# 复习

## hadoop的shuffle过程

### mapTask先把数据放到缓冲区，达到阈值后，溢写到磁盘，溢写过程中merge

### 环形缓冲区

## spark的shuffle流程

## onyarn 的作业提交流程

## spark的任务调度流程

## hive调优

## 描述HIVE的架构以及角色功能

## 阐述hive中的sort by ，order by ，cluster by ，distrbute by的功能原理

## 描述HBASE的架构以及角色功能

## 什么是数据倾斜，以及解决方案

## zk

### 节点 2类4种

### 提供文件树和监听组件，监听一次性

## 数据库 TestDB 中有一张名为 User\_Actioninfo 的表，表的内容格式如图所示， User\_ActionInof 表存储用户行为信息，包括登录、退出、发布信息等动作。

|  |  |  |
| --- | --- | --- |
| UserId | ActionType | ActionTime |
| 23453 | Login | 1320911837 |
| 23454 | Login | 1320911838 |
| 23454 | SendMessage | 1320911845 |
| 23454 | SendMessage | 1320911987 |
| 23453 | Logout | 1320912346 |

### 注意：

#### UserId:表示用户 ID，Login:表示登录行为，Logout:表示用户登出行为，SendMessage：表示用户发消息行为，ActioinTime:表示用户行为事件。

#### 用户可以在一天内发生多次的登录、发消息和登出行为。

### Question 1:请统计 2011-11-01 日到 2011-11-24 日每天登录用户数和登录总次数，按照日期升序。请写出相应的 SQL:

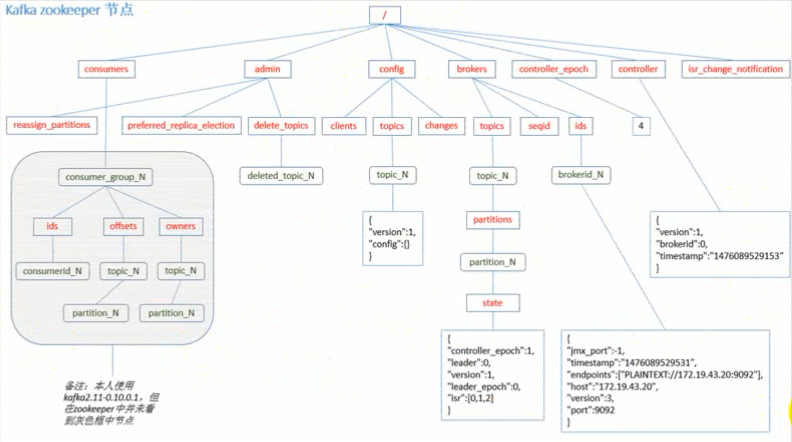
#### 每天登录总次数和每天登录用户数 PV，UV

### Question 2:现产品部门推出一个新产品方案，需要了解最近一个月内发消息的用户的频次分布情况（即一个月内，发消息 1 次的用户数量，发消息 2 次的用户数量……），按照消息次数升序排序。

|  |
| --- |
| */\*每天登录的用户数\*/* **SELECT** *COUNT*(*\**) "PV",*COUNT*(**distinct UserId**) "UV",*DATE\_FORMAT*(*FROM\_UNIXTIME*(**ActionTime**),**'%Y%m%d'**) days **FROM user\_actioninfo WHERE ActionTime** > *UNIX\_TIMESTAMP*("2011/11/01") **AND ActionTime** < *UNIX\_TIMESTAMP*("2011-11-24") **AND actionType** = "login" **GROUP BY** days **ORDER BY ActionTime**; */\*最近一个月内发消息的用户的频次分布情况（即一个月内，发消息 1 次的用户数量，发消息 2 次的用户数量……）  ，按照消息次数升序排序\*/* **SELECT** *COUNT*(*\**),**UserId from user\_actioninfo WHERE** *DATE\_SUB*(*CURDATE*(), **INTERVAL** 30 **DAY**) <= *DATE\_FORMAT*(*FROM\_UNIXTIME*(**ActionTime**),**'%Y%m%d'**) **AND ActionType** = **'SendMessage' GROUP BY UserId ORDER BY ActionTime**; |

## kafka

### kafka在zk中维护的目录结构



#### 早期版本中offset信息基于group/topic/partition管理，以group来维护zk上的offset，为什么不以consumer为单位来维护？

##### 以gourp为单位意味着只记录group的在各个分区消费的offset，而不记录具体的consumer在各分区消费的offset，假如consumer挂了，分区rebalance，其余的分区只需要去zk查group消费到了哪里，继续消费即可

#### kafka中，分区以一个目录维护，

#### consumer的分区分配

##### 轮循和range

##### rebalance

### MQ

### 基本架构与message结构

### offset

#### 在zk和本地都会存

### 高低阶消费者

#### 高阶，自动提交offset，按时间间隔，丢失和重复都有可能

### 分区分配

#### range，robin

### 高吞吐

#### 分区，分区有几个，相比不分区处理速度就提高几倍

### 数据可靠性

#### p到t。producer有3种模式，也就是acks属性，怎么才算是写入成功

##### 0只要发了就不管了，就算成功，速度快，可靠性低

##### 1leader落盘成功

##### -1leader和follower落盘成功才返回ack

#### t到c。通过使用低阶

### 与Spark Streaming

#### receiver

#### director

## flume

#### 基本构成与event结构

##### 3

##### 在组件间流转的信息，event分成2部分，header + body，跟kafka的record一样，也是kv形式。

###### header中的数据结构是kv

###### 在flume和kafka的整合中，会有根据消息的类型发送到不同topic的需求。此时flume的kafkaSink的topic不能写死，写成变量的形式%{key}。此时可以先根据body判断消息的类型，在header中设置key=topic，此时kafkaSink就可以获取到要发送到的sink。这样就实现了动态指定event的发送目的地

这一步在kafka的flume的拦截器中进行，里面可以获取到每个event。==》kafkaSink是kafka的一个producer，在topic的前面。所以可以在kafka的API中对flume的数据进行处理。

###### 给event分配分区时，也一样的道理。但此时header的kv中的key不能随意设，必须就叫key，value就是分区号。因为要把event封装为producerRecord，会自动把key作为分区的key。

##### kafka的record如果没指定key，第一条消息会随机发给一个分区，然后把这个随机的分区的id放到一个内存中的数据结构中，然后在设定的时间范围内，所有没有key的消息都会发往这个分区，时间一过，数据结构清空，重新选一个随机的分区，之后这段时间内的消息都会发往这个分区。此处跟之前学的不同，之前是以类似于轮循的方式

