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 所有的表数据在一个流中*

*\* 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

## OrderDetail样例类

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

*/\*\**

*\* Author lzc*

*\* Date 2020/8/29 7:43 上午*

*\*/*

**case class** OrderDetail***(***id: Long,

order\_id: Long,

sku\_id: Long,

order\_price: Double,

sku\_num: Long,

sku\_name: String,

create\_time: String,

**var** spu\_id: Long = 0L, *//作为维度数据 要关联进来*

**var** tm\_id: Long = 0L,

**var** category3\_id: Long = 0L,

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

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

**var** category3\_name: String = **null*) {***

**def** mergeSkuInfo***(***skuInfo: SkuInfo***)*** = ***{***

**this**.spu\_id = skuInfo.spu\_id.toLong

**this**.tm\_id = skuInfo.tm\_id.toLong

**this**.category3\_id = skuInfo.category3\_id.toLong

**this**.spu\_name = skuInfo.spu\_name

**this**.tm\_name = skuInfo.tm\_name

**this**.category3\_name = skuInfo.category3\_name

**this**

***}***

***}***

## DwdOrderDetailApp 数据写入到DWD层

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

**import** com.atguigu.gmall.realtime.BaseApp

**import** com.atguigu.gmall.realtime.bean.***{***OrderDetail, SkuInfo***}***

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

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

**import** org.apache.kafka.clients.producer.***{***KafkaProducer, ProducerRecord***}***

**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.CustomSerializer

**import** org.json4s.JsonAST.***{***JInt, JLong, JString***}***

**import** org.json4s.jackson.***{***JsonMethods, Serialization***}***

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

*/\*\**

*\* Author lzc*

*\* Date 2020/8/29 7:23 上午*

*\*/*

**object** DwdOrderDetailApp **extends** BaseApp ***{***

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

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

**override var** *topic*: String = **"ods\_order\_detail"**

**object** StringToLong **extends** CustomSerializer***[***Long***](***formats => ***( {***

**case** JString***(***x***)*** => x.toLong

**case** JInt***(***x***)*** => x.toLong

***}***, ***{***

**case** x: Long => JLong***(***x***)***

**case** x: Int => JInt***(***x***)***

***}))***

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

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

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

*// 1. 数据封装*

**val** orderDetailStream = sourceStream.map***(***record => ***{***

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

System.*out*.println***(***record.value***())***;

JsonMethods.parse***(***record.value***())***.extract***[***OrderDetail***]***

***})***

*// 2. 关联 sku\_info 维度表*

**val** spark: SparkSession = SparkSession

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

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

.getOrCreate***()***

**import** spark.implicits.\_

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

rdd.cache***()***

*// 2.1 获取素有需要的 SkuId*

**val** skuIds = rdd.map***(***\_.sku\_id***)***.collect***()***.distinct

**if *(***skuIds.length > 0***) {***

*// 2.2 skuId-> orderDetail*

**val** skuIdToOrderDetail = rdd.map***(***orderDetail => ***(***orderDetail.sku\_id.toString, orderDetail***))***

*// 2.2 加载 SkuInfo*

**val** skuIdToSkuInfo = SparkSqlUtil

.*getRDD****[***SkuInfo***](***spark, **s"select \* from gmall\_sku\_info where id in('$*{***skuIds.mkString***(*"','"*)}*')"*)***

.map***(***skuInfo => ***(***skuInfo.id, skuInfo***))***

*// 2.3 skuIdToOrderDetail join skuIdToSkuInfo*

skuIdToOrderDetail.join***(***skuIdToSkuInfo***)***.map ***{***

**case *(***\_, ***(***orderDetail, skuInfo***))*** => orderDetail.mergeSkuInfo***(***skuInfo***)***

***}***.foreachPartition***(***orderDetailIt => ***{***

*// 2.4 把 orderDetail 信息写入到 kafka*

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

orderDetailIt.foreach***(***orderDetail => ***{***

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

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

***})***

producer.close***()***

***})***

***}***

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

***})***

***}***

***}***

# 订单表和订单详情表双流join

**把订单表和订单详情表join到一起做成更宽的宽表**

## 不同类型的表join的特点

**order\_info和oder\_detail都是事实表, 事实表与事实表的join与事实表与维度表的join是不一样的.**

* **事实表与维度表join的特点:**

维度表的数据一般都是早于事实表, 所以事实表能够获取到维度表的全部数据, join的时候相对比较方便

* **事实表与事实表join的特点:**

事实表的数据都是实时产生的, 都是流数据. 所以join的时候只能同批次进行join.

