电商数仓项目--实时计算（预警）

# 需求分析

## 1 简介

**实时预警**，是一种经常出现在实时计算中的业务类型。根据日志数据中系统报错异常，或者用户行为异常的检测，产生对应预警日志。 预警日志通过图形化界面的展示，可以提醒监控方，需要及时核查问题，并采取应对措施。

## 2 需求说明：

**需求：同一设备，5分钟内三次及以上用不同账号登录并领取优惠劵，并且在登录到领劵过程中没有浏览商品。达到以上要求则产生一条预警日志。**

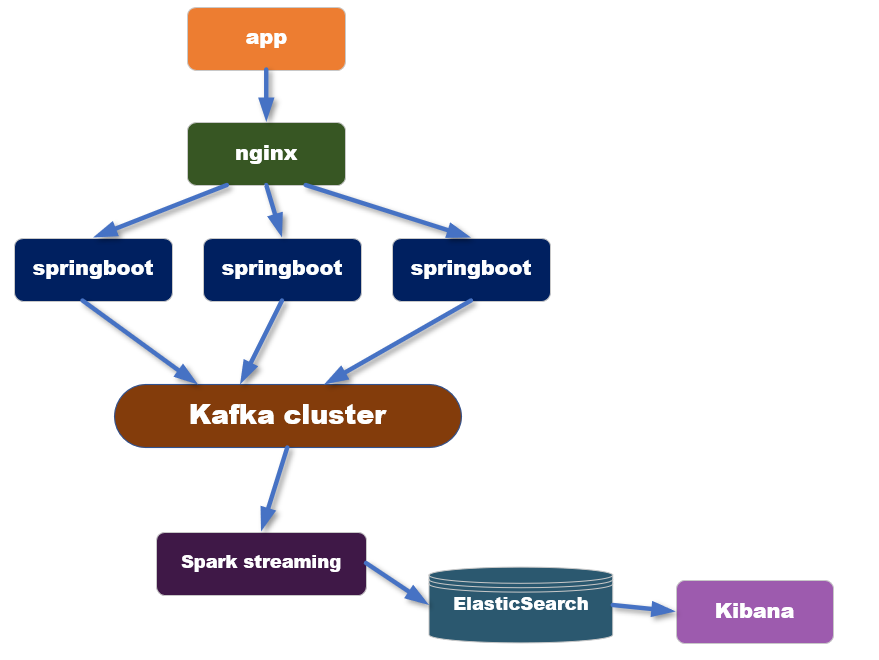
**同一设备，每分钟只记录一次预警。**

## 3 预警日志格式

|  |  |
| --- | --- |
| **mid** | **设备id** |
| **uids** | 领取优惠券登录过的uid |
| **itemIds** | 优惠券涉及的商品id |
| **events** | 发生过的行为 |
| **ts** | 发生预警的时间戳 |

# 第二章 整体流程设计

## 1 框架流程



## 2 开发思路

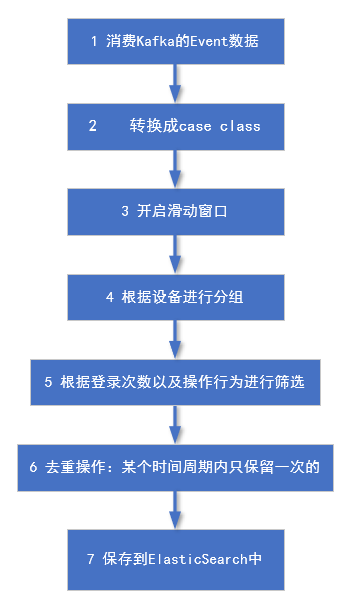
1. 从kafka中消费数据，根据条件进行过滤筛选，生成预警日志。
2. 预警日志保存到ElasticSearch中
3. 利用Kibana 快速搭建可视化图形界面

# 第三章 实时计算模块

## 1 、筛选条件分析

* + 同一设备
  + 5分钟内
  + 三次不同账号登录
  + 领取优惠券
  + 没有浏览商品
  + 同一设备每分钟只记录一次预警

## 2 、 数据处理流程图



## 3 代码开发

### 3.1 增加事件日志样例类 -- EventInfo

|  |
| --- |
| **case class** EventInfo(mid:String,  uid:String,  appid:String,  area:String,  os:String,  ch:String,  `type`:String,  evid:String ,  pgid:String ,  npgid:String ,  itemid:String,  **var** logDate:String,  **var** logHour:String,  **var** ts:Long  ) |

