

尚硅谷大数据项目之 Spark 实时（分层处理）

（作者：尚硅谷研究院）

版本：V2.0

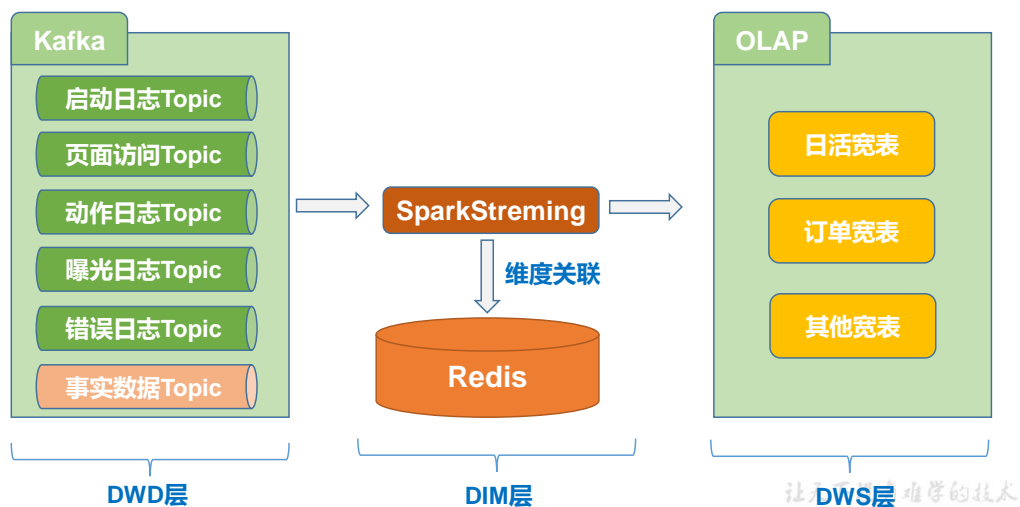
第 1 章 DWD 到 DWS 层数据处理概要

首先明确的是数据的聚合操作，一律交给 OLAP 来完成，因为 OLAP 数据库的特性都是非常利于数据聚合统计的，可以说这也是 OLAP 的本职工作。

那么在此之前实时计算就要完成一些 OLAP 不是特别方便或者性能并不好的操作，比如 JOIN 操作，比如分组去重(需要开窗)或者复杂的数据计算等等，实际情况还要看 OLAP 的选型。那这些工作可以交由实时计算完成。

从 ODS 到 DWD 层主要负责原始数据的整理拆分，形成一个一个的业务事实 topic。

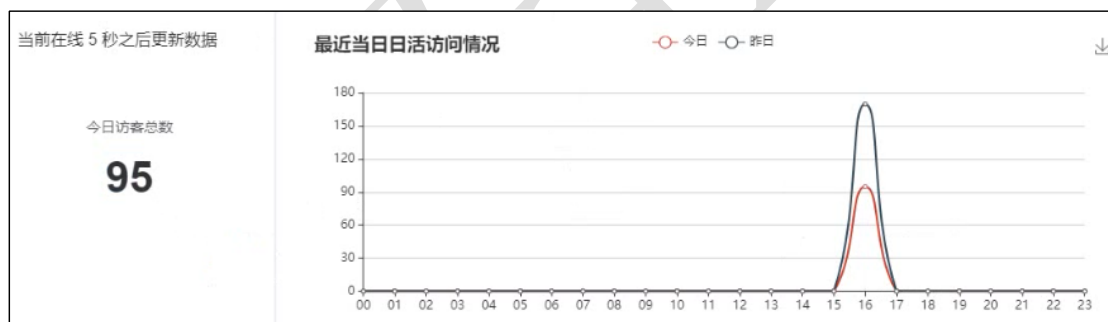
从 DWD 层到 DWS 层主要负责把单个的业务事实 topic 变为面向统计的事实明细宽表，然后保存到 OLAP 中。



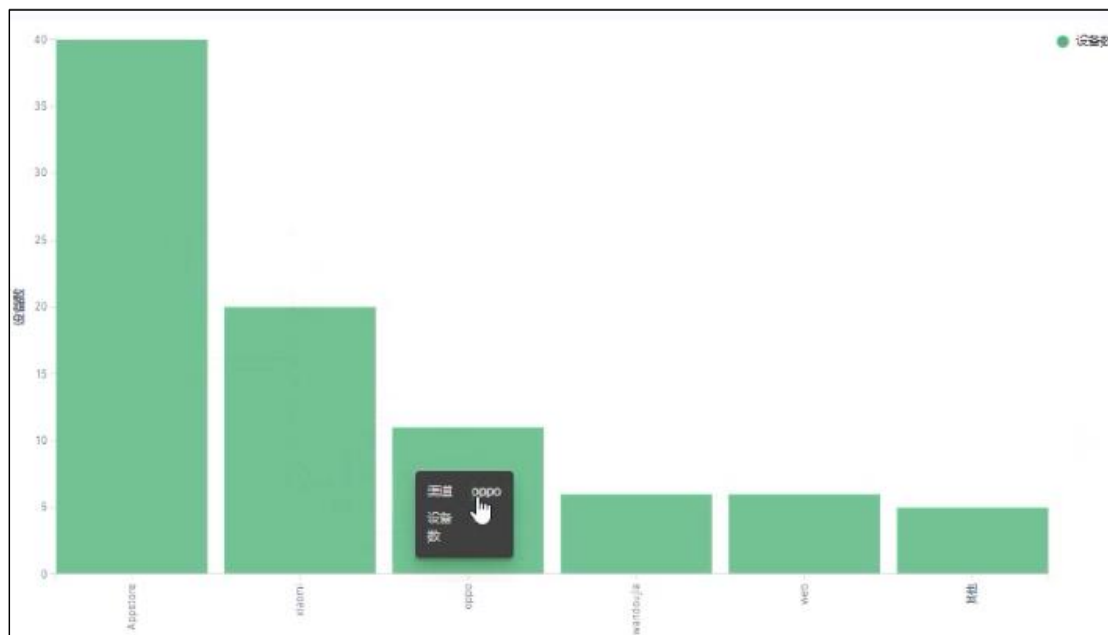
第 2 章 任务:日活宽表

2.1 需求举例

1) 当日用户首次登录（日活）分时趋势图



2) 不同渠道柱形对比图:



2.2 任务分析

2.2.1 去重

由于日活代表了用户当日的首次访问，因此除了当日的首次访问以外的其他访问，一律需要过滤掉。每个用户每天可能启动多次。要想计算日活，我们只需要把当前用户每天的第一次启动日志获取即可，所以要对启动日志进行去重，相当于做了一次清洗。

