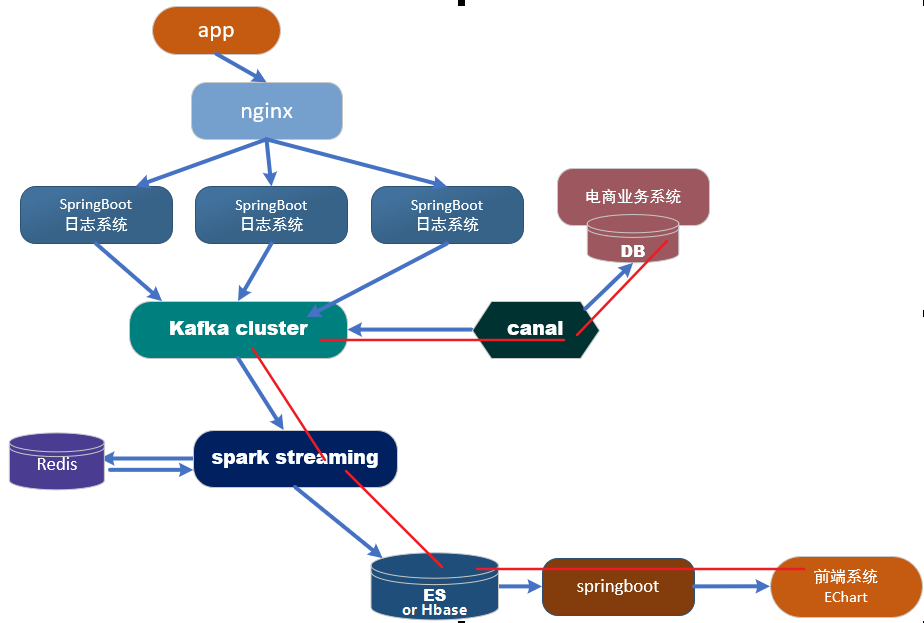
电商数仓项目--实时计算（交易额）

# 采集数据

## 1 框架流程



## 2 canal 入门

### 2.1 什么是 canal

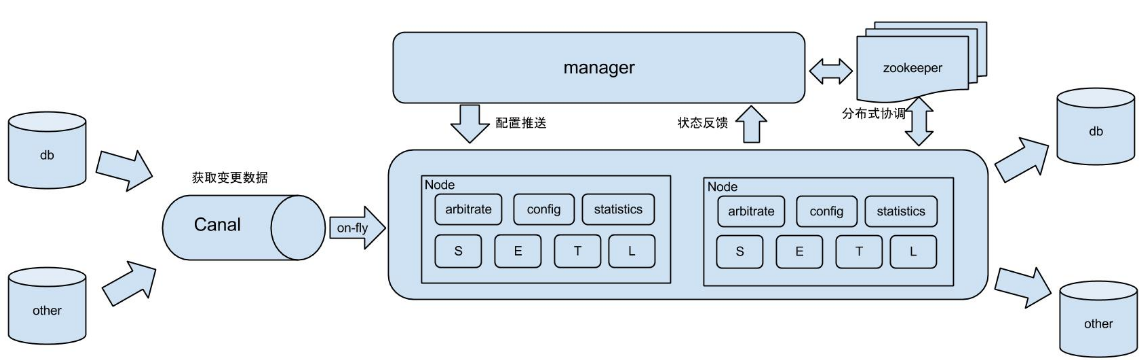
阿里巴巴B2B公司，因为业务的特性，卖家主要集中在国内，买家主要集中在国外，所以衍生出了杭州和美国异地机房的需求，从2010年开始，阿里系公司开始逐步的尝试基于数据库的日志解析，获取增量变更进行同步，由此衍生出了增量订阅&消费的业务。

canal是用java开发的基于数据库增量日志解析，提供增量数据订阅&消费的中间件。目前，canal主要支持了MySQL的binlog解析，解析完成后才利用canal client 用来处理获得的相关数据。（数据库同步需要阿里的otter中间件，基于canal）。

