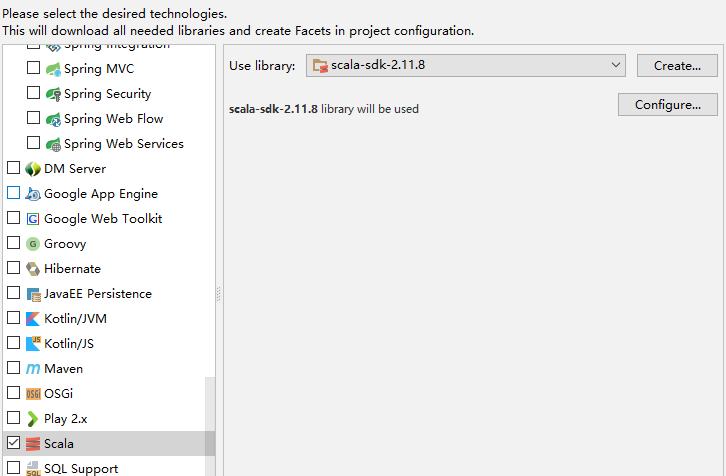
电商数仓项目--实时计算

# 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HBASE或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">  <parent>  <artifactId>gmall2019\_dw</artifactId>  <groupId>com.bawei.gmall2019.dw</groupId>  <version>1.0-SNAPSHOT</version>  </parent>  <modelVersion>4.0.0</modelVersion>   <artifactId>dw-realtime</artifactId>   <dependencies>   <dependency>  <groupId>com.bawei.gmall2019.dw</groupId>  <artifactId>dw-common</artifactId>  <version>1.0-SNAPSHOT</version>  </dependency>   <dependency>  <groupId>org.apache.spark</groupId>  <artifactId>spark-core\_2.11</artifactId>  </dependency>  <dependency>  <groupId>org.apache.spark</groupId>  <artifactId>spark-streaming\_2.11</artifactId>  </dependency>  <dependency>  <groupId>org.apache.kafka</groupId>  <artifactId>kafka-clients</artifactId>  <version>0.10.2.1</version>  </dependency>  <dependency>  <groupId>org.apache.spark</groupId>  <artifactId>spark-streaming-kafka-0-10\_2.11</artifactId>  </dependency>    <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  <version>2.9.0</version>  </dependency>   <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>  </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>3.2.2</version>  <executions>  <execution>  *<!-- 声明绑定到maven的compile阶段 -->* <goals>  <goal>compile</goal>  <goal>testCompile</goal>  </goals>  </execution>  </executions>  </plugin>     </plugins>  </build> </project> |

### 3.2 工具类

#### 3.2.1 MykafkaUtil

|  |
| --- |
| **import** org.apache.kafka.common.serialization.StringDeserializer **import** java.util.Properties  **import** org.apache.kafka.clients.consumer.ConsumerRecord **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* **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"** -> **"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"** -> (**true**: 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  } } |

#### 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  } } |

### 3.2 制作case class Startuplog

|  |
| --- |
| **case class** StartUpLog(mid:String,  uid:String,  appid:String,  area:String,  os:String,  ch:String,  logType:String,  vs:String,  **var** logDate:String,  **var** logHour:String,  **var** ts:Long  ) {  } |

### 3.4 业务类—消费kafka

|  |
| --- |
| **object** RealtimeStartupApp {   **def** main(args: Array[String]): Unit = {  **val** sparkConf: SparkConf = **new** SparkConf().setMaster(**"local[\*]"**).setAppName(**"gmall2019"**)  **val** sc = **new** SparkContext(sparkConf)  **val** ssc = **new** StreamingContext(sc,*Seconds*(10))   **val** startupStream: InputDStream[ConsumerRecord[String, String]] = MyKafkaUtil.*getKafkaStream*(GmallConstants.*KAFKA\_TOPIC\_STARTUP*,ssc)  *// startupStream.map(\_.value()).foreachRDD{ rdd=> // println(rdd.collect().mkString("\n")) // }* **val** startupLogDstream: DStream[StartUpLog] = startupStream.map(\_.value()).map { log =>  *// println(s"log = ${log}")* **val** startUpLog: StartUpLog = JSON.*parseObject*(log, *classOf*[StartUpLog])  startUpLog  }  } |

