

本科毕业论文

Locez

<https://locez.com>

摘 要

数据信息化步伐越来越快，现代社会几乎人们所有的操作都会在互联网以及现代终端设备上进行，在这些操作背后往往会产生海量的数据，而这些数据对企业来说绝对不单单只是一个流水账。大数据技术的发展使得人们具有处理这些数据的能力，各企业可通过分析用户行为数据，从而洞察出用户的喜好，从而为用户带来更方便贴心的功能，大数据的应用已经在各个行业展开了。

本研究基于当前流行的 Spark 大数据处理框架，对商城模型的数据做了简要实时的处理分析。使用 Canal 与 Kafka 对接商城数据源，在前端部分则采用 node.js 与 socket.io 实时推送数据到浏览器，并利用 echarts 绘图展示。本研究一共统计了以下几个指标：用户地区分布与在线人数、商品销量前五、订单状态变化、价格趋势变化。其中用户地区分布、在线人数都采用总量统计；而商品销量前五、订单状态、价格趋势变化使用了滑动窗口操作（window）只进行最近一段时间内的统计。在 Spark 中，这些指标的计算使用了 Spark Streaming 与 Structured Streaming 两种流式处理方式对用户的数据进行处理，在本文可以看出这两种方式的异同以及它们是如何交互与相互转换。本文还探讨了如何在流式数据处理中使用 mapWithState 与 mapGroupsWithState 进行跨批次的数据状态保存以及更新，这些都是在实际开发流式处理程序会遇到的问题。

关键词：Spark 流式处理 商城统计 实时处理

Real-time Data Statistics System

Locez

Abstract: The development of data informatization is growing faster and faster. In modern society, a majority of operations are carried out on the Internet and modern terminal equipment. Behind these operations, an amount of data is always generated, and the meaning of this data is definitely beyond a record for enterprises. In addition, with the development of big data technology, people have the ability to process this data. Also, by analyzing user behavior data, enterprises can gain insights into users' preferences. Thus, they can provide users with more convenient and intimate functions. In conclusion, big data applications have been developed in various industries.

This study makes a brief real-time analysis of the data of the mall model, which is Based on the current popular Spark big data processing framework and use Canal and Kafka to dock the mall data source. In the front-end part, I use node.js and socket.io to push data to the browser in real time, and use echarts to display it. Besides, in this study, the following indicators were counted: user area distribution and online number, top five sales of goods, changes in order status, and price trends. Among them, the user area distribution and online number of people use the total statistics; while the top five of the product sales, the order status, and the price trend change use the sliding window operation only for the latest period of time. In Spark, the calculation of these indicators uses Spark Streaming and Structured Streaming to process the user's data. In this paper, we can see the similarities and differences between the two methods and how they interact and convert. In addition, this paper also descripts about how to use mapWithState and mapGroupsWithState to perform cross-batch data state

saving and updating in streaming data processing. These are the problems which will be encountered in the actual development of streaming programs.

Key words: spark streaming processing mall statisticals real-time processing

目 录

1 前言.....	1
1.1 选题背景.....	1
1.2 大数据计算框架发展.....	1
1.3 系统开发的意义.....	2
2 理论基础与方法.....	3
2.1 实时数据流处理框架.....	3
2.2 可行性分析.....	5
3 组件简介与特性.....	7
3.1 Canal.....	7
3.1.1 Canal 简介.....	7
3.1.2 Canal 特性.....	7
3.2 Kafka 分布式消息队列.....	8
3.2.1 Kafka 简介.....	8
3.2.2 Kafka 特性.....	9
3.3 Spark.....	9
3.3.1 Spark 简介与特性.....	9
3.3.2 Spark Streaming 简介与特点.....	10
3.3.3 Structured Streaming 简介与特点.....	11
3.4 Node.js 与 Socket.io.....	13
3.4.1 Node.js 简介.....	13

3.4.2 Socket.io 简介	13
3.5 ECharts	14
3.5.1 ECharts 简介	14
3.5.2 ECharts 特性	14
4 系统详细设计	15
4.1 需求分析	15
4.2 项目整体架构	15
4.3 技术环境	17
4.4 数据库设计	18
4.5 数据格式与接口约定	20
4.6 逻辑设计	22
4.6.1 用户地区分布与在线人数占比	23
4.6.2 商品销量前五	26
4.6.3 订单状态	28
4.6.4 价格变化趋势	30
4.6.5 数据展示	31
5. 测试与展示	33
5.1 测试环境	33
5.2 测试数据	33
5.3 测试结果	34
5.4 结果分析	35

5.5 结果展示.....	36
6 总结与展望.....	40
6.1 总结.....	40
6.2 展望.....	40
参考文献.....	42
致谢.....	44

1 前言

1.1 选题背景

在线购物在中国已经是非常成熟的销售方式，淘宝、京东、拼多多都是国内的大销售平台，人们已经习惯于在线购买商品，在线购物极大方便了消费者。而在一个商城系统中，往往会产生许多数据，包括但不限于用户浏览商品的记录，商品销售量，订单交易量，这些数据往往只是安静的躺在数据库，并没有被好好利用。

近年来，大数据一词非常热门，“大数据”是一个统称，具体到多大才算是大数据也没有一定的标准，但是基于大数据产生的一系列计算框架却告诉我们可以如何处理这些数据。传统大数据处理通常会经历采集、汇总、清洗、计算、得出结果这样一系列的步骤。而近年来流式数据处理渐渐成熟，使得我们可以“实时”对数据进行一些处理。

1.2 大数据计算框架发展

MapReduce 是 Google 提出的一个软件架构或者说是编程范式，用于大规模数据集并行计算，它把计算分为两个阶段，第一阶段“Map（映射）”操作，不断把一组键值对映射成新的键值对，第二阶段“Reduce（归纳）”操作，将拥有相同键值的数据进行归纳计算。Map Reduce 通过将数据集分发给每个计算节点进行计算，计算节点会周期性的将结果返回，所以理论上没有计算能力上限。

Apache Hadoop 是一款支持数据密集型的分布式开源软件框架，它基于 Google 发表的 MapReduce 和 Google 文件系统（一种专有的分布式文件系统，非开源软件）的论文实现而成。Apache Hadoop 是一个平台，它包括了 Hadoop 内核、MapReduce、以及 Hadoop 分布式文件系统（HDFS）和 Hadoop YARN（任务调度与集群资源管理框架），还衍生了一些项目：Apache Hive 和 Apache HBase 等（Tom White, 2015）。

因为 Hadoop 使用 HDFS 存储所有计算节点的数据，所以可以为该集群提供非常高的带宽，增加吞吐量。

Hadoop 会将所有的中间结果存储在磁盘上以变容错，但是这样性能上会大幅下降。2009 年，加州大学伯克利分校 AMPLab 的 Matei Zaharia 开创了 Spark，后捐赠给 Apache 软件基金会，成为 Apache Spark 项目。Spark 能在数据还未写入到磁盘的时候便在内存中进行运算与分析，得益于在内存直接计算减少了磁盘 IO，因此其运算速度能做到比 Hadoop MapReduce 运算速度快上一百倍，极大增加了大数据处理的速度，并且其在改善运算速度以及计算范式的同时还兼容 HDFS 等 Hadoop 已有的优秀组件，更加方便的扩充了自己的生态。而后还推出了 Spark Streaming，Structured Streaming，使得 Spark 能进行流式计算，对数据进行实时处理，这为本文所叙述的且将要实现的系统带来了极大的便利。当然流式处理还包括了其它的一些框架，例如 Apache Storm，Apache Flink 等（此处不再介绍，可参考第二章实时数据流处理框架）。

1.3 系统开发的意义

在在线商城系统中，由于用户操作，会产生许多数据，商城的用户量越大，这种数据的产生量就越大。而大数据的一个基本目的就是从海量的数据中，计算得到想要的结果。

在商城系统中，在线人数、订单量、商品销售量、某个商品的最近价格变化趋势如果能够实时统计出来，并且绘制成一定直观的图表展现给决策者，往往能够帮助决策者在经营过程中做好更好的决策。相较于以往的，例如统计上个月这样不够实时的系统，实时的好处则是不会有滞后性，不会导致决策者错过商机或者规避风险的机会。

2 理论基础与方法

2.1 实时数据流处理框架

大数据的发展已经经过了十多年，在这十多年中衍生了众多的计算平台，以及应用场景。它们经过多年的验证，被证明已经是成熟、稳定、有效的解决方案，而且对所有的大数据应用都有比较通用的架构分层，如图 1 所示：

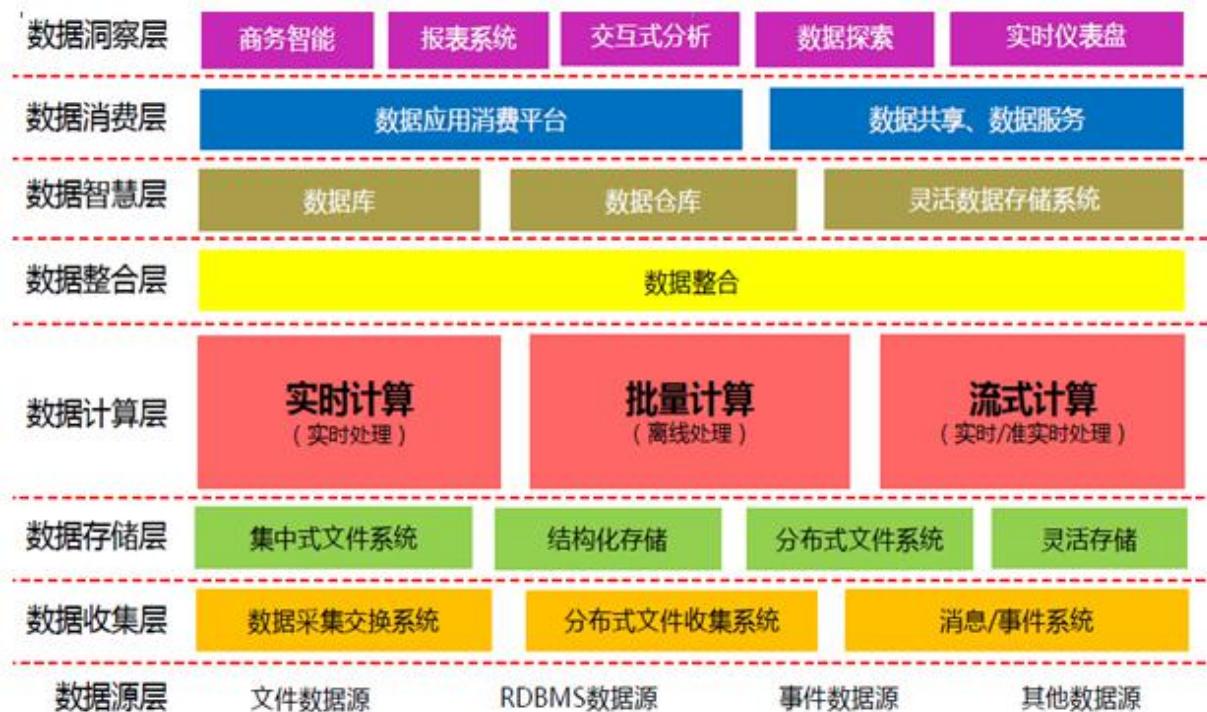


图 1 通用大数据应用架构图

而在数据计算层中，就是常说的大数据处理引擎，目前有基于批量计算（离线处理）的，也有基于流式处理（实时计算）的，甚至还有两者相结合（流/批结合准实时）的。本文只将只关注实时计算框架，目前较为通用的实时计算框架有：Spark、Storm、Flink 这三种，他们之间的差异（岳猛，2016），如表 1 所示：

