尚硅谷大数据项目之实时数仓(日活)

(作者：尚硅谷大数据研发部)

版本：V1.0.0

# 第1章 日活需求概述

* 什么叫日活:

1. **通常**: 打开应用的用户即为活跃用户，不考虑用户的使用情况。每天一台设备打开多次会被计为一个活跃用户。 也就是只需要统计第一次打开即可
2. **游戏用户:** 每天打开(登录)游戏的用户数（针对游戏DAU的定义）

咱们项目中使用第一种日活定义

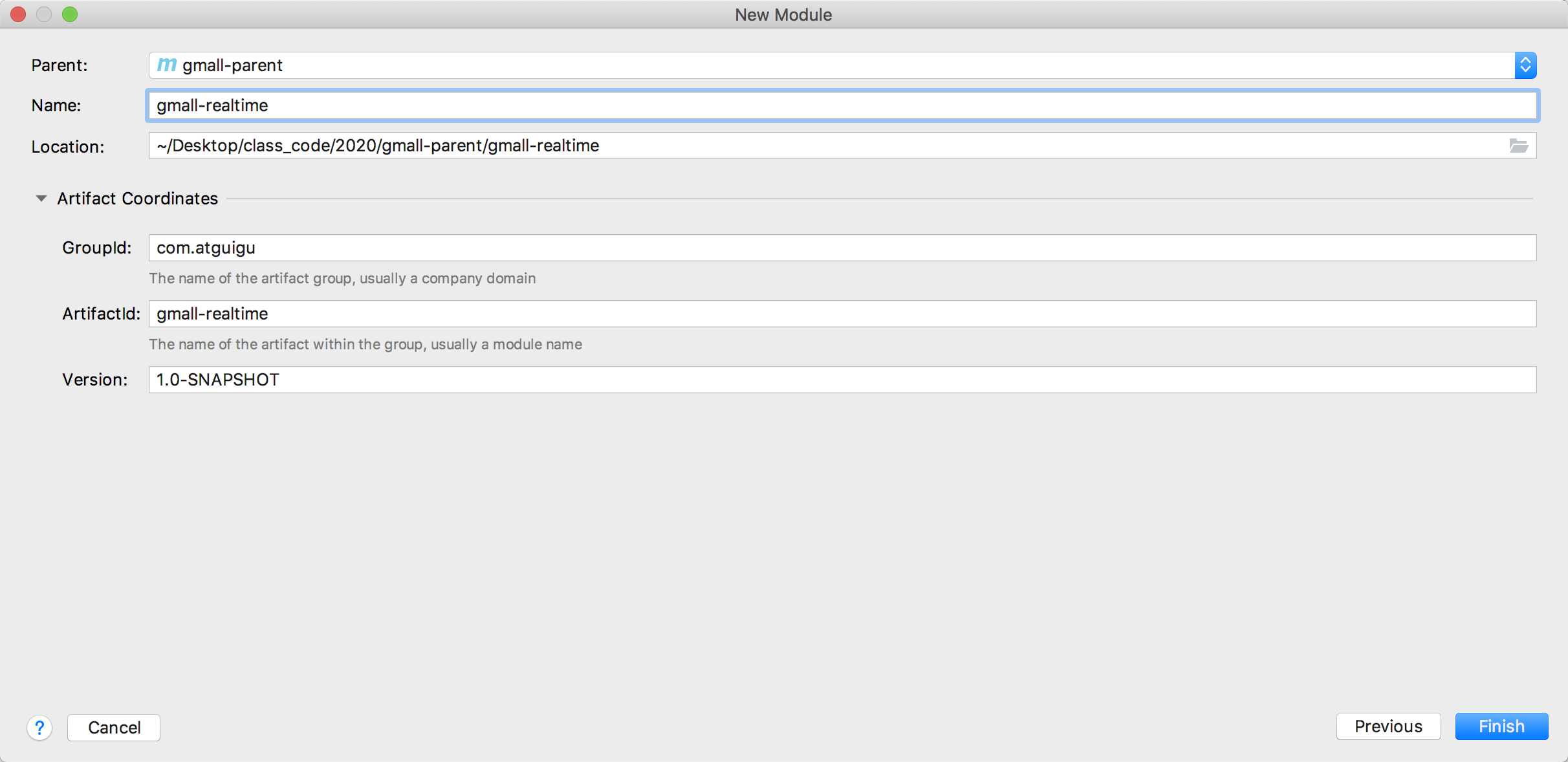
日活统计思路:

1. 从 kafka 读取用户启动日志
2. 当天只保留每个用户的第一次启动记录, 过滤掉每个用户其他启动记录: 需要借助于 Redis
3. 然后把第一次启动记录保存在es 以供其他应用查询
4. 开发web服务器, 从es读取数据, 提供前端展示.
5. 前端显示日活和当日小时日活数据



# 第2章 搭建实时处理模块

使用spark-streaming来搭建实时模块



## 在pom.xml中添加需要的依赖

*<*dependency*>*

*<*groupId*>*org.apache.spark*</*groupId*>*

*<*artifactId*>*spark-core\_2.12*</*artifactId*>*

*<*version*>*3.0.0*</*version*>*

*</*dependency*>*

*<*dependency*>*

*<*groupId*>*org.apache.spark*</*groupId*>*

*<*artifactId*>*spark-streaming\_2.12*</*artifactId*>*

*<*version*>*3.0.0*</*version*>*

*</*dependency*>*

*<*dependency*>*

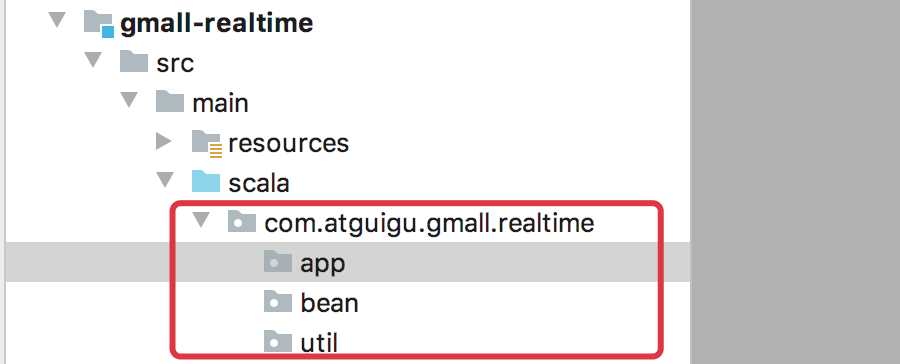
*<*groupId*>*org.apache.spark*</*groupId*>*

*<*artifactId*>*spark-streaming-kafka-0-10\_2.12*</*artifactId*>*

*<*version*>*3.0.0*</*version*>*

*</*dependency*>*

## 创建需要的package



## 添加需要的配置文件

### log4j.properties

在resource目录下添加log4j.properties文件

log4j.appender.atguigu.MyConsole=org.apache.log4j.ConsoleAppender

log4j.appender.atguigu.MyConsole.target=System.err

log4j.appender.atguigu.MyConsole.layout=org.apache.log4j.PatternLayout

log4j.appender.atguigu.MyConsole.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %10p (%c:%M) - %m%n

