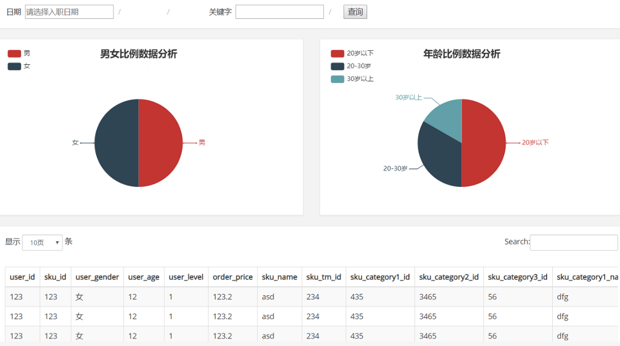
电商数仓项目-实时计算（灵活分析）

# 需求分析

## 1 灵活查询的场景

数仓(实时与离线)中存储了大量的明细数据，但是hadoop存储的数仓计算必须经过mr ，所以即时交互性非常糟糕。为了方便数据分析人员查看信息，数据平台需要提供一个能够根据文字及选项等条件，进行灵活分析判断的数据功能。



## **2 需求详细**：

**输入参数：**

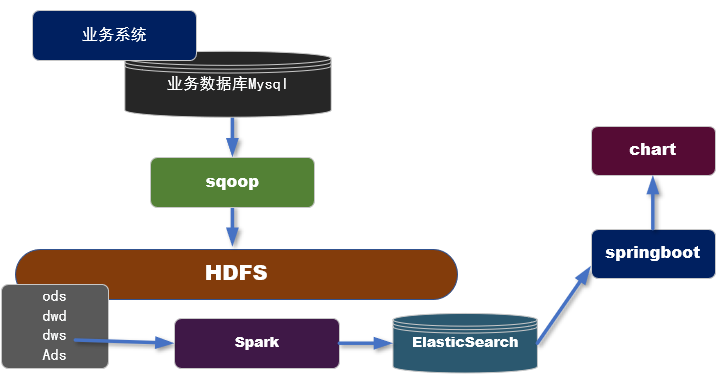
|  |  |
| --- | --- |
| 日期 | 查询数据的日期 |
| 关键字 | 根据商品名称涉及到的词进行搜索 |

**返回结果**

|  |  |  |
| --- | --- | --- |
| 饼图 | 男女比例占比 | 男 ，女 |
| **年龄比例占比** | 20岁以下，20-30岁 ，30岁以上 |
| 购买行为数据明细 | | 包括，用户id,性别年龄，级别，购买的时间，商品价格，订单状态，等信息。  可翻页。 |

# 第二章 架构分析

## 1 T+1 模式



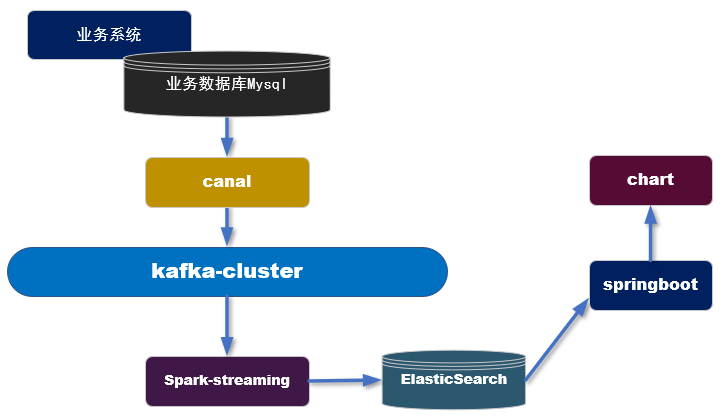
### 1.1 实现步骤：

1. 利用sqoop等工具，从业务数据库中批量抽取数据
2. 利用数仓作业，在dws层组织宽表（用户购买行为）
3. 开发spark的批处理任务，把dws层的宽表导入到ES中
4. 从ES读取数据发布接口，对接可视化模块。

### 1.2 特点：

* **优点**： 可以利用在离线作业处理好的dws层宽表，直接导出一份到ES进行快速交互的分析。
* **缺点**：因为要用离线处理的后的结果在放入ES,所以时效性等同于离线数据。

