知乎 Flink 取代 Spark Streaming 的实战之路

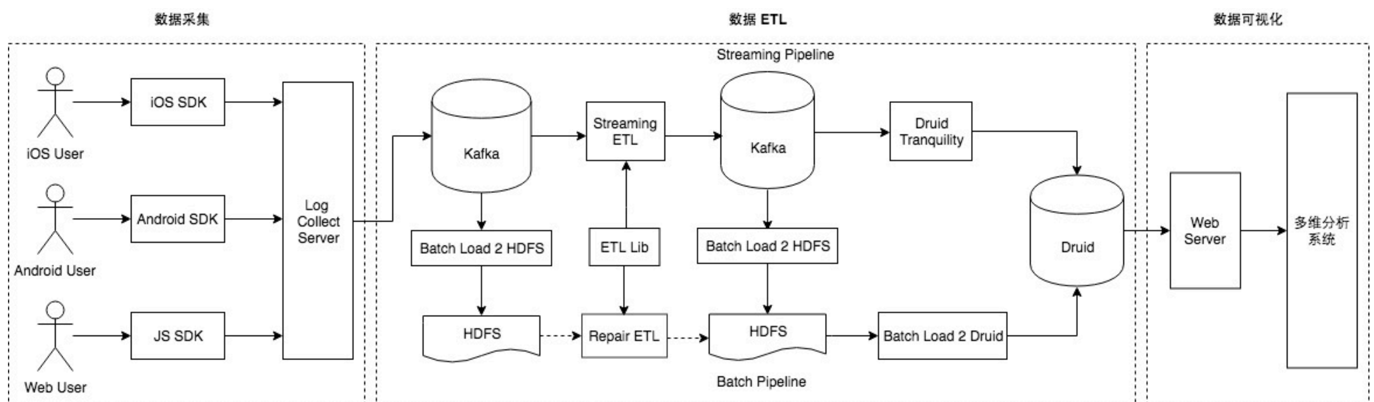
“数据智能” (Data Intelligence) 有一个必须且基础的环节，就是数据仓库的建设，同时，数据仓库也是公司数据发展到一定规模后必然会提供的一种基础服务。从智能商业的角度来讲，数据的结果代表了用户的反馈，获取结果的及时性就显得尤为重要，快速的获取数据反馈能够帮助公司更快的做出决策，更好的进行产品迭代，实时数仓在这一过程中起到了不可替代的作用。

本文主要讲述知乎的实时数仓实践以及架构的演进，这包括以下几个方面

- 实时数仓 1.0 版本，主题： ETL 逻辑实时化，技术方案：Spark Streaming。
- 实时数仓 2.0 版本，主题：数据分层，指标计算实时化，技术方案：Flink Streaming。
- 实时数仓未来展望：Streaming SQL 平台化，元信息管理系统化，结果验收自动化。

实时数仓 1.0 版本

1.0 版本的实时数仓主要是对流量数据做实时 ETL，并不计算实时指标，也未建立起实时数仓体系，实时场景比较单一，对实时数据流的处理主要是为了提升数据平台的服务能力。实时数据的处理向上依赖数据的收集，向下关系到数据的查询和可视化，下图是实时数仓 1.0 版本的整体数据架构图。



如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

第一部分是数据采集，由三端 SDK 采集数据并通过 Log Collector Server 发送到 Kafka。第二部分是数据 ETL，主要完成对原始数据的清洗和加工并分实时和离线导入 Druid。第三部分是数据可视化，由 Druid 负责计算指标并通过 Web Server 配合前端完成数据可视化。

其中第一、三部分的相关内容请分别参考：知乎客户端埋点流程、模型和平台技术，Druid 与知乎数据分析平台，此处我们详细介绍第二部分。由于实时数据流的稳定性不如离线数据流，当实时流出现问题后需要离线数据重刷历史数据，因此实时处理部分我们采用了 lambda 架构。

Lambda 架构有高容错、低延时和可扩展的特点，为了实现这一设计，我们将 ETL 工作分为两部分：Streaming ETL 和 Batch ETL。

Streaming ETL

这一部分我会介绍实时计算框架的选择、数据正确性的保证、以及 Streaming 中一些通用的 ETL 逻辑，最后还会介绍 Spark Streaming 在实时 ETL 中的稳定性实践。

计算框架选择

在 2016 年年初，业界用的比较多的实时计算框架有 Storm 和 Spark Streaming。Storm 是纯流式框架，Spark Streaming 用 Micro Batch 模拟流式计算，前者比后者更实时，后者比前者吞吐量大且生态系统更完善，考虑到知乎的日志量以及初期对实时性的要求，我们选择了 Spark Streaming 作为实时数据的处理框架。