表 1 实时计算框架对比

框架	模式	延迟	吞吐量	API	易用性	成熟性
Spark	流/批结合	秒级	高	高级	高	高
Storm	流处理	亚秒	低	低级	低	高
Flink	流/批结合	亚秒	高	高级	高	低

从表中我们可以看出 Storm 与 Flink 的延迟是比 Spark 更低的，更加适用于对实时要求高的系统，而 Spark 中为秒级延迟，适用于准实时的系统。其次 Spark 与 Flink 的吞吐量都比较高，且提供高级 API，支持 SQL 查询，这两个框架在数据处理以及易用性方面都比较有优势。一方面结合本项目的商城数据统计意义，我们对实时的要求并不需要达到亚秒级，只需要达到秒级就可以了，因而选用 Spark 框架作为计算框架。另一方面，Spark 的成熟度较高，拥有许多其它可以与它对接的生态，文档资料多，这会为开发人员带来一定的便利性，Spark 生态图如图 2 所示：

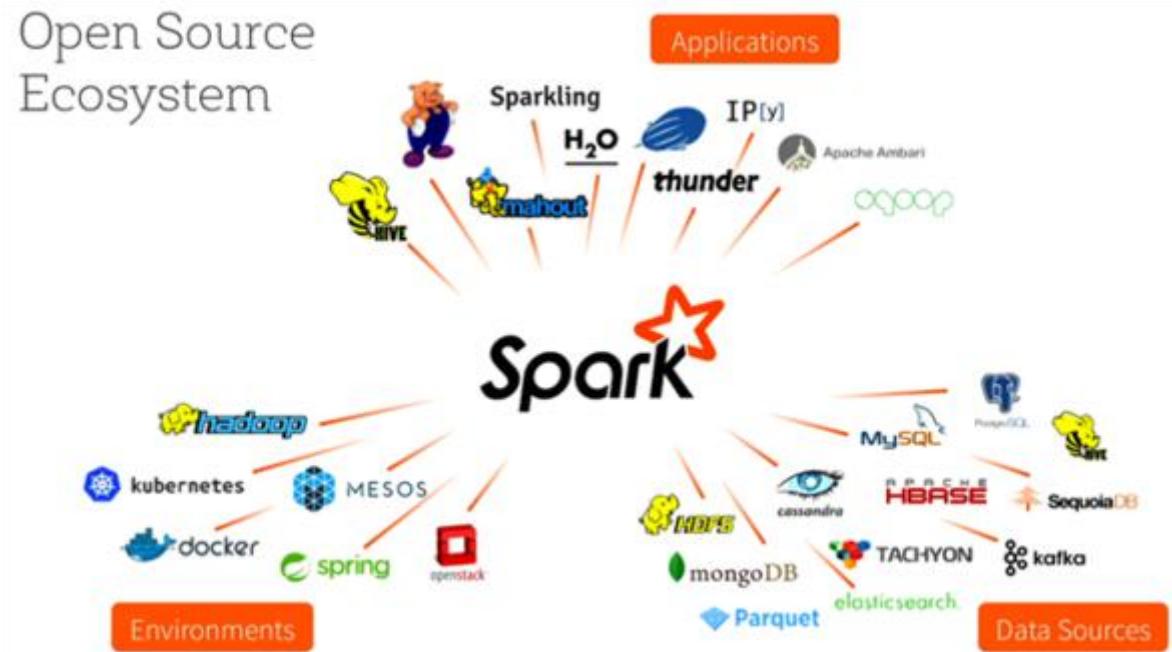


图 2 Spark 生态图

2.2 可行性分析

在进行实际应用开发之前，需要对项目进行可行性分析。可行性分析将从多个方面论证项目开发的可行性，并且为项目开发的决策提供可靠的依据，其目的是用最小的代价在尽可能短的时间内确定问题能否解决。以下将从技术、经济、社会和法律四个方面进行论证。

(1) 技术可行性

本项目是要基于用户商城行为数据进行实时处理并分析，用户产生行为后会在数据库中产生消息条目，可以将其推送到分布式消息队列，然后再在 Spark 框架中进行消费、处理。而 Spark Streaming 支持流式数据处理，因此可以达到实时的目的，最后再将结果输出到分布式消息队列。最后从结果队列中订阅，再将数据绘成一定的图表推送给用户。每个技术点都能找到相应的组件及社区支持，因此从技术上来说该项目是可行的。

(2) 经济可行性

本项目后端大数据部分数据库使用 MySQL 社区版，消息队列选用 Kafka，大数据框架使用 Spark，后端开发语言为 Scala。而本项目 Web 后端以及前端语言均采用 JavaScript，后端选用 Node.js，前端图表绘制采用 ECharts 图形库，实时推送到浏览器使用 Socket.io 库。这些软件以及开发工具都是开源软件，不需要进行商业上的收费，因此从经济上来说该项目是可行的。

(3) 社会可行性

本项目主要关注点为实时数据流处理，这在现在各种要求时延比较低的系统中都具有实际意义。此外本项目选择的是商城模型这种通用的数据源，在社会上也具有较高的实际应用场景，因此从社会上来说该项目是可行的。

(4) 法律可行性

本项目开发将严格的遵守国家相关法律规定，并且开发的所有模块功能都不涉及用户个人隐私数据分析，更不会将数据直接暴露给外界，切实保障用户信息安全。此外本项目仅用于学习探究，不作任何商业用途。

3 组件简介与特性

3.1 Canal

3.1.1 Canal 简介

Canal (水管/管道/沟渠) 是阿里巴巴推出的 MySQL 数据库的 binlog 日志的增量订阅和消费组件，它使用 Java 语言开发并且基于数据库增量日志解析，提供数据增量订阅和消费。其实现原理为 Canal 模拟 MySQL 的 Slave 交互协议，伪装自己为一个 MySQL Slave，向 Master 发送 dump 协议。Master 收到 dump 请求后，推送自己的 binary log 给 Canal，而后 Canal 解析 binary log (阿里巴巴, 2018)，其原理如图 3 所示：

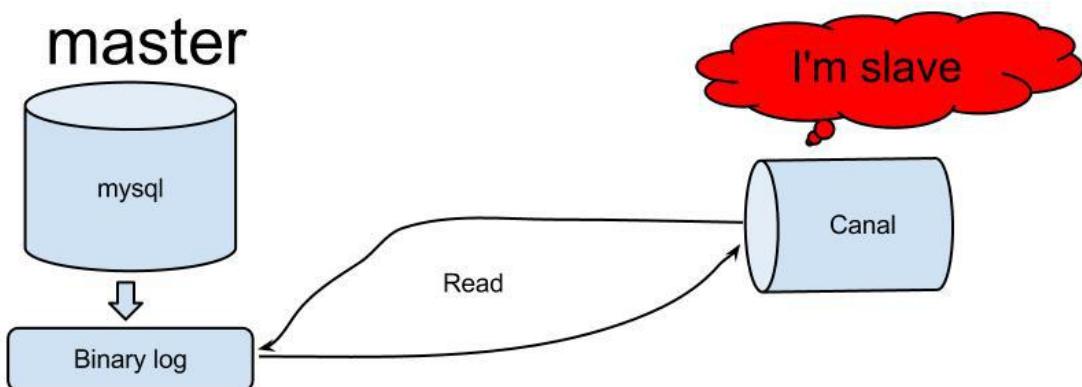


图 3 Canal 工作原理图

3.1.2 Canal 特性

- MySQL binlog 解析
- 实时
- 原生支持 Kafka/RocketMQ 消息队列
- 原生支持 Prometheus 监控

- 客户端支持多语言：Java、C#、Go 等

3.2 Kafka 分布式消息队列

3.2.1 Kafka 简介

Kafka 是 Apache 软件基金会下的开源流处理平台，最早由 LinkedIn 开发，并在 2011 年开源。项目主要由 Scala 和 Java 语言编写，目标是为处理实时数据提供一个高吞吐、统一、低延时的平台，但其持久化层本质上是一个大规模发布/订阅消息队列。因此社区主流是将其当作一个消息队列系统，而选择其它数据处理框架。

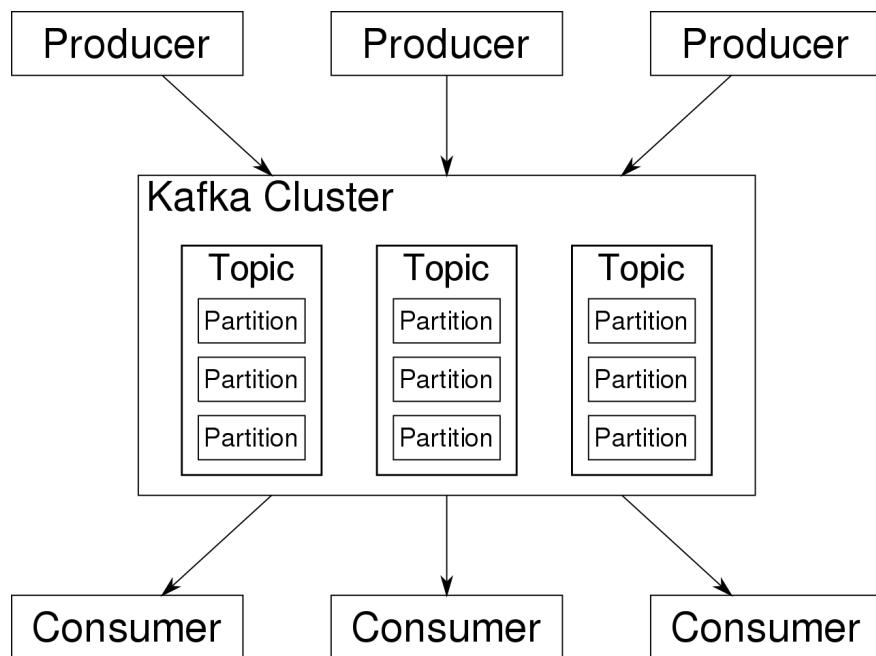


图 4 Kafka 框架图

如图 4，Kafka 主要采用生产者消费者模型，使用 Topic 区分不同的消息队列主题。为了提高消息处理效率，一个 Topic 下可以有多个分区，可以使用多个进程在这些分区写入或读取数据。

3.2.2 Kafka 特性

- 高吞吐量、低延迟：每秒可处理几十万条消息
- 持久性、可靠性：消息持久化到本地磁盘，支持数据备份防丢失
- 可扩展：Kafka 集群可不停机扩展节点
- 容错性：允许集群中节点失效，但应至少保证一个节点正常工作
- 高并发：支持上千客户端同时读写

3.3 Spark

3.3.1 Spark 简介与特性

Spark 是一个开源的集群计算框架，其在内存中运算，能做到比 Hadoop 运算速度快 100 倍。其主要由六个部分构成：Spark 核心与 RDDs（弹性分布式数据集）、Spark SQL、Spark Streaming、Structured Streaming、MLlib 以及 GraphX。

其中 Spark Streaming 以及 Structured Streaming 两种流式计算库（具体参见下文）本文都使用到了。Spark 提供了 80 多个高效的运算符帮助我们构建并行程序，还支持多种语言：Java、Scala、Python、R 以及 SQL，并且还提供了一个交互式的环境。

