尚硅谷电商数仓项目

--实时计算（ads层）

版本：V 1.0

张晨

# ADS 聚合层

ads层，主要是根据各种报表及可视化来生成统计数据。通常这些报表及可视化都是基于某些维度的汇总统计。

## 1 需求

热门商品统计（作业）

热门品类统计（作业）

热门品牌统计

交易用户性别对比（作业）

交易用户年龄段对比（作业）

交易额省市分布（作业）

## 2 分析

以热门商品统计为例

统计表分为三个部分

时间点、 维度 、 度量

时间点：即统计结果产生的时间，或者本批次数据中业务日期最早的时间。

维度：统计维度，比如地区、商品名称、性别

度量：汇总的数据，比如金额、数量

**每个批次进行一次聚合，根据数据的及时性要求，可以调整批次的时间长度。**

**聚合后的结果存放到数据库中。**

## 3 数据库的选型与难点

**聚合数据本身并不麻烦，利用reducebykey或者groupbykey都可以聚合。**

**但是麻烦的是实现精确性一次消费。**

**因为聚合数据不是明细，没有确定的主键，所以没有办法实现幂等。**

**那么如果想实现精确一次消费，就要考虑利用关系型数据库的事务处理。**

**用本地事务管理最大的问题是数据保存操作要放在driver端变成单线程操作。性能降低。 但是由于本业务保存的是聚合后的数据所以数据量并不大，即使单线程保存也是可以接受的。**

**因此数据库和偏移量选用mysql进行保存。**

## 4 代码实现

### 4.1 工具类

#### pom.xml 增加

|  |
| --- |
| <**dependency**>  <**groupId**>org.scalikejdbc</**groupId**>  <**artifactId**>scalikejdbc\_2.11</**artifactId**>  <**version**>2.5.0</**version**> </**dependency**> *<!-- scalikejdbc-config\_2.11 -->* <**dependency**>  <**groupId**>org.scalikejdbc</**groupId**>  <**artifactId**>scalikejdbc-config\_2.11</**artifactId**>  <**version**>2.5.0</**version**> </**dependency**> <**dependency**> <**groupId**>mysql</**groupId**> <**artifactId**>mysql-connector-java</**artifactId**> <**version**>5.1.47</**version**> </**dependency**> |

#### MysqlUtil 用于查询Mysql数据库

|  |
| --- |
| **import** java.sql.{Connection, DriverManager, ResultSet, ResultSetMetaData, Statement} **import** com.alibaba.fastjson.JSONObject **import** scala.collection.mutable.ListBuffer  **object** MysqlUtil {   **def** main(args: Array[String]): Unit = {  **val** list: List[ JSONObject] = *queryList*(**"select \* from offset\_2020"**)  *println*(list)  }   **def** queryList(sql:String):List[JSONObject]={  Class.*forName*(**"com.mysql.jdbc.Driver"**)  **val** resultList: ListBuffer[JSONObject] = **new** ListBuffer[ JSONObject]()  **val** conn: Connection = DriverManager.*getConnection*(**"jdbc:mysql://hadoop2:3306/gmall1122\_rs?characterEncoding=utf-8&useSSL=false"**,**"root"**,**"123123"**)  **val** stat: Statement = conn.createStatement  *println*(sql)  **val** rs: ResultSet = stat.executeQuery(sql )  **val** md: ResultSetMetaData = rs.getMetaData  **while** ( rs.next ) {  **val** rowData = **new** JSONObject();  **for** (i <-1 to md.getColumnCount ) {  rowData.put(md.getColumnName(i), rs.getObject(i))  }  resultList+=rowData  }   stat.close()  conn.close()  resultList.toList  }  } |

OffsetManagerM 用于查询Mysql数据库中的偏移量

|  |
| --- |
| **import** java.util  **import** com.alibaba.fastjson.JSONObject **import** org.apache.kafka.common.TopicPartition **import** redis.clients.jedis.Jedis  **object** OffsetManagerM {    */\*\*  \* 从Mysql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)   **val** offsetJsonObjList: List[JSONObject] = MysqlUtil.*queryList*(**"SELECT group\_id ,topic,partition\_id , topic\_offset FROM offset\_2020 where group\_id='"**+groupId+**"' and topic='"**+topic+**"'"**)   jedisClient.close()  **if**(offsetJsonObjList!=**null**&&offsetJsonObjList.size==0){  **null** }**else** {   **val** kafkaOffsetList: List[(TopicPartition, Long)] = offsetJsonObjList.map { offsetJsonObj =>  (**new** TopicPartition(offsetJsonObj.getString(**"topic"**),offsetJsonObj.getIntValue(**"partition\_id"**)), offsetJsonObj.getLongValue(**"topic\_offset"**))  }  kafkaOffsetList.toMap  }  }   } |