log4j.rootLogger=error,atguigu.MyConsole

### config.properties

在resource目录下添加config.properties文件

# kafka 配置开始

**kafka.servers**=**hadoop102:9092,hadoop103:9092,hadoop104:9092**

**kafka.group**=**bigdata**

# kafka 配置结束

# redis 配置开始

**redis.host**=**hadoop102**

**redis.port**=**6379**

# redis 配置结束

## 添加需要的工具类

在包com.atguigu.gmall.realtime.util下添加需要的工具类

### 添加读取配置工具类

**package** com.atguigu.gmall.realtime.util

**import** java.io.InputStream

**import** java.util.Properties

*/\*\**

*\* Author lzc*

*\* Date 2020/8/13 6:58 下午*

*\*/*

**object** ConfigUtil ***{***

**private val** *is*: InputStream = ClassLoader.*getSystemResourceAsStream****(*"config.properties"*)***

**private val** *properties* = **new** Properties***()***

*properties*.load***(****is****)***

**def** getProperty***(***propertyName: String***)***: String = *properties*.getProperty***(***propertyName***)***

***}***

### 添加消费kafka数据工具类

**package** com.atguigu.gmall.realtime.util

**import** org.apache.kafka.common.TopicPartition

**import** org.apache.kafka.common.serialization.StringDeserializer

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

**import** org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe

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

**import** org.apache.spark.streaming.kafka010.LocationStrategies.*PreferConsistent*

**import** scala.collection.mutable

**object** MyKafkaUtil ***{***

**val** *kafkaParams* = mutable.Map***[***String, Object***](***

**"bootstrap.servers"** -> ConfigUtil.*getProperty****(*"kafka.servers"*)***,

**"key.deserializer"** -> *classOf****[***StringDeserializer***]***,

**"value.deserializer"** -> *classOf****[***StringDeserializer***]***,

**"group.id"** -> ConfigUtil.*getProperty****(*"kafka.group"*)***,

**"auto.offset.reset"** -> **"latest"**, *// 如果有保存 offset, 则从保存位置开始消费, 没有则从latest开始消费*

**"enable.auto.commit"** -> ***(*true**: java.lang.Boolean***)***

***)***

**def** getKafkaStream***(***ssc: StreamingContext, topic: String***)*** = ***{***

KafkaUtils

.*createDirectStream****[***String, String***](***

ssc,

*PreferConsistent*,

*Subscribe****[***String, String***](****Set****(***topic***)***, *kafkaParams****)***

***)***

.map***(***\_.value***)***

***}***

***}***

### 添加获取redis客户端工具类

**package** com.atguigu.gmall.realtime.util

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

*/\*\**

*\* Author lzc*

*\* Date 2020/8/13 7:18 下午*

*\*/*

**object** RedisUtil ***{***

**val** *host* = ConfigUtil.*getProperty****(*"redis.host"*)***

**val** *port* = ConfigUtil.*getProperty****(*"redis.port"*)***.toInt

**def** getClient = ***{***

**new** Jedis***(****host*, *port****)***

***}***

***}***

## 添加需要用到的样例类StartupLog

把启动日志用样例类(StartupLog)进行封装.

启动日志字段比较丰富, 在样例类中我们只保留了我们感兴趣的字段. 可以根据需要添加其他字段到样例类中

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

**import** java.text.SimpleDateFormat

**import** java.util.Date

**case class** StartupLog***(***mid: String,

uid: String,

ar: String,

ba: String,

ch: String,

md: String,

os: String,

vc: String,

ts: Long,

**var** logDate: String = **null**, *// 年月日 2020-07-15*

**var** logHour: String = **null*) {*** *//小时 10*

**private val** *date* = **new** Date***(***ts***)***

logDate = **new** SimpleDateFormat***(*"yyyy-MM-dd"*)***.format***(****date****)***

logHour = **new** SimpleDateFormat***(*"HH"*)***.format***(****date****)***

***}***

*/\**

*{*

*"common":{*

*"ar":"440000",*

*"ba":"iPhone",*

"ch":"Appstore",

*"md":"iPhone X",*

*"mid":"mid\_26",*

*"os":"iOS 13.2.9",*

*"uid":"477",*

*"vc":"v2.1.134"*

*},*

*"start":{*

*"entry":"icon",*

*"loading\_time":1925,*

*"open\_ad\_id":6,*

*"open\_ad\_ms":8828,*

*"open\_ad\_skip\_ms":1129*

*},*

*"ts":1597319770000*

*}*

*\*/*

## 创建DauApp

在包com.atguigu.gmall.realtime.app中创建object: DauApp

### 从kakfa获取启动日志流

**package** com.atguigu.gmall.realtime.app

**import** com.atguigu.gmall.realtime.bean.StartupLog

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

**import** org.apache.spark.SparkConf

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

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

**import** org.json4s.JsonAST.***{***JObject, JValue***}***

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

*/\*\**

*\* Author lzc*

*\* Date 2020/8/13 8:18 下午*

*\*/*

**object** DauApp ***{***

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

*// 1. 创建一个StreamingContext*

**val** conf = **new** SparkConf***()***.setMaster***(*"local[2]"*)***.setAppName***(*"DauApp"*)***

**val** ssc = **new** StreamingContext***(***conf, *Seconds****(***3***))***

*// 2. 获取启动日志流*

**val** sourceStream: DStream***[***String***]*** = MyKafkaUtil.*getKafkaStream****(***ssc, **"gmall\_startup\_topic"*)***