#### source-channel内部结构

#### 拦截器

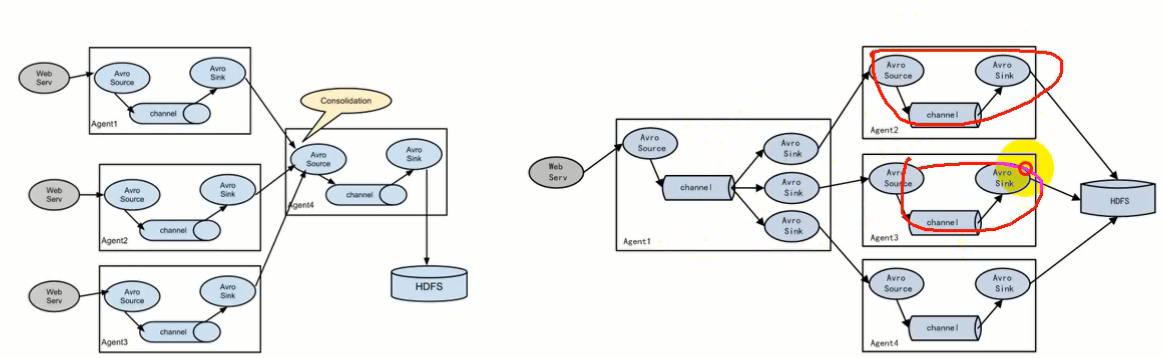
##### flume也可以有拦截器

##### 拦截器里的方法如果返回null，当前event被过滤，不建议在flume拦截器中进行数据处理，因为会导致超时

#### 配置参数的动态指定

#### channel-sink内部架构

##### 2



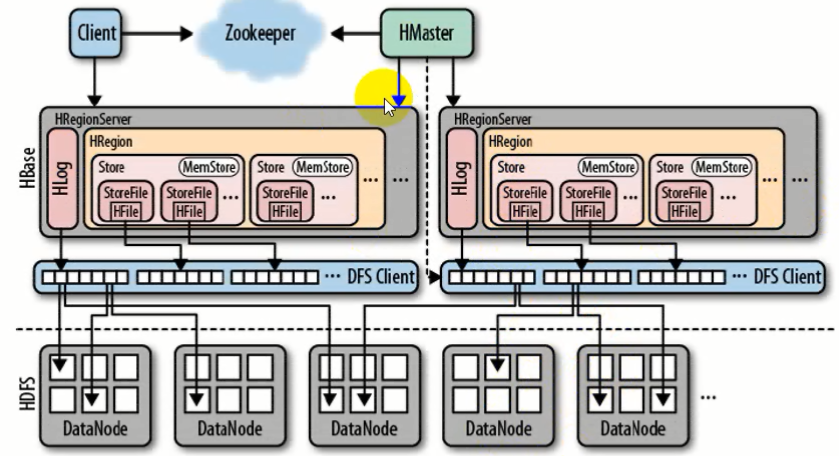
#### 与kafka的对接

#### 拓扑结构

#### 监控

## HBase

### Hbase架构



### 原理

#### HBase的存储以单元格为单位

##### 从其底层数据的记录来看，每个值、单元格，都包含rowkey+列族+列的信息==》这也就是为什么列族不要太多，列名不要太长，因为这些在每个单元格中都会存



#### 先横再纵

#### 数据存储格式

#### 读写流程

##### 读

###### 先从regionServer的meta表获取元数据

meta表存的数据啥样？

table，起始rowkey，对应region

HBase中的数据以rowkey的自然顺序正序排序，rowkey’唯一，meta表中只记录起始rowkey，因为region的行数是已知的

###### 从meta表获取就直接通过rowkey获取对应region，然后遍历

##### 写

#### 列族

##### 生产环境下列族一般就一个，列族过多小文件就多，

##### 如果列族数据量差距大，大列族要不停的分区，小列族的数据会过于分散

##### 官方文档推荐不超过2个

#### rowkey设计

##### rowkey可以几个字段拼接在一起，预分区，经常进行精确查询的字段拼前面

##### 定长，唯一，

##### 散列和预分区相结合

###### 根据分区号在rowkey中设置key，方便散列

##### 如果时间相关，一般放在散列字段后面。散列字段一般放第一位

##### 结合场景

##### HBase只能根据rowkey查==》

###### rowkey的设计很重要，需要查的字段拼到rowkey中。

###### 经常查的放前面，HBase查询的时候，rowkey的字段的先后顺序会影响查询，会先拿rowkey的前面部分去匹配，匹配不上才匹配后半部分

#### 二级索引

##### mysql使用普通字段索引主键，跨表事务

##### HBase支持行级事务

##### 协处理器

## sqoop

### 关系型数据库和HBase之间导数据

### 批量导入

### sqoop支持的比较少，DataX支持的多

## spark

# propaedutics

## 大数据平台的数据绝大多数来源于日志数据，而绝大多数的大数据框架解决的问题是日志数据的采集、清洗、存储和分析

### 日志数据不仅包括登录日志，还包括行为日志，比如用户点击了一个商品，都会被记录下来

### 埋点

#### 点击某链接则会生成相应日志信息，则这个链接称为埋点

## 数据仓库

### 数据仓库是为了给公司所有部门提供统一的数据出口，可以把所有系统的数据放到数据仓库系统

#### 建立一套完善的数据仓库系统后，所有部门的数据都可以以统一的规范存，所有部门的数据统一处理，获取的结果是综合了所有部门的数据

### 数据仓库的使用架构

#### ODS，将各部门的的数据源原样放到ODS层进行表重建，表名，列类型都不变

#### DWD，

##### 针对ODS层的数据进行ETL，去敏，归一化

##### 完成维度建模，创建维度表和实时表

#### DWS 轻度聚合，按照不同的粒度，比如日粒度、周粒度进行轻度汇总，初步处理

#### DM

##### 不同需求对应不同的主题，从DM层获取不同的数据用来进行特有分析。

### 数据仓库相关知识点



## 大数据的应用

### 产品信息分析

#### 产品信息分析用来分析产品的用户使用情况，从海量产品使用信息中得到统计信息用来指导产品的迭代和优化，判断产品发展趋势，及时做出调整

#### 比如日活，日新增，留存用户

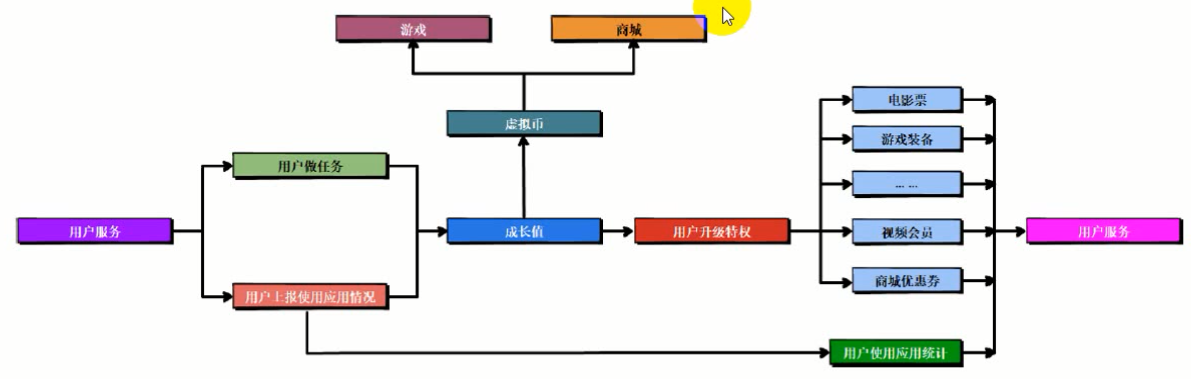


#### 原理

##### 给App植入一个sdk，会采集用户的行为日志

### 用户行为分析

#### 将用户的行为如果达到某个标准，就抽象为完成某个任务，然后根据行为给用户画像，给用户的记录设置很多标签，根据这些标签进行广告推送和商品推荐



### 人工智能基础

#### 人工智能发展了100年，最近才提速是因为大数据 + 硬件的发展

# 电商分析平台

## src

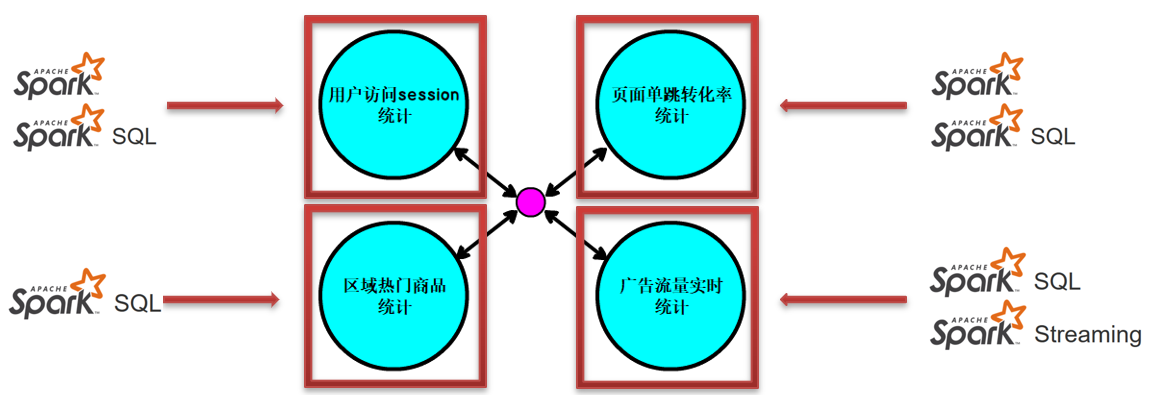
### 在访问电商网站时，我们的一些访问行为会产生相应的埋点日志（例如点击、搜索、下单、购买等），这些埋点日志会被发送给电商的后台服务器，大数据部门会根据这些埋点日志中的数据分析用户的访问行为，并得出一系列的统计指标，借助这些统计指标指导电商平台中的商品推荐、广告推送和网站优化等工作。

### 上报到服务器的埋点日志数据会经过数据采集、过滤、存储、分析、可视化这一完整流程，电商平台通过对海量用户行为数据的分析，可以对用户建立精准的用户画像，同时，对于用户行为的分析，也可以帮助电商网站找到网站的优化思路，从而在海量用户数据的基础上对网站进行改进和完善。

## 日志采集

## 需求分析

### 一共10个，分4部分

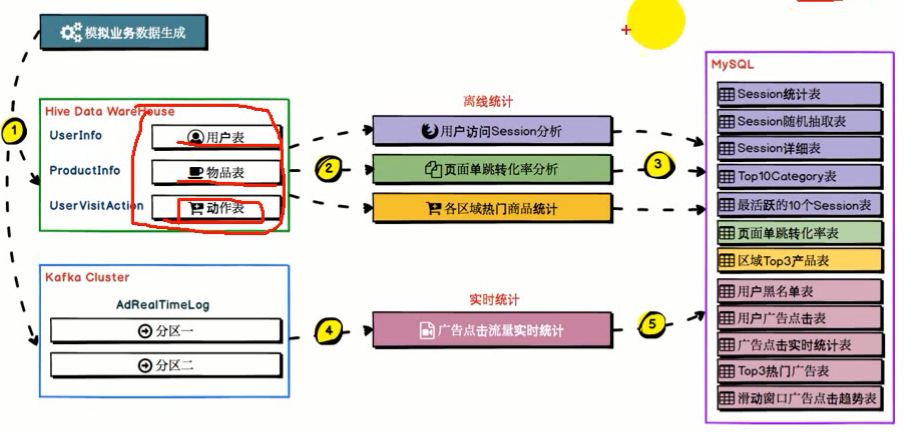


### 数据源

#### 2类，离线和实时

##### 比如日活，只能统计昨天的日活，就只能用离线数据。每天凌晨把数据导入hdfs，然后再导入hive

##### 此项目直接从hive拿离线数据



#### 离线模块的数据源分为3个表，用于前三个需求

##### 动作表



###### 列

动作分4种，点击、搜索、下单、购买，以上4种行为都会生成一条记录。

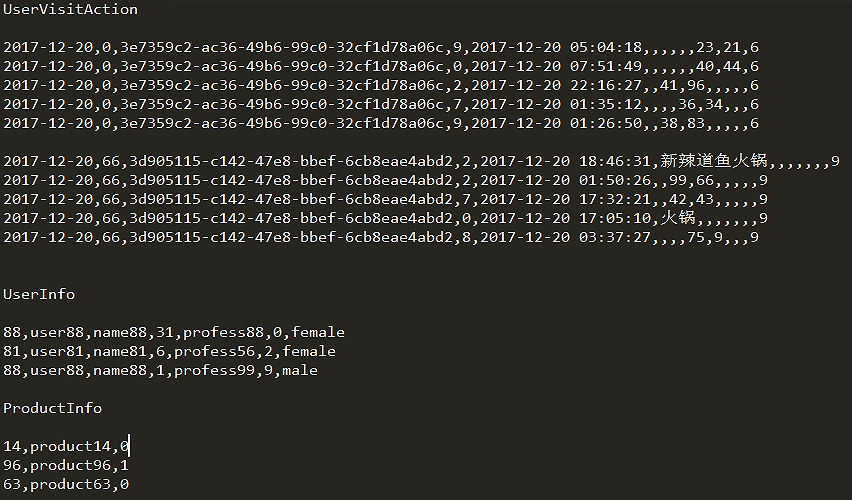
此处的session跟web中的session不同，从打开网站到离开网站，这个过程会有很多动作，一个sessionId对应动作表中的多条数据