数据正确性保证

Spark Streaming 的端到端 Exactly-once 需要下游支持幂等、上游支持流量重放，这里我们在 Spark Streaming 这一层做到了 At-least-once，正常情况下数据不重不少，但在程序重启时可能会重发部分数据，为了实现全局的 Exactly-once，我们在下游做了去重逻辑，关于如何去重后面我会讲到。

通用 ETL 逻辑

ETL 逻辑和埋点的数据结构息息相关，我们所有的埋点共用同一套 Proto Buffer Schema，大致如下所示。

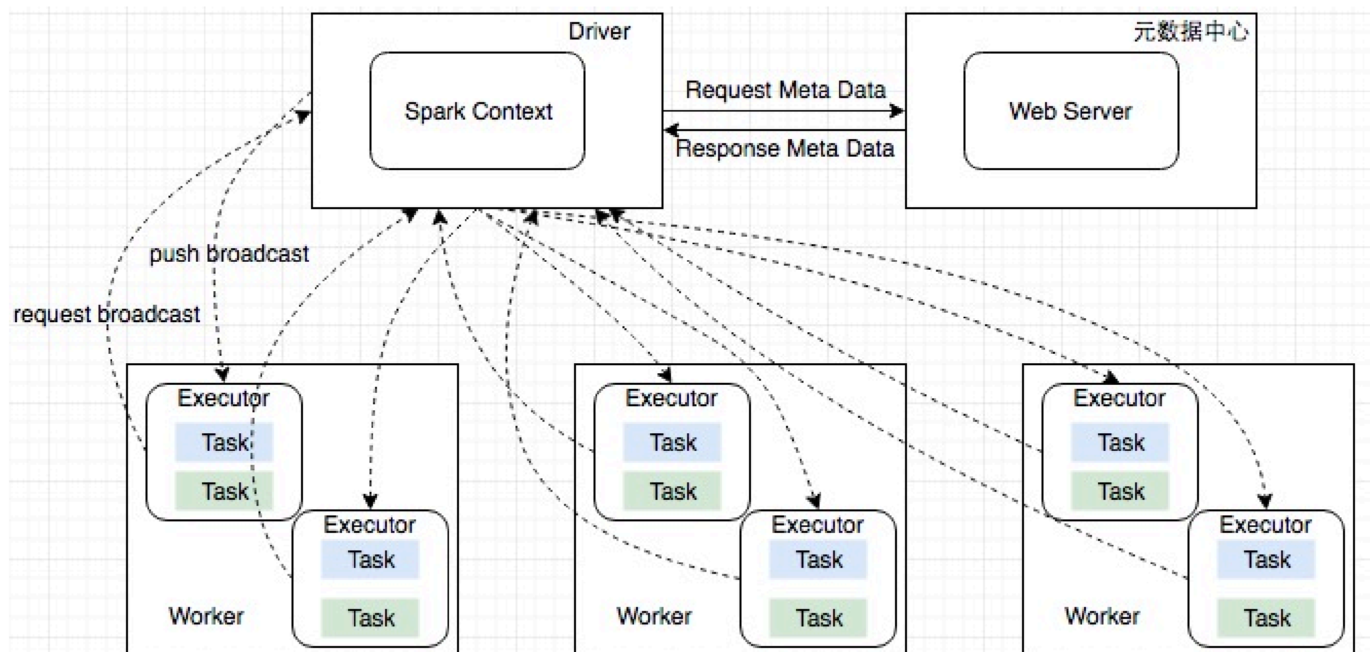
```
message LogEntry {
  optional BaseInfo base = 1;
  optional DetailInfo detail = 2;
  optional ExtraInfo extra = 3;
}
```

BaseInfo：日志中最基本的信息，包括用户信息、客户端信息、时间信息、网络信息等日志发送时的必要信息。DetailInfo：日志中的视图信息，包括当前视图、上一个视图等用于定位用户所在位置的信息。ExtraInfo：日志中与特定业务相关的额外信息。

针对上述三种信息我们将 ETL 逻辑分为通用和非通用两类，通用逻辑和各个业务相关，主要应用于 Base 和 Detail 信息，非通用逻辑则是由需求方针对某次需求提出，主要应用于 Extra 信息。这里我们列举 3 个通用逻辑进行介绍，这包括：动态配置 Streaming、UTM 参数解析、新老用户识别。

动态配置 Streaming

由于 Streaming 任务需要 7 * 24 小时运行，但有些业务逻辑，比如：存在一个元数据信息中心，当这个元数据发生变化时，需要将这种变化映射到数据流上方便下游使用数据，这种变化可能需要停止 Streaming 任务以更新业务逻辑，但元数据变化的频率非常高，且在元数据变化后如何及时通知程序的维护者也很难。动态配置 Streaming
为我们提供了一个解决方案，该方案如下图所示。