ssc.start***()***

ssc.awaitTermination***()***

***}***

***}***

注意: 测试是否可以正确获到数据, 通过sourceStream.print来完成测试

### 封装日志数据为样例类:StartupLog

在scala代码中, 解析json最好使用scala专用的json解析工具:json4s

由于spark内部已经在使用该工具, 所以不需要额外导入, 直接使用即可

**package** com.atguigu.gmall.realtime.app

**import** com.atguigu.gmall.realtime.bean.StartupLog

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

**import** org.apache.spark.SparkConf

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

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

**import** org.json4s.JsonAST.***{***JObject, JValue***}***

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

*/\*\**

*\* Author lzc*

*\* Date 2020/8/13 8:18 下午*

*\*/*

**object** DauApp ***{***

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

***// 省略其余代码…***

*// 2.1 把json 字符串数据封装到一个样例类对象中*

**val** startupLogStream = *parseToStartupLog****(***sourceStream***)***

startupLogStream.print***()***

***// 省略其余代码…***

***}***

*/\*\**

*\* 把流中的字符串解析为样例类 StartupLog 类型*

*\** **@param** *sourceStream*

*\** **@return**

*\*/*

**private def** parseToStartupLog***(***sourceStream: DStream***[***String***])*** = ***{***

sourceStream.map***(***jsonString => ***{***

**val** j: JValue = JsonMethods.parse***(***jsonString***)***

**val** jCommon = j \ **"common"**

**val** jTs = j \ **"ts"**

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

jCommon.merge***(****JObject****(*"ts"** -> jTs***))***.extract***[***StartupLog***]***

***})***

***}***

***}***

### 使用redis对数据进行去重

把mid的值写入到redis的set中, 为方便给set设置过期时间, 每天一个set, 所以key要体现出来日期.

Key可以这么设计: mids:2020-09-01

* **待优化的去重设计**

/\*\*

\* 按照 mid, 对流的数据进行去重

\*

\* **@param** startupLogStream

\* **@return**

\*/

**def** distinct**(**startupLogStream: DStream**[**StartupLog**])** = **{**

**val** preKey = **"mids:"**

startupLogStream.filter**(**startupLog => **{**

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

**val** key = preKey + startupLog.logDate

// 把 mid 写入到 Set 中, 如果返回值为 1 表示写入成功, 返回值为 0 表示不是首次写入, 数据重复

**val** result= client.sadd**(**key, startupLog.mid**)**

client.close**()**

println**(**result**)**

println**(**result == 1**)**

result == 1 // 首次写入的保留下来, 非首次写入的过滤掉

**})**

**}**

**缺点:** 每条日志都要建立一个到redis的连接, 连接数过多, 对redis压力过大.

* **优化后的去重设计**

应该1个分区创建一个到redis的连接. 使用mapPartiions完成去重

/\*\*

\* 按照 mid, 对流的数据进行去重

\* 优化后

\* **@param** startupLogStream

\* **@return**

\*/

**def** distinct**(**startupLogStream: DStream**[**StartupLog**])** = **{**

**val** preKey = **"mids:"**

startupLogStream.mapPartitions**(**startupLogIt => **{**

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

**val** result = startupLogIt.filter**(**startupLog => **{**

**val** key = preKey + startupLog.logDate

**val** result= client.sadd**(**key, startupLog.mid**)**

client.close**()**

result == 1

**})**

client.close**()**

result

**})**

**}**

### 去重后的数据写入到es

* **创建index模板**

PUT \_template/gmall\_dau\_info\_template

{

"index\_patterns": [

"gmall\_dau\_info\*"

],

"settings": {

"number\_of\_shards": 3

},

"aliases": {

"{index}-query": {

},

"gmall\_dau\_info-query": {

}

},

"mappings": {

"\_doc": {

"properties": {

"mid": {

"type": "keyword"

