大数据“重磅炸弹”——实时计算框架 Flink

前言

小田，你看能不能做个监控大屏实时查看促销活动销售额（GMV）？

小朱，搞促销活动的时候能不能实时统计下网站的 PV/UV 啊？

小鹏，我们现在搞促销活动能不能实时统计销量 Top5 啊？

小李，怎么回事啊？现在搞促销活动结果服务器宕机了都没告警，能不能加一个？

小刘，服务器这会好卡，是不是出了什么问题啊，你看能不能做个监控大屏实时查看机器的运行情况？

小赵，我们线上的应用频繁出现 Error 日志，但是只有靠人肉上机器查看才知道情况，能不能在出现错误的时候及时告警通知？

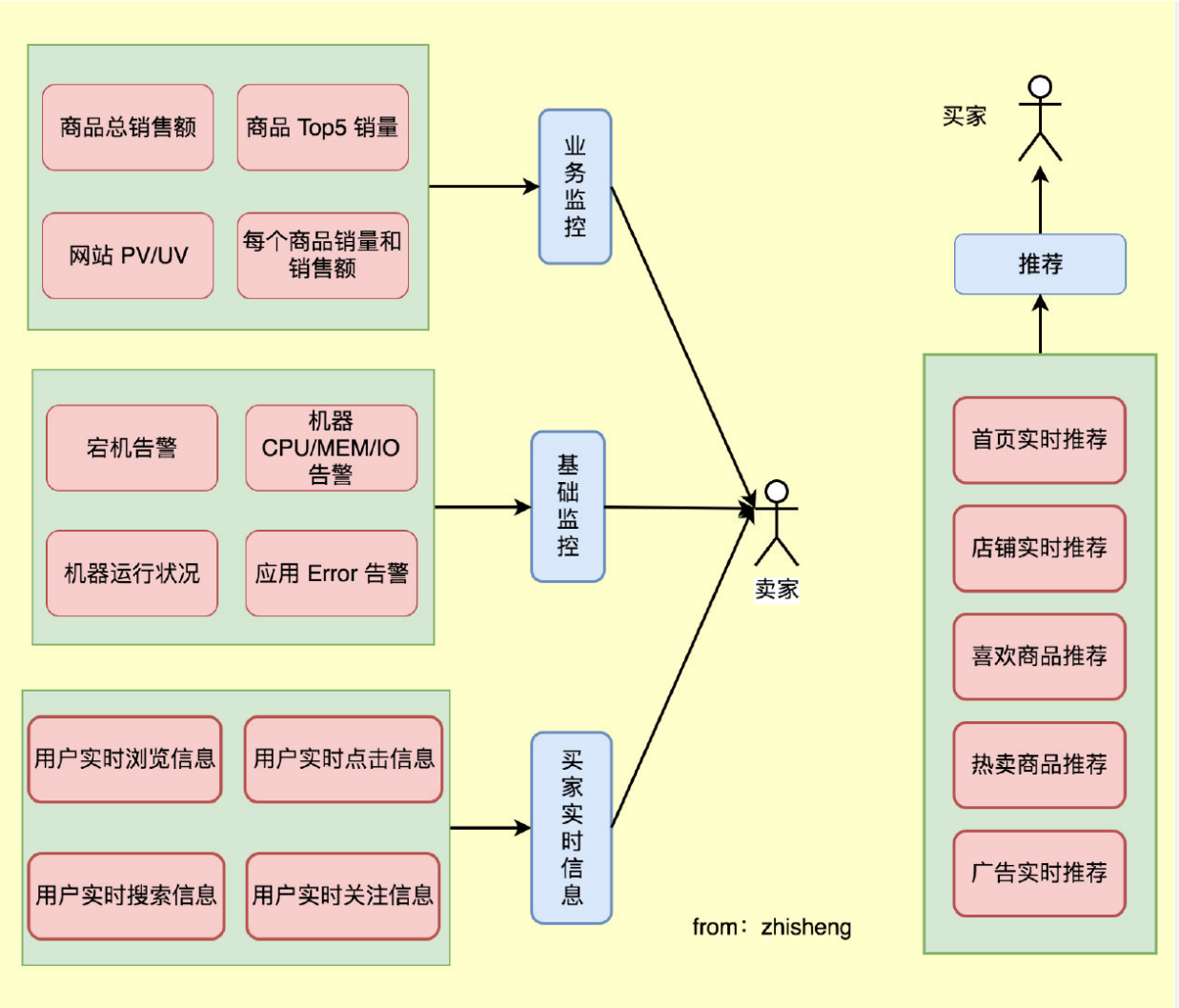
小夏，我们 1 元秒杀促销活动中有件商品被某个用户薅了 100 件，怎么都没有风控啊？

小宋，你看我们搞促销活动能不能根据每个顾客的浏览记录实时推荐不同的商品啊？

。。。

你是不是经常体验或看到上面这些场景？

那这些场景对应着什么业务需求呢？我们来总结下，大概如下：



初看这些需求，是不是感觉很难？

那么我们接下来来分析一下该怎么去实现？

从这些需求来看，最根本的业务都是需要**实时查看数据信息**，那么首先我们得想想如何去采集这些实时数据，然后将采集的实时数据进行实时的计算，最后将计算后的结果下发到第三方。

数据实时采集

就上面这些需求，我们需要采集些什么数据呢？

1、买家搜索记录信息

2、买家浏览的商品信息

3、买家下单订单信息

4、网站的所有浏览记录

5、机器 CPU/MEM/IO 信息

6、应用日志信息

数据实时计算

采集后的数据实时上报后，需要做实时的计算，那我们怎么实现计算呢?