## 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. 每条数据经过过滤，去掉redis中已有的用户
3. ？？

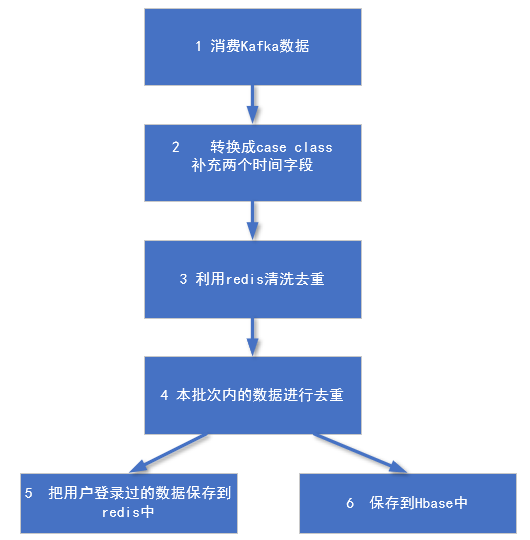
### 4.2 设计Redis的kv

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

### 4.3 业务代码

|  |
| --- |
| **import** java.util  **import** java.text.SimpleDateFormat **import** java.util.Date  **import** com.alibaba.fastjson.JSON **import** com.bawei.gmall.constant.GmallConstants **import** com.bawei.gmall2019.realtime.bean.StartupLog **import** com.bawei.gmall2019.realtime.util.{MyKafkaUtil, RedisUtil} **import** org.apache.hadoop.conf.Configuration **import** org.apache.kafka.clients.consumer.ConsumerRecord **import** org.apache.spark.SparkConf **import** org.apache.spark.broadcast.Broadcast **import** org.apache.spark.rdd.RDD **import** org.apache.spark.streaming.dstream.{DStream, InputDStream} **import** org.apache.spark.streaming.{Seconds, StreamingContext} **import** redis.clients.jedis.Jedis **import** org.apache.phoenix.spark.\_  **object** DauApp {   **def** main(args: Array[String]): Unit = {  **val** sparkConf: SparkConf = **new** SparkConf().setMaster(**"local[\*]"**).setAppName(**"dau\_app"**)  **val** ssc = **new** StreamingContext(sparkConf,*Seconds*(5))  *// 1 消费kafka*   **val** inputDstream: InputDStream[ConsumerRecord[String, String]] = MyKafkaUtil.*getKafkaStream*(GmallConstants.*KAFKA\_TOPIC\_STARTUP*,ssc)   *//2 数据流 转换 结构变成case class 补充两个时间字段* **val** startuplogDstream: DStream[StartupLog] = inputDstream.map { record =>  **val** jsonStr: String = record.value()  **val** startupLog: StartupLog = JSON.*parseObject*(jsonStr, *classOf*[StartupLog])   **val** dateTimeStr: String = **new** SimpleDateFormat(**"yyyy-MM-dd HH"**).format(**new** Date(startupLog.ts))  **val** dateArr: Array[String] = dateTimeStr.split(**" "**)  startupLog.logDate = dateArr(0)  startupLog.logHour = dateArr(1)  startupLog  }    startuplogDstream.cache()    *// 3 利用用户清单进行过滤 去重 只保留清单中不存在的用户访问记录* **val** filteredDstream: DStream[StartupLog] = startuplogDstream.transform { rdd =>  **val** jedis: Jedis = RedisUtil.*getJedisClient //driver //按周期执行* **val** dateStr: String = **new** SimpleDateFormat(**"yyyy-MM-dd"**).format(**new** Date())   **val** key = **"dau:"** + dateStr  **val** dauMidSet: util.Set[String] = jedis.smembers(key)  jedis.close()   **val** dauMidBC: Broadcast[util.Set[String]] = ssc.sparkContext.broadcast(dauMidSet)  *println*(**"过滤前："** + rdd.count())  **val** filteredRDD: RDD[StartupLog] = rdd.filter { startuplog => *//executor* **val** dauMidSet: util.Set[String] = dauMidBC.value  !dauMidSet.contains(startuplog.mid)  }  *println*(**"过滤后："** + filteredRDD.count())  filteredRDD   }   *// 4 批次内进行去重：：按照mid 进行分组，每组取第一个值* **val** groupbyMidDstream: DStream[(String, Iterable[StartupLog])] = filteredDstream.map(startuplog=>(startuplog.mid,startuplog)).groupByKey()  **val** distictDstream: DStream[StartupLog] = groupbyMidDstream.flatMap { **case** (mid, startupLogItr) =>  startupLogItr.toList.take(1)  }    *// 5 保存今日访问过的用户(mid)清单 -->Redis 1 key类型 ： set 2 key ： dau:2019-xx-xx 3 value : mid* distictDstream.foreachRDD{rdd=>  *//driver* rdd.foreachPartition{ startuplogItr=>  **val** jedis:Jedis=RedisUtil.*getJedisClient //executor* **for** (startuplog <- startuplogItr ) {  **val** key= **"dau:"**+startuplog.logDate  jedis.sadd(key,startuplog.mid)  *println*(startuplog)  }  jedis.close()  } }      ssc.start()  ssc.awaitTermination()   }  } |

