OPPO 实时数仓揭秘：从顶层设计实现离线与实时的平滑迁移

一、建设背景

关于 OPPO 移动互联网业务

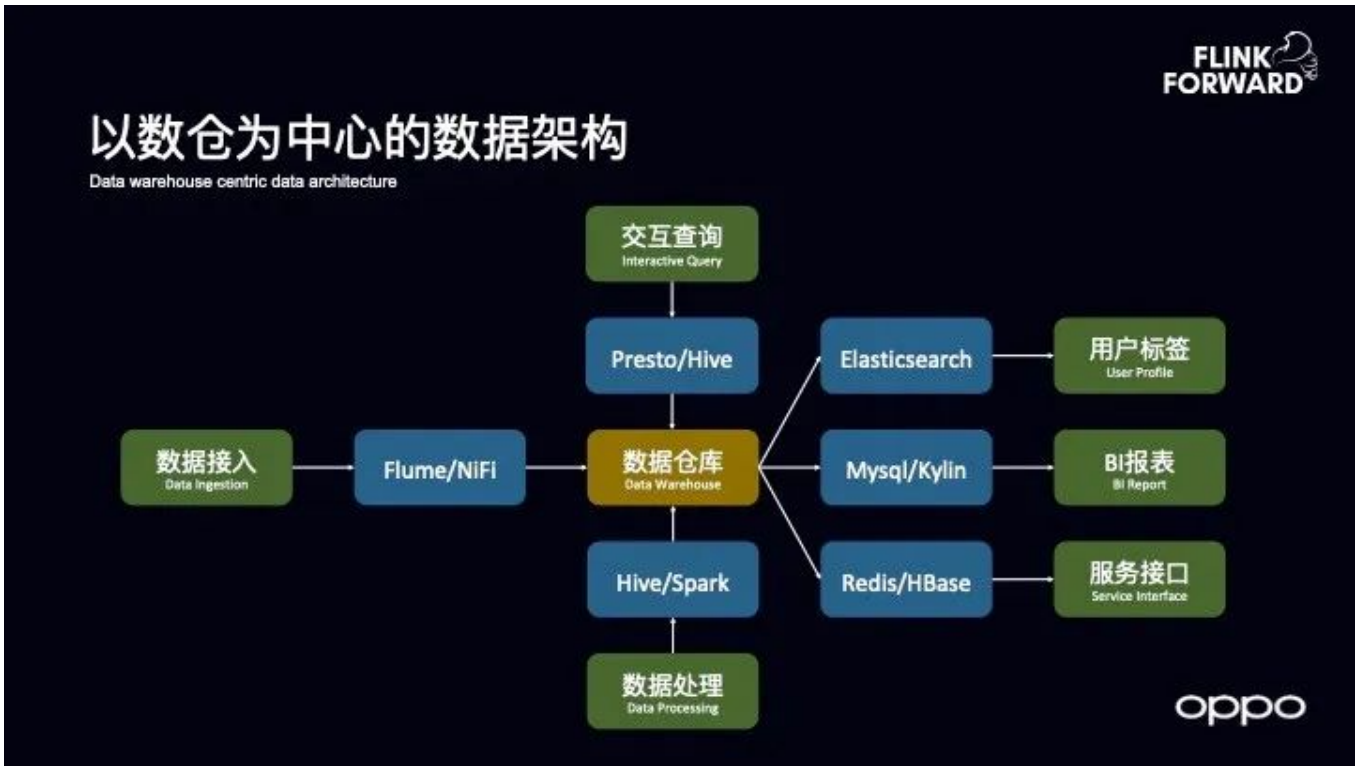
大家都认为 OPPO 是一家手机公司，但大家可能并不清楚，其实 OPPO 也会做与移动互联网相关的业务。在 2019 年 12 月，OPPO 发布了自己定制的手机操作系统 ColorOS 7.0 版本。目前包括海外市场在内，ColorOS 的日活已经超过了 3 亿。ColorOS 内置了很多移动互联网服务，包括应用商店、云服务、游戏中心等，而这些服务的日活也达到了几千万级别。

以数仓为核心的数据架构

为了支撑这些移动互联网服务，OPPO 建立了如下图以数仓为核心的数据架构。图中蓝色的部分，相信大家应该都很熟悉，这部分基本上都是一些开源的组件，从数据接入，到基于数仓实现交互式查询、数据处理，再到数据应用。其中的应用主要分为三个方面：

- 第一是会将数据导入到 ES 里面去做一些用户的标签以及人群的定向投放等。
- 第二是将数据导入到 MySQL 或者 Kylin 里面去做 BI 报表。
- 第三是将数据放到 Redis 或者 HBase 里面去做服务接口。

在过去几年的时间里面，OPPO 内部的这套以数仓为核心的数据架构已经逐渐开始成熟了。



以数仓为核心的数据架构

但是随着业务的发展以及数据规模的不断膨胀，OPPO 对于数仓实时化的诉求越来越强烈。OPPO 对于数仓实时化的诉求可以分为两个维度，即业务维度和平台维度。

- **对于业务维度而言**，越来越需要去做精细化的运营，也越来越需要去挖掘数据的价值，所以无论是实时报表、实时标签还是实时接口等都需要实时化能力。
- **对于平台维度来讲**，也需要实时化。因为整体的数据规模越来越大，通常像传统“T+1”的数据处理模式使得在凌晨的时候服务压力非常大。如果能够将整个集群的压力均摊到全天的 24 小时里面去，那么整个集群的使用效率就会更高一些。所以，即使从调度任务、用户标签的导入等来看，如果能够非常及时地发现数据的异常，对于平台而言也是需要很多的实时化能力。