Spark 主要特点为：

- 支持多语言
- 可扩展至 8000 个节点
- 能交互式访问数据集
- 可扩展性、高吞吐量、可容错性
- 支持 SQL 结构化查询

- 提供 MLlib 机器学习算法与 GraphX 图形处理算法库

3.3.2 Spark Streaming 简介与特点

Spark Streaming 是 Spark 核心 API 的一个扩展，它为我们提供了在 Spark 中处理流式数据的能力。如图 5，Spark Streaming 支持多种数据源，同时还提供了诸如 map、reduce、join、和 window 这样的高效函数，方便我们处理数据。



图 5 Spark Streaming 支持的数据源

DStreams (Discretized Streams) 是 Spark Streaming 中的基础抽象，代表了 Spark Streaming 中流的概念，但其本身并不是最细微的粒度。DStreams 并非传统意义上的流，其底层是由 RDD (Resilient Distributed Dataset) 实现的。Spark Streaming 根据一定的时间间隔（这个时间间隔便是我们处理流数据的间隔），将数据汇集成一批，封装成一个 RDD。如图 6 所示，DStream 由一系列 RDDs 构成，因此 Spark Streaming 是以批处理的思想为我们提供了一个处理流式数据的实现。每批数据进入到 Spark，都是一个 RDD，RDD 是不可以直接修改的，但是可以通过 map (映射) 等函数生成一个新的 RDD (张安站, 2015; Apache Spark, 2018a)，所以我们可以在一个 RDD 中判断数据的内容，并决定其新转化的 RDD 的形式。



图 6 RDD 与 DStream 的关系

Spark Streaming 还提供了窗口操作，窗口操作会将一定时间内的 RDD 累计下来，并形成一个新的 RDD。这个新的 RDD 包含了这一段时间所有的内流入 Spark 的所有数据 (Apache Spark, 2018a)，我们可以利用这个特性去计算一段时间内的统计，其中转化示意图如图 7 所示：

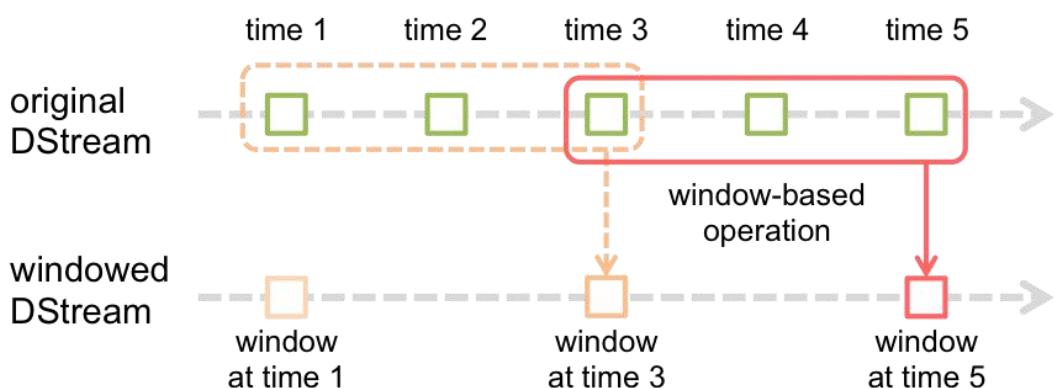


图 7 窗口操作转化 RDD 示意图

3.3.3 Structured Streaming 简介与特点

Spark Streaming 是较早出来的 API，其支持让我们直接操作底层 RDD，后来 Spark 的开发者们又开发了 Dataset/DataFrame API，这种一种结构化的数据表示方式，可以用 Spark SQL 进行查询。而基于此结构化数据的思想演变出了 Structured Streaming，这是基于 Spark SQL 引擎的流式处理。Spark SQL 引擎会随着流式数据的持续到达而

不断计算更新结果，其看起来就像是使用者在对一个数据库表进行全表查询。这种直观的操作方式让其比直接操作底层 RDD 的 Spark Streaming 有了一个飞越的进步，性能比 Spark Streaming 更好。在 Spark2.3 中，更是实现了一个更低时延（可低至 1 毫秒）的处理模式——持续处理。但是 Spark 强大之处在于，开发者可以在任何时候随意混合使用这些 API，灵活性非常强，本项目出于探究目的，对两种流式编程方式都进行了使用，混合。在生产环境中应该尽可能使用 Structured Streaming 进行编程优化，Structured Streaming 无法办到的再转化为底层 RDD 操作。

Structured Streaming 将数据流看作是一张没有边界的表，新的数据流入 Spark 的时候，将其当成新的一行添加到这个表的尾部，使得我们可以对这张表使用 Spark SQL 查询数据。它实际上会在内部维护两张表，一个是上面所说的包含所有输入的没有边界的输入表。另一张是结果表，输出结果也不断被增加到结果表中，最后输出模式也有三种：Complete Mode、Append Mode、Update Mode (Apache Spark, 2018c)。它同样支持窗口操作，不过与 Spark Streaming 基于处理时间的窗口操作不同，Structured Streaming 中的窗口操作的时间是基于数据流入 Spark 的时间，也就是数据的真实时间，这对 IoT 场景来说具有非常实际的意义。使用 Structured Streaming 处理单词统计的过程如图 8 所示：

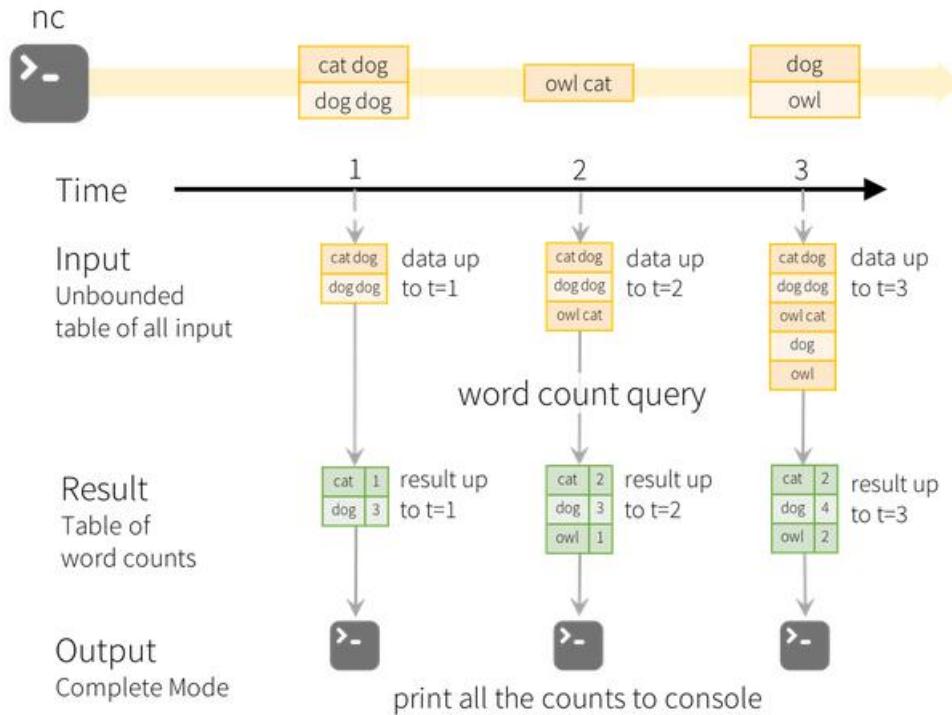


图 8 Structured Streaming 处理单词统计例子

3.4 Node.js 与 Socket.io

3.4.1 Node.js 简介

Node.js 是一个基于 Chrome V8 引擎的能够在服务端运行的 JavaScript 运行时。其使用事件驱动、非阻塞与异步输入输出模型来提高性能。在 Node.js 出现之前 JavaScript 通常只用于浏览器脚本语言，现在可以通过 Node.js 用 JavaScript 实现服务端了。此处选择 Node.js 也是为了统一用 JavaScript 语言去处理服务端和客户端，更加方便。

3.4.2 Socket.io 简介

Socket.io 是一个面向实时应用的 JavaScript 库，它能进行服务器与客户端进行双向通信，其主要使用 WebSocket 协议，也可回退到其他协议，Socket.io 会自动选择适合的协议进行双向通信，也同样因为这个特性客户端以及服务端都只能使用 Socket.io，

不能使用其它标准的 WebSocket 实现。它是基于事件机制的，通过触发事件便可以实时将数据推送到另一端，并且另一端的回调函数做出对应的动作。主要特点或用途：实时分析、聊天室实现、二进制流数据、文档协同。

3.5 ECharts

3.5.1 ECharts 简介

ECharts 是一个百度开源的可视化库，它使用 JavaScript 实现，能流畅在 PC 和移动设备上运行，兼容绝大多数浏览器，底层依赖矢量图形库：ZRender。ECharts 可以为我们提供直观，交互丰富，并且可以高度个性化定制的可视化图表。

3.5.2 ECharts 特性

- 丰富可视化类型：提供折线图、柱状图、散点图、饼图、地图、热力图等图表
- 支持多种数据格式：多种数据格式无需转换可直接使用
- 支持处理千万级数据量，加载多少渲染多少
- 进行移动端优化、多种渲染方案，可跨平台使用
- 支持图表交互
- 动态数据：能根据填入的数据变化自动选择适合的动画进行演变
- 特效绚丽

4 系统详细设计

4.1 需求分析

一个通用的商城的数据库往往包含一个用户表保存用户个人信息、一个订单表用于保存订单信息、还有商品表保存每件商品的属性、以及订单明细表（一个订单会购买多种商品）。基于这些通用的数据，可以抽象到一些以下希望统计的属性：

- (1) 用户地区分布
- (2) 商品销量统计
- (3) 用户在线人数与离线人数
- (4) 订单状态
- (5) 商品的价格变化趋势

4.2 项目整体架构

系统将被分为三个部分：数据获取、数据处理、结果展示。以下将一一说明三个部分的架构：

(1) 数据获取

因为目前主流数据库为 MySQL，所以后端数据库将采用 MySQL 存储数据（项目更具有通用意义），然后利用阿里巴巴开源的 Canal 组件模拟 Slave 从 MySQL 主库中同步数据，然后投递到 Kafka 分布式消息队列。

(2) 数据处理

Spark Streaming 从 Kafka 订阅数据并形成 DStream，在 Dstream 上或者 RDD 上可以进行 MapReduce 范式的编程，也可利用 Structured Streaming 将数据形成规

则的表进行 Spark SQL 查询。本文将在此基础上计算各种指标（计算指标请参见逻辑设计），并将最终结果再次投递到 Kafka 消息队列。

(3) 结果展示

结果展示页面采用百度开源绘图框架 ECharts 进行绘图。后端选用 Node.js，从 Kafka 结果消息队列中读取数据，然后通过 Socket.io 即时推送到客户端浏览器上，更新图表数据，无需用户手动刷新。

以 Kafka 消息队列作为媒介连接三个部分，可以解耦系统，将来系统中有任何一部分的组件需要更换都可以单独更换无需修改整体，本项目的整体框架以及数据流向如图 9 所示：