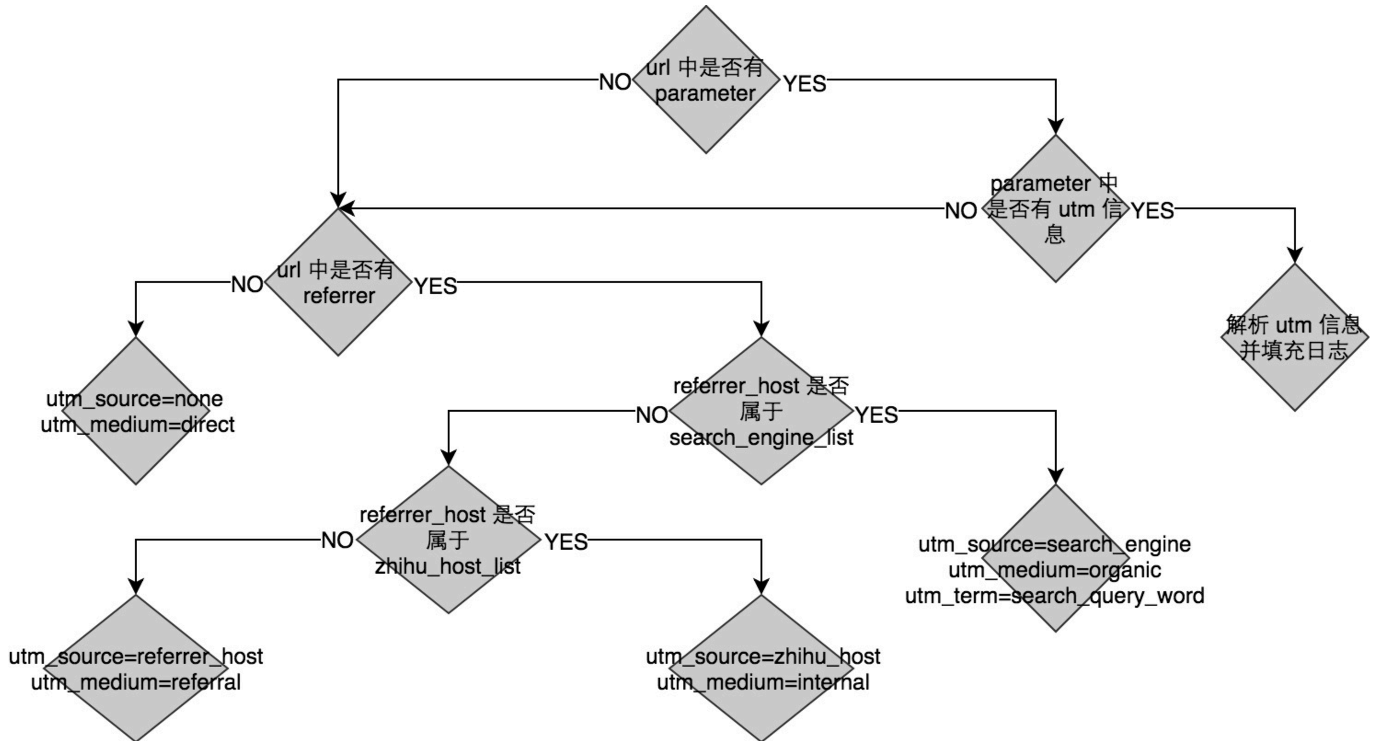


如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

我们可以把经常变化的元数据作为 Streaming Broadcast
变量，该变量扮演的角色类似于只读缓存，同时针对该变量可设置 TTL，缓存过期后 Executor 节点会重新向 Driver 请求最新的变量。通过这种机制可以非常自然的将元数据的变化映射到数据流上，无需重启任务也无需通知程序的维护者。

UTM 参数解析

UTM 的全称是 Urchin Tracking Module，是用于追踪网站流量来源的利器，关于 UTM
背景知识介绍可以参考网上其他内容，这里不再赘述。下图是我们解析 UTM 信息的完整逻辑。