实时计算中的去重是一个比较常见的需求，可以有多种方式实现，比如：将状态存在 Redis 中；存在关系型数据库中；通过 Spark 自身的 `updateStateByKey`（checkPoint 小文件等问题比较麻烦、不方便管理、程序无法变更升级）等

我们这里结合 Redis 实现对当前用户启动日志去重操作

2.2.2 维度关联

由于要针对各种对于不同角度的“日活”分析，而 OLAP 数据库中尽量减少 join 操作，所以在实时计算中要考虑其会被分析的维度，补充响应相应的维度数据，形成宽表。

由于维度数据已经保存在固定容器中了(redis)，所以在实时计算中，维度与事实数据关联并不是通过 join 算子完成。而是在流中查询固定容器来实现维度补充。

2.2.3 输出到 OLAP 中

把清洗和组合好的宽表数据写入到 OLAP 中。这个部分要注意做到尽可能做到：

批量写入：主要是为了减少 IO 次数提高写入性能，同时也能尽可能减少 OLAP 数据内部的小文件的生成。

幂等性写入：为了保证数据的精确一次消费，通过在实时计算中通过控制提交偏移量的位置避免数据丢失，同时通过幂等性的写入避免重复数据。

2.3 功能实现

2.3.1 准备 Bean

```
package com.atguigu.gmall.realtime.bean

case class DauInfo(
    //基本的页面访问日志的数据
    var mid :String,
    var user_id:String,
    var province_id:String,
    var channel:String,
    var is_new:String,
    var model:String,
    var operate_system:String,
    var version_code:String,
    var page_id:String ,
    var page_item:String,
    var page_item_type:String,
    var during_time:Long,

    //用户性别 年龄
    var user_gender : String ,
    var user_age : String ,

    //地区信息

    var province_name : String ,
    var province_iso_code: String ,
    var province_3166_2 :String ,
    var province_area_code : String,

    //日期
    var dt : String ,
    var hr : String ,
    var ts : Long
){

    def this(){
        this(null,null,null,null,null,null,null,null,null,null,null,0L,null,null,null,null,null,null,null,null,0L)
    }
}
```

2.3.2 准备对象拷贝工具类

```
package com.atguigu.gmall.realtime.util

import java.lang.reflect.{Field, Method, Modifier}

import com.atguigu.gmall.realtime.bean.{ DauInfo, PageLog }

import scala.util.control.Breaks

/**
 * 实现对象属性拷贝.
 */
object MyBeanUtils {

  /**
   * 将 srcObj 中属性的值拷贝到 destObj 对应的属性上.
   */
  def copyProperties(srcObj : AnyRef , destObj: AnyRef): Unit ={
    if(srcObj == null || destObj == null ){
      return
    }
    //获取到 srcObj 中所有的属性
    val srcFields: Array[Field] =
srcObj.getClass.getDeclaredFields

    //处理每个属性的拷贝
    for (srcField <- srcFields) {
      Breaks.breakable{
        //get / set
        // Scala 会自动为类中的属性提供 get、 set 方法
        // get : fieldName()
        // set : fieldName_$eq(参数类型)

        //getMethodName
        var getMethodName : String = srcField.getName
        //setMethodName
        var setMethodName : String = srcField.getName+"_$eq"

        //从 srcObj 中获取 get 方法对象,
        val getMethod: Method =
srcObj.getClass.getDeclaredMethod(getMethodName)
        //从 destObj 中获取 set 方法对象
        // String name;
        // getName()
        // setName(String name ){ this.name = name }
        val setMethod: Method =
        try{
          destObj.getClass.getDeclaredMethod(setMethodName,
srcField.getType)
        }catch{
          // NoSuchMethodException
          case ex : Exception => Breaks.break()
        }
      }
    }
  }
}
```

```
//忽略 val 属性
val destField: Field =
destObj.getClass.getDeclaredField(srcField.getName)
if(destField.getModifiers.equals(Modifier.FINAL)){
    Breaks.break()
}
//调用 get 方法获取到 srcObj 属性的值，再调用 set 方法将获取到的属性
值赋值给 destObj 的属性
setMethod.invoke(destObj, getMethod.invoke(srcObj))
}

}

}
```

2.3.3 消费 Kafka 数据

```
object DwDDauApp {
    def main(args: Array[String]): Unit = {
        //1.环境
        val sparkconf: SparkConf = new
SparkConf().setAppName("dwd_dau_app").setMaster("local[4]")
        val ssc = new StreamingContext(sparkconf,Seconds(5))

        val topic = "DWD_PAGE_LOG"
        val groupId = "dwd_dau_group"

        //2.读取偏移量
        val offsets: Map[TopicPartition, Long] =
MyOffsetUtils.getOffset(topic,groupId)

        //3.接收 Kafka 数据
        var kafkaDStream: DStream[ConsumerRecord[String, String]] =
null
        if(offsets != null && offsets.nonEmpty ){
            kafkaDStream =
MyKafkaUtils.getKafkaDStream(topic,ssc,offsets,groupId)
        }else{
            kafkaDStream =
MyKafkaUtils.getKafkaDStream(topic,ssc,groupId)
        }

        //4. 提取偏移量结束点
        var offsetRanges: Array[OffsetRange] = null
        kafkaDStream = kafkaDStream.transform(
            rdd => {
                offsetRanges =
rdd.asInstanceOf[HasOffsetRanges].offsetRanges
            }
        )
    }
}
```

```
//5. 转化结构
val pageLogDStream: DStream[PageLog] = kafkaDStream.map(
  record => {
    val jsonStr: String = record.value()
    val pageLog: PageLog = JSON.parseObject(jsonStr,
      classOf[PageLog])
    pageLog
  }
)
pageLogDStream.print(100)
}
```

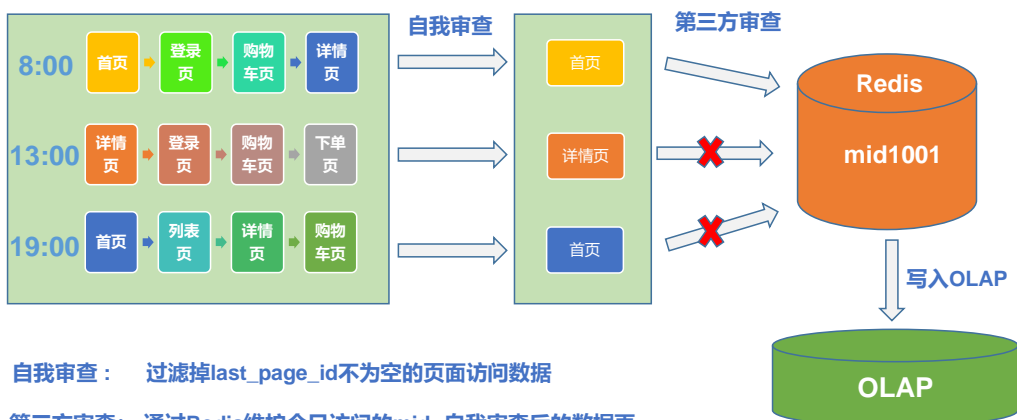
2.3.4 去重

1) 分析

日活的统计我们只需要考虑用户的首次访问行为，不同企业判断用户活跃的方式不同，可以通过启动数据或者页面数据来分析，我们采用页面数据来统计日活，页面数据中包含用户所有的访问行为，而我们只需要首次访问行为，所以可以在每批次的数据中先将包含有 last_page_id 的数据过滤掉，剩下的数据再与第三方（redis）中所记录的今日访问用户进行比对。