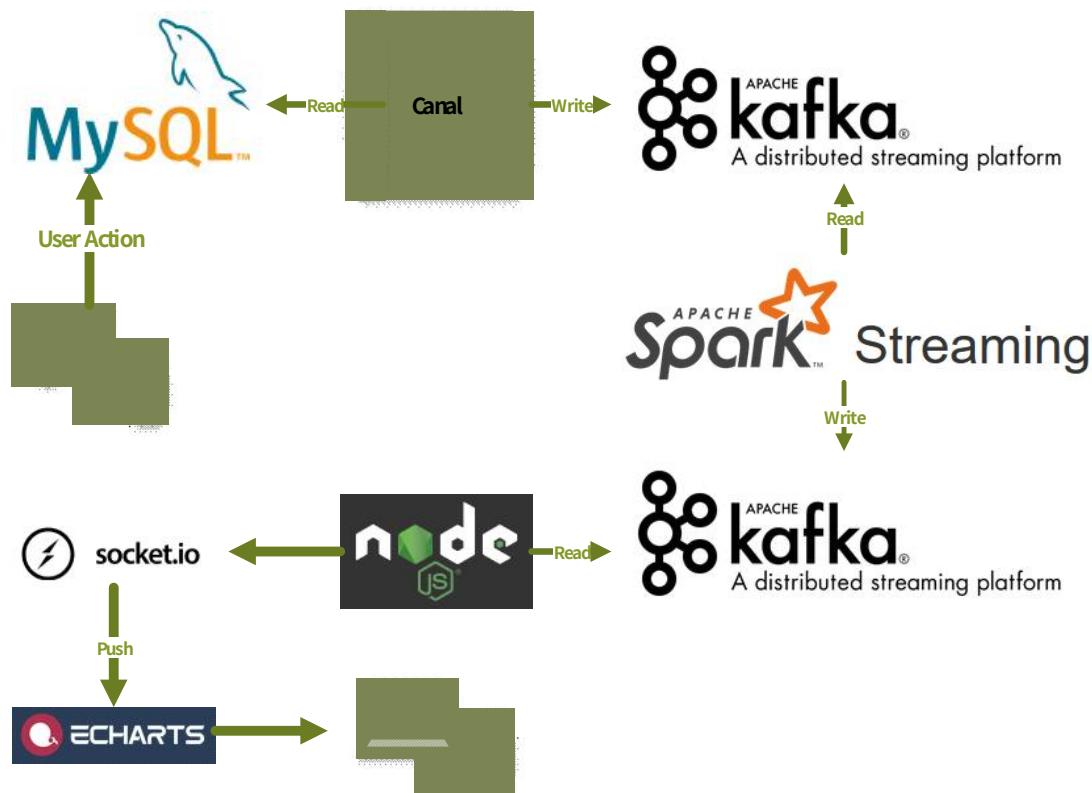


图 9 项目整体框架图

4.3 技术环境

(1) 开发环境

操作系统: Windows 10

开发语言:

Scala-2.11.3

JavaScript

Node.js- v8.15.1

IDE:

IntelliJ IDEA 2018.3.4 x64

Visual Studio Code

剩余组件版本:

MySQL: 5.6.43

Canal: 1.1.2

Kafka: 2.1.1

Spark: 2.4.0

Socket.io: 2.2.0

ECharts: 4.0.6

JDK: 1.8

(2) 服务器环境:

操作系统: CentOS Linux release 7.6.1810 (Core)

内存: 4G

数量: 3

4.4 数据库设计

系统将选用 MySQL 作为对接数据库，使用本系统原则上需要一套现有的商城系统。数据库部分不作为本系统的核心，但为了模拟商城的这种行为，因此构造如表 2 所示的四个数据库表：

表 2 数据库表清单

表名	作用描述
users	存储用户个人信息
orders	存储订单状态信息
products	存储商品信息
order_details	存储订单明细信息

用户表有六个字段，其结构如表 3 所示：

表 3 users 数据表结构

字段	描述	类型	长度	主键/外键	约束
id	用户 id	int		主键	auto_increment
user_name	用户名	varchar	255	否	not null

mobile_nu	手机号	varchar	11	否	not null
m					
city	城市	varchar	10	否	not null
online	是否在线	varchar	10	否	not null
register_ti	注册时间	timestamp		否	not null
me					

订单表有四个字段，其结构如表 4 所示：

表 4 orders 数据表结构

字段	描述	类型	长度	主键/外键	约束
id	订单 id	int		主键	auto_increm ent
user_id	用户 id	int		外键	not null
order_stat	订单状态	varchar	10	否	not null
us					
order_time	创建时间	timestamp		否	not null

订单明细表有六个字段，其结构如表 5 所示：

表 5 order_details 数据表结构

字段	描述	类型	长度	主键	约束
id	订单明细 id	int		主键	auto_increm ent

order_id	订单 id	int		外键	not null
p_id	商品 id	int		外键	not null
p_name	商品名称	varchar	255	否	not null
buy_count	购买数量	int		否	not null
total_price	总价	float	(10,2)	否	not null

商品表有四个字段，其结构如表 6 所示：

表 6 products 数据表结构

字段	描述	类型	长度	主键	约束
id	订单明细 id	int		主键	auto_increment
p_name	商品名称	varchar	255	否	not null
p_category	商品分类	varchar	10	否	not null
unit_price	商品单价	float	(10,2)	否	not null

4.5 数据格式与接口约定

在数据库中插入一条记录以后，经过 Canal 解析，投递到 Kafka 消息队列中的数据格式为 Json（一种可阅读的以键值为基础的文本数据格式）。该 Json 属性如表 7 所示：

表 7 Canal 解析后的 Json 格式（部分属性）

属性名	描述

data	数组，经过任意操作后，数据库现有的记录值
database	数据库名称
table	数据库表名
old	数组，数据操作前的值
type	INSERT、UPDATE 等操作

从 Spark 中运算得到的结果也将使用 Json 格式，再投递到结果消息队列，该格式因为需要满足前端绘图需要（可直接用于绘图，无需再在 Node.js 服务端做转换），会相对比较复杂，其中主要属性说明如表 8：

表 8 Spark 输出 Json

属性名	描述
map	用户地区分布数据，数组类型，每项包含 name 跟 value 两个属性
online	在线人数数据，数组类型，每项包含 name 跟 value 两个属性
bar	销量柱状图数据，包含 xAxis 和 yAxis 两个属性，两个属性均为数组
order_status	订单状态数据，数组类型，每项包含 name 跟 value 两个属性
s	
stackedline	价格变化折线图，该结构较为复杂，不在此表格列举

4.6 逻辑设计

因为 Canal 会将数据逐条投递到 Kafka，因此会发现在如果计算两个或以上的指标所需要处理的数据在一张表上，需要一次性完成这些计算操作。因为在流式计算中，一条数据被消费了就过去了，便假定不存在了。此处可以把流式计算想象成一条水管，数据就是水管中的水，数据流过一个地方那么就不会第二次流过。但是系统所计算的数据，有一些是有状态的，也即需要之前的结果。Spark 为我们提供了一种保存之前数据的方法，在下文也会提及。此外，本程序基于探究的目的，仍然使用了 Spark Streaming 底层 API——直接操作 RDD，同样也着眼于全局使用了 Structured Streaming 的基于 Spark SQL 引擎的新式结构化流处理 API，二者甚至会混用。接下来分别描述各部分的逻辑设计以及难点。

本文采用 Spark 提供的 Kafka 工具类读取流数据，`SparkStreamingContex` 对象会有一个时间间隔，该时间间隔决定了过多长的时间将数据汇集成一个 RDD，为了下文方便引用，称该时间为“间隔时间 T”。在读取数据创建 RDD 后，将会根据 json 字段的“table”属性数分别处理。每个表对应的功能结构如图 10 所示：

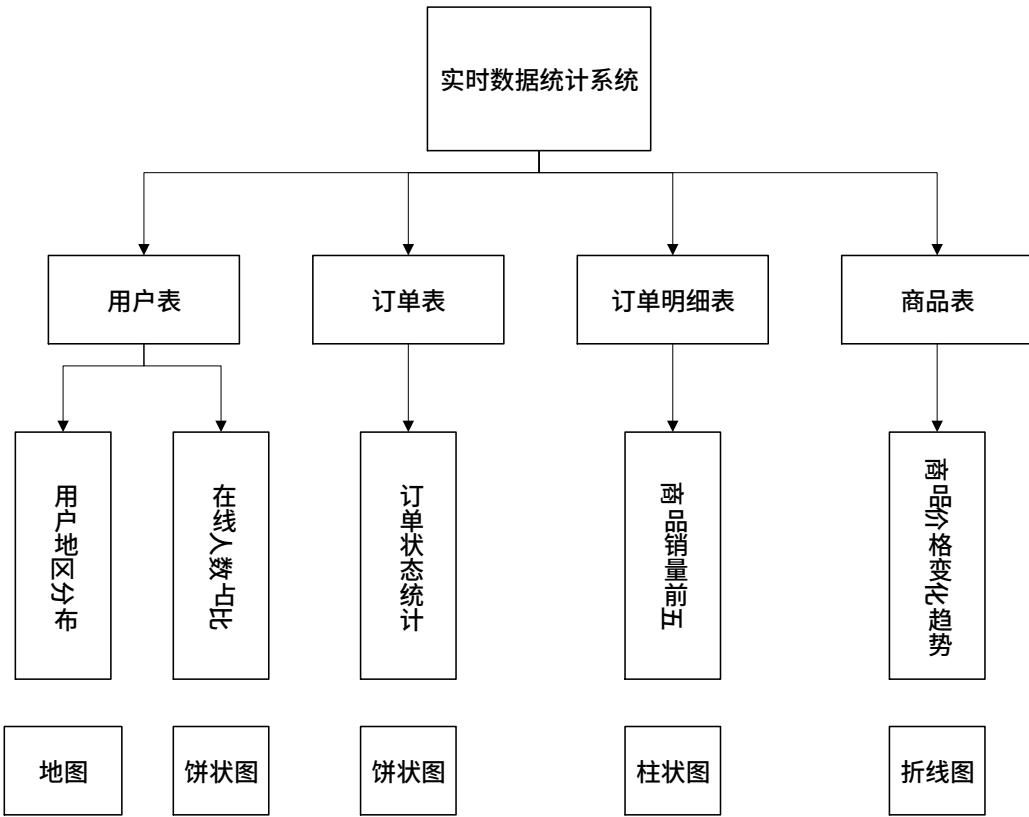


图 10 系统功能结构图

4.6.1 用户地区分布与在线人数占比

用户地区信息与在线状态均保存在用户表，因此这两项指标的计算需要同时完成。此处定义一个 case 类，通过 json4s 库将 json 文本解析成对应的对象结构：

```

case class User(id: Option[String], user_name: Option[String],
mobile_num: Option[String],
city: Option[String], online: Option[String], register_time:
Option[String])
  
```

Option 是允许该值为空的意思，因为在 UPDATE 操作中，原始 json 数据的 old 属性只会提供那些更改了的属性值，并不会像 data 属性一样把整个表传递过来，此处是为了兼容性，以下其它表的处理同理。

在用户表中，要统计的信息为用户地区分布和在线人数统计，因此将主要关注两个操作：一个是 INSERT，另一个是 UPDATE。因为这两个操作都会影响这两个指标。相对于用户注销账号的情况，为了简化模型，此处暂不考虑。通过分析可知，在 INSERT 操作中，需要做的是把某地区的人数加一，把离线人数加一（默认注册账号后不登录，用户为离线状态）。而在 UPDATE 操作中，可以配合 old 属性的内容，得知是哪个地区的用户需要减一，然后根据 data 属性的内容新地区人数加一。同样也可以知道用户的登录状态是从离线转为在线还是从在线转为离线，然后做对应的操作。