### 3.2 增加预警样例类 -- CouponAlertInfo

|  |
| --- |
| **case class** CouponAlertInfo(mid:String,  uids:java.util.HashSet[String],  itemIds:java.util.HashSet[String],  events:java.util.List[String],  ts:Long) {  } |

### 3.3 预警业务类

|  |
| --- |
| **import** com.alibaba.fastjson.JSON **import** com.bawei.gmall.constant.GmallConstants **import** com.bawei.gmall2019.realtime.bean.{CouponAlertInfo, EventInfo} **import** com.bawei.gmall2019.realtime.util.{MyEsUtil, MyKafkaUtil} **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** scala.util.control.Breaks.\_  **object** AlertApp {   **def** main(args: Array[String]): Unit = {  **val** sparkConf: SparkConf = **new** SparkConf().setMaster(**"local[\*]"**).setAppName(**"event\_app"**)   **val** ssc = **new** StreamingContext(sparkConf,*Seconds*(5))   **val** inputDstream: InputDStream[ConsumerRecord[String, String]] = MyKafkaUtil.*getKafkaStream*(GmallConstants.*KAFKA\_TOPIC\_EVENT*,ssc)   *//1 格式转换成样例类* **val** eventInfoDstream: DStream[EventInfo] = inputDstream.map { record =>  **val** jsonstr: String = record.value()  **val** eventInfo: EventInfo = JSON.*parseObject*(jsonstr, *classOf*[EventInfo])  eventInfo  }  *//2 开窗口* **val** eventInfoWindowDstream: DStream[EventInfo] = eventInfoDstream.window(*Seconds*(30),*Seconds*(5))   *//3同一设备 分组* **val** groupbyMidDstream: DStream[(String, Iterable[EventInfo])] = eventInfoWindowDstream.map(eventInfo=>(eventInfo.mid,eventInfo)).groupByKey()   *//4 判断预警 // 在一个设备之内 // 1 三次及以上的领取优惠券 (evid coupon) 且 uid都不相同 // 2 没有浏览商品(evid clickItem)* **val** checkCouponAlertDStream: DStream[(Boolean, CouponAlertInfo)] = groupbyMidDstream.map { **case** (mid, eventInfoItr) =>  **val** couponUidsSet = **new** util.HashSet[String]()  **val** itemIdsSet = **new** util.HashSet[String]()  **val** eventIds = **new** util.ArrayList[String]()  **var** notClickItem: Boolean = **true** breakable(  **for** (eventInfo: EventInfo <- eventInfoItr) {  eventIds.add(eventInfo.evid) *//用户行为* **if** (eventInfo.evid == **"coupon"**) {  couponUidsSet.add(eventInfo.uid) *//用户领券的uid* itemIdsSet.add(eventInfo.itemid) *//用户领券的商品id* } **else if** (eventInfo.evid == **"clickItem"**) {  notClickItem = **false** break()  }  }  )  *//组合成元组 （标识是否达到预警要求，预警信息对象）* (couponUidsSet.size() >= 3 && notClickItem, *CouponAlertInfo*(mid, couponUidsSet, itemIdsSet, eventIds, System.*currentTimeMillis*()))  }   *//过滤* **val** filteredDstream: DStream[(Boolean, CouponAlertInfo)] = checkCouponAlertDStream.filter{\_.\_1}    *//增加一个id 用于保存到es的时候进行去重操作* **val** alertInfoWithIdDstream: DStream[(String, CouponAlertInfo)] = filteredDstream.map { **case** (flag, alertInfo) =>  **val** period: Long = alertInfo.ts / 1000L / 60L  **val** id: String = alertInfo.mid + **"\_"** + period.toString  (id, alertInfo)   }    alertInfoWithIdDstream.foreachRDD{rdd=>  rdd.foreachPartition{alertInfoWithIdIter=>  MyEsUtil.*insertBulk*( GmallConstants.*ES\_INDEX\_COUPON\_ALERT* ,alertInfoWithIdIter.toList)   }  }   ssc.start()  ssc.awaitTermination()   }    } |

# 第四章 ElasticSearch 的保存

## 1 ES 集群搭建

参考《ElasticSearch集群安装》手册

## 2 在ES 上建好索引

其实即使不提前建立索引，ES也是可以将数据保存进去的。这种情况，ES会根据第一条要插入的数据进行推断，但是ES的这种推断往往不够准确。

比如：要区分字段要不要进行索引，字段要不要进行分词，如果分词选用哪个分词器等等。

建立索引语句（包含Mapping）

|  |
| --- |
| PUT gmall\_coupon\_alert  {  "mappings": {  "\_doc":{  "properties":{  "mid":{  "type":"keyword"  },  "uids":{  "type":"keyword"  },  "itemIds":{  "type":"keyword"  },  "events":{  "type":"keyword"  },  "ts":{  "type":"date"  }  }  }  }  } |