页面ID，电商网站的页面都会有ID

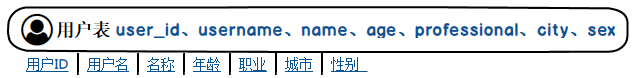
动作日期精确到天，动作时间精确到秒

查询词到付款产品ID都是跟动作类型相关

###### 数据的格式



##### 用户表



###### 可跟动作表关联，可以分析特定用户的行为

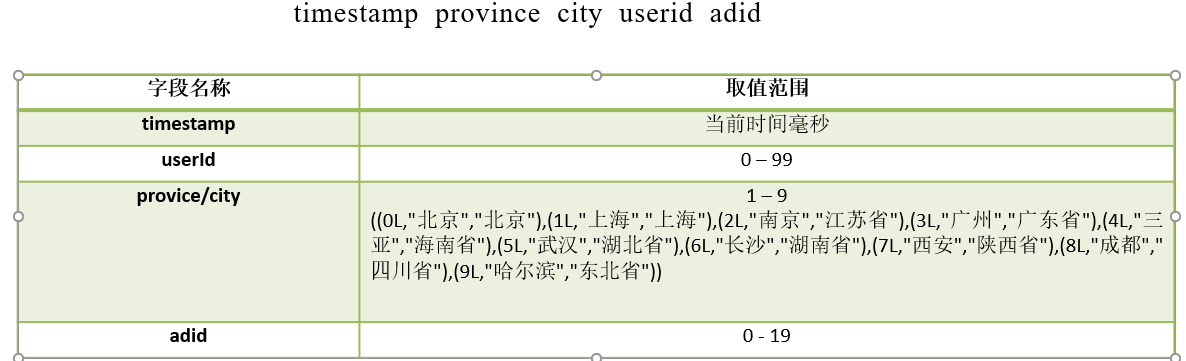
##### 产品表



#### 实时模块的数据源，

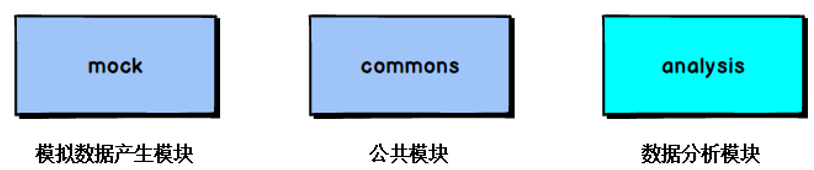
##### 用于实时统计广告点击量，用户每在埋点点击一条广告，就会生成一条kafka的消息

##### 程序每5秒向Kafka集群写入数据，格式如下



### 项目架构

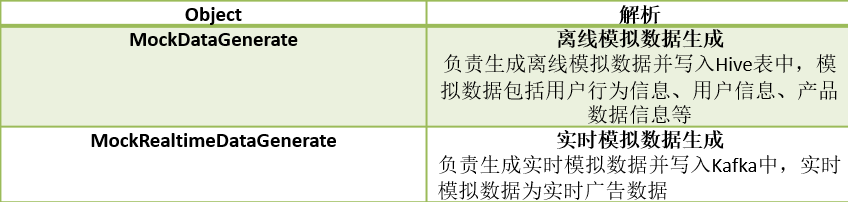
#### 图



##### 公共模块就是各种工具类

#### mock

##### 2种



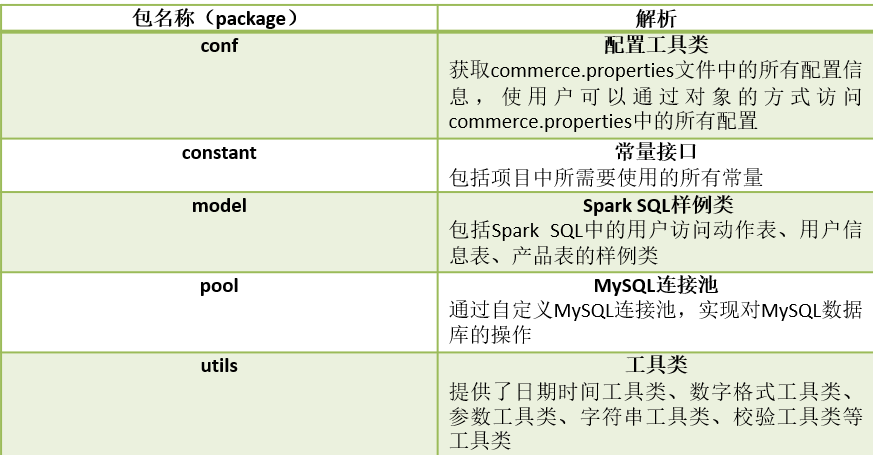
##### note

###### 一个userId对应多个sessionId，一个sessionId对应多个action记录。sessionId是不能重复的

##### principle

###### userId从100个里随机，一个userId生成10个sessionId，通过UUID保证不重复。每个用户生成0-100条action，

#### common

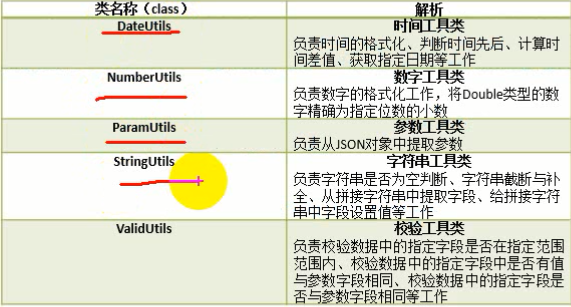


##### 读取配置文件

##### 常量类

##### 对象池。 连接池底层用的对象池，此项目种用来替代连接池

##### 工具类

。

## 项目一

### 用户访问Session

#### 用户访问session，是在电商平台的角度定义的会话概念，指的就是，从用户第一次进入首页直到最后操作完（可能做了几十次、甚至上百次操作），离开网站，关闭浏览器，或者长时间没有做操作，那么session就结束了，对应一次访问过程。以上用户在网站内一次访问的动作的集合，就称之为一次session。简单理解，session就是某一天某一个时间段内，某个用户对网站从打开/进入，到做了大量操作，到最后关闭浏览器的过程，就叫做session。

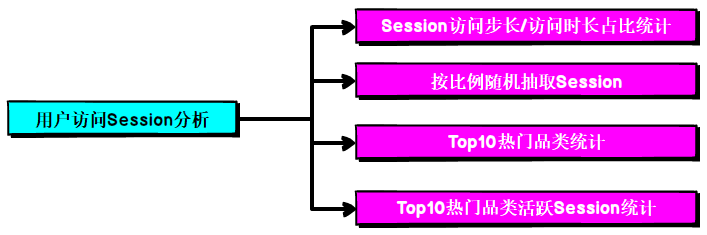
##### session实际上就是一个电商网站中最基本的数据。那么面向消费者/用户端的大数据分析，最基本的就是面向用户访问行为/用户访问session的分析。

##### 用户的每一次操作，其实可以理解为一个action，在本项目中我们关注用户的点击、搜索、下单、支付等行为。

##### 访问步长：session中action的个数

##### 访问时长：session的action最晚动作时间 - 最早动作时间

#### 分为4个子需求



##### 先把SessionId转为key，然后按SessionId groupByKey，然后计算最晚的动作时间和最早时间的差，就是session访问时长，访问步长就是每个动作的时间

##### 不可能统计所有的session，只分析有代表性的数据，比如抽取10天数据中的1000条==》每天100条==》每个小时多少条，一般中午和晚上数据密度大，会把12点和7点的数据多抽取一些

##### 按点击量二次排序

##### 按热门品类找到点击这个品类最多的session的ID

### 需求一

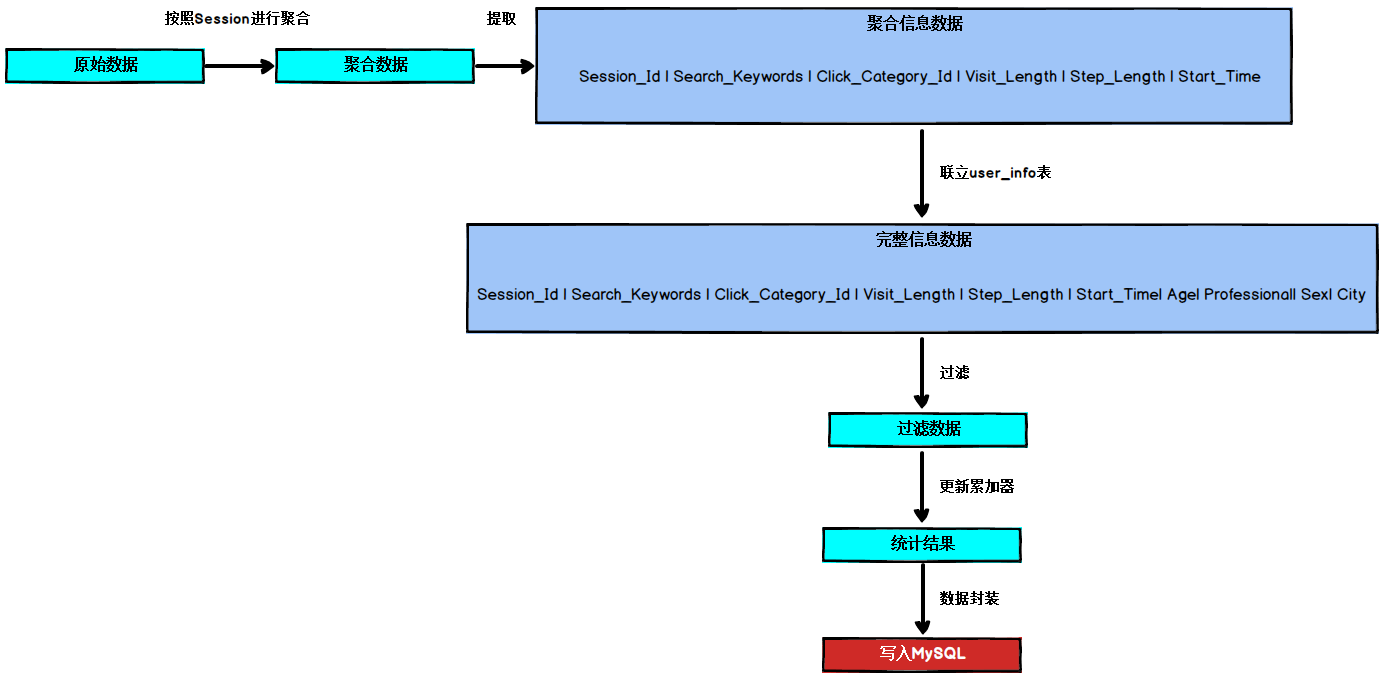
#### 具体需求

##### 统计出符合筛选条件的session中，访问时长在1s~3s、4s~6s、7s~9s、10s~30s、30s~60s、1m~3m、3m~10m、10m~30m、30m，访问步长在1\_3、4\_6、…以上各个范围内的各种session的占比。