但是由于网络传输或者其他一些原因, 会有可能导致同时产生的数据, 并不能在同批次达到流中, 导致join的时候join不到对方的数据.

所以, 流与流的join比较麻烦有难度.

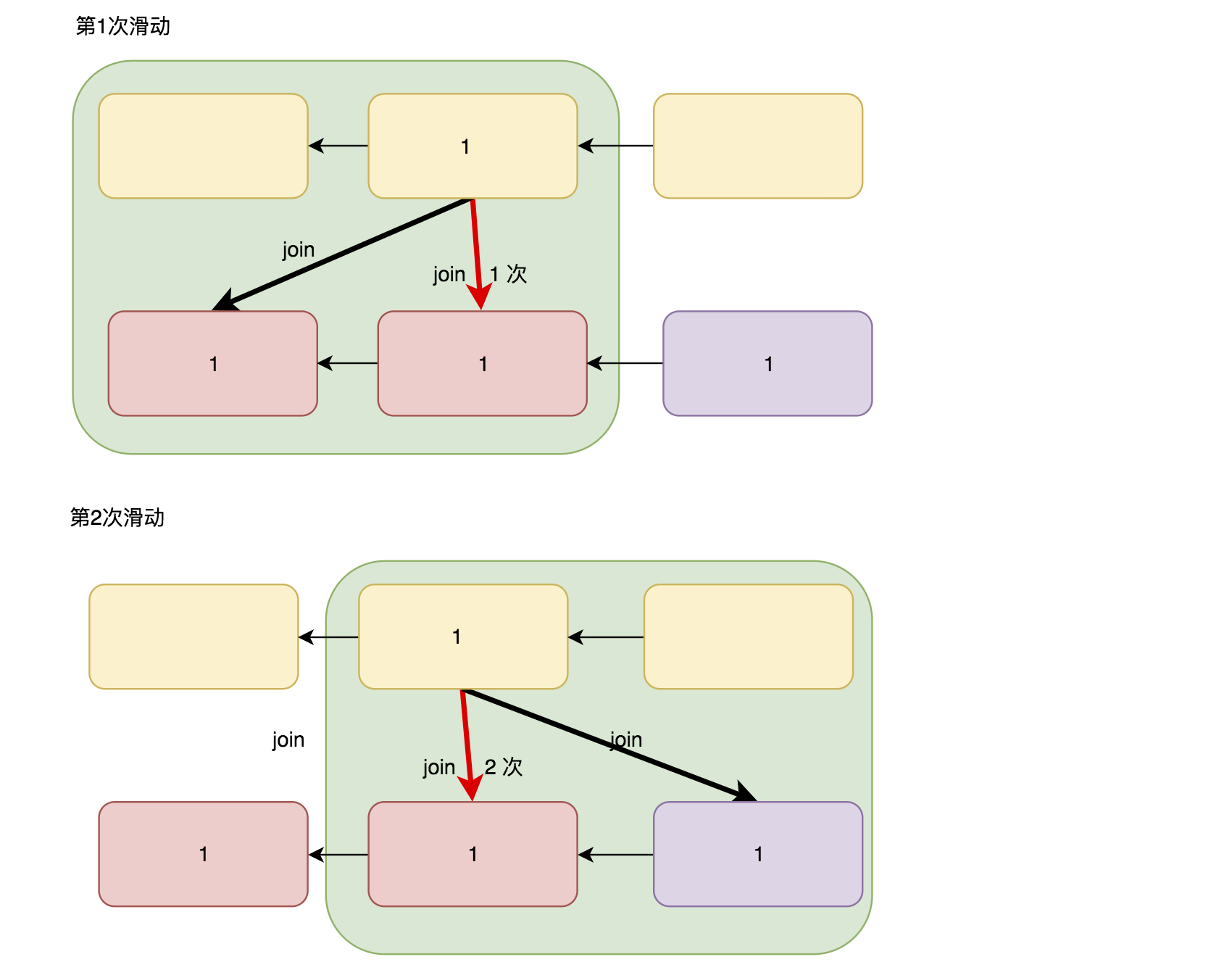
## 双流join的实现思路

实现双流join有2种思路:

1. **使用滑动window完成join**

由于各种原因, 两张表中同时产生的时候, 不能同批次的到的, 但是实际并不会差太多批次. 在join的时候, 可以使用滑动窗口来覆盖多个批次, 从而可以让同时产生的数据处于同一窗口中.

