruptedException
 */
@Override
protected void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
    /*
hadoop,hive,hbase
hive,storm
hive,hbase,kafka
spark,flume,kafka,storm
hbase,hadoop,hbase
hive,spark,storm
    */
    String line = value.toString();
    String[] split = line.split(",");
    //遍历我们切割出来的单词
    for (String word : split) {
        text.set(word);
        intWritable.set(1);
    }
}

```

```

        //写出我们的 k2 v2 这里的类型跟我们的 k2 v2 保持一致
        context.write(text,intWritable);

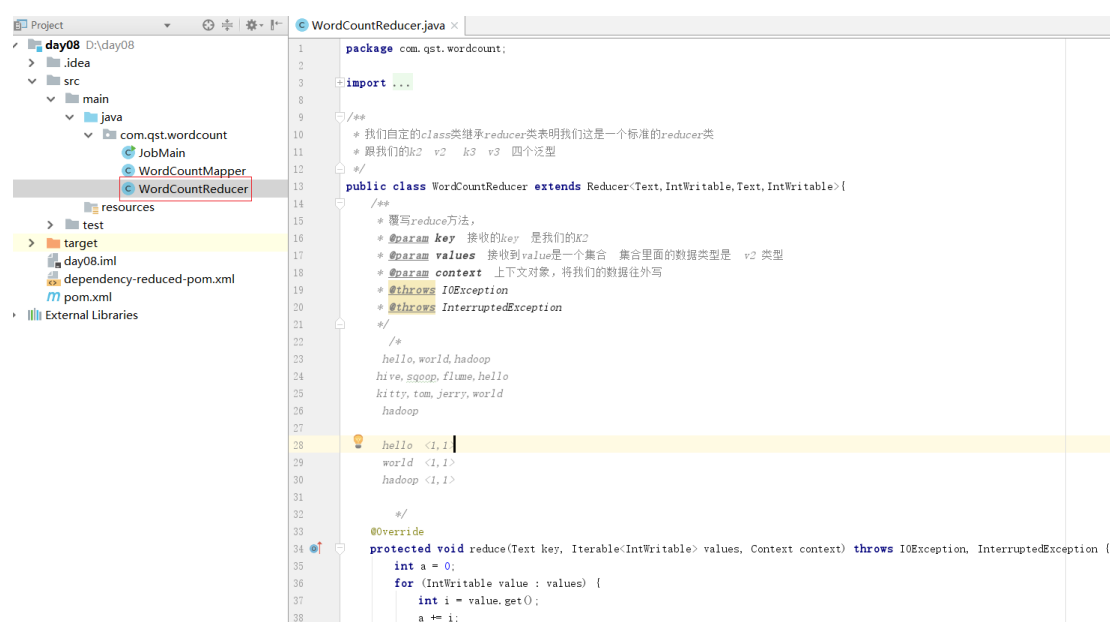
    }

}

}

```

定义一个 reducer 类



```

package com.qst.wordcount;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

/**
 * 我们自定的 class 类继承 reducer 类表明我们这是一个标准的 reducer 类
 * 跟我们的 k2 v2 k3 v3 四个泛型
 */

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
 */

/*
hadoop,hive,hbase
hive,storm
hive,hbase,kafka
spark,flume,kafka,storm
hbase,hadoop,hbase
hive,spark,storm
    hadoop <1,1>
    hive <1,1,1,1>
    hbase<1,1,1>