#### 思路

##### 流程

###### 一个sessionId对应多个action，获取sessionId为key，其余字段为action字段聚合的DS，然后跟user\_info 关联，得到4个子需求的数据源，然后根据4个子需求进行进行转换和计算



###### 通过sparkSQL查出得到DF然后转为DS，或者使用get和getAs操作df获取。

转DS，implicit + as

##### note

###### 拼sql时如果少 + ，会报expected的错

###### 把DS使用rdd转成RDD后能获取到RDD[T]，相当于RDD中每个元素都是T类型

###### commerce.properties 的task.params.json的ETL时间要设置正确，因为数据是根据当前时间设置的

###### map是一对一，flatmap是多对一

###### 在函数体中接参数时，用模式匹配和 xxx =>都可以，但用模式匹配接tuple时可以设置kv的变量，避免使用\_1，\_2

###### foldLeft，一种reduce的操作

以下2组函数内部等价

 这里的acc就是other，模式匹配重新命名了变量

##### 转换过程

###### 对actionRDD进行格式转换，转为sessionId -> List(action)，

###### 然后转为userId -> 属性字符串，以userId为key是为了跟user\_info join

###### userId -> 属性字符串跟userId -> userFullInfo join，sessonId -> 带user信息的属性字符串

##### accumulator

###### extends ，重写6方法，手动设置存值的成员变量

extends AccumulatorV2[IN,OUT]

设置存值成员变量

重写6个方法

isZero 判断是否初始化

copy 复制一个当前accu的对象，返回值是对象

reset 清零

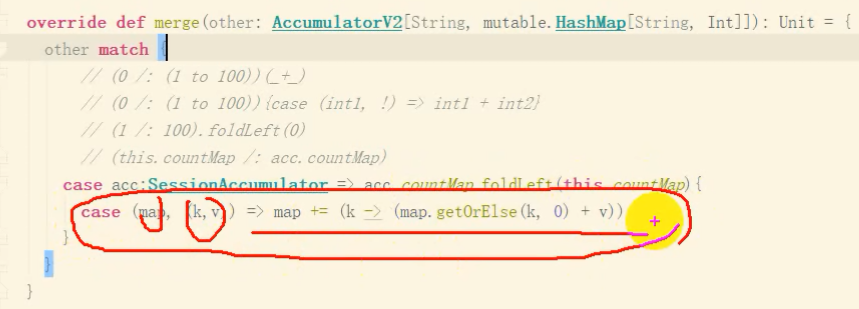
add 添加，为merge提供数据

merge 设置聚合规则

value 一般就是返回存值变量

###### merge是两两聚合，自动调用参数函数并传参

example：聚合2个map，因为最终的结果要保存在其中一个map中，而且map使用+=会导致覆盖key而不是累加，所以一个计算用累加相同key的v的方式，一个计算用+=进行覆盖更新



###### note

模式匹配可以代替强转和给tuple的kv设置变量，避免使用\_1,\_2

使用asInstanceOf[T]强转，也可以使用模式匹配代替

mutable.HashMap和util.HashMap 不同，方法也不一样

foldLeft方法2端都是map时，是拿参数map的每个tuple同调用者整体的map进行计算

merge中foldLeft中2个操作数谁调用谁是影响结果的，如果是Left，要让this的作为参数

##### 将累加器结果存到mysql中

###### 写到mysql只能转成DF

###### 算百分比时操作数要转为double，不然如果分子小于分母，永远都是0

###### 只有行动算子才能触发accumulator的执行，所以使用累加器前，即使没有意义，也要将累加器的数据源RDD执行一次行动，比如count，foreach

|  |
| --- |
| **import** sparkSession.**implicits**.\_ sessionRatioRDD.toDF().write  .format(**"jdbc"**)  .option(**"url"**, **ConfigurationManager**.*config*.getString(**Constants**.*JDBC\_URL*))  .option(**"user"**, **ConfigurationManager**.*config*.getString(**Constants**.*JDBC\_USER*))  .option(**"password"**, **ConfigurationManager**.*config*.getString(**Constants**.*JDBC\_PASSWORD*))  .option(**"dbtable"**, **"session\_stat\_ratio\_0416"**)  .mode(**SaveMode**.*Append*)  .save() |

### 需求二

#### 需求

##### 在符合过滤条件的session中，随机抽取100个session。当存在N天的数据时，100个session抽取指标在天之间平均分配，也就是说，无论数据是否均匀，每天都抽取100/N条。确定了每天抽几条后，再按小时session数占当天session总数的比例确定抽取的小时session数。

###### 随机抽取

先确定抽多少

再决定怎么抽

###### 如何得到每天的session数量？100/N

###### 每小时的session数？ 每小时的session数/当天总session数 \* 100/N

当天总session数：使用map将key从sessionId改为startTime，这种改变数据结构的操作在大数据操作中非常常见

每小时的session数：再次改变数据结构，改成以小时为key

#### 确定需求，将需求转为计算机语言，明确需要的操作数，然后通过转换数据结构获取操作数，最后得到结果

#### 化简

##### 想要将想要的数据抽取，先明确从什么数据中按什么规则抽取到什么地方

###### 从以sessionId为key，关联了userInfo的属性字符串为key的源数据中

###### 按总共100条，每天100/天数 条，每个小时按条数比例抽取。

###### 明确每个小时的条数x，就可以从每个小时总共多少条中随机生成x个不同的index

###### 然后拿[天，[小时，session集合]]和[天[小时，index的集合]]进行匹配，最终获取封装了taskUUID，sessionId，fullInfo的数据

###### 存到mysql中

df.write.formate(“jdbc”).option(4).mode(Append).save

### 需求三

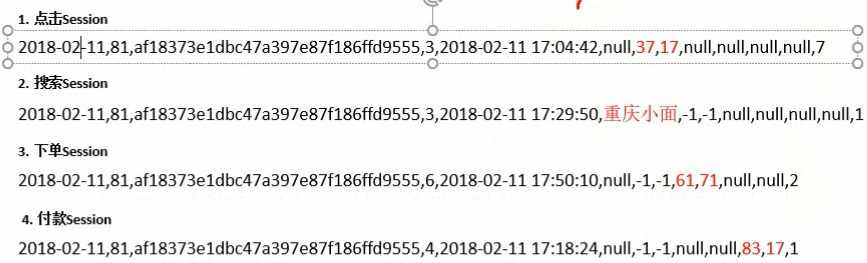
#### 需求

##### 在符合条件的用户行为数据中，获取点击、下单和支付数量排名前10的品类(不是商品)。在Top10的排序中，按照点击数量、下单数量、支付数量的次序进行排序，即优先考虑点击数量。

##### 一个简单的二次排序

#### propaedeutics

##### action表有4种行为



###### 点击，点击的品类和商品只有一个，long类型

###### 下单和付款，一次下单可以下多个商品，所以一次下单或付款行为的品类和商品都可以有多个，所以是String类型

#### 化简

##### 排序需要每个参与排序的字段都实现comparable接口(在scala中是order)或者传入一个comparator，此场景中都是int或String，已经实现

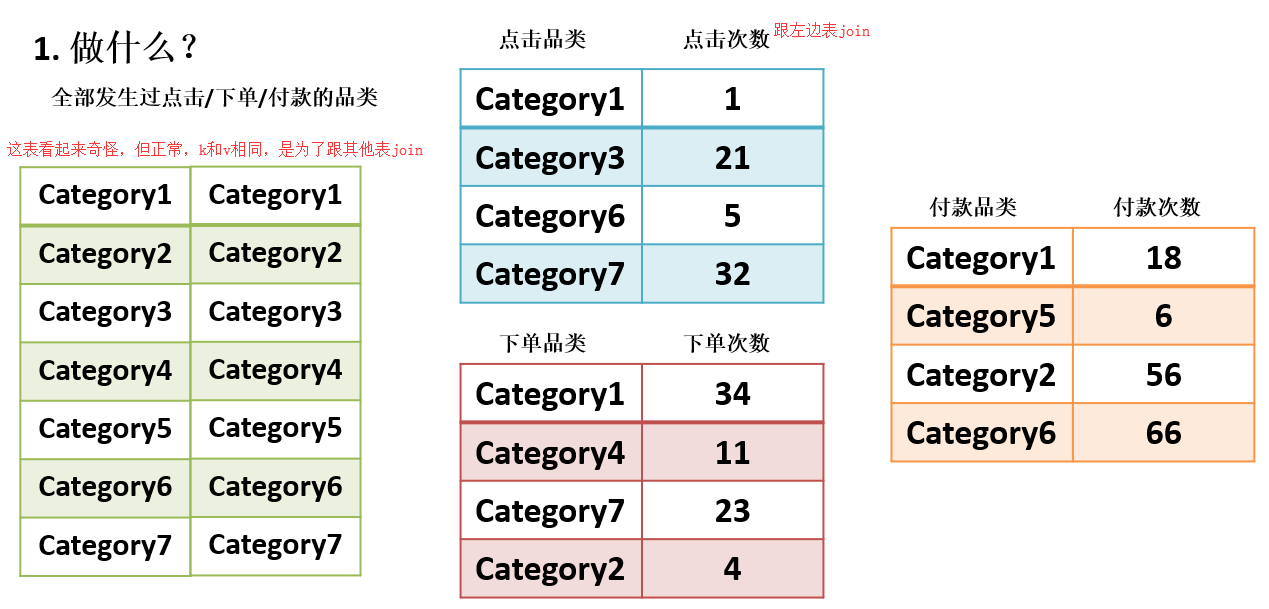
##### 排序需要根据key或者RDD中的元素是个对象或者可以把属性字符串中的信息提取出来成为变量==》需要对源数据进行转化

###### 最终获取如下的数据结构



###### 要获取上面的结构，要先获取全部的除搜索以外的sessionId -> action的列表，这里的action都是点击、下单、付款的一种，这些action对应着一个或多个品类。先获取所有发生过点击、下单、付款的categoryId，这里肯定会有重复，因为一个品类会对应多个行为，所以要distinct。

###### 然后分别获取所有有点击、下单、付款行为的categoryId和行动次数的RDD，然后以reduce累加的方式join，就是点击的跟总的join，然后把结果跟下单的join，然后结果再跟付款的join，最终就能获取上面的结果

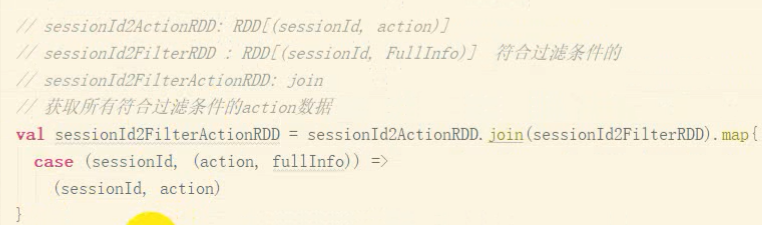


##### 此时的数据源经过之前的2个需求，有2个源数据，一个是经过过滤和随机筛选的ARDD[sessionId，属性字符串]，一个是仅仅进行了时间过滤的BRDD[sessionId,action]。想要得到经过过滤和随机筛选但元素泛型是sessionId -> action 的RDD，直接用join即可