### 4.4 流程图



## 5 代码开发3 --- 保存到Hbase 中

### 5.1 Phoenix --- Hbase的SQL化插件

技术详情参见 《大数据技术之phoenix》

### 5.2 利用Phoenix建立数据表

|  |
| --- |
| create table gmall2019\_dau  (  mid varchar,  uid varchar,  appid varchar,  area varchar,  os varchar,  ch varchar,  type varchar,  vs varchar,  logDate varchar,  logHour varchar,  ts bigint  CONSTRAINT dau\_pk PRIMARY KEY (mid, logDate)  ); |

### 5.3 pom.xml 中增加依赖

|  |
| --- |
| <**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**>  </**dependency**> |

### 5.4 业务保存代码

|  |
| --- |
| *//把数据写入hbase+phoenix* distictDstream.foreachRDD{rdd=>  rdd.saveToPhoenix(**"GMALL2019\_DAU"**,*Seq*(**"MID"**, **"UID"**, **"APPID"**, **"AREA"**, **"OS"**, **"CH"**, **"TYPE"**, **"VS"**, **"LOGDATE"**, **"LOGHOUR"**, **"TS"**) ,**new** Configuration,Some(**"hadoop1,hadoop2,hadoop3:2181"**)) } |

# 日活数据查询接口



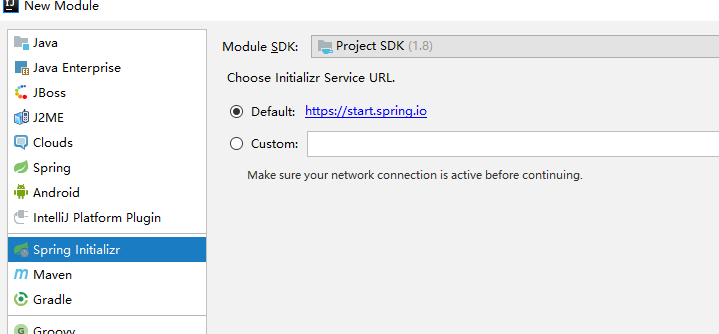
## 1 访问路径

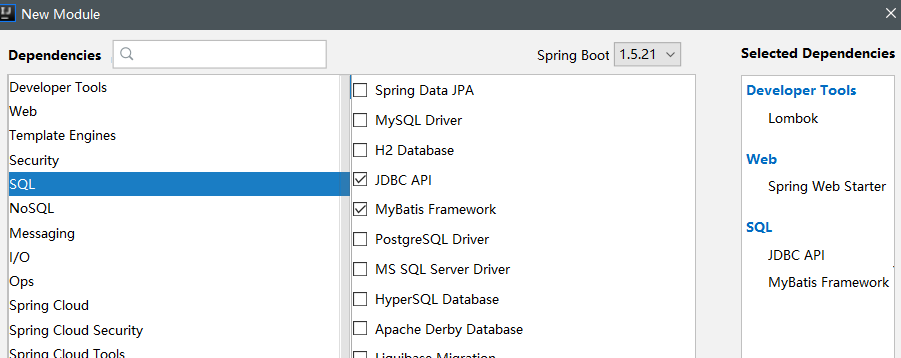
|  |  |
| --- | --- |
| 总数 | http://publisher:8070/realtime-total?date=2019-02-01 |
| 分时统计 | http://publisher:8070/realtime-hour?id=dau&date=2019-02-01 |

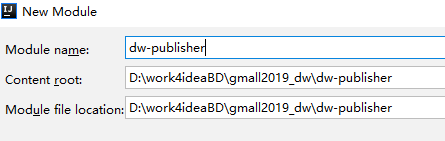
## 2 要求数据格式

|  |  |
| --- | --- |
| 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.bawei.gmall2019.dw</groupId>  <version>1.0-SNAPSHOT</version>  </parent>  <groupId>com.bawei.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.bawei.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 -->* <dependency>  <groupId>org.apache.phoenix</groupId>  <artifactId>phoenix-core</artifactId>  <version>4.14.2-HBase-1.3</version> </dependency>    <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-jdbc</artifactId> </dependency>  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.3.4</version> </dependency>  <dependency>  <groupId>com.google.guava</groupId>  <artifactId>guava</artifactId>  <version>20.0</version> </dependency>      </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    spring.datasource.driver-class-name=org.apache.phoenix.jdbc.PhoenixDriver spring.datasource.url=jdbc:phoenix:hadoop1,hadoop2,hadoop3:2181 spring.datasource.data-username= spring.datasource.data-password=  *# mybatis #mybatis.typeAliasesPackage=com.example.phoenix.entity* mybatis.mapperLocations=classpath:mapper/\*.xml mybatis.configuration.map-underscore-to-camel-case=true |