### 4.2 数据库准备

|  |
| --- |
| **创建专用保存数据结果的数据库**  create database gmall1122\_rs  **用于保存偏移量**  CREATE TABLE `offset\_1122` (  `group\_id` varchar(200) NOT NULL,  `topic` varchar(200) NOT NULL,  `partition\_id` int(11) NOT NULL,  `topic\_offset` bigint(20) DEFAULT NULL,  PRIMARY KEY (`group\_id`,`topic`,`partition\_id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8  **用户保存商品聚合结果**  CREATE TABLE `spu\_order\_final\_detail\_amount\_stat` ( stat\_time datetime ,spu\_id varchar(20) ,spu\_name varchar(200),amount decimal(16,2) ,  PRIMARY KEY (`stat\_time`,`spu\_id`,`spu\_name`)  )ENGINE=InnoDB DEFAULT CHARSET=utf8 |

### 4.3 实时计算代码

|  |
| --- |
| **import** java.text.SimpleDateFormat **import** java.util.Date  **import** com.alibaba.fastjson.JSON **import** com.atguigu.gmall1122.realtime.bean.OrderDetailWide **import** com.atguigu.gmall1122.realtime.util.{MyKafkaUtil, OffsetManager, OffsetManagerM} **import** org.apache.kafka.clients.consumer.ConsumerRecord **import** org.apache.kafka.common.TopicPartition **import** org.apache.spark.{SparkConf, rdd} **import** org.apache.spark.streaming.{Seconds, StreamingContext} **import** org.apache.spark.streaming.dstream.{DStream, InputDStream} **import** org.apache.spark.streaming.kafka010.{HasOffsetRanges, OffsetRange} **import** scalikejdbc.{DB, SQL} **import** scalikejdbc.config.DBs   **object** SpuAmountSumApp {    **def** main(args: Array[String]): Unit = {  **val** sparkConf: SparkConf = **new** SparkConf().setMaster(**"local[\*]"**).setAppName(**"ads\_spu\_amount\_sum\_app"**)   **val** ssc = **new** StreamingContext(sparkConf, *Seconds*(5))  **val** topic = **"DWS\_ORDER\_DETAIL\_WIDE"**;  **val** groupId = **"ads\_spu\_amount\_sum\_group"** *///////////////////// 偏移量处理///////////////////////////* **val** offset: Map[TopicPartition, Long] = OffsetManagerM.*getOffset*(groupId, topic)   **var** inputDstream: InputDStream[ConsumerRecord[String, String]] = **null** *// 判断如果从redis中读取当前最新偏移量 则用该偏移量加载kafka中的数据 否则直接用kafka读出默认最新的数据* **if** (offset != **null** && offset.size > 0) {  inputDstream = MyKafkaUtil.*getKafkaStream*(topic, ssc, offset, groupId)  } **else** {  inputDstream = MyKafkaUtil.*getKafkaStream*(topic, ssc, groupId)  }   *//取得偏移量步长* **var** offsetRanges: Array[OffsetRange] = **null  val** inputGetOffsetDstream: DStream[ConsumerRecord[String, String]] = inputDstream.transform { rdd =>  offsetRanges = rdd.asInstanceOf[HasOffsetRanges].offsetRanges  rdd  }   **val** orderDstreamDetailWideDstream: DStream[OrderDetailWide] = inputGetOffsetDstream.map { record =>  **val** jsonStr: String = record.value()  **val** orderDetailWide: OrderDetailWide = JSON.*parseObject*(jsonStr, *classOf*[OrderDetailWide])  orderDetailWide  }  **val** orderWideWithSpuDstream: DStream[(String, Double)] = orderDstreamDetailWideDstream.map(orderWide=>(orderWide.spu\_id+**":"**+orderWide.spu\_name,orderWide.final\_detail\_amount))   **val** spuAmountDstream: DStream[(String, Double)] = orderWideWithSpuDstream.reduceByKey(\_+\_)   spuAmountDstream.foreachRDD { rdd =>  **val** resultArr: Array[(String, Double)] = rdd.collect()  **if** (resultArr != **null** && resultArr.size > 0) {  DBs.setup()  DB.*localTx*(**implicit** session => {  **val** dateTime: String = **new** SimpleDateFormat(**"yyyy-MM-dd HH:mm:ss"**).format(**new** Date())  **for** ((spu, amount) <- resultArr) {  **val** spuArr: Array[String] = spu.split(**":"**)  **val** spuId: String = spuArr(0)  **val** spuName: String = spuArr(1)  *SQL*(**"INSERT INTO spu\_order\_final\_detail\_amount\_stat(stat\_time,spu\_id, spu\_name, amount) VALUES (?,?,?,?)"**).bind(dateTime, spuId, spuName, amount).update().apply()  }  **throw new** RuntimeException(**"测试异常！！"**)  **for** (offset <- offsetRanges) {  *//主键相同替换 主键不同新增  SQL*(**"replace INTO offset\_2020(group\_id,topic, partition\_id, topic\_offset) VALUES (?,?,?,?)"**).bind(groupId, topic, offset.partition, offset.untilOffset).update().apply()  }   }   )  }  }  ssc.start()  ssc.awaitTermination()  }  } |

