尚硅谷大数据项目之实时数仓(dws层)

(作者：尚硅谷大数据研发部)

版本：V1.0.0

# 数据写入到dwd

## 把order\_info信息写入到dwd层

前面已经把order\_info的信息写入到了es中,来处理首单问题. 为了进行进一步的计算, 我们把order\_info的信息另外再存一份到kafka, 作为dwd层数, 为dws层的双流join做准备.

resultStream.foreachRDD***(***rdd => ***{***

rdd.cache***()***

*// 写入hbase ..*

*// 写入到es …*

*// 写数据到 kafka 的 dwd 层*

rdd.foreachPartition***(***orderInfoIt => ***{***

**val** producer: KafkaProducer***[***String, String***]*** = MyKafkaUtil.*getKafkaProducer****()***

orderInfoIt.foreach***(***orderInfo => ***{***

producer.send***(*new** ProducerRecord***[***String, String***](*"dwd\_order\_info"**, Serialization.*write****(***orderInfo***)))***

***})***

producer.close***()***

***})***

OffsetManager.*saveOffsets****(***offsetRanges, *groupId*, *topic****)***

***})***

测试dwd\_order\_info 是否有数据

## 准备实时维度表:

### 在phoenix中创建其他需要的维度表

下面4个维度表, 订单详情事实表需要使用

* gmall\_spu\_info

create table gmall\_spu\_info (id varchar primary key, spu\_name varchar)SALT\_BUCKETS = 2

* gmall\_base\_trademark

create table gmall\_base\_trademark (id varchar primary key, tm\_name varchar)SALT\_BUCKETS = 2

* gmall\_base\_category3

create table gmall\_base\_category3 (id varchar primary key, name varchar, category2\_id varchar)SALT\_BUCKETS = 2;

* gmall\_sku\_info

create table gmall\_sku\_info (id varchar primary key, spu\_id varchar, price varchar, sku\_name varchar, tm\_id varchar, category3\_id varchar, create\_time varchar, category3\_name varchar, spu\_name varchar, tm\_name varchar) SALT\_BUCKETS = 2;

### 维度表数据导入到hbase

为了方便管理, 我们使用一个app导入所有的维度表.

前面的一些代码需要重构!

* **在MykafkaUtil中新增可以同时消费多个topic的方法**

**def** getKafkaStream***(***ssc: StreamingContext, groupId: String, topics: Set***[***String***]***, fromOffsets: Map***[***TopicPartition, Long***])*** = ***{***

*// 把 offset 自动提交设置为 false, 我们需要手动提交offset*

*kafkaParams****(*"enable.auto.commit"*)*** = ***(*false**: java.lang.Boolean***)***

*kafkaParams****(*"group.id"*)*** = groupId

KafkaUtils

.*createDirectStream****(***

ssc,

*PreferConsistent*,

*Subscribe****[***String, String***](***topics, *kafkaParams*, fromOffsets***)***

***)***

***}***

* **在OffsetManager中添加可以读取保存多个topic偏移量的方法**

主要为了一个流中可以同时消费多个topic的需求

**// 读取偏移量**

**def** readOffsets***(***groupId: String, **topics: Set[String]**): Map[TopicPartition, Long] = {

**import** scala.collection.JavaConverters.\_

**val** client: Jedis = RedisUtil.getClient

**val** partitionToOffsetMap = topics.map(topic => {

client

.hgetAll***(*s"offset:$**{groupId}**:$**{topic}**"**)

.asScala

.map {

**case** (partition, offset) =>

**new** TopicPartition(topic, partition.toInt) -> offset.toLong

}

.toMap

}).reduce((map1, map2) => {

map1 ++ map2

})

client.close()

partitionToOffsetMap

}

// 保存偏移量

**def** saveOffsets(offsetRanges: ListBuffer[OffsetRange], groupId: String, topics: Set[String]): Unit = {

**if** (offsetRanges.isEmpty) **return**

**val** client: Jedis = RedisUtil.getClient

offsetRanges.foreach(offsetRange => {

println(

**s"""**

**|===========**

**|topic: partition -> offset: $**{offsetRange.topic}**: $**{offsetRange.partition} **-> $**{offsetRange.untilOffset}

**| =========="""**.stripMargin)

client.hset(**s"offset:$**{groupId}**:$**{offsetRange.topic}**"**, offsetRange.partition.toString, offsetRange.untilOffset.toString)

})

client.close()