假设 mid1001 今日的访问记录如下



让天下没有难学的技术

2) 代码实现

```
//6. 筛选去重
// 6.1 自我审查 凡是有 last_page_id, 说明不是本次会话的第一个页面，直接过滤掉
val filterDStreamByLastPage: DStream[PageLog] =
pageLogDStream.filter(
  pageLog => {
    pageLog.last_page_id == null
  }
)
```

```
    }  
  )  
  // 6.2 第三方审查 所有会话的第一个页面， 去 redis 中检查是否是今天的第一  
次  
  val pageLogFilterDStream: DStream[PageLog] =  
filterDStreamByLastPage.mapPartitions(  
  pageLogIter => {  
    //获取 redis 连接  
    val jedis: Jedis = MyRedisUtils.getJedisClient  
    val filterList: ListBuffer[PageLog] =  
ListBuffer[PageLog]()  
    val pageLogList: List[PageLog] = pageLogIter.toList  
    println("过滤前 : " + pageLogList.size)  
    for (pageLog <- pageLogList) {  
      val sdf = new SimpleDateFormat("yyyy-MM-dd")  
      val dateStr: String = sdf.format(new Date(pageLog.ts))  
      val dauKey: String = s"DAU:$dateStr"  
      val ifNew: lang.Long = jedis.sadd(dauKey, pageLog.mid)  
      //设置过期时间  
      jedis.expire(dauKey, 3600 * 24)  
      if (ifNew == 1L) {  
        filterList.append(pageLog)  
      }  
    }  
    jedis.close()  
    println("过滤后: " + filterList.size)  
    filterList.toIterator  
  }  
)  
//pageLogFilterDStream.print(10)
```

2.3.5 维度合并

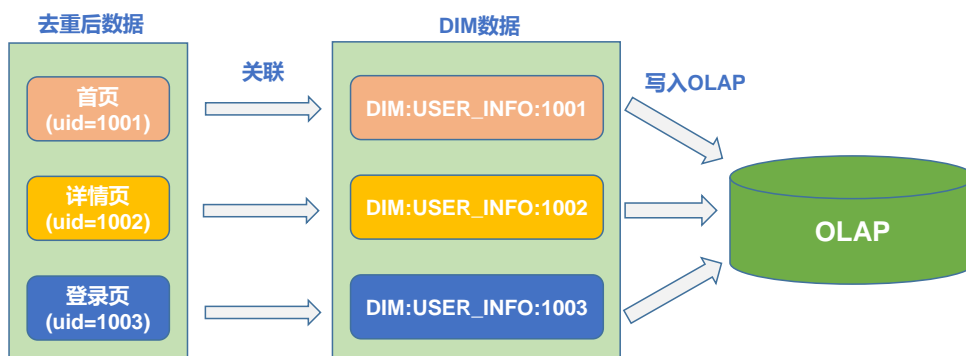
1) 分析

由于要针对各种对于不同角度的“日活”分析，而 OLAP 数据库中尽量减少 join 操作，所以在实时计算中要考虑其会被分析的维度，补充相应的维度数据，形成宽表。

由于维度数据已经保存在固定容器中了，所以在实时计算中，维度与事实数据关联并不是通过 join 算子完成。而是在流中查询固定容器（redis/mysql/hbase）来实现维度补充。



用户信息维度关联



让天下没有难学的技术

2) 代码实现

```
//7. 维度合并
val      dauInfoDStream:      DStream[DauInfo]      =
pageLogFilterDStream.mapPartitions(
  pageLogIter => {
    val jedis: Jedis = MyRedisUtils.getJedisClient
    val      dauInfoList:      ListBuffer[DauInfo]    =
ListBuffer[DauInfo]()

    for (pageLog <- pageLogIter) {
      //用户信息关联
      val dimUserKey = s"DIM:USER_INFO:${pageLog.user_id}"
      val userInfoJson: String = jedis.get(dimUserKey)
      val      userInfoJsonObj:      JSONObject      =
JSON.parseObject(userInfoJson)
      //提取生日
      val      birthday:      String      =
userInfoJsonObj.getString("birthday")
      //提取性别
      val gender: String = userInfoJsonObj.getString("gender")
      //生日处理为年龄
      var age: String = null
      if (birthday != null) {
        //闰年无误差
        val      birthdayDate:      LocalDate      =
LocalDate.parse(birthday)
        val nowDate: LocalDate = LocalDate.now()
        val period: Period = Period.between(birthdayDate,
nowDate)
        val years: Int = period.getYears
        age = years.toString
      }

      val dauInfo = new DauInfo()
```

```
//将 PageLog 的字段信息拷贝到 DauInfo 中
MyBeanUtils.copyProperties(pageLog, dauInfo)
dauInfo.user_gender = gender
dauInfo.user_age = age

//TODO 地区维度关联
val provinceKey: String =
s"DIM:BASE_PROVINCE:${pageLog.province_id}"
val provinceJson: String = jedis.get(provinceKey)
if(provinceJson!= null && provinceJson.nonEmpty ){
    val provinceJsonObj: JSONObject =
JSON.parseObject(provinceJson)
    dauInfo.province_name =
provinceJsonObj.getString("name")
    dauInfo.province_area_code =
provinceJsonObj.getString("area_code")
    dauInfo.province_3166_2 =
provinceJsonObj.getString("iso_3166_2")
    dauInfo.province_iso_code =
provinceJsonObj.getString("iso_code")
}

//日期补充
val dateFormat = new SimpleDateFormat("yyyy-MM-dd HH")
val dtDate = new Date(dauInfo.ts)
val dtHr: String = dateFormat.format(dtDate)
val dtHrArr: Array[String] = dtHr.split(" ")
dauInfo.dt = dtHrArr(0)
dauInfo.hr = dtHrArr(1)

dauInfoList.append(dauInfo)

}
jedis.close()
dauInfoList.toIterator
}
)
dauInfoDStream.print(100)
```