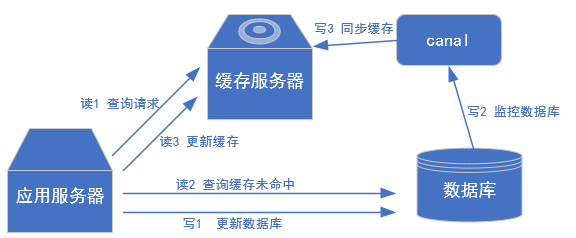
### 2.2 使用场景

1）原始场景： 阿里otter中间件的一部分

otter是阿里用于进行异地数据库之间的同步框架，canal是其中一部分。



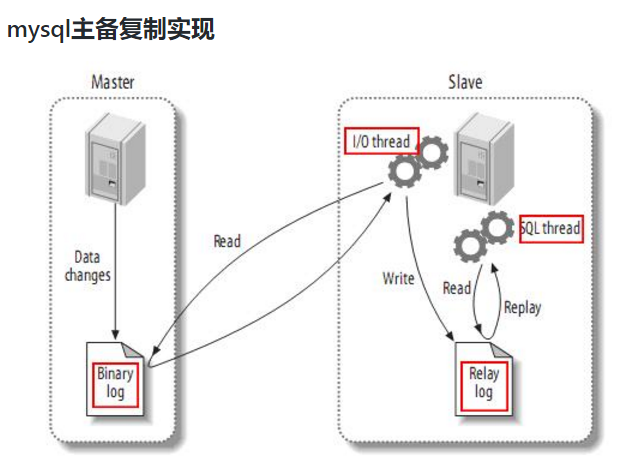
2） 常见场景1：更新缓存



3） 场景2：抓取业务数据新增变化表，用于制作拉链表。

4） 场景3：抓取业务表的新增变化数据，用于制作实时统计。

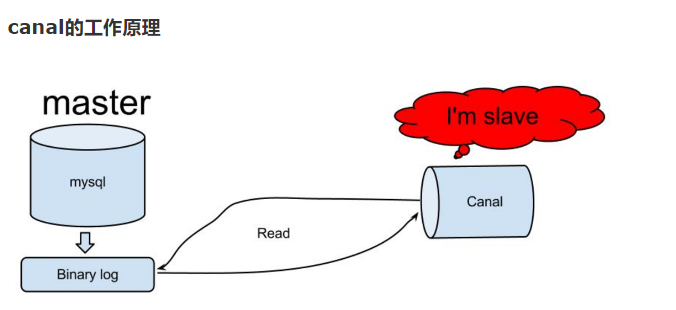
### 2.3 canal的工作原理



复制过程分成三步：

1. Master主库将改变记录，写到二进制日志(binary log)中
2. Slave从库向mysql master发送dump协议，将master主库的binary log events拷贝到它的中继日志(relay log)；
3. Slave从库读取并重做中继日志中的事件，将改变的数据同步到自己的数据库。

canal的工作原理很简单，就是**把自己伪装成slave，假装从master复制数据**。



### 2.4 mysql的binlog

#### 2.4.1 什么是binlog

MySQL的二进制日志可以说是MySQL最重要的日志了，它记录了所有的DDL和DML(除了数据查询语句)语句，以事件形式记录，还包含语句所执行的消耗的时间，MySQL的二进制日志是事务安全型的。

一般来说开启二进制日志大概会有1%的性能损耗 。二进制有两个最重要的使用场景:

其一：MySQL Replication在Master端开启binlog，Mster把它的二进制日志传递给slaves来达到master-slave数据一致的目的。

其二：自然就是数据恢复了，通过使用mysqlbinlog工具来使恢复数据。

二进制日志包括两类文件：二进制日志索引文件（文件名后缀为.index）用于记录所有的二进制文件，二进制日志文件（文件名后缀为.00000\*）记录数据库所有的DDL和DML(除了数据查询语句)语句事件。

