IO Format Class

Now, We have already understand Input AND Output Formats.

Now, We will seen Input Class and Output Class function.

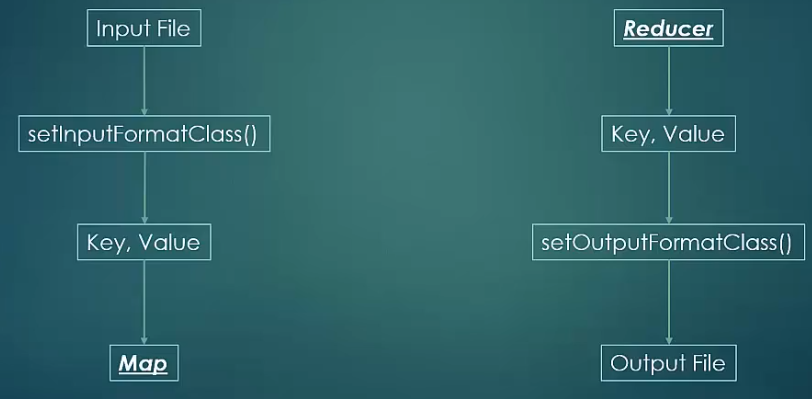
Lets Understand main idea behind the function. The map gets its input as the form of (key, Value) pair.

\*\* Data Types of keys and Value are defined in mapper Class Definition.

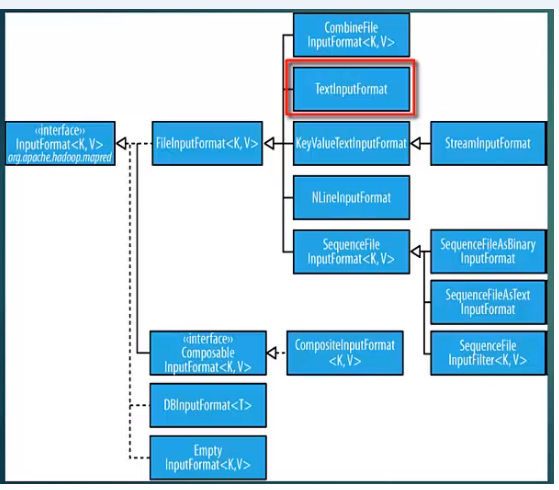
So, Hadoop has to supply Key, Value pair as it reads the data from the file.

* Programmers controls this reading mechanism and key value parsing by using **set Input Format** Class Function.

Similarly When Reducer emits the key value pair then in this case it is the **SetOutputFormat** Function which gives programmers Control over how it is be written to the Output File.



Now, Lets See various Input Formats:::



1. **Combine File Input Format Class:::** This is Used in the cases where Many Small files needed to be combined as a Input.

* **The Problem with many Files as a Input is that Advantage of Data Locality is being Lost.**
* Combine file input format class has little Advantage that It is an Inbuilt Mechanism of considering Data Locality so it still performs good with many files.
* Although the case are there where multiple input Files is not good case for map reduce for performance perspective and should be avoided.
* Combine file input Format is an abstract Class and Would be needed to be customized as per the scenario.

1. **Text Input Format::**It supplies the map key as long writable which is byte offset from the start of the file.

**And the Value as text line which excluding any line terminator.**

\*\*\* This is Default Format\*\*\*

1. **Key Value Text Input Format**

This is Used in the case where Keys are already Present in the input file and Key, Value is separated be a Delimiter.

\*\* Delimiter is by Default TAB character.

But It can be customized through

mapreduce.input.keyvaluelinerecordreader.key.value.seperator

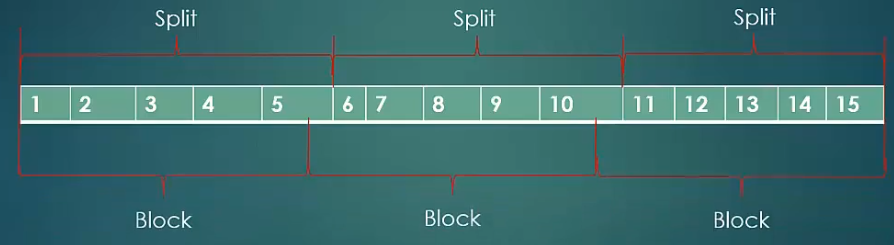
1. Nlineinput Format:

This divides the input into splits with fixed no of lines.

So, If No of lines= 5

Then Every map will be distributed with 5 no of Lines as a Input.

* **Ex this can happen so that the input splits may cross over HDFS block Size**



\*\* In this case Suppose split size ends at 5th Record and value of that record crossover the boundary of the block and other block is available at the other mapper Machine.

\*\* So In this case portion of the record which is not locally present would be fetched from the network.

This loss in the data locality costs less than 1 % of the Overhead Time

**100 Mb= 2 Data Blocks in HDFS**

**In The processing Part::**

**Map - Reduce**

**Map will Start::**

**We need key , Value for that**

**we need to use Input Formats::**

**Suppose we have used Nline input Format then It will Converts all the input as a Line no as key and Value as REcords.**

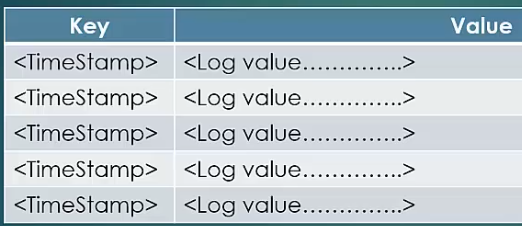
**Line =5**

**Key Value**

1. Sequence File Input Format::

Sequence file is special flat file which consisting of binary encoded key value Pair.