2.3.6 写入 ES 并提交偏移量

1) 建立索引模板

```
PUT _template/gmall_dau_info_template
{
  "index_patterns": ["gmall_dau_info*"],
  "settings": {
    "number_of_shards": 3
  },
  "aliases" : {
    "{index}-query": {},
    "gmall_dau_info_all":{}
  },
  "mappings": {
    "properties":{
      "mid":{
```

```
        "type": "keyword"
    },
    "user_id": {
        "type": "keyword"
    },
    "province_id": {
        "type": "keyword"
    },
    "channel": {
        "type": "keyword"
    },
    "is_new": {
        "type": "keyword"
    },
    "model": {
        "type": "keyword"
    },
    "operate_system": {
        "type": "keyword"
    },
    "version_code": {
        "type": "keyword"
    },
    "page_id": {
        "type": "keyword"
    },
    "page_item": {
        "type": "keyword"
    },
    "page_item_type": {
        "type": "keyword"
    },
    "during_time": {
        "type": "long"
    },
    "user_gender": {
        "type": "keyword"
    },
    "user_age": {
        "type": "integer"
    },
    "province_name": {
        "type": "keyword"
    },
    "province_iso_code": {
        "type": "keyword"
    },
    "province_3166_2": {
        "type": "keyword"
    },
    "province_area_code": {
        "type": "keyword"
    },
    "dt": {
        "type": "keyword"
    },
    },
```

```
        "hr":{
            "type":"keyword"
        },
        "ts":{
            "type":"date"
        }
    }
}
```

2) ES 工具类

```
package com.atguigu.gmall.realtime.util

import com.alibaba.fastjson.JSON
import com.alibaba.fastjson.serializer.SerializeConfig
import org.apache.http.HttpHost
import org.elasticsearch.action.bulk.BulkRequest
import org.elasticsearch.action.index.IndexRequest
import org.elasticsearch.client.{RequestOptions, RestClient,
RestClientBuilder, RestHighLevelClient}
import org.elasticsearch.common.xcontent.XContentType

/**
 * ES 工具类
 */
object MyESUtils {

    //声明 es 客户端
    var esClient : RestHighLevelClient = build()

    /**
     * 批量数据异步写入
     * 通过指定 id 实现异步
     */
    def bulkSaveIdempotent(sourceList: List[(String, AnyRef)], indexName: String): Unit = {
        if(sourceList != null && sourceList.nonEmpty){
            // BulkRequest 实际上就是由 7 多个单条 IndexRequest 的组合
            val bulkRequest = new BulkRequest()

            for ((docId, sourceObj) <- sourceList) {
                val indexRequest = new IndexRequest()
                indexRequest.index(indexName)
                val movieJsonStr: String = JSON.toJSONString(sourceObj,
                    new SerializeConfig(true))
                indexRequest.source(movieJsonStr, XContentType.JSON)
                indexRequest.id(docId)
                bulkRequest.add(indexRequest)
            }

            esClient.bulk(bulkRequest, RequestOptions.DEFAULT)
        }
    }
}
```

```
* 批量数据写入
*/
def bulkSave(sourceList: List[AnyRef], indexName: String):
Unit = {
  // BulkRequest 实际上就是由多个单条 IndexRequest 的组合
  val bulkRequest = new BulkRequest()

  for (source <- sourceList) {
    val indexRequest = new IndexRequest()
    indexRequest.index(indexName)
    val movieJsonStr: String = JSON.toJSONString(source, new
SerializeConfig(true))
    indexRequest.source(movieJsonStr, XContentType.JSON)
    bulkRequest.add(indexRequest)
  }

  esClient.bulk(bulkRequest, RequestOptions.DEFAULT)
}
/**
 * 单条数据幂等写入
 * 通过指定 id 实现幂等
 */
def saveIdempotent(source: (String, AnyRef), indexName:
String): Unit = {
  val indexRequest = new IndexRequest()
  indexRequest.index(indexName)
  val movieJsonStr: String = JSON.toJSONString(source._2, new
SerializeConfig(true))
  indexRequest.source(movieJsonStr, XContentType.JSON)
  indexRequest.id(source._1)
  esClient.index(indexRequest, RequestOptions.DEFAULT)
}
/**
 * 单条数据写入
 */
def save(source: AnyRef, indexName: String): Unit = {
  val indexRequest = new IndexRequest()
  indexRequest.index(indexName)
  val movieJsonStr: String = JSON.toJSONString(source, new
SerializeConfig(true))
  indexRequest.source(movieJsonStr, XContentType.JSON)
  esClient.index(indexRequest, RequestOptions.DEFAULT)
}

/**
 * 销毁
 */
def destroy(): Unit = {
  esClient.close()
  esClient = null
}

/**
 * 创建 es 客户端对象
 */
```

```
def build():RestHighLevelClient = {
    val builder: RestClientBuilder = RestClient.builder(new
    HttpHost("hadoop102",9200))
    val esClient = new RestHighLevelClient(builder)
    esClient
}

/**
 * 获取 esclient
 */
def getClient(): RestHighLevelClient = {
    esClient
}
}
```

3) 业务代码

主程序调用工具类批量写入 ES，其实我们前面已经使用 Redis 进行了去重操作，基本上是可以保证幂等性的。如果更严格的保证幂等性，那我们在批量向 ES 写数据的时候，指定 Index 的 id 即可。

```
// TODO 8.写入 ES
dauInfoDStream.foreachRDD(
    rdd => {
        rdd.foreachPartition(
            dauInfoIter => {
                // 因为是日活宽表，一天之内 mid 是不应该重复的
                //转换数据结构，保证幂等写入
                val dauInfos: List[(String, DauInfo)] =
                dauInfoIter.toList.map(dauInfo => (dauInfo.mid,dauInfo))
                if(dauInfos.size > 0 ){
                    //从数据中获取日期，拼接 ES 的索引名
                    val dauInfoT: (String, DauInfo) = dauInfos(0)
                    val dt = dauInfoT._2.dt
                    MyESUtils.bulkSaveIdempotent(dauInfos,s"gmall_dau_info_$dt")
                }
            }
        )
        // TODO 9.提交偏移量
        MyOffsetUtils.saveOffset(topic,groupId,offsetRanges)
    }
)
```

2.4 状态数据还原

2.4.1 问题现象

想象一个比较极端的情况，如果某个用户某天的首次访问数据写入 redis 后，接下来在写入到 es 的过程中,程序挂掉。会出现什么问题？

2.4.2 分析

程序挂掉，偏移量还未提交，重启后会触发数据的重试，但是因为 redis 中记录了相关的数据，所以该数据会被过滤掉。因此此数据，就再也无法进入 es，也就意味着丢失。

这个问题的本质就是，状态数据与最终数据库的数据以及偏移量，没有形成原子性事务造成的。

当然可以通过事务数据库的方式解决该问题，而我们的项目中没有选择使用支持事务的数据库，例如 MySQL 等。在既有的环境下我们依然有很多破解方案，例如进行状态还原，在启动程序前，将 ES 中已有的数据的 mid 提取出来，覆盖到 Redis 中，这样就能保证 Redis 和 ES 数据的同步。