1、计算所有商品的总销售额

2、统计单个商品的销量，最后求 Top5

3、关联用户信息和浏览信息、下单信息

4、统计网站所有的请求 IP 并统计每个 IP 的请求数量

5、计算一分钟内机器 CPU/MEM/IO 的平均值、75 分位数值

6、过滤出 Error 级别的日志信息

数据实时下发

实时计算后的数据，需要及时的下发到下游，这里说的下游代表可能是:

1、告警方式（邮件、短信、钉钉、微信）

在计算层会将计算结果与阈值进行比较，超过阈值触发告警，让运维提前收到通知，及时做好应对措施，减少故障的损失大小。

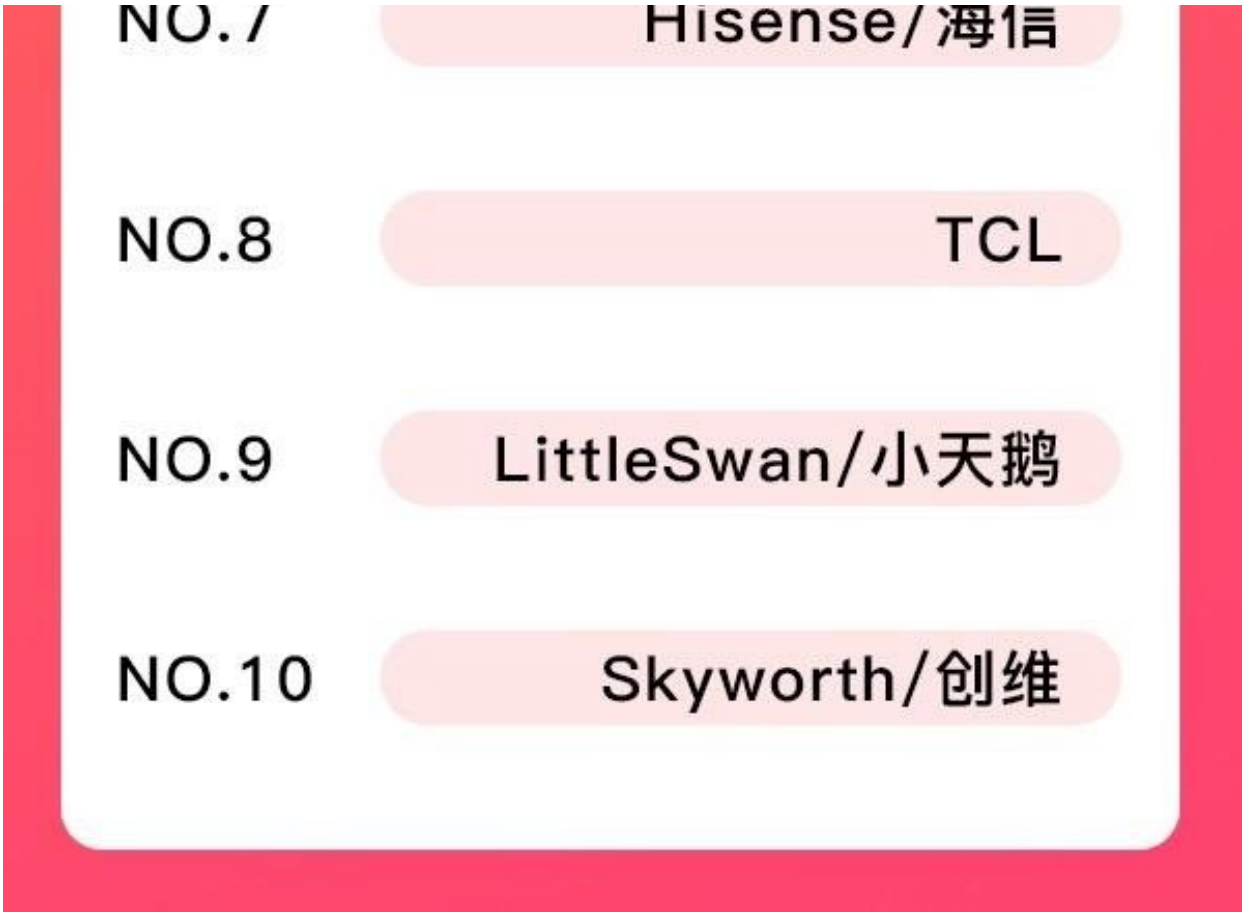
2、存储（消息队列、DB、文件系统等）

数据存储后，监控大盘（Dashboard）从存储（ElasticSearch、HBase 等）里面查询对应指标的数据就可以查看实时的监控信息，做到对促销活动的商品销量、销售额，机器 CPU、MEM 等有实时监控，运营、运维、开发、领导都可以实时查看并作出对应的措施。

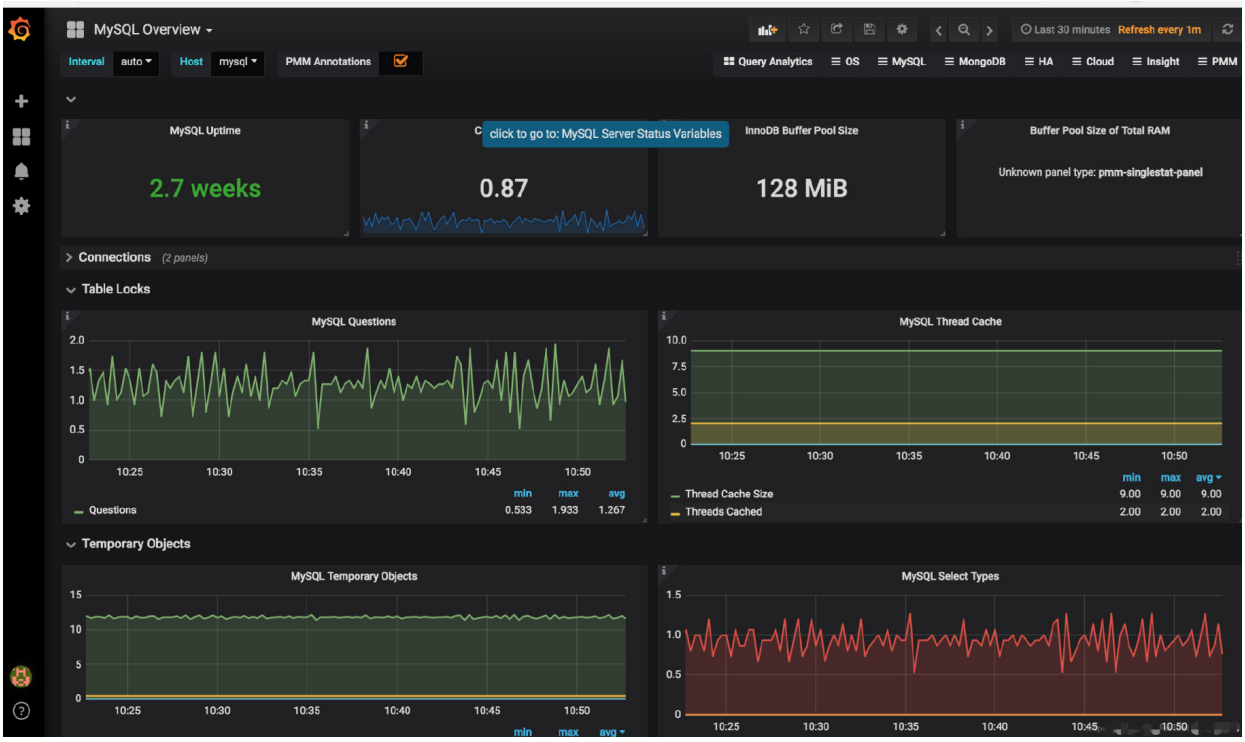
- 让运营知道哪些商品是爆款，哪些店铺成交额最多，哪些商品成交额最高，哪些商品浏览量最