## 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 |
| 数据层 | DauMapper | 数据层查询的interface |
| DauMapper.xml | 数据层查询的实现配置 |
| 主程序 | GmallPublisherApplication | 增加扫描包 |

### 5.1 GmallPublisherApplication 增加扫描包

|  |
| --- |
| @SpringBootApplication @MapperScan(basePackages = **"com.bawei.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.bawei.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  PublisherService **publisherService**;   @GetMapping(**"realtime-total"**)  **public** String realtimeHourDate(@RequestParam(**"date"**) String date) {  List<Map> list = **new** ArrayList<Map>();  *// 日活总数* **int** dauTotal = **publisherService**.getDauTotal(date);  Map dauMap=**new** HashMap<String,Object>();  dauMap.put(**"id"**,**"dau"**);  dauMap.put(**"name"**,**"新增日活"**);  dauMap.put(**"value"**,dauTotal);  list.add(dauMap);   *// 新增用户* **int** newMidTotal = **publisherService**.getNewMidTotal(date);  Map newMidMap=**new** HashMap<String,Object>();  newMidMap.put(**"id"**,**"new\_mid"**);  newMidMap.put(**"name"**,**"新增用户"**);  newMidMap.put(**"value"**,newMidTotal);  list.add(newMidMap);   **return** JSON.*toJSONString*(list);  }    @GetMapping(**"realtime-hours"**)  **public** String realtimeHourDate(@RequestParam(**"id"**) String id,@RequestParam(**"date"**) String date){   **if**( **"dau"**.equals(id)){  Map dauHoursToday = **publisherService**.getDauHours(date);  JSONObject jsonObject = **new** JSONObject();  jsonObject.put(**"today"**,dauHoursToday);  String yesterdayDateString=**""**;  **try** {  Date dateToday = **new** SimpleDateFormat(**"yyyy-MM-dd"**).parse(date);  Date dateYesterday = DateUtils.*addDays*(dateToday, -1);  yesterdayDateString=**new** SimpleDateFormat(**"yyyy-MM-dd"**).format(dateYesterday);   } **catch** (ParseException e) {  e.printStackTrace();  }  Map dauHoursYesterday = **publisherService**.getDauHours(yesterdayDateString);  jsonObject.put(**"yesterday"**,dauHoursYesterday);  **return** jsonObject.toJSONString();  }    **if**( **"new\_order\_totalamount"**.equals(id)){  String newOrderTotalamountJson = **publisherService**.getNewOrderTotalAmountHours(date);  **return** newOrderTotalamountJson;  }  **return null**;  }  } |

### 5.3 service层

|  |
| --- |
| **public interface** PublisherService {   **public int** getDauTotal(String date );   **public** Map getDauHours(String date );    } |

### 5.4 service层实现类

|  |
| --- |
| @Service **public class** PublisherServiceImpl **implements** PublisherService{   @Autowired DauMapper **dauMapper**;  @Override **public** Integer getDauTotal(String date) {  **return dauMapper**.selectDauTotal(date); }  @Override **public** Map getDauHour(String date) {  HashMap dauHourMap=**new** HashMap();  List<Map> dauHourList = **dauMapper**.selectDauTotalHourMap(date);  **for** (Map map : dauHourList) {  dauHourMap.put(map.get(**"LH"**),map.get(**"CT"**));  }   **return** dauHourMap; }  } |

### 5.5 数据层 mapper

|  |
| --- |
| **import** java.util.List; **import** java.util.Map; **public interface** DauMapper {   **public** Integer selectDauTotal(String date);  **public** List<Map> selectDauTotalHourMap(String date); } |

### 5.6 数据层 实现配置

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <!DOCTYPE mapper SYSTEM "http://mybatis.org/dtd/mybatis-3-mapper.dtd" *>* <mapper namespace="com.bawei.gmall2019.publisher.mapper.DauMapper">  <select id="selectDauTotal" resultType="Integer">  select *count*(*\**) from gmall2019\_dau where logdate=#{date}  </select>    <select id="selectDauTotalHourMap" resultMap="dauTotalHour">  select LOGHOUR lh, *count*(*\**) ct from gmall2019\_dau where LOGDATE=#{date}  group by LOGHOUR  </select>  <resultMap id="dauTotalHour" type="java.util.Map" autoMapping="true"> </resultMap>   </mapper> |

## 6 根据查询条件增加索引

|  |
| --- |
| create local index idx\_logdate\_loghour on gmall2019\_dau(logdate,loghour) |

## 7 搭建可视化工程进行对接