尚硅谷电商数仓项目

--实时计算（dws层）

版本：V 1.2

张晨

# 双流合并

除了事实表与维表进行合并形成宽表，还需要事实表与事实表进行合并形成更大的宽表。

## 1 双流合并的问题

由于两个流的数据是独立保存，独立消费，很有可能同一业务的数据，分布在不同的批次。因为join算子只join同一批次的数据。如果只用简单的join流方式，会丢失掉不同批次的数据。

## 2 解决策略

**一种 利用滑动窗口进行join 然后再进行去重**

**第二种 把数据存入缓存 ，关联时进行join后 再去查询缓存中的数据，来弥补不同批次的问题。**

## 3 代码实现

**case class**

|  |
| --- |
| **case class** OrderDetailWide(  **var** order\_detail\_id:Long =0L,  **var** order\_id: Long=0L,  **var** order\_status:String=**null**,  **var** create\_time:String=**null**,  **var** user\_id: Long=0L,  **var** sku\_id: Long=0L,  **var** sku\_price: Double=0D,  **var** sku\_num: Long=0L,  **var** sku\_name: String=**null**,  **var** benefit\_reduce\_amount:Double =0D ,  **var** original\_total\_amount:Double =0D ,  **var** feight\_fee:Double=0D,  **var** final\_total\_amount: Double =0D ,  **var** final\_detail\_amount:Double=0D,   **var** if\_first\_order:String=**null**,   **var** province\_name:String=**null**,  **var** province\_area\_code:String=**null**,   **var** user\_age\_group:String=**null**,  **var** user\_gender:String=**null**,   **var** dt: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** ) {  **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**.order\_status=orderInfo.order\_status  **this**.create\_time=orderInfo.create\_time  **this**.dt=orderInfo.create\_date   **this**.benefit\_reduce\_amount =orderInfo.benefit\_reduce\_amount  **this**.original\_total\_amount =orderInfo.original\_total\_amount  **this**.feight\_fee =orderInfo.feight\_fee  **this**.final\_total\_amount =orderInfo.final\_total\_amount    **this**.province\_name=orderInfo.province\_name  **this**.province\_area\_code=orderInfo.province\_area\_code   **this**.user\_age\_group=orderInfo.user\_age\_group  **this**.user\_gender=orderInfo.user\_gender   **this**.if\_first\_order=orderInfo.if\_first\_order   **this**.user\_id=orderInfo.user\_id  }  }    **def** mergeOrderDetail(orderDetail: OrderDetail): Unit ={  **if**(orderDetail!=**null**){  **this**.order\_detail\_id=orderDetail.id  **this**.sku\_id=orderDetail.sku\_id  **this**.sku\_name=orderDetail.sku\_name  **this**.sku\_price=orderDetail.order\_price  **this**.sku\_num=orderDetail.sku\_num   **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   }  } |