多；


2018年天猫双11
24小时TOP品牌榜
大家电
NO.1
Haier/海尔
NO.2
Midea/美的v
NO.3
Xiaomi/小米
NO.4
GREE/格力
NO.5
西门子家电
NO.6
AUX/奥克斯

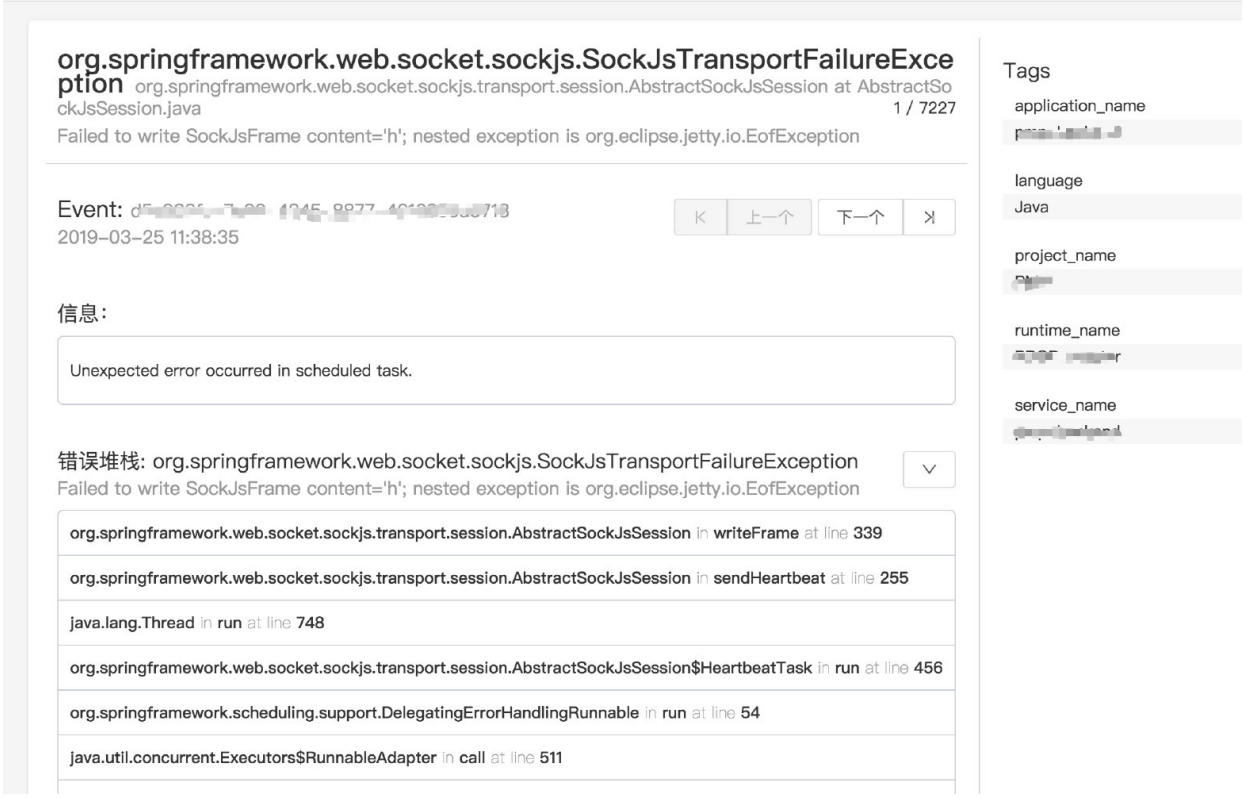


- 让运维可以时刻了解机器的运行状况，出现宕机或者其他不稳定情况可以及时处理；



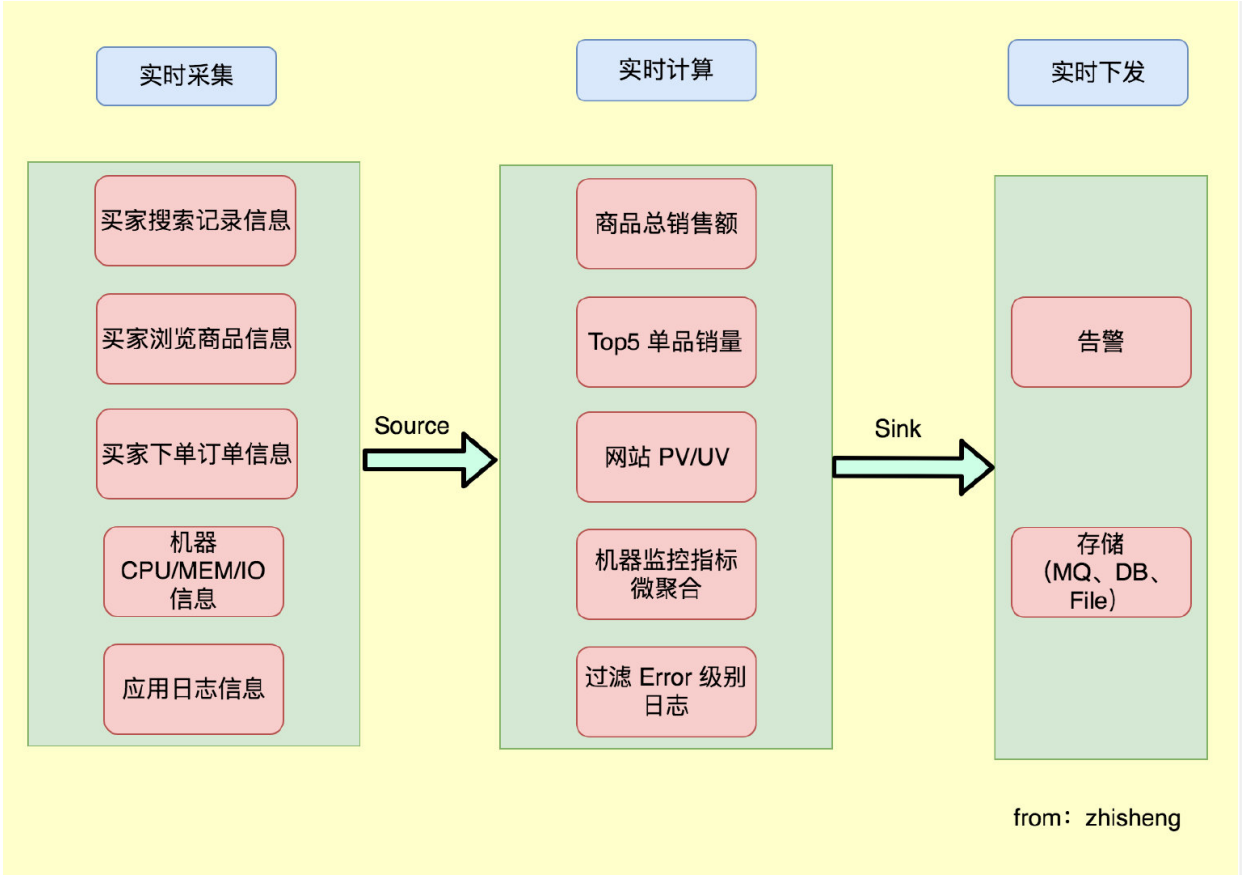
- 让开发知道自己项目运行的情况，从 Error 日志知道出现了哪些 Bug；

错误详情



- 让领导知道这次促销赚了多少 money。

从数据采集到数据计算再到数据下发，整个流程在上面的场景对实时性要求还是很高的，任何一个地方出现问题都将影响最后的效果！

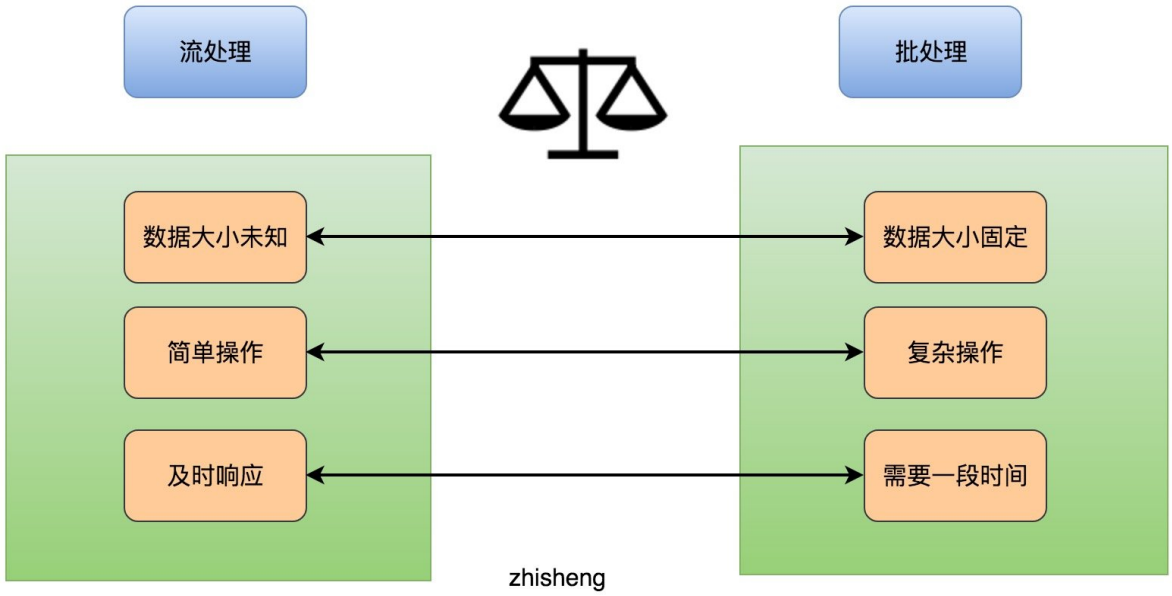


说到实时计算，这里不得不讲一下和传统的离线计算的区别：

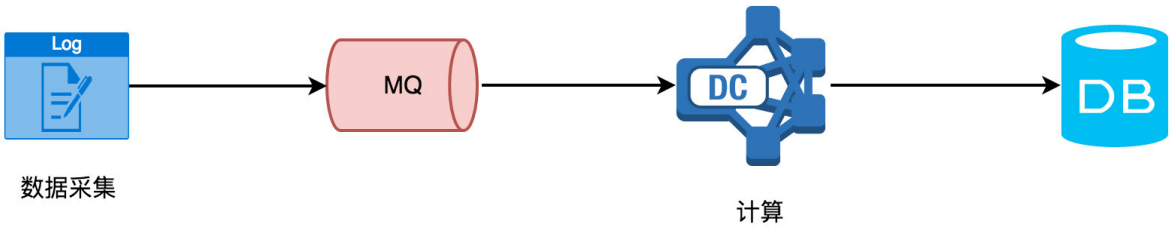
实时计算 VS 离线计算

再讲这两个区别之前，我们先来看看流处理和批处理的区别：

流处理与批处理：

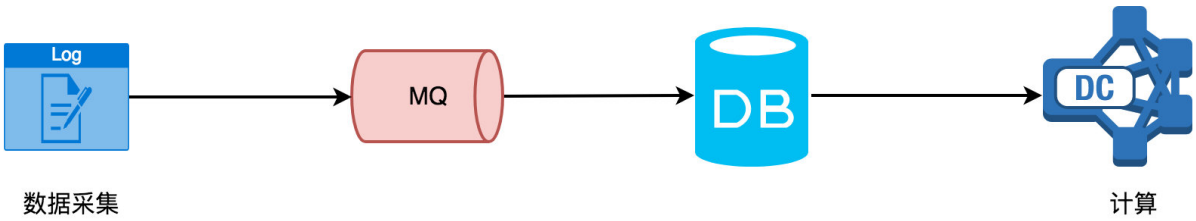


看完流处理与批处理这两者的区别之后，我们来抽象一下前面文章的场景需求（实时计算）：