二、顶层设计

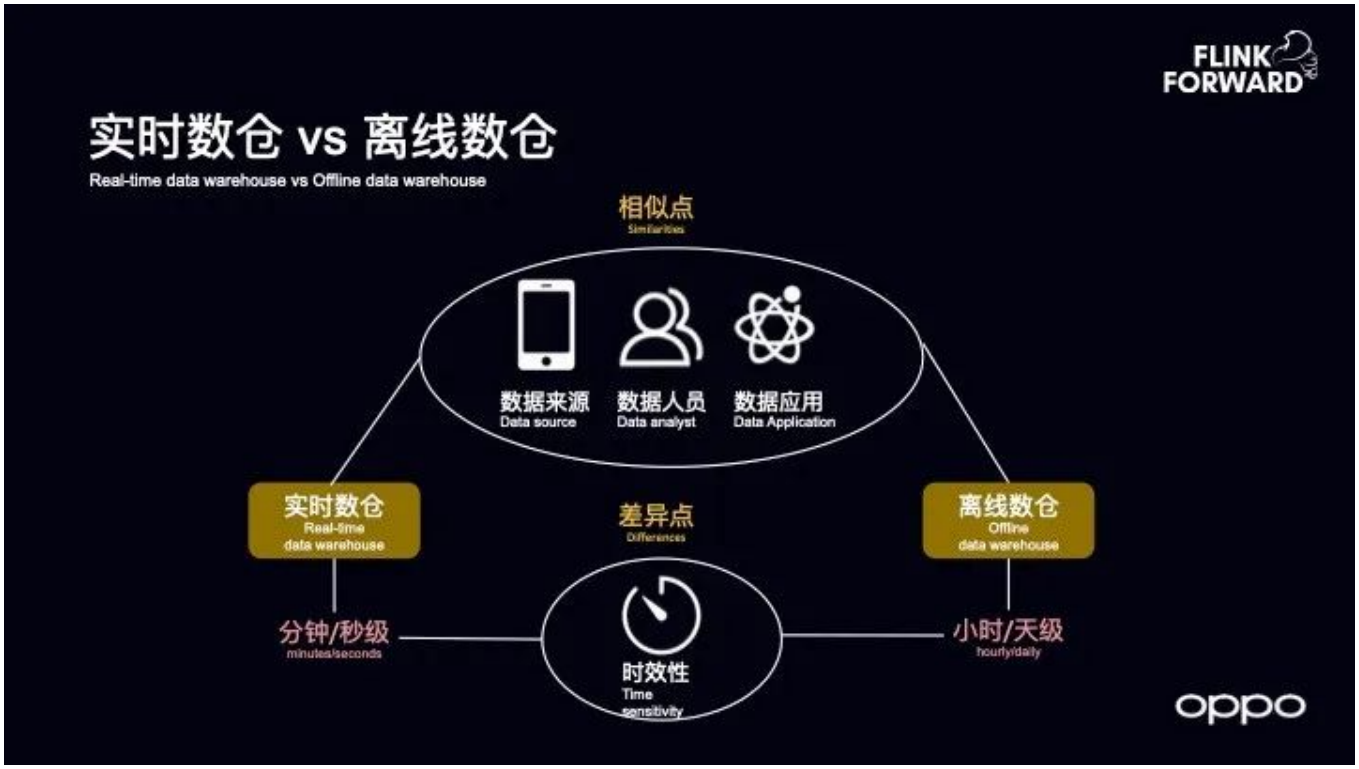
实时数仓的现状

目前 OPPO 实时数仓的规模是 Flink 已经达到了 500 多个节点，Kafka 大概达到了 200 多个节点。在元数据维度，实时数据库表达到了 500 多张，实时作业大概有 300 多个。在数据规模维度，每天总数据处理量超过了 10 万亿，峰值大概超过每秒 3 亿。

实时数仓 VS 离线数仓

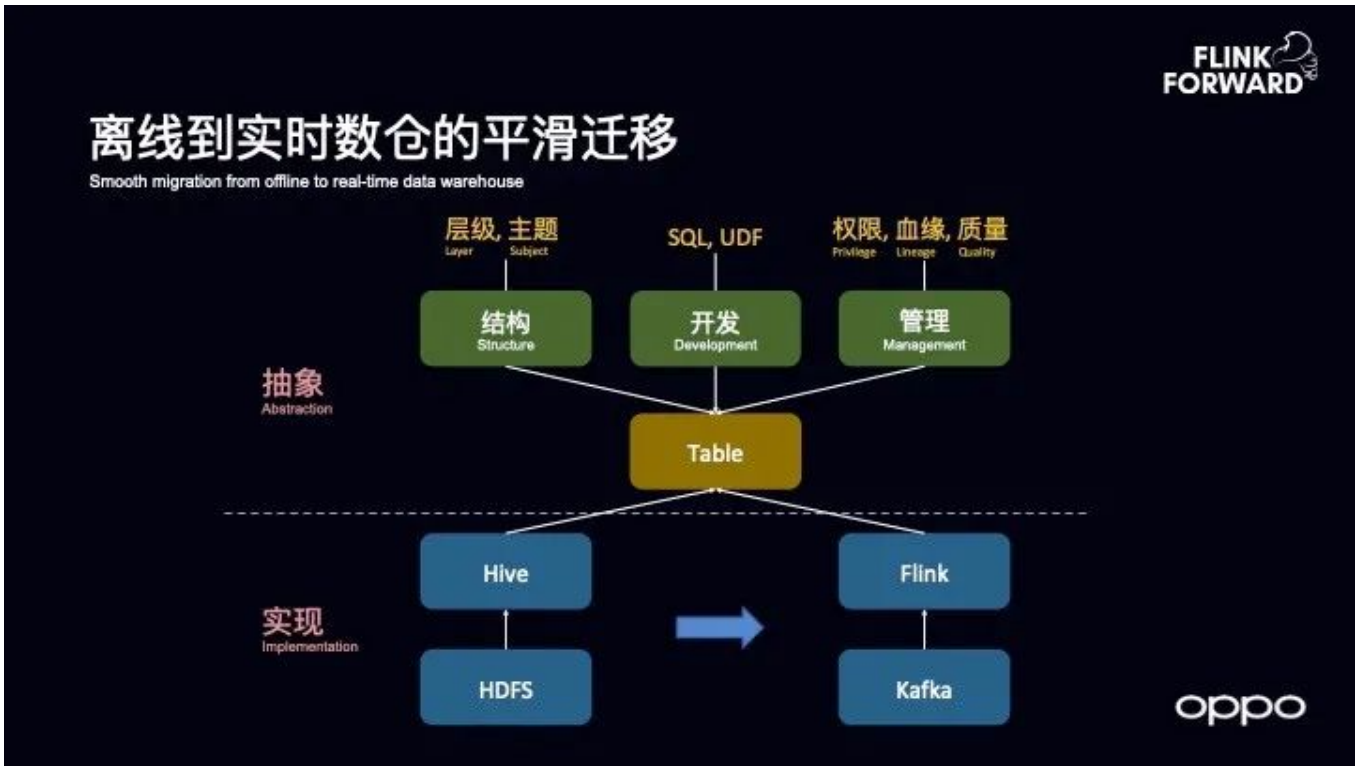
谈到实时数仓的顶层设计，也不得不谈到实时数仓的底层逻辑，因为底层逻辑决定顶层设计，而底层逻辑则来自于实时的观察。

下图中将实时数仓和离线数仓放在一起进行了对比，发现两者的相似性很多，无论是数据来源、数据使用者、数据开发人员以及数据应用都非常相似，两者最大的差异点在于时效性，因为实时数仓中数据的时效性需要达到分钟级或者秒级。



离线到实时数仓的平滑迁移

当有了对于底层逻辑的观察之后，就能够推导出顶层设计情况。OPPO 希望所设计出来的实时数仓能够实现从离线到实时的平滑迁移，之前大家如何使用和开发离线数仓，如今到了实时数仓也希望大家如何开发和使用。通常而言，当设计一款产品或者平台的时候，可以划分为两层，即底层实现和上层抽象。对于底层实现而言，可能会有不同的技术，从 Hive 到 Flink，从 HDFS 到 Kafka。而在上层抽象而言，则希望对于用户而言是透明的。