## 3 保存ES

### 3.1 pom.xml 增加依赖

|  |
| --- |
| <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> |

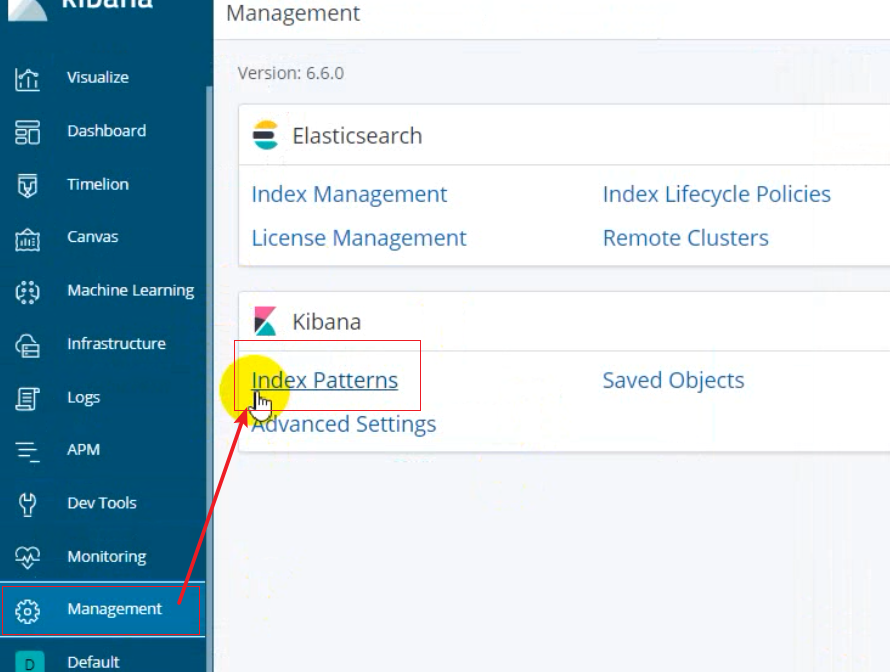
### 3.2 保存ES的工具类

|  |
| --- |
| **import** java.util **import** java.util.Objects  **import** io.searchbox.client.{JestClient, JestClientFactory} **import** io.searchbox.client.config.HttpClientConfig **import** io.searchbox.core.{Bulk, BulkResult, Index} **import** collection.JavaConversions.\_  **object** MyEsUtil {  **private val** *ES\_HOST* = **"http://hadoop1"  private val** *ES\_HTTP\_PORT* = 9200  **private var** *factory*:JestClientFactory = **null** */\*\*  \* 获取客户端  \*  \** **@return** *jestclient  \*/* **def** getClient: JestClient = {  **if** (*factory* == **null**) *build*()  *factory*.getObject  }   */\*\*  \* 关闭客户端  \*/* **def** close(client: JestClient): Unit = {  **if** (!Objects.*isNull*(client)) **try** client.shutdownClient()  **catch** {  **case** e: Exception =>  e.printStackTrace()  }  }   */\*\*  \* 建立连接  \*/* **private def** build(): Unit = {  *factory* = **new** JestClientFactory  *factory*.setHttpClientConfig(**new** HttpClientConfig.Builder(*ES\_HOST* + **":"** + *ES\_HTTP\_PORT*).multiThreaded(**true**)  .maxTotalConnection(20) *//连接总数* .connTimeout(10000).readTimeout(10000).build)   }  *// 批量插入数据到ES* **def** insertBulk(indexName:String,docList:List[(String,Any)]): Unit ={  **if**(docList.size>0){  **val** jest: JestClient = *getClient* **val** bulkBuilder = **new** Bulk.Builder().defaultIndex(indexName).defaultType(**"\_doc"**)  **for** ((id,doc) <- docList ) {  **val** indexBuilder = **new** Index.Builder(doc)  **if**(id!=**null**){  indexBuilder.id(id)  }  **val** index: Index = indexBuilder.build()  bulkBuilder.addAction(index)  }  **val** bulk: Bulk = bulkBuilder.build()  **var** items: util.List[BulkResult#BulkResultItem] = **null  try** {  items = jest.execute(bulkBuilder.build()).getItems   } **catch** {  **case** ex: Exception => *println*(ex.toString)  } **finally** {  *close*(jest)  *println*(**"保存"** + items.size() + **"条数据"**)  **for** (item <- items) {  **if** (item.*error* != **null** && item.*error*.nonEmpty) {  *println*(item.*error*)  *println*(item.*errorReason*)  }  }   }  }  }     **def** main(args: Array[String]): Unit = {  **val** jest: JestClient = *getClient* **val** index: Index = **new** Index.Builder(Stud(**"zhang3"**,**"zhang33"**)).index(**"gmall2019\_stud"**).`type`(**"\_doc"**).id(**"stu123"**).build()  jest.execute(index)   }    **case class** Stud(name:String ,nickname:String){   } |

