尚硅谷电商数仓项目--实时计算

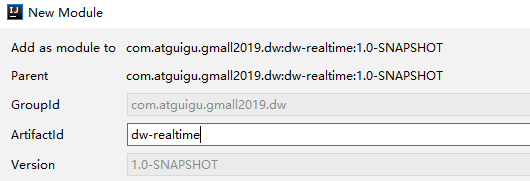
版本：V 1.6

张晨

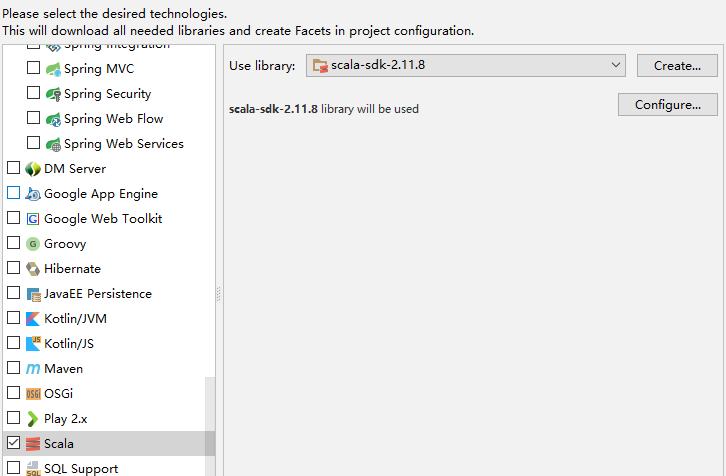
# 实时处理模块



## 1 模块搭建



添加scala框架



## 2 代码思路

* 消费kafka中的数据。
* 利用redis过滤当日已经计入的日活设备。
* 把每批次新增的当日日活信息保存到ES中。
* 从ES中查询出数据，发布成数据接口，通可视化化工程调用。

## 3 代码开发之消费Kafka

### 3.1 配置

#### 3.1.1 config.properties

|  |
| --- |
| *# Kafka配置* kafka.broker.list=hadoop1:9092,hadoop2:9092,hadoop3:9092 *# Redis配置* redis.host=hadoop1 redis.port=6379 |

#### 3.1.2 pom.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>com.atguigu.gmall2020</groupId>  <artifactId>gmall2020-realtime</artifactId>  <version>1.0-SNAPSHOT</version>   <properties>  <spark.version>2.4.0</spark.version>  <scala.version>2.11.8</scala.version>  <kafka.version>1.0.0</kafka.version>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <java.version>1.8</java.version>  </properties>   <dependencies>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>1.2.56</version>  </dependency>     <dependency>  <groupId>org.apache.spark</groupId>  <artifactId>spark-core\_2.11</artifactId>  <version>${spark.version}</version>  </dependency>  <dependency>  <groupId>org.apache.spark</groupId>  <artifactId>spark-streaming\_2.11</artifactId>  <version>${spark.version}</version>  </dependency>  <dependency>  <groupId>org.apache.kafka</groupId>  <artifactId>kafka-clients</artifactId>  <version>${kafka.version}</version>   </dependency>  <dependency>  <groupId>org.apache.spark</groupId>  <artifactId>spark-streaming-kafka-0-10\_2.11</artifactId>  <version>${spark.version}</version>  </dependency>   <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  <version>2.9.0</version>  </dependency>    <dependency>  <groupId>org.apache.phoenix</groupId>  <artifactId>phoenix-spark</artifactId>  <version>4.14.2-HBase-1.3</version>  </dependency>   <dependency>  <groupId>org.apache.spark</groupId>  <artifactId>spark-sql\_2.11</artifactId>  <version>${spark.version}</version>  </dependency>   <dependency>  <groupId>io.searchbox</groupId>  <artifactId>jest</artifactId>  <version>5.3.3</version>  <exclusions>  <exclusion>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-api</artifactId>  </exclusion>  </exclusions>  </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>   </dependencies>   <build>  <plugins>  *<!-- 该插件用于将Scala代码编译成class文件 -->* <plugin>  <groupId>net.alchim31.maven</groupId>  <artifactId>scala-maven-plugin</artifactId>  <version>3.4.6</version>  <executions>  <execution>  *<!-- 声明绑定到maven的compile阶段 -->* <goals>  <goal>compile</goal>  <goal>testCompile</goal>  </goals>  </execution>  </executions>  </plugin>   <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-assembly-plugin</artifactId>  <version>3.0.0</version>  <configuration>  <descriptorRefs>  <descriptorRef>jar-with-dependencies</descriptorRef>  </descriptorRefs>  </configuration>  <executions>  <execution>  <id>make-assembly</id>  <phase>package</phase>  <goals>  <goal>single</goal>  </goals>  </execution>  </executions>  </plugin>  </plugins>  </build> </project> |