但是会出现一条数据, 出现在多个窗口中, 需要把聚合后的数据做去重处理.

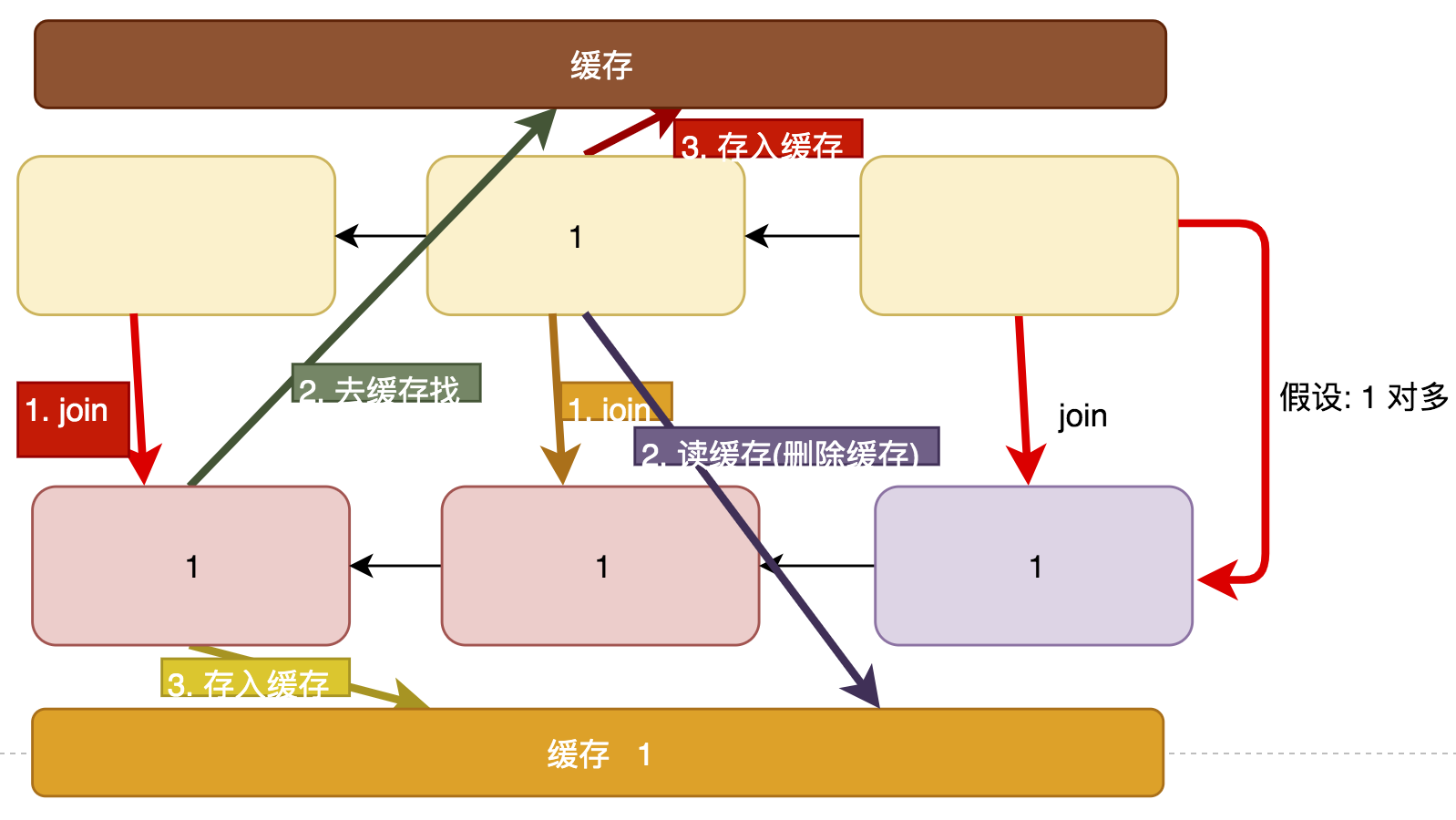


1. **使用缓存实现**

Join的时候, 如果在对方流的同批次中找不到数据, 则可以去对方的缓存中查找.

缓存必须是第3方的缓存, 不能是spark-Streaming中的自己的内存.