# 第五章 Kibana发布可视化界面

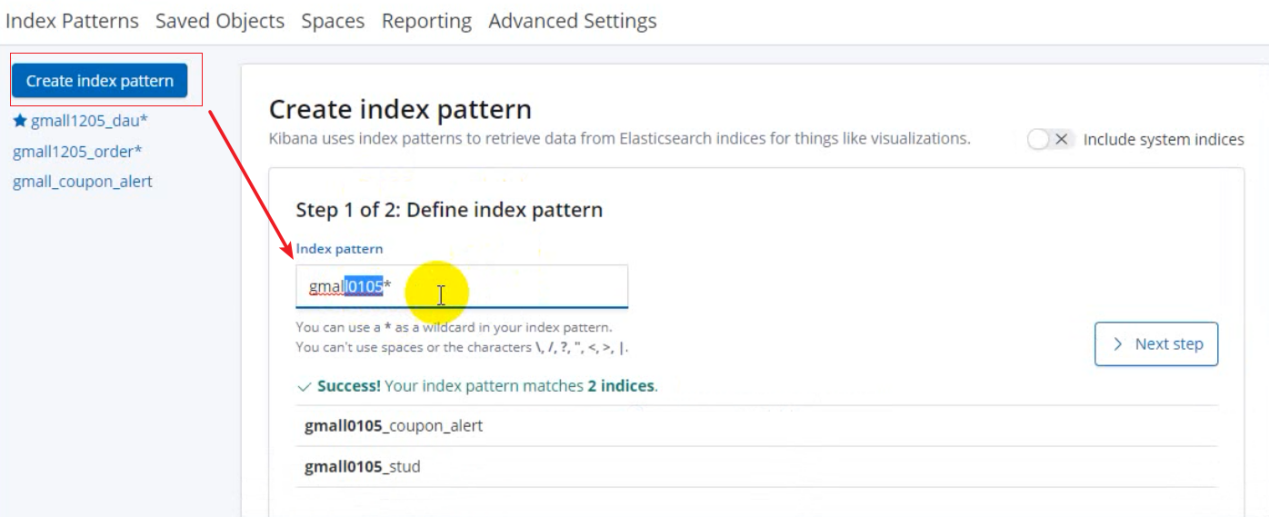
## 1 建立index pattern

建立数据源表达式



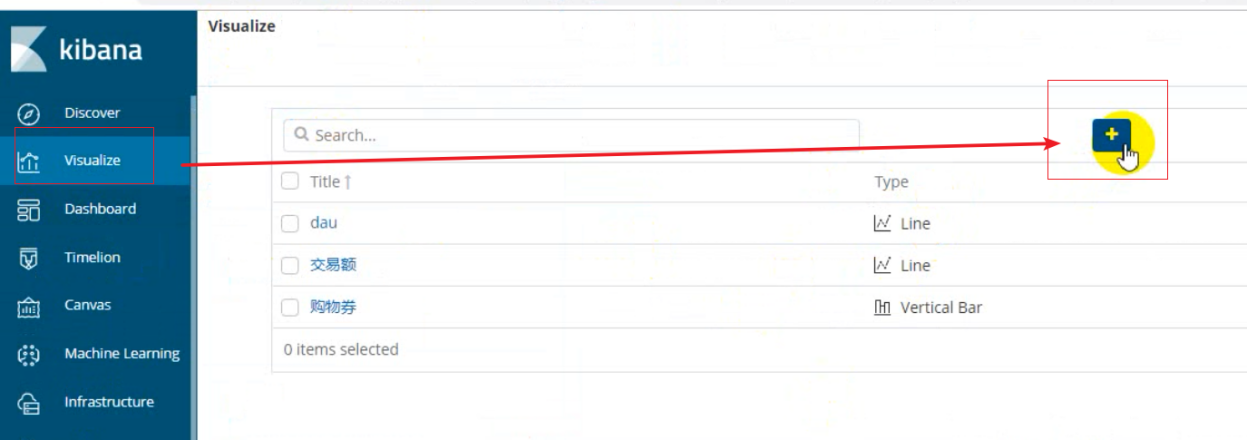
利用通配符（\*），可以覆盖多个索引，比如order\_index\_2019\_05\*