## 2 T+0 模式



### 2.1 实现步骤：

1. 利用canal抓取对应的数据表的实时新增变化数据,推送到Kafka
2. 在spark-streaming中进行转换，过滤，关联组合成宽表的结构。
3. 保存到ES中
4. 从ES读取数据发布接口，对接可视化模块。

### 2.2 特点

* **优点**： 实时产生数据，时效性非常高。
* **缺点**： 因为从kafka中得到的是原始数据，所以要利用spark-streaming要进行加工处理，相对来说要比批处理方式麻烦，比如join操作。

# 第三章 实时采集数据

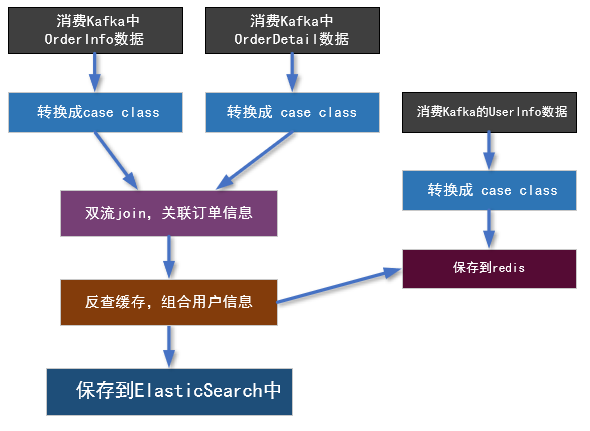
## 1 在canal 模块中增加要追踪的表

代码

|  |
| --- |
| **public class** CanalHandler {   **private** List<CanalEntry.RowData> rowDatasList;  String tableName;  CanalEntry.EventType eventType;   **public** CanalHandler(List<CanalEntry.RowData> rowDatasList, String tableName, CanalEntry.EventType eventType) {  **this**.rowDatasList = rowDatasList;  **this**.tableName = tableName;  **this**.eventType = eventType;  }  //根据不同业务的类型发送不同主题  **public void** handle(){  **if**(eventType.equals(CanalEntry.EventType.INSERT)&&tableName.equals(**"order\_info"**)){  sendRowList2Kafka(GmallConstants.KAFKA\_TOPIC\_ORDER);  }**else if**((eventType.equals(CanalEntry.EventType.INSERT)||eventType.equals(CanalEntry.EventType.UPDATE))&&tableName.equals(**"user\_info"**)){  sendRowList2Kafka(GmallConstants.KAFKA\_TOPIC\_USER);  }**else if**(eventType.equals(CanalEntry.EventType.INSERT)&&tableName.equals(**"order\_detail"**)){  sendRowList2Kafka(GmallConstants.KAFKA\_TOPIC\_ORDER\_DETAIL);  }   }  // 统一处理发送kafka  **private void** sendRowList2Kafka(String kafkaTopic){  **for** (CanalEntry.RowData rowData : rowDatasList) {  List<CanalEntry.Column> afterColumnsList = rowData.getAfterColumnsList();  JSONObject jsonObject = **new** JSONObject();  **for** (CanalEntry.Column column : afterColumnsList) {   System.out.println(column.getName()+**"--->"**+column.getValue());  jsonObject.put(column.getName(),column.getValue());  }  **try** {  Thread.sleep(**new** Random().nextInt(5)\*1000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  MyKafkaSender.send(kafkaTopic,jsonObject.toJSONString());  }   }  } |

