**中间件实现技术课程实验报告**

课程名称：中间件实现技术

项目名称：Flink图书实时热销排名

学 院：信息学院

专 业：电子信息

指导老师：胡鹤

报 告 人：王培培

学 号: 2020104250

时 间：2021.11.7

|  |
| --- |
| **一、实验背景和目的：**  实时处理是指从数据产生到根据该数据计算的结果产生之间的这段延迟可以满足业务的需求, 假如业务需求是延迟不超过10ms，而你的处理延迟为15ms，就不能算实时处理，而假如业务要求处理数据的延迟为30min，而你的数据可以在20min内计算出来，这也算实时处理。  Flink是集流式处理和批处理于一身的统一的计算引擎，Flink认为，流处理是常态，批处理是流处理的一种特殊情况，在无界流中去某段范围内的全量数据进行处理就是批处理，于是Flink把核心放在流处理上，实现了一些高级的也是必须的功能，例如数据状态、事件时间、分布式快照、watermark等。  同时Flink具备以下特性：1高吞吐，低延迟，高性能；2高度灵活的流式窗口；  3状态计算的 Exactly-once 语义；4轻量级的容错机制；5支持 EventTime 及乱序事件；6流批统一引擎。  所以本项目我们采用Flink作为实时计算引擎  同时我们使用Kafka作为消息队列，Kafka 是一个比较早的消息队列，但是它是一个非常稳定的消息队列。  本项目我们使用Kafka模拟用户点击购买事件。  实验目的：每隔5秒钟输出最近一小时（实验使用10min）内点击量最多的前 N（这里选取3个）个商品/图书。  二、实验平台：  Kubernetes:1.21.2  docker:20.10.7  Flink:1.8.0  Kafka:2.3.1  Java:1.8  三、实验过程和主要代码介绍：  需求分析   1. 告诉 Flink 框架基于时间做窗口，我们这里用processingTime，不用自带时间戳   2）过滤出图书点击行为数据  3）按一小时的窗口大小，每5秒钟统计一次，做滑动窗口聚合（Sliding Window）  4）聚合，输出窗口中点击量前N名的商品  代码实现  向Kafka发消息模拟购买事件  package cn.edu.neu.tiger; import org.apache.flink.api.common.serialization.SimpleStringSchema; import org.apache.flink.streaming.api.datastream.DataStreamSource; import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment; import org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011; //import org.apache.flink.streaming.util.serialization.KeyedSerializationSchemaWrapper;  import java.util.Properties;  */\*\*  \* Created by on 2019/1/14.  \*/* public class KafkaProducer {    public static void main(String[] args) throws Exception{   StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();   DataStreamSource<String> text = env.addSource(new MyNoParalleSource()).setParallelism(1);   Properties properties = new Properties();  properties.setProperty("bootstrap.servers", "10.96.39.90:9092");  //new FlinkKafkaProducer("topn",new KeyedSerializationSchemaWrapper(new SimpleStringSchema()),properties,FlinkKafkaProducer.Semantic.EXACTLY\_ONCE);  FlinkKafkaProducer011<String> producer = new FlinkKafkaProducer011("topn",new SimpleStringSchema(),properties); /\*  //event-timestamp事件的发生时间  producer.setWriteTimestampToKafka(true); \*/  text.addSink(producer);  env.execute();  } }//  其中的：MyNoParalleSource 是作者自己实现的一个并行度为1的发送器，用来向kafka发送数据：  package cn.edu.neu.tiger;  import org.apache.flink.api.common.functions.MapFunction; import org.apache.flink.streaming.api.datastream.DataStream; import org.apache.flink.streaming.api.datastream.DataStreamSource; import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment; import org.apache.flink.streaming.api.functions.source.SourceFunction; import org.apache.flink.streaming.api.windowing.time.Time;  import java.util.ArrayList; import java.util.List; import java.util.Random;  */\*\*  \* Created by wangzhiwu on 2019/1/12.  \*/* //使用并行度为1的source public class MyNoParalleSource implements SourceFunction<String> {//1   //private long count = 1L;  private boolean isRunning = true;   */\*\*  \* 主要的方法  \* 启动一个source  \* 大部分情况下，都需要在这个run方法中实现一个循环，这样就可以循环产生数据了  \*  \** ***@param*** *ctx  \** ***@throws*** *Exception  \*/* @Override  public void run(SourceContext<String> ctx) throws Exception {  while(isRunning){  //图书的排行榜  List<String> books = new ArrayList<>();  books.add("Pyhton从入门到放弃");//10  books.add("Java从入门到放弃");//8  books.add("Php从入门到放弃");//5  books.add("C++从入门到放弃");//3  books.add("Scala从入门到放弃");//0-4  int i = new Random().nextInt(5);  ctx.collect(books.get(i));   //每1秒产生一条数据  Thread.sleep(1000);  }  }  //取消一个cancel的时候会调用的方法  @Override  public void cancel() {  isRunning = false;  } }  可见，我们每过1秒向Kafka的topn这个topic随机发送一本书的名字用来模拟购买行为。  