处理该部分逻辑的流程大致如图 11：

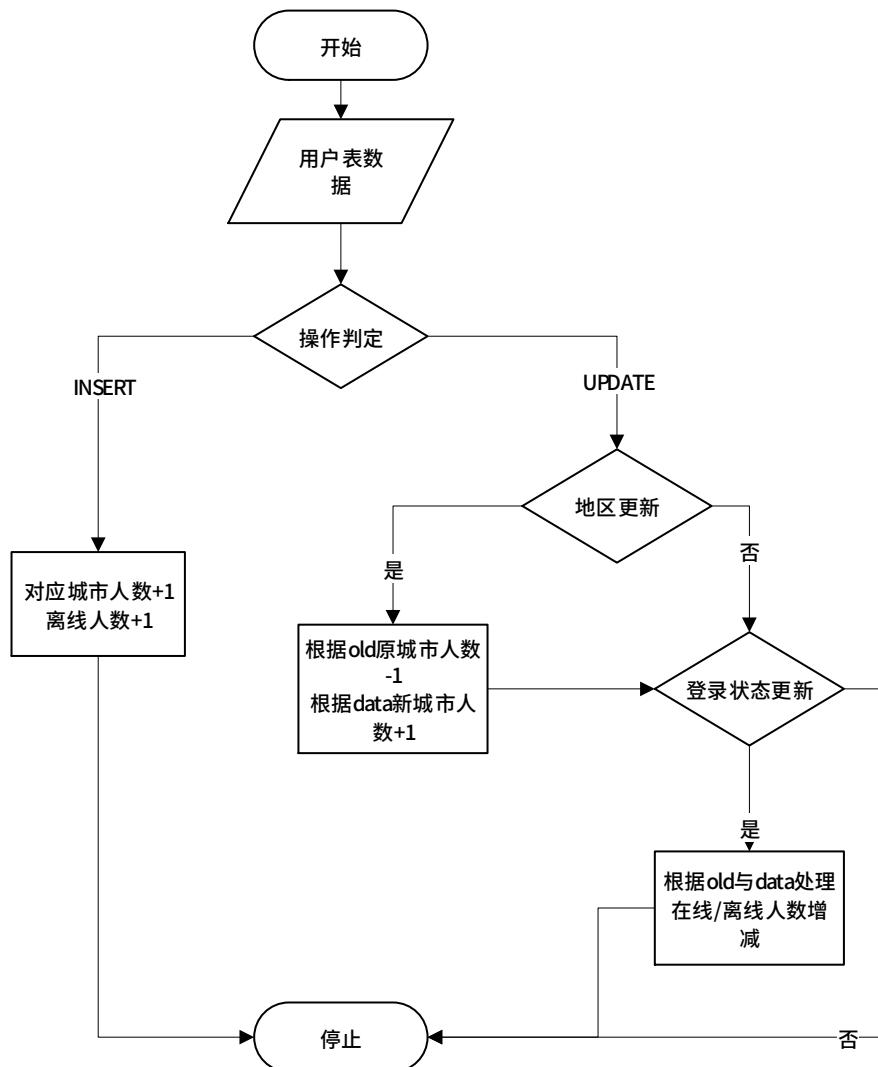


图 11 用户表处理逻辑流程

在上面的流程中，如果是一个平常所写的普通程序，已经是简单到不能再简单了。但是在 Spark Streaming 中，数据是按照批次来的，系统需要保存的总人数以及在线统计这种数据是跨批次的，是有状态的，不是处理后就可以丢弃的数据。此处可以将该数据写入 redis 缓存，需要的时候再读取回来，但是这样非常麻烦。Spark 为此提供了两种可以跨批次保存状态的方法，一种是通过 `updateStateByKey`，另一种则是 `mapWithState`。由于因为前者性能问题才出现了后者的改进（Tathagata Das, Shixiong Zhu, 2016），所以此处使用后者实现状态保存。`mapWithState` 方法签名如下：

```
def mapWithState[StateType: ClassTag, MappedType: ClassTag](
    spec: StateSpec[K, V, StateType, MappedType]
): MapWithStateDStream[K, V, StateType, MappedType]
```

该方法接受一个 `StateSpec` 对象，并将流转换为 `MapWithStateDStream`（在该方法后的流均为有状态的流），需要这样使用 `mapWithState`：

```
Stream.mapWithState(StateSpec.function(mappingFunc))
```

因此真正定义的是 `mappingFunc`，需要在这个函数内做状态更新，该方法签名如下：

```
mappingFunc (key: type, value: Option[type], state: State[type])
```

为了使用此函数，必须将 RDD 转换为 `(key,value)` 这样的 RDD，然后才能在该 RDD 上调用 `mapWithState` 方法，其中的 `key` 与 `mappingFunc` 中的 `key` 一致，`value` 代

表当前的值，可以是任意类型的，state 表示已经保存的状态，该类型与 value 应该是一致的。在此处，使用了“userMap”作为 key，然后其 value 使用了 scala.collection.mutable.Map[String, Int] 类型，这样便可以根据一个 key，取出一个 Map 映射然后对这个映射进行操作了。该 mappingFunc 最终实现如下：

```
val mappingFunc = (key: String, current: Option[scala.collection.mutable.Map[String, Int]], state: State[scala.collection.mutable.Map[String, Int]]) => {

    key match {
        case "userMap" =>
            val preMap = state.getOption().getOrElse(scala.collection.mutable.Map[String, Int]())
            val curMap = current.getOrElse(scala.collection.mutable.Map[String, Int]())
            val newMap = preMap ++ curMap.map(t => t._1 -> (t._2 + preMap.getOrElse(t._1, 0)))
            val output = (key, newMap)
            state.update(newMap)
            output
        case _ => ("", scala.collection.mutable.Map[String, Int]()
    }
}
```

4.6.2 商品销量前五

商品销量信息存储在订单明细表中，因此需要定义一个与订单明细表对应的 case 类用于 json 数据转换：

```
case class Order_detail(id: Option[String], order_id: Option[String], p_id: Option[String],
                        p_name: Option[String], buy_count: Option[String], total_price: Option[String])
```

在订单明细表中，主要关注 `buy_count` 字段，需要把跨批次的相同商品的数量计算出来。在此处不再使用 `mapWithState` 去保存状态，而是选择使用 Spark Streaming 中名为 `window` 的窗口操作。Window 操作会将一定时间内的数据积累下来，形成一个新的 RDD，然后再根据一定的时间滑动窗口，这两个时间都必须是时间间隔 T 的倍数。滑动窗口间隔时间在此处选择与时间间隔 T 相同，而积累数据的时间，我们可以根据需求选择一周或者一个月，在此处将计算最近 30 天的商品销量前五。

窗口操作的逻辑流程大致如图 12：

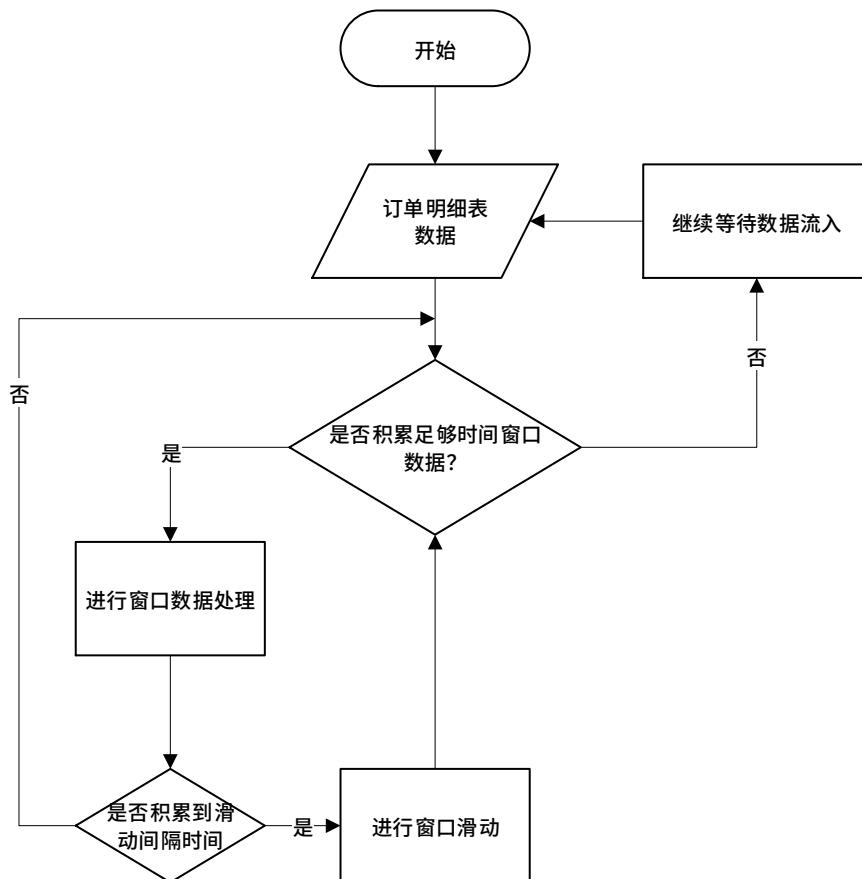


图 12 窗口操作流程图

在实际的代码中，使用了 `reduceByKeyAndWindow` 这个方法进行窗口操作，该方法会将所有 key 相同的值进行 Reduce 操作，此处的 Reduce 操作为将两个值相加。在该操作后形成新的 RDD 就是这段时间内累计的所有商品以及他们对应的销量。对该 RDD 进行按照值排序，最后取出前五，并生成 json 格式数据投递到 Kafka 结果队列。

4.6.3 订单状态

订单表所对应的 case 类：

```
case class Order(id: Option[String], user_id: Option[String], order_status:  
Option[String],  
order_time: Option[String])
```

一个订单状态会经历几个状态：待付款、待发货、待收货、完成。而这几个状态都是对同一个订单而言的，因此是一个需要保存状态的流。此处将要统计最近 30 天的订单状态。上面介绍的两种方法已经能实现这种操作，但是在此处将在 Spark Streaming 中使用 Structured Streaming 进行数据处理（因为一开始数据流入为 Spark Streaming），这种处理方法也可以在 Spark 中清洗数据，清洗成符合规则的数据后，再使用 Structured Streaming API 进行 DataSet/DataFrame 优化处理。该部分处理逻辑如图 13 所示：

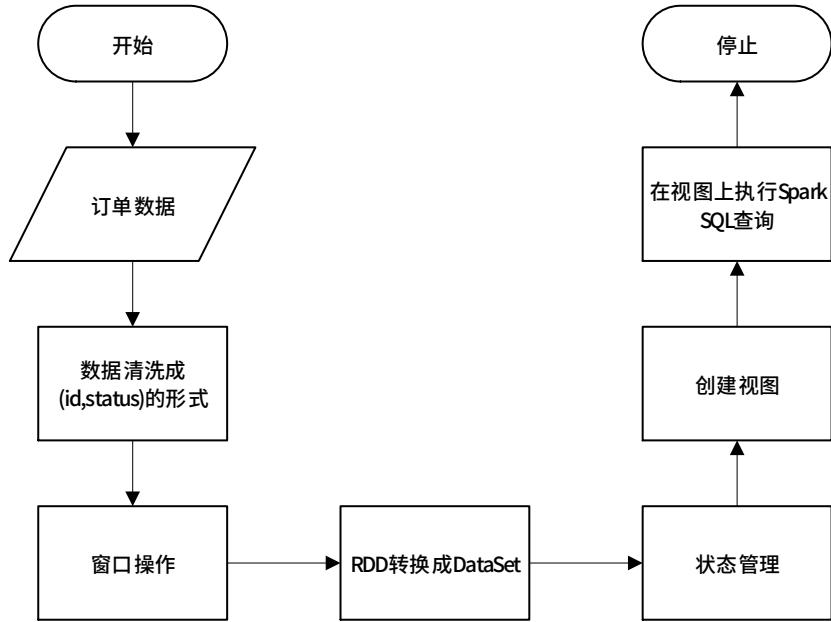


图 13 订单状态统计流程图

在 Structured Streaming 中，将使用 `mapGroupsWithState` 方法进行状态管理 (Tathagata Das, 2017)，该方法需要在 `groupByKey` 方法后调用，key 应为能区分不同 item 的值，此处我们选择订单 ID 作为 key。`mapGroupsWithState` 的参数含义与 `mapWithState` 方法的含义相似，可参照上文，具体实现为：

```

def updateOrderState(key: String, current: Iterator[Order_Status],
state: GroupState[Order_Status]): Order_Status = {
    val newValue = current.next()
    state.update(newValue)
    newValue
}