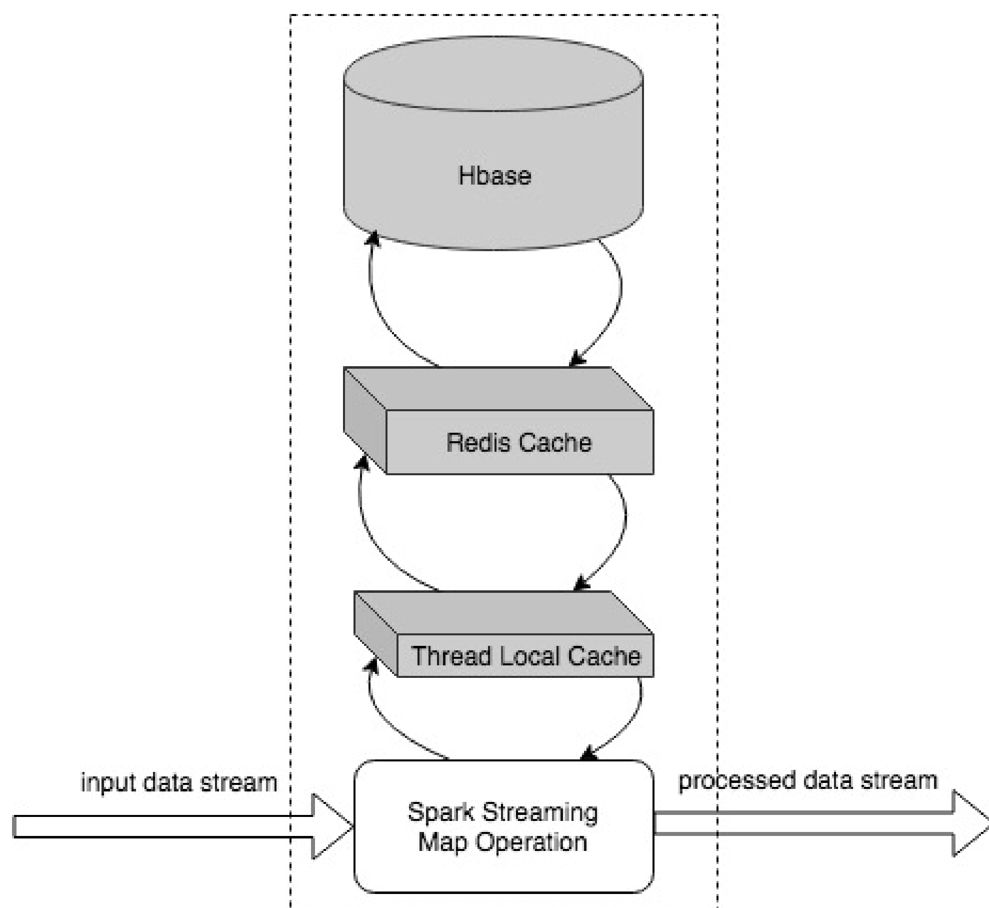
如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

流量数据通过 UTM 参数解析后，我们可以很容易满足以下需求

- 查看各搜索引擎导流情况以及这些流量来自于哪些热门搜索词。
- 市场部某次活动带来的流量大小，如：页面浏览数、独立访问用户数等。
- 从站内分享出去的链接在各分享平台（如：微信、微博）被浏览的情况。

新老用户识别

对于互联网公司而言，增长是一个永恒的话题，实时拿到新增用户量，对于增长运营十分重要。例如：一次投放 n 个渠道，如果能拿到每个渠道的实时新增用户数，就可以快速判断出那些渠道更有价值。我们用下图来表达 Streaming ETL 中是如何识别新老用户的。



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

判断一个用户是不是新用户，最简单的办法就是维护一个历史用户池，对每条日志判断该用户是否存在于用户池中。 由于日志量巨大，为了不影响 Streaming
任务的处理速度，我们设计了两层缓存：Thread Local Cache 和 Redis Cache，同时用 HBase
做持久化存储以保存历史用户。访问速度：本地内存 > 远端内存 >
远端磁盘，对于我们这个任务来说，只有 1% 左右的请求会打到 HBase，日志高峰期
26w/s，完全不会影响任务的实时性。当然本地缓存 LruCache 的容量大小和 Redis
的性能也是影响实时性的两个因素。

Streaming ETL 除了上述几个通用场景外，还有一些其他逻辑，这些逻辑的存在有的是为了满足下游更方便的使用数据的需求，有的是对某些错误埋点的修复，总之 Streaming ETL
在整个实时数仓中处于指标计算的上游，有着不可替代的作用。

Spark Streaming 在实时数仓 1.0 中的稳定性实践

Spark Streaming 消费 Kafka 数据推荐使用 Direct 模式。我们早期使用的是 High Level 或者叫 Receiver 模式并使用了 checkpoint 功能，这种方式在更新程序逻辑时需要删除 checkpoint 否则新的程序逻辑就无法生效。另外，由于使用了 checkpoint 功能，Streaming 任务会保持和 Hdfs 通信，可能会因为 NameNode 的抖动导致 Streaming 任务抖动。因此，推荐使用 Direct 模式，关于这种模式和 Receiver 模式的详细对比，可以参考官方文档。