},

"uid": {

"type": "keyword"

},

"ar": {

"type": "keyword"

},

"ba": {

"type": "keyword"

},

"ch": {

"type": "keyword"

},

"md": {

"type": "keyword"

},

"os": {

"type": "keyword"

},

"vc": {

"type": "keyword"

},

"ts": {

"type": "date"

},

"logDate": {

"type": "keyword"

},

"logHour": {

"type": "keyword"

}

}

}

}

}

* **创建写入es的工具类**

**package** com.atguigu.gmall.realtime.util

**import** io.searchbox.client.config.HttpClientConfig

**import** io.searchbox.client.***{***JestClient, JestClientFactory***}***

**import** io.searchbox.core.***{***Bulk, Index***}***

*/\*\**

*\* Author atguigu*

*\* Date 2020/7/20 13:58*

*\*/*

**object** EsUtil ***{***

**val** *esUrl* = **"http://hadoop102:9200"**

*// 创建es客户端工厂*

**val** *factory* = **new** JestClientFactory

**val** *conf* = **new** HttpClientConfig.Builder***(****esUrl****)***

.connTimeout***(***1000 \* 10***)***

.readTimeout***(***1000 \* 10***)***

.maxTotalConnection***(***100***)***

.multiThreaded***(*true*)***

.build***()***

*factory*.setHttpClientConfig***(****conf****)***

***// 同时插入多条数据***

**def** insertBulk***(***index: String, source: Iterator***[***Object***])*** = ***{***

**val** client: JestClient = *factory*.getObject

**val** bulkBuilder = **new** Bulk.Builder***()***

.defaultIndex***(***index***)***

.defaultType***(*"\_doc"*)***

source.foreach ***{***

**case *(***id: String, source***)*** =>

**val** builder = **new** Index.Builder***(***source***)***.id***(***id***)***

bulkBuilder.addAction***(***builder.build***())***

**case** source =>

**val** builder = **new** Index.Builder***(***source***)***

bulkBuilder.addAction***(***builder.build***())***

***}***

client.execute***(***bulkBuilder.build***())***

client.shutdownClient***()***

***}***

*// 插入单条数据*

**def** insertSingle***(***index: String, source: Object, id: String = **null*)***: Unit = ***{***

**val** client: JestClient = *factory*.getObject

**val** action: Index = **new** Index.Builder***(***source***)***

.index***(***index***)***

.`type`***(*"\_doc"*)***

.id***(***id***)***

.build***()***

client.execute***(***action***)***

client.shutdownClient***()***

***}***

***}***

* **把日活明细写入到es**

/\*\*

\* 把数据写入到 es 中

\*

\* **@param stream**

\* **@return**

\*/

**def** saveToES**(**stream: DStream**[**StartupLog**])** = **{**

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

rdd.foreachPartition**(**startupLogIt => **{**

**val** today: String = LocalDate.now.toString

// 比如使用gmall\_dau\_info\_2020-08-20 作为 index

EsUtil.insertBulk**(s"gmall\_dau\_info\_$**today**"**, startupLogIt**)**

**})**

**})**

**}**

# 第3章 精准一次消费kafka数据

## 消费kafka数据的3种语义

**精确一次消费（Exactly-once）** 是指消息一定会被处理且只会被处理一次。不多不少就一次处理。

如果达不到精确一次消费，可能会达到另外两种情况：

**至少一次消费（at least once）**，主要是保证数据不会丢失，但有可能存在数据重复问题。

**最多一次消费 （at most once）**，主要是保证数据不会重复，但有可能存在数据丢失问题。

如果同时解决了数据丢失和数据重复的问题，那么就实现了精确一次消费的语义了。

## 重复或丢失的原因

* **数据何时会丢失：**

比如实时计算任务进行计算，到数据结果存盘之前，进程崩溃，假设在进程崩溃前kafka调整了偏移量，那么kafka就会认为数据已经被处理过，即使进程重启，kafka也会从新的偏移量开始，所以之前没有保存的数据就被丢失掉了。

* **数据何时会重复：**

如果数据计算结果已经存盘了，在kafka调整偏移量之前，进程崩溃，那么kafka会认为数据没有被消费，进程重启，会重新从旧的偏移量开始，那么数据就会被2次消费，又会被存盘，数据就被存了2遍，造成数据重复。

## 如何实现精准一次消费

Spark-streaming消费kafka是实现的哪种消费语义取决于偏移量的存储方式和时机.