```

要使用 Structured Streaming，还需要 Spark Session 入口，然后使用 `toDF` 方法将 RDD 转为 DataFrame，调用 `as` 方法还可转为有类型的 DataSet。转换成功后，

便可创建一个视图，然后在此视图上使用 Spark SQL 进行查询。此外无需为每次转换都生成一个 Spark Session，可以使用单例模式缓存它：

```
object SparkSessionSingleton {  
    @transient private var instance: SparkSession = _  
    def getInstance(sparkConf: SparkConf): SparkSession = {  
        if (instance == null) {  
            instance = SparkSession  
                .builder  
                .config(sparkConf)  
                .getOrCreate()  
        }  
        instance  
    }  
}
```

4.6.4 价格变化趋势

商品表所对应的 case 类：

```
case class Product(id: Option[String], p_name: Option[String],  
                  p_category: Option[String], unit_price: Option[String])
```

在价格变化趋势中，需要统计最近 30 天的价格变化趋势，并给出变化趋势最明显的 10 种商品。这里采用计算数据方差的办法，然后根据方差对数据进行排序，并取出前十价格变化最明显的商品。

在该处理中，不但要在窗口操作中要对数据进行 Reduce 操作（将同一商品的价格按顺序集中到一起），还应该在窗口操作后能还原数据，这样一来才能在最后汇总进行计算。此处使用字符串拼接的方式，Reduce 操作将商品价格变为如下的字符串形式：

"价格 a,价格 b,价格 c,..."

key 仍为商品自身，因此等窗口时间积累后，便可以重新切割字符串形成数组去计算方差，然后以（方差，（商品，价格数据））的形式进行排序。商品价格变化趋势的整个流程如图 14 所示：

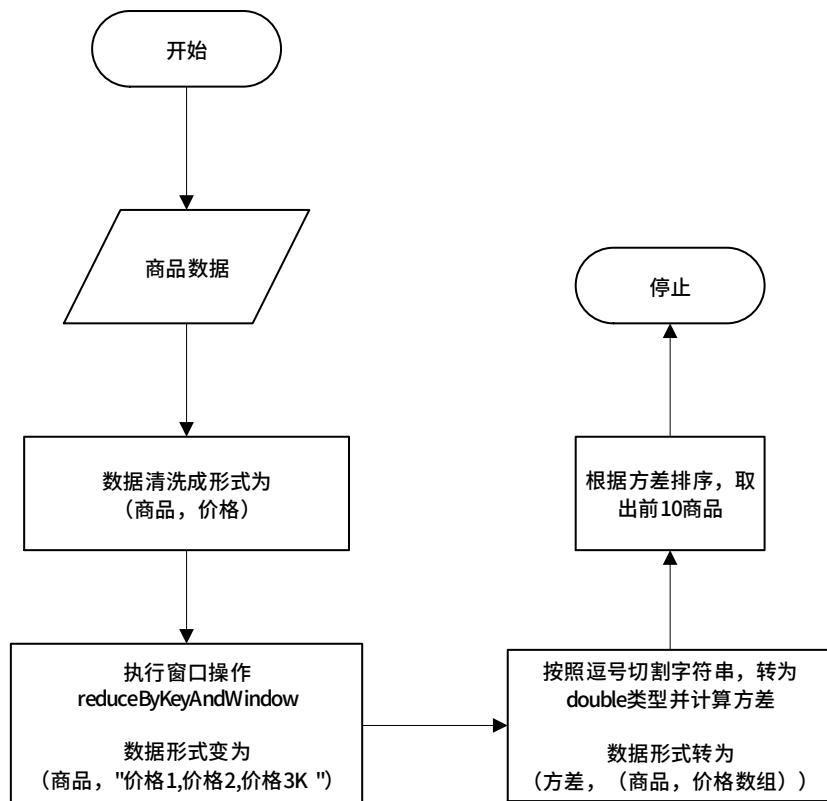


图 14 商品价格变化趋势处理流程

4.6.5 数据展示

在 Spark 中处理完计算后，json 数据会被投递到 Kafka 结果队列中。使用 Node.js 进行订阅，然后利用 Socket.io 的事件机制，触发一个事件，并将数据推送到客户端（浏览器），客户端将数据填充到 ECharts 中，完成绘图展示。无数据流通过时的展示页面如图 15 所示，有数据流入的完整图形，请参见第五章的结果与展示部分。

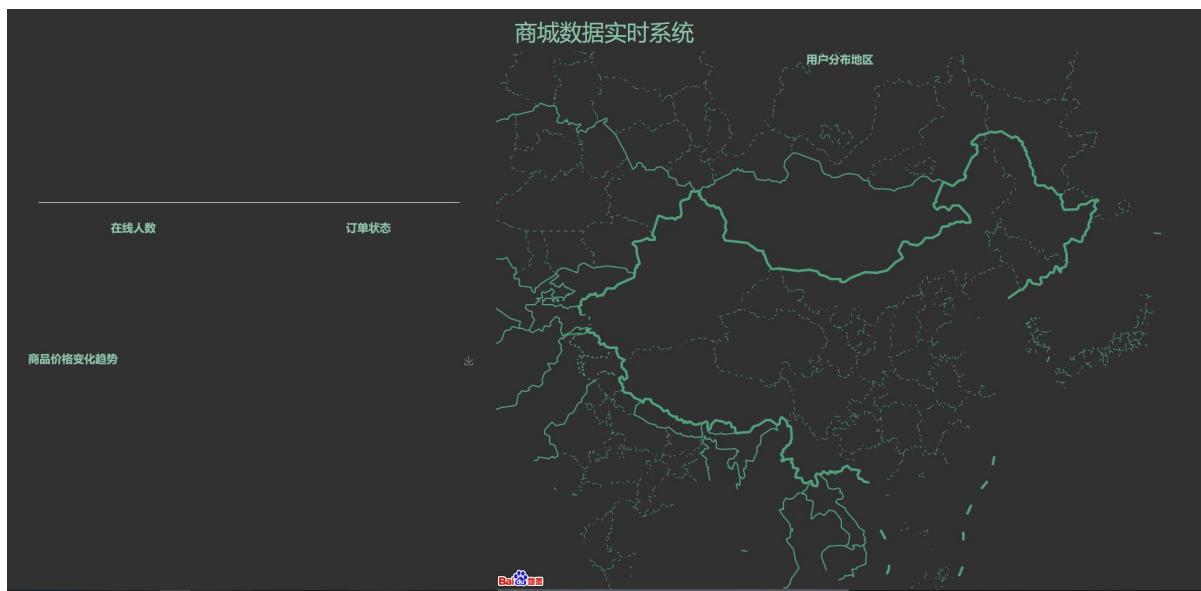


图 15 无数据流通过的前端页面

5. 测试与展示

5.1 测试环境

Spark 环境部署在校内三台 Centos7 服务器上， Kafka 部署在其中一台服务器上。

但是为了便于测试以及观察结果，由本机 windows 10 充当 driver 节点，以及结果展示节点，这样会存在以下几个需要注意的地方：

- (1) 对于需要从 worker 节点进行排序的操作在处理后，都需要回收到 driver 节点
- (2) driver 节点处理后要求写入到 Kafka
- (3) 最后 Node.js 展示部分又会从 Kafka 读取数据，再推送到浏览器

服务器配置：Centos7， E7-4830 v3， 4G 内存

driver 配置：windows 10， i7-4710MQ， 16G 内存

而在 Spark 中，窗口处理的时间不再为 30 天，而是设置为 5 分钟，滑动窗口间隔时间为 10 秒， Spark 从 Kafka 汇集数据时间间隔 T 也为 10 秒。

此处将采用对数据库直接插入数据来进行商城行为的模拟，脚本会在五分钟内对数据库进行操作插入或更新操作，并记录操作条目数量。Node.js 服务端会在接收到数据后，顺便将时间打印出来，将根据这些推算系统大致的性能。其中脚本使用 ruby 语言编写，其要插入的数据参见 5.2。为了减少窗口处理时间造成的误差，在测试之前 Spark 程序将会启动空跑 5 分钟以上，保证有需要窗口操作的数据流入的时候也能被马上处理。

5.2 测试数据

测试脚本会进行如下的操作：

- (1) 持续插入用户数据

插入数据如：(1， 露西， 18600001234， 广州， false， 2019-04-10 00:00:00)

(2) 循环随机修改用户在线状态

如将上述名为露西的用户的在线状态：从离线变为在线， false->true，

从在线变为离线， true->false

(3) 插入商品数据

插入数据如：(1, iPhone 7, mobile, 7000.00)

(4) 循环随机修改商品价格

对已经存在的商品原来的价格加上（或减去）一个三位数的值：

(1, iPhone 7, mobile, 7000.00+400.00)

(5) 持续插入订单数据

插入数据如：(1, 1, 待付款, 2019-04-10 00:00:00)

(6) 循环随机修改订单状态

订单状态将根据如下规则变化：

(1,1,待付款,2019-04-10 00:00:00)->(1,1,代发货,2019-04-10 00:00:00)

(1,1,代发货,2019-04-10 00:00:00)->(1,1,待收货,2019-04-10 00:00:00)

(1,1,待收货,2019-04-10 00:00:00)->(1,1,完成,2019-04-10 00:00:00)

(7) 持续插入订单明细数据

插入数据如：(1, 1, 1, iPhone 7, 10, 70000.00)

5.3 测试结果

测试结果如图 16，图 17 所示。从图 17 中可知，最后一条直接操作（相对于窗口）的输出时间，可以计算出时延大致为：108 秒，而从数据输入到数据完全处理完输出经历了一共 409 秒，所以大致估算出整个系统 TPS 为：483.03。

```
[root@centos7 tmp]# ruby bigdata.rb
#<Mysql2::Result:0x000000002b9aa80>
2019-04-07 23:25:22 +0800
end 2019-04-07 23:30:23 +0800
row 197562
```

图 16 插入脚本输出

```
direct map Sun Apr 07 2019 23:32:11 GMT+0800 (中国标准时间)
direct online Sun Apr 07 2019 23:32:11 GMT+0800 (中国标准时间)
direct map Sun Apr 07 2019 23:32:11 GMT+0800 (中国标准时间)
direct online Sun Apr 07 2019 23:32:11 GMT+0800 (中国标准时间)
window order_detail Sun Apr 07 2019 23:32:11 GMT+0800 (中国标准时间)
window stackline Sun Apr 07 2019 23:32:11 GMT+0800 (中国标准时间)
window order_status Sun Apr 07 2019 23:32:12 GMT+0800 (中国标准时间)
window order_detail Sun Apr 07 2019 23:32:20 GMT+0800 (中国标准时间)
```