2.4.3 解决方案

1) 在 ES 工具类中添加方法

```
/**
 * 查询指定的字段
 */
def searchField(indexName: String, fieldName: String): List[String] = {
    //判断索引是否存在
    val getIndexRequest: GetIndexRequest = new GetIndexRequest(indexName)
    val isExists: Boolean =
        esClient.indices().exists(getIndexRequest, RequestOptions.DEFAULT)
    if(!isExists){
        return null
    }
    //正常从 ES 中提取指定的字段
    val mids: ListBuffer[String] = ListBuffer[String]()
    val searchRequest: SearchRequest = new SearchRequest(indexName)
    val searchSourceBuilder: SearchSourceBuilder = new SearchSourceBuilder()
    searchSourceBuilder.fetchSource(fieldName, null).size(100000)
    searchRequest.source(searchSourceBuilder)
    val searchResponse: SearchResponse =
        esClient.search(searchRequest, RequestOptions.DEFAULT)
    val hits: Array[SearchHit] = searchResponse.getHits.getHits
    for (hit <- hits) {
        val sourceMap: util.Map[String, AnyRef] = hit.getSourceAsMap
        val mid: String = sourceMap.get(fieldName).toString
        mids.append(mid)
    }
    mids.toList
}
```

2) 在主程序中添加状态还原方法

```
/**
 * 状态还原
 *
 * 在每次启动实时任务时，进行一次状态还原。以 ES 为准，将所有的 mid 提取
 出来，覆盖到 Redis 中。
 */

def revertDauState(): Unit = {
  //从 ES 中查询到所有的 mid
  val date: LocalDate = LocalDate.now()
  val indexName : String = s"gmall_dau_info_1018_${date}"
  val fieldName : String = "mid"
  val mids: List[ String ] = MyEsUtils.searchField(indexName ,
fieldName)
  //删除 redis 中记录的状态（所有的 mid）
  val jedis: Jedis = MyRedisUtils.getJedisFromPool()
  val redisDauKey : String = s"DAU:${date}"
  jedis.del(redisDauKey)
  //将从 ES 中查询到的 mid 覆盖到 Redis 中
  if(mids != null && mids.size > 0 ){
    /*for (mid <- mids) {
      jedis.sadd(redisDauKey , mid )
    }*/
    val pipeline: Pipeline = jedis.pipelined()
    for (mid <- mids) {
      pipeline.sadd(redisDauKey , mid ) //不会直接到 redis 执行
    }

    pipeline.sync() // 到 redis 执行
  }

  jedis.close()
}
```

2) 调整 ES 参数

此方法由于需要把当日全部 mid 取出，会受到 es 默认的结果返回数限制。需要修改配置扩大返回结果数，如下：

```
PUT /_settings
{
  "index.max_result_window" : "5000000"
}
```

3) 修补主程序，在整个程序加载数据前进行状态还原

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

  //0.还原状态
  revertDauState()