#### 2.4.2 binlog的开启

在mysql的配置文件(Linux: /etc/my.cnf , Windows: \my.ini)下,修改配置

在[mysqld] 区块

设置/添加

|  |
| --- |
| log-bin=mysql-bin |

这个表示binlog日志的前缀是mysql-bin ，以后生成的日志文件就是 mysql-bin.123456 的文件后面的数字按顺序生成。 每次mysql重启或者到达单个文件大小的阈值时，新生一个文件，按顺序编号。

### 2.4.3 binlog的分类设置

mysql binlog的格式，那就是有三种，分别是STATEMENT,MIXED,ROW。

在配置文件中可以选择配置

|  |
| --- |
| binlog\_format=row |

区别：

**1） statement**

语句级，binlog会记录每次一执行写操作的语句。

相对row模式节省空间，但是可能产生不一致性，比如

update tt set create\_date=now()

如果用binlog日志进行恢复，由于执行时间不同可能产生的数据就不同。

优点： 节省空间

缺点： 有可能造成数据不一致。

**2） row**

行级， binlog会记录每次操作后每行记录的变化。

优点：保持数据的绝对一致性。因为不管sql是什么，引用了什么函数，他只记录执行后的效果。

缺点：占用较大空间。

**3） mixed**

statement的升级版，一定程度上解决了，因为一些情况而造成的statement模式不一致问题

在某些情况下譬如：

当函数中包含 UUID() 时；

包含 AUTO\_INCREMENT 字段的表被更新时；

执行 INSERT DELAYED 语句时；

用 UDF 时；

会按照 ROW的方式进行处理

优点：节省空间，同时兼顾了一定的一致性。

缺点：还有些极个别情况依旧会造成不一致，另外statement和mixed对于需要对binlog的监控的情况都不方便。

## 3 mysql的准备

### 3.1 导入模拟业务数据库

### 3.2 赋权限

在mysql中执行

|  |
| --- |
| GRANT SELECT, REPLICATION SLAVE, REPLICATION CLIENT ON \*.\* TO 'canal'@'%' IDENTIFIED BY 'canal' ; |

### 3.3 修改/etc/my.cnf文件

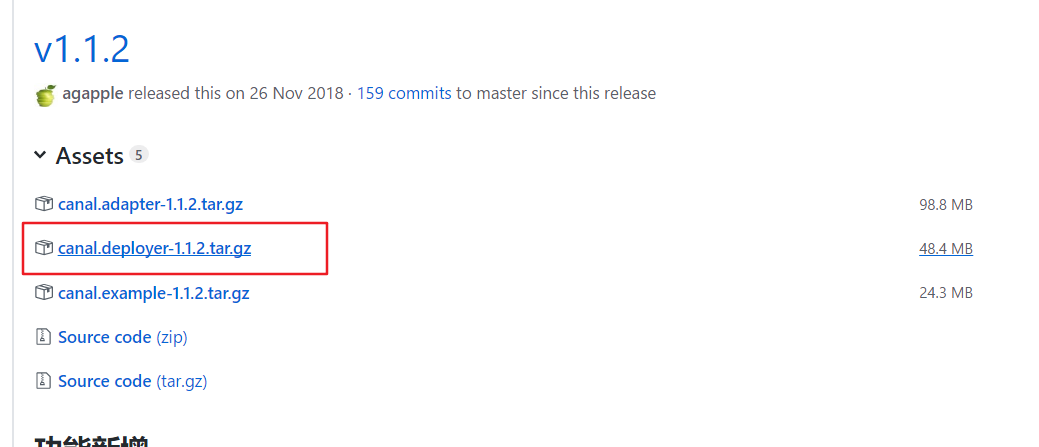
|  |
| --- |
| server-id= 1  log-bin=mysql-bin  binlog\_format=row  binlog-do-db=gmallXXXXX |