### 3.2 工具类

#### 3.2.1 MykafkaUtil

|  |
| --- |
| **import** java.util.Properties  **import** org.apache.kafka.clients.consumer.ConsumerRecord **import** org.apache.kafka.common.TopicPartition **import** org.apache.kafka.common.serialization.StringDeserializer **import** org.apache.spark.streaming.StreamingContext **import** org.apache.spark.streaming.dstream.InputDStream **import** org.apache.spark.streaming.kafka010.{ConsumerStrategies, KafkaUtils, LocationStrategies}  **object** MyKafkaUtil {  **private val** *properties*: Properties = PropertiesUtil.*load*(**"config.properties"**)  **val** *broker\_list* = *properties*.getProperty(**"kafka.broker.list"**)   *// kafka消费者配置* **var** *kafkaParam* = collection.mutable.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"** -> **"gmall\_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,但是kafka宕机容易丢失数据  //如果是false，会需要手动维护kafka偏移量* **"enable.auto.commit"** -> (**false**: java.lang.Boolean)  )   *// 创建DStream，返回接收到的输入数据  // LocationStrategies：根据给定的主题和集群地址创建consumer  // LocationStrategies.PreferConsistent：持续的在所有Executor之间分配分区  // ConsumerStrategies：选择如何在Driver和Executor上创建和配置Kafka Consumer  // ConsumerStrategies.Subscribe：订阅一系列主题* **def** getKafkaStream(topic: String,ssc:StreamingContext ): InputDStream[ConsumerRecord[String,String]]={  **val** dStream = KafkaUtils.*createDirectStream*[String,String](ssc, LocationStrategies.*PreferConsistent*,ConsumerStrategies.*Subscribe*[String,String](*Array*(topic),*kafkaParam* ))  dStream  }    **def** getKafkaStream(topic: String,ssc:StreamingContext,groupId:String): InputDStream[ConsumerRecord[String,String]]={  *kafkaParam*(**"group.id"**)=groupId  **val** dStream = KafkaUtils.*createDirectStream*[String,String](ssc, LocationStrategies.*PreferConsistent*,ConsumerStrategies.*Subscribe*[String,String](*Array*(topic),*kafkaParam* ))  dStream  }   **def** getKafkaStream(topic: String,ssc:StreamingContext,offsets:Map[TopicPartition,Long],groupId:String): InputDStream[ConsumerRecord[String,String]]={  *kafkaParam*(**"group.id"**)=groupId  **val** dStream = KafkaUtils.*createDirectStream*[String,String](ssc, LocationStrategies.*PreferConsistent*,ConsumerStrategies.*Subscribe*[String,String](*Array*(topic),*kafkaParam*,offsets))  dStream  } } |

#### 3.2.2 PropertiesUtil

|  |
| --- |
| **object** PropertiesUtil {   **def** main(args: Array[String]): Unit = {  **val** properties: Properties = PropertiesUtil.*load*(**"config.properties"**)   *println*(properties.getProperty(**"kafka.broker.list"**))  }   **def** load(propertieName:String): Properties ={  **val** prop=**new** Properties();  prop.load(**new** InputStreamReader(Thread.*currentThread*().getContextClassLoader.getResourceAsStream(propertieName) , **"UTF-8"**))  prop  }  } |

#### 3.3.3 RedisUtil

|  |
| --- |
| **object** RedisUtil {   **var** *jedisPool*:JedisPool=**null   def** getJedisClient: Jedis = {  **if**(*jedisPool*==**null**){ *// println("开辟一个连接池")* **val** config = PropertiesUtil.*load*(**"config.properties"**)  **val** host = config.getProperty(**"redis.host"**)  **val** port = config.getProperty(**"redis.port"**)   **val** jedisPoolConfig = **new** JedisPoolConfig()  jedisPoolConfig.setMaxTotal(100) *//最大连接数* jedisPoolConfig.setMaxIdle(20) *//最大空闲* jedisPoolConfig.setMinIdle(20) *//最小空闲* jedisPoolConfig.setBlockWhenExhausted(**true**) *//忙碌时是否等待* jedisPoolConfig.setMaxWaitMillis(500)*//忙碌时等待时长 毫秒* jedisPoolConfig.setTestOnBorrow(**true**) *//每次获得连接的进行测试   jedisPool*=**new** JedisPool(jedisPoolConfig,host,port.toInt)  } *// println(s"jedisPool.getNumActive = ${jedisPool.getNumActive}")  // println("获得一个连接")  jedisPool*.getResource  } } |

log4j.properties

|  |
| --- |
| **log4j.appender.atguigu.MyConsole**=**org.apache.log4j.ConsoleAppender log4j.appender.atguigu.MyConsole.target**=**System.out 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** |

### 3.3 业务类—消费kafka

|  |
| --- |
| **def** main(args: Array[String]): Unit = {  **val** sparkConf: SparkConf = **new** SparkConf().setMaster(**"local[\*]"**).setAppName(**"dau\_app"**)  **val** ssc = **new** StreamingContext(sparkConf, *Seconds*(5))  **val** groupId = **"GMALL\_DAU\_CONSUMER"  val** topic = **"GMALL\_START"  val** startupInputDstream: InputDStream[ConsumerRecord[String, String]] = MyKafkaUtil.*getKafkaStream*(topic, ssc)    **val** startLogInfoDStream: DStream[JSONObject] = startupInputDstream.map { record =>  **val** startupJson: String = record.value()  **val** startupJSONObj: JSONObject = JSON.*parseObject*(startupJson)  **val** ts: lang.Long = startupJSONObj.getLong(**"ts"**)  startupJSONObj  }  startLogInfoDStream.print(100)  ssc.start() ssc.awaitTermination()  }  } |