无论是离线还是实时，最终都希望数仓的核心抽象就是一个 Table，围绕着这个核心的抽象，上面还有三个维度的抽象。

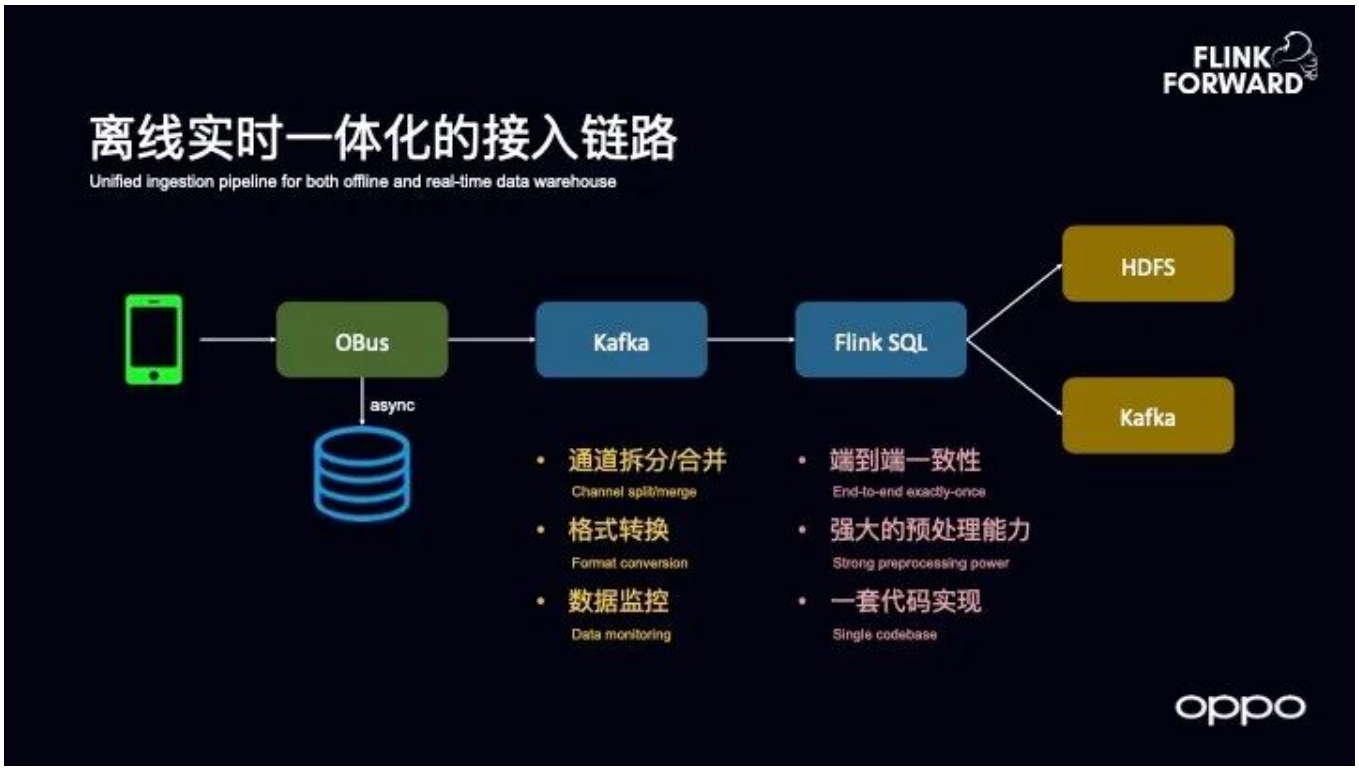
- 第一个抽象就是**数仓的结构**，根据不同的结构能够划分不同的主题域和层次。
- 第二个抽象就是**数仓的开发模式**，基本上都是 SQL+UDF 的开发模式。
- 第三个抽象就是**管理**，从管理上来看，数仓无非就是如何管理其权限以及数据的血缘和质量。

从以上三个抽象维度来看，我们希望从离线到实时能够将抽象保持一致的，这样对于用户而言成本是最低的。接下来则会为大家介绍如何将迁移的成本保持最低。

离线实时一体化接入链路

首先为大家介绍离线实时一体化接入链路，OPPO 的数据从手机端到 OBus 内部数据收集服务，收集之后会统一落入到Kafka中去，再通过 Flink SQL 的任务可以同时落入 HDFS 和 Kafka 中去。Flink 可以实现数据通道的拆分，对于 OPPO 这样一个手机公司而言，很多 APP 上报都是通过同一条通道，因此在将数据落入到数仓之前需要对于数据通道进行拆分，根据不同的业务和属性做一些拆分，除此之外还会做一些格式的转换。另外一部分功能就是实现数据的监控，因为将数据落入到 HDFS 时需要有一个很重要的问题就是分区感知问题，比如离线 ETL 任务如何知道分区已经结束了。

OPPO 的做法是根据端到端不同数据的对账实现的，因此需要在 Flink SQL 这一层完整地记录收到多少条数据，写入了多少条数据，然后和前面的 OBus 做一个数据对账的对比，如果对比结果在一定范围之内，就可以写一个成功文件，这样就可以让后端的 ETL 任务开始运行。