It looks like



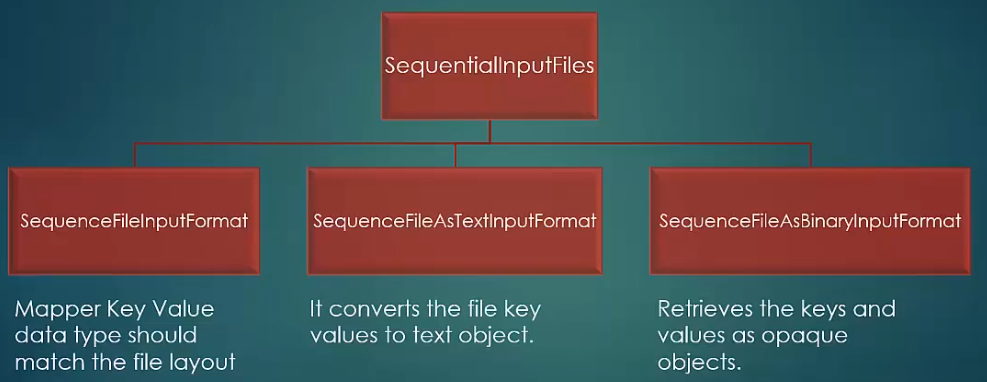
These are special file and Directly can’t be processed as Text Objects.

These files have **sync Point mechanism** and are compressible.

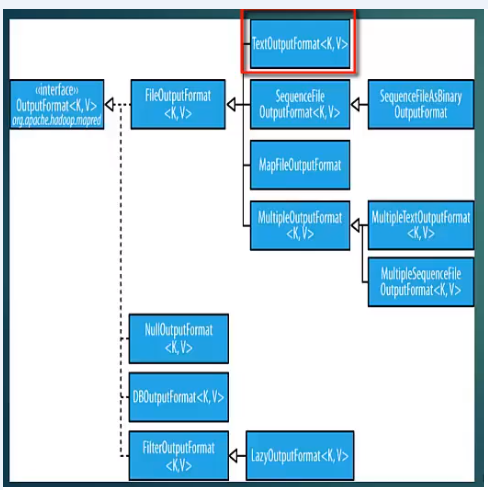
\*\* Sequence files are mostly used in the scenario where Output of one map reduce Job is taken as INPUT of other Map reduce Job.

These are good with the sorting as well so the intermediate map results are returned in the sequence file format. So to provide ease of sorting before the results are fed into reducer.

\*\* So, There are 3 file formats which processes Sequence Files::



**OUTPUT FILE FORMATS**



1. Text Output Format --- Default Output Format…

In this Output Format Keys and Values converted to strings. The keys and values are separated by Delimiter which canbe controlled using the properties::

“ Mapreduce.output.textoutputformat.seperator”

By Default Tab character is being used for Delimiter.

1. Sequence File Output Format::

Sequence file is special flat file which consisting of binary encoded key value Pair.

1. MapFileOutputFormat::

Map Files are special Sequence File with Index Lookup.

First is Map File which contains the data and second file is the Index File which is used for look Up the Data.

1. MultipleOutputFormat::

This Output Format is being used in the scenario when there is multiple output File is needed.

It provides greater control over the output File name.

Refer the notes : Hadoop.apache.org/docs

**Experiment with File Output Format**

We create multiple file per reducer for word count Job..

I.e Ouput Segregated alphabetically

I.e. It puts all the word starting with A in first file

And ------------------------------------------- B in other File

And --------------------------------------------c in other File and So on … etc…

Program::

//learning MapReduce

// Word Count Reducer - Multiple Output Variant

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.output.MultipleOutputs;

public class MultipleOutputReducer

extends Reducer<Text, IntWritable, Text, IntWritable> {

private MultipleOutputs<Text, IntWritable> multipleOutputs;

@Override

public void setup(Context context)

throws IOException, InterruptedException {

multipleOutputs = new MultipleOutputs<Text, IntWritable>(context);

}

@Override

public void reduce(Text key, Iterable<IntWritable> values, Context context)

throws IOException, InterruptedException {

int sum = 0;

for (IntWritable value : values) {

sum += value.get();

}

multipleOutputs.write(key, new IntWritable(sum), key.toString().substring(0, 1));

}

@Override

public void cleanup(Context context)

throws IOException, InterruptedException {

multipleOutputs.close();

}

}

// Learning MapReduce

// Word Count driver - Multiple output variant

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.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class MultiOutWordCount extends Configured implements Tool {

@Override

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

Job job = new Job(getConf());

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.setMapperClass(WordCountMapper.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

job.setReducerClass(MultipleOutputReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

return job.waitForCompletion(true) ? 0 : 1;

}

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

int exitCode = ToolRunner.run(new MultiOutWordCount(), args);

System.exit(exitCode);

}

}

\*\* In Reducer Class we declare the private object Type of type multiple Output

Then we override setup and refer with the context object.

And Then we write it

multipleOutput.write (---key, new IntWritable(sum), key.toString().substring(0, 1));---------- )

AND the we write the Key , Value and 3rd is filename.