## 4 代码开发2---去重

### 4.1 思路 ：

1. 把今日新增的活跃用户保存到redis中
2. 根据保存反馈得到用户是否已存在

### 4.2 设计Redis的kv

|  |  |
| --- | --- |
| key | value |
| dau:2019-01-22 | 设备id |

### 4.3 业务代码

**通过redis过滤**

|  |
| --- |
| **val** dauLoginfoDstream: DStream[JSONObject] = startLogInfoDStream.transform { rdd =>  *println*(**"前："** + rdd.count())  **val** logInfoRdd: RDD[JSONObject] = rdd.mapPartitions { startLogInfoItr =>  **val** jedis: Jedis = RedisUtil.*getJedisClient* **val** dauLogInfoList = **new** ListBuffer[JSONObject]  **val** startLogList: List[JSONObject] = startLogInfoItr.toList   **for** (startupJSONObj <- startLogList) {  **val** ts: lang.Long = startupJSONObj.getLong(**"ts"**)  **val** dt: String = **new** SimpleDateFormat(**"yyyy-MM-dd"**).format(**new** Date(ts))  **val** dauKey = **"dau:"** + dt  **val** ifFirst: lang.Long = jedis.sadd(dauKey, startupJSONObj.getJSONObject(**"common"**).getString(**"mid"**))  **if** (ifFirst == 1L) {  dauLogInfoList += startupJSONObj  }  }  jedis.close()  dauLogInfoList.toIterator  }  *// println("后：" + logInfoRdd.count())* logInfoRdd } |

## 5 代码开发3 --- 保存到ES中

### 5.1 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-query":{}  },  "mappings": {  "\_doc":{  "properties":{  "mid":{  "type":"keyword"  },  "uid":{  "type":"keyword"  },  "ar":{  "type":"keyword"  },  "ch":{  "type":"keyword"  },  "vc":{  "type":"keyword"  },  "dt":{  "type":"keyword"  },  "hr":{  "type":"keyword"  },  "mi":{  "type":"keyword"  },  "ts":{  "type":"date"  }      }  }  }  } |

### 5.2 建立case class

|  |
| --- |
| **case class** DauInfo(  mid:String,  uid:String,  ar:String,  ch:String,  vc:String,  **var** dt:String,  **var** hr:String,  **var** mi:String,  ts:Long) {  } |

### 5.3 pom.xml 中增加依赖

|  |
| --- |
| <**dependency**>  <**groupId**>io.searchbox</**groupId**>  <**artifactId**>jest</**artifactId**>  <**version**>5.3.3</**version**>  <**exclusions**>  <**exclusion**>  <**groupId**>org.slf4j</**groupId**>  <**artifactId**>slf4j-api</**artifactId**>  </**exclusion**>  </**exclusions**> </**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**>  *<!-- https://mvnrepository.com/artifact/org.elasticsearch/elasticsearch -->* <**dependency**>  <**groupId**>org.elasticsearch</**groupId**>  <**artifactId**>elasticsearch</**artifactId**>  <**version**>2.4.6</**version**> </**dependency**> |

### 5.4 业务保存代码

调整数据结构

|  |
| --- |
| **val** dauDstream: DStream[DauInfo] = dauLoginfoDstream.map { startupJsonObj =>  **val** dtHr: String = **new** SimpleDateFormat(**"yyyy-MM-dd HH:mm"**).format(**new** Date(startupJsonObj.getLong(**"ts"**)))  **val** dtHrArr: Array[String] = dtHr.split(**" "**)  **val** dt = dtHrArr(0)  **val** timeArr = dtHrArr(1).split(**":"**)  **val** hr = timeArr(0)  **val** mi = timeArr(1)  **val** commonJSONObj: JSONObject = startupJsonObj.getJSONObject(**"common"**)  DauInfo(commonJSONObj.getString(**"mid"**), commonJSONObj.getString(**"uid"**), commonJSONObj.getString(**"mid"**), commonJSONObj.getString(**"ch"**)  , commonJSONObj.getString(**"vc"**), dt, hr, mi, startupJsonObj.getLong(**"ts"**)) } |

保存，写入ES

|  |
| --- |
| dauDstream.foreachRDD{rdd=>  rdd.foreachPartition{dauInfoItr=>  **val** dauInfoWithIdList: List[(String, DauInfo)] = dauInfoItr.toList.map(dauInfo=>(dauInfo.dt+ **"\_"**+dauInfo.mid,dauInfo))  **val** dateStr: String = **new** SimpleDateFormat(**"yyyyMMdd"**).format(**new** Date())  MyEsUtil.*bulkInsert*(dauInfoWithIdList,**"gmall\_dau\_info\_"**+dateStr)    }  } |