实时计算需要不断的从 MQ 中读取采集的数据，然后处理计算后往 DB 里存储，在计算这层你无法感知到会有多少数据量过来、要做一些简单的操作（过滤、聚合等）、及时将数据下发。

相比传统的离线计算，它却是这样的：



在计算这层，它从 DB（不限 MySQL，还有其他的存储介质）里面读取数据，该数据一般就是固定的（前一天、前一星期、前一个月），然后再做一些复杂的计算或者统计分析，最后生成可供直观查看的报表（dashboard）。

离线计算有它自己的特点：

1、数据量大且时间周期长（一天、一星期、一个月、半年、一年）

2、在大量数据上进行复杂的批量运算

3、数据在计算之前已经固定，不再会发生变化

4、能够方便的查询批量计算的结果

实时计算特点：

在大数据中与离线计算对应的则是实时计算，那么实时计算有什么特点呢？由于应用场景的各不相同，所以这两种计算引擎接收数据的方式也不太一样：离线计算的数据是固定的（不再会发生变化），通常离线计算的任务都是定时的，如：每天晚上 0 点的时候定时计算前一天的数据，生成报表；然而实时计算的数据源却是流式的。

这里我不得不讲讲什么是流式数据呢？我的理解是比如你在淘宝上下单了某个商品或者点击浏览了某件商品，你就会发现你的页面立马就会给你推荐这种商品的广告和类似商品的店铺，这种就是属于实时数据处理然后作出相关推荐，这类数据需要不断的从你在网页上的点击动作中获取数据，之后进行实时分析然后给出推荐。