**实时计算代码**

|  |
| --- |
| **def** main(args: Array[String]): Unit = {    **val** sparkConf: SparkConf = **new** SparkConf().setAppName(**"dws\_order\_detail\_wide\_app"**).setMaster(**"local[\*]"**)  **val** ssc = **new** StreamingContext(sparkConf, *Seconds*(5))  **val** orderInfoTopic = **"DW\_ORDER\_INFO"  val** orderDetailTopic = **"DW\_ORDER\_DETAIL"  val** groupId = **"dws\_order\_detail\_wide\_consumer"  val** orderInfoOffsets: Map[TopicPartition, Long] = OffsetManager.*getOffset*(groupId, orderInfoTopic)  **val** orderDetailOffsets: Map[TopicPartition, Long] = OffsetManager.*getOffset*(groupId, orderDetailTopic)    **val** orderInputDstream: InputDStream[ConsumerRecord[String, String]] = MyKafkaUtil.*getKafkaStream*(orderInfoTopic, ssc, orderInfoOffsets, groupId)   **val** orderDetailInputDstream: InputDStream[ConsumerRecord[String, String]] = MyKafkaUtil.*getKafkaStream*(orderDetailTopic, ssc, orderDetailOffsets, groupId)   **var** orderInfoOffsetRanges: Array[OffsetRange] = Array.*empty*[OffsetRange]  **val** orderInputNDstream: DStream[ConsumerRecord[String, String]] = orderInputDstream.transform { rdd =>  orderInfoOffsetRanges = rdd.asInstanceOf[HasOffsetRanges].offsetRanges  rdd  }  **var** orderDetailOffsetRanges: Array[OffsetRange] = Array.*empty*[OffsetRange]  **val** orderDetailInputNDstream: DStream[ConsumerRecord[String, String]] = orderDetailInputDstream.transform { rdd =>  orderDetailOffsetRanges = rdd.asInstanceOf[HasOffsetRanges].offsetRanges  rdd  }    *//把订单和订单明细 转换为 case class的流* **val** orderInfoDstream: DStream[OrderInfo] = orderInputNDstream.map { record =>  **val** jsonString: String = record.value()  **val** orderInfo: OrderInfo = JSON.*parseObject*(jsonString, *classOf*[OrderInfo])  orderInfo  }   **val** orderDetailDstream: DStream[OrderDetail] = orderDetailInputNDstream.map(record => JSON.*parseObject*(record.value, *classOf*[OrderDetail]))   **val** orderInfoWinDstream: DStream[OrderInfo] = orderInfoDstream.window(*Seconds*(15), *Seconds*(5))  **val** orderDetailWinDstream: DStream[OrderDetail] = orderDetailDstream.window(*Seconds*(15), *Seconds*(5))  orderInfoWinDstream.cache()  orderDetailWinDstream.cache()  *// orderInfoWinDstream.print(1000)  // orderDetailWinDstream.print(1000)    // orderinfo 和 orderDetail 的双流join* **val** orderInfoWithKeyDstream: DStream[(Long, OrderInfo)] = orderInfoWinDstream.map(orderInfo => (orderInfo.id, orderInfo))  **val** orderDetailWithKeyDstream: DStream[(Long, OrderDetail)] = orderDetailWinDstream.map(orderDetail => (orderDetail.order\_id, orderDetail))   **val** orderJoinDstream: DStream[(Long, (OrderInfo, OrderDetail))] = orderInfoWithKeyDstream.join(orderDetailWithKeyDstream)   **val** orderDetailWideDstream: DStream[OrderDetailWide] = orderJoinDstream.map { **case** (id, (orderInfo, orderDetail)) => **new** OrderDetailWide(orderInfo, orderDetail) }    *//去重* **val** orderDetailWideFilteredDstream: DStream[OrderDetailWide] = orderDetailWideDstream.transform { rdd =>  *println*(**"前："** + rdd.count())  **val** logInfoRdd: RDD[OrderDetailWide] = rdd.mapPartitions { orderDetailWideItr =>  **val** jedis: Jedis = RedisUtil.*getJedisClient* **val** orderDetailWideFilteredList = **new** ListBuffer[OrderDetailWide]  **val** orderDetailWideList: List[OrderDetailWide] = orderDetailWideItr.toList   *println*(orderDetailWideList.map(orderDetailWide => orderDetailWide.order\_id).mkString(**","**))  **for** (orderDetailWide <- orderDetailWideList) {   **val** orderDetailWideKey = **"order\_detail\_wide:"** + orderDetailWide.dt  **val** ifFirst: lang.Long = jedis.sadd(orderDetailWideKey, orderDetailWide.order\_detail\_id.toString)  **if** (ifFirst == 1L) {  orderDetailWideFilteredList += orderDetailWide  }  }  jedis.close()  orderDetailWideFilteredList.toIterator  }  logInfoRdd.cache()  *println*(**"后："** + logInfoRdd.count())  logInfoRdd  }  orderDetailWideFilteredDstream.map(orderwide=>(orderwide.order\_id,orderwide.final\_total\_amount,orderwide.original\_total\_amount, orderwide.sku\_price,orderwide.sku\_num,orderwide.final\_detail\_amount)).print(1000) ssc.start() ssc.awaitTermination()  **}** |