*/

@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;
    }
}

```

```

    }

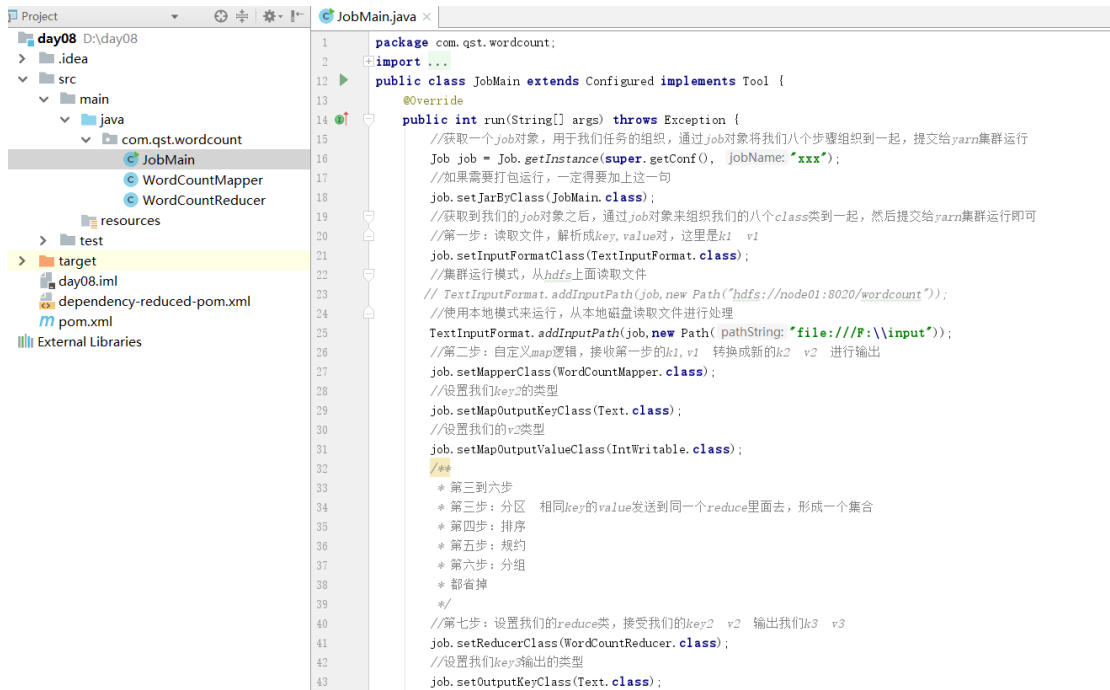
    //将我们的数据写出去

    context.write(key,new IntWritable(a));

}

}

```



```
package com.qst.wordcount;
```

```
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.Text;
```

```
import org.apache.hadoop.mapreduce.Job;
```

```
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
```

```
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

```
import org.apache.hadoop.util.Tool;
```

```
import org.apache.hadoop.util.ToolRunner;
```

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

```
    @Override
```

```

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

    //获取一个 job 对象，用于我们任务的组织，通过 job 对象将我们八个
    步骤组织到一起，提交给 yarn 集群运行

    Job job = Job.getInstance(super.getConf(), "xxx");

    //如果需要打包运行，一定得要加上这一句

    job.setJarByClass(JobMain.class);

    //获取到我们的 job 对象之后，通过 job 对象来组织我们的八个 class
    类到一起，然后提交给 yarn 集群运行即可

    //第一步：读取文件，解析成 key,value 对，这里是 k1 v1

    job.setInputFormatClass(TextInputFormat.class);

    //集群运行模式，从 hdfs 上面读取文件

    //
    TextInputFormat.addInputPath(job, new
    Path("hdfs://node01:8020/wordcount"));

    //使用本地模式来运行，从本地磁盘读取文件进行处理

    TextInputFormat.addInputPath(job, new
    Path("file:///F:\\input"));

    //第二步：自定义 map 逻辑，接收第一步的 k1,v1 转换成新的 k2 v2
    进行输出

    job.setMapperClass(WordCountMapper.class);

    //设置我们 key2 的类型

    job.setMapOutputKeyClass(Text.class);

    //设置我们的 v2 类型

    job.setMapOutputValueClass(IntWritable.class);

    /**
     * 第三到六步
     * 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
     * 第四步：排序
     * 第五步：规约
    
```

```

    * 第六步：分组
    * 都省掉
    */
    //第七步：设置我们的 reduce 类，接受我们的 key2 v2 输出我们 k3
    v3

    job.setReducerClass(WordCountReducer.class);

    //设置我们 key3 输出的类型
    job.setOutputKeyClass(Text.class);

    //设置我们 value3 的输出类型
    job.setOutputValueClass(IntWritable.class);

    //第八步：设置我们的输出类 outputformat

    job.setOutputFormatClass(TextOutputFormat.class);

    //输出路径输出到 hdfs 上面去，表示我打包到集群上面去运行
    //
    TextOutputFormat.setOutputPath(job, new
    Path("hdfs://node01:8020/wordcountout"));

    //使用本地模式来运行
    TextOutputFormat.setOutputPath(job, new
    Path("file:///F:\\output"));

    //提交我们的任务

    boolean b = job.waitForCompletion(true);

    return b?0:1;

}

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

    Configuration configuration = new Configuration();

    //提交我们的 job 任务

    //任务完成之后，返回一个状态码值，如果状态码值是 0，表示程序运