表示覆盖order\_index\_2019\_05\_01, order\_index\_2019\_05\_02 等多个索引。



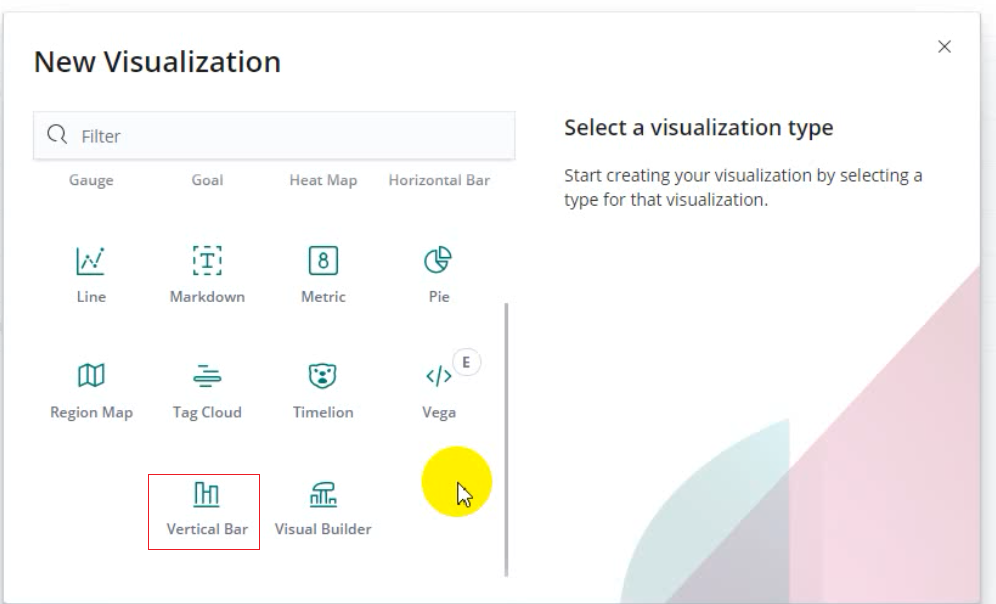
## 2 建立 visualize

### 2.1 新增一个可视化图



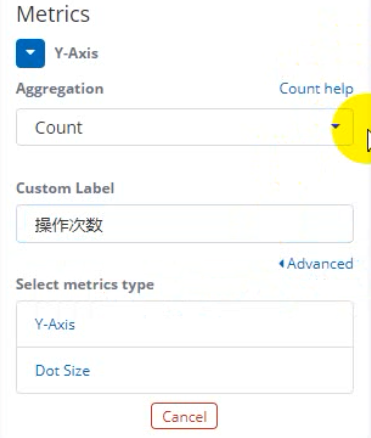
### 2.2 选择一个图形类型，

本案例选择柱形图（Vertical Bar ）



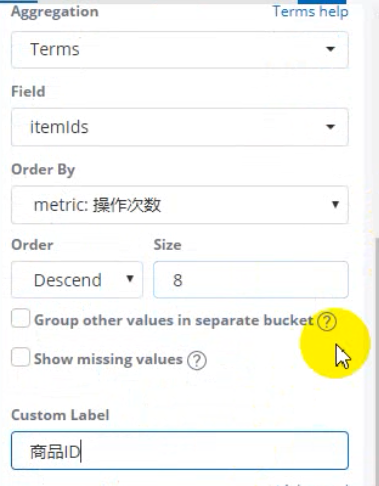
### 2.3 作图

#### 2.3.1 纵坐标



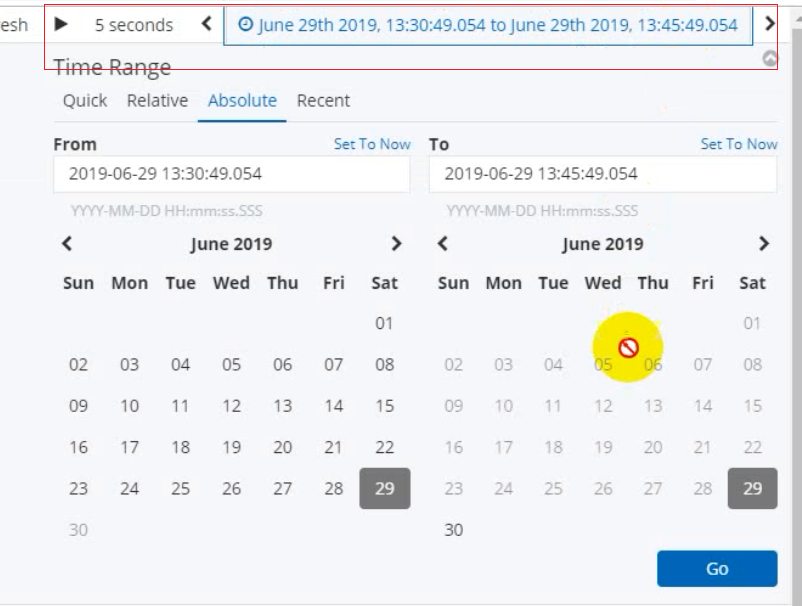