## 6、 精确一次消费

### 6.1 定义

**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。

### 6.2 问题如何产生

**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。

### 6.3 如何解决

#### 策略一：利用关系型数据库的事务进行处理。

出现丢失或者重复的问题，核心就是偏移量的提交与数据的保存，不是原子性的。如果能做成要么数据保存和偏移量都成功，要么两个失败。那么就不会出现丢失或者重复了。

这样的话可以把存数据和偏移量放到一个事务里。这样就做到前面的成功，如果后面做失败了，就回滚前面那么就达成了原子性。

**问题与限制**

但是这种方式有限制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。

#### 策略二：手动提交偏移量+幂等性处理

咱们知道如果能够同时解决数据丢失和数据重复问题，就等于做到了精确一次消费。

那咱们就各个击破。

首先解决**数据丢失问题**，办法就是要等数据保存成功后再提交偏移量，所以就必须手工来控制偏移量的提交时机。

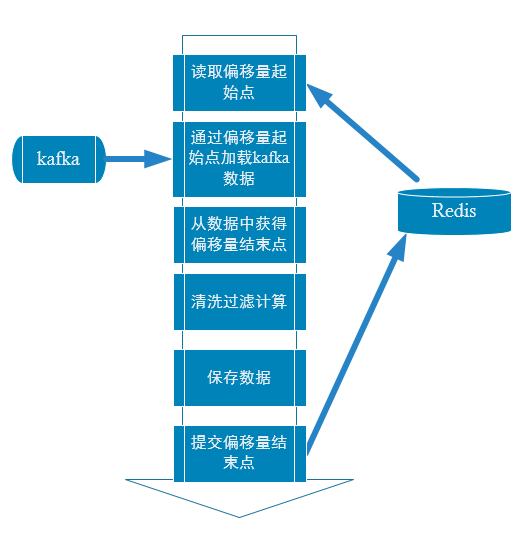
但是如果数据保存了，没等偏移量提交进程挂了，数据会被**重复消费**。怎么办？那就要把数据的保存做成幂等性保存。即同一批数据反复保存多次，数据不会翻倍，保存一次和保存一百次的效果是一样的。如果能做到这个，就达到了幂等性保存，就不用担心数据会重复了。

**难点**

话虽如此，在实际的开发中手动提交偏移量其实不难，难的是幂等性的保存，有的时候并不一定能保证。所以有的时候只能优先保证的数据不丢失。数据重复难以避免。即只保证了**至少一次消费的语义**。

### 6.4 手动提交偏移

#### 流程



**为什么用redis保存偏移量：**

本身kafka 0.9版本以后consumer的偏移量是保存在kafka的**\_\_consumer\_offsets**主题中。但是如果用这种方式管理偏移量，有一个限制就是在提交偏移量时，数据流的元素结构不能发生转变，即提交偏移量时数据流，必须是InputDStream[ConsumerRecord[String, String]] 这种结构。但是在实际计算中，数据难免发生转变，或聚合，或关联，一旦发生转变，就无法在利用以下语句进行偏移量的提交：

xxDstream.asInstanceOf[CanCommitOffsets].commitAsync(offsetRanges)

。

所以实际生产中通常会利用zookeeper,redis,mysql等工具对偏移量进行保存。

#### 偏移量管理类

|  |
| --- |
| **import** java.util  **import** org.apache.kafka.common.TopicPartition **import** org.apache.spark.streaming.kafka010.OffsetRange **import** redis.clients.jedis.Jedis **import** scala.collection.JavaConversions.\_ **object** OffsetManager {    */\*\*  \* 从Redis中读取偏移量  \** **@param groupId** *\** **@param topic** *\** **@return** *\*/* **def** getOffset(groupId:String,topic:String):Map[TopicPartition,Long]={  **var** offsetMap=*Map*[TopicPartition,Long]()   **val** jedisClient: Jedis = RedisUtil.*getJedisClient* **val** redisOffsetMap: util.Map[String, String] = jedisClient.hgetAll(**"offset:"**+groupId+**":"**+topic)  jedisClient.close()  **if**(redisOffsetMap!=**null**&&redisOffsetMap.isEmpty){  **null** }**else** {   **val** redisOffsetList: List[(String, String)] = redisOffsetMap.toList   **val** kafkaOffsetList: List[(TopicPartition, Long)] = redisOffsetList.map { **case** ( partition, offset) =>  (**new** TopicPartition(topic, partition.toInt), offset.toLong)  }  kafkaOffsetList.toMap  }  }   */\*\*  \* 偏移量写入到Redis中  \** **@param groupId** *\** **@param topic** *\** **@param offsetArray** *\*/* **def** saveOffset(groupId:String,topic:String ,offsetArray:Array[OffsetRange]):Unit= {  **if** (offsetArray != **null** && offsetArray.size > 0) {  **val** offsetMap: Map[String, String] = offsetArray.map { offsetRange =>  **val** partition: Int = offsetRange.partition  **val** untilOffset: Long = offsetRange.untilOffset  (partition.toString, untilOffset.toString)  }.toMap   **val** jedisClient: Jedis = RedisUtil.*getJedisClient* jedisClient.hmset(**"offset:"** + groupId + **":"** + topic, offsetMap)  jedisClient.close()  }  }  } |