保证 Spark Streaming 任务的资源稳定。以 Yarn 为例，运行 Streaming 任务的队列能够分配到的最小资源小于了任务所需要的资源，任务会出现频繁丢失 Executor 的情况，这会导致 Streaming 任务变慢，因为丢失的 Executor 所对应的数据需要重新计算，同时还需要重新分配 Executor。

Spark Streaming 消费 Kafka 时需要做数据流限速。默认情况下 Spark Streaming 以尽可能大的速度读取消息队列，当 Streaming 任务挂了很久之后再次被启动时，由于拉取的数据量过大可能会导致上游的 Kafka 集群 IO 被打爆进而出现 Kafka 集群长时间阻塞。可以使用 Streaming Conf 参数做限速，限定每秒拉取的最大速度。

Spark Streaming 任务失败后需要自动拉起。长时间运行发现，Spark Streaming 并不能 7 * 24h 稳定运行，我们用 Supervisor 管理 Driver 进程，当任务挂掉后 Driver 进程将不复存在，此时 Supervisor 将重新拉起 Streaming 任务。

Batch ETL

接下来要介绍的是 Lambda 架构的第二个部分：Batch ETL，此部分我们需要解决数据落地、离线 ETL、数据批量导入 Druid 等问题。针对数据落地我们自研了 map reduce 任务 Batch Loader，针对数据修复我们自研了离线任务 Repair ETL，离线修复逻辑和实时逻辑共用一套 ETL Lib，针对批量导入 ProtoParquet 数据到 Druid，我们扩展了 Druid 的导入插件。

Repair ETL

数据架构图中有两个 Kafka，第一个 Kafka 存放的是原始日志，第二个 Kafka 存放的是实时 ETL 后的日志，我们将两个 Kafka 的数据全部落地，这样做的目的是为了保证数据链路的稳定性。因为实时 ETL 中有大量的业务逻辑，未知需求的逻辑也许会给整个流量数据带来安全隐患，而上游的 Log Collect Server 不存在任何业务逻辑只负责收发日志，相比之下第一个 Kafka 的数据要安全和稳定的多。Repair ETL 并不是经常启用，只有当实时 ETL 丢失数据或者出现逻辑错误时，才会启用该程序用于修复日志。

Batch Load 2 HDFS

前面已经介绍过，我们所有的埋点共用同一套 Proto Buffer Schema，数据传输格式全部为二进制。我们自研了落地 Kafka PB 数据到 Hdfs 的 Map Reduce 任务 BatchLoader，该任务除了落地数据外，还负责对数据去重。在 Streaming ETL 阶段我们做到了 At-least-once，通过此处的 BatchLoader 去重我们实现了全局 Exactly-once。BatchLoader 除了支持落地数据、对数据去重外，还支持多目录分区（p_date/p_hour/p_plaform/p_logtype）、数据回放、自依赖管理（早期没有统一的调度器）等。截止到目前，BatchLoader 落地了 40+ 的 Kakfa Topic 数据。

Batch Load 2 Druid

采用 Tranquility 实时导入 Druid，这种方式强制需要一个时间窗口，当上游数据延迟超过窗值后会丢弃窗口之外的数据，这种情况会导致实时报表出现指标错误。为了修复这种错误，我们通过 Druid 发起一个离线 Map Reduce 任务定期重导上一个时间段的数据。通过这里的 Batch 导入和前面的实时导入，实现了实时数仓的 Lambda 架构。

实时数仓 1.0 的几个不足之处

到目前为止我们已经介绍完 Lambda 架构实时数仓的几个模块，1.0 版本的实时数仓有以下几个不足

- 所有的流量数据存放在同一个 Kafka Topic 中，如果下游每个业务线都要消费，这会导致全量数据被消费多次，Kafka 出流量太高无法满足该需求。
- 所有的指标计算全部由 Druid 承担，Druid 同时兼顾实时数据源和离线数据源的查询，随着数据量的暴涨 Druid 稳定性急剧下降，这导致各个业务的核心报表不能稳定产出。
- 由于每个业务使用同一个流量数据源配置报表，导致查询效率低下，同时无法对业务做数据隔离和成本计算。

实时数仓 2.0 版本

随着数据量的暴涨，Druid 中的流量数据源经常查询超时同时各业务消费实时数据的需求也开始增多，如果继续沿用实时数仓 1.0 架构，需要付出大量的额外成本。于是，在实时数仓 1.0 的基础上，我们建立起了实时数仓 2.0，梳理出了新的架构设计并开始着手建立实时数仓体系，新的架构如下图所示。