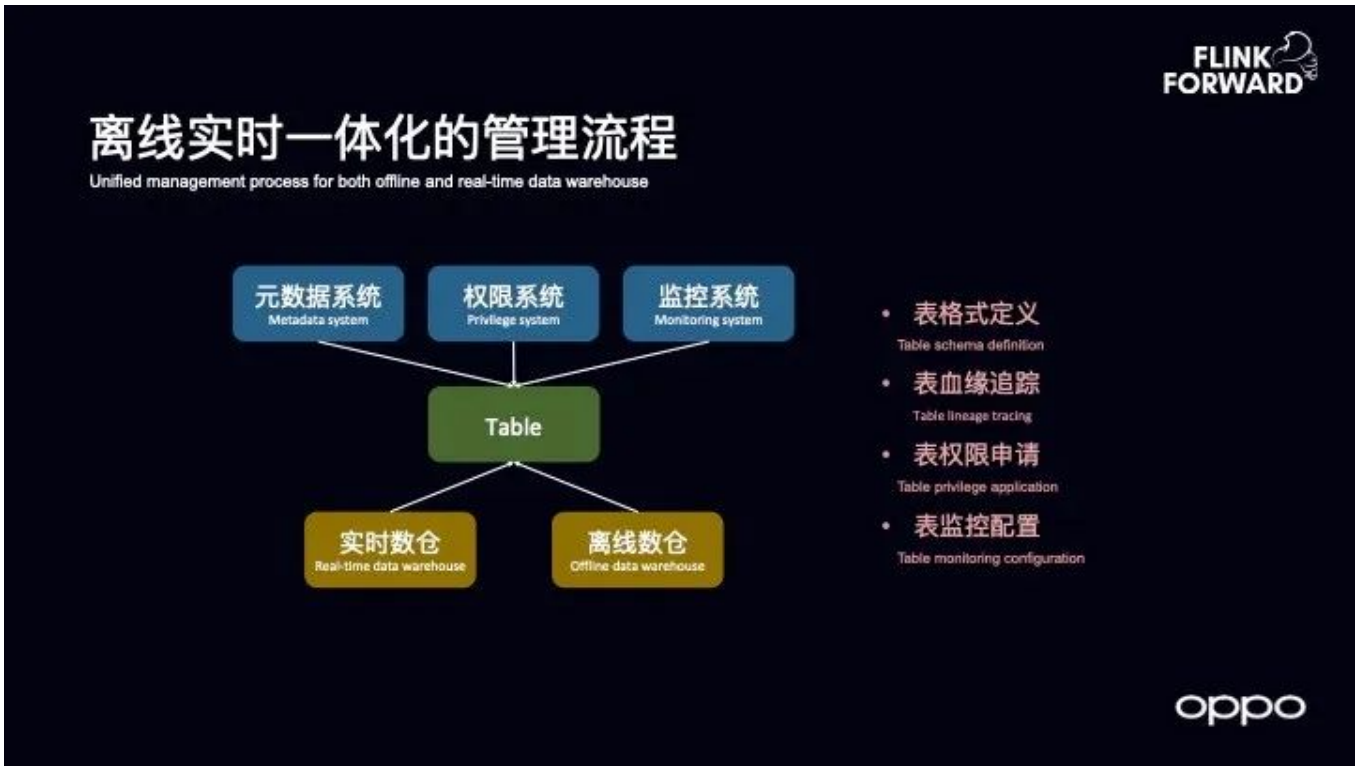


使用 Flink SQL 所 带来的好处在于：

- **第一**，Flink SQL 可以保证端到端的一致性，无论是从 Kafka 到 Kafka，还是从 Kafka 到 HDFS，都能够保证端到端的数据一致性，这一点对于接入链路而言是非常重要的。
- **第二**， Flink SQL 具有强大的数据预处理能力，OPPO 过去在数据接入通道里面使用过 Flume 等，但是这些组件的数据处理性能很难提升上去，因此需要追加很多机器来实现性能提升。而使用 Flink 之后，使得数据处理能力有了巨大提升。
- **第三**，能够使用一套代码来实现将数据落入到 HDFS 和 Kafka 里面去，因此大大降低了维护成本。

离线实时一体化的管理流程

对于数仓的管理流程而言，无非就是元数据是如何管理的，表的字段是如何定义的，表的血缘如何追踪以及表的权限如何管理，以及表的监控如何实现。如今在 OPPO 内部，离线和实时数仓的这些管理流程能够做到一致，首先两者使用的流程是一致的，其次表的 Schema 的定义以及表的血缘能够保证一致，而不需要用户重新申请和定义。



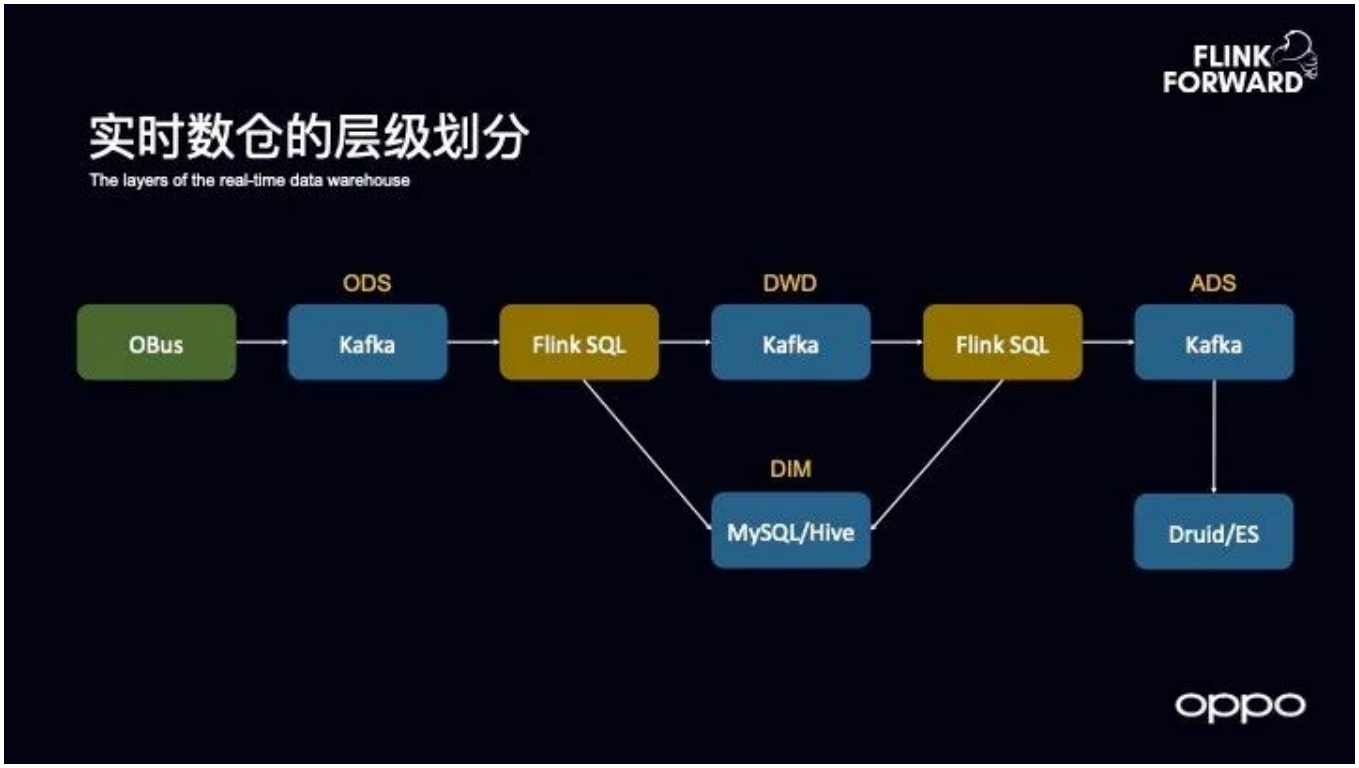
离线实时一体化的开发环境

对于数仓的开发而言，抽象下来可以分为三个层面，即离线批处理的开发、流式开发以及交互式查询。而对于用户而言，希望能够保证用户体验的一致，并且希望实现开发流程的统一。

实时数仓的层级划分

如下图所示的是 OPPO 实时数仓的分层结构，从接入层过来之后，所有的数据都是会用 Kafka 来支撑的，数据接入进来放到 Kafka 里面实现 ODS 层，然后使用 Flink SQL 实现数据的清洗，然后就变到了 DWD 层，中间使

用 Flink SQL 实现一些聚合操作，就到了 ADS 层，最后根据不同的业务使用场景再导入到ES等系统中去。当然，其中的一些维度层位于 MySQL 或者 Hive 中。



SQL 一统天下的数据架构

对于数仓领域的近期发展而言，其中很有意思的一点是：无论是离线还是实时的数据架构，都慢慢演进成了 SQL 一统天下的架构。无论是离线还是实时是数据仓库，无论是接入，查询、开发还是业务系统都是在上面写 SQL 的方式。

