美团点评 Flink 实时数仓应用经验分享

简介： 实时处理技术，是强调当前处理状态的一门技术，所以我们认为这两个相对对立的方案重叠在一起的时候，它注定不是用来解决一个比较广泛问题的一种方案。于是，我们把实时数仓建设的目的定位为解决由于传统数据仓库数据时效性低解决不了的问题。

整理 | 青渊（Flink 社区志愿者）
校对 | 青雉（Flink 社区志愿者）
作者 | 黄伟伦@美团点评

摘要：本文根据 Apache Flink 系列直播整理而成，由美团点评数据系统研发工程师黄伟伦老师分享。主要内容如下：

1. 实时数仓建设目的
2. 如何建立实时数仓
3. 仓库质量保证

Tips：点击「阅读原文」链接可查看作者原版 PPT 及分享视频～

实时数仓建设目的

解决传统数仓的问题

实时数仓是一个很容易让人产生混淆的概念。实时数仓本身似乎和把 PPT 黑色的背景变得更白一样，从传统的经验来讲，我们认为数仓有一个很重要的功能，即能够记录历史。通常，数仓都是希望从业务上线的第一天开始有数据，然后一直记录到现在。

但实时处理技术，又是强调当前处理状态的一门技术，所以我们认为这两个相对对立的方案重叠在一起的时候，它注定不是用来解决一个比较广泛问题的一种方案。于是，我们把实时数仓建设的目的定位为解决由于传统数据仓库数据时效性低解决不了的问题。

由于这个特点，我们给定了两个原则：

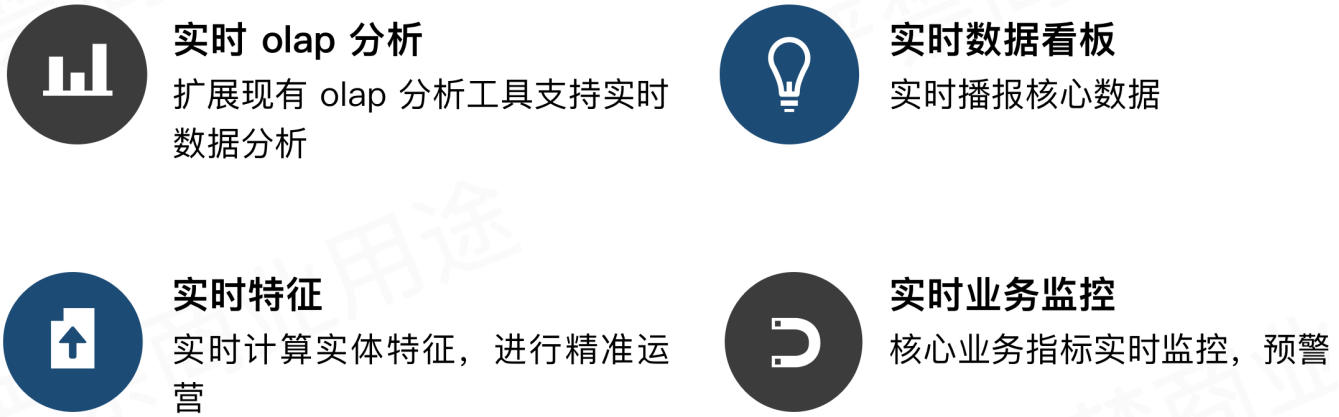
- 传统数仓能解决的问题，实时数仓就不解决了。比如上个月的一些历史的统计，这些数据是不会用实时数仓来建设的。
- 问题本身就不太适合用数仓来解决，也不用实时数仓解决。比如业务性很强的需求，或者是对时效性要求特别高的需求。这些需求我们也不建议通过实时数仓这种方式来进行解决。

当然为了让我们整个系统看起来像是一个数仓，我们还是给自己提了一些要求的。这个要求其实跟我们建立离线数仓的要求是一样的，首先实时的数仓是需要面向主题的，然后具有集成性，并且保证相对稳定。

