TopWordFinderTopologyPartA.java

import backtype.storm.Config;

import backtype.storm.LocalCluster;

import backtype.storm.StormSubmitter;

import backtype.storm.topology.BasicOutputCollector;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.TopologyBuilder;

import backtype.storm.topology.base.BaseBasicBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

/\*\*

\* This topology counts the words from sentences emmited from a random sentence spout.

\*/

public class TopWordFinderTopologyPartA {

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

TopologyBuilder builder = new TopologyBuilder();

Config config = new Config();

config.setDebug(true);

/\*

----------------------TODO-----------------------

Task: wire up the topology

NOTE:make sure when connecting components together, using the functions setBolt(name,…) and setSpout(name,…),

you use the following names for each component:

RandomSentanceSpout -> "spout"

SplitSentenceBolt -> "split"

WordCountBolt -> "count"

------------------------------------------------- \*/

config.setMaxTaskParallelism(3);

LocalCluster cluster = new LocalCluster();

cluster.submitTopology("word-count", config, builder.createTopology());

//wait for 60 seconds and then kill the topology

Thread.sleep(60 \* 1000);

cluster.shutdown();

}

}

TopWordFinderTopologyPartB.java

import backtype.storm.Config;

import backtype.storm.LocalCluster;

import backtype.storm.StormSubmitter;

import backtype.storm.topology.BasicOutputCollector;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.TopologyBuilder;

import backtype.storm.topology.base.BaseBasicBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

/\*\*

\* This topology reads a file and counts the words in that file

\*/

public class TopWordFinderTopologyPartB {

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

TopologyBuilder builder = new TopologyBuilder();

Config config = new Config();

config.setDebug(true);

/\*

----------------------TODO-----------------------

Task: wire up the topology

NOTE:make sure when connecting components together, using the functions setBolt(name,…) and setSpout(name,…),

you use the following names for each component:

FileReaderSpout -> "spout"

SplitSentenceBolt -> "split"

WordCountBolt -> "count"

------------------------------------------------- \*/

config.setMaxTaskParallelism(3);

LocalCluster cluster = new LocalCluster();

cluster.submitTopology("word-count", config, builder.createTopology());

//wait for 2 minutes and then kill the job

Thread.sleep( 2 \* 60 \* 1000);

cluster.shutdown();

}

}

TopWordFinderTopologyPartC.java

import backtype.storm.Config;

import backtype.storm.LocalCluster;

import backtype.storm.StormSubmitter;

import backtype.storm.topology.BasicOutputCollector;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.TopologyBuilder;

import backtype.storm.topology.base.BaseBasicBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

/\*\*

\* This topology reads a file, splits the senteces into words, normalizes the words such that all words are

\* lower case and common words are removed, and then count the number of words.

\*/

public class TopWordFinderTopologyPartC {

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

TopologyBuilder builder = new TopologyBuilder();

Config config = new Config();

config.setDebug(true);

/\*

----------------------TODO-----------------------

Task: wire up the topology

NOTE:make sure when connecting components together, using the functions setBolt(name,…) and setSpout(name,…),

you use the following names for each component:

FileReaderSpout -> "spout"

SplitSentenceBolt -> "split"

WordCountBolt -> "count"

NormalizerBolt -> "normalize"

------------------------------------------------- \*/

config.setMaxTaskParallelism(3);

LocalCluster cluster = new LocalCluster();

cluster.submitTopology("word-count", config, builder.createTopology());

//wait for 2 minutes then kill the job

Thread.sleep(2 \* 60 \* 1000);

cluster.shutdown();

}

}

TopWordFinderTopologyPartD.java

import backtype.storm.Config;

import backtype.storm.LocalCluster;

import backtype.storm.StormSubmitter;

import backtype.storm.topology.BasicOutputCollector;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.TopologyBuilder;

import backtype.storm.topology.base.BaseBasicBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

/\*\*

\* This topology reads a file and counts the words in that file, then finds the top N words.

\*/

public class TopWordFinderTopologyPartD {

private static final int N = 10;

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

TopologyBuilder builder = new TopologyBuilder();

Config config = new Config();

config.setDebug(true);

/\*

----------------------TODO-----------------------

Task: wire up the topology

NOTE:make sure when connecting components together, using the functions setBolt(name,…) and setSpout(name,…),

you use the following names for each component:

FileReaderSpout -> "spout"

SplitSentenceBolt -> "split"

WordCountBolt -> "count"

NormalizerBolt -> "normalize"

TopNFinderBolt -> "top-n"

------------------------------------------------- \*/

config.setMaxTaskParallelism(3);

LocalCluster cluster = new LocalCluster();

cluster.submitTopology("word-count", config, builder.createTopology());

//wait for 2 minutes and then kill the job

Thread.sleep(2 \* 60 \* 1000);

cluster.shutdown();

}

}

WordCountBolt.java

import backtype.storm.topology.BasicOutputCollector;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.base.BaseBasicBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

import java.util.HashMap;

import java.util.Map;

public class WordCountBolt extends BaseBasicBolt {

Map<String, Integer> counts = new HashMap<String, Integer>();

@Override

public void execute(Tuple tuple, BasicOutputCollector collector) {

String word = tuple.getString(0);

Integer count = counts.get(word);

if (count == null)

count = 0;

count++;

counts.put(word, count);

collector.emit(new Values(word, count));

}

@Override

public void declareOutputFields(OutputFieldsDeclarer declarer) {

declarer.declare(new Fields("word", "count"));