}

* **升级BaseApp, 得到BaseAppV2**

主要为了满足一个流中同时消费多个topic数据

**package** com.atguigu.gmall.realtime

**import** com.atguigu.gmall.realtime.util.***{***MyKafkaUtil, OffsetManager***}***

**import** org.apache.kafka.clients.consumer.ConsumerRecord

**import** org.apache.kafka.common.TopicPartition

**import** org.apache.spark.SparkConf

**import** org.apache.spark.streaming.dstream.DStream

**import** org.apache.spark.streaming.kafka010.***{***HasOffsetRanges, OffsetRange***}***

**import** org.apache.spark.streaming.***{***Seconds, StreamingContext***}***

**import** scala.collection.mutable.ListBuffer

*/\*\**

*\* Author lzc*

*\* Date 2020/8/26 7:58 下午*

*\*/*

**abstract class** BaseAppV2 ***{***

**var** appName: String

**var** groupId: String

**var** totalCores: Int

**var** topics: Set***[***String***]***

**def** run***(***ssc: StreamingContext, offsetRanges: ListBuffer***[***OffsetRange***]***, sourceStream: DStream***[***ConsumerRecord***[***String, String***]])***

**def** main***(***args: Array***[***String***])***: Unit = ***{***

**val** conf: SparkConf = **new** SparkConf***()***.setMaster***(*s"local[$**totalCores**]"*)***.setAppName***(***appName***)***

**val** ssc: StreamingContext = **new** StreamingContext***(***conf, *Seconds****(***3***))***

**val** fromOffsets: Map***[***TopicPartition, Long***]*** = OffsetManager.*readOffsets****(***groupId, topics***)***

**val** offsetRanges: ListBuffer***[***OffsetRange***]*** = ListBuffer.empty***[***OffsetRange***]***

**val** sourceStream: DStream***[***ConsumerRecord***[***String, String***]]*** = MyKafkaUtil

.*getKafkaStream****(***ssc, groupId, topics, fromOffsets***)***

.transform***(***rdd => ***{***

rdd.cache***()***

offsetRanges.clear

**val** newOffsetRanges: Array***[***OffsetRange***]*** = rdd.asInstanceOf***[***HasOffsetRanges***]***.offsetRanges

offsetRanges ++= newOffsetRanges

rdd

***})***

run***(***ssc, offsetRanges, sourceStream***)***

ssc.start***()***

ssc.awaitTermination***()***

***}***

***}***

* **添加需要的维度表样例类**

**case class** BaseTrademark***(***tm\_id:String , tm\_name:String***)***

**case class** BaseCategory3***(***id: String,

name: String,

category2\_id: String***)***

**case class** SpuInfo***(***id: String, spu\_name: String***)***

**case class** SkuInfo***(***id: String,

spu\_id: String,

price: String,

sku\_name: String,

tm\_id: String,

category3\_id: String,

create\_time: String,

**var** category3\_name: String = **null**,

**var** spu\_name: String = **null**,

**var** tm\_name: String = **null*)***

* **创建导维度表的App**

**package** com.atguigu.gmall.realtime.dwd

**import** java.util.Properties

**import** com.atguigu.gmall.realtime.BaseAppV2

**import** com.atguigu.gmall.realtime.bean.\_

**import** com.atguigu.gmall.realtime.util.OffsetManager

**import** org.apache.kafka.clients.consumer.ConsumerRecord

**import** org.apache.spark.rdd.RDD

**import** org.apache.spark.sql.SparkSession

**import** org.apache.spark.streaming.StreamingContext

**import** org.apache.spark.streaming.dstream.DStream

**import** org.apache.spark.streaming.kafka010.OffsetRange

**import** org.json4s.Formats

**import** org.json4s.jackson.JsonMethods

**import** scala.collection.mutable.ListBuffer

*/\*\**

*\* Author lzc*

*\* Date 2020/8/28 5:30 下午*

*\**

*\* 把所有需要的维度表导入到 hbse 中*

*\* 1. 优化 BaseApp*

*\* 2. 优化 OffsetManage 的设计*

*\*/*

**object** DwdDimApp **extends** BaseAppV2 ***{***

**implicit val** *f* = org.json4s.DefaultFormats

**override var** *appName*: String = **"DwdDimApp"**

**override var** *groupId*: String = **"DwdDimApp"**

**override var** *totalCores*: Int = 2

**override var** *topics*: Set***[***String***]*** = *Set****(***

**"ods\_user\_info"**,

**"ods\_sku\_info"**,

**"ods\_spu\_info"**,

**"ods\_base\_category3"**,

**"ods\_base\_province"**,

**"ods\_base\_trademark"*)***

**def** save***[***A <: Product***](***rdd: RDD***[(***String, String***)]***, topic: String, tableName: String, cols: Seq***[***String***])(*implicit** formats: Formats, mf: scala.reflect.Manifest***[***A***]) {***

**import** org.apache.phoenix.spark.\_

rdd.filter***(***\_.\_1 == topic***)***

.map***(***\_.\_2***)***

.map***(***json => ***{***

JsonMethods.parse***(***json***)***.extract***[***A***](****f*, mf***)***

***})***

.saveToPhoenix***(***

tableName,

cols,

zkUrl = *Option****(*"hadoop102,hadoop1hadoop104:2181"*))***

***}***

**override def** run***(***ssc: StreamingContext,

offsetRanges: ListBuffer***[***OffsetRange***]***,

sourceStream: DStream***[***ConsumerRecord***[***String, String***]])***: Unit = ***{***