第3方缓存一般使用redis



## 创建join后存储数据的样例类

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

*/\*\**

*\* Author lzc*

*\* Date 2020/8/29 10:17 上午*

*\**

*\* 1. 粒度和 order\_detail 相同*

*\*/*

**case class** OrderWide***(*** *// 来源 OrderInfo*

**var** order\_id: Long = 0L,

**var** province\_id: Long = 0L,

**var** order\_status: String = **null**,

**var** user\_id: Long = 0L,

**var** final\_total\_amount: Double = 0D, *// 实际支付总金额 = 原始总金额-优惠+运费*

**var** benefit\_reduce\_amount: Double = 0D, *// 优惠金额*

**var** original\_total\_amount: Double = 0D, *// 原始总金额 = ∑sku\_price\*sku\_num*

**var** feight\_fee: Double = 0D, *// 运费*

**var** expire\_time: String = **null**,

**var** create\_time: String = **null**,

**var** operate\_time: String = **null**,

**var** create\_date: String = **null**,

**var** create\_hour: String = **null**,

**var** is\_first\_order: Boolean = **false**,

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

**var** province\_area\_code: String = **null**,

**var** province\_iso\_code: String = **null**,

**var** user\_age\_group: String = **null**,

**var** user\_gender: String = **null**,

*//开源: OderDetail*

**var** order\_detail\_id: Long = 0L,

**var** sku\_id: Long = 0L,

**var** sku\_price: Double = 0L, *// 在 OrderDetail 中叫 order\_price*

**var** sku\_num: Long = 0L,

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

**var** spu\_id: Long = 0L,

**var** tm\_id: Long = 0L,

**var** category3\_id: Long = 0L,

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

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

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

*// 需要计算的分摊金额*

**var** final\_detail\_amount: Double = 0D***) {***

**def this*(***orderInfo: OrderInfo, orderDetail: OrderDetail***) {***

**this**

mergeOrderInfo***(***orderInfo***)***

mergeOrderDetail***(***orderDetail***)***

***}***

**def** mergeOrderInfo***(***orderInfo: OrderInfo***)***: Unit = ***{***

**if *(***orderInfo != **null*) {***

**this**.order\_id = orderInfo.id

**this**.province\_id = orderInfo.province\_id

**this**.order\_status = orderInfo.order\_status

**this**.user\_id = orderInfo.user\_id

**this**.final\_total\_amount = orderInfo.final\_total\_amount

**this**.benefit\_reduce\_amount = orderInfo.benefit\_reduce\_amount

**this**.original\_total\_amount = orderInfo.original\_total\_amount

**this**.feight\_fee = orderInfo.feight\_fee

**this**.expire\_time = orderInfo.expire\_time

**this**.create\_time = orderInfo.create\_time

**this**.create\_date = orderInfo.create\_date

**this**.create\_hour = orderInfo.create\_hour

**this**.is\_first\_order = orderInfo.is\_first\_order

**this**.province\_name = orderInfo.province\_name

**this**.province\_area\_code = orderInfo.province\_area\_code

**this**.province\_iso\_code = orderInfo.province\_iso\_code

**this**.user\_age\_group = orderInfo.user\_age\_group

**this**.user\_gender = orderInfo.user\_gender

***}***

***}***

**def** mergeOrderDetail***(***orderDetail: OrderDetail***)***: Unit = ***{***

**if *(***orderDetail != **null*) {***

**this**.order\_detail\_id = orderDetail.id

**this**.sku\_id = orderDetail.sku\_id

**this**.sku\_num = orderDetail.sku\_num

**this**.sku\_name = orderDetail.sku\_name

**this**.sku\_price = orderDetail.order\_price

**this**.spu\_id = orderDetail.spu\_id

**this**.tm\_id = orderDetail.tm\_id

**this**.category3\_id = orderDetail.category3\_id

**this**.spu\_name = orderDetail.spu\_name

**this**.tm\_name = orderDetail.tm\_name

**this**.category3\_name = orderDetail.category3\_name

***}***

***}***

***}***

## 升级BaseApp

可以同时读多个topic, 得到多个流.(每个表得到一个流)

**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** BaseAppV3 ***{***

**var** appName: String

**var** groupId: String

**var** totalCores: Int

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

**def** run***(***ssc: StreamingContext, offsetRanges: ListBuffer***[***OffsetRange***]***, topicToStreamMap: Map***[***String, 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***)***

