OutputFormat

|  |  |
| --- | --- |
| **Objective:** | Write a custom OutputFormat class. |
| **Location of Files:** | ~/materials/data |
| **Successful Outcome:** | A MapReduce jobs that outputs the growth (or loss) of stock dividends. |
| **Before You Begin:** | Open Eclipse in your lab instance. |
| **JAR File** | dividend.jar |
| **Eclipse Project** | custom\_sorting |
| **Exercise directory** | ~/workspace/custom\_sorting |

1.   Open the **custom\_sorting** project in Eclipse.

2.      Define a Custom OutputFormat Class

3.   Open **DividendDriver.java** and look in the run method. What is the current Output Format of this job?

4.   You are going to replace the current Output Format with a custom class that you define. Start by adding a new class to the customsort package named DividendOutputFormat that extends FileOuputFormat.

5.   What should the data types of the K and V generics be?

6.   Change K to the appropriate data type, and change V to the appropriate data type.

7.   Use Eclipse to generate the unimplemented getRecordWriter method from the parent class FileOutputFormat.

8.   Change the parameter name of **TaskAttemptContext** from arg0 to context.

9.   Within getRecordWriter, create a file to write the output to, using the job name and the task ID as part of the filename:

int partition = context.getTaskAttemptID().getTaskID().getId();  
Path outputDir =FileOutputFormat.getOutputPath(context);  
Path file = new Path(outputDir.getName()+ Path.SEPARATOR+ context.getJobName()+ "\_" + partition);   
FileSystem fs =file.getFileSystem(context.getConfiguration());   
FSDataOutputStream fileOut = fs.create(file);

NOTE: Using the task ID is a clever trick because we need to ensure the output files each have a unique name for each Reducer. Otherwise, you will get an exception at runtime if two Reducers try to write output to the same filename.

10.   Pass the FSDataOutputStream object to a new DividendRecordWriter and return it:

return new DividendRecordWriter(fileOut);

11.   Save your changes to **DividendOutputFormat.java**. You will have a compiler error because the DividendRecordWriter class does not exist yet, but you will define that next.

12.      Define a Custom RecordWriter

13.   Add a new static inner class to DividendOutputFormat named DividendRecordWriter that extends RecordWriter.

14.   Change the key generic K to NullWritable and the value generic V to DividendChange.

15.   Add two fields to DividendRecordWriter:

public final String SEPARATOR = ",";

private DataOutputStream out;

16.   Add a constructor to DividendRecordWriter that has a DataOutputStream parameter and assigns it to the out field.

17.   Let Eclipse generate the unimplemented close and write methods for your DividendRecordWriter class.

18.   In the close method, close the output stream of the out field.

19.   Change the parameter names of the write method from arg0 and arg1 to key and value, respectively.

20.     In the write method, instantiate a new StringBuilder object:

StringBuilder result = new StringBuilder();

21.   Using the append method of StringBuilder and the data in the DividendChange parameter, build a string that looks like:

ABC,2012-07-14,0.045

22.   Append a newline character to the end of result.

23.   Use the write method of the out parameter to write out the StringBuilder instance:

out.write(result.toString().getBytes());

24.   Save your changes to **DividendOutputFormat.java**.

25.   Modify the Job

26.   Open **DividendDriver.java**.

27.   In the **run()** method, change the output format of this job from TextOutputFormat to DividendOutputFormat.

28.   Set the number of reduce tasks to 3.

29.   Save your changes to **DividendDriver.java**.

30.    Run the Job

31.   Build the project to create **dividend.jar**

32.   Run the job:

$ yarn jar dividendgrowth.jar

33.   When the job is finished, you should see 3 files named DividendJob\_# in the growth folder of HDFS.

-rw-r--r-- 1 [username] hdfs 76138 growth/DividendJob\_0

-rw-r--r-- 1 [username] hdfs 72193 growth/DividendJob\_1

-rw-r--r-- 1 [username] hdfs 89279 growth/DividendJob\_2

34.   Where did these filenames come from?

35.   View the contents of DividendJob\_0:

$ hdfs dfs -cat growth/DividendJob\_0  
 ...AZZ,2000-03-08,0.065 AZZ,2001-03-08,-0.06 AZZ, 2001-04-10,0.06AZZ,2010-01-28,0.16999999999999998

Notice the “A” stocks are in DividendJob\_0, the “B” stocks are in DividendJob\_1, and the “C” stocks are in DividendJob\_2.

36.      Configure Multiple Outputs

37.   In this step, you will learn how to use the MulitpleOutputs class to configure a Reducer to send output to multiple files. Start by commenting out the line of code in the run method of DividendJob where you set the output format to DividendOutputFormat:

//job.setOutputFormatClass(DividendOutputFormat.class);

38.   In the run method, define two named outputs, one for positive growth dividends and one for negative growth:

MultipleOutputs.addNamedOutput(job, "positive",TextOutputFormat.class, NullWritable.class, DividendChange.class);

MultipleOutputs.addNamedOutput(job, "negative",TextOutputFormat.class, NullWritable.class, DividendChange.class);

39.  Add a field of type MultipleOutputs<NullWritable, DividendChange> named mos to the DividendGrowthReducer class.

40. Initialize the mos field in the setup method of DividendGrowthReducer.

41. In the reduce method of DividendGrowthReducer, comment the line of code where you write out the key/value pair using context. Instead, write out the key/value pairs to the named outputs based on whether the growth is positive or negative:

if(growth > 0) {

mos.write("positive",outputKey,outputValue,"pos");

} else {

mos.write("negative",outputKey,outputValue,"neg");

}

42. Based on the code above, what will the names of the output files be when this code executes?

43.   Add the cleanup method to DividendGrowthReducer. Within cleanup, invoke the close method on mos.

IMPORTANT: If you notice your MapReduce jobs that use a MultipleOutputs format is generating empty files, it is probably because you forgot to close them!

44. Run the Job Again

45. Save your changes to DividendDriver, and then build and run the job again.

46.   Look in the **growth** folder for two sets of files from each reduer: **neg-r-0000#** and **pos-r-0000#**. Verify that all the positive growths appear in the **pos-r-0000#** files and all the negative growths appear in **neg-r-0000#**.

47.      Remove the Empty Output File

48.   Look at the contents of **part-r-00000** in the **growth** folder of the **CustomSort** project. Why does this file exist, and why is it empty?

49.   You can avoid the empty files getting created by using the LazyOutputFormat class. Add the following line of code to the run method of DividendJob:

LazyOutputFormat.setOutputFormatClass(job, TextOutputFormat.class);

50.   Save your changes to **DividendDriver.java** and build and run the program again.

51.   Refresh the **growth** folder and the **part-r-0000#** files should not appear this time.

Result: In this lab, you wrote a custom output format that outputs DividendChange objects. You learned how to specify the file name and write a custom RecordWriter. You also saw how to use the MulitpleOutputs class to output data to multiple files.

**END**