**注意点：join时尽量不要出现shuffle。**

**如何解决：**

**在join前的数据保证分区是一对一的关系，利用kafka发送时的分区键，两张表的分区键和分区数保持一致。**

# 订单明细实付金额分摊

## 1 需求

主订单的应付金额【origin\_total\_amount】一般是由所有订单明细的商品单价\*数量汇总【sku\_price\*sku\_num】组成。

但是由于优惠、运费等都是以订单为单位进行计算的，所以减掉优惠、加上运费会得到一个最终实付金额【final\_total\_amount】。

但问题在于如果是以商品进行交易额分析，也要把优惠、运费的效果分摊到购买的每个商品中。

## 2 如何分摊呢？

一般是由订单明细每种商品的消费占总订单的比重进行分摊，比如总价1000元的商品，

由分别由600元和400元的A、B两种商品组成， 但是经过打折和加运费后，实际付款金额变为810，那么A的分摊实付金额为486元和B的分摊实付金额为324元。

## 3 麻烦的情况：

由于明细的分摊是由占比而得，那就会进行除法，除法就有可能出现除不尽的情况。

比如：原价90元 ，三种商品每件30元。没有优惠但有10元运费，总实付金额为100元。按占比分摊各三分之一，就会出现三个33.33元。加起来就会出现99.99元。就会出现差一分钱的情况。

而我们要求所有订单明细的实付分摊加总必须和订单的总实付相等。

所以我们要的是100=33.33+33.33+33.34

## 4 解决思路：

核心思路：就是需要用两种算法来计算金额

1) 算法一：

如果 **计算时该明细不是最后一笔**

使用乘除法公式：: 实付分摊金额/实付总金额= （数量\*单价）/原始总金额

调整移项可得 实付分摊金额=（数量\*单价）\*实付总金额 / 原始总金额

2) 算法二：

如果 **计算时该明细是最后一笔**

使用减法公式：

实付分摊金额= 实付总金额 - （其他明细已经计算好的【实付分摊金额】的合计）

3) 判断是否是最后一笔

判断公式： 如果 该条明细 （数量\*单价）== 原始总金额 -（其他明细 【数量\*单价】的合计）

4) 整个计算中需要的两个合计值：

* 其他明细已经计算好的【实付分摊金额】的合计
* 订单的已经计算完的明细的【数量\*单价】的合计

如何保存这两个合计？保存在redis中。

|  |  |
| --- | --- |
| type | hash |
| key | order\_split\_amount:[order\_id] |
| field | split\_amount\_sum , origin\_amount\_sum |
| value | 合计值 |