*println****(***fromOffsets***)***

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

**val** topicToStreamMap = topics

.map***(***topic => ***{***

**val** ownOffsets = fromOffsets.filter***(***\_.\_1.topic***()*** == topic***)***

**val** stream = MyKafkaUtil

.*getKafkaStream****(***ssc, groupId, topic, ownOffsets***)***

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

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

offsetRanges ++= newOffsetRanges

rdd

***})***

***(***topic, stream***)***

***})***

.toMap *// 转成 map 的目的是方便操作, 取出需要的流*

run***(***ssc, offsetRanges, topicToStreamMap***)***

ssc.start***()***

ssc.awaitTermination***()***

***}***

***}***

*/\**

*kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list hadoop102:9092 --topic dwd\_order\_info --time -1*

*\*/*

## 双流join思路1实现(自学)

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

**import** com.atguigu.gmall.realtime.BaseAppV3

**import** com.atguigu.gmall.realtime.bean.***{***OrderDetail, OrderInfo, OrderWide***}***

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

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

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

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

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

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

**import** redis.clients.jedis.Jedis

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

*/\*\**

*\* Author lzc*

*\* Date 2020/8/29 10:52 上午*

*\*/*

**object** DwsOrderWideApp **extends** BaseAppV3 ***{***

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

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

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

**override var** *topics*: Set***[***String***]*** = *Set****(*"dwd\_order\_info"**, **"dwd\_order\_detail"*)***

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

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

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

*// 1. 得到两个事实表的流, 为了 join 不丢数据, 并添加窗口*

**val** orderInfoStream = topicToStreamMap***(*"dwd\_order\_info"*)***

.map***(***record => ***{***

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

**val** orderInfo = JsonMethods.parse***(***record.value***())***.extract***[***OrderInfo***]***

***(***orderInfo.id, orderInfo***)***

***})***

.window***(****Seconds****(***24***)***, *Seconds****(***3***))***

**val** orderDetailStream = topicToStreamMap***(*"dwd\_order\_detail"*)***

.map***(***record => ***{***

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

**val** orderDetail = JsonMethods.parse***(***record.value***())***.extract***[***OrderDetail***]***

***(***orderDetail.order\_id, orderDetail***)***

***})***

.window***(****Seconds****(***24***)***, *Seconds****(***3***))***

*// 2. 对2 个流进行 join 此处使用内连接*

**val** orderWideStream: DStream***[***OrderWide***]*** = orderInfoStream.join***(***orderDetailStream***)***.map ***{***

**case *(***\_, ***(***orderInfo, orderDetail***))*** => **new** OrderWide***(***orderInfo, orderDetail***)***

***}***

*// 3. 对重复 join 的数据进行去重*

**val** orderWideDistinctStream = orderWideStream.mapPartitions***(***orderWideIt => ***{***

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

**val** result = orderWideIt.filter***(***orderWide => ***{***

**val** oneOrZero = client.sadd***(*s"order\_join:$*{***orderWide.order\_id***}*"**, orderWide.order\_detail\_id.toString***)***

client.expire***(*s"order\_join:$*{***orderWide.order\_id***}*"**, 60***)*** *// 给每个 key 单独设置过期时间*