|  |  |
| --- | --- |
| **Aggregation** | **聚合方法** |
| **Customer Lable** | 纵坐标的说明标签 |

#### 2.3.2 横坐标

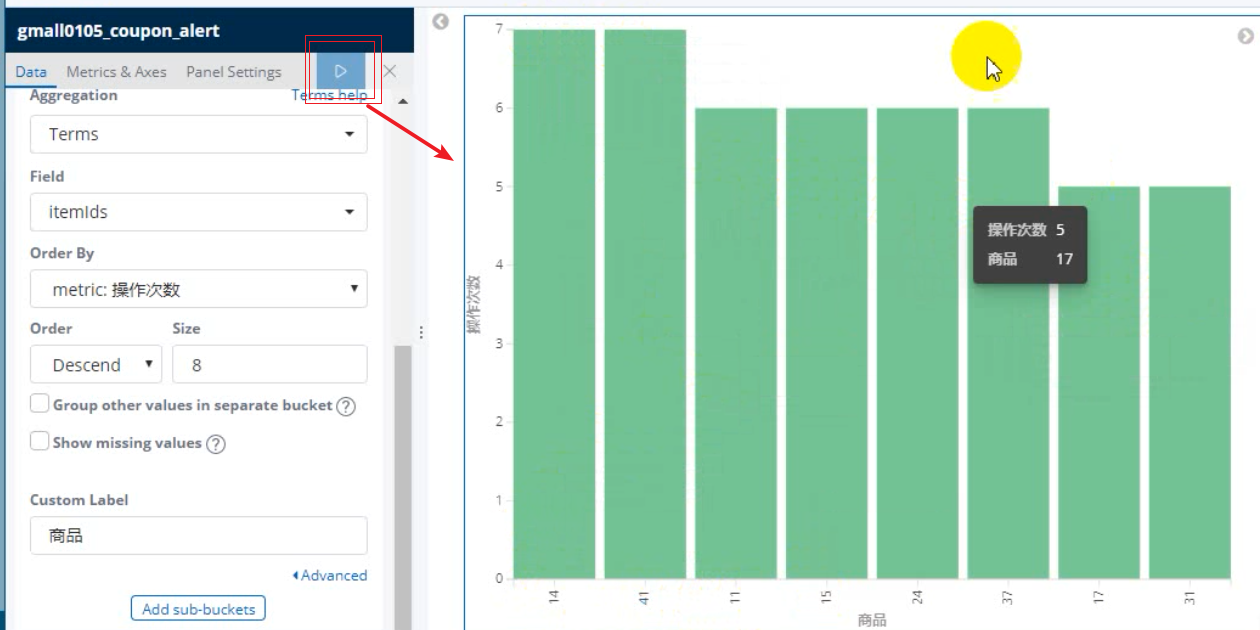


|  |  |
| --- | --- |
| **Aggregation** | **分组字段** |
| **Field** | 分组字段 |
| **ORDER by** | 排序方式 |
| **Order** | 升序降序 |
| **Size** | 列出前n名 |
| **Customer Lable** | 纵坐标的说明标签 |