5、实现代码

|  |
| --- |
| **val** orderWideWithSplitDstream: DStream[OrderDetailWide] = orderDetailWideDStream.mapPartitions { orderWideItr =>  **val** jedis: Jedis = RedisUtil.*getJedisClient  // 1 先从redis取 两个合计 【实付分摊金额】的合计，【数量\*单价】的合计* **val** orderWideList: List[OrderDetailWide] = orderWideItr.toList  **for** (orderWide <- orderWideList) {  *// type ? hash key? order\_split\_amount:[order\_id] field split\_amount\_sum ,origin\_amount\_sum value ? 累积金额* **val** key = **"order\_split\_amount:"** + orderWide.order\_id  **val** orderSumMap: util.Map[String, String] = jedis.hgetAll(key)  **var** splitAmountSum = 0D  **var** originAmountSum = 0D  **if** (orderSumMap != **null** && orderSumMap.size() > 0) {  **val** splitAmountSumString: String = orderSumMap.get(**"split\_amount\_sum"**)  splitAmountSum = splitAmountSumString.toDouble   **val** originAmountSumString: String = orderSumMap.get(**"origin\_amount\_sum"**)  originAmountSum = originAmountSumString.toDouble  }  *// 2 先判断是否是最后一笔 ： （数量\*单价）== 原始总金额 -（其他明细 【数量\*单价】的合计）* **val** detailOrginAmount: Double = orderWide.sku\_num \* orderWide.sku\_price *//单条明细的原始金额 数量\*单价* **val** restOriginAmount: Double = orderWide.final\_total\_amount - originAmountSum  **if** (detailOrginAmount == restOriginAmount) {  *//3.1 最后一笔 用减法 ：实付分摊金额= 实付总金额 - （其他明细已经计算好的【实付分摊金额】的合计）* orderWide.final\_detail\_amount = orderWide.final\_total\_amount - splitAmountSum  } **else** {  *//3.2 不是最后一笔 用乘除 实付分摊金额=（数量\*单价）\*实付总金额 / 原始总金额* orderWide.final\_detail\_amount = detailOrginAmount \* orderWide.final\_total\_amount / orderWide.original\_total\_amount  orderWide.final\_detail\_amount= Math.*round*(orderWide.final\_detail\_amount\*100D)/100D  }  *// 4 进行合计保存* splitAmountSum += orderWide.final\_detail\_amount  originAmountSum += detailOrginAmount  orderSumMap.put(**"split\_amount\_sum"**, splitAmountSum.toString)  orderSumMap.put(**"origin\_amount\_sum"**, originAmountSum.toString)  jedis.hmset(key, orderSumMap)  }  jedis.close()  orderWideList.toIterator } |

# 保存到clickhouse

## 1 clickhouse 安装及入门，参见《尚硅谷clickhouse》课件

## 2 在clickhouse中建立表

|  |
| --- |
| create table order\_wide (  order\_detail\_id UInt64,  order\_id UInt64,  order\_status String,  create\_time DateTime,  user\_id UInt64,  sku\_id UInt64,  sku\_price Decimal64(2),  sku\_num UInt64,  sku\_name String,  benefit\_reduce\_amount Decimal64(2),  original\_total\_amount Decimal64(2),  feight\_fee Decimal64(2),  final\_total\_amount Decimal64(2),  final\_detail\_amount Decimal64(2),  if\_first\_order String,  province\_name String,  province\_area\_code String,  user\_age\_group String,  user\_gender String,  dt Date,  spu\_id UInt64,  tm\_id UInt64,  category3\_id UInt64,  spu\_name String,  tm\_name String,  category3\_name String  )engine =ReplacingMergeTree(create\_time)  partition by dt  primary key (order\_detail\_id)  order by (order\_detail\_id ) |

## 3 在sparkstreaming中增加写入clickhouse部分

### 3.1 pom.xml

添加

|  |
| --- |
| <**dependency**>  <**groupId**>ru.yandex.clickhouse</**groupId**>  <**artifactId**>clickhouse-jdbc</**artifactId**>  <**version**>0.1.55</**version**> </**dependency**> |

### 3.2 sparkstreaming 写入clickhouse

|  |
| --- |
| **val** sparkSession = SparkSession.*builder*()  .appName(**"order\_detail\_wide\_spark\_app"**)  .getOrCreate()  **import** sparkSession.implicits.\_ orderDetailWideDStream.foreachRDD{rdd=>   **val** df: DataFrame = rdd.toDF()  df.write.mode(SaveMode.*Append*)  .option(**"batchsize"**, **"100"**)  .option(**"isolationLevel"**, **"NONE"**) *// 设置事务* .option(**"numPartitions"**, **"4"**) *// 设置并发* .option(**"driver"**,**"ru.yandex.clickhouse.ClickHouseDriver"**)  .jdbc(**"jdbc:clickhouse://hdp1:8123/test1"**,**"order\_wide"**,**new** Properties())  } |