离线数仓和实时数仓的区别在于离线数据仓库是一个保存历史累积的数据，而我们在建设实时数仓的时候，我们只保留上一次批处理到当前的数据。这个说法非常的拗口，但是实际上操作起来还是蛮轻松的。

通常来讲解决方案是保留大概三天的数据，因为保留三天的数据的话，可以稳定地保证两天完整的数据，这样就能保证，在批处理流程还没有处理完昨天的数据的这段间隙，依然能够提供一个完整的数据服务。

实时数仓的应用场景



- **实时 OLAP 分析**

OLAP 分析本身就非常适合用数仓去解决的一类问题，我们通过实时数仓的扩展，把数仓的时效性能力进行提升。甚至可能在分析层面上都不用再做太多改造，就可以使原有的 OLAP 分析工具具有分析实时数据的能力。

- **实时数据看板**

这种场景比较容易接受，比如天猫双11的实时大屏滚动展示核心数据的变化。实际上对于美团来讲，不光有促销上的业务，还有一些主要的门店业务。对于门店的老板而言，他们可能在日常的每一天中也会很关心自己当天各个业务线上的销售额。

- **实时特征**

实时特征指通过汇总指标的运算来对商户或者用户标记上一些特征。比如多次购买商品的用户后台会判定为优质用户。另外，商户销售额稿，后台会认为该商户商的热度更高。然后，在做实时精准运营动作时可能会优先考虑类似的门店或者商户。