整体实现代码如下：  package cn.edu.neu.tiger;  import org.apache.flink.api.common.functions.FlatMapFunction; import org.apache.flink.api.common.functions.MapFunction; import org.apache.flink.api.common.serialization.SimpleStringSchema; import org.apache.flink.api.java.tuple.Tuple2; import org.apache.flink.streaming.api.TimeCharacteristic; import org.apache.flink.streaming.api.datastream.DataStream; import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment; import org.apache.flink.streaming.api.functions.windowing.ProcessAllWindowFunction; import org.apache.flink.streaming.api.windowing.assigners.SlidingProcessingTimeWindows; import org.apache.flink.streaming.api.windowing.assigners.TumblingProcessingTimeWindows; import org.apache.flink.streaming.api.windowing.time.Time; import org.apache.flink.streaming.api.windowing.windows.TimeWindow; import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer011; import org.apache.flink.util.Collector;  import java.sql.Timestamp; import java.util.Comparator; import java.util.Map; import java.util.Properties; import java.util.TreeMap;  */\*\*  \* Created by on 2019/1/17.  \*/* public class TopN {   public static void main(String[] args) throws Exception{   */\*\*  \*  \* 书1 书2 书3  \* （书1,1） (书2，1) （书3,1）  \*  \*  \*/* //每隔5秒钟 计算过去1小时 的 Top 3 商品  StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();   env.setParallelism(1);   env.setStreamTimeCharacteristic(TimeCharacteristic.ProcessingTime); //以processtime作为时间语义    Properties properties = new Properties();  properties.setProperty("bootstrap.servers", "10.96.39.90:9092");  FlinkKafkaConsumer011<String> input = new FlinkKafkaConsumer011<>("topn", new SimpleStringSchema(), properties);   //从最早开始消费 位点  input.setStartFromEarliest();    DataStream<String> stream = env  .addSource(input);   DataStream<Tuple2<String, Integer>> ds = stream  .flatMap(new LineSplitter()); //将输入语句split成一个一个单词并初始化count值为1的Tuple2<String, Integer>类型    DataStream<Tuple2<String, Integer>> wcount = ds  .keyBy(0)  .window(SlidingProcessingTimeWindows.of(Time.seconds(600),Time.seconds(5)))  //key之后的元素进入一个总时间长度为600s,每5s向后滑动一次的滑动窗口  .sum(1);// 将相同的key的元素第二个count值相加   wcount  .windowAll(TumblingProcessingTimeWindows.of(Time.seconds(5)))//(shu1, xx) (shu2,xx)....  //所有key元素进入一个5s长的窗口（选5秒是因为上游窗口每5s计算一轮数据，topN窗口一次计算只统计一个窗口时间内的变化）  .process(new TopNAllFunction(3))  .print(); //redis sink redis -> 接口   env.execute();  }//       private static final class LineSplitter implements  FlatMapFunction<String, Tuple2<String, Integer>> {   public void flatMap(String value, Collector<Tuple2<String, Integer>> out) {  // normalize and split the line  //String[] tokens = value.toLowerCase().split("\\W+");   // emit the pairs  /\*for (String token : tokens) {  if (token.length() > 0) {  out.collect(new Tuple2<String, Integer>(token, 1));  }  }\*/   //（书1,1） (书2，1) （书3,1）  out.collect(new Tuple2<String, Integer>(value, 1));  }  }   private static class TopNAllFunction  extends  ProcessAllWindowFunction<Tuple2<String, Integer>, String, TimeWindow> {   private int topSize = 3;   public TopNAllFunction(int topSize) {   this.topSize = topSize;  }   public void process(   ProcessAllWindowFunction<Tuple2<String, Integer>, String, TimeWindow>.Context arg0,  Iterable<Tuple2<String, Integer>> input,  Collector<String> out) throws Exception {   TreeMap<Integer, Tuple2<String, Integer>> treemap = new TreeMap<Integer, Tuple2<String, Integer>>(  new Comparator<Integer>() {   @Override  public int compare(Integer y, Integer x) {  return (x < y) ? -1 : 1;  }   }); //treemap按照key降序排列，相同count值不覆盖   for (Tuple2<String, Integer> element : input) {  treemap.put(element.f1, element);  if (treemap.size() > topSize) { //只保留前面TopN个元素  treemap.pollLastEntry();  }  }   // for (Map.Entry<Integer, Tuple2<String, Integer>> entry : treemap // .entrySet()) {  out.collect("=================\n热销图书列表:\n"+ new Timestamp(System.currentTimeMillis()) + treemap.toString() + "\n===============\n"); // }   }   }   }//  四、实验结果： |