# 发布数据接口

## 1 代码清单

|  |  |  |
| --- | --- | --- |
| 控制层 | PublisherController | 实现接口的web发布 |
| 服务层 | ClickhouseService | 数据业务查询interface |
| ClickhouseServiceImpl | 业务查询的实现类 |
| 数据层 | OrderMapper | 数据层查询的interface |
| OrderMapper.xml | 数据层查询的实现配置 |

## 2 接口

### 2.1 访问路径

|  |  |
| --- | --- |
| 总数 | http://publisher:8070/realtime-total?date=2019-02-01 |
| 分时统计 | http://publisher:8070/realtime-hour?id=order\_amount&date=2019-02-01 |

### 2.2 要求数据格式

|  |  |
| --- | --- |
| 总数 | [{"id":"dau","name":"新增日活","value":1200},  {"id":"new\_mid","name":"新增设备","value":233 },  {"id":"order\_amount","name":"新增交易额","value":1000.2 }] |
| 分时统计 | {"yesterday":{"11":383,"12":123,"17":88,"19":200 },  "today":{"12":38,"13":1233,"17":123,"19":688 }} |

## 3 代码开发

### 3.1 pom.xml

|  |
| --- |
| <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-jdbc</**artifactId**> </**dependency**>   <**dependency**>  <**groupId**>org.mybatis.spring.boot</**groupId**>  <**artifactId**>mybatis-spring-boot-starter</**artifactId**>  <**version**>1.3.4</**version**> </**dependency**>   <**dependency**>  <**groupId**>ru.yandex.clickhouse</**groupId**>  <**artifactId**>clickhouse-jdbc</**artifactId**>  <**version**>0.1.55</**version**> </**dependency**> |

### 3.2 OrderMapper

|  |
| --- |
| **import** java.util.List; **import** java.util.Map;  **public interface** OrderMapper {   *//1 查询当日交易额总数* **public** BigDecimal selectOrderAmountTotal(String date);   *//2 查询当日交易额分时明细* **public** List<Map> selectOrderAmountHourMap(String date);  } |

### 3.3 OrderMapper.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <!DOCTYPE mapper SYSTEM "http://mybatis.org/dtd/mybatis-3-mapper.dtd" *>* <mapper namespace="com.atguigu.gmall0105.publisher.mapper.OrderMapper">  <select id="selectOrderAmountTotal" resultType="java.math.BigDecimal">  select *sum*(final\_total\_amount) sum\_amount from order\_wide where dt=#{date}  </select>   <select id="selectOrderAmountHourMap" resultMap="orderAmountHour" >  select toHour(create\_time) hr ,*sum*(final\_total\_amount) am from order\_wide where dt=#{date} group by toHour(create\_time)   </select>  <resultMap id="orderAmountHour" type="java.util.Map" autoMapping="true">  </resultMap>  </mapper> |

### 3.4 application.properties

添加：

|  |
| --- |
| spring.datasource.driver-class-name=ru.yandex.clickhouse.ClickHouseDriver spring.datasource.url=jdbc:clickhouse://hdp1:8123/test1   mybatis.mapperLocations=classpath:mapper/\*.xml mybatis.configuration.map-underscore-to-camel-case=true |

### 3.5 增加扫描包路径

|  |
| --- |
| @SpringBootApplication @MapperScan(basePackages = **"com.atguigu.gmallXXXXXXX.publisher.mapper"**) **public class** Gmall2019PublisherApplication{   **public static void** main(String[] args) {  SpringApplication.*run*(Gmall2019PublisherApplication.**class**, args);  }  } |

### 3.6 ClickHouseService

|  |
| --- |
| **public** BigDecimal getOrderAmount(String date);  **public** Map getOrderAmountHour(String date); |