### 4.4 关于本地事务保存MySql

**此处引用了一个 scala的MySQL工具：scalikeJdbc**

**配置文件： 默认使用 application.conf**

|  |
| --- |
| db.default.driver="com.mysql.jdbc.Driver" db.default.url="jdbc:mysql://hadoop2/gmall1122\_rs?characterEncoding=utf-8&useSSL=false" db.default.user="root" db.default.password="123123" |

**加载配置**

|  |
| --- |
| DBs.setup() |

本地事务提交数据

|  |
| --- |
| DB.*localTx*(**implicit** session => {  *SQL*(**"INSERT INTO spu\_order\_final\_detail\_amount\_stat(stat\_time,spu\_id, spu\_name, amount) VALUES (?,?,?,?)"**).bind(dateTime, spuId, spuName, amount).update().apply()    *SQL*(**"replace INTO offset\_2020(group\_id,topic, partition\_id, topic\_offset) VALUES (?,?,?,?)"**).bind(groupId, topic, offset.partition, offset.untilOffset).update().apply()    } ) |

凡是在 DB.*localTx*(**implicit** session => { } )中的SQL全部被本地事务进行关联，一条失败全部回滚。

# 发布接口

发布接口的目的是为可视化工具提供数据服务。

发布接口的地址和参数都要根据可视化工具的要求进行设置。

后面的可视化工具选用了阿里云服务的DataV，由于DataV对地址没有要求(可以自行配置)，只对返回数据格式有一定要求。最好可以提前了解一下数据格式的要求。

或者可以不考虑接口格式，先完成service的查询，然后再controller针对不同的格式要求在进行调整。

## 1 配置文件

### 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>      <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-jdbc</artifactId> </dependency>    <dependency> <groupId>mysql</groupId> <artifactId>mysql-connector-java</artifactId> <version>5.1.47</version>  </dependency>  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.3.4</version> </dependency>    </dependencies>   <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |
|  |

### application.properties

|  |
| --- |
| server.port=8070    logging.level.root=error    spring.datasource.driver-class-name= **com.mysql.jdbc.Driver**  spring.datasource.url= **jdbc:mysql://hadoop2/gmall1122\_rs?characterEncoding=utf-8&useSSL=false**  spring.datasource.data-username=root spring.datasource.data-password=123123  *# mybatis* mybatis.mapperLocations=classpath:mapper/\*.xml mybatis.configuration.map-underscore-to-camel-case=true |

**其中mybatis.mapperLocations的作用：能够让spring容器找到mapper.xml，用于和mapper接口进行配对。**

## 5 代码部分

|  |  |  |
| --- | --- | --- |
| 控制层 | PublisherController | 实现接口的web发布 |
| 服务层 | MySQLService | 数据业务查询interface |
| MySQLServiceImpl | 业务查询的实现类 |
| 数据层 | TrademarkAmountSumMapper | 数据层查询的interface |
| TrademarkAmountSum.xml | 数据层查询的实现配置,实质上是Mapper接口的“实现类”。 |
| 主程序 | GmallPublisherApplication | 增加扫描包 |