三、落地实践

前面为大家分享了 OPPO 实时数仓实践的顶层设计，当然这部分并没有全部实现，接下来为大家分享 OPPO 已经有的落地实践，

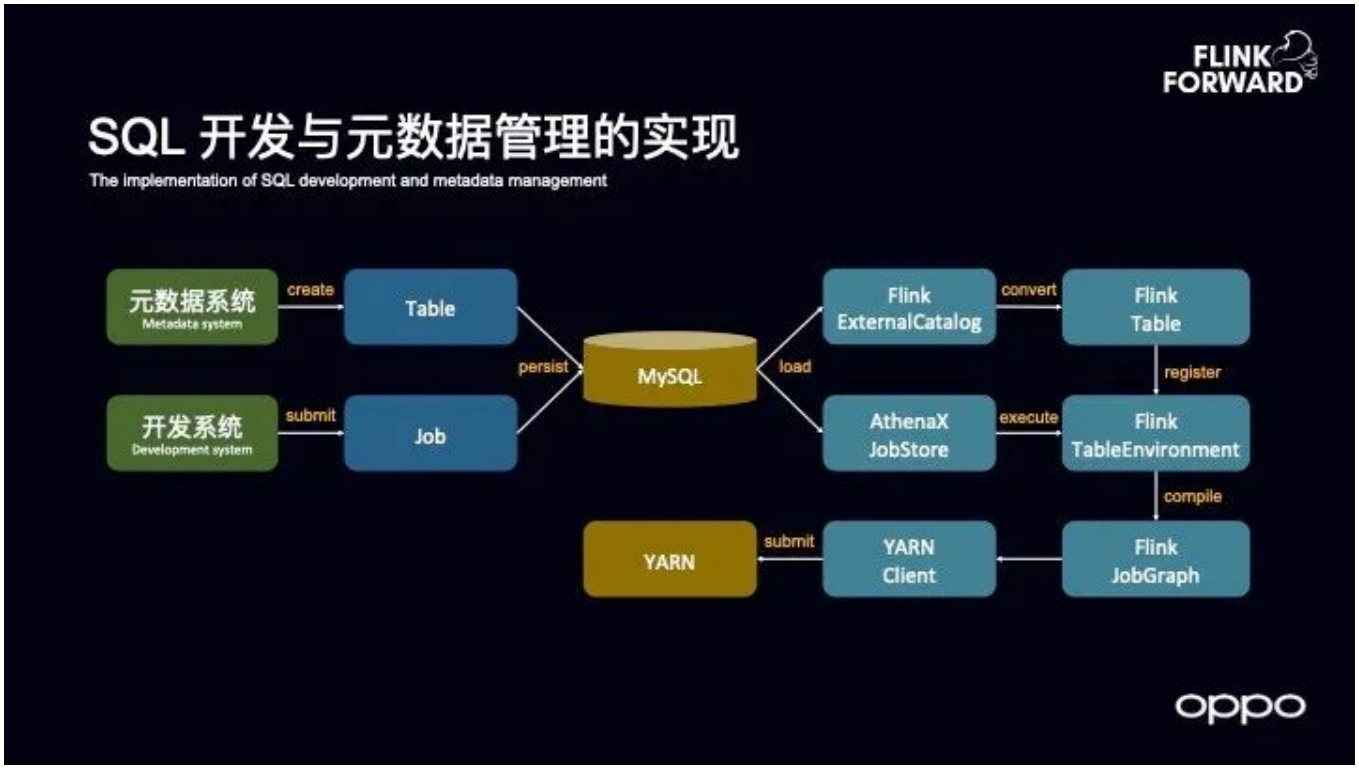
SQL 开发与元数据管理的实现

想要做实时数仓所需要的第一步就是支持 SQL 的开发与元数据管理的实现。OPPO 在这部分的设计大致如下图所示。

这里需要元数据系统和开发系统，需要能够在元数据系统中创建实时表并在开发系统里面创建实时作业并写 SQL，而无论是创建 Table 还是 Job，都需要能够持久化到 MySQL 里面去。

然后再去扩展 Flink 里面的组件，并将其从 MySQL 里面加载出来。

- 对于表而言，可以扩展 Flink 的 Catalog，通过 Catalog 可以从 MySQL 中加载出来，再转化成 Flink 内部表达的数据表。
- 对于作业而言，OPPO 则使用了谷歌开源的框架，通过对于 Job Store 的实现可以从数据源头比如 MySQL 来加载这个作业，将这个作业提交给 Flink 的 Table 环境来做作业的编译，最终定义成为 Job Graph，然后提交给 YARN，这样的流程就是支撑 OPPO 实时数仓的框架。