图 17 Node.js 控制台输出

5.4 结果分析

这个测试结果有点出乎意料，在我看来可能由于以下几个原因：

(1) 通过计算可得知该系统整体的 TPS 约为 483.03，MySQL 的写入的 TPS 为 658.54。这是一个非常低的数据，起初以为是机器负载过高，导致资源被占用，所以 MySQL 插入才如此之慢。后在本地机器验证排查，这是 MySQL 单库多表做写入操作的正常速度，要解决只能分库分表继续优化。但是由于本人精力有限，暂时无法继续优化，至少该结果看来瓶颈似乎不在于本系统，而更多表现在数据库插入方面。

(2) 时延为 108 秒，这其中是我本地机器充当 driver 节点的，数据在 worker 节点计算后，需要汇集到 driver 节点中排序（服务器到本地），然后再写入回 Kafka（本地到服务器），Node.js 再从 Kafka 服务器取回数据（服务器到本地），这三个步骤的网络延迟应该也占了一部分原因。不过总的来说，108 秒的时延在一个准实时的系统里面，是一个能接受的数值。作为一个统计展示面板我认为这个数值可以优化，但是不优化也在接受范围内了。

5.5 结果展示

脚本后台持续插入后，前端展示部分的整个页面变化情况如图 18，图 19，图 20 所示：

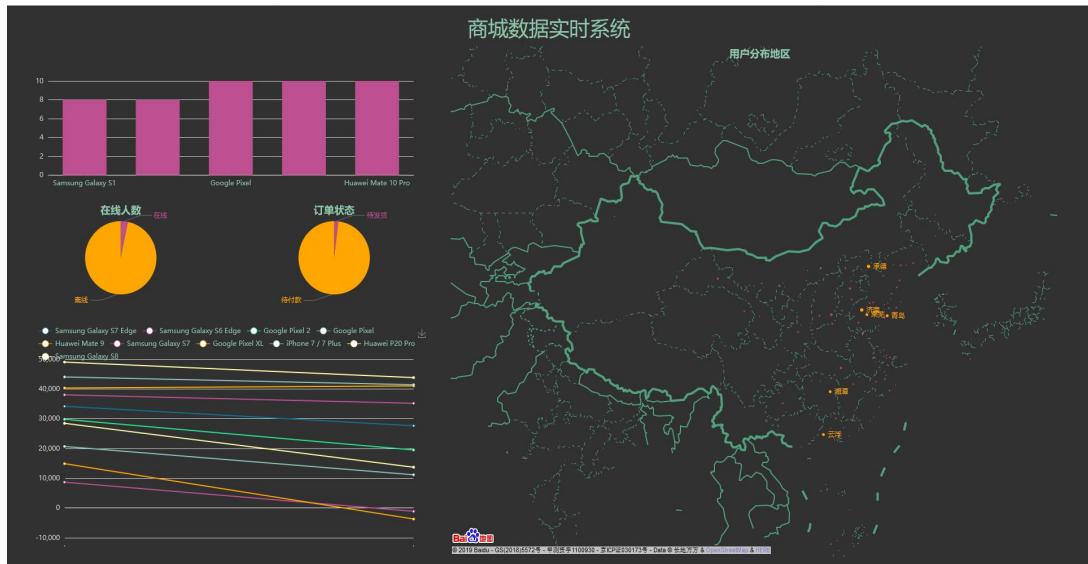


图 18 程序刚运行时

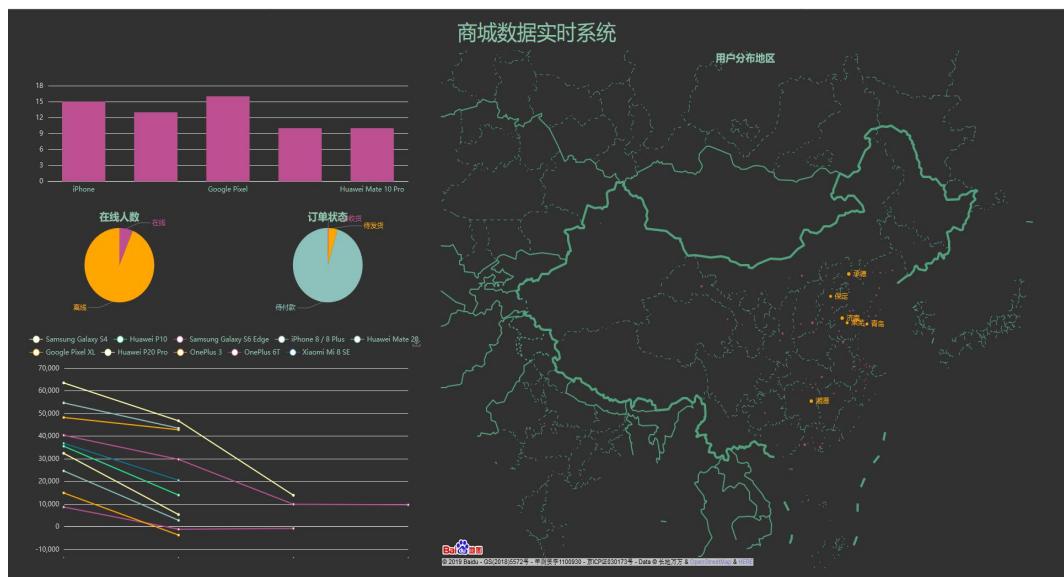


图 19 程序运行一段时间后

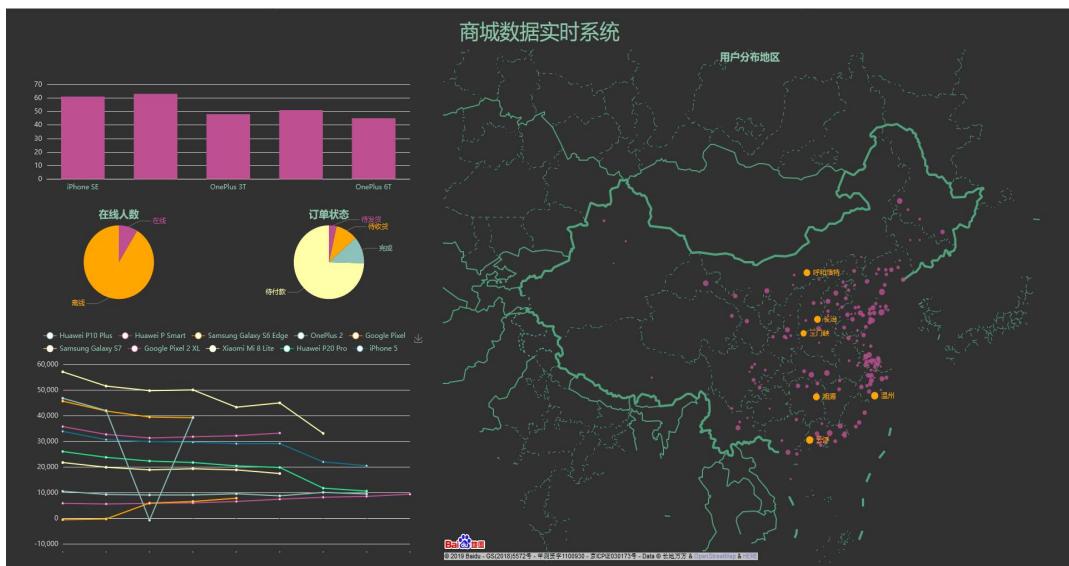


图 20 程序稳定运行

用户地区分布地图会将前五的用黄橙色标出，如图 21 所示，若出现六个点，则为有并列的情况。

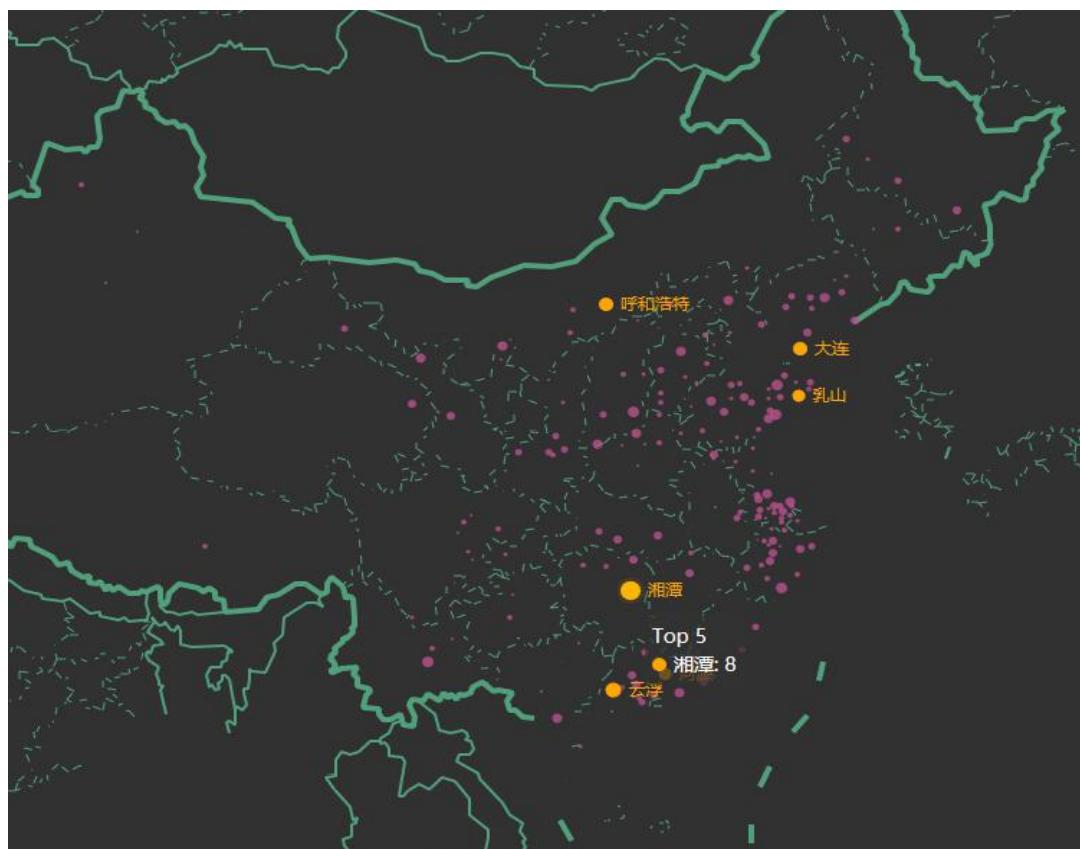


图 21 用户地区分布图

图 22 为商品销量前五柱状图，其中横坐标的名字太长的话，就不再显示在横坐标中，将鼠标放到该柱状图即会显示商品名。

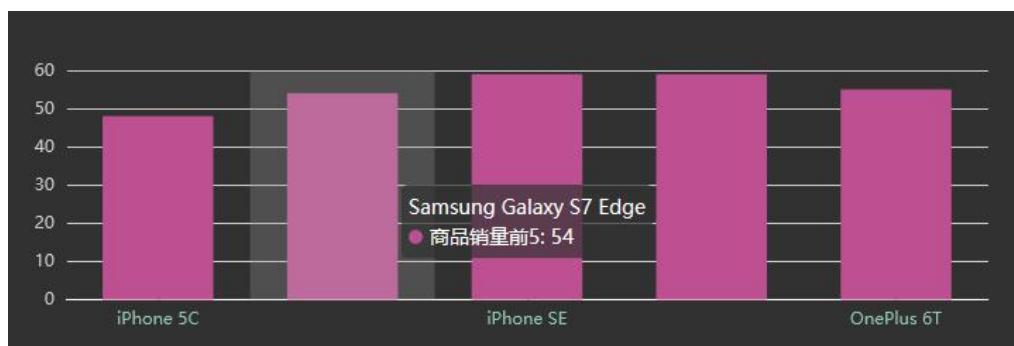


图 22 商品销量柱状图

在线人数与订单状态采用饼状图展示，将鼠标放到图上会有具体的数值，以及占总数的百分比，如图 23 所示：

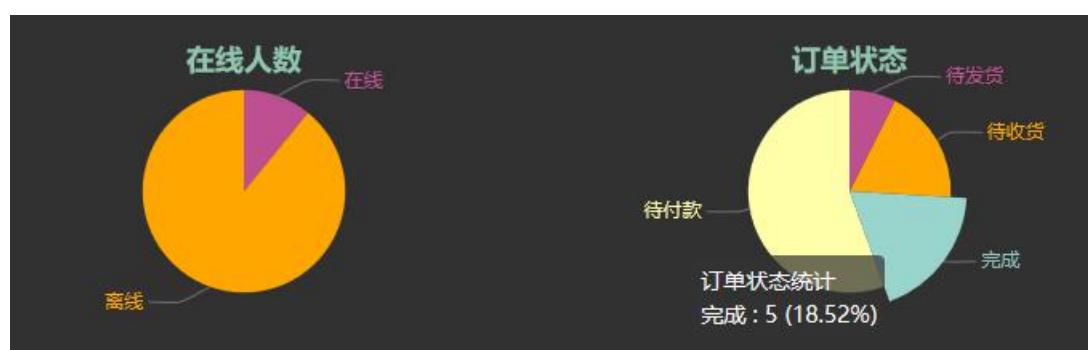


图 23 饼状图

商品价格趋势变化使用折线图绘制，最多会绘制 10 个商品的价格变化趋势图。商品价格变化次数多则绘制的点会多，图上没有给出具体的排名，但是我们仍然能从图形上看出前十当中的哪个商品价格变化最大，其变化趋势对应图 24 与图 25：

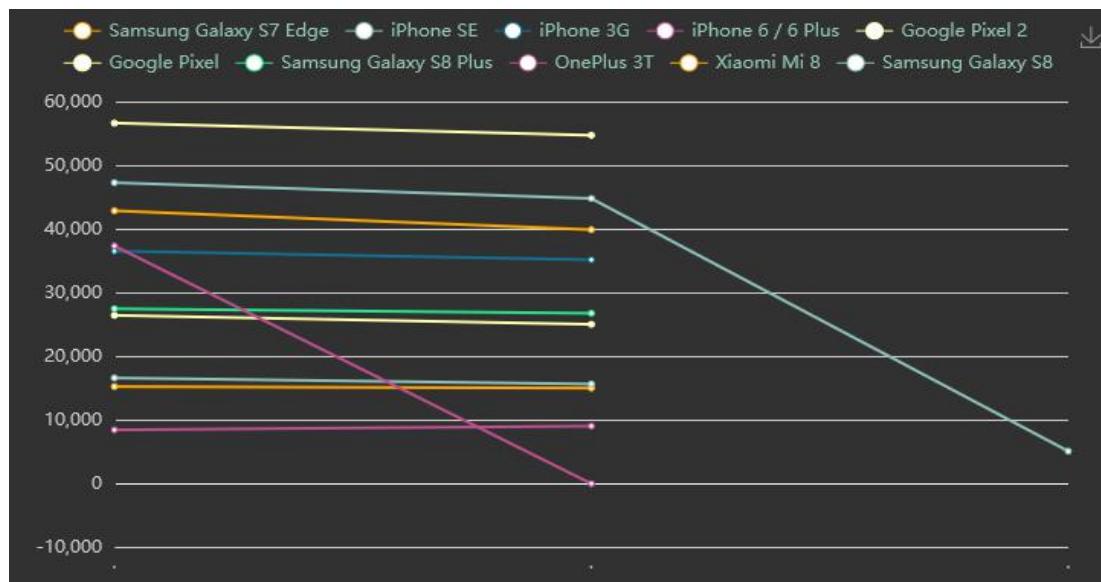


图 24 商品价格变化 1

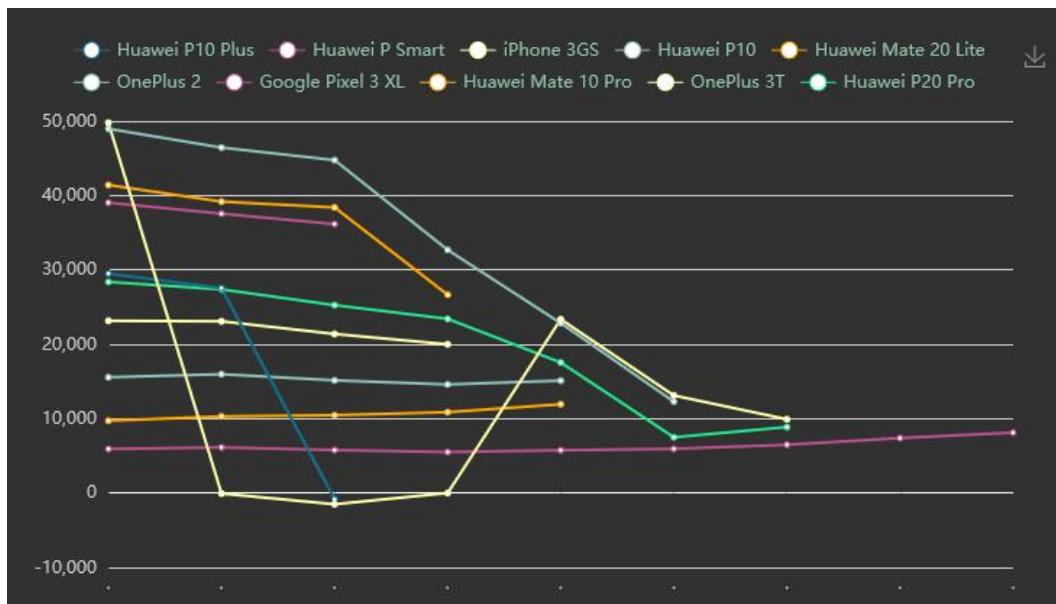


图 25 商品价格变化 2

6 总结与展望

6.1 总结

以上内容就是本系统的设计思路与功能实现，对一个典型的商城系统数据进行了简单的统计，并通过图表的形式实时展现给用户，所见即所得。该系统因为出于学习探究的目的混用了 Spark Streaming 与 Structured Streaming 两种处理方式，因此若真要将该系统用于实际的生产环境，还可以进一步优化成所有都是基于 Spark SQL 查询的处理形式。但此处个人时间与精力都有限，并没有做这一步。系统还有许多其它可以改进的地方，例如对用户最近在线上浏览的商品，商品分类等占比进行分析，得出用户在寻找哪类商品，然后进行精准的商品推荐。

此外，Structured Streaming 适合处理结构化的数据，但不适合处理无规则的数据，但是正如本文所示，我们可以通过 Spark Streaming 处理成结构化的数据再使用 Structured Streaming 的 API，这种混用的方式将会大大提升我们的流处理的场景适用性。

6.2 展望

在没有做该系统之前，我对大数据以及实时的认识都还比较模糊，但是做完以后我对这两者有了自己的认识和见解。大数据的数据是海量的，而我们所开发出来的大数据框架则让我们有了处理这些海量数据的能力，它不是虚无缥缈的，而是真正存在的，因为我们编程的时候面对的是具体的一些样本数据，但是只要根据这个写出来，就能完成超大规模的数据处理。而对实时性来说，我曾经认为实时就是要马上出结果，现在我觉得实时应该是一条数据产生就会被马上处理，相较于传统的要堆积到一定时间再进行批处

理，这种有数据产生即处理就是实时。基于此处两点我自己认识，我对实时大数据的发展有如下几个想法：

(1) 物联网