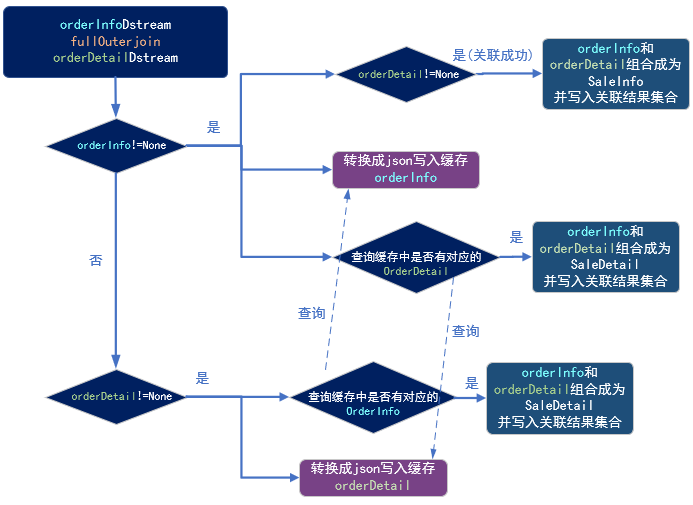
# 第四章 实时数据处理

## 1 数据处理流程



## 2 难点：双流join

### 2.1 程序流程图



### 2.2 代码

#### 样例类

SaleDetail

|  |
| --- |
| **import** java.text.SimpleDateFormat **import** java.util     **case class** SaleDetail(  **var** order\_detail\_id:String =**null**,  **var** order\_id: String=**null**,  **var** order\_status:String=**null**,  **var** create\_time:String=**null**,  **var** user\_id: String=**null**,  **var** sku\_id: String=**null**,  **var** user\_gender: String=**null**,  **var** user\_age: Int=0,  **var** user\_level: String=**null**,  **var** sku\_price: Double=0D,  **var** sku\_name: String=**null**,  **var** dt: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**.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.toDouble    }  }   **def** mergeUserInfo(userInfo: UserInfo): Unit ={  **if**(userInfo!=**null**){  **this**.user\_id=userInfo.id   **val** formattor = **new** SimpleDateFormat(**"yyyy-MM-dd"**)  **val** date: util.Date = formattor.parse(userInfo.birthday)  **val** curTs: Long = System.*currentTimeMillis*()  **val** betweenMs= curTs-date.getTime  **val** age=betweenMs/1000L/60L/60L/24L/365L   **this**.user\_age= age.toInt  **this**.user\_gender=userInfo.gender  **this**.user\_level=userInfo.user\_level   }  }    } |

OrderDetail

|  |
| --- |
| **case class** OrderDetail(  id:String ,  order\_id: String,  sku\_name: String,  sku\_id: String,  order\_price: String,  img\_url: String,  sku\_num: String  ) {  } |

UserInfo

|  |
| --- |
| **case class** UserInfo(id:String ,  login\_name:String,  user\_level:String,  birthday:String,  gender:String) {  } |