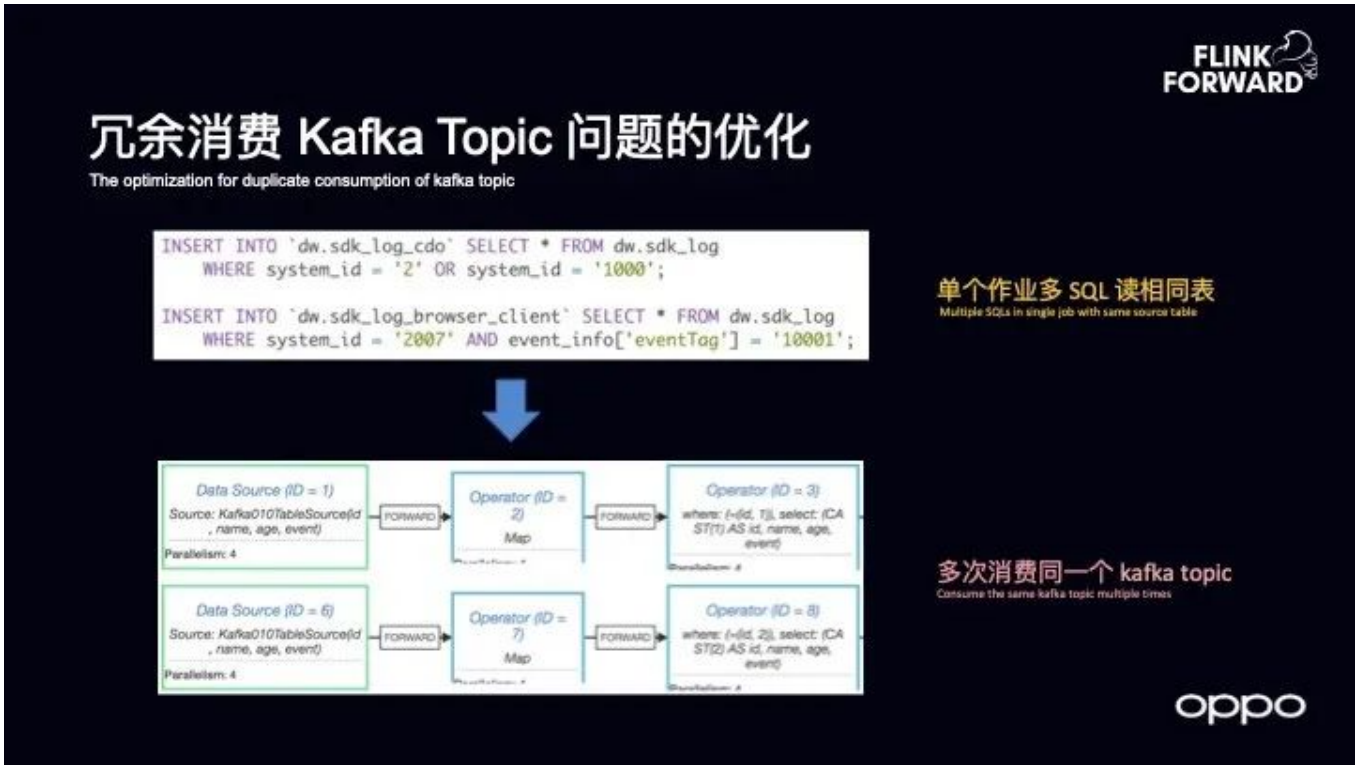
冗余消费 Kafka Topic 问题的优化

在 OPPO 的场景下，我们发现了自己所存在的一个很棘手的问题，那就是很多用户在写 SQL 的时候会出现同一个作业需要写多个 SQL，比如刚才提到的接入场景，如果想要做通道的拆分，通常而言需要来自同一个表格，经过不同的过滤，然后导入到不同的数据表里面去，而 OPPO 希望在单个作业中就能够实现这样的表达。

但是这样做所带来的问题就是将多个 SQL 放在一个作业里面执行就会生成多个 Data Source，多个 Data Source 就会重复地消费 Kafka，这就使得 Kafka 集群的压力非常大，原因是很多 Kafka 机器的写入和读取的操

作比例差距非常大，一个 SQL 的作业可能会读取很多次 Kafka 的 Topic。而这是没有必要的，因为对于同一次作业而言，只需要消费一次 Kafka 即可，接下来数据可以在 Flink 内部进行消化和传播。

OPPO 针对于上述问题实现了一个非常巧妙的优化，因为 Flink 的 SQL 会生成一个 Job Graph，在这之前会生成一个 Stream Graph。而 OPPO 通过改写 Stream Graph，使得无论用户提交多少个 SQL，对应只有一个 Data Source，这样就降低了对于 Kafka 的消费量，而且带为用户来了很大的收益。



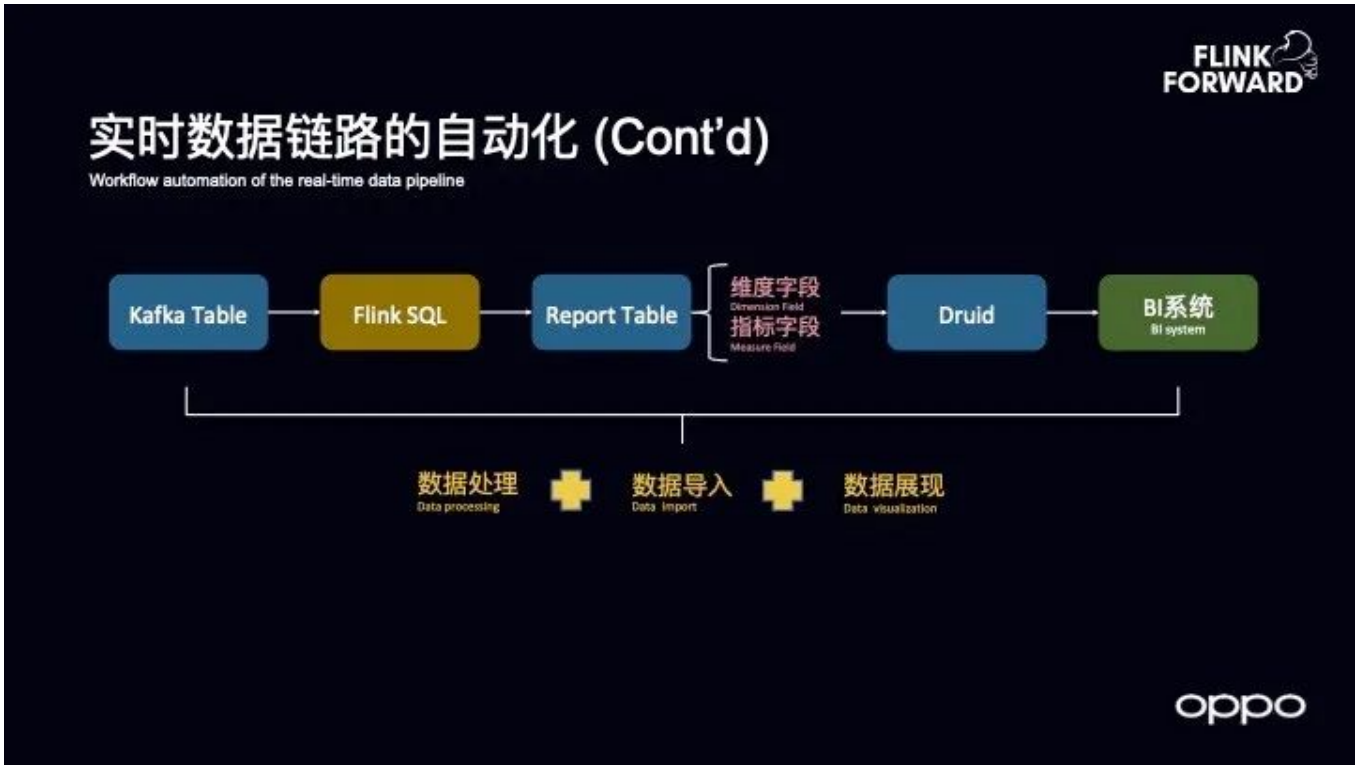
实时数据链路的自动化

线上 BI 的实时报表是非常通用的场景，对于实时报表而言，往往需要三个环节的配合：

- 第一个环节是数据分析师去写 SQL 实现对数据的处理；
- 第二个环节是从一个数据表过来统计或者清洗，再写入到 Kafka 里面去，通过平台的研发人员再将数据打入到 Druid 里面去；
- 最终的一个环节就是用户需要去 BI 系统中查看报表，因此就需要从 Druid 这张表导入到 BI 系统中去。

上述链路中的数据处理、数据导入和数据展现三个环节是比较割裂的，因此需要三种不同角色的人员来介入做这件事情，因此 OPPO 希望能够打通实时数据链路。OPPO 做了如下图所示的实时数据链路的自动化，对于 Kafka 的表做了抽象，而对于用户而言，其就是用于做 BI 展示的表，Kafka 的表需要定义哪些是维度、哪些是指标，这是做报表展示最基本的字段定义。