#### 进一步化简

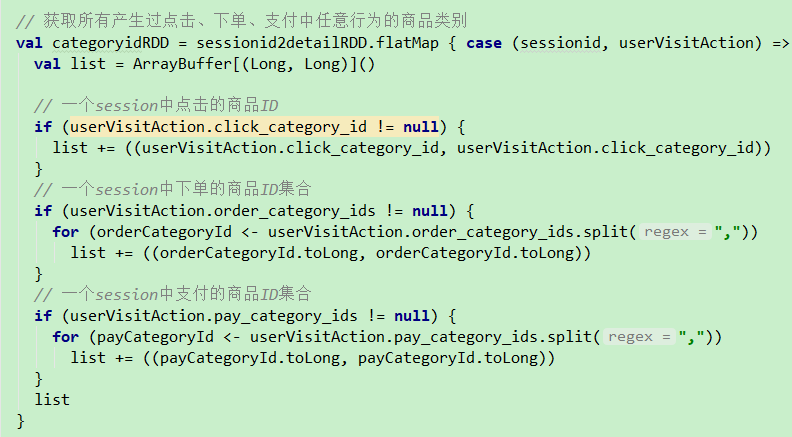
##### 上面的思路虽然正确，但顺序不合逻辑，是先考虑结果再考虑数据源，现在有了上面的思路，在开始构思代码

##### 得到经过过滤和随机筛选但元素泛型是sessionId -> action 的RDD，直接用join即可



##### 获取到sessionId -> action 的RDD之后，为了拼出3.6.2 - 2的数据结构，需要把cid变为key，value实际上没用，但只有kv才能调用join方法，所以此处随意写，写0也行

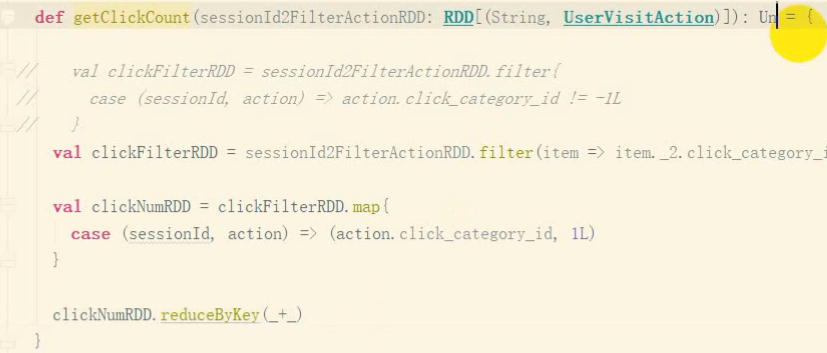
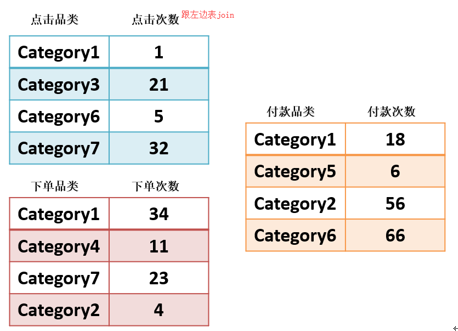
###### 下单和付款的action的品类是个字符串有多个，用逗号分割的

 ==》 

###### 此时数据里会有重复的，因为一个商品对应多个行为，有可能既点击又下单，此时要使用distinct去重，这样不会导致行为的减少，因为后join时3种行为要分别join

##### 求商品对应的点击次数

###### 先过滤出对应的行为，然后把sessionId -> acton 转成 cid -> 1，然后使用reduceByKey(\_+\_)

 ==》

##### 然后join得到



##### 现在开始排序

###### 排序的原理

将RDD的元素map成(SortKey,value)的形式。SortKey extends Order[T] ，里面重写compare方法，然后RDD调用sortByKey即可

##### 排完take

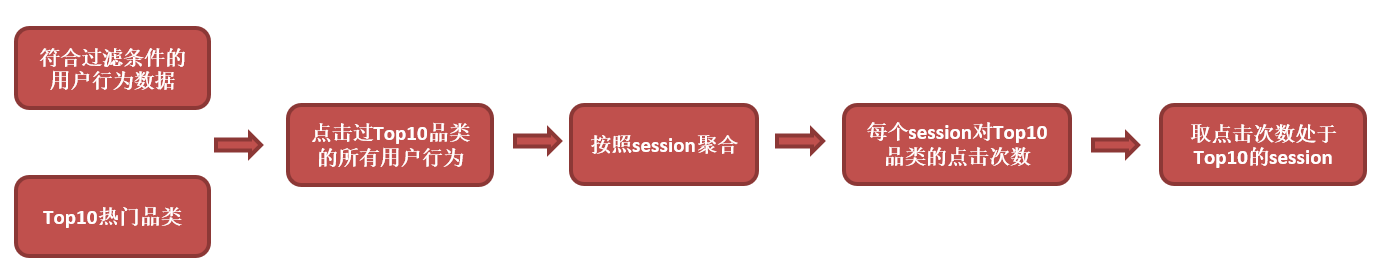
##### 最后存到mysql write.format.option.mode.save

### 需求四

#### 需求

##### 在需求三中的3种top10商品中获取点击次数最高的10个session

#### 化简



##### 所有行为的RDD filter top10cid

##### 点击过top10的所有action，按sessionId聚合，每个session对top10商品的点击次数，然后take10

## 项目二

### 需求一

#### 页面单跳转化率

##### 用户的访问行为是通过在页面间不断跳转来实现的，此需求中我们只关注跳转不关注具体的行为。用户的访问路径形成一个跳转链，通过前后2个页面切片的比值可以获取页面单跳转化率

###### 页面切片就是用户从一个页面到另一个页面的总的数量。



##### 到订单页面100万，结果支付只有10万==》找问题，解决问题

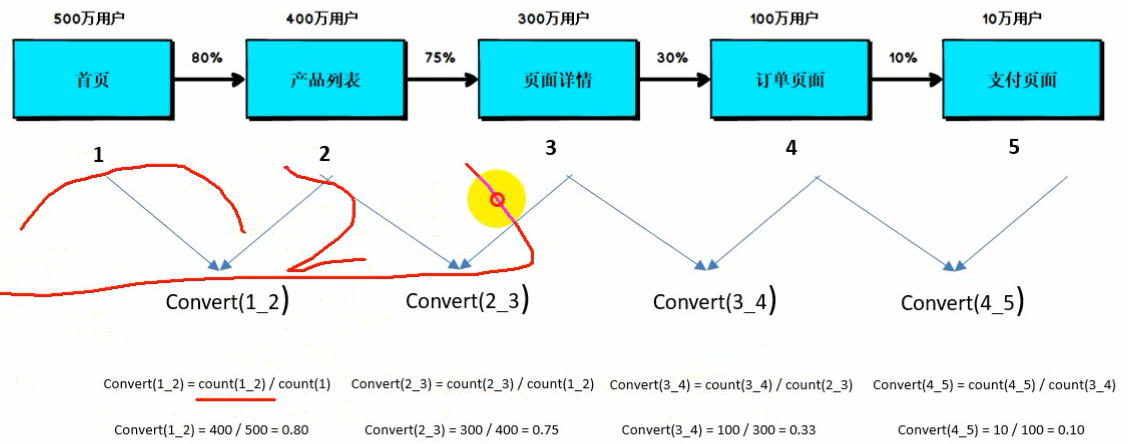
###### 产品经理，可以根据这个指标，去尝试分析整个网站/产品，各个页面的表现怎么样，是不是需要去优化产品的布局；吸引用户最终可以进入最后的支付页面。

###### 数据分析师，可以基于此数据，做更深一步的计算和分析。

###### 企业管理层，可以看到整个公司的网站，各个页面的之间的跳转的表现如何，做到心里有数，可以适当调整公司的经营战略或策略。

#### 单跳转化率，从1到2 的 / 0到1的，而不是到2的 / 到1的，转化率是依赖于2个页面的

#### 化简

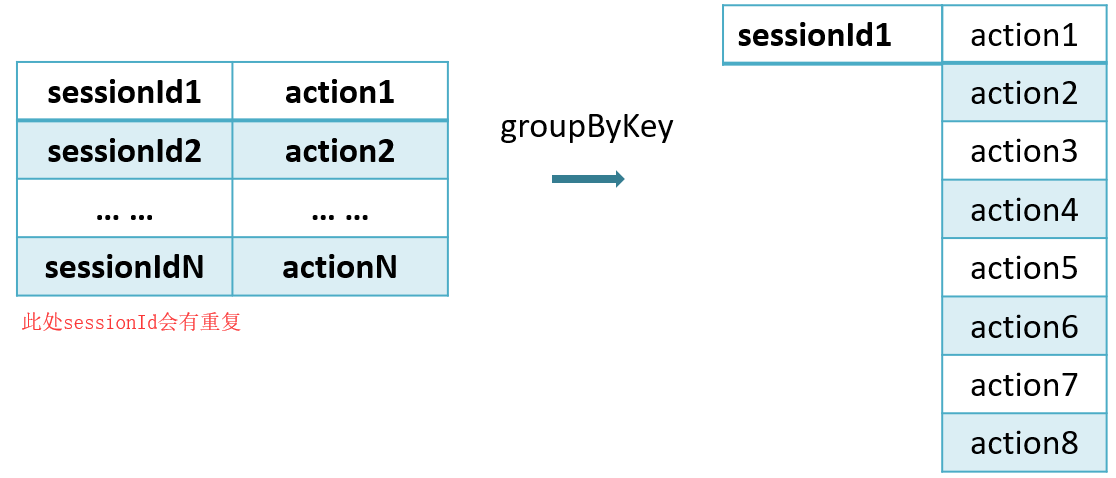


##### 求转化率，先要获取各个行为的数量，比如从首页跳到产品列表的行为的个数， action记录中有一个pageId用来存这个行为发生的页面。比如page1的action是100，page2的action是50，则转化率为50%。==》这样的思路是求pageId为1的action，再求pageId为2的action，然后求比值。==》这样不行，理解就错了，因为一个转化率是依赖于2个页面的，比如到page2的页面，不一定只有1，别的都有可能，要求1-2的，只能求从1跳到2的