  //1.环境
```



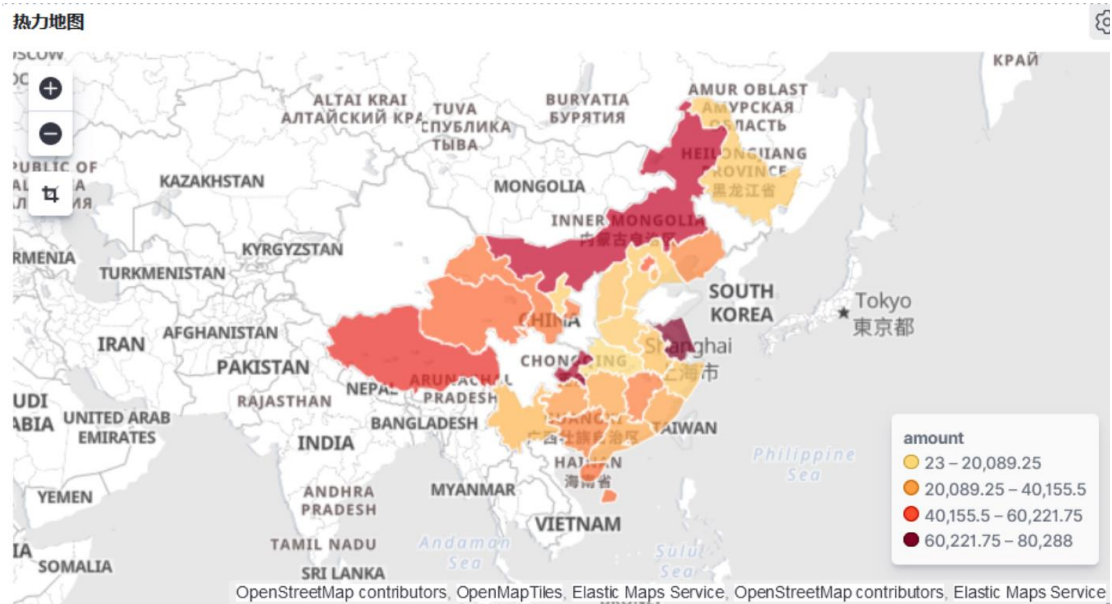
```
//2. 读取偏移量
```

```
//3. ....
```

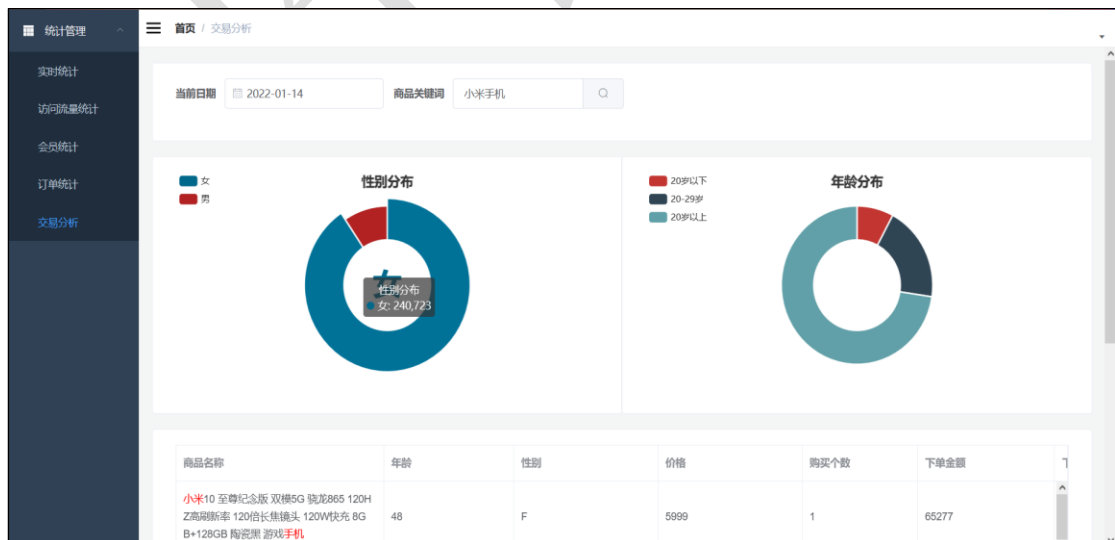
第 3 章 任务:订单业务宽表

3.1 需求举例

1) 订单热力地图



2) 订单交易分析



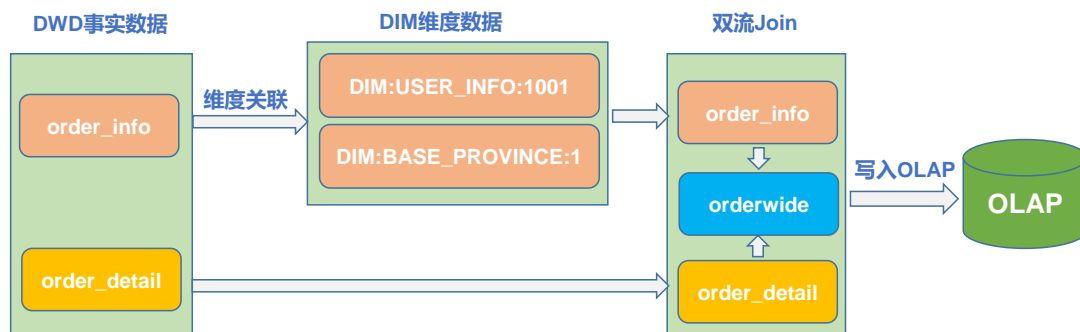
3.2 需求分析

由于在很多 OLAP 数据库对聚合过滤都是非常强大，但是大表间的关联都不是长项。

所以在数据进入 OLAP 前，尽量提前把数据关联组合好，不要在查询的时候临时进行 Join 操作。所谓提前关联好，在实时计算中进行流 Join。



维度关联 + 流Join



让天下没有难学的技术

3.2.1 流与维度关联

流与维度表之间的合并, 通常是事实表与维度表之间的关联。

3.2.2 双流 join

流与流之间的合并，也可以叫双流 join，通常是几乎同时产生的事实表的关联。

3.3 功能实现

3.3.1 准备 Bean

1) OrderInfo, 封装订单表数据

```
package com.atguigu.gmall.realtime.bean

case class OrderInfo(
    id: Long = 0L,
    province_id: Long = 0L,
    order_status: String = null,
    user_id: Long = 0L,
    total_amount: Double = 0D,
    activity_reduce_amount: Double = 0D,
    coupon_reduce_amount: Double = 0D,
    original_total_amount: Double = 0D,
    feight_fee: Double = 0D,
    feight_fee_reduce: Double = 0D,
    expire_time: String = null,
    refundable_time: String = null,
    create_time: String = null,
```

理得到

```
operate_time: String=null,
var create_date: String=null, // 把其他字段处

var create_hour: String=null,

var province_name:String=null,//查询维度表得到
var province_area_code:String=null,
var province_3166_2_code:String=null,
var province_iso_code:String=null,

var user_age :Int=0, //查询维度表得到
var user_gender:String=null
) {
}
}
```

2) OrderDetail, 封装订单详情数据

```
package com.atguigu.gmall.realtime.bean

case class OrderDetail(
    id : Long ,
    order_id :Long ,
    sku_id : Long ,
    order_price : Double ,
    sku_num : Long ,
    sku_name :String ,
    create_time : String ,
    split_total_amount: Double = 0D,
    split_activity_amount: Double =0D,
    split_coupon_amount:Double = 0D
) {
}
}
```

3) OrderWide, 订单宽表

```
package com.atguigu.gmall.realtime.bean

import com.atguigu.gmall.realtime.util.MyBeanUtils

case class OrderWide(
    var detail_id: Long =0L,
    var order_id:Long=0L,
    var sku_id: Long=0L,
    var order_price: Double=0D,
    var sku_num:Long=0L,
    var sku_name: String=null,
    var split_total_amount:Double=0D,
    var split_activity_amount:Double=0D,
    var split_coupon_amount:Double=0D,

    var province_id: Long=0L,
    var order_status: String=null,
    var user_id: Long=0L,
    var total_amount: Double=0D,
    var activity_reduce_amount: Double=0D,
```

```
var coupon_reduce_amount: Double=0D,
var original_total_amount: Double=0D,
var feight_fee: Double=0D,
var feight_fee_reduce: Double=0D,
var expire_time: String =null,
var refundable_time:String =null,
var create_time: String=null,
var operate_time: String=null,
var create_date: String=null,
var create_hour: String=null,

var province_name:String=null,
var province_area_code:String=null,
var province_3166_2_code:String=null,
var province_iso_code:String=null,

var user_age :Int=0,
var user_gender:String=null

) {

def this(orderInfo : OrderInfo ,orderDetail: OrderDetail){
  this
  mergeOrderInfo(orderInfo)
  mergeOrderDetail(orderDetail)
}

def mergeOrderInfo(orderInfo: OrderInfo): Unit ={
  if(orderInfo != null ){
    MyBeanUtils.copyProperties(orderInfo,this)
    this.order_id = orderInfo.id
  }
}

def mergeOrderDetail (orderDetail: OrderDetail): Unit ={
  if(orderDetail != null ){
    MyBeanUtils.copyProperties(orderDetail,this)
    this.detail_id = orderDetail.id
  }
}
}
```

3.3.2 消费 Kafka 数据

```
object DwDOrderApp {
  def main(args: Array[String]): Unit = {
    // .1 准备环境
    val sparkconf: SparkConf = new
    SparkConf().setAppName("dwd_dau_app").setMaster("local[4]")
    val ssc = new StreamingContext(sparkconf,Seconds(5))

    val orderInfoTopic = "DWD_ORDER_INFO_I"
    val orderDetailTopic = "DWD_ORDER_DETAIL_I"
    val groupId = "dwd_order_group"
```

```
//2.读取偏移量
val orderInfoOffsets: Map[TopicPartition, Long] =
MyOffsetUtils.getOffset(orderInfoTopic,groupId)
val orderDetailOffsets: Map[TopicPartition, Long] =
MyOffsetUtils.getOffset(orderDetailTopic,groupId)

//3.接收 Kafka 数据
//order_info
var orderInfoKafkaDStream: DStream[ConsumerRecord[String,
String]] = null
if(orderInfoOffsets != null && orderInfoOffsets.nonEmpty ){
    orderInfoKafkaDStream =
MyKafkaUtils.getKafkaDStream(orderInfoTopic,ssc,orderInfoOffsets,
groupId)
}else{
    orderInfoKafkaDStream =
MyKafkaUtils.getKafkaDStream(orderInfoTopic,ssc,groupId)
}