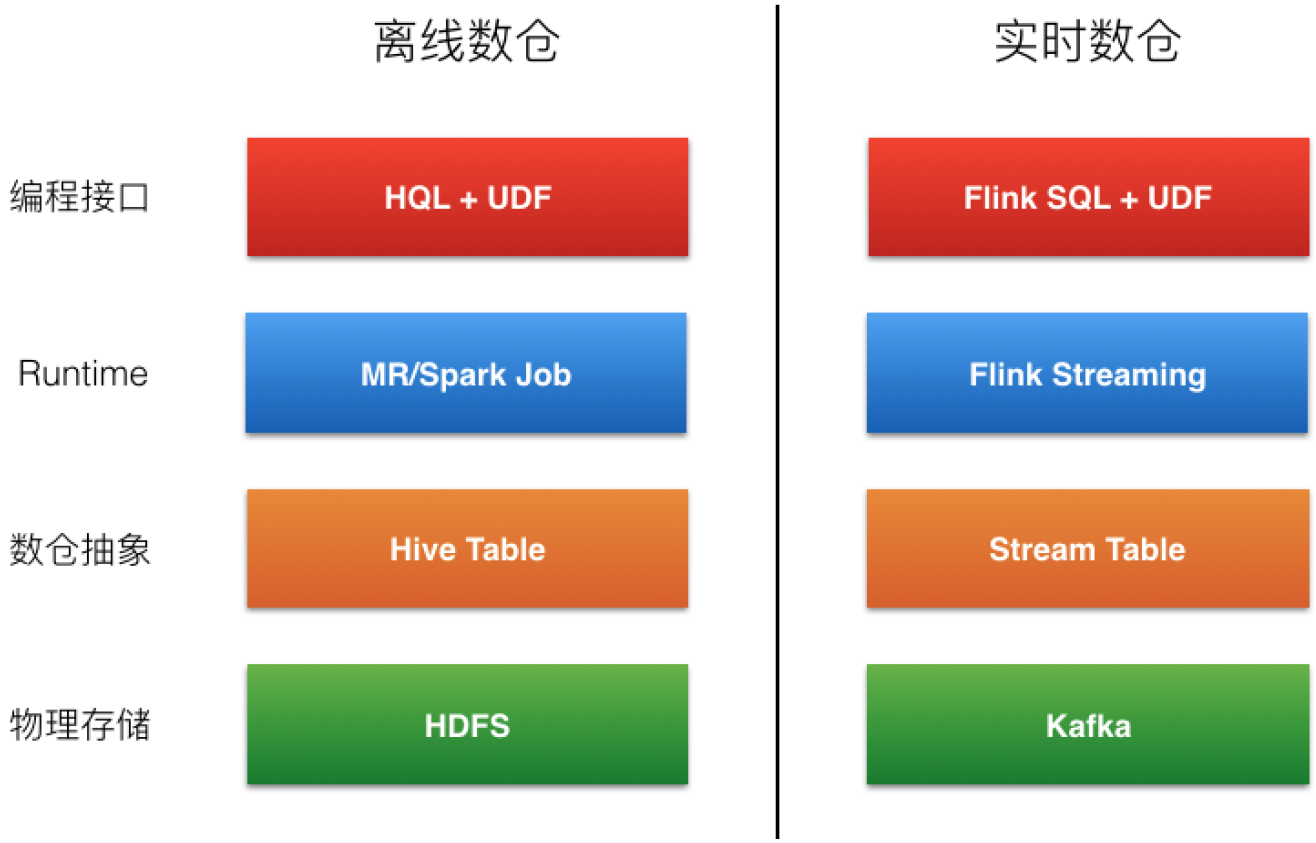
- **实时业务监控**

美团点评也会对一些核心业务指标进行监控，比如说当线上出现一些问题的时候，可能会导致某些业务指标下降，我们可以通过监控尽早发现这些问题，进而来减少损失。

如何建设实时数仓

实时数仓概念映射

我们通过离线数仓开发和实时数仓开发的对应关系表，帮助大家快速清晰的理解实时数仓的一些概念。



- 编程方式

离线开发最常见的方案就是采用 Hive SQL 进行开发，然后加上一些扩展的 udf 。映射到实时数仓里来，我们会使用 Flink SQL ，同样也是配合 udf 来进行开发。