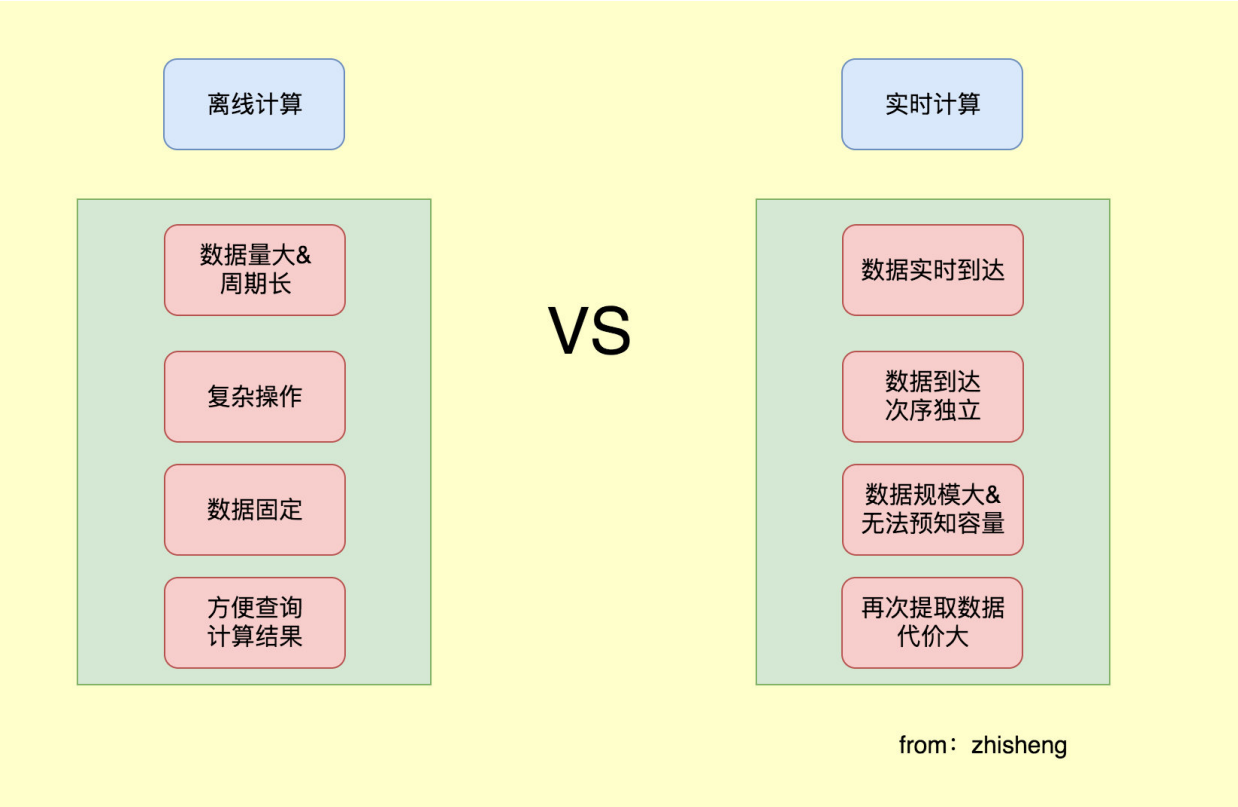
流式数据具有的特点：

1、数据实时到达

2、数据到达次序独立，不受应用系统所控制

3、数据规模大且无法预知容量

4、原始数据一经处理，除非特意保存，否则不能被再次取出处理，或者再次提取数据代价昂贵



实时计算的优势：

实时计算一时爽，一直实时计算一直爽，对于持续生成最新数据的场景，采用流数据处理是非常有利的。例如，再监控服务器的一些运行指标的时候，能根据采集上来的实时数据进行判断，当超出一定阈值的时候发出警报，进行提醒作用。再如通过处理流数据生成简单的报告，如五分钟的窗口聚合数据平均值。复杂的事情还有在流数据中进行数据多维度关联、聚合、塞选，从而找到复杂事件中的根因。更为复杂的是做一些复杂的数据分析操作，如应用机器学习算法，然后根据算法处理后的数据结果提取出有效的信息，作出、给出不一样的推荐内容，让不同的人可以看见不同的网页（千人千面）。

实时计算场景

前面说了这么多场景，这里我们总结一下实时计算常用的场景有哪些呢?

1、交通信号灯数据

2、道路上车流量统计（拥堵状况）

3、公安视频监控

4、服务器运行状态监控

5、金融证券公司实时跟踪股市波动，计算风险价值

6、数据实时 ETL

7、银行或者支付公司涉及金融盗窃的预警

。。。

另外我自己在我的群里也有做过调研（不完全统计），他们在公司 Flink（一个实时计算框架）使用场景有这些：

总结一下大概有下面这四类：

1、实时数据存储

实时数据存储的时候做一些微聚合、过滤某些字段、数据脱敏，组建数据仓库，实时 ETL。

2、实时数据分析

实时数据接入机器学习框架（TensorFlow）或者一些算法进行数据建模、分析，然后动态的给出商品推荐、广告推荐

3、实时监控告警

金融相关涉及交易、实时风控、车流量预警、服务器监控告警、应用日志告警

4、实时数据报表

活动营销时销售额/销售量大屏，TopN 商品

使用实时数据流面临的挑战

1、数据处理唯一性（如何保证数据只处理一次？至少一次？最多一次？）

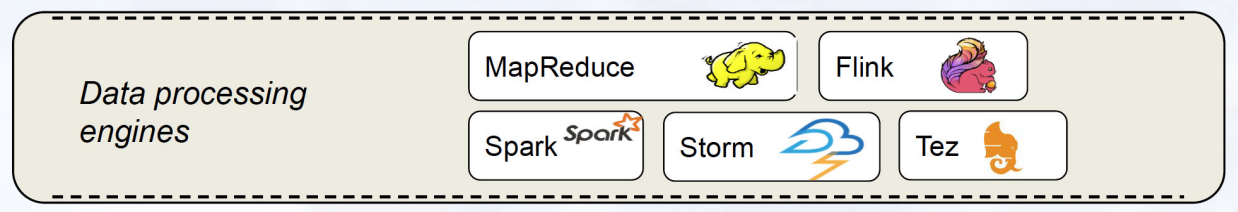
2、数据处理的及时性（采集的实时数据量太大的话可能会导致短时间内处理不过来，如何保证数据能够及时的处理，不出现数据堆积？）

3、数据处理层和存储层的可扩展性（如何根据采集的实时数据量的大小提供动态扩缩容？）

4、数据处理层和存储层的容错性（如何保证数据处理层和存储层高可用，出现故障时数据处理层和存储层服务依旧可用？）

实时计算框架介绍

目前市面上有 Flink、Spark Streaming、Storm、Kafka Stream，网上讲这几个区别的文章也有不少，不过我保持 **不同框架有不同的场景，没有绝对的牛逼，也没有绝对的 low 比，历史总在演进，新的框架会逐步把老的框架拍死在沙滩上** 观点。