根据自定义偏移量加载的读取kafka数据

|  |
| --- |
| **def** main(args: Array[String]): Unit = {  **val** sparkConf: SparkConf = **new** SparkConf().setMaster(**"local[\*]"**).setAppName(**"dau\_app"**)  **val** ssc = **new** StreamingContext(sparkConf, *Seconds*(5))  **val** groupId = **"GMALL\_DAU\_CONSUMER"  val** topic = **"GMALL\_START"** *//从redis读取偏移量* **val** startupOffsets: Map[TopicPartition, Long] = OffsetManager.*getOffset*(groupId,topic)   *//根据偏移起始点获得数据* **val** startupInputDstream: InputDStream[ConsumerRecord[String, String]] = MyKafkaUtil.*getKafkaStream*(topic, ssc,startupOffsets,groupId)    *//获得偏移结束点* **var** startupOffsetRanges: Array[OffsetRange] = Array.*empty*[OffsetRange]  **val** startupInputGetOffsetDstream: DStream[ConsumerRecord[String, String]] = startupInputDstream.transform { rdd =>  startupOffsetRanges = rdd.asInstanceOf[HasOffsetRanges].offsetRanges  rdd  }    **val** startLogInfoDStream: DStream[JSONObject] = startupInputGetOffsetDstream.map { record =>  **val** startupJson: String = record.value()  **val** startupJSONObj: JSONObject = JSON.*parseObject*(startupJson)  **val** ts: lang.Long = startupJSONObj.getLong(**"ts"**)  startupJSONObj    }  startLogInfoDStream.print(100)  …..  …..  …..  …..  dauDstream.foreachRDD{rdd=>  rdd.foreachPartition{dauInfoItr=>  *///可以观察偏移量* if(startupOffsetRanges!=null&&startupOffsetRanges.size>0){  val offsetRange: OffsetRange = startupOffsetRanges(TaskContext.get().partitionId())  println("from:"+offsetRange.fromOffset +" --- to:"+offsetRange.untilOffset) }  **val** dauInfoWithIdList: List[(String, DauInfo)] = dauInfoItr.toList.map(dauInfo=>(dauInfo.dt+ **"\_"**+dauInfo.mid,dauInfo))  **val** dateStr: String = **new** SimpleDateFormat(**"yyyyMMdd"**).format(**new** Date())  MyEsUtil.*bulkInsert*(dauInfoWithIdList,**"gmall\_dau\_info\_"**+dateStr)    } //在保存时最后提交偏移量  OffsetManager.*saveOffset*(groupId ,topic, startupOffsetRanges)  *//如果流发生了转换，无法用以下方法提交偏移量 //dauDstream.asInstanceOf[CanCommitOffsets].commitAsync(startupOffsetRanges)*  }  …..  ….. |

# 日活数据查询接口



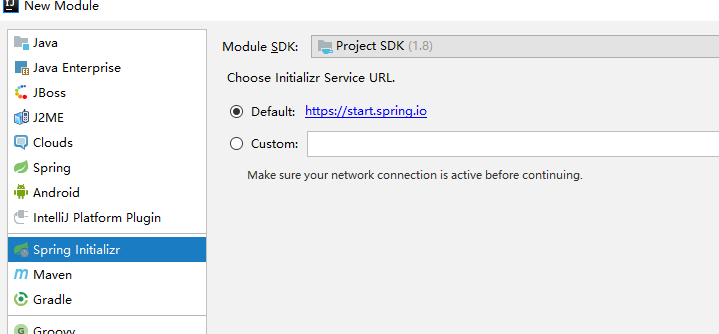
## 1 访问路径

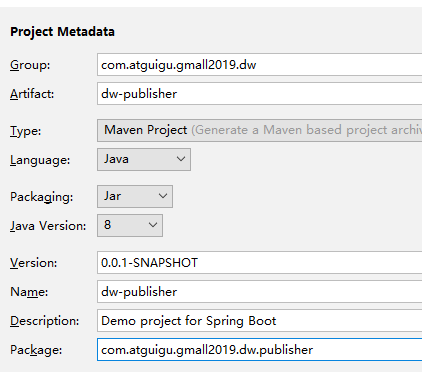
|  |  |
| --- | --- |
| 总数 | http://publisher:8070/realtime-total?date=2019-02-01 |
| 分时统计 | http://publisher:8070/realtime-hour?id=dau&date=2019-02-01 |

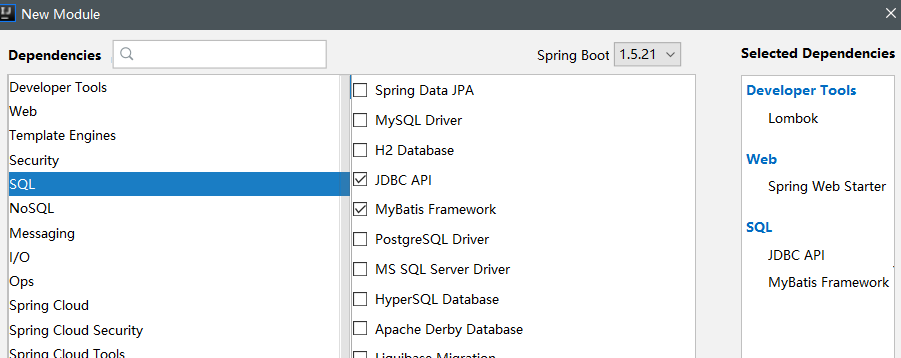
## 2 要求数据格式

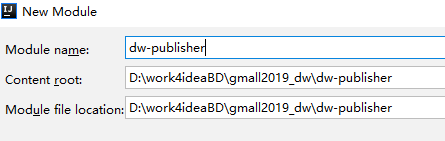
|  |  |
| --- | --- |
| 总数 | [{"id":"dau","name":"新增日活","value":1200},  {"id":"new\_mid","name":"新增设备","value":233} ] |
| 分时统计 | {"yesterday":{"11":383,"12":123,"17":88,"19":200 },  "today":{"12":38,"13":1233,"17":123,"19":688 }} |