未来会是万物互联的时代，一个设备采集的数据将会实时上班，配合实时大数据处理，能有非常多的应用场景：交通道路规划、智能穿戴设备分析用户当前身体状态、公共区域实时分析监控图像预防犯罪的发生等。

(2) 服务监控日志分析

每一种在生产环境长期运行的服务都可能存在异常的情况，但是这些异常的日志，可能并没有引起服务完全挂掉，但是会使目标结果与预期结果不一致的情况。一般归档也不会去查询这些日志，所以服务很可能就会长期异常运行，而基于实时大数据平台可以开发出每生产一条日志消息就分析一条日志的应用，从而确保服务健康稳定长期运行。

(3) 精准内容推送

用户在浏览信息的时候会有一定的时间性，过了这段时间便会离开。在这段时间内如何用用户喜欢的内容吸引住用户尤为重要，若是基于历时数据进行推送，可能会有偏差，应当从用户最近的几个行为中进行实时分析处理，然后进行精准内容推送。

参 考 文 献

阿 里 巴 巴 . Canal 项 目 介 绍 [EB/OL]. 2018.

<https://github.com/alibaba/canal/wiki/Introduction>

阿里 巴巴. Canal Kafka RocketMQ QuickStart[EB/OL]. 2019.

<https://github.com/alibaba/canal/wiki/Canal-Kafka-RocketMQ-QuickStart>

Tom White. Hadoop 权威指南[M]. 华东师范大学数据科学与工程学院译, O'REILLY.

第

三版. 北京: 清华大学出版社, 2015: 10-11, 14-15, 19-24

岳猛. Apache 流框架 Flink, Spark Streaming, Storm 对比分析[EB/OL]. 2016.

<https://bigdata.163yun.com/product/article/5>

张安站.Spark 技术内幕：深入解析 Spark 内核架构设计与实现原理[M].北京：机械工业出

版社，2015: 17-18

Apache Spark. Spark Streaming Programming Guide[EB/OL]. 2018a.

<https://spark.apache.org/docs/2.4.0/streaming-programming-guide.html>

Apache Spark. Spark Streaming + Kafka Integration Guide (Kafka broker version 0.10.0 or

higher)[EB/OL]. 2018b.

<https://spark.apache.org/docs/2.4.0/streaming-kafka-0-10-integration.html>

Apache Spark. Structured Streaming Programming Guide[EB/OL]. 2018c.

<https://spark.apache.org/docs/2.4.0/structured-streaming-programming-guide.html>

Tathagata Das. Deep dive into Stateful Stream Processing in Structured Streaming[R]. Dublin:

Spark Summit Europe 2017, 2017

Tathagata Das, Shixiong Zhu. Faster Stateful Stream Processing in Apache Spark Streaming

[EB/OL]. 2016.

<https://databricks.com/blog/2016/02/01/faster-stateful-stream-processing-in-apache-spark-streaming.html>

致 谢

至此，本文也结束了，这也意味着我的大学生涯即将结束了。在这四年的大学生活中，有过迷茫，有过焦虑，不过最终都还是克服了。在此衷心的感谢任何帮助过我的人，我一定不忘初心，继续奋战。特别要感谢老师的耐心解答，在课题开始之前给了一些很不错的建议与学习资料，为我指明了研究的方向，少走了许多弯路。

其次感谢我的大学，以及我大学认识的所有人，是他们的帮助与支持才让我从众多困难中走出来。最后感谢我的父母，姑姑以及家里人，对我的体谅，是他们的信任，让我体验了一个无拘无束的大学生活，也是他们多年的付出，才有了今天的我。

大学是一个很美好的时期，这段时期我永生难忘，如果可以再读一次大学，我一定义无反顾把这四年没做好的事情补上，但是不可能了。不过在以后的日子，我一定会更加努力，变得更加优秀，因为此后社会不会像母校这样纵容我了。