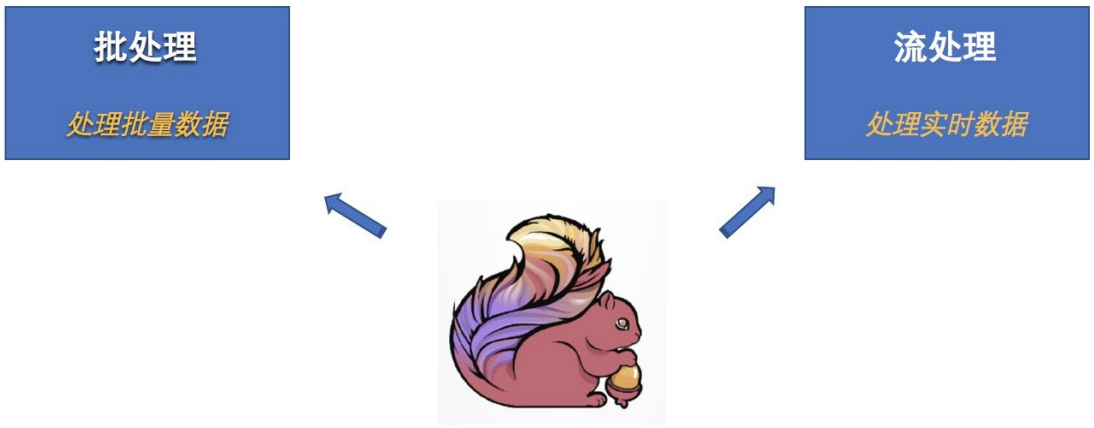


这篇文章我就只讲一下 Flink，因为后面我会在 GitChat 开一门 Flink 专题的课程，主要是通过多个实战案例带大家深入理解 Flink 。

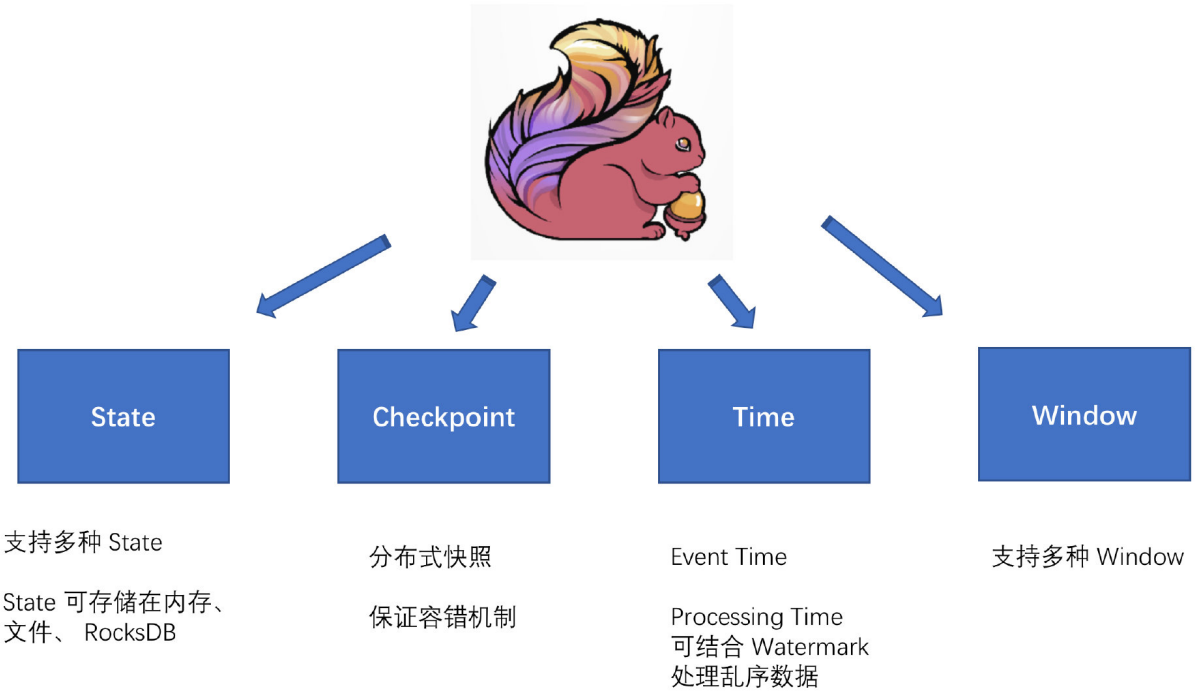
Flink 介绍

其实之前我已经在自己的博客里面写过一篇介绍 Flink 的文章: 《从0到1学习Flink》—— Apache Flink 介绍，感兴趣的可以查看一下，下面我们讲解一下其中的几点比较重要的特性。

1、在 Flink 中，它不仅可以处理批数据，也可以处理流数据

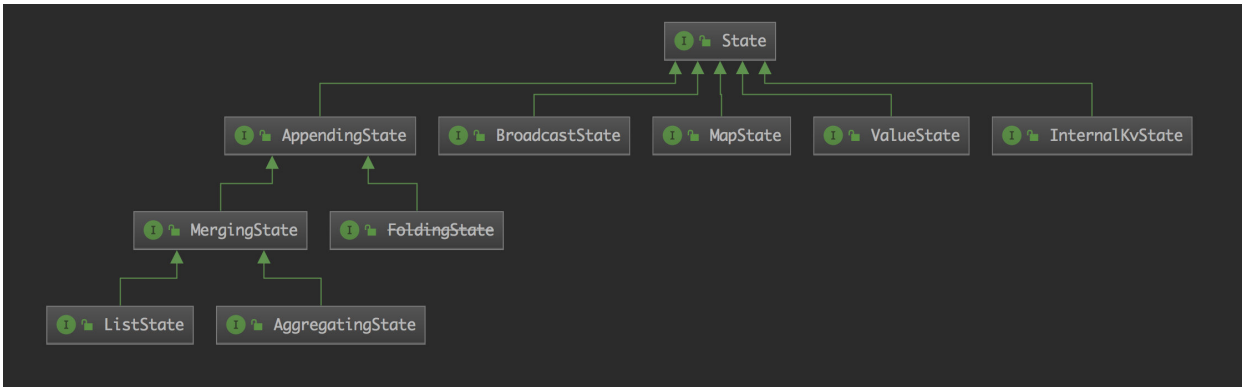


2、Flink 提供了 State、Checkpoint、Time、Window



多种 State（ListState、MapState、ValueState、BroadcastState、InternalKvState 等）

Flink 中自带了多种 State，并且 State 可以存储在内存、文件、RocksDB 中。



Checkpoint

Checkpoint 是 Flink 实现容错机制最核心的功能，它能够根据配置周期性地基于 Stream 中各个 Operator 的状态来生成 Snapshot，从而将这些状态数据定期持久化存储下来，当 Flink 程序一旦意外崩溃时，重新运行程序时可以有选择地从这些 Snapshot 进行恢复，从而修正因为故障带来的程序数据状态中断。

另外 Flink 中也有 Savepoint，它会在 Flink Job 之外存储自包含结构的 Checkpoint，它使用 Flink 的 Checkpointing 机制来创建一个非增量的 Snapshot，里面包含 Streaming 程序的状态，并将 Checkpoint 的数据存储到外部存储系统中。