### 3.4 重启Mysql

## 4 canal 安装

### 4.1 canal的下载

https://github.com/alibaba/canal/releases

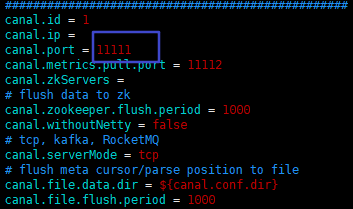


把canal.deployer-1.1.2.tar.gz拷贝到linux,解压缩

### 4.2 修改canal的配置

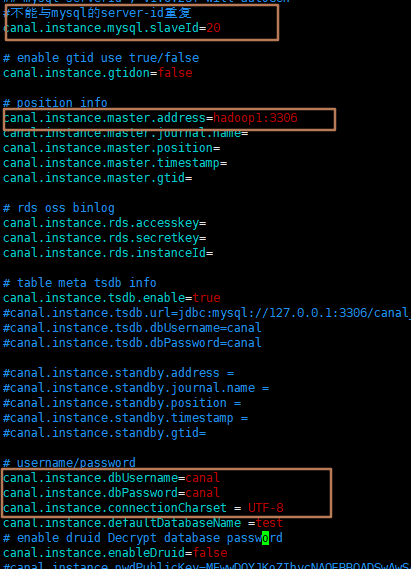
vim conf/canal.properties

这个文件是canal的基本通用配置，主要关心一下端口号，不改的话默认就是11111



vim conf/example/instance.properties

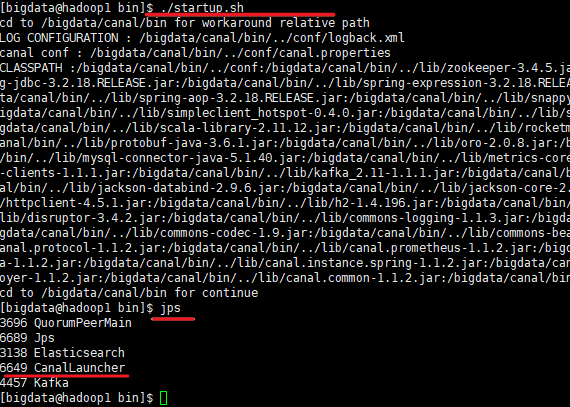
instance.properties是针对要追踪的mysql的实例配置



4.3 启动canal

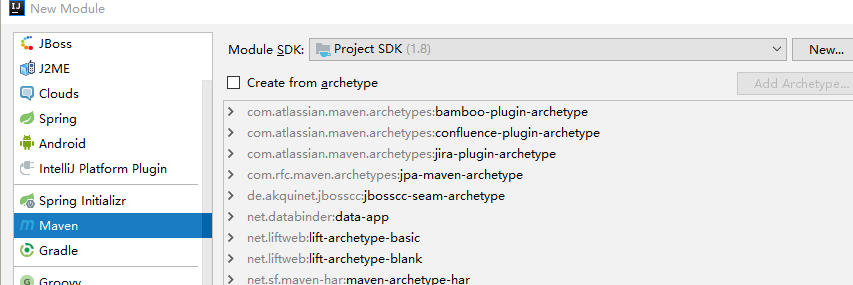
启动canal

|  |
| --- |
| ./bin/startup.sh |



## 5 数据监控模块---抓取订单数据

### 5.1 创建canal模块



### 5.2 pom.xml

|  |
| --- |
| <**dependencies**>  <**dependency**>  <**groupId**>com.bawei.gmall2019.dw</**groupId**>  <**artifactId**>dw-common</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>  </**dependency**>      *<!-- https://mvnrepository.com/artifact/com.alibaba.otter/canal.client -->* <**dependency**>  <**groupId**>com.alibaba.otter</**groupId**>  <**artifactId**>canal.client</**artifactId**>  <**version**>1.1.2</**version**>  </**dependency**>  *<!-- https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients -->* <**dependency**>  <**groupId**>org.apache.kafka</**groupId**>  <**artifactId**>kafka-clients</**artifactId**>  <**version**>0.11.0.0</**version**>  </**dependency**> </**dependencies**> |