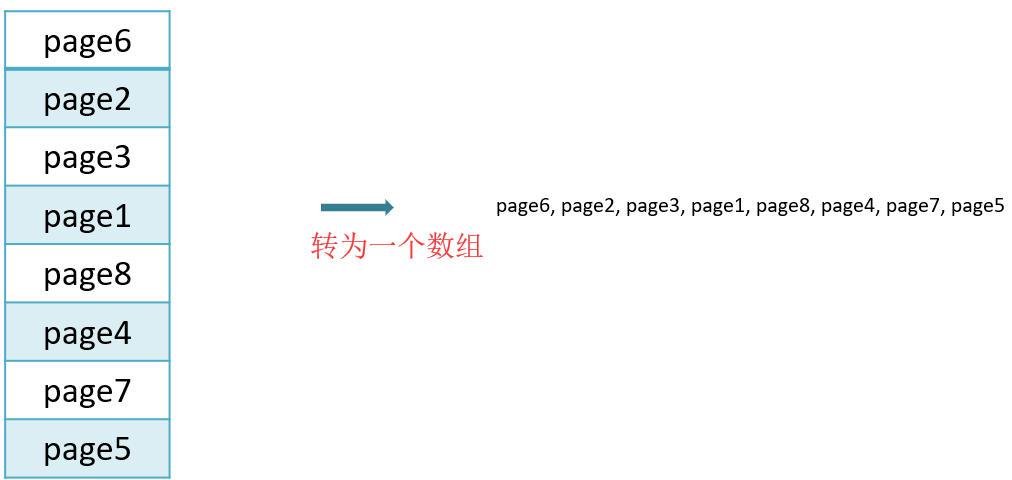
##### 求转化率，就要求一个用户的访问路径，也就是一个用户访问的一个页面和它访问的下一个页面==》只能在session范围内求==》求sessionId -> action 的RDD，然后以时间正序排序，就可以获取一个用户在一个session内的访问路径，就可以得到点击page1到page2的单跳的数量，就可以得到这个session内的单挑转化率，就能得到总的转化率

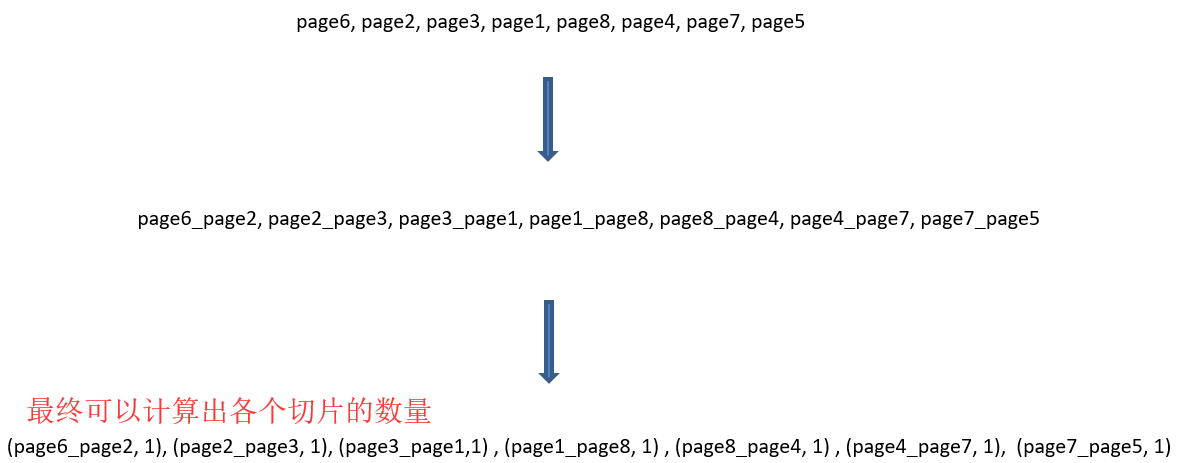
###### 一个跳转路径叫一个切片或者切面a\_b

#### 进一步化简









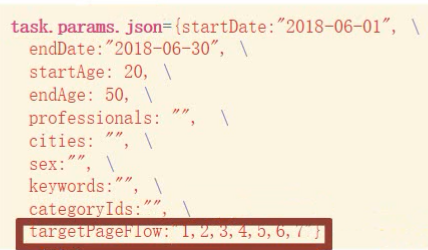


#### 代码

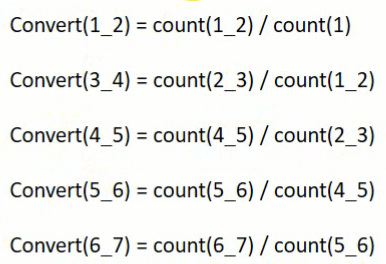
##### 化简

###### 页面访问流是由哪几个页面组成

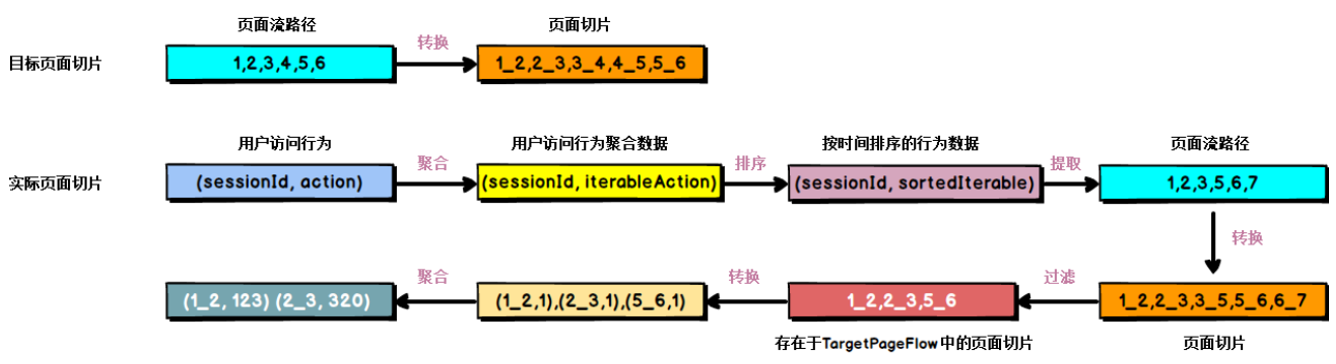
在全局过滤条件中限定了



==》 3\_5会被过滤掉



##### process



##### note

###### countByKey

## 项目三

### 需求

#### 区域热门商品统计

##### 该模块主要实现每天统计出各个区域的top3热门商品。

##### 不同地区的经济发展水平不同，地理环境及气候不同，人们的风土人情和风俗习惯不同，因此对于不同商品的需求不同，根据区域热门商品的统计，可以让公司决策层更好的对不同类型商品进行布局，使商品进入最需要他的区域。

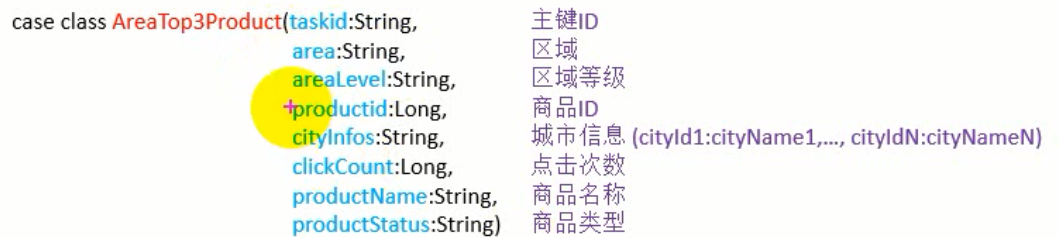
##### 该模块可以让企业管理层看到电商平台在不同区域出售的商品的整体情况，从而对公司的商品相关的战略进行调整。

#### 各区域top3商品统计，top3 通过2次排序

#### 数据源



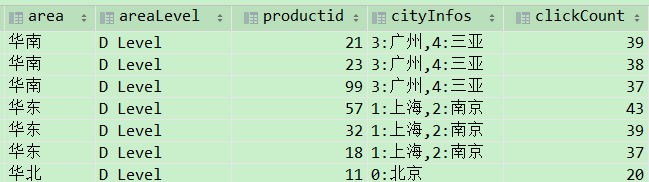
#### 存到mysql的bean



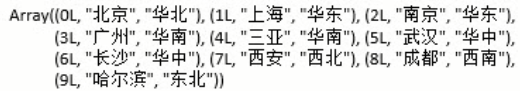
#### 区域

##### 此项目中的位置层级 级别 - Area - city，要求最后输出的时候全部都要输出

###### 以area和pid分组，用case when显示出level



##### aciton中只有cityId，这里在内存中创建一个区域的RDD，然后跟结果RDD join一下即可

###### 最后用ABCD来表示区域，要用case when来判断area

##### 从action中获取到cityId，但聚合是根据area，为了防止细粒度的city信息丢失，要自定义一个UDF函数

### 化简

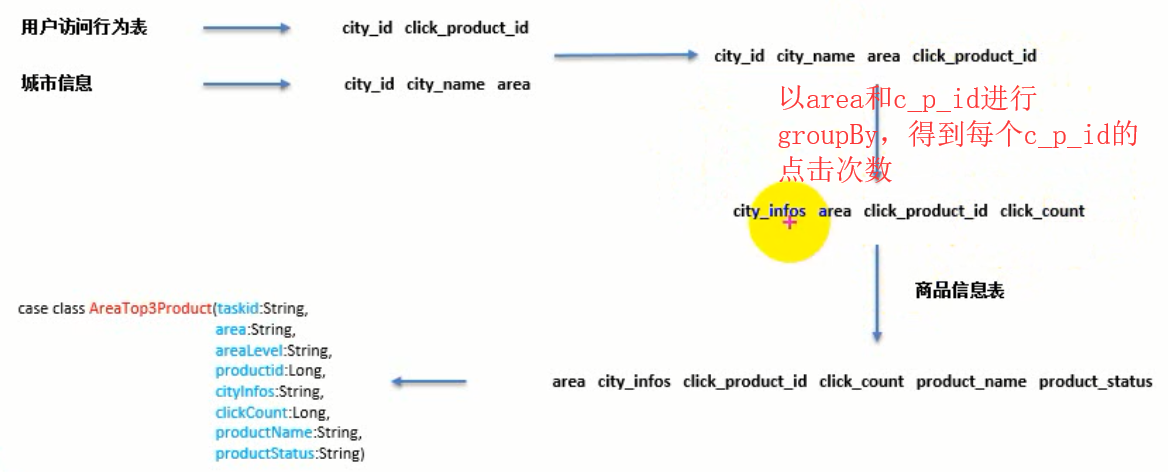
#### top3的评价标准是商品的点击次数，区域 以area和pid分组，用case when显示出level