如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

原始层

实时数仓 1.0 我们只对流量数据做 ETL 处理，在 2.0 版本中我们加入了对业务库的变更日志 Binlog 的处理，Binlog 日志在原始层为库级别或者 Mysql 实例级别，即：一个库或者实例的变更日志存放在同一个 Kafka Topic 中。同时随着公司业务的发展不断有新 App 产生，在原始层不仅采集「知乎」日志，像知乎极速版以及内部孵化项目的埋点数据也需要采集，不同 App 的埋点数据仍然使用同一套 PB Schema。

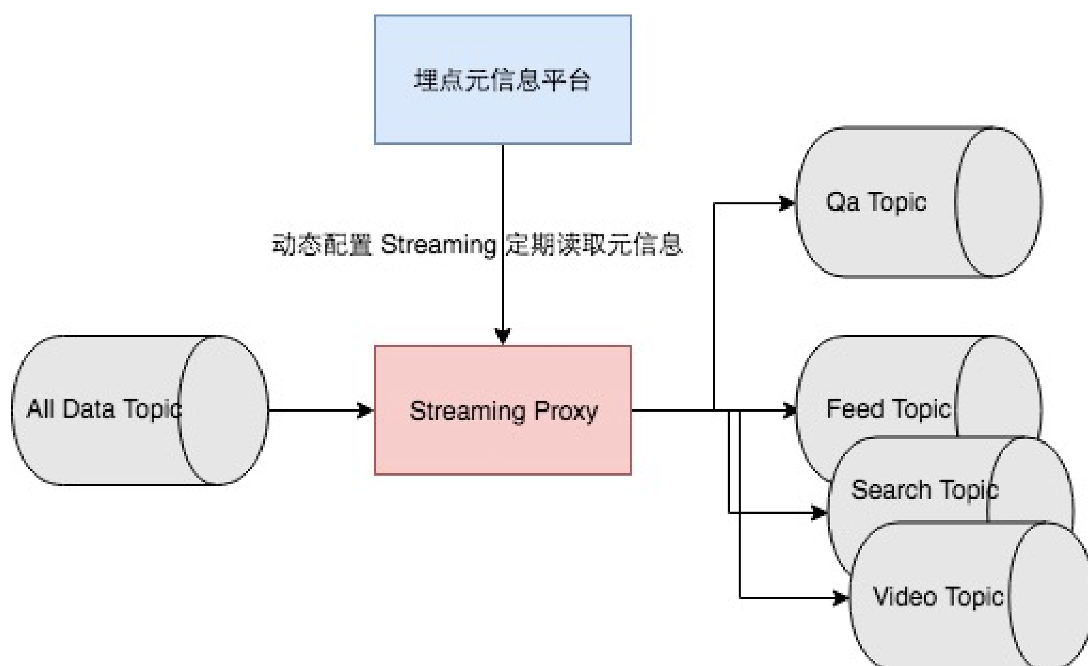
明细层

明细层是我们的 ETL 层，这一层数据是由原始层经过 Streaming ETL 后得到。其中对 Binlog 日志的处理主要是完成库或者实例日志到表日志的拆分，对流量日志主要是做一些通用 ETL 处理，由于我们使用的是同一套 PB 结构，对不同 App 数据处理的逻辑代码可以完全复用，这大大降低了我们的开发成本。

汇总层之明细汇总

明细汇总层是由明细层通过 ETL 得到，主要以宽表形式存在。业务明细汇总是由业务事实明细表和维度表 Join

得到，流量明细汇总是由流量日志按业务线拆分和流量维度 Join 得到。流量按业务拆分后可以满足各业务实时消费的需求，我们在流量拆分这一块做到了自动化，下图演示了流量数据自动切分的过程。



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

Streaming Proxy 是流量分发模块，它消费上游 ETL 后的全量数据并定期读取埋点元信息，通过将流量数据与元信息数据进行「Join」完成按业务进行流量拆分的逻辑，同时也会对切分后的流量按业务做 ETL 处理。
只要埋点元信息中新增一个埋点，那么这个埋点对应的数据就会自动切分到该业务的 Kafka 中，最终业务 Kafka 中的数据是独属于当前业务的且已经被通用 ETL 和业务 ETL 处理过，这大大降低了各个业务使用数据的成本。

汇总层之指标汇总

指标汇总层是由明细层或者明细汇总层通过聚合计算得到，这一层产出了绝大部分的实时数仓指标，这也是与实时数仓 1.0 最大的区别。知乎是一个生产内容的平台，对业务指标的汇总我们可以从内容角度和用户角度进行汇总，从内容角度我们可以实时统计内容（内容可以是答案、问题、文章、视频、想法）的被点赞数、被关注数、被收藏数等指标，从用户角度我可以实时统计用户的粉丝数、回答数、提问数等指标。对流量指标的汇总我们分为各业务指标汇总和全局指标汇总。对各业务指标汇总，我们可以实时统计首页、搜索、视频、想法等业务的卡片曝光数、卡片点击数、CTR 等，对全局指标汇总我们主要以实时会话为主，实时统计一个会话内的 PV 数、卡片曝光数、点击数、浏览深度、会话时长等指标。