### 5.3 通用监视类 CanalClient

|  |  |  |
| --- | --- | --- |
| **对象名称** | **介绍** | **包含内容** |
| **message** | 一次canal从日志中抓取的信息，一个message包含多个sql(event) | 包含 一个Entry集合 |
| **entry** | 对应一个sql命令，一个sql可能会对多行记录造成影响。 | 序列化的数据内容storeValue |
| **rowchange** | 是把entry中的storeValue反序列化的对象。 | 1 rowdatalist 行集  2 eventType(数据的变化类型 insert update delete create alter drop) |
| **RowData** | 出现变化的数据行信息 | 1 afterColumnList (修改后)  2 beforeColumnList（修改前） |
| **column:** | 一个RowData里包含了多个column，每个column包含了 name和 value | 1 columnName  2 columnValue |

|  |
| --- |
| **import** com.alibaba.otter.canal.client.CanalConnector; **import** com.alibaba.otter.canal.client.CanalConnectors; **import** com.alibaba.otter.canal.protocol.CanalEntry; **import** com.alibaba.otter.canal.protocol.Message; **import** com.bawei.gmall2019.canal.handler.CanalHandler; **import** com.google.protobuf.InvalidProtocolBufferException;  **import** java.net.InetSocketAddress; **import** java.util.List;  **public class** CanalClient {    **public void** watch(String hostname,**int** port,String destination ,String tables) {  *//连接* CanalConnector canalConnector = CanalConnectors.*newSingleConnector*(**new** InetSocketAddress(hostname, port), destination, **""**, **""**);  **while** (**true**){  canalConnector.connect();  canalConnector.subscribe(tables);  Message message = canalConnector.get(100);  **int** size = message.getEntries().size();   **if**(size==0){  System.***out***.println(**"没有数据！！休息5秒"**);   **try** {  Thread.*sleep*(5000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }   }**else**{  **for** (CanalEntry.Entry entry : message.getEntries()) {  **if**( entry.getEntryType()== CanalEntry.EntryType.***ROWDATA*** ){   CanalEntry.RowChange rowChange=**null**;  **try** {  rowChange = CanalEntry.RowChange.*parseFrom*(entry.getStoreValue());  } **catch** (InvalidProtocolBufferException e) {  e.printStackTrace();  }  String tableName = entry.getHeader().getTableName();*// 表名* CanalEntry.EventType eventType = rowChange.getEventType();*//insert update delete？* List<CanalEntry.RowData> rowDatasList = rowChange.getRowDatasList();*//行集 数据* CanalHandler canalHandler = **new** CanalHandler(eventType, tableName, rowDatasList);  canalHandler.handle() ;   }  }  }  }  }  } |