### 5.1 GmallPublisherApplication 增加扫描包

**作用：能够让spring容器找到mapper接口用于和mapper.xml进行配对**

|  |
| --- |
| @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

|  |
| --- |
| **package** com.atguigu.gmall1122.publisher.controller;   **import** com.alibaba.fastjson.JSON; **import** com.atguigu.gmall1122.publisher.service.MysqlService; **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.util.List; **import** java.util.Map;  @RestController **public class** DataVController {   *//路径和参数随便定 ，但是返回值要看datav的需要* @Autowired  MysqlService **mysqlService**;   @GetMapping(**"trademark-sum"**)  **public** String trademarkSum(@RequestParam(**"start\_date"**) String startDate, @RequestParam(**"end\_date"**) String endDate){  **if**(startDate.length()==0|| endDate.length()==0){  **return "参数不能为空！"**;  }  startDate = startDate.replace(**"\_"**, **" "**);  endDate = endDate.replace(**"\_"**, **" "**);  List<Map> trademardSum = **mysqlService**.getTrademardSum(startDate, endDate);   **return** JSON.*toJSONString*(trademardSum) ;    }    } |

### 5.3 service层

|  |
| --- |
| **public interface** MysqlService {    **public** List<Map> getTrademardSum(String startDate,String endDate); } |

### 5.4 service层实现类

|  |
| --- |
| @Service **public class** MysqlServiceImpl **implements** MysqlService {   @Autowired  TrademarkAmountSumMapper **trademarkAmountSumMapper**;   @Override  **public** List<Map> getTrademardSum(String startDate, String endDate) {  **return trademarkAmountSumMapper**.selectTradeSum(startDate,endDate);  } } |

### 5.5 数据层 mapper

|  |
| --- |
| **public interface** TrademarkAmountSumMapper {   **public** List<Map> selectTradeSum(@Param(**"start\_Date"**) String startDate , @Param(**"end\_Date"**)String endDate); } |

### 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.atguigu.gmall1122.publisher.mapper.TrademarkAmountSumMapper">  <select id="selectTradeSum" resultMap="trademarkSumMap">  select tm\_name ,*sum*(amount) amount from `trademark\_amount\_sum\_stat`  where stat\_time &gt;= #{start\_Date} and stat\_time &lt; #{end\_Date}  group by tm\_id ,tm\_name  order by *sum*(amount) desc  </select>   <resultMap id="trademarkSumMap" type="java.util.Map" autoMapping="true">  </resultMap>  </mapper> |