- 作业执行层面

离线处理的执行层面一般是 MapReduce 或者 Spark Job ，对应到实时数仓就是一个持续不断运行的 Flink Streaming 的程序。

- 数仓对象层面

离线数仓实际上就是在使用 Hive 表。对于实时数仓来讲，我们对表的抽象是使用 Stream Table 来进行抽象。

- 物理存储

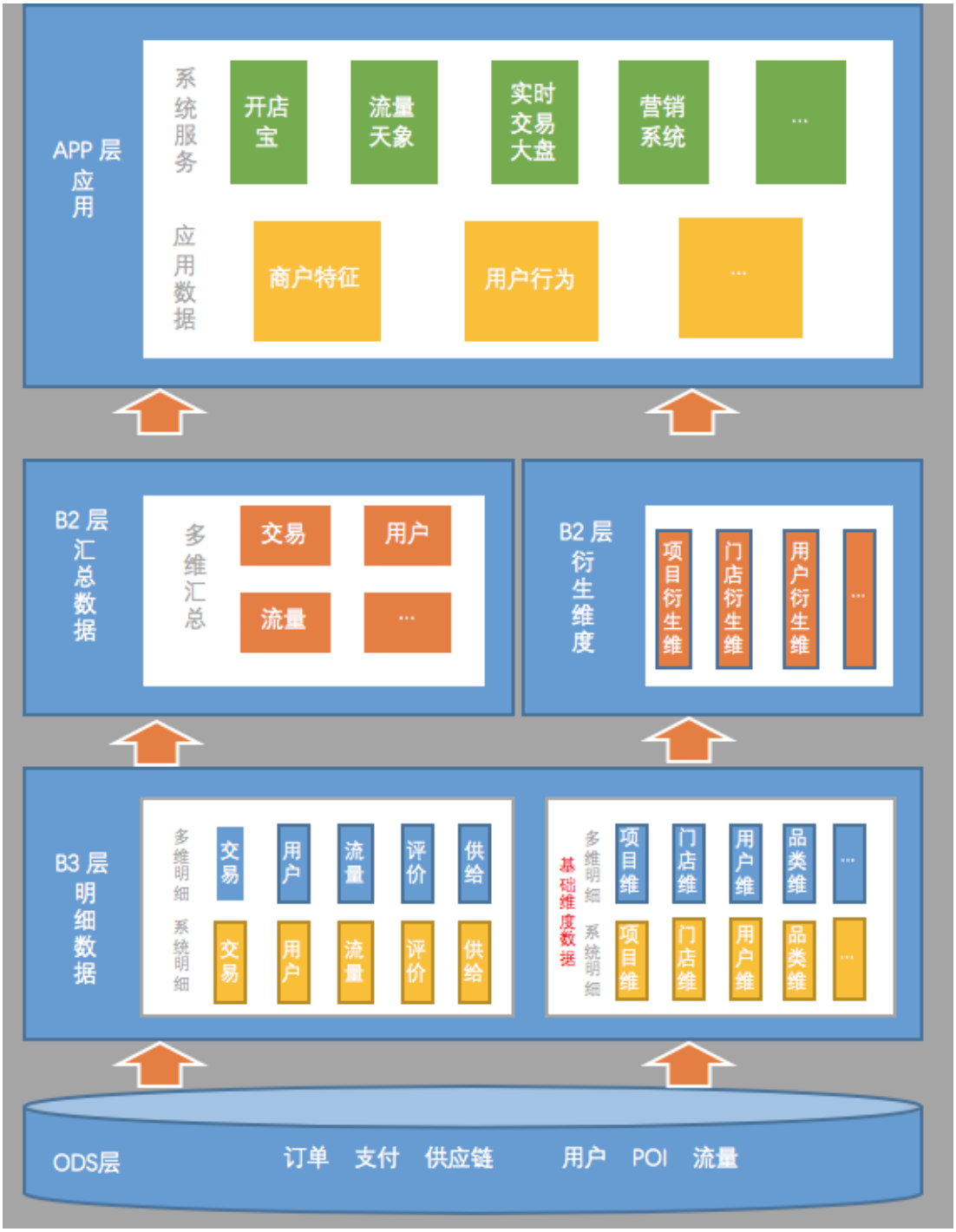
离线数仓，我们多数情况下会使用 HDFS 进行存储。实时数仓，我们更多的时候会采用像 Kafka 这样的消息队列来进行数据的存储。

实时数仓的整体架构

在此之前我们做过一次分享，是关于为什么选择 Flink 来做实时数仓，其中重点介绍了技术组件选型的原因和思路，具体内容参考《美团点评基于 Flink 的实时数仓建设实践》。本文分享的主要内容是围绕数据本身来进行的

，下面是我们目前的实时数仓的数据架构图。

> 《美团点评基于 Flink 的实时数仓建设实践》
> https://tech.meituan.com/2018/10/18/meishi-data-flink.html



从数据架构图来看，实时数仓的数据架构会跟离线数仓有很多类似的地方。比如分层结构；比如说 ODS 层，明细层、汇总层，乃至应用层，它们命名的模式可能都是一样的。尽管如此，实时数仓和离线数仓还是有很多的区别的。

跟离线数仓主要不一样的地方，就是实时数仓的层次更少一些。

以我们目前建设离线数仓的经验来看，数仓的第二层远远不止这么简单，一般都会有一些轻度汇总层这样的概念，其实第二层会包含很多层。另外一个就是应用层，以往建设数仓的时候，应用层其实是在仓库内部的。在应用层建设好后，会建同步任务，把数据同步到应用系统的数据库里。

在实时数仓里面，所谓 APP 层的应用表，实际上就已经在应用系统的数据库里了。上图，虽然画了 APP 层，但它其实并不算是数仓里的表，这些数据本质上已经存过去了。