#### SaleApp

|  |
| --- |
| **def** main(args: Array[String]): Unit = {  **val** sparkConf: SparkConf = **new** SparkConf().setMaster(**"local[\*]"**).setAppName(**"sale\_app"**)  **val** ssc = **new** StreamingContext(sparkConf,*Seconds*(5))   **val** inputOrderDstream: InputDStream[ConsumerRecord[String, String]] = MyKafkaUtil.*getKafkaStream*(GmallConstant.*KAFKA\_TOPIC\_ORDER*,ssc)  **val** inputOrderDetailDstream: InputDStream[ConsumerRecord[String, String]] = MyKafkaUtil.*getKafkaStream*(GmallConstant.*KAFKA\_TOPIC\_ORDER\_DETAIL*,ssc)   *//整理 转换* **val** orderInfoDstream: DStream[OrderInfo] = inputOrderDstream.map { record =>  **val** jsonString: String = record.value()  *// 1 转换成case class* **val** orderInfo: OrderInfo = JSON.*parseObject*(jsonString, *classOf*[OrderInfo])  *// 2 脱敏 电话号码 1381\*\*\*\*\*\*\** **val** telTuple: (String, String) = orderInfo.consignee\_tel.splitAt(4)  orderInfo.consignee\_tel = telTuple.\_1 + **"\*\*\*\*\*\*\*"** *// 3 补充日期字段* **val** datetimeArr: Array[String] = orderInfo.create\_time.split(**" "**)  orderInfo.create\_date = datetimeArr(0) *//日期* **val** timeArr: Array[String] = datetimeArr(1).split(**":"**)  orderInfo.create\_hour = timeArr(0) *//小时* orderInfo  }    **val** orderDetailDStream: DStream[OrderDetail] = inputOrderDetailDstream.map { record =>  **val** jsonString: String = record.value()  **val** orderDetail: OrderDetail = JSON.*parseObject*(jsonString, *classOf*[OrderDetail])  orderDetail  }  *// 双流join 前 要把流变为kv 结构* **val** orderInfoWithKeyDstream: DStream[(String, OrderInfo)] = orderInfoDstream.map(orderInfo =>(orderInfo.id,orderInfo))  **val** orderDetailWithKeyDstream: DStream[(String, OrderDetail)] = orderDetailDStream.map(orderDetail=>(orderDetail.order\_id,orderDetail))   *//为了不管是否能够关联左右 ，都要保留左右两边的数据 采用full join* **val** fullJoinDStream: DStream[(String, (Option[OrderInfo], Option[OrderDetail]))] = orderInfoWithKeyDstream.fullOuterJoin(orderDetailWithKeyDstream)   **val** saleDetailDstream: DStream[SaleDetail] = fullJoinDStream.mapPartitions { partitionItr =>  **val** jedis: Jedis = RedisUtil.*getJedisClient* **implicit val** formats = org.json4s.DefaultFormats  **val** saleDetailList = ListBuffer[SaleDetail]()  **for** ((orderId, (orderInfoOption, orderDetailOption)) <- partitionItr) {  **if** (orderInfoOption != None) {  *println*(**" 主表有数据 ！"**)  **val** orderInfo: OrderInfo = orderInfoOption.get  *// 1 组合关联* **if** (orderDetailOption != None) {  *println*(**" 主表有数据 ！且从表有数据 成功关联"**)  **val** orderDetail: OrderDetail = orderDetailOption.get  *//组合成一个SaleDetail* **val** saleDetail = **new** SaleDetail(orderInfo, orderDetail)  *// 存放到sale集合中* saleDetailList += saleDetail  }   *//2 写缓存 key 类型 : string key名 [order\_info:order\_id] value -> orderinfoJson  println*(**" 主表有数据 ！写入缓存"**)  **val** orderInfoKey = **"order\_info:"** + orderId   *// fastjson无法转换 case class 为json  // val orderInfoJson: String = JSON.toJSONString(orderInfo)  // json4s* **val** orderInfoJson: String = Serialization.*write*(orderInfo)  jedis.setex(orderInfoKey, 300, orderInfoJson)   *// 3 查询缓存* **val** orderDetailKey = **"order\_detail:"** + orderId  **val** orderDetailJson: String = jedis.get(orderDetailKey)  **val** orderDetailSet: util.Set[String] = jedis.smembers(orderDetailKey)  **import** collection.JavaConversions.\_  **for** ( orderDetailJson <- orderDetailSet ) {  *println*(**" 查询到从表缓存数据进行关联"**)  **val** orderDetail: OrderDetail = JSON.*parseObject*(orderDetailJson, *classOf*[OrderDetail])  **val** saleDetail = **new** SaleDetail(orderInfo, orderDetail)  saleDetailList += saleDetail  }  } **else if** (orderDetailOption != None) { *//主表没有数据 从表有数据  println*(**"主表没有数据 从表有数据 "**)  **val** orderDetail: OrderDetail = orderDetailOption.get  *//1 查询缓存 查询主表  println*(**"查询主表缓存"**)  **val** orderInfoKey = **"order\_info:"** + orderId  **val** orderInfoJson: String = jedis.get(orderInfoKey)  **if** (orderInfoJson != **null** && orderInfoJson.size > 0) {  **val** orderInfo: OrderInfo = JSON.*parseObject*(orderInfoJson, *classOf*[OrderInfo])  **val** saleDetail = **new** SaleDetail(orderInfo, orderDetail)  saleDetailList += saleDetail  }  *// 2 从表写缓存 // 从表缓存设计问题 //要体现一个主表下多个从表的结构1：n keytype: set key order\_detail:order\_id members -> 多个 order\_detailjson  println*(**"写从表缓存"**)  **val** orderDetailKey = **"order\_detail:"** + orderId  **val** orderDetailJson: String = Serialization.*write*(orderDetail)  jedis.sadd(orderDetailKey,orderDetailJson)  jedis.expire(orderDetailKey,300)  *//jedis.setex(orderDetailKey, 300, orderDetailJson)* }  }  jedis.close()  saleDetailList.toIterator  }    saleDetailDstream.foreachRDD{rdd=>  *println*(rdd.collect().mkString(**"\n"**))  }     ssc.start()  ssc.awaitTermination()  } |