## 3 搭建发布工程









## 4 配置文件

### 4.1 pom.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <parent>  <artifactId>gmall2019\_dw</artifactId>  <groupId>com.atguigu.gmall2019.dw</groupId>  <version>1.0-SNAPSHOT</version>  </parent>  <groupId>com.atguigu.gmall2019.dw.publisher</groupId>  <artifactId>dw-publisher</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>dw-publisher</name>  <description>Demo project for Spring Boot</description>   <properties>  <java.version>1.8</java.version>  </properties>   <dependencies>    <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  *<!-- https://mvnrepository.com/artifact/io.searchbox/jest -->* <dependency>  <groupId>com.atguigu.gmall2019.dw</groupId>  <artifactId>dw-common</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>   <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  *<!-- https://mvnrepository.com/artifact/org.jsoup/jsoup -->*    </dependencies>   <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

### 4.2 application.properties

|  |
| --- |
| server.port=8070    logging.level.root=error |

## 5 代码部分

|  |  |  |
| --- | --- | --- |
| 控制层 | PublisherController | 实现接口的web发布 |
| 服务层 | PublisherService | 数据业务查询interface |
| PublisherServiceImpl | 业务查询的实现类 |
| 主程序 | GmallPublisherApplication | 增加扫描包 |

### 5.1 GmallPublisherApplication 增加扫描包

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

### 5.2 controller层

|  |
| --- |
| **import** com.alibaba.fastjson.JSON; **import** com.alibaba.fastjson.JSONObject; **import** com.atguigu.gmall2019.dw.publisher.service.PublisherService; **import** org.apache.commons.lang.time.DateUtils; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.web.bind.annotation.GetMapping; **import** org.springframework.web.bind.annotation.RequestParam; **import** org.springframework.web.bind.annotation.RestController;  **import** java.text.ParseException; **import** java.text.SimpleDateFormat; **import** java.util.\*;  @RestController **public class** PublisherController {   @Autowired EsService **esService**;  *//@RequestMapping(value = "realtime-total" ,method = RequestMethod.GET)* @GetMapping(**"realtime-total"**) **public** String realtimeTotal(@RequestParam(**"date"**) String dt){  List<Map> totalList=**new** ArrayList<>();  HashMap dauMap = **new** HashMap<>();  dauMap.put(**"id"**,**"dau"**);  dauMap.put(**"name"**,**"新增日活"**);  Long dauTotal = **esService**.getDauTotal(dt);  dauMap.put(**"value"**,dauTotal);   totalList.add(dauMap);   HashMap newMidMap = **new** HashMap<>();  newMidMap.put(**"id"**,**"dau"**);  newMidMap.put(**"name"**,**"新增设备"**);  newMidMap.put(**"value"**,233);   totalList.add(newMidMap);   **return** JSON.*toJSONString*(totalList);  }  @GetMapping(**"realtime-hour"**) **public** String realtimeHour(@RequestParam(value = **"id"**,defaultValue =**"-1"** ) String id ,@RequestParam(**"date"**) String dt ){  **if**(id.equals(**"dau"**)){  Map<String,Map> hourMap=**new** HashMap<>();  Map dauHourTdMap = **esService**.getDauHour(dt);  hourMap.put(**"today"**,dauHourTdMap);  String yd = getYd(dt);  Map dauHourYdMap = **esService**.getDauHour(yd);  hourMap.put(**"yesterday"**,dauHourYdMap);  **return** JSON.*toJSONString*(hourMap);  }  **return null**; }  **private** String getYd(String td){  SimpleDateFormat dateFormat = **new** SimpleDateFormat(**"yyyy-MM-dd"**);  String yd=**null**;  **try** {  Date tdDate = dateFormat.parse(td);  Date ydDate = DateUtils.*addDays*(tdDate, -1);  yd=dateFormat.format(ydDate);  } **catch** (ParseException e) {  e.printStackTrace();  **throw new** RuntimeException(**"日期格式转变失败"**);  }  **return** yd; } |

### 5.3 service层

|  |
| --- |
| **import** java.util.Map;  **public interface** EsService {   *//日活的总数查询* **public** Long getDauTotal(String date);  *//日活的分时查询* **public** Map getDauHour(String date); } |

### 5.4 service层实现类

|  |
| --- |
| @Service **public class** EsServiceImpl **implements** EsService {    **public static void** main(String[] args) {  **new** EsServiceImpl().getDauTotal(**"2020-05-14"**);  }   @Autowired  JestClient **jestClient**;   @Override  **public** Long getDauTotal(String date) {  SearchSourceBuilder searchSourceBuilder = **new** SearchSourceBuilder();  searchSourceBuilder.query(**new** MatchAllQueryBuilder());   String query = searchSourceBuilder.toString();  date=date.replace(**"-"**,**""**);  String indexName=**"gmall1122\_dau\_info\_"**+date+**"-query"**;  Search search = **new** Search.Builder(query).addIndex(indexName).addType(**"\_doc"**).build();  Long total=0L;  **try** {  SearchResult searchResult = **jestClient**.execute(search);  **if**(searchResult.getTotal()!=**null**){  total = searchResult.getTotal();  }  } **catch** (IOException e) {  e.printStackTrace();  **throw new** RuntimeException(**"查询ES异常"**);  }  **return** total;  }   @Override  **public** Map getDauHour(String date) {   SearchSourceBuilder searchSourceBuilder = **new** SearchSourceBuilder();   TermsAggregationBuilder aggregationBuilder = AggregationBuilders.*terms*(**"groupby\_hr"**).field(**"hr"**).size(24);  searchSourceBuilder.aggregation(aggregationBuilder);   String query = searchSourceBuilder.toString();   date=date.replace(**"-"**,**""**);  String indexName=**"gmall1122\_dau\_info\_"**+date+**"-query"**;   Search search = **new** Search.Builder(query).addIndex(indexName).addType(**"\_doc"**).build();  Map aggsMap=**new** HashMap();  **try** {  SearchResult searchResult = **jestClient**.execute(search);  **if**(searchResult.getAggregations().getTermsAggregation(**"groupby\_hr"**)!=**null**){  List<TermsAggregation.Entry> buckets = searchResult.getAggregations().getTermsAggregation(**"groupby\_hr"**).getBuckets();  **for** (TermsAggregation.Entry bucket : buckets) {  aggsMap.put( bucket.getKey(),bucket.getCount());  }  }   } **catch** (IOException e) {  e.printStackTrace();  **throw new** RuntimeException(**"查询ES异常"**);  }   **return** aggsMap;  } } |