为什么主题层次要少一些？是因为在实时处理数据的时候，每建一个层次，数据必然会产生一定的延迟。

为什么汇总层也会尽量少建？是因为在汇总统计的时候，往往为了容忍一部分数据的延迟，可能会人为的制造一些延迟来保证数据的准确。

举例，统计事件中的数据时，可能会等到 10:00:05 或者 10:00:10再统计，确保 10:00 前的数据已经全部接受到位了，再进行统计。所以，汇总层的层次太多的话，就会更大的加重人为造成的数据延迟。

建议尽量减少层次，特别是汇总层一定要减少，最好不要超过两层。明细层可能多一点层次还好，会有这种系统明细的设计概念。

第二个比较大的不同点就是在于数据源的存储。

在建设离线数仓的时候，可能整个数仓都全部是建立在 Hive 表上，都是跑在 Hadoop 上。但是，在建设实时数仓的时候，同一份表，我们甚至可能会使用不同的方式进行存储。

比如常见的情况下，可能绝大多数的明细数据或者汇总数据都会存在 Kafka 里面，但是像维度数据，可能会存在像 Tair 或者 HBase 这样的 kv 存储的系统中，实际上可能汇总数据也会存进去，具体原因后面详细分析。除了整体结构，我们也分享一下每一层建设的要点。

■ ODS 层的建设

数据来源尽可能统一，利用分区保证数据局部有序

首先第一个建设要点就是 ODS 层，其实 ODS 层建设可能跟仓库不一定有必然的关系，只要使用 Flink 开发程序，就必然都要有实时的数据源。目前主要的实时数据源是消息队列，如 Kafka。而我们目前接触到的数据源，主要还是以 binlog、流量日志和系统日志为主。

这里面我主要想讲两点：

首先第一个建设要点就是 ODS层，其实ODS层建设可能跟这个仓库不一定有必然的关系，只要你使用这个flink开发程序，你必然都要有这种实时的数据源。目前的主要的实时数据源就是消息队列，如kafka。我们目前接触到的数据源，主要还是以binlog、流量日志和系统日志为主。

这里面我主要想讲两点，一个这么多数据源我怎么选？我们认为以数仓的经验来看：

首先就是数据源的来源尽可能要统一。这个统一有两层含义：

- 第一个统一就是实时的数据源本身要跟自己统一，比如你选择从某个系统接入某一种数据，要么都从binlog来接，要么都从系统日志来接，最好不要混着接。在不知道数据生产的流程的情况下，一部分通过binlog接入一部分通过系统日志接入，容易出现数据乱序的问题。
- 第二个统一是指实时和离线的统一，这个统一可能更重要一点。虽然我们是建设实时数仓，但是本质上还是数仓，作为一个团队来讲，仓库里的指标的计算逻辑和数据来源应该完全一致，不能让使用数据的人产生误解。如果一个数据两个团队都能为你提供，我们建议选择跟离线同学一致的数据来源。包括我们公司本身也在做一些保证离线和实时采用的数据源一致的工作。

第二个要点就是数据乱序的问题，我们在采集数据的时候会有一个比较大的问题，可能同一条数据，由于分区的存在，这条数据先发生的状态后消费到，后发生的状态先消费到。我们在解决这一问题的时候采用的是美团内部的一个数据组件。

其实，保证数据有序的主要思路就是利用 kafka 的分区来保证数据在分区内的局部有序。至于具体如何操作，可以参考《美团点评基于 Flink 的实时数仓建设实践》。这是我们美团数据同步部门做的一套方案，可以提供非常丰富的策略来保证同一条数据是按照生产顺序进行保序消费的，实现在源头解决数据乱序的问题。

■ DW 层的建设

解决原始数据中数据存在噪声、不完整和数据形式不统一的情况。形成规范，统一的数据源。如果可能的话尽可能和离线保持一致。

明细层的建设思路其实跟离线数仓的基本一致，主要在于如何解决 ODS 层的数据可能存在的数据噪声、不完整和形式不统一的问题，让它在仓库内是一套满足规范的统一的数据源。我们的建议是如果有可能的话，最好入什么仓怎么入仓，这个过程和离线保持一致。