如果要实现从端到端(输入-输出)精准一次消费, 有3种方式可选:

* **偏移量存储在checkpoint中**

如果开启spark的checkpoint, offset则会存储在checkpoint中.

这个很容易开启, 但是有缺点:

缺点1: 输出可能重复. 如果要实现输出不重复, 则输出数据的存储系统要是幂等的.

缺点2: 如果源码发生了更改,则不能从以前的checkpoint中读取数据. 所以有可能丢失数据.

* **使用kafka自己来保存偏移量**

可以使用kafka提供的commit API把offset存储在kafka的一个特殊的topic中. 可以使用stream.asInstanceOf[CanCommitOffsets].commitAsync(offsetRanges)提交offset, 但是stream必须是从kafka直接出来的流, 不能经过任何的转换. 而且, 输出系统也必须是幂等的.

* **我们自己保存偏移量**

方案1: 自己保存偏移量+事务处理

把偏移量和数据保存在支持事务的数据库中, 让他们处于同一个事务中: 要么同时成功, 要么同时失败

限制: 但是就是数据必须都要放在某一个关系型数据库中，无法使用其他功能强大的nosql数据库。如果保存的数据量较大一个数据库节点不够，多个节点的话，还要考虑分布式事务的问题。

方案2: 自己保存偏移量+幂等处理

1. 如何保证数据不丢失?

把握提交偏移量的时机: 等数据保存成功之后, 再提交偏移量, 则数据不丢失.

1. 如何保证数据不重复

如果数据已经保存, 偏移量提交之前系统崩溃, 则重启之后数据会重复保存, 使用支持幂等的系统可以避免数据重复.

## 具体实现方案

我们采用自己保存偏移量+幂等处理的方案

1. 偏移量可以保存在 mysql, redis, zookeeper中, 追求速度保存在redis中.

使用hash来存储每个分区的offset

key value(hash)

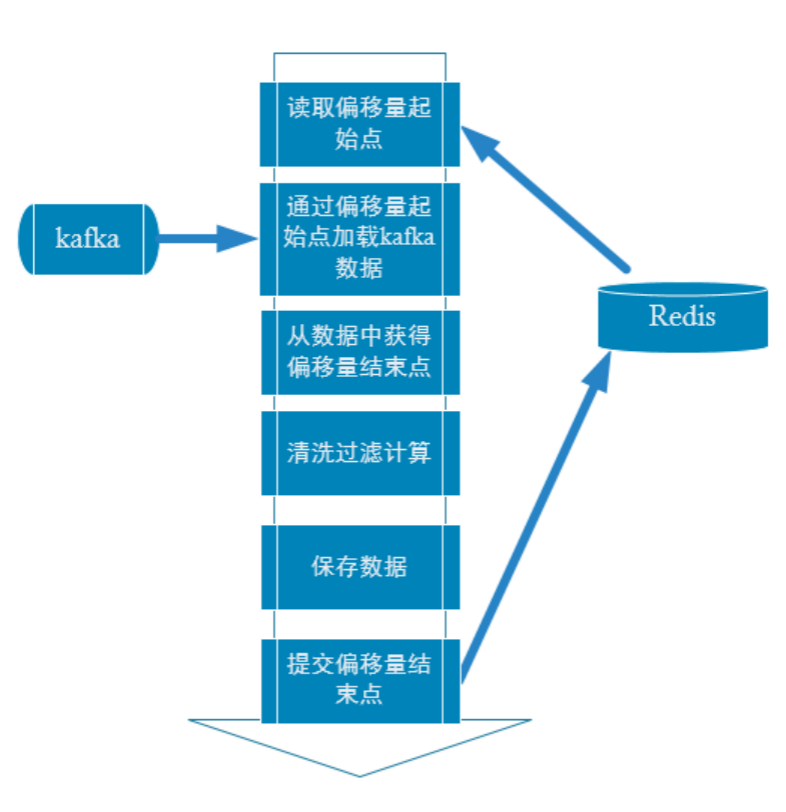
"offset:groupid:topic" field value

partiton\_id offset

partiton\_id offset

...