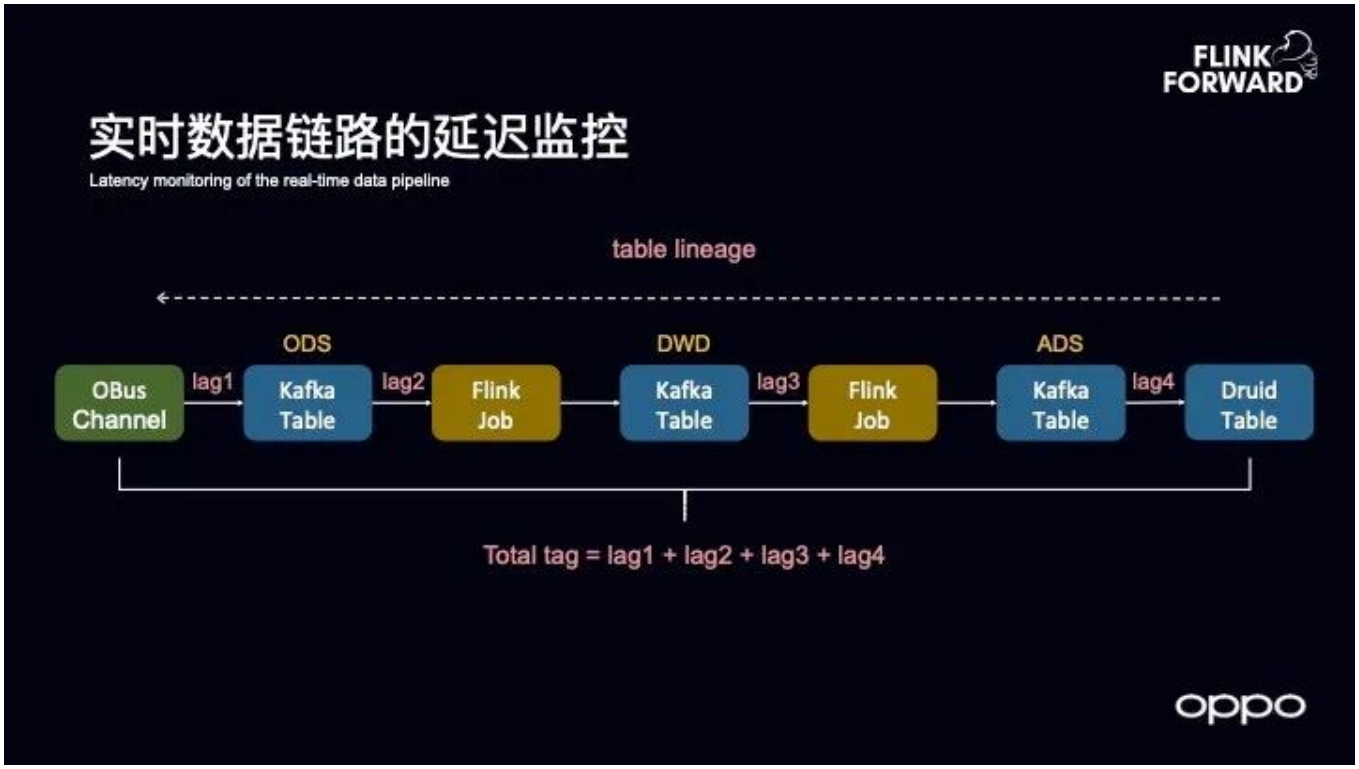
当完成了上述任务之后，就可以将整个实时数据链路以自动化的方式串起来。当用户将 SQL 写完之后，可以自动化地探测 Report Table 需要导入到 Druid 里面去，以及哪些是指标，哪些是维度，并且可以将数据从 Druid 自动地导入到 BI 系统。这样一来，对于用户而言只需要写一个 SQL，之后就可以在 BI 系统之上看到报表了。



实时数据链路的延迟监控

之前，OPPO 做数据链路的延迟监控时也属于单个点进行监控的，可以从下图中看出至少有三级的 Kafka 的 Topic，对于每个 Topic 都存在延迟的监控。而对于用户而言，关注的并不是点，而是面，也就是最终展现的数据报表中延迟情况如何。

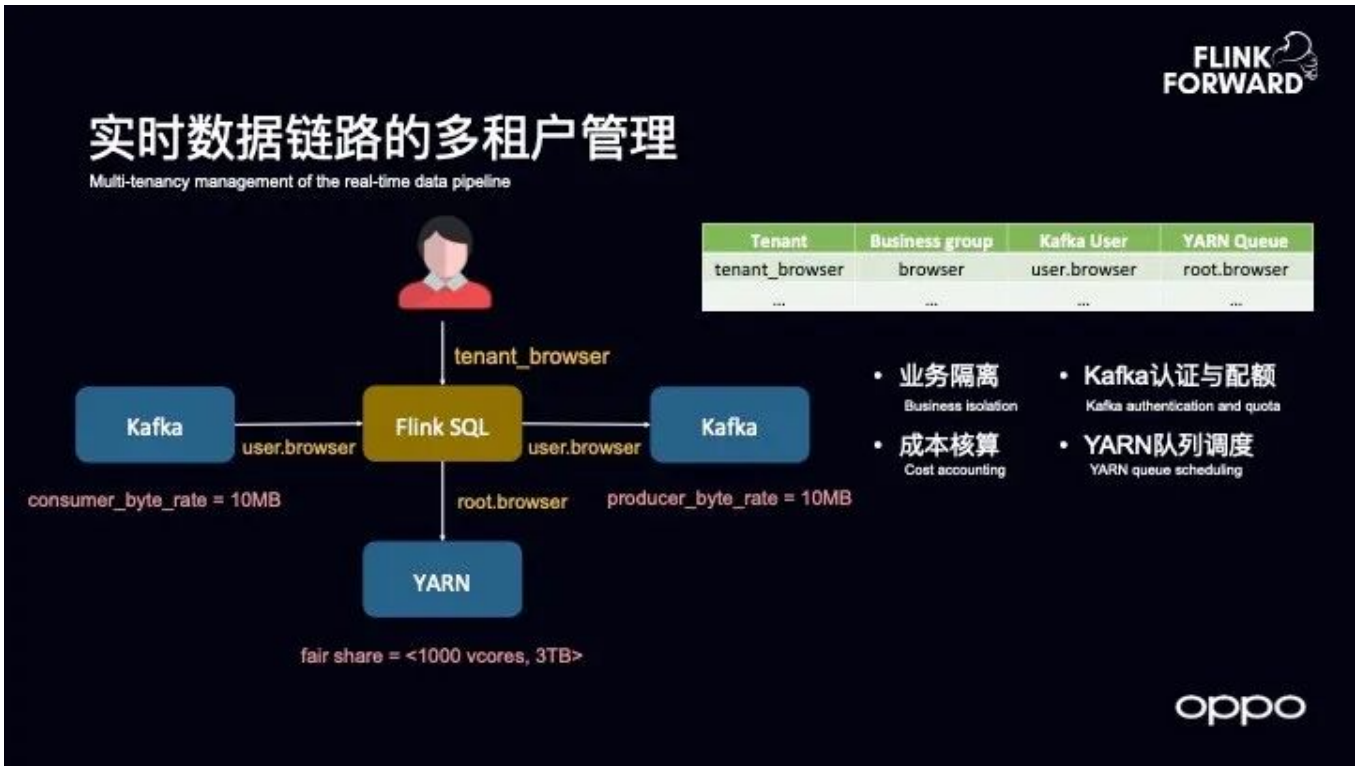
因此， OPPO 也实现了全链路的延迟监控，从接入的通道开始到每一层的 Kafka 消费，都将其 lag 情况汇总起来，探索到每一级的 Flink SQL 表的血缘关系。有了这样的血缘关系之后就可以从 Druid 表推导到前面所接入的链路是哪一个，然后将总体延迟加起来，这样就可以反映出整体链路的延迟情况。