}

}

FileReaderSpout.java

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.Map;

import backtype.storm.spout.SpoutOutputCollector;

import backtype.storm.task.TopologyContext;

import backtype.storm.topology.IRichSpout;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Values;

public class FileReaderSpout implements IRichSpout {

private SpoutOutputCollector \_collector;

private TopologyContext context;

@Override

public void open(Map conf, TopologyContext context,

SpoutOutputCollector collector) {

/\*

----------------------TODO-----------------------

Task: initialize the file reader

------------------------------------------------- \*/

this.context = context;

this.\_collector = collector;

}

@Override

public void nextTuple() {

/\*

----------------------TODO-----------------------

Task:

1. read the next line and emit a tuple for it

2. don't forget to sleep when the file is entirely read to prevent a busy-loop

------------------------------------------------- \*/

}

@Override

public void declareOutputFields(OutputFieldsDeclarer declarer) {

declarer.declare(new Fields("word"));

}

@Override

public void close() {

/\*

----------------------TODO-----------------------

Task: close the file

------------------------------------------------- \*/

}

@Override

public void activate() {

}

@Override

public void deactivate() {

}

@Override

public void ack(Object msgId) {

}

@Override

public void fail(Object msgId) {

}

@Override

public Map<String, Object> getComponentConfiguration() {

return null;

}

}

NormalizerBolt.java

import backtype.storm.topology.BasicOutputCollector;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.base.BaseBasicBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

import java.util.Arrays;

import java.util.List;

/\*\*

\* A bolt that normalizes the words, by removing common words and making them lower case.

\*/

public class NormalizerBolt extends BaseBasicBolt {

private List<String> commonWords = Arrays.asList("the", "be", "a", "an", "and",

"of", "to", "in", "am", "is", "are", "at", "not", "that", "have", "i", "it",

"for", "on", "with", "he", "she", "as", "you", "do", "this", "but", "his",

"by", "from", "they", "we", "her", "or", "will", "my", "one", "all", "s", "if",

"any", "our", "may", "your", "these", "d" , " ", "me" , "so" , "what" , "him" );

@Override

public void execute(Tuple tuple, BasicOutputCollector collector) {

/\*

----------------------TODO-----------------------

Task:

1. make the words all lower case

2. remove the common words

------------------------------------------------- \*/

}

@Override

public void declareOutputFields(OutputFieldsDeclarer declarer) {

declarer.declare(new Fields("word"));

}

}

RandomSentenceSpout.java

import backtype.storm.spout.SpoutOutputCollector;

import backtype.storm.task.TopologyContext;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.base.BaseRichSpout;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Values;

import backtype.storm.utils.Utils;

import java.util.Map;

import java.util.Random;

public class RandomSentenceSpout extends BaseRichSpout {

SpoutOutputCollector \_collector;

Random \_rand;

@Override

public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {

\_collector = collector;

\_rand = new Random();

}

@Override

public void nextTuple() {

Utils.sleep(100);

String[] sentences = new String[]{ "the cow jumped over the moon", "an apple a day keeps the doctor away",

"four score and seven years ago", "snow white and the seven dwarfs", "i am at two with nature" };

String sentence = sentences[\_rand.nextInt(sentences.length)];

\_collector.emit(new Values(sentence));

}

@Override

public void ack(Object id) {

}

@Override

public void fail(Object id) {

}

@Override

public void declareOutputFields(OutputFieldsDeclarer declarer) {

declarer.declare(new Fields("word"));

}

}

SplitSentenceBolt.java

import backtype.storm.topology.BasicOutputCollector;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.base.BaseBasicBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

public class SplitSentenceBolt extends BaseBasicBolt {

@Override

public void execute(Tuple tuple, BasicOutputCollector collector) {

String sentence = tuple.getString(0);

String[]words=sentence.split("[\\s~`!@#$%^&\*(-)+=\_:;'\",.<>?/\\\\0-9"+"\\]\\[\\}\\{]+");

for(String word:words){

collector.emit(new Values(word));

}

}

@Override

public void declareOutputFields(OutputFieldsDeclarer declarer) {

declarer.declare(new Fields("word"));

}

}

TopNFinderBolt.java

import backtype.storm.topology.BasicOutputCollector;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.base.BaseBasicBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

import java.util.HashMap;

/\*\*

\* a bolt that finds the top n words.

\*/

public class TopNFinderBolt extends BaseBasicBolt {

private HashMap<String, Integer> currentTopWords = new HashMap<String, Integer>();

private int N;

private long intervalToReport = 20;

private long lastReportTime = System.currentTimeMillis();

public TopNFinderBolt(int N) {

this.N = N;

}

@Override

public void execute(Tuple tuple, BasicOutputCollector collector) {

/\*

----------------------TODO-----------------------

Task: keep track of the top N words

------------------------------------------------- \*/

//reports the top N words periodically

if (System.currentTimeMillis() - lastReportTime >= intervalToReport) {

collector.emit(new Values(printMap()));

lastReportTime = System.currentTimeMillis();

}

}

@Override

public void declareOutputFields(OutputFieldsDeclarer declarer) {

declarer.declare(new Fields("top-N"));

}

public String printMap() {

StringBuilder stringBuilder = new StringBuilder();

stringBuilder.append("top-words = [ ");

for (String word : currentTopWords.keySet()) {

stringBuilder.append("(" + word + " , " + currentTopWords.get(word) + ") , ");

}

int lastCommaIndex = stringBuilder.lastIndexOf(",");

stringBuilder.deleteCharAt(lastCommaIndex + 1);

stringBuilder.deleteCharAt(lastCommaIndex);

stringBuilder.append("]");

return stringBuilder.toString();

}

}