### 5.4 业务处理类： CanalHandler

|  |
| --- |
| **import** com.alibaba.fastjson.JSONObject; **import** com.alibaba.otter.canal.protocol.CanalEntry; **import** com.bawei.gmall2019.canal.util.MyKafkaSender; **import** com.bawei.gmall2019.common.constant.GmallConstant;      **public class** CanalHandler {   CanalEntry.EventType **eventType**;   String **tableName**;   List<CanalEntry.RowData> **rowDataList**;   **public** CanalHandler(CanalEntry.EventType eventType, String tableName, List<CanalEntry.RowData> rowDataList) {  **this**.**eventType** = eventType;  **this**.**tableName** = tableName;  **this**.**rowDataList** = rowDataList;  }   **public void** handle(){  *//下单操作* **if**(**"order\_info"**.equals(**tableName**)&& CanalEntry.EventType.***INSERT***==**eventType**){  rowDateList2Kafka( GmallConstant.***KAFKA\_TOPIC\_ORDER***);  }**else if** (**"user\_info"**.equals(**tableName**)&& (CanalEntry.EventType.***INSERT***==**eventType**||CanalEntry.EventType.***UPDATE***==**eventType**)) {  rowDateList2Kafka( GmallConstant.***KAFKA\_TOPIC\_USER***);  }   }    **private void** rowDateList2Kafka(String kafkaTopic){  **for** (CanalEntry.RowData rowData : **rowDataList**) {  List<CanalEntry.Column> columnsList = rowData.getAfterColumnsList();  JSONObject jsonObject = **new** JSONObject();  **for** (CanalEntry.Column column : columnsList) {   System.***out***.println(column.getName()+**"::::"**+column.getValue());  jsonObject.put(column.getName(),column.getValue());  }   MyKafkaSender.*send*(kafkaTopic,jsonObject.toJSONString());  }   } } |

5.5 kafka发送工具类： MykafkaSender

|  |
| --- |
| **public class** MyKafkaSender {   **public static** KafkaProducer<String, String> *kafkaProducer*=**null**;    **public static** KafkaProducer<String, String> createKafkaProducer(){  Properties properties = **new** Properties();  properties.put(**"bootstrap.servers"**, **"hadoop1:9092,hadoop2:9092,hadoop3:9092"**);  properties.put(**"key.serializer"**, **"org.apache.kafka.common.serialization.StringSerializer"**);  properties.put(**"value.serializer"**, **"org.apache.kafka.common.serialization.StringSerializer"**);  KafkaProducer<String, String> producer = **null**;  **try** {  producer = **new** KafkaProducer<String, String>(properties);   }**catch** (Exception e){  e.printStackTrace();  }  **return** producer;  }   **public static void** send(String topic,String msg){  **if**(*kafkaProducer*==**null**){  *kafkaProducer*=*createKafkaProducer*();  }  *kafkaProducer*.send(**new** ProducerRecord<String, String>(topic,msg));  } |

5 kafka客户端测试

|  |
| --- |
| /bigdata/kafka\_2.11-0.11.0.2/bin/kafka-console-consumer.sh --bootstrap-server hadoop1:9092,hadoop2:9092,hadoop3:9092 --topic **GMALL\_ORDER** |

# 实时处理

## 1 数据库建表

|  |
| --- |
| create table gmall2019\_order\_info  ( id varchar primary key ,  province\_id varchar,  consignee varchar,  order\_comment varchar,  consignee\_tel varchar,  order\_status varchar,  payment\_way varchar,  user\_id varchar,  img\_url varchar,  total\_amount double,  expire\_time varchar,  delivery\_address varchar,  create\_time varchar,  operate\_time varchar,  tracking\_no varchar,  parent\_order\_id varchar,  out\_trade\_no varchar,  trade\_body varchar,  create\_date varchar,  create\_hour varchar  ) |

## 2 case class

|  |
| --- |
| **case class** OrderInfo(  id: String,  province\_id: String,  consignee: String,  order\_comment: String,  **var** consignee\_tel: String,  order\_status: String,  payment\_way: String,  user\_id: String,  img\_url: String,  total\_amount: Double,  expire\_time: String,  delivery\_address: String,  create\_time: String,  operate\_time: String,  tracking\_no: String,  parent\_order\_id: String,  out\_trade\_no: String,  trade\_body: String,  **var** create\_date: String,  **var** create\_hour: String   ) |