尤其是一些数据来源比较统一，但是开发的逻辑经常变化的系统，这种情况下，我们可能采用的其实是一套基于配置的入仓规则。可能离线的同学有一套入仓的系统，他们配置好规则就知道哪些数据表上数据要进入实时数仓，以及要录入哪些字段，然后实时和离线是采用同一套配置进行入仓，这样就可以保证我们的离线数仓和实时数仓在 DW 层长期保持一个一致的状态。

实际上建设 DW 层其实主要的工作主要是以下4部分。

唯一标红的就是模型的规范化，其实模型的规范化，是一个老生常谈的问题，可能每个团队在建设数仓之前，都会先把自己的规范化写出来。但实际的结果是我们会看到其实并不是每一个团队最终都能把规范落地。

在实时的数仓建设当中，我们要特别强调模型的规范化，是因为实施数仓有一个特点，就是本身实时作业是一个7×24 小时调度的状态，所以当修改一个字段的时候，可能要付出的运维代价会很高。在离线数仓中，可能改了

某一个表，只要一天之内把下游的作业也改了，就不会出什么问题。但是实时数仓就不一样了，只要改了上游的表结构，下游作业必须是能够正确解析上游数据的情况下才可以。

另外使用像 kafka 这样的系统，它本身并不是结构化的存储，没有元数据的概念，也不可能像改表一样，直接把之前不规范的表名、表类型改规范。要在事后进行规范代价会很大。所以建议一定要在建设之初就尽快把这些模型的规范化落地，避免后续要投入非常大的代价进行治理。

- 重复数据处理

除了数据本身我们会在每条数据上额外补充一些信息，应对实时数据生产环节的一些常见问题

内容	生成逻辑	解决问题
唯一键	标记唯一一条数	解决重复数据问题
主键	标记唯一一行数据	分区保证数据有序
版本	对应表结构的版本	解决表结构变化问题
批次	当数据发生重导时更新批次	解决数据重导

- 唯一键和主键

我们会给每一条数据都补充一个唯一键和一个主键，这两个是一对的，唯一键就是标识是唯一一条数据的，主键是标记为一行数据。一行数据可能变化很多次，但是主键是一样的，每一次变化都是其一次唯一的变化，所以会有一个唯一键。唯一键主要解决的是数据重复问题，从分层来讲，数据是从我们仓库以外进行生产的，所以很难保证我们仓库以外的数据是不会重复的。

可能有些人交付数据给也会告知数据可能会有重复。生成唯一键的意思是指我们需要保证 DW 层的数据能够有一个标识，来解决可能由于上游产生的重复数据导致的计算重复问题。生成主键，其实最主要在于主键在 kafka 进行分区操作，跟之前接 ODS 保证分区有序的原理是一样的，通过主键，在 kafka 里进行分区之后，消费数据的时候就可以保证单条数据的消费是有序的。

- 版本和批次

版本和批次这两个其实又是一组。当然这个内容名字可以随便起，最重要的是它的逻辑。

首先，版本。版本的概念就是对应的表结构，也就是 schema 一个版本的数据。由于在处理实时数据的时候，下游的脚本依赖表上一次的 schema 进行开发的。当数据表结构发生变化的时候，就可能出现两种情况：第一种情况，可能新加或者删减的字段并没有用到，其实完全不用感知，不用做任何操作就可以了。另外一种情况，需要用到变动的字段。此时会产生一个问题，在 Kafka 的表中，就相当于有两种不同的表结构的数据。这时候其实需要一个标记版本的内容来告诉我们，消费的这条数据到底应该用什么样的表结构来进行处理，所以要加一个像版本这样的概念。