#### scala 中把 样例类转换成为JSON字符串

pom.xml

|  |
| --- |
| <**dependency**>  <**groupId**>org.json4s</**groupId**>  <**artifactId**>json4s-native\_2.11</**artifactId**>  <**version**>3.5.4</**version**> </**dependency**> |

|  |
| --- |
| **import** org.json4s.native.Serialization  **implicit val** formats=org.json4s.DefaultFormats  **val** orderInfoJson: String = Serialization.*write*(orderInfo) |

## 3 采集userInfo进入缓存

|  |
| --- |
| **val** inputUserDstream: InputDStream[ConsumerRecord[String, String]] = MyKafkaUtil.*getKafkaStream*(GmallConstants.*KAFKA\_TOPIC\_USER*,ssc)  *// 把userInfo 保存到缓存中* inputUserDstream.map{record=>  **val** userInfo: UserInfo = JSON.*parseObject*(record.value(), *classOf*[UserInfo])  userInfo }.foreachRDD{rdd:RDD[UserInfo]=>  **val** userList: List[UserInfo] = rdd.collect().toList  **val** jedis: Jedis = RedisUtil.*getJedisClient* **implicit val** formats=org.json4s.DefaultFormats  **for** (userInfo <- userList ) { *// string set list hash zset  //设计user\_info redis type hash key user\_info , field user\_id ,value user\_info\_json* **val** userkey=**"user\_info"  val** userJson: String = Serialization.*write*(userInfo)  jedis.hset(userkey,userInfo.id,userJson)  }  jedis.close() } |

## 4 反查缓存并关联userInfo

|  |
| --- |
| **val** fullSaleDetailDstream: DStream[SaleDetail] = saleDetailDstream.mapPartitions { saleIter =>  **val** jedis: Jedis = RedisUtil.*getJedisClient* **val** userList: ListBuffer[SaleDetail] = ListBuffer[SaleDetail]()  **for** (saleDetail <- saleIter) {   **val** userInfoJson: String = jedis.hget(**"user\_info"**, saleDetail.user\_id)  **val** userinfo: UserInfo = JSON.*parseObject*(userInfoJson, *classOf*[UserInfo])  saleDetail.mergeUserInfo(userinfo)  userList += saleDetail  }  jedis.close()  userList.toIterator } |

## 5 保存购买明细进入ES中

### 5.1 ES索引建立

|  |
| --- |
| PUT gmall0105\_sale\_detail  {  "mappings" : {  "\_doc" : {  "properties" : {  "order\_detail\_id" : {  "type" : "keyword"  },  "order\_id" : {  "type" : "keyword"  },  "create\_time" : {  "type" : "date" ,  "format" : "yyyy-MM-dd HH:mm:ss"  },  "dt" : {  "type" : "date"  },  "order\_status" : {  "type" : "keyword"  },  "sku\_id" : {  "type" : "keyword"  },  "sku\_name" : {  "type" : "text",  "analyzer": "ik\_max\_word"  },  "sku\_price" : {  "type" : "float"  },  "user\_age" : {  "type" : "long"  },  "user\_gender" : {  "type" : "keyword"  },  "user\_id" : {  "type" : "keyword"  },  "user\_level" : {  "type" : "keyword",  "index" : false  }  }  }  }  } |

### 5.2 保存ES代码

|  |
| --- |
| fullSaleDetailDstream.foreachRDD{rdd=>  **val** saleDetailList: List[SaleDetail] = rdd.collect().toList  **val** saleDetailWithKeyList: List[(String, SaleDetail)] = saleDetailList.map(saleDetail=>(saleDetail.order\_detail\_id,saleDetail))  MyEsUtil.*insertBulk*(GmallConstants.*ES\_INDEX\_SALE\_DETAIL*,saleDetailWithKeyList)  } |