# 可视化

## 1 DataV

**阿里云网址**： <https://datav.aliyun.com/>

**官方帮助手册：** <https://help.aliyun.com/document_detail/30360.html>

阿里云有两大数据可视化服务，一个是QuickBI，一个就是DataV。

QuickBI定位BI工具定位由数据分析师使用，通过灵活配置各种多维分析、深度钻取，生成各种报表和可交互的图形化展示。

而DataV ,倾向于定制数据大屏，针对运营团队使用的信息丰富炫酷的监控型可视化工具。

## 2 数据源

DataV的数据源主要是两方面，阿里云数据服务体系内的数据源和外部数据源。

阿里云数据服务体系内的数据源，类型非常多，包括RDS服务，ADS服务，TableStore服务等等。

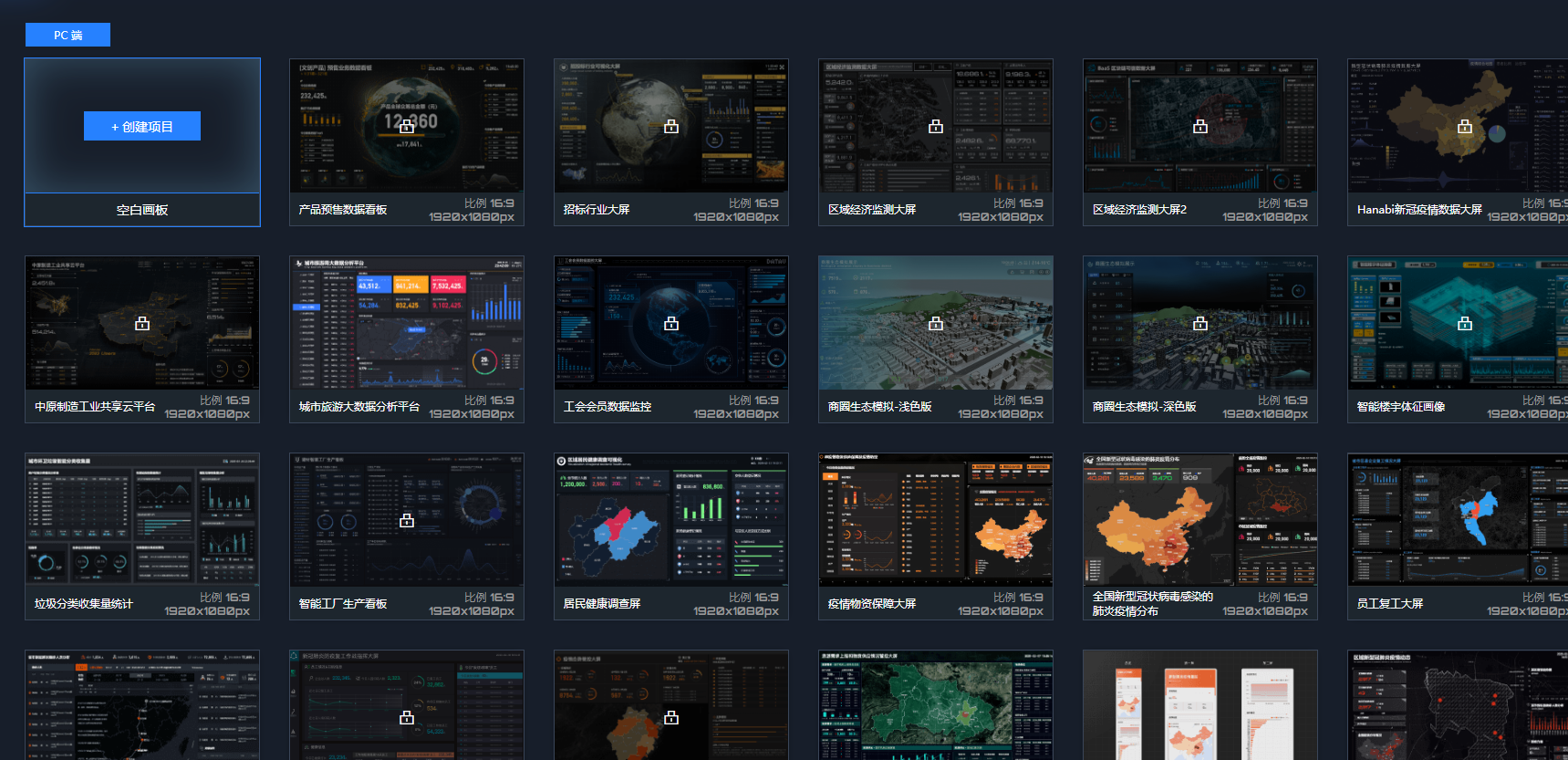
本文只介绍基于外部数据源的配置方式。 外部数据源就是要发布出可以外网访问的地址，每一个可视化组件都要对应一个访问地址。