oneOrZero == 1

***})***

client.close***()***

result

***})***

*// 5. 写入到 ClickHouse*

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

rdd.cache***()***

*println****(*"时间戳.....开始"*)***

rdd.collect***()***.foreach***(****println****)***

*println****(*"时间戳.....结束"*)***

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

***})***

***}***

***}***

*/\**

*1. 如何判断是否重复 join*

*类似于前面的判断日活*

*重复 join 的时候"order\_detail\_id" 肯定是相同的*

*所以, 把 "order\_detail\_id" 存入 set 中*

*1. 第一次存入, 返回 1 (不重复)*

*2. 第二次存入, 返回 0 (重复)*

*3. 把重复的过滤掉(返回值为0 的去掉)*

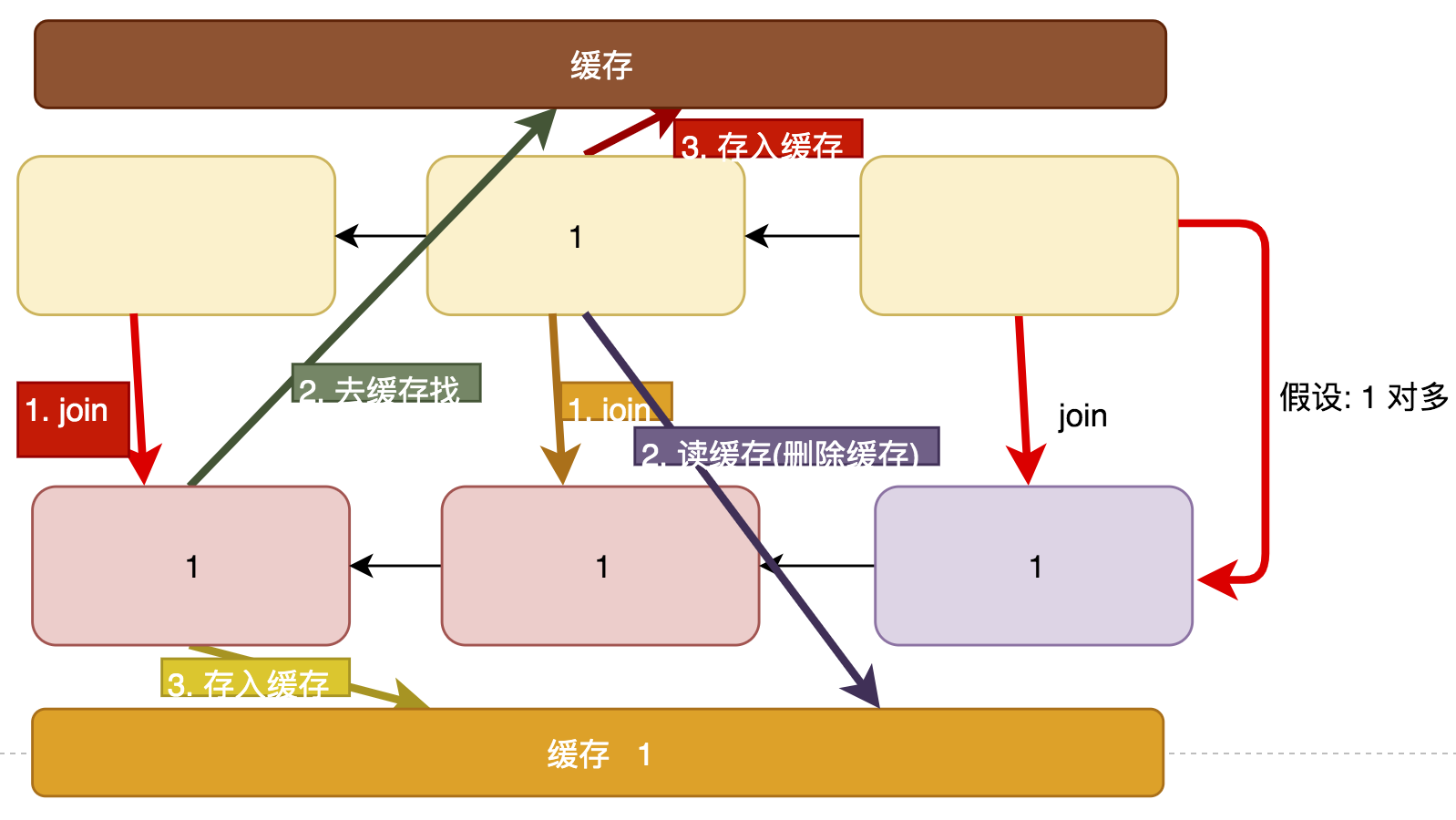
*key value(set)*

*"order\_join:${order\_id}" "1\_2", "1\_3",... order\_detail\_id, ...*

*说明: 每个order 一个key, 方便对 key 进行过期自动删除. 否则会长期占用内存*

*\*/*

## 双流join思路2实现(讲解)



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

**import** java.lang

**import** com.atguigu.gmall.realtime.BaseAppV3

**import** com.atguigu.gmall.realtime.bean.***{***OrderDetail, OrderInfo, OrderWide***}***

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

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

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

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

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

**import** org.json4s.jackson.***{***JsonMethods, Serialization***}***

**import** redis.clients.jedis.Jedis

**import** scala.collection.mutable

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

*/\*\**

*\* Author lzc*

*\* Date 2020/8/29 10:52 上午*

*\*/*

**object** DwsOrderWideAppV2 **extends** BaseAppV3 ***{***

**override var** *appName*: String = **"DwsOrderWideAppV2"**

**override var** *groupId*: String = **"DwsOrderWideAppV2"**

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

**override var** *topics*: Set***[***String***]*** = *Set****(*"dwd\_order\_info"**, **"dwd\_order\_detail"*)***

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

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

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

*// 1. 得到两个事实表的流, 为了 join 不丢数据, 并添加窗口*

**val** orderInfoStream = topicToStreamMap***(*"dwd\_order\_info"*)***

.map***(***record => ***{***

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

**val** orderInfo = JsonMethods.parse***(***record.value***())***.extract***[***OrderInfo***]***

***(***orderInfo.id, orderInfo***)***

***})***

**val** orderDetailStream = topicToStreamMap***(*"dwd\_order\_detail"*)***

.map***(***record => ***{***

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

**val** orderDetail = JsonMethods.parse***(***record.value***())***.extract***[***OrderDetail***]***

***(***orderDetail.order\_id, orderDetail***)***

***})***

*// 2. 对2 个流进行 join 必须使用全连接*

**val** orderWideStream: DStream***[***OrderWide***]*** = orderInfoStream

.fullOuterJoin***(***orderDetailStream***)***

.mapPartitions***(***it => ***{***

*// 1. 建立到redis的连接*