```


行成功

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

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

```
}
```

```
}
```

The screenshot shows an IDE with a project named 'day08'. The package structure is 'com.qst.wordcount'. The 'JobMain.java' file is open, showing the following code:

```
1 package com.qst.wordcount;
2 import ...
12 public class JobMain extends Configured implements Tool {
13     @Override
14     public int run(String[] args) throws Exception {
15         //获取一个job对象，用于我们任务的组织，通过job对象将我们八个步骤组织到一起，提交给yarn集群运行
16         Job job = Job.getInstance(super.getConf(), "jobName: 'xxx'");
17         //如果需要打包运行，一定要加上这一句
18         job.setJarByClass(JobMain.class);
19         //获取到我们的job对象之后，通过job对象来组织我们的八个class类到一起，然后提交给yarn集群运行即可
20         //第一步：读取文件，解析成key,value对，这里是k1 v1
21         job.setInputFormatClass(TextInputFormat.class);
22         //集群运行模式，从hdfs上面读取文件
23         // TextInputFormat.addInputPath(job, new Path("hdfs://node01:8020/wordcount"));
24     }
25 }
```

The output window shows the following logs:

```
19/07/08 06:22:29 INFO mapred.JobClient: FILE: Number of bytes written=340588
19/07/08 06:22:29 INFO mapred.JobClient: FILE: Number of read operations=0
19/07/08 06:22:29 INFO mapred.JobClient: FILE: Number of large read operations=0
19/07/08 06:22:29 INFO mapred.JobClient: FILE: Number of write operations=0
19/07/08 06:22:29 INFO mapred.JobClient: Map-Reduce Framework
19/07/08 06:22:29 INFO mapred.JobClient: Map input records=21
19/07/08 06:22:29 INFO mapred.JobClient: Map output records=30
19/07/08 06:22:29 INFO mapred.JobClient: Map output bytes=240
19/07/08 06:22:29 INFO mapred.JobClient: Input split bytes=93
19/07/08 06:22:29 INFO mapred.JobClient: Combine input records=0
19/07/08 06:22:29 INFO mapred.JobClient: Combine output records=0
19/07/08 06:22:29 INFO mapred.JobClient: Reduce input groups=9
19/07/08 06:22:29 INFO mapred.JobClient: Reduce shuffle bytes=0
19/07/08 06:22:29 INFO mapred.JobClient: Reduce input records=30
19/07/08 06:22:29 INFO mapred.JobClient: Reduce output records=9
19/07/08 06:22:29 INFO mapred.JobClient: Spilled Records=60
19/07/08 06:22:29 INFO mapred.JobClient: Total committed heap usage (bytes)=458227712
Process finished with exit code 0
```

此电脑 > Software (F:) > output

名称	修改日期	类型	大小
._SUCCESS.crc	2019/7/8 6:43	CRC 文件	1 KB
.part-r-00000.crc	2019/7/8 6:43	CRC 文件	1 KB
._SUCCESS	2019/7/8 6:43	文件	0 KB
part-r-00000	2019/7/8 6:43	文件	1 KB

part-r-00000

1	flume	1
2	hadoop	2
3	hbase	4
4	hive	4
5	kafka	2
6	spark	2
7	storm	3
8		

提醒：本地运行完成之后，就可以打成 jar 包放到服务器上面去运行了，实际工

作当中，都是将代码打成 jar 包，开发 main 方法作为程序的入口，然后放到集群上面去运行