# 第五章 灵活查询数据接口开发

## 1 传入路径及参数

|  |
| --- |
| http://localhost:8070/sale\_detail?date=2019-04-01&&startpage=1&size=5&keyword=手机小米 |

## 2 返回值

|  |
| --- |
| {"total":62,"stat":[{"options":[{"name":"20岁以下","value":0.0},{"name":"20岁到30岁","value":25.8},{"name":"30岁及30岁以上","value":74.2}],"title":"用户年龄占比"},{"options":[{"name":"男","value":38.7},{"name":"女","value":61.3}],"title":"用户性别占比"}],"detail":[{"user\_id":"9","sku\_id":"8","user\_gender":"M","user\_age":49.0,"user\_level":"1","sku\_price":8900.0,"sku\_name":"Apple iPhone XS Max (A2104) 256GB 深空灰色 移动联通电信4G手机 双卡双待","sku\_tm\_id":"86","sku\_category1\_id":"2","sku\_category2\_id":"13","sku\_category3\_id":"61","sku\_category1\_name":"手机","sku\_category2\_name":"手机通讯","sku\_category3\_name":"手机","spu\_id":"1","sku\_num":6.0,"order\_count":2.0,"order\_amount":53400.0,"dt":"2019-02-14","es\_metadata\_id":"wPdM7GgBQMmfy2BJr4YT"},{"user\_id":"5","sku\_id":"8","user\_gender":"F","user\_age":36.0,"user\_level":"4","sku\_price":8900.0,"sku\_name":"Apple iPhone XS Max (A2104) 256GB 深空灰色 移动联通电信4G手机 双卡双待","sku\_tm\_id":"86","sku\_category1\_id":"2","sku\_category2\_id":"13","sku\_category3\_id":"61","sku\_category1\_name":"手机","sku\_category2\_name":"手机通讯","sku\_category3\_name":"手机","spu\_id":"1","sku\_num":5.0,"order\_count":1.0,"order\_amount":44500.0,"dt":"2019-02-14","es\_metadata\_id":"wvdM7GgBQMmfy2BJr4YT"},{"user\_id":"19","sku\_id":"8","user\_gender":"F","user\_age":43.0,"user\_level":"5","sku\_price":8900.0,"sku\_name":"Apple iPhone XS Max (A2104) 256GB 深空灰色 移动联通电信4G手机 双卡双待","sku\_tm\_id":"86","sku\_category1\_id":"2","sku\_category2\_id":"13","sku\_category3\_id":"61","sku\_category1\_name":"手机","sku\_category2\_name":"手机通讯","sku\_category3\_name":"手机","spu\_id":"1","sku\_num":7.0,"order\_count":2.0,"order\_amount":62300.0,"dt":"2019-02-14","es\_metadata\_id":"xvdM7GgBQMmfy2BJr4YU"},{"user\_id":"15","sku\_id":"8","user\_gender":"M","user\_age":66.0,"user\_level":"4","sku\_price":8900.0,"sku\_name":"Apple iPhone XS Max (A2104) 256GB 深空灰色 移动联通电信4G手机 双卡双待","sku\_tm\_id":"86","sku\_category1\_id":"2","sku\_category2\_id":"13","sku\_category3\_id":"61","sku\_category1\_name":"手机","sku\_category2\_name":"手机通讯","sku\_category3\_name":"手机","spu\_id":"1","sku\_num":3.0,"order\_count":1.0,"order\_amount":26700.0,"dt":"2019-02-14","es\_metadata\_id":"xvdM7GgBQMmfy2BJr4YU"}]  } |

## 3 编写DSL语句

|  |
| --- |
| GET gmall0105\_sale\_detail/\_search  {  "query": {  "bool": {  "filter": {  "term": {  "dt": "2019-02-14"  }  },  "must": [  {"match":{  "sku\_name": {  "query": "小米手机",  "operator": "and"  }  }    }  ]  }  }  , "aggs": {  "groupby\_age": {  "terms": {  "field": "user\_age"  }  }  }  ,  "size": 2  , "from": 0  } |