1. 日活明细数据保存在es中, 存储document的时候, 如果document 的id相同, 则后面的会替换前面的, 所以是支持幂等的.



### 3.4.1 改进MyKafkaUtil

因为是手动维护offset, 则创建流的时候需要指定每个分区的offset

增加新的方法

/\*\*

\* 从指定的位置开始消费 Kafka 数据

\*

\* **@param ssc**

\* **@param groupId**

\* **@param topic**

\* **@param fromOffsets**

\* **@return**

\*/

**def** getKafkaStream**(**ssc: StreamingContext, groupId: String, topic: String, fromOffsets: Map**[**TopicPartition, Long**])** = **{**

// 把 offset 自动提交设置为 false, 我们需要手动提交offset

kafkaParams**("enable.auto.commit")** = **(false**: java.lang.Boolean**)**

kafkaParams**("group.id")** = groupId

KafkaUtils

.createDirectStream**(**

ssc,

PreferConsistent,

Subscribe**[**String, String**](**Set**(**topic**)**, kafkaParams, fromOffsets**)**

**)**

**}**

### 3.4.2 增加OffsetManager工具类

在包com.atguigu.gmall.realtime.util下增下加工具类: OffsetManager

该工具实现对offset的管理: 读取和保存

* 读取:

需要在实时系统启动的时候从redis读取offset, 然后开始消费数据. 只需要读取变一次一次即可

* 保存:

每消费一个批次的数据, 成功保存到es之后, 再手动保存消费过的offset

**package** com.atguigu.gmall.realtime.util

**import** java.util

**import** org.apache.kafka.common.TopicPartition

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

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

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

*/\*\**

*\* Author lzc*

*\* Date 2020/8/16 7:22 下午*

*\*/*

**object** OffsetManager ***{***

*/\*\**

*\* 从 redis 读取 offset*

*\**

*\** **@param groupId**

*\** **@param topic**

*\** **@return**

*\*/*

**def** readOffsets***(***groupId: String, topic: String***)***: Map***[***TopicPartition, Long***]*** = ***{***

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

**val** partitionToOffsetMap: util.Map***[***String, String***]*** = client.hgetAll***(*s"offset:$*{***groupId***}*:$*{***topic***}*"*)***

client.close***()***

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

partitionToOffsetMap

.asScala

.map ***{***

**case *(***partition, offset***)*** =>

**new** TopicPartition***(***topic, partition.toInt***)*** -> offset.toLong

***}***

.toMap

***}***

*/\*\**

*\* 把 offset 存储到 redis 中*

*\**

*\** **@param offsetRanges**

*\*/*

**def** saveOffsets***(***offsetRanges: ListBuffer***[***OffsetRange***]***, groupId: String, topic: String***)***: Unit = ***{***

**if*(***offsetRanges.isEmpty***)* return**

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

*// 获取分区和该分区的偏移量*

**val** fieldToValue = offsetRanges

.map***(***offsetRange => ***{***

offsetRange.partition.toString -> offsetRange.untilOffset.toString

***})***

.toMap

.asJava

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

*println****(*"topic->offset: "** + fieldToValue***)***

client.hmset***(*s"offset:$*{***groupId***}*:$*{***topic***}*"**, fieldToValue***)***