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

*// 2. 涉及到redis的操作*

**val** result = it.flatMap ***{***

*// order\_info 和order\_detail的数据同时到达*

**case *(***orderId, ***(***Some***(***orderInfo***)***, Some***(***orderDetail***)))*** =>

*println****(*s"$*{***orderId***}* some some"*)***

*// 1. 把order\_info信息写入到缓存*

*cacheOrderInfo****(***client, orderInfo***)***

*// 2. 把order\_info的信息和oder\_detail的信息封装到一起*

**val** orderWide: OrderWide = OrderWide***()***.mergeOrderInfo***(***orderInfo***)***.mergeOrderDetail***(***orderDetail***)***

*// 3. 去order\_detail的缓存中查找对应的信息. 注意: 需要删除order\_detail中的信息*

*// 3.1 先获取根order\_id相关的所有的key 3.2. 根据key回去对应的value(order\_detail)*

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

**val** orderWides: mutable.Set***[***OrderWide***]*** = client.keys***(*s"order\_detail:$*{***orderInfo.id***}*:\*"*)***.asScala.map***(***key => ***{***

**val** orderDetailJson: String = client.get***(***key***)***

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

**val** orderDetail: OrderDetail = JsonMethods.parse***(***orderDetailJson***)***.extract***[***OrderDetail***]***

*// order\_detail的缓存中的数据join成功之后,需要删除, 否则会出现重复数据*

client.del***(***key***)***

OrderWide***()***.mergeOrderInfo***(***orderInfo***)***.mergeOrderDetail***(***orderDetail***)***

***})***

orderWides += orderWide

orderWides

*// order\_info 和order\_detail的数据没有同时到达*

**case *(***orderId, ***(***Some***(***orderInfo***)***, None***))*** =>

*println****(*s"$*{***orderId***}* some none"*)***

*// 1. 把order\_info信息写入到缓存*

*cacheOrderInfo****(***client, orderInfo***)***

*// 2. 去order\_detail的缓存中查找对应的信息. 注意: 需要删除order\_detail中的信息*

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

**val** orderWides: mutable.Set***[***OrderWide***]*** = client.keys***(*s"order\_detail:$*{***orderInfo.id***}*:\*"*)***.asScala.map***(***key => ***{***

**val** orderDetailJson: String = client.get***(***key***)***

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

**val** orderDetail: OrderDetail = JsonMethods.parse***(***orderDetailJson***)***.extract***[***OrderDetail***]***

*// order\_detail的缓存中的数据join成功之后,需要删除, 否则会出现重复数据*

client.del***(***key***)***

OrderWide***()***.mergeOrderInfo***(***orderInfo***)***.mergeOrderDetail***(***orderDetail***)***

***})***

orderWides

**case *(***orderId, ***(***None, Some***(***orderDetail***)))*** =>

*println****(*s"$*{***orderId***}* none some"*)***

*// 1. 先去缓存查找对应的orderInfo*

**val** orderInfoString: String = client.get***(*s"order\_info:$*{***orderDetail.order\_id***}*"*)***

*// 2. 如果找到, 则组合成saleDetail. 如果没有找到, 应该把order\_detail缓存*

**if *(***orderInfoString != **null*) {***

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

**val** orderInfo = JsonMethods.parse***(***orderInfoString***)***.extract***[***OrderInfo***]***

OrderWide***()***.mergeOrderInfo***(***orderInfo***)***.mergeOrderDetail***(***orderDetail***)*** :: *Nil*