#### 2.3.3 右上角选择时间范围和刷新间隔



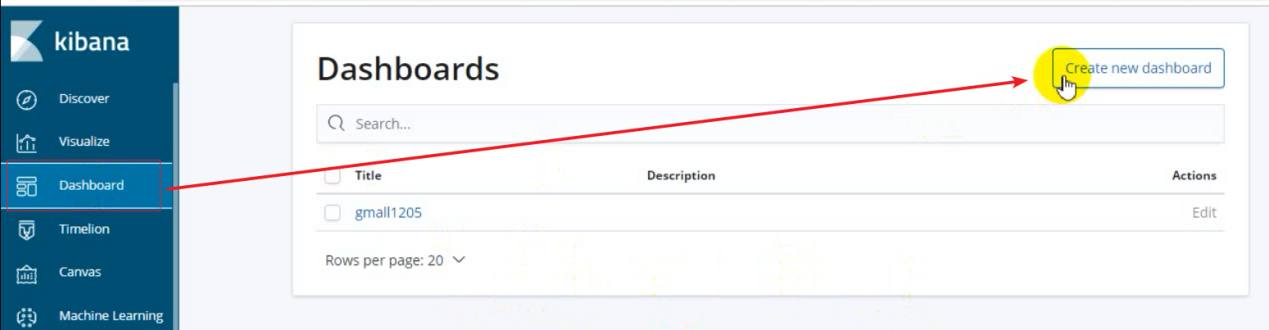
运行产生效果图



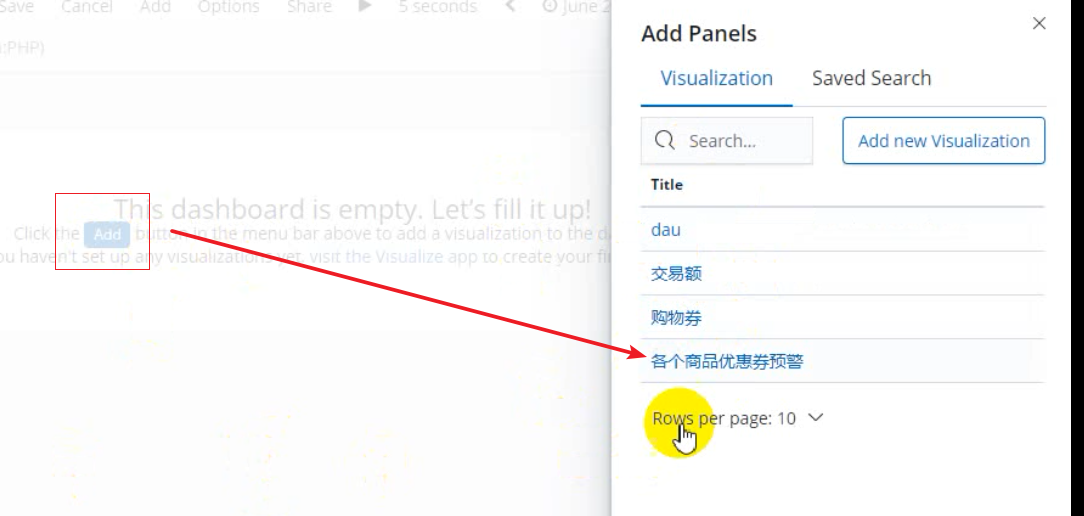
## 3 建立 Dashboard

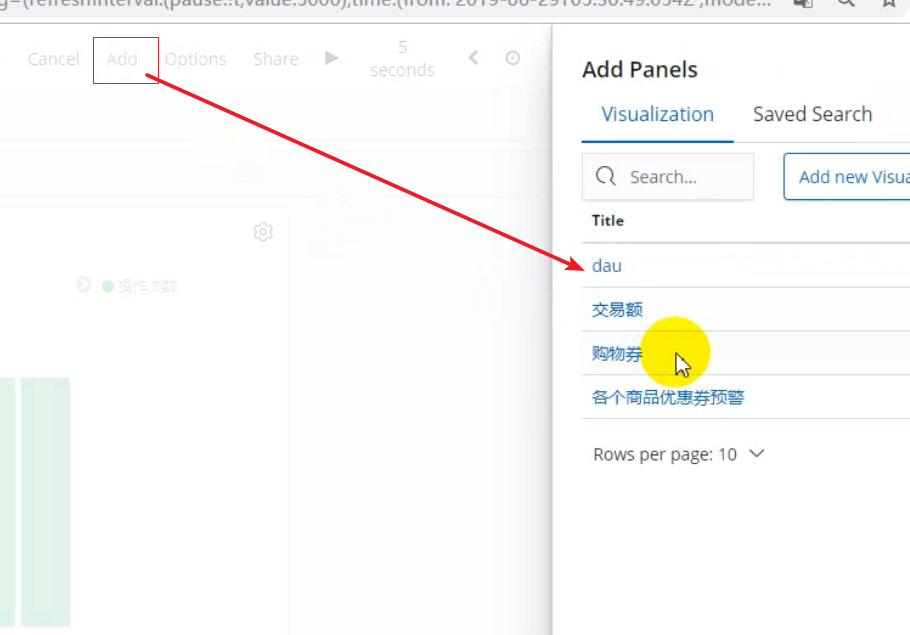
Dashboard是一个可以放很多个可视化图的大仪表盘，你可以把之前设计好的多个可视化图，放置在一个仪表盘中，一起显示。