## 3 sparkstreaming消费kafka并保持到Hbase中

|  |
| --- |
| **import** com.alibaba.fastjson.JSON **import** com.bawei.gmall2019.common.constant.GmallConstant **import** com.bawei.gmall2019.realtime.bean.OrderInfo **import** com.bawei.gmall2019.realtime.util.MyKafkaUtil **import** org.apache.hadoop.conf.Configuration **import** org.apache.kafka.clients.consumer.ConsumerRecord **import** org.apache.spark.SparkConf **import** org.apache.spark.streaming.dstream.{DStream, InputDStream} **import** org.apache.spark.streaming.{Seconds, StreamingContext}  **import** org.apache.phoenix.spark.\_  **object** OrderApp {   **def** main(args: Array[String]): Unit = {  **val** sparkConf: SparkConf = **new** SparkConf().setMaster(**"local[\*]"**).setAppName(**"order\_app"**)   **val** ssc = **new** StreamingContext(sparkConf,*Seconds*(5))   **val** inputDstream: InputDStream[ConsumerRecord[String, String]] = MyKafkaUtil.*getKafkaStream*(GmallConstant.*KAFKA\_TOPIC\_ORDER*,ssc)  *// inputDstream.map(\_.value()).foreachRDD(rdd=> // println(rdd.collect().mkString("\n")) // )  //* **val** orderInfoDstrearm: DStream[OrderInfo] = inputDstream.map {  \_.value()  }.map { orderJson =>  **val** orderInfo: OrderInfo = JSON.*parseObject*(orderJson, *classOf*[OrderInfo])  *//日期* **val** createTimeArr: Array[String] = orderInfo.create\_time.split(**" "**)  orderInfo.create\_date = createTimeArr(0)  **val** timeArr: Array[String] = createTimeArr(1).split(**":"**)  orderInfo.create\_hour = timeArr(0)  *// 收件人 电话 脱敏* orderInfo.consignee\_tel = **"\*\*\*\*\*\*\*"** + orderInfo.consignee\_tel.splitAt(7).\_2  orderInfo  }    orderInfoDstrearm.foreachRDD { rdd =>    **val** configuration = **new** Configuration()  *println*(rdd.collect().mkString(**"\n"**))  rdd.saveToPhoenix(**"GMALL2019\_ORDER\_INFO"**, *Seq*(**"ID"**,**"PROVINCE\_ID"**, **"CONSIGNEE"**, **"ORDER\_COMMENT"**, **"CONSIGNEE\_TEL"**, **"ORDER\_STATUS"**, **"PAYMENT\_WAY"**, **"USER\_ID"**,**"IMG\_URL"**, **"TOTAL\_AMOUNT"**, **"EXPIRE\_TIME"**, **"DELIVERY\_ADDRESS"**, **"CREATE\_TIME"**,**"OPERATE\_TIME"**,**"TRACKING\_NO"**,**"PARENT\_ORDER\_ID"**,**"OUT\_TRADE\_NO"**, **"TRADE\_BODY"**, **"CREATE\_DATE"**, **"CREATE\_HOUR"**), configuration, Some(**"hadoop1,hadoop2,hadoop3:2181"**))   }   ssc.start()  ssc.awaitTermination()    }  } |

# 第三章 数据接口发布

## 1 代码清单

|  |  |  |
| --- | --- | --- |
| 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 | OrderMapper | 数据层查询的interface |
| OrderMapper.xml | 数据层查询的实现配置 |

## 2 接口

### 2.1 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=order\_amount&date=2019-02-01 |

### 2.2 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 },  {"id":"order\_amount","name":"新增交易额","value":1000.2 }] |
| 分时统计 | {"yesterday":{"11":383,"12":123,"17":88,"19":200 },  "today":{"12":38,"13":1233,"17":123,"19":688 }} |

## 3 代码开发

### 3.1 OrderMapper

|  |
| --- |
| **import** java.util.List; **import** java.util.Map;  **public interface** OrderMapper {   *//1 查询当日交易额总数* **public** Double selectOrderAmountTotal(String date);   *//2 查询当日交易额分时明细* **public** List<Map> selectOrderAmountHourMap(String date);  } |