***}* else *{***

*// a: 先把order\_details缓存*

*cacheOrderDetail****(***client, orderDetail***)***

*// b: 返回空集合*

*Nil*

***}***

***}***

client.close***()***

result

***})***

*// 4. 写入到 ClickHouse*

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

rdd.cache***()***

*println****(*"时间戳.....开始"*)***

rdd.collect***()***.foreach***(****println****)***

*println****(*"时间戳.....结束"*)***

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

***})***

***}***

**def** cacheOrderInfo***(***client: Jedis, orderInfo: OrderInfo***)***: Unit = ***{***

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

client.setex***(*s"order\_info:$*{***orderInfo.id***}*"**, 60 \* 10, Serialization.*write****(***orderInfo***))***

***}***

**def** cacheOrderDetail***(***client: Jedis, orderDetail: OrderDetail***)***: Unit = ***{***

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

client.setex***(*s"order\_detail:$*{***orderDetail.order\_id***}*:$*{***orderDetail.id***}*"**, 60 \* 10, Serialization.*write****(***orderDetail***))***

***}***

***}***

# 计算订单明细实付金额分摊

## 计算分摊的思路

1. 按比例求分摊: 分摊金额/实际付款金额 = 个数\*单价/该单原始总金额
2. 所以: 分摊金额 = 个数\*单价\*实际付款金额/该单原始总金额
3. 由于有除法的存在, 结果我们需要做四舍五入, 会导致精度的丢失
4. 一个订单对应多个详情, 每个详情均做四舍五入, 他们的和可能与该订单的实际支付总金额不等

订单 详情 1 详情 2 详情 3

原始: 120 40 40 40

分摊: 100 33.33 33.33 33.33

1. 需要消除这个bug

如果有 3 个详情:

前面的详情使用乘除: 分摊金额 = 个数\*单价\*实际付款金额/该单原始总金额

最后一个详情使用减法: 分摊金额 = 实际付款总金额 - ∑前面分摊总金额

1. 难点: 如何知道最后一个详情

**当前详情**

个数\*成单价 == 该单原始总金额 - ∑前面详情的个数\*单价

1. 需要记录的状态:

∑前面分摊总金额

∑前面详情的个数\*单价

## 计算分摊的具体实现

// 4. 计算分摊金额

**val** result = orderWideDistinctStream.mapPartitions**(**orderWideIt => **{**

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

**val** r = orderWideIt.map**(**orderWide => **{**

**val** preTotalKey = **s"pre\_total:${**orderWide.order\_id**}"**

**val** preSharesKey = **s"pre\_shares:${**orderWide.order\_id**}"**

// 1. 获取前面详情的原始总金额(不包括当前详情的)

**val** preTotalTemp = client.get**(**preTotalKey**)**

**val** preTotal = **if (**preTotalTemp == **null)** 0D **else** preTotalTemp.toDouble

// 2. 获取前面详情的分摊总金额(不包括当前详情的)

**val** preSharesTemp = client.get**(**preSharesKey**)**

**val** preShares = **if (**preSharesTemp == **null)** 0D **else** preSharesTemp.toDouble

// 3. 判断是否最后一个详情

**val** current: Double = orderWide.sku\_price \* orderWide.sku\_num

**if (**current == orderWide.original\_total\_amount - preTotal**) {** // 个数\*成单价 == 该单原始总金额 - ∑前面详情的个数\*单价

// 3.1 是最后一详情 分摊金额 = 实际付款总金额 - ∑前面分摊总金额

orderWide.final\_detail\_amount = orderWide.final\_total\_amount - preShares

// 删除 redis 的对应的 key

client.del**(**preTotalKey**)**

client.del**(**preSharesKey**)**

println**("最后一详情")**

**} else {**

// 3.2 不是最后一详情 分摊金额 = 个数\*单价\*实际付款金额/该单原始总金额

**val** share: Double = orderWide.sku\_num \* orderWide.sku\_price \* orderWide.final\_total\_amount / orderWide.original\_total\_amount

// 3.33333333 \* 100 => 333.33333 => 333 => 3.33

orderWide.final\_detail\_amount = Math.round**(**share \* 100**)** / 100

// 保存 ∑前面分摊总金额 ∑前面详情的个数\*单价

**val** d1: lang.Double = client.incrByFloat**(**preTotalKey, current**)**

**val** d2: lang.Double = client.incrByFloat**(**preSharesKey, Math.round**(**share \* 100**)** / 100**)**

println**(s"不是最后一详情: ${**d1**}, ${**d2**}")**

**}**

orderWide

**})**

client.close**()**

r

**})**

# 数据写入到clickhuse(dws)