### 3.1 新增一个Dashboard



### 3.2 加入多个可视化图

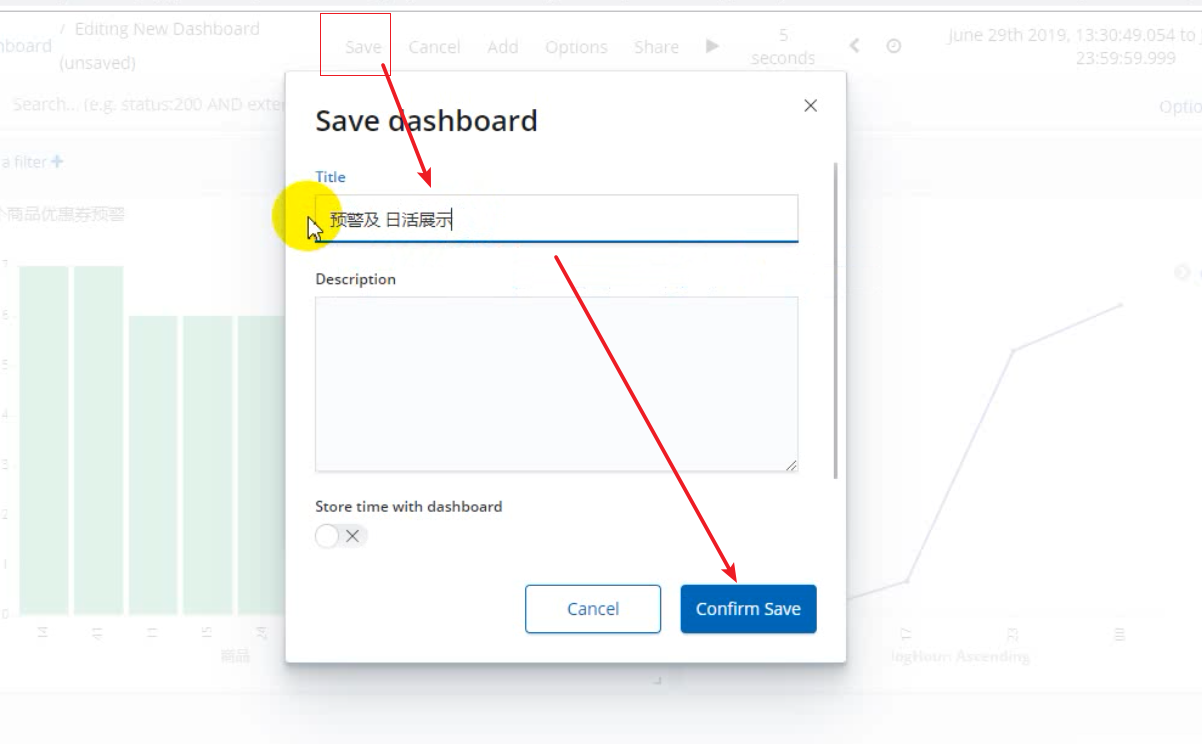




### 3.3 形成一个含多个图的仪表盘

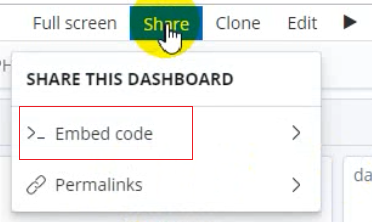
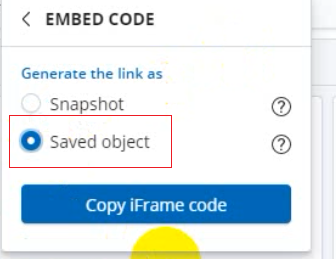


### 3.4 最后保存

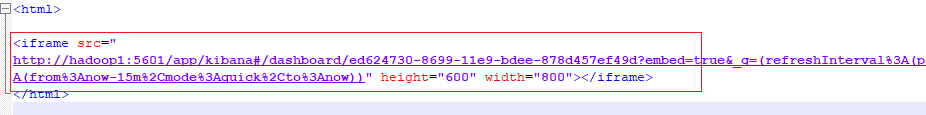


## 4 分享到网页中

点击最上方的share按钮

可以吧剪切板中的iframe代码嵌入到网页代码中



这样就可以在一张网页中显示kibana中的仪表盘