#### 本质求的是area + pid对应的action的点击次数

##### 使用case when显示出level，使用UDF把城市信息显示出来(group by 只能写聚合函数)，使用if显示商品的状态，使用窗口函数排序

### 代码

#### process



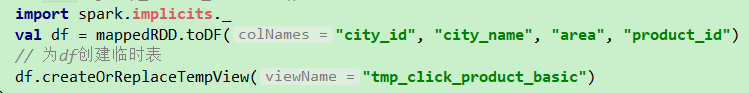
#### 创建包含cityId，pid，cityname，area的临时表，这个表一条记录对应一个action，area和pid是为了group，cityId、cityname是为了使用area排序时，不丢失细粒度的城市信息，最后要把area和city信息全都输出==》cityInfo必然冗余。

##### 先从用户点击行为表中获取c->p的RDD[cityId->clickProductId]，再获取city信息RDD[city\_id -> (city\_id,city\_name,area)]，然后join RDD[cityId -> (clickProductId,(city\_id,city\_name,area))]，使用joinRDD map为map为RDD[(cityId,city\_name,area,pid)] 然后据此创建一个临时表，这个表包含了要查询的所有信息，除了商品的详细信息，比如name。tmp\_click\_product\_basic一条数据代表一次点击行为

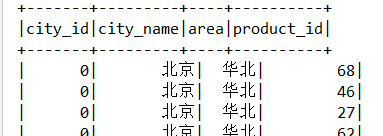
|  |
| --- |
| */\* 将cityId2PidRDD和cityid2CityInfoRDD join，转为一个临时表 RDD[cityId->clickProductId] RDD[city\_id -> (city\_id,city\_name,area)] join完RDD[cityId -> (clickProductId,(city\_id,city\_name,area))] 要map为RDD[(cityId,city\_name,area,pid)]  \*/ generateTempClickProductBasicTable*(sparkSession, cityId2PidRDD, cityid2CityInfoRDD) sparkSession.sql(**"select \* from tmp\_click\_product\_basic"**).show() |

##### note

###### 创建临时表一定要使用DF创建且要设置列名，还要隐式转换



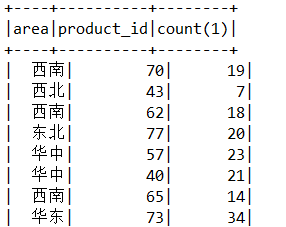
###### 查show之后，默认值显示top20



#### 此时有了点击行为的表，现在只要对area和pid进行group，然后select count(\*) 就可以得到结果，如下的sql

##### sparkSession.sql("select area,product\_id,count(\*) from tmp\_click\_product\_basic group by area,product\_id ").show()

###### 此时，就可以得到西南地区的70商品的点击数，此时排序即可



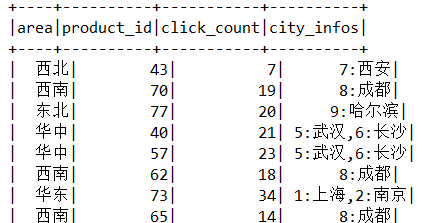
##### 但这样会丢失city信息，虽然需求上没要求，但可以使用聚合函数来在记录中保存cityId的信息，因为后面使用group by，不能直接select city\_id，city\_name

###### sparkSQL的UDAF在group by之后执行，会把当前组内的所有入参全部传入然后返回一个值，作为当前组的一列显示。UDAF是聚合函数，是可以出现在group by的sql中的，并且UDAF的入参可以是from中表的所有字段，不用必须是group by的字段

UDF在sql中的计算顺序是先执行from，列出from中的所有数据，然后join、on、where、group by、UDF。使用group by后，select后面只能查group by的列和聚合函数(也就是返回一个值的函数)，但聚合函数的入参可以是from中的列。因为每个组只有一行。

group by后产生的虚拟表并不是只剩下group by的列，所有列都在，只不过能显示出来的是有gb列和聚合的普通列

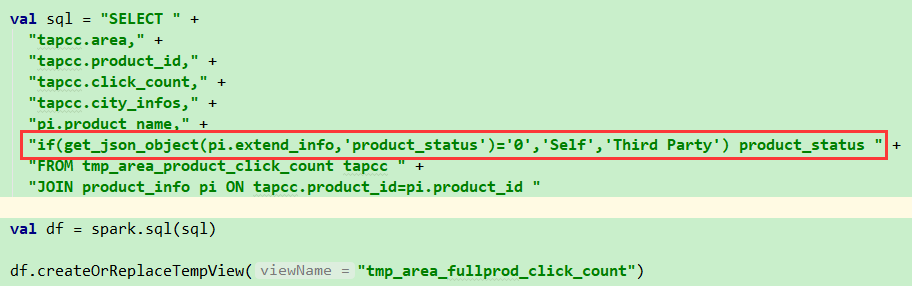
###### 使用UDF来将当前action的cityid和cityName拼接成一个字符串并且放到临时表中，临时表如下

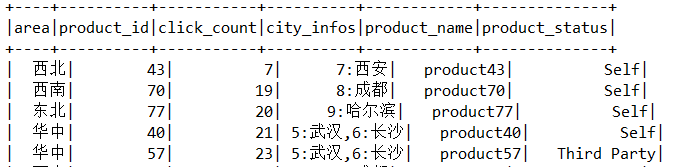


#### 然后可以跟productInfo表关联，假如product的详细信息

##### productInfo的extends\_info属性是个json串，需要写一个解析json的udf函数

##### 商品状态(自营或者第三方)显示出来是数字，需要sql中的if函数显示出对应的状态





#### 排序

##### 开窗函数

###### 同聚合函数一样，对行的集合组进行聚合计算。

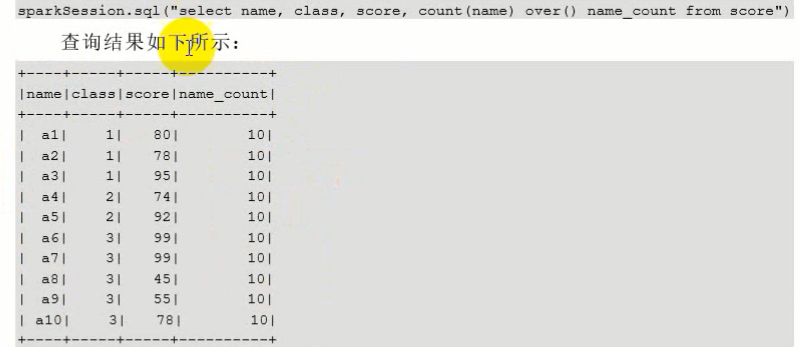
###### 用于定义一个行为窗口，窗口指的是要计算的行的集合，可以对一组值进行计算，不需要使用GB子句对数据分组，能在一行中同时返回数据列和聚合列。不像group by那样只能放在一行。

###### 分类

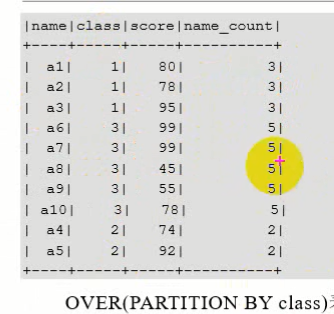


聚合

over() 括号里没有内容表示不分组，为所有的行输出一个count(\*)

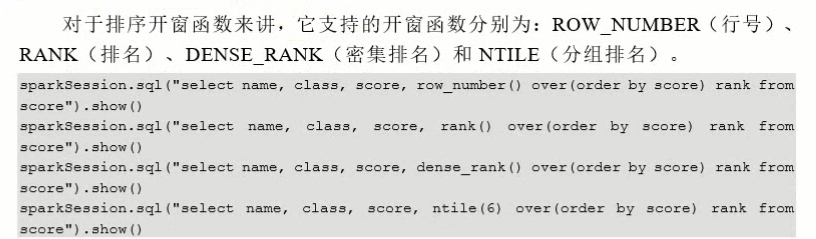


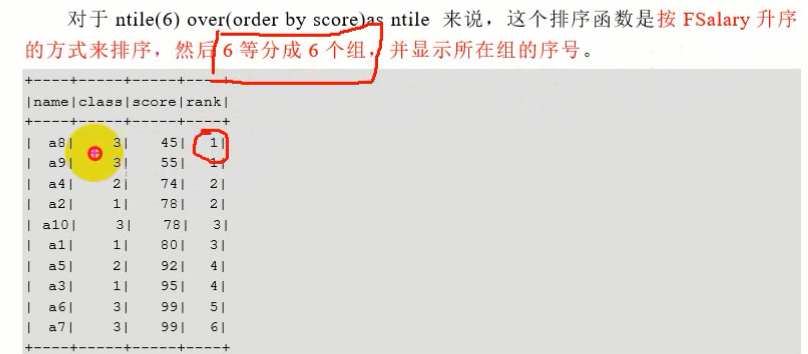
over中加入partition by表示前面的count(\*) 按by 后面的列分组



排序

R\_N相同也有次序，rank 有并列会跳跃，D\_R有并列无条约，NTILE分组排名





##### 使用开窗函数来排序，使用CASE WHEN 来将area转化为level

###### 内表使用row\_number() 来排序

###### 外表使用case when显示level

|  |
| --- |
| **val** sql = **"SELECT "** +  **"area,"** +  **"CASE "** +  **"WHEN area='China North' OR area='China East' THEN 'A Level' "** +  **"WHEN area='China South' OR area='China Middle' THEN 'B Level' "** +  **"WHEN area='West North' OR area='West South' THEN 'C Level' "** +  **"ELSE 'D Level' "** +  **"END area\_level,"** +  **"product\_id,"** +  **"city\_infos,"** +  **"click\_count,"** +  **"product\_name,"** +  **"product\_status "** +  **"FROM ("** +  **"SELECT "** +  **"area,"** +  **"product\_id,"** +  **"click\_count,"** +  **"city\_infos,"** +  **"product\_name,"** +  **"product\_status,"** +  **"row\_number() OVER (PARTITION BY area ORDER BY click\_count DESC) rank "** +  **"FROM tmp\_area\_fullprod\_click\_count "** +  **") t "** +  **"WHERE rank<=3"** |