## 4 代码开发

### 4.1 代码清单

|  |  |  |
| --- | --- | --- |
| bean | Stat | 饼图 |
| Option | 饼图中的选项 |
| 控制层 | PublisherController | 增加getSaleDetail方法，调用服务层方法得到数据并根据web接口和参数组织整理返回值 |
| 服务层 | PublisherService | 增加getSaleDetail方法 |
| PublisherServiceImpl | 实现 getSaleDetail方法，依据DSL语句查询ElasticSearch |

### 4.2 pom.xml

|  |
| --- |
| *<!--- ES依赖包-->* <**dependency**>  <**groupId**>io.searchbox</**groupId**>  <**artifactId**>jest</**artifactId**>  <**version**>5.3.3</**version**>  </**dependency**>  <**dependency**>  <**groupId**>net.java.dev.jna</**groupId**>  <**artifactId**>jna</**artifactId**>  <**version**>4.5.2</**version**> </**dependency**>  <**dependency**>  <**groupId**>org.codehaus.janino</**groupId**>  <**artifactId**>commons-compiler</**artifactId**>  <**version**>2.7.8</**version**> </**dependency**>  <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-data-elasticsearch</**artifactId**> </**dependency**> |

### 4.3 配置 application.properties

|  |
| --- |
| *#es* **spring.elasticsearch.jest.uris**=**http://hadoop1:9200** |

### 4.4 PublisherServiceImpl

|  |
| --- |
| @Override **public** Map getSaleDetail(String date, String keyword, **int** pageSize, **int** pageNo) {     SearchSourceBuilder searchSourceBuilder = **new** SearchSourceBuilder();  *//过滤 匹配* BoolQueryBuilder boolQueryBuilder = **new** BoolQueryBuilder();  boolQueryBuilder.filter(**new** TermQueryBuilder(**"dt"**,date));  boolQueryBuilder.must(**new** MatchQueryBuilder(**"sku\_name"**,keyword).operator(MatchQueryBuilder.Operator.***AND***));  searchSourceBuilder.query(boolQueryBuilder);  *// 性别聚合* TermsBuilder genderAggs = AggregationBuilders.*terms*(**"groupby\_user\_gender"**).field(**"user\_gender"**).size(2);  searchSourceBuilder.aggregation(genderAggs);  *// 年龄聚合* TermsBuilder ageAggs = AggregationBuilders.*terms*(**"groupby\_user\_age"**).field(**"user\_age"**).size(100);  searchSourceBuilder.aggregation(ageAggs);  *// 行号= （页面-1） \* 每页行数* searchSourceBuilder.from((pageNo-1)\*pageSize);  searchSourceBuilder.size(pageSize);   System.***out***.println(searchSourceBuilder.toString());   Search search = **new** Search.Builder(searchSourceBuilder.toString()).addIndex(GmallConstant.***ES\_INDEX\_SALE\_DETAIL***).addType(**"\_doc"**).build();  Map resultMap=**new** HashMap(); *//需要总数， 明细，2个聚合的结果* **try** {  SearchResult searchResult = **jestClient**.execute(search);  *//总数* Long total = searchResult.getTotal();   *//明细* List<SearchResult.Hit<Map, Void>> hits = searchResult.getHits(Map.**class**);  List<Map> saleDetailList=**new** ArrayList<>();  **for** (SearchResult.Hit<Map, Void> hit : hits) {  saleDetailList.add(hit.**source**) ;  }  *//年龄聚合结果* Map ageMap=**new** HashMap();  List<TermsAggregation.Entry> buckets = searchResult.getAggregations().getTermsAggregation(**"groupby\_user\_age"**).getBuckets();  **for** (TermsAggregation.Entry bucket : buckets) {  ageMap.put(bucket.getKey(),bucket.getCount());  }  *//性别聚合结果* Map genderMap=**new** HashMap();  List<TermsAggregation.Entry> genderbuckets = searchResult.getAggregations().getTermsAggregation(**"groupby\_user\_gender"**).getBuckets();  **for** (TermsAggregation.Entry bucket : genderbuckets) {  genderMap.put(bucket.getKey(),bucket.getCount());  }   resultMap.put(**"total"**,total);  resultMap.put(**"list"**,saleDetailList);  resultMap.put(**"ageMap"**,ageMap);  resultMap.put(**"genderMap"**,genderMap);      } **catch** (IOException e) {  e.printStackTrace();  }  **return** resultMap; } |