## 3 配置步骤

### 3.1 首先来到首页，在我的可视化标签中，选择【新建可视化】



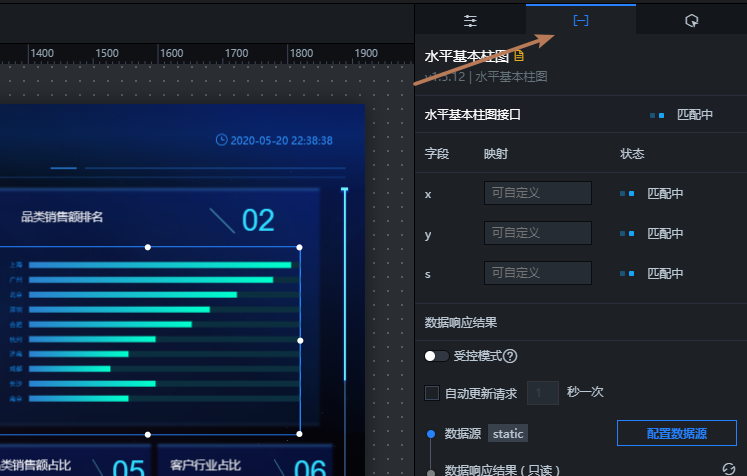
### 2 选择合适的大屏模板



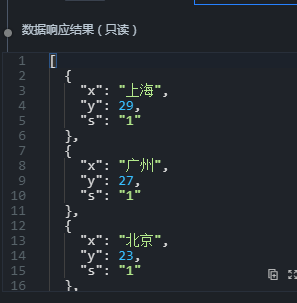
### 3 选中你要配置的组件



### 4 选择左侧中间的标签页



### 5 观察左侧下方的静态数据，实现对应的接口程序

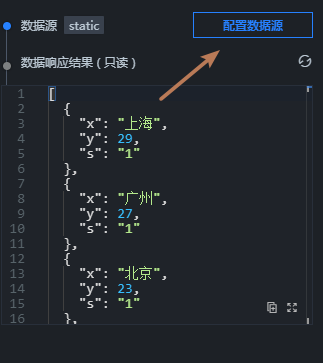


### 6 根据数据结构调整web接口的响应数据

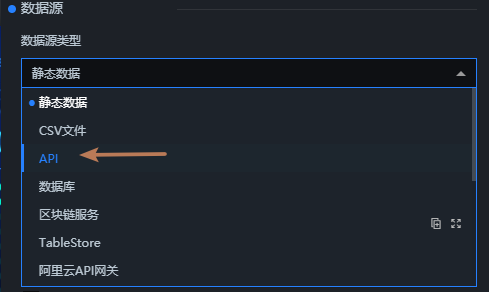
|  |
| --- |
| @GetMapping(**"trademark-sum"**) **public** String trademarkSum(@RequestParam(**"start\_date"**) String startDate, @RequestParam(**"end\_date"**) String endDate){  **if**(startDate.length()==0|| endDate.length()==0){  **return "参数不能为空！"**;  }  startDate = startDate.replace(**"\_"**, **" "**);  endDate = endDate.replace(**"\_"**, **" "**);  List<Map> trademarkSumList = **mysqlService**.getTrademarkSum(startDate, endDate);   *//根据DataV图形数据要求进行调整， x :品牌 ,y 金额， s 1* List<Map> datavList=**new** ArrayList<>();  **for** (Map trademardSumMap : trademarkSumList) {  Map map = **new** HashMap<>();  map.put(**"x"**,trademardSumMap.get(**"tm\_name"**));  map.put(**"y"**,trademardSumMap.get(**"amount"**));  map.put(**"s"**,1);  datavList.add(map);  }   **return** JSON.*toJSONString*(datavList) ;  } |

### 7 配置组件的数据源

点击【配置数据源】



数据源的页面选择API



选择API后，填写URL，数据接口访问路径

**（实现个人电脑发布服务需要内网穿透，请参考第二章）**



调整自动更新速度



下方能看到数据发生变化



可视化效果



# 内网穿透

## 1 作用

通常个人电脑无论是连接WIFI上网还是用网线上网，都是属于局域网里边的，外网无法直接访问到你的电脑，内网穿透可以让你的局域网中的电脑实现被外网访问功能。

## 2 工具

目前国内网穿透工具很多，常见的比如花生壳、Ngrok。

官网：

花生壳：<https://hsk.oray.com>

Ngrok: <http://www.ngrok.cc>

本文以介绍**花生壳**为主

## 3 准备工作

首先注册、登录

并且需要实名认证（要提供身份证正反面照片）



4 下载安装电脑客户端

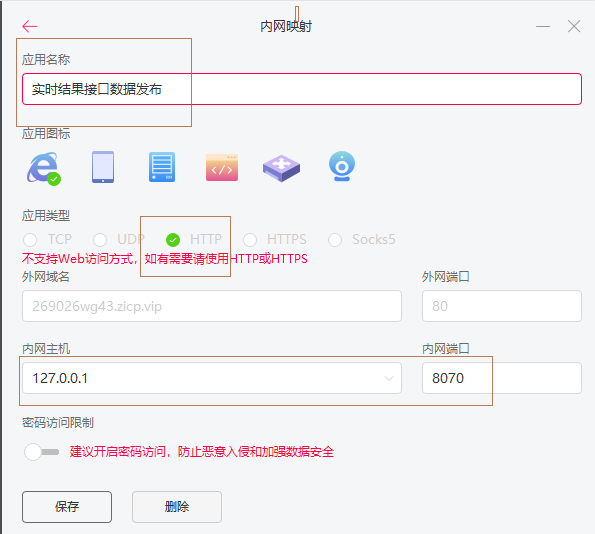
5 在客户端进行配置

在登录后的界面



在右下角点击加号

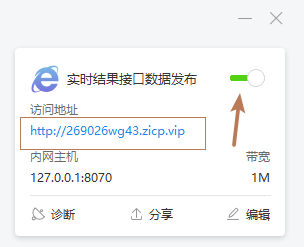
6 进行内网穿透的核心配置



7 发布

保存后就可以用下方图中箭头处使用开关来确认发布。

发布地址就如图中网址



8 测试：