## 6 搭建可视化工程进行对接

拷贝前端工程，启动主程序启动。

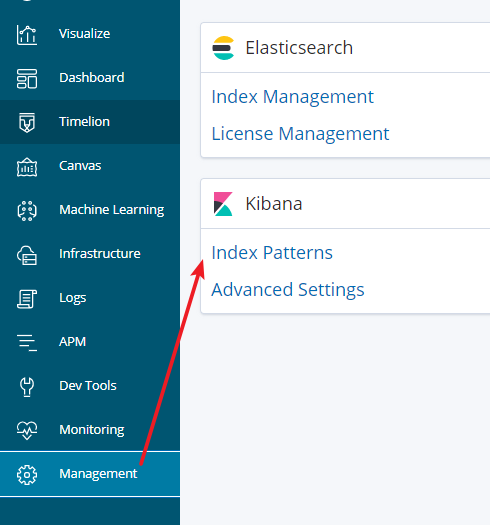
# 第三章 利用kibana 搭建数据可视化

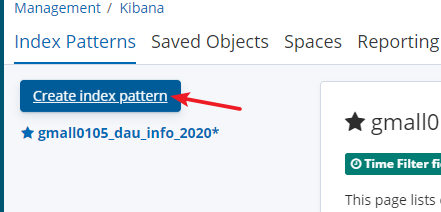
如果数据保存在Elasticsearch那么利用kibana进行可视化展示是一种非常高效的可视化解决方案。

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。

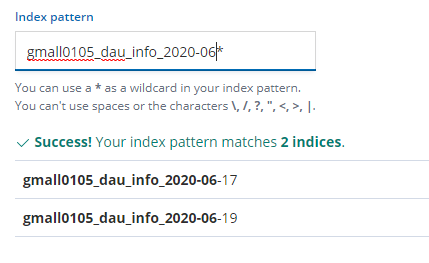
## 步骤一 ：创建 index patten 其实就是创建数据源 确定数据范围。

在 Management中 创建index patterns

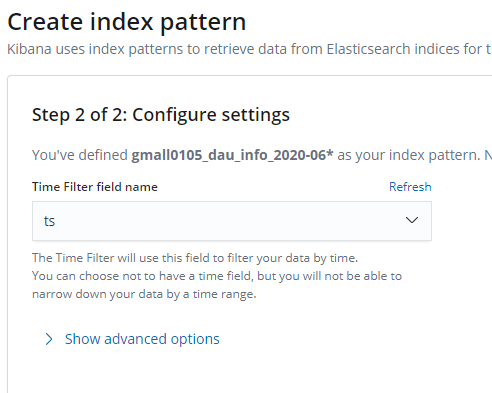




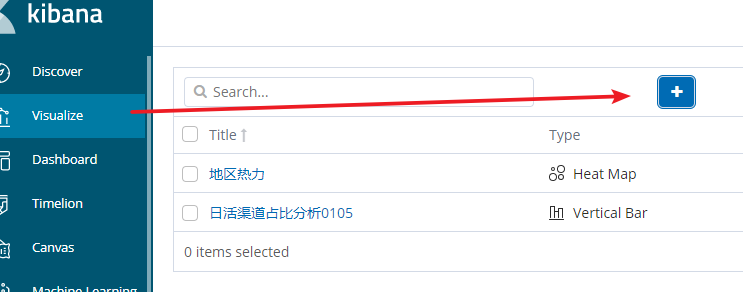
由于索引是以天为时间单位建立，通过用“\*”来控制后缀可以得到不同的数据范围。



选择一个时间字段，用于对可视化图形灵活筛选时间范围。

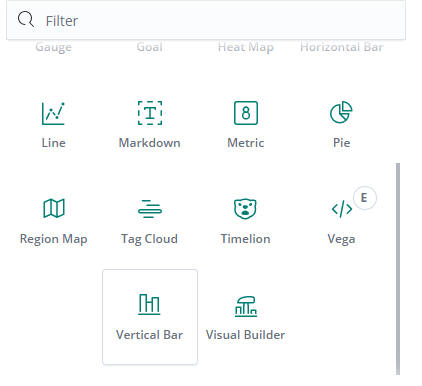


## 步骤二： 配置单图

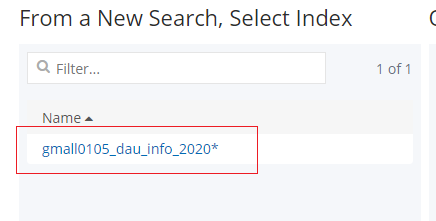


在VIsualize选择加号

选择一个可视化图形



选择一个index pattern



### 柱状图配置

Y轴配置度量值

1 Aggregation:聚合方法

2 Customer Label: Y轴的显示标签

X 轴配置维度列

1 Aggregation: 分组方法

2 Field： 分组字段

3 Order by : 分组如何排序

4 Order 排序顺序

5 size 列出前N名

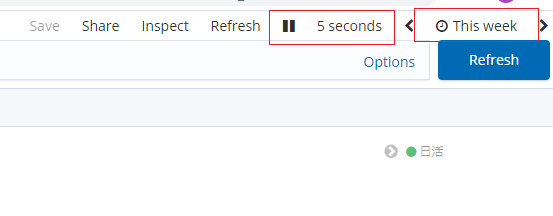
6 Group other …. : 是否把不在TopN的分组组合起来组成一个Other列