**val** spark: SparkSession = SparkSession

.*builder****()***

.config***(***ssc.sparkContext.getConf***)***

.getOrCreate***()***

**import** spark.implicits.\_

sourceStream

.map***(***record => ***(***record.topic***()***, record.value***()))*** *// 因为数据来源于多个 topic, 所以需要知道每条数据所属的 topic*

.foreachRDD***(***rdd => ***{***

rdd.cache***()***

*topics*.foreach ***{***

**case "ods\_user\_info"** =>

*save****[***UserInfo***](***rdd,

**"ods\_user\_info"**,

**"gmall\_user\_info"**,

*Seq****(*"ID"**, **"USER\_LEVEL"**, **"BIRTHDAY"**, **"GENDER"**, **"AGE\_GROUP"**, **"GENDER\_NAME"*))***

**case "ods\_sku\_info"** =>

*save****[***SkuInfo***](***rdd,

**"ods\_sku\_info"**,

**"gmall\_sku\_info"**,

*Seq****(*"ID"**, **"SPU\_ID"**, **"PRICE"**, **"SKU\_NAME"**, **"TM\_ID"**, **"CATEGORY3\_ID"**, **"CREATE\_TIME"**, **"CATEGORY3\_NAME"**, **"SPU\_NAME"**, **"TM\_NAME"*))***

*// 需要和 gmall\_spu\_info gmall\_base\_category3 gmall\_base\_trademark 连接, 然后得到所有字段*

*// 使用 spark-sql 完成*

**import** org.apache.phoenix.spark.\_

**val** url = **"jdbc:phoenix:hadoop102,hadoop103,hadoop104:2181"**

spark.read.jdbc***(***url, **"gmall\_sku\_info"**, **new** Properties***())***.createOrReplaceTempView***(*"sku"*)***

spark.read.jdbc***(***url, **"gmall\_spu\_info"**, **new** Properties***())***.createOrReplaceTempView***(*"spu"*)***

spark.read.jdbc***(***url, **"gmall\_base\_category3"**, **new** Properties***())***.createOrReplaceTempView***(*"category3"*)***

spark.read.jdbc***(***url, **"gmall\_base\_trademark"**, **new** Properties***())***.createOrReplaceTempView***(*"tm"*)***

spark.sql***(***

**"""**

**|select**

**| sku.id as id,**

**| sku.spu\_id spu\_id,**

**| sku.price price,**

**| sku.sku\_name sku\_name,**

**| sku.tm\_id tm\_id,**

**| sku.category3\_id category3\_id,**

**| sku.create\_time create\_time,**

**| category3.name category3\_name,**

**| spu.spu\_name spu\_name,**

**| tm.tm\_name tm\_name**

**|from sku**

**|join spu on sku.spu\_id=spu.id**

**|join category3 on sku.category3\_id=category3.id**

**|join tm on sku.tm\_id=tm.id**

**|"""**.stripMargin***)***

.as***[***SkuInfo***]***

.*rdd*

.saveToPhoenix***(***

**"gmall\_sku\_info"**,

*Seq****(*"ID"**, **"SPU\_ID"**, **"PRICE"**, **"SKU\_NAME"**, **"TM\_ID"**, **"CATEGORY3\_ID"**, **"CREATE\_TIME"**, **"CATEGORY3\_NAME"**, **"SPU\_NAME"**, **"TM\_NAME"*)***,

zkUrl = *Option****(*"hadoop102,hadoop103,hadoop104:2181"*))***

**case "ods\_spu\_info"** =>

*save****[***SpuInfo***](***rdd,

**"ods\_spu\_info"**,

**"gmall\_spu\_info"**,

*Seq****(*"ID"**, **"SPU\_NAME"*))***

**case "ods\_base\_category3"** =>

*save****[***BaseCategory3***](***rdd,

**"ods\_base\_category3"**,

**"gmall\_base\_category3"**,

*Seq****(*"ID"**, **"NAME"**, **"CATEGORY2\_ID"*))***

**case "ods\_base\_province"** =>

*save****[***ProvinceInfo***](***rdd,

**"ods\_base\_province"**,

**"gmall\_province\_info"**,

*Seq****(*"ID"**, **"NAME"**, **"AREA\_CODE"**, **"ISO\_CODE"*))***

**case "ods\_base\_trademark"** =>

*save****[***BaseTrademark***](***rdd,

**"ods\_base\_trademark"**,

**"gmall\_base\_trademark"**,

*Seq****(*"ID"**, **"TM\_NAME"*))***

**case** topic => **throw new** UnsupportedOperationException***(*s"不支持消费此 $*{***topic***}*"*)***

***}***

OffsetManager.*saveOffsets****(***offsetRanges, *groupId*, *topics****)***

***})***

***}***

***}***

# 订单详情表写入dwd层

把订单详情表join sku\_info只有的数据写入到dwd层

# 订单表和订单详情表双流join

# 计算订单明细实付金额分摊

# 数据写入到clickhuse(dws)s