实时数据链路的多租户管理

对于实时数据链路而言，多租户管理同样非常重要。OPPO 在这部分的实践的核心是两点：

- 其中一点 Kafka 里面的认证和配额机制，当有了认证和配额机制之后可以对于用户做配额管理，比如对于 Kafka 的消费速度、生产速度等。
- 另外一点就是用户在向 YARN 上面提交的作业的时候也可以指定队列，这样就可以指定用户消耗多少资源。



四、未来展望

更便捷的 SQL 开发

因为 OPPO 现在的实时数仓是基于 SQL 做的，所以在未来希望能够具有更好的、更便捷的 SQL 开发能力，总结来下就是以下四点：

- **表达能力**：虽然 Flink SQL 正在朝着标准 SQL 不断演进，但是目前一些场景仍旧无法满足，比如在一个 SQL 里面做多个窗口的统计等，因此需要增强Flink SQL 的表达能力。
- **连接类型**：如今，实时数据仓库的应用越来越多，因此也需要扩充更多的连接器，比如 Redis 等的 Sink。
- **开发模板**：谷歌开源了 Dataflow Template，这是因为用户在做统计、汇总等很多的情况下，方法是通用的，因此对于用户而言这些通用操作可以做成模板，避免重复编写 SQL。
- **开发规范**：这也是 OPPO 在线上实践中所观察到的问题，很多数据分析师写的 SQL 的性能很差，开发人员在定位问题时往往会发现 SQL 的编写不规范，只需要进行一些小优化即可提升性能，因此未来需要将这些能力沉淀到系统里面去。

更细力度的资源调度

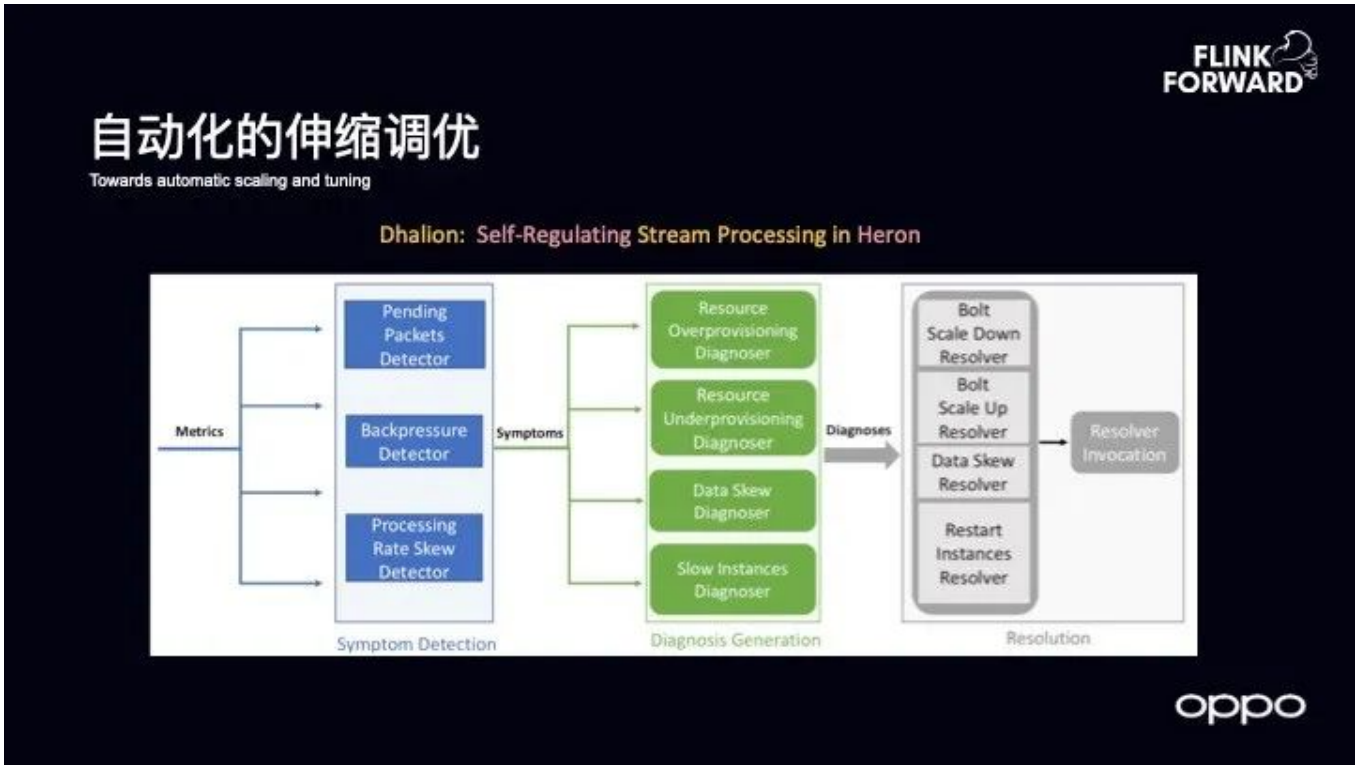
目前，OPPO 是基于 YARN 做 Flink 的集群调度，而 YARN 的调度是基于 VCore 以及内存维度实现的。在线上运行时就发现一些机器的 CPU 利用率很高，另外一些却很低，这是因为不同的 SQL 处理的复杂度以及计算密集度是不同的，如果还是和以前一样分配 VCore，那么很可能导致对于资源的利用率不同，因此未来需要考虑将 SQL 对于资源的调度加入到考虑范围内，尽量避免资源的倾斜。

自动化的参数配置

对于数据分析师而言，大家都知道Flink里面存在一些高级配置。除了写 SQL 之外，还有很多其他的知识，比如操作的并发度、状态后台以及水位间隔等，但是用户往往会很难掌握如何配置这些复杂参数，因此 OPPO 希望未来能够将这些复杂的参数配置实现自动化。通过理解数据的分布情况和 SQL 的复杂情况，自动地配置这些参数。

自动化的伸缩调优

更进一步，可以从自动化实现自适应，变成智能化，也就是自动化的伸缩调优。之所以要做自动化的伸缩，主要是因为两点，第一，数据分布本身就是存在波动性的；第二，机器在不同的时间段也存在不同的状态，因此需要及时探测和修复。因此，自动化的伸缩调优对于大规模集群的成本节省是至关重要的。



作者介绍：

张俊，OPPO 大数据平台研发负责人，主导了 OPPO 涵盖“数据接入-数据治理-数据开发-数据应用”全链路的数据中台建设。2011-硕士毕业于上海交通大学，曾先后工作于摩根士丹利、腾讯，具有丰富的数据系统研发经验，目前重点关注数仓建设、实时计算、OLAP 引擎方向，同时也是Flink开源社区贡献者。