### 4.5 bean

|  |
| --- |
| @Data @AllArgsConstructor **public class** Option {  String **name**;   Double **value**; } |

|  |
| --- |
| @Data @AllArgsConstructor  **public class** Stat {   String **title**;   List<Option> **options**;  } |

lombok注解说明

|  |  |
| --- | --- |
| @Data | 注解会自动增加getter 和setter方法 |
| @ AllArgsConstructor | 会自动增加包含全部属性的构造函数 |

需要pom.xml增加依赖

|  |
| --- |
| <**dependency**>  <**groupId**>org.projectlombok</**groupId**>  <**artifactId**>lombok</**artifactId**>  <**optional**>true</**optional**> </**dependency**> |

### 4.6 PublisherController

|  |
| --- |
| @GetMapping(**"sale\_detail"**) **public** String getSaleDetail(@RequestParam(**"date"**)String date ,@RequestParam(**"startpage"**) **int** startpage,@RequestParam(**"size"**) **int** size,@RequestParam(**"keyword"**)String keyword){  Map saleMap = publisherService.getSaleDetail(date, keyword, startpage, size);  Long total = (Long)saleMap.get(**"total"**);  List<Map> saleDetailList = (List)saleMap.get(**"detail"**);  Map ageMap =(Map) saleMap.get(**"ageMap"**);  Map genderMap =(Map) saleMap.get(**"genderMap"**);     *// genderMap 整理成为 OptionGroup* Long femaleCount =(Long) genderMap.get(**"F"**);  Long maleCount =(Long) genderMap.get(**"M"**);  **double** femaleRate = Math.round(femaleCount \* 1000D / total) / 10D;  **double** maleRate = Math.round(maleCount \* 1000D / total) / 10D;  List<Option> genderOptions=**new** ArrayList<>();  genderOptions.add( **new** Option(**"男"**, maleRate));  genderOptions.add( **new** Option(**"女"**, femaleRate));  OptionGroup genderOptionGroup = **new** OptionGroup(**"性别占比"**, genderOptions);  *// ageMap 整理成为 OptionGroup* Long age\_20Count=0L;  Long age20\_30Count=0L;  Long age30\_Count=0L;    **for** (Object o : ageMap.entrySet()) {  Map.Entry entry = (Map.Entry) o;  String agekey =(String) entry.getKey();  **int** age = Integer.parseInt(agekey);  Long ageCount =(Long) entry.getValue();  **if**(age <20){  age\_20Count+=ageCount;  }**else if**(age>=20&&age<30){  age20\_30Count+=ageCount;  }**else**{  age30\_Count+=ageCount;  }  }   Double age\_20rate=0D;  Double age20\_30rate=0D;  Double age30\_rate=0D;   age\_20rate = Math.round(age\_20Count \* 1000D / total) / 10D;  age20\_30rate = Math.round(age20\_30Count \* 1000D / total) / 10D;  age30\_rate = Math.round(age30\_Count \* 1000D / total) / 10D;  List<Option> ageOptions=**new** ArrayList<>();  ageOptions.add( **new** Option(**"20岁以下"**,age\_20rate));  ageOptions.add( **new** Option(**"20岁到30岁"**,age20\_30rate));  ageOptions.add( **new** Option(**"30岁以上"**,age30\_rate));  OptionGroup ageOptionGroup = **new** OptionGroup(**"年龄占比"**, ageOptions);   List<OptionGroup> optionGroupList=**new** ArrayList<>();  optionGroupList.add(genderOptionGroup);  optionGroupList.add(ageOptionGroup);   SaleInfo saleInfo = **new** SaleInfo(total, optionGroupList, saleDetailList);   **return** JSON.toJSONString(saleInfo); |

## 5对接可视化模块