//order_detail
var orderDeatilKafkaDStream: DStream[ConsumerRecord[String,
String]] = null
if(orderDetailOffsets != null &&
orderDetailOffsets.nonEmpty ){
    orderDeatilKafkaDStream =
MyKafkaUtils.getKafkaDStream(orderDetailTopic,ssc,orderDetailOffs
ets,groupId)
}else{
    orderDeatilKafkaDStream =
MyKafkaUtils.getKafkaDStream(orderDetailTopic,ssc,groupId)
}

//4. 提取偏移量结束点
//order_info

var orderInfoOffsetRanges: Array[OffsetRange] = null
orderInfoKafkaDStream = orderInfoKafkaDStream.transform(
    rdd => {
        orderInfoOffsetRanges =
rdd.asInstanceOf[HasOffsetRanges].offsetRanges
    }
)

//order_detail
var orderDetailOffsetRanges: Array[OffsetRange] = null
orderDeatilKafkaDStream = orderDeatilKafkaDStream.transform(
    rdd => {
        orderDetailOffsetRanges =
rdd.asInstanceOf[HasOffsetRanges].offsetRanges
    }
)
```

```
//5. 转换结构

val orderInfoDStream: DStream[OrderInfo] =
orderInfoKafkaDStream.map(
  record => {
    val jsonStr: String = record.value()
    val orderInfo: OrderInfo = JSON.parseObject(jsonStr,
classOf[OrderInfo])
    orderInfo
  }
)

val orderDetailDStream: DStream[OrderDetail] =
orderDeatilKafkaDStream.map(
  record => {
    val jsonStr: String = record.value()
    val orderDetail: OrderDetail = JSON.parseObject(jsonStr,
classOf[OrderDetail])
    orderDetail
  }
)
}
```

3.3.3 维度合并

```
//6. 维度合并 (补充用户年龄性别、地区信息)
val orderInfoWithDimDStream: DStream[OrderInfo] =
orderInfoDStream.mapPartitions(
  orderInfoIter => {
    val jedis: Jedis = MyRedisUtils.getJedisClient
    val orderInfoList: List[OrderInfo] = orderInfoIter.toList
    for (orderInfo <- orderInfoList) {

      //补充用户信息
      val userInfoKey = s"DIM:USER_INFO:${orderInfo.user_id}"
      val userInfoJson: String = jedis.get(userInfoKey)
      val userInfoJsonObj: JSONObject =
JSON.parseObject(userInfoJson)
      //获取性别
      orderInfo.user_gender =
userInfoJsonObj.getString("gender")
      //获取年龄
      val birthday: String =
userInfoJsonObj.getString("birthday")
      val birthdayDate: LocalDate = LocalDate.parse(birthday)
      val nowDate: LocalDate = LocalDate.now()
      val period: Period = Period.between(birthdayDate,
nowDate)
      val age: Int = period.getYears
      orderInfo.user_age = age

      //补充日期字段
      val dateTimeArr: Array[String] =
orderInfo.create_time.split(" ")
    }
  }
)
```

```

        orderInfo.create_date = dateTimeArr(0)
        orderInfo.create_hour = dateTimeArr(1).split(":")(0)

        //TODO 补充地区信息

        val provinceKey = "DIM:BASE_PROVINCE:" +
orderInfo.province_id
        val provinceJson: String = jedis.get(provinceKey)
        val provinceJsonObj: JSONObject =
JSON.parseObject(provinceJson)
        orderInfo.province_name =
provinceJsonObj.getString("name")
        orderInfo.province_area_code =
provinceJsonObj.getString("area_code") // ali datav quickbi
baidu suger
        orderInfo.province_3166_2_code =
provinceJsonObj.getString("iso_3166_2") // kibana
        orderInfo.province_iso_code =
provinceJsonObj.getString("iso_code") // superset

    }

    jedis.close()
    orderInfoList.toIterator
}
)

```

3.3.4 双流 join

1) 分析

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。

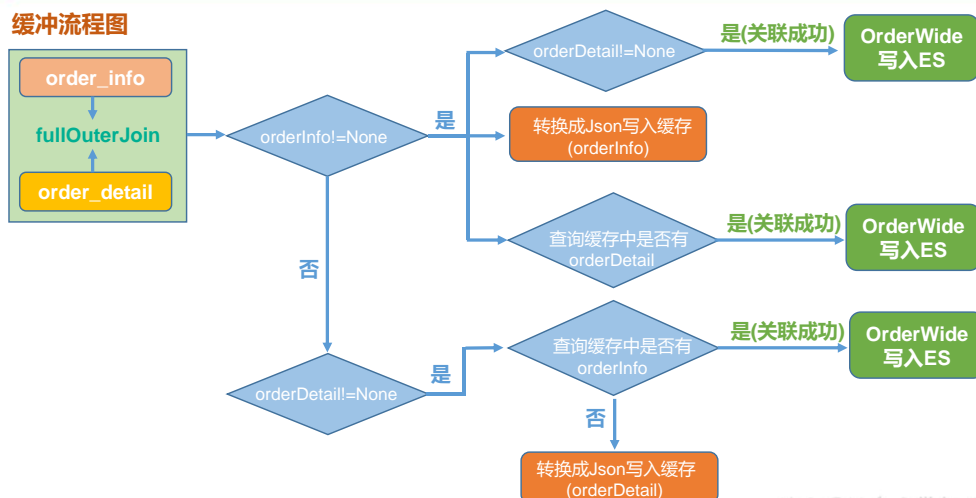
2) 解决策略

- (1) 增大采集周期
- (2) 利用滑动窗口进行 join 然后再进行去重
- (3) 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) 程序流程图（缓存策略）



缓冲流程图



让天下没有难学的技术

4) 代码实现

```

//双流 join
// 如果要做 Join 操作，必须是 DStream[K,V] 和 DStream[K,V]
val orderInfoWithKeyDStream: DStream[(Long, OrderInfo)] =
    orderInfoWithDimDStream.map(orderInfo =>
        (orderInfo.id,orderInfo))

val orderDetailWithKeyDStream: DStream[(Long, OrderDetail)] =
    orderDetailDStream.map(
        orderDetail =>
        (orderDetail.order_id,orderDetail))

//join 只能实现统一批次的数据进行 Join,如果有数据延迟，延迟的数据就不能
//join 成功，就会有数据丢失。
//val joinDStream: DStream[(Long, (OrderInfo, OrderDetail))]
= orderInfoWithKeyDStream.join(orderDetailWithKeyDStream)
//joinDStream.print(1000)