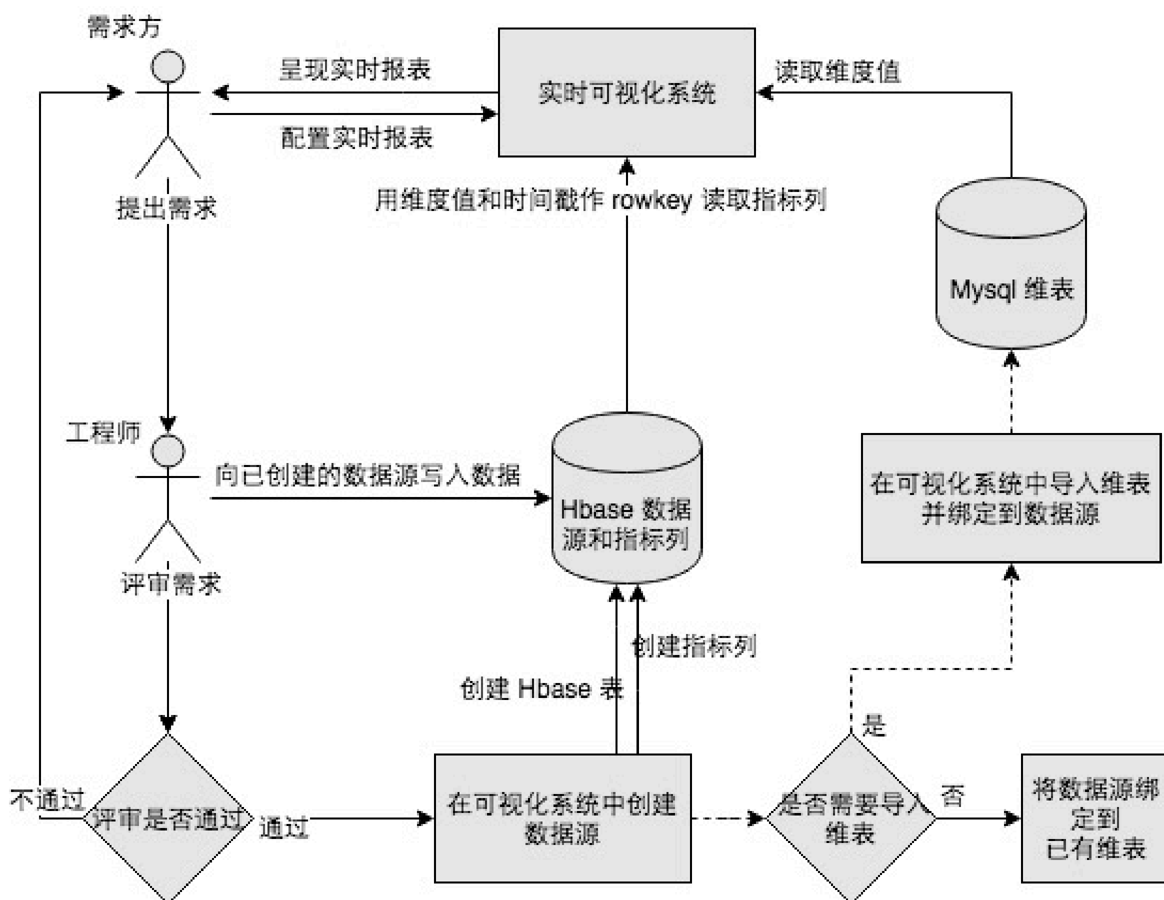
指标汇总层的存储选型

不同于明细层和明细汇总层，指标汇总层需要将实时计算好的指标存储起来以供应用层使用。我

们根据不同的场景选用了 HBase 和 Redis 作为实时指标的存储引擎。Redis 的场景主要是满足带 Update 操作且 OPS 较高的需求，例如：实时统计全站所有内容（问题、答案、文章等）的累计 PV 数，由于浏览内容产生大量的 PV 日志，可能高达几万或者几十万每秒，需要对每一条内容的 PV 进行实时累加，这种场景下选用 Redis 更为合适。HBase 的场景主要是满足高频 Append 操作、低频随机读取且指标列较多的需求，例如：每分钟统计一次所有内容的被点赞数、被关注数、被收藏数等指标，将每分钟聚合后的结果行 Append 到 HBase
并不会带来性能和存储量的问题，但这种情况下 Redis 在存储量上可能会出现瓶颈。