#### 最后封装为RDD[AreaTop3Product]，然后保存到mysql

#### note

##### DF也可以使用toDF方法，用来设置列名，方便取值的使用



### summary

#### sparkSQL在进行较复杂操作时，要使用sql的情况下，要把多个RDD转成临时表，然后使用sql操作这些临时表。这样避免使用较繁琐的算子来计算。

## 项目四

### 需求



#### 4个子需求，涉及到mysql中的5张表

#### 广告流量实时统计

##### 我们要获取用户点击广告的行为，并针对这一行为进行计算和统计。用户每次点击一个广告以后，会产生相应的埋点日志；在大数据实时统计系统中，会通过某些方式将数据写入到分布式消息队列中（Kafka）。

##### 日志发送给后台web服务器（nginx），nginx将日志数据负载均衡到多个Tomcat服务器上，Tomcat服务器会不断将日志数据写入Tomcat日志文件中，写入后，就会被日志采集客户端（比如flume agent）所采集，随后写入到消息队列中（kafka），我们的实时计算程序会从消息队列中（kafka）去实时地拉取数据，然后对数据进行实时的计算和统计。

##### 这个模块的意义在于，让产品经理、高管可以实时地掌握到公司打的各种广告的投放效果。以便于后期持续地对公司的广告投放相关的战略和策略，进行调整和优化；以期望获得最好的广告收益。

### note

#### 启动kafka kafka-server-start.sh -daemon /opt/kafka\_2.11-0.11.0.2/config/server.properties &

### 数据源

#### 模拟数据，创建kafkaProducer向kafka集群发

##### kafka集群就是topic集群

### 页面代码的环境搭建

|  |
| --- |
| */\* 创建上下文，搭建页面代码环境 构建Spark上下文 创建Spark客户端 创建kafka的连接配置 创建kafka消费者配置  \*/* **val** sparkConf = **new** SparkConf().setAppName(**"streamingRecommendingSystem"**).setMaster(**"local[\*]"**) **val** spark = **SparkSession**.*builder*().config(sparkConf).getOrCreate() **val** sc = spark.sparkContext **val** ssc = **new** StreamingContext(sc, *Seconds*(5)) *// 获取Kafka配置* **val** broker\_list = **ConfigurationManager**.*config*.getString(**"kafka.broker.list"**) **val** topics = **ConfigurationManager**.*config*.getString(**"kafka.topics"**)  */\*  kafka消费者配置  创建kafka消费者需要几个配置  \*/* **val** kafkaParam = *Map*(  **"bootstrap.servers"** -> broker\_list, *//用于初始化链接到集群的地址* **"key.deserializer"** -> *classOf*[**StringDeserializer**],  **"value.deserializer"** -> *classOf*[**StringDeserializer**],  *//用于标识这个消费者属于哪个消费团体* **"group.id"** -> **"commerce-consumer-group"**,  *//如果没有初始化偏移量或者当前的偏移量不存在任何服务器上，可以使用这个配置属性  //可以使用这个配置，latest自动重置偏移量为最新的偏移量* **"auto.offset.reset"** -> **"latest"**,  *//如果是true，则这个消费者的偏移量会在后台自动提交* **"enable.auto.commit"** -> (**false**: java.lang.**Boolean**) ) |

#### 创建DStream

##### 需要传入ssc。

##### 因为Executor是消费者，需要设置分区，和消费者的分区策略

##### 还有ConsumerStrategies：选择如何在Driver和Executor上创建和配置Kafka Consumer，需要传入要订阅的topic和kafka连接参数。

##### val adRealTimeLogDStream = KafkaUtils.createDirectStream[String, String](ssc, LocationStrategies.PreferConsistent, ConsumerStrategies.Subscribe[String, String](Array(topics), kafkaParam))

#### 初步转化

##### DStream是多个RDD组成，RDD由message组成，message (k,v) v就是发过来的封装了5个属性的字符串。

##### k不需要，只需要v

#### 黑名单过滤

##### 使用AdBlacklistDAO.findAll()从mysql查出黑名单的对象的数组，AdBlacklist里就一个属性userid

##### 在filter中根据黑名单过滤

### 需求一 动态黑名单

#### 需求 动态维护黑名单

##### 比如一个广告一个用户一天点了100次，恶意访问，就需要拉黑，在实时处理获取DStream时要把这条记录filter掉，这个黑名单是动态的

##### 黑名单

###### 在mysql中维护了一张黑名单表，就一个字段，被拉黑的userid

###### 意味着每拿到一个message，都要根据黑名单进行过滤

#### 化简

##### 过滤的代码上面已经实现

##### 黑名单就是mysql中的一张记录了要过滤的的userid的表

##### 动态的内涵是实时统计单位时间内点击次数，超过100就存到黑名单表中

###### 一天内一个userid对一个广告的点击次数超过100，就把这个userid存到表中

##### 要实现如上的功能，需要统计一天内一个userid点击一个广告的点击数==》需要一张表

###### 这个表肯定不能是hive或者sparkSQL中的数据仓库

###### mysql的表可以频繁的读写吗？

并不是一个message读写一次，而是一个RDD一次

###### 因是日期\_userid\_ad，果是点击数 ==》 以日期\_userid\_ad为key，聚合后算conunt DStream没有countByKey，只能通过reduceByKey(\_+\_)

日期只需要精确到天

#### DStream的算子

##### DStream的API和里面的RDD的API不同，有一些一样，比如map，但有的只能用RDD操作，比如distinct

##### transform，将dstream中的每个batch RDD(也就是RDD)进行处理

###### 有些场景需要操作RDD，比如distinct，有些要操作里面元素，当然操作元素可以通过操作RDD进行，但直接操作比较方便，比如可以直接使用dstream调用map

##### foreachRDD，遍历每个RDD而不是RDD中每个元素。和transform的区别就是foreachRDD没返回值

###### foreach，遍历里面RDD的每个元素。

##### foreachPartition，一个循环元素是一个分区里的所有数据的集合，一般用来向DB插入创建连接对象时使用，这样避免一条一个链接对象，各个partition创建的连接对象连的是同一个mysql

#### 代码

#### summary

##### 大数据的计算中，分为2个阶段，一个是数据的转换，把数据加工成想要的样子，获取操作数。一个是计算，根据操作数计算结果。实际上也就对应了map和reduce。只不过因为加入了分布式，要考虑性能和健壮性，所以复杂度上升了。==》

###### 最重要的就是明确需求，抽象为计算的结果，然后由结果推导出操作数。

###### 化简：明确指标，设置维度，设置key，value

##### 加深了对流式处理的理解，代码，mysql表都是不断变化的。

###### 跟离线计算相比，同数据采集的工作练习更为紧密

##### DStream

###### Driver不能关，因为数据一直在流入

###### 一个采集周期一个RDD，放到DStream中流动

### 需求二

#### 需求

##### 各省各城市广告点击量实时统计

##### 这里统计的是总量，上面的需求一统计的也是总量，但实现思路完全不同，需求一是将每个RDD的总数求出来，往数据库插入的时候算的总数。而需求二是用updateStateByKey，在向数据库保存之前已经有了总量。

###### 需求二的计算状态的，每个RDD的计算都依赖于前面RDD的累加值

#### 化简

##### 需求是各省各城市每个广告点击量的总数。维度是省市+广告。一个维度对应一个action。

##### 把经过黑名单过滤的数据转成省市+广告为key，1L为v的DSt，然后updateStateByKey即可

#### updateStateByKey

##### 把每次的累加结果封装为checkpoint存到磁盘，然后每次checkpoint和新的RDD的整个value序列计算累加值。==》byKey说明要是pairRDD

##### 此需求中，省市+广告为key，1L为v。每次计算，都是每个RDD先分组，计算v的总数，然后以kv的形式存到checkpoint中，下一个RDD来时，仍然先分组key -> Seq[value]，然后从checkpoint中把上次的key -> count读进来，根据key把上次的count和Seq[value]合并

#### note

##### Option用来处理返回值，封装非空判断

##### 设置checkpoint

**val** ssc = **new** StreamingContext(sc, *Seconds*(5))  
ssc.checkpoint(**"./streaming\_checkpoint"**)

### 需求三

#### 需求

##### 每天每个省top3广告

#### 化简

##### 指标 点击量排序

##### 维度 天，省，广告

###### 可以把需求二的数据拿来用，但需求二粒度细，有市，把市从key中去掉后，会导致相同的key 有多条数据，groupBy后即可

##### 排序使用sparkSQL方便，可以implicits - toDF - createTempView，然后使用窗口函数进行排序

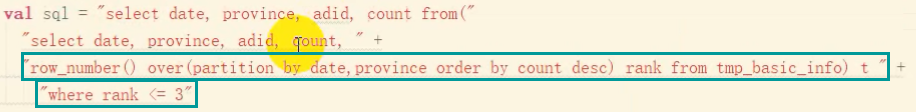
#### 窗口函数

##### syntax

###### 排序规则 over(partition by 分区(实际上就是分组)列 order by 排序列) 别名

###### 设置好排序规则后要在外层在设个查询设置排多少序

##### 此需求中，就是按日期和省分组，根据count desc排序。有重复有跳跃。



#### summary

##### 大数据的计算中，有些适合用代码，有些适合用sql，比如复杂的排序，转成临时表用窗口函数查就很方便

##### 从DF中get可以用getAs

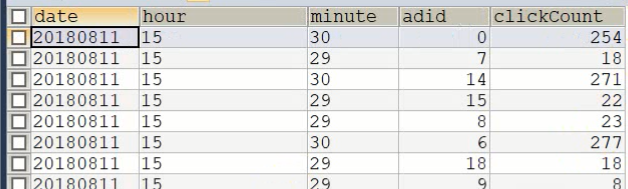
### 需求四

#### 需求

##### 最近一小时的广告点击量实时统计

##### 只需要统计最近一小时每分钟的点击数即可

###### 最后结果表



#### 化简

##### 采集周期是5秒，可使用窗口操作

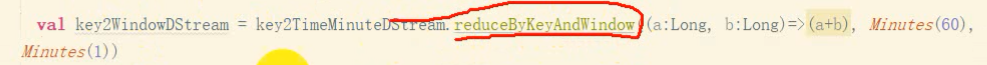
##### 最近一小时其实就是最近60分钟，粒度没必要细到秒，可以统计最近60分钟中每分钟的点击量，然后求和

##### 维度 天-小时-分钟-广告id 。 指标 点击次数

#### 窗口算子

##### 就是在正常的流中，进行窗口操作，设置参数和参数函数即可。window只不过是限制了相应操作的范围，比如对一个RDD执行reduceByKey，没有window都是对全部执行，如果设置了window(范围，步长)，就是对最近的范围内每步长执行一次reduceByKey

##### 此需求中，把key是天-小时-分钟-广告id，a和b分别是相邻2分钟的接收到的数据的value



#### note

##### scala的数据结构有时需要转成java的才能调用java的方法，比如toArray