Time

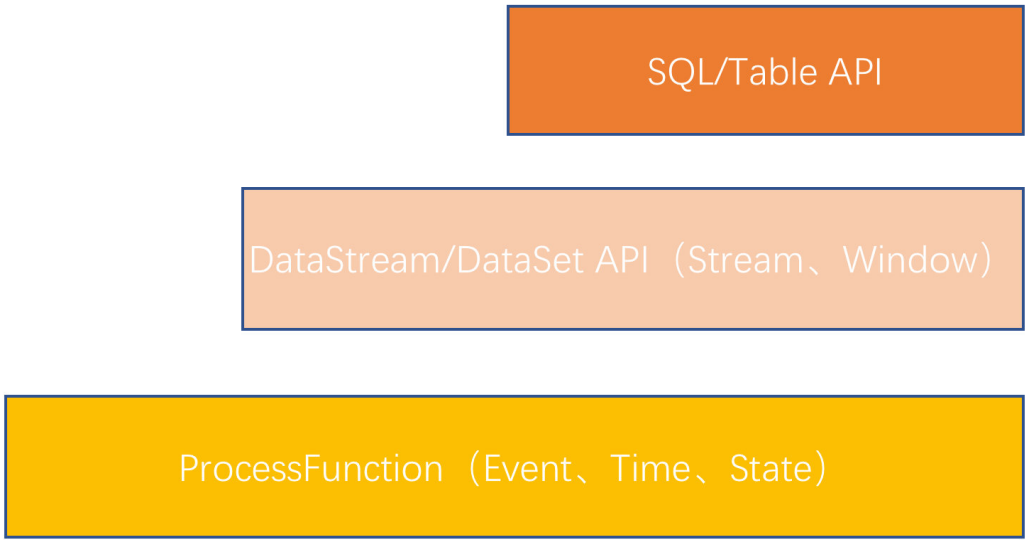
Flink 中的 Time 有三种 Processing Time、Event Time、Ingestion Time，你可以看下我之前的文章：《从0到1学习Flink》—— Flink 中几种 Time 详解，通过 Time 并结合 Watermark 可以处理乱序的数据（延迟到达）。

Window

Flink 中同样也是自带了很多 Window，比如：Time Window、Count Window 等

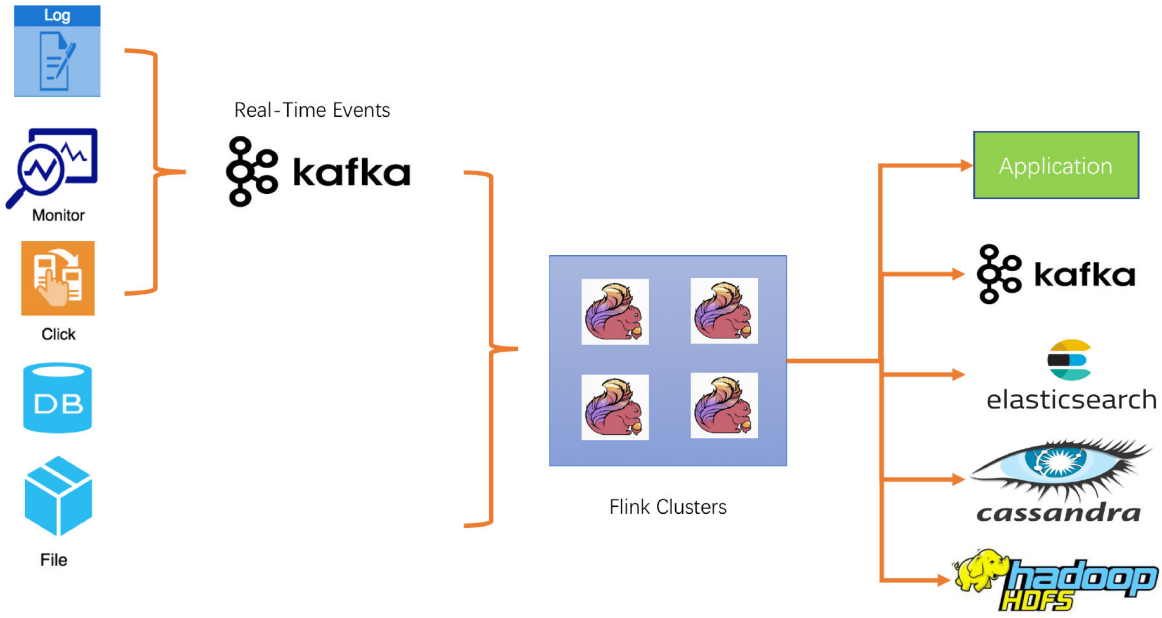
你可以参考我之前的文章：《从0到1学习Flink》—— 介绍Flink中的Stream Windows

3、Flink API



4、Flink 扮演的角色

如下图所示，Flink 将采集后的数据进行一系列的处理后然后往下发：



总结

本文从日常老板的需求口吻来讲解现在越来越多的实时性要求高的需求，并将这些需求做了个归纳统计，然后从需求里面得到了该如何去实现这类需求，是需要实时采集、实时计算、实时下发，并用图片把需求完成后的效果图展示了出来，接着我们分析了对实时性要求高的计算这块，然后将离线计算与实时计算进行了对比，批处理与流处理进行对比，离线计算的特点与实时计算的特点，加上我自己的调研结果，归纳了实时计算的四种使用场景，提出了使用实时计算时要面临的挑战，因为各种需求，也就造就了现在出现不断的实时计算框架，接着看了下市场上所有的实时框架，但是因为这类对比的文章网上比较多，因此我只介绍了 Flink 的特性和其 API，毕竟后面 Flink 才是我们专题的主场。

通过这篇文章的学习，你可以知道实时计算有哪些场景，你的公司这些场景是不是也可以换成 Flink 来做？同时也知道了实时计算与离线计算的区别，并初步认识了一下这个好玩好用的实时计算框架——Flink。

今天我们就到这里为此，感兴趣的可以持续关注我们后面的 Flink 专栏文章，后面系列文章中我们通过多个实战和大型案例来讲解 Flink，也欢迎在文末交流沟通你们公司是不是也在使用 Flink？使用 Flink 遇到什么坑？大概哪些场景使用到了 Flink？有没有更多的场景可以使用 Flink？