### 3.2 OrderMapper.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <!DOCTYPE mapper SYSTEM "http://mybatis.org/dtd/mybatis-3-mapper.dtd" *>* <mapper namespace="com.bawei.gmall0105.publisher.mapper.OrderMapper">  <select id="selectOrderAmountTotal" resultType="java.lang.Double">  select *sum*(total\_amount) sum\_amount from gmall0105\_order\_info where create\_date=#{date}  </select>   <select id="selectOrderAmountHourMap" resultMap="orderAmountHour" >  select create\_hour, *cast*(*sum*(total\_amount) as **double**) sum\_amount from gmall0105\_order\_info where create\_date=#{date} group by create\_hour  </select>  <resultMap id="orderAmountHour" type="java.util.Map" autoMapping="true">  </resultMap>  </mapper> |

### 3.3 PublisherService

|  |
| --- |
| **public** Double getOrderAmount(String date);  **public** Map getOrderAmountHour(String date); |

### 3.4 PublisherServiceImpl

|  |
| --- |
| @Override **public** Double getOrderAmount(String date) {  **return orderMapper**.selectOrderAmountTotal(date); }  @Override **public** Map getOrderAmountHour(String date) {  List<Map> mapList = **orderMapper**.selectOrderAmountHourMap(date);  Map orderAmountHourMap=**new** HashMap();  **for** (Map map : mapList) {  orderAmountHourMap.put(map.get(**"CREATE\_HOUR"**), map.get(**"SUM\_AMOUNT"**));  }  **return** orderAmountHourMap; } |

### 3.5 PublisherController

|  |
| --- |
| @GetMapping(**"realtime-total"**) **public** String getRealtimeTotal(@RequestParam(**"date"**) String date){  Long dauTotal = **publisherService**.getDauTotal(date);   List totalList =**new** ArrayList();   Map dauMap=**new** HashMap();  dauMap.put(**"id"**,**"dau"**);  dauMap.put(**"name"**,**"新增日活"**);  dauMap.put(**"value"**, dauTotal );  totalList.add(dauMap);   Map newMidMap=**new** HashMap();  newMidMap.put(**"id"**,**"newMid"**);  newMidMap.put(**"name"**,**"新增设备"**);  newMidMap.put(**"value"**, 233 );  totalList.add(newMidMap);   Map orderAmountMap=**new** HashMap();  orderAmountMap.put(**"id"**,**"order\_amount"**);  orderAmountMap.put(**"name"**,**"新增交易额"**);  Double orderAmount = **publisherService**.getOrderAmount(date);  orderAmountMap.put(**"value"**,orderAmount);  totalList.add(orderAmountMap);   **return** JSON.*toJSONString*(totalList) ; }  @GetMapping(**"realtime-hour"**) **public** String getRealtimeHour(@RequestParam(**"id"**)String id ,@RequestParam(**"date"**) String tdate){  **if**(**"dau"**.equals(id)){  Map hourMap=**new** HashMap();  Map dauHourTMap = **publisherService**.getDauHour(tdate);  String ydate = getYdate(tdate);  Map dauHourYMap = **publisherService**.getDauHour(ydate);  hourMap.put(**"yesterday"**,dauHourYMap);  hourMap.put(**"today"**,dauHourTMap);  **return** JSON.*toJSONString*(hourMap);  }**else if**(**"order\_amount"**.equals(id)){  Map hourMap=**new** HashMap();  Map orderHourTMap = **publisherService**.getOrderAmountHour(tdate);  String ydate = getYdate(tdate);  Map orderHourYMap = **publisherService**.getOrderAmountHour(ydate);  hourMap.put(**"yesterday"**,orderHourYMap);  hourMap.put(**"today"**,orderHourTMap);  **return** JSON.*toJSONString*(hourMap);   }  **return null**; } |

## 4 索引优化

|  |
| --- |
| create local index IDX\_GMALLXXX\_ORDER\_CREATE\_DATE\_HOUR on gmallXXX\_order\_info(create\_date,create\_hour ) |

## 5 对接可视化模块