7 Show Missing Values 是否把空值数据汇总起来组成一个Missing列。

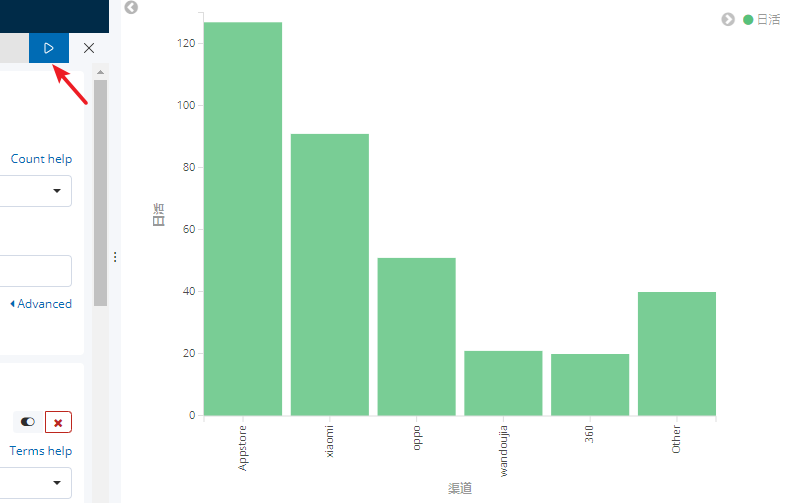
8 Custom Label

Customer Label: X轴的显示标签

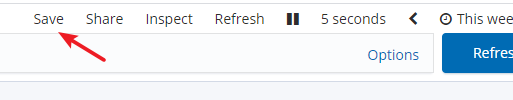
右上角选择刷新频率和时间范围



配置好后点击 执行按钮



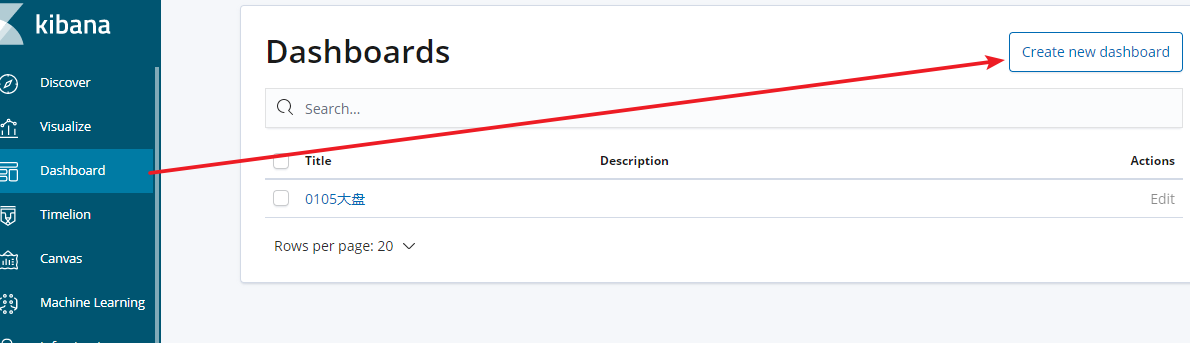
最后保存，命名。



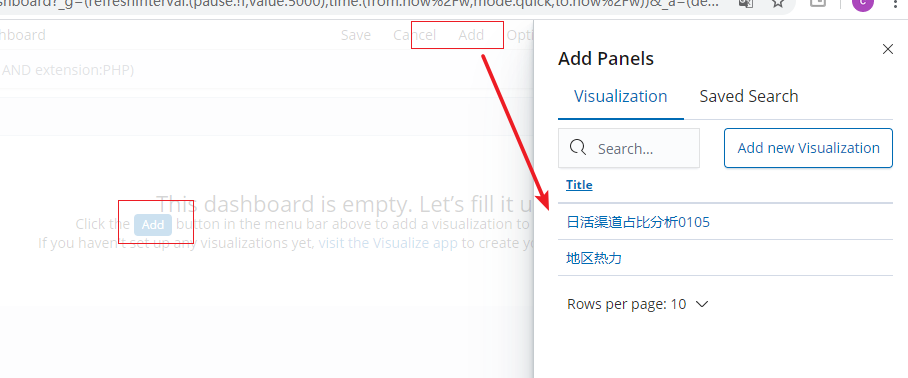
## 步骤三 配置仪表盘

仪表盘 Dashboard，就是配置多个图在一个页面上同时展示。

选择Dashboard 然后选择Create new DashBoard



然后选择add



从之前配置的可视化图中，选择图形。

选好后选择save保存。就可以在Dashboard列表中查看大盘图了。