### 3.7 ClickHouseServiceImpl

|  |
| --- |
| @Service **public class** ClickHouseServiceImpl **implements** ClickHouseService {   @Autowired  OrderMapper **orderMapper**;   @Override  **public** BigDecimal getOrderAmount(String date) {  **return orderMapper**.selectOrderAmountTotal(date);  }   @Override  **public** Map getOrderAmountHour(String date) {  List<Map> mapList = **orderMapper**.selectOrderAmountHourMap(date);  Map orderAmountHourMap=**new** HashMap();  **for** (Map map : mapList) {  orderAmountHourMap.put(map.get(**"hr"**), map.get(**"am"**));  }  **return** orderAmountHourMap;  }  } |

### 3.5 PublisherController

|  |
| --- |
| @RestController **public class** PublisherController {   @Autowired  EsService **esService**;   @Autowired  ClickHouseService **clickHouseService**;   @RequestMapping(value = **"realtime-total"**,method = RequestMethod.***GET***)  **public** String realtimeTotal(@RequestParam(**"date"**) String dt){  List<Map<String,Object>> rsList=**new** ArrayList<>();   Map<String,Object> dauMap = **new** HashMap();  dauMap.put(**"id"**,**"dau"**);  dauMap.put(**"name"**,**"新增日活"**);  Long dauTotal=0L;  **try** {  dauTotal = **esService**.getDauTotal(dt);  }**catch** ( Exception e){  e.printStackTrace();  }  **if**(dauTotal!=**null**){  dauMap.put(**"value"**,dauTotal);  }**else** {  dauMap.put(**"value"**,0L);  }   rsList.add(dauMap);   Map<String,Object> newMidMap = **new** HashMap();  newMidMap.put(**"id"**,**"new\_mid"**);  newMidMap.put(**"name"**,**"新增设备"**);  newMidMap.put(**"value"**,233);  rsList.add(newMidMap);    Map<String,Object> orderAmountMap = **new** HashMap();  orderAmountMap.put(**"id"**,**"order\_amount"**);  orderAmountMap.put(**"name"**,**"新增交易额"**);  BigDecimal orderAmount = **clickHouseService**.getOrderAmount(dt).setScale(2, RoundingMode.***HALF\_UP***);  orderAmountMap.put(**"value"**,orderAmount);   rsList.add(orderAmountMap);   **return** JSON.*toJSONString*(rsList);  }   @GetMapping(**"realtime-hour"**)  **public** String realtimeHour(@RequestParam(**"id"**) String id ,@RequestParam(**"date"**) String dt){  **if**(id.equals(**"dau"**)){  Map dauHourMapTD = **esService**.getDauHour(dt);  String yd = getYd(dt);  Map dauHourMapYD = **esService**.getDauHour(yd);   Map<String,Map<String,Long>> rsMap=**new** HashMap<>();  rsMap.put(**"yesterday"**,dauHourMapYD);  rsMap.put(**"today"**,dauHourMapTD);  **return** JSON.*toJSONString*(rsMap);  }**else if**(id.equals(**"order\_amount"**)){  Map orderAmountHourMapTD = **clickHouseService**.getOrderAmountHour(dt);  String yd = getYd(dt);  Map orderAmountHourMapYD = **clickHouseService**.getOrderAmountHour(yd);   Map<String,Map<String,BigDecimal>> rsMap=**new** HashMap<>();  rsMap.put(**"yesterday"**,orderAmountHourMapYD);  rsMap.put(**"today"**,orderAmountHourMapTD);  **return** JSON.*toJSONString*(rsMap);  }**else**{  **return null**;  }   }   **private** String getYd(String today){  SimpleDateFormat dateFormat = **new** SimpleDateFormat(**"yyyy-MM-dd"**);  **try** {  Date todayDate = dateFormat.parse(today);  Date ydDate = DateUtils.*addDays*(todayDate, -1);  **return** dateFormat.format(ydDate);   } **catch** (ParseException e) {  e.printStackTrace();  **throw new** RuntimeException(**"日期格式不正确"**);  }   }    } |