client.close***()***

***}***

***}***

### 3.4.3 改进后的DauApp

**package** com.atguigu.gmall.realtime.app

**import** java.time.LocalDate

**import** com.atguigu.gmall.realtime.bean.StartupLog

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

**import** org.apache.spark.SparkConf

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

**import** org.apache.spark.streaming.kafka010.***{***HasOffsetRanges, OffsetRange***}***

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

**import** org.json4s.JsonAST.***{***JObject, JValue***}***

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

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

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

*/\*\**

*\* Author lzc*

*\* Date 2020/8/13 8:18 下午*

*\*/*

**object** DauApp\_1 ***{***

**val** *groupId* = **"atguigu"**

**val** *topic* = **"gmall\_startup\_topic"**

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

*// 1. 创建一个StreamingContext*

**val** conf = **new** SparkConf***()***.setMaster***(*"local[2]"*)***.setAppName***(*"DauApp"*)***

**val** ssc = **new** StreamingContext***(***conf, *Seconds****(***3***))***

*// 启动实时系统时, 读取上次停止的位置*

**val** fromOffsets = OffsetManager.*readOffsets****(****groupId*, *topic****)***

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

*// 需要保存的 offset. 保存的时候需要每次从这里读取数据, 需要使用可变集合,不能使用不可变集合*

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

**val** sourceStream = MyKafkaUtil

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

.transform***(***rdd => ***{*** *// 此 transform 只获取这次的 offset 的位置.注意: 只能在此处获取*

*// 先清空存储 offsetRanges 的集合*

offsetRanges.clear

*// 强转 rdd为OffsetRange, 然后获取 offsetRanges*

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

*// 再把新读到的 offset 储存进去*

offsetRanges ++= newOffsetRanges

rdd

***})***

**val** startupLogStream = *parseToStartupLog****(***sourceStream.map***(***\_.value***()))***

**val** filteredStartupLogStream = *distinct****(***startupLogStream***)***

filteredStartupLogStream.cache***()***

*saveToES****(***filteredStartupLogStream, offsetRanges***)***

ssc.start***()***

ssc.awaitTermination***()***

***}***

*/\*\**

*\* 把流中的字符串解析为样例类 StartupLog 类型*

*\**

*\** **@param sourceStream**

*\** **@return**

*\*/*

**private def** parseToStartupLog***(***sourceStream: DStream***[***String***])***: DStream***[***StartupLog***]*** = ***{***

sourceStream.map***(***jsonString => ***{***

**val** j: JValue = JsonMethods.parse***(***jsonString***)***

**val** jCommon = j \ **"common"**

**val** jTs = j \ **"ts"**

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

jCommon.merge***(****JObject****(*"ts"** -> jTs***))***.extract***[***StartupLog***]***

***})***

***}***

*/\*\**

*\* 按照 mid, 对流的数据进行去重*

*\* 优化后*

*\**

*\** **@param startupLogStream**

*\** **@return**

*\*/*

**def** distinct***(***startupLogStream: DStream***[***StartupLog***])*** = ***{***

**val** preKey = **"mids:"**

startupLogStream.mapPartitions***(***startupLogIt => ***{***

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

**val** result = startupLogIt.filter***(***startupLog => ***{***

**val** key = preKey + startupLog.logDate

**val** result = client.sadd***(***key, startupLog.mid***)***

client.close***()***

result == 1

***})***

client.close***()***

result

***})***

***}***

*/\*\**

*\* 把数据写入到 es 中*

*\* 并把偏移量写入到 redis 中*

*\**

*\** **@param stream**

*\** **@param offsetRanges**

*\*/*

**def** saveToES***(***stream: DStream***[***StartupLog***]***, offsetRanges: ListBuffer***[***OffsetRange***])*** = ***{***

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

rdd.foreachPartition***(***startupLogIt => ***{***

**val** today: String = LocalDate.*now*.toString

*// 使用gmall\_dau\_info\_2020-08-20 作为 index, 虽然前面已经去重了, 为了体现 es 的幂等性,我们使用 mid 作为 document 的 id 来避免重复*

EsUtil.*insertBulk****(*s"gmall\_dau\_info\_$**today**"**, startupLogIt.map***(***log => log.mid -> log***))***

***})***

*// 数据保存到 es 之后, 保存 offsets.*

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

***})***

***}***

***}***

# 第4章 利用kibana快速搭建数据可视化

如果数据保存在Elasticsearch那么利用kibana进行可视化展示是一种非常高效的可视化解决方案。

这种kibana可视化方案，优势是快速高效。但是对于展示效果的定制化和炫酷程度不能有太高的要求。

# 第5章 日活统计数据查询接口