//通过状态或者缓存来解决。

val orderJoinDStream: DStream[(Long, (Option[OrderInfo],
Option[OrderDetail]))] =
orderInfoWithKeyDStream.fullOuterJoin(orderDetailWithKeyDStream)

val orderWideDStream: DStream[OrderWide] =
orderJoinDStream.flatMap {
    case (orderId, (orderInfoOpt, orderDetailOpt)) => {
        val orderWideList: ListBuffer[OrderWide] =
ListBuffer[OrderWide]()
        val jedis: Jedis = MyRedisUtils.getJedisClient
        //1.主表在
        if (orderInfoOpt != None) {
            val orderInfo: OrderInfo = orderInfoOpt.get
            //1.1 判断从表是否存在

```



```
if (orderDetailOpt != None) {
    //如果从表在， 将主和从合并
    val orderDetail: OrderDetail = orderDetailOpt.get
    val orderWide = new OrderWide(orderInfo, orderDetail)
    orderWideList.append(orderWide)
}

//1.2 主表写缓存
//type: String
//key : ORDER_JOIN:ORDER_INFO:[ID]
//value : orderInfoJson
//写入 API: set
//读取 API: get
//过期时间: 小时~天 24 小时
val orderInfoKey =
s"ORDER_JOIN:ORDER_INFO:${orderInfo.id}"
val orderInfoJson: String = JSON.toJSONString(orderInfo,
new SerializeConfig(true))
jedis.setex(orderInfoKey, 3600 * 24, orderInfoJson)

//1.3 主表读缓存
// type : set
// key : ORDER_JOIN:ORDER_DETAIL:[ORDERID]
// value : orderDetailJson
//写入 API: sadd
//读取 API: smembers
//过期时间: 小时~天 24 小时
val orderDetailKey =
s"ORDER_JOIN:ORDER_DETAIL:${orderInfo.id}"
val orderDetailJsonSet: util.Set[String] =
jedis.smembers(orderDetailKey)
import scala.collection.JavaConverters._
if (orderDetailJsonSet != null &&
orderDetailJsonSet.size() > 0) {
    for (orderDetailJson <- orderDetailJsonSet.asScala) {
        val orderDetail: OrderDetail =
JSON.parseObject(orderDetailJson, classOf[OrderDetail])
        val orderWide = new OrderWide(orderInfo,
orderDetail)
        orderWideList.append(orderWide)
    }
}
} else {
    //2. 主表不在，从表在
    val orderDetail: OrderDetail = orderDetailOpt.get
    //2.1 从表读缓存
    val orderInfoKey =
s"ORDER_JOIN:ORDER_INFO:${orderDetail.order_id}"
    val orderInfoJson: String = jedis.get(orderInfoKey)
    if (orderInfoJson != null && orderInfoJson.nonEmpty) {
        val orderInfo: OrderInfo =
JSON.parseObject(orderInfoJson, classOf[OrderInfo])
        val orderWide = new OrderWide(orderInfo, orderDetail)
```

```

        orderWideList.append(orderWide)
    }

    //2.2 从表写缓存
    val orderDetailKey = "ORDER_JOIN:ORDER_DETAIL:${orderDetail.order_id}"
    val orderDetailJson = JSON.toJSONString(orderDetail, new SerializeConfig(true))
    jedis.sadd(orderDetailKey, orderDetailJson)
    jedis.expire(orderDetailKey, 3600 * 24)
}

jedis.close()
orderWideList
}
}
orderWideDStream.print(100)

```

3.3.5 写入 ES 并提交偏移量

1) 建索引模板

```

PUT _template/gmall_order_wide_template
{
  "index_patterns": ["gmall_order_wide*"],
  "settings": {
    "number_of_shards": 3
  },
  "aliases": {
    "{index}-query": {},
    "gmall_order_wide-query": {}
  },
  "mappings": {
    "properties": {
      "detail_id": {
        "type": "keyword"
      },
      "order_id": {
        "type": "keyword"
      },
      "sku_id": {
        "type": "keyword"
      },
      "sku_num": {
        "type": "long"
      },
      "sku_name": {
        "type": "text",
        "analyzer": "ik_max_word"
      },
      "order_price": {
        "type": "float"
      },
      "split_total_amount": {
        "type": "float"
      }
    }
  }
}

```

```
"split_activity_amount" : {
  "type" : "float"
},
"split_coupon_amount" : {
  "type" : "float"
},
"province_id" : {
  "type" : "keyword"
},
"order_status" : {
  "type" : "keyword"
},
"user_id" : {
  "type" : "keyword"
},
"total_amount" : {
  "type" : "float"
},
"activity_reduce_amount" : {
  "type" : "float"
},
"coupon_reduce_amount" : {
  "type" : "float"
},
"original_total_amount" : {
  "type" : "float"
},
"feight_fee" : {
  "type" : "float"
},
"feight_fee_reduce" : {
  "type" : "float"
},
"expire_time" : {
  "type" : "date" ,
  "format" : "yyyy-MM-dd HH:mm:ss"
},
"refundable_time" : {
  "type" : "date" ,
  "format" : "yyyy-MM-dd HH:mm:ss"
},
"create_time" : {
  "type" : "date" ,
  "format" : "yyyy-MM-dd HH:mm:ss"
},
"operate_time" : {
  "type" : "date" ,
  "format" : "yyyy-MM-dd HH:mm:ss"
},
"create_date" : {
  "type" : "keyword"
},
"create_hour" : {
  "type" : "keyword"
}
```

```
    },
    "province_name" : {
      "type" : "keyword"
    },
    "province_area_code" : {
      "type" : "keyword"
    },
    "province_3166_2_code" : {
      "type" : "keyword"
    },
    "province_iso_code" : {
      "type" : "keyword"
    },
    "user_age" : {
      "type" : "long"
    },
    "user_gender" : {
      "type" : "keyword"
    }
  }
}
```

2) 保存 ES 代码

```
//写入 es
orderWideDStream.foreachRDD(
  rdd => {
    //driver
    rdd.foreachPartition(
      orderWideIter => {
        //executor
        val orderWideList: List[(String, OrderWide)] =
          orderWideIter.toList.map(orderWide =>
            (orderWide.detail_id.toString , orderWide))
        if(orderWideList.size > 0 ){
          val orderWideT: (String , OrderWide) =
            orderWideList(0)
          val dt: String = orderWideT._2.create_date
          val indexName = s"gmall_order_wide_$dt"

          MyESUtils.bulkSaveIdempotent(orderWideList,indexName)

        }
      }
    )
    //提交偏移量
    MyOffsetUtils.saveOffset(orderInfoTopic,groupId ,
      orderInfoOffsetRanges)

    MyOffsetUtils.saveOffset(orderDetailTopic,groupId,orderDetailOffsetRanges)
  }
)
```

第 4 章 总结