指标计算打通指标系统和可视化系统

指标口径管理依赖指标系统，指标可视化依赖可视化系统，我们通过下图的需求开发过程来讲解如何将三者联系起来。



如果想及时了解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

1. 需求方整理好需求文档后向数仓工程师提出需求并约会议评审需求，需求文档中必须包含指标的计算口径和指标对应的维度。
2. 数仓工程师根据需求文档对需求进行评审，评审不通过则返回需求方进一步整理需求并重新提需。
3. 在需求评审通过后，数仓工程师开始排期开发
    - 首先在可视化系统中创建一个数据源，这个数据源是后期配置实时报表的数据源

，创建数据源也即在 HBase 中创建一张 HBase 表。
    - 针对该数据源创建指标列，创建指标列也即在 HBase 列族中创建列，创建指标列的同时会将该指标信息录入指标管理系统。
    - 针对该数据源绑定维表，这个维表是后期配置多维报表时选用维度值要用的，如果要绑定的维表已经存在，则直接绑定，否则需要导入维表。
    - 一个完整的数据源创建后，数仓工程师才能开发实时应用程序，通过应用程序将多维指标实时写入已创建的数据源中。
4. 需求方根据已创建的数据源直接配置实时报表。

应用层

应用层主要是使用汇总层数据以满足业务需求。应用层主要分三块：1. 通过直接读取指标汇总数据做实时可视化，满足固化的实时报表需求，这部分由实时大盘服务承担；2. 推荐算法等业务直接消费明细汇总数据做实时推荐；3. 通过 Tranquility 程序实时摄入明细汇总数据到 Druid，满足实时多维即席分析需求。

实时数仓 2.0 中的技术实现

相比实时数仓 1.0 以 Spark Streaming 作为主要实现技术，在实时数仓 2.0 中，我们将 Flink 作为指标汇总层的主要计算框架。Flink 相比 Spark Streaming 有更明显的优势，主要体现在：低延迟、Exactly-once 语义支持、Streaming SQL 支持、状态管理、丰富的时间类型和窗口计算、CEP 支持等。

我们在实时数仓 2.0 中主要以 Flink 的 Streaming SQL 作为实现方案。使用 Streaming SQL 有以下优点：易于平台化、开发效率高、维度成本低等。目前 Streaming SQL 使用起来也有一些缺陷：1. 语法和 Hive SQL 有一定区别，初使用时需要适应；2.UDF 不如 Hive 丰富，写 UDF 的频率高于 Hive。

实时数仓 2.0 取得的进展

1. 在明细汇总层通过流量切分满足了各个业务实时消费日志的需求。目前完成流量切分的业务达到 14+，由于各业务消费的是切分后的流量，Kafka 出流量下降了一个数量级。
2. 各业务核心实时报表可以稳定产出。由于核心报表的计算直接由数仓负责，可视化系统直接读取实时结果，保证了实时报表的稳定性，目前多个业务拥有实时大盘，实时报表达 40+。
3. 提升了即席查询的稳定性。核心报表的指标计算转移到数仓，Druid 只负责即席查询，多维分析类的需求得到了满足。
4. 成本计算需求得到了解决。由于各业务拥有了独立的数据源且各核心大盘由不同的实时程序负责，可以方便的统计各业务使用的存储资源和计算资源

实时数仓未来展望

从实时数仓 1.0 到 2.0，不管是数据架构还是技术方案，我们在深度和广度上都有了更多的积累。随着公司业务的快速发展以及新技术的诞生，实时数仓也会不断的迭代优化。短期可预见的我们会从以下方面进一步提升实时数仓的服务能力。

1. Streaming SQL 平台化。目前 Streaming SQL 任务是以代码开发 maven 打包的方式提交任务，开发成本高，后期随着 Streaming SQL 平台的上线，实时数仓的开发方式也会由 Jar 包转变为 SQL 文件。
2. 实时数据元信息管理系统化。对数仓元信息的管理可以大幅度降低使用数据的成本，离线数仓的元信息管理已经基本完善，实时数仓的元信息管理才刚刚开始。
3. 实时数仓结果验收自动化。对实时结果的验收只能借助与离线数据指标对比的方式，以 Hive 和 Kafka 数据源为例，分别执行 Hive SQL 和 Flink SQL，统计结果并对比是否一致实现实时结果验收的自动化。

作者简介
数据工程团队是知乎技术中台的核心团队之一，该团队主要由数据平台、基础平台、数据仓库、AB Testing 四个子团队的 31 位优秀工程师组成。
本文转载自：https://www.infoq.cn/article/Y1jbo_3ZMAQkMOm8loeY

本博客文章除特别声明，全部都是原创！
转载本文请加上：转载自过往记忆（https://www.iteblog.com/）
本文链接: 【】（）