第二，批次。批次实际上是一个更不常见的场景，有些时候可能会发生数据重导，它跟重启不太一样，重启作业可能就是改一改，然后接着上一次消费的位置启动。而重导的话，数据消费的位置会发生变化。

比如，今天的数据算错了，领导很着急让我改，然后我需要把今天的数据重算，可能把数据程序修改好之后，还要设定程序，比如从今天的凌晨开始重新跑。这个时候由于整个数据程序是一个 7x24 小时的在线状态，其实原先的数据程序不能停，等重导的程序追上新的数据之后，才能把原来的程序停掉，最后使用重导的数据来更新结果层的数据。

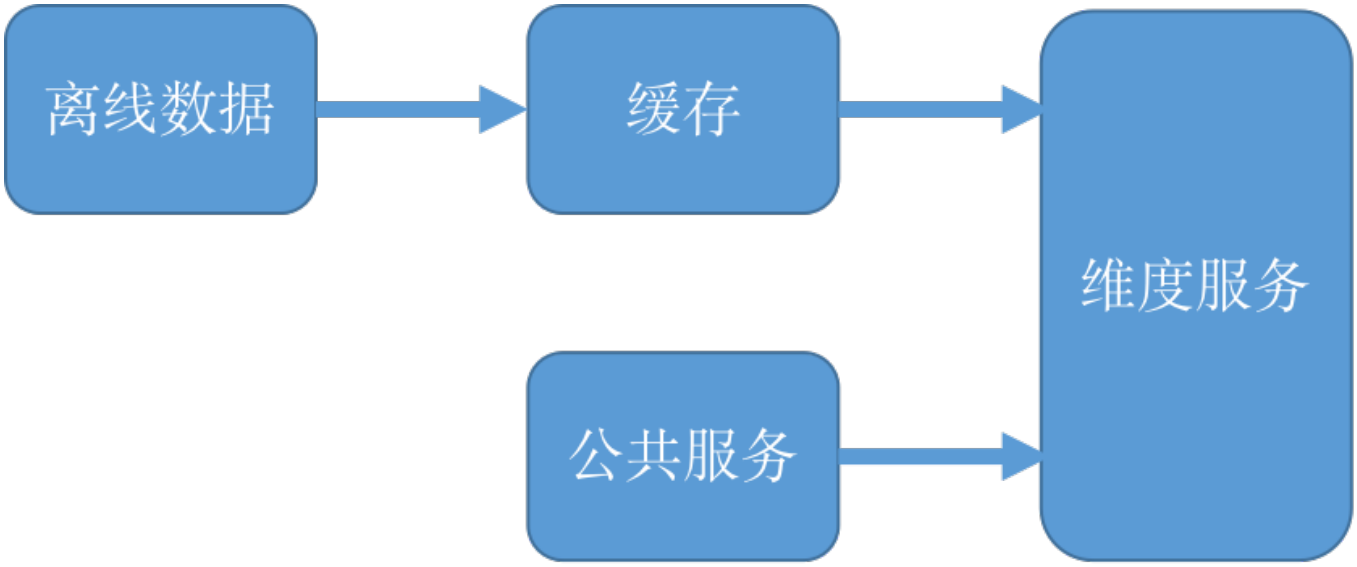
在这种情况下，必然会短暂的存在两套数据。这两套数据想要进行区分的时候，就要通过批次来区分。其实就是所有的作业只消费指定批次的数据，当重导作业产生的时候，只有消费重导批次的作业才会消费这些重导的数据，然后数据追上之后，只要把原来批次的作业都停掉就可以了，这样就可以解决一个数据重导的问题。

■ 维度数据建设

> 其次就是维度数据，我们的明细层里面包括了维度数据。关于维度的数据的处理，实际上是先把维度数据分成了两大类采用不同的方案来进行处理。

- 变化频率低的维度

第一类数据就是一些变化频率比较低的数据，这些数据其实可能是一些基本上是不会变的数据。比如说，一些地理的维度信息、节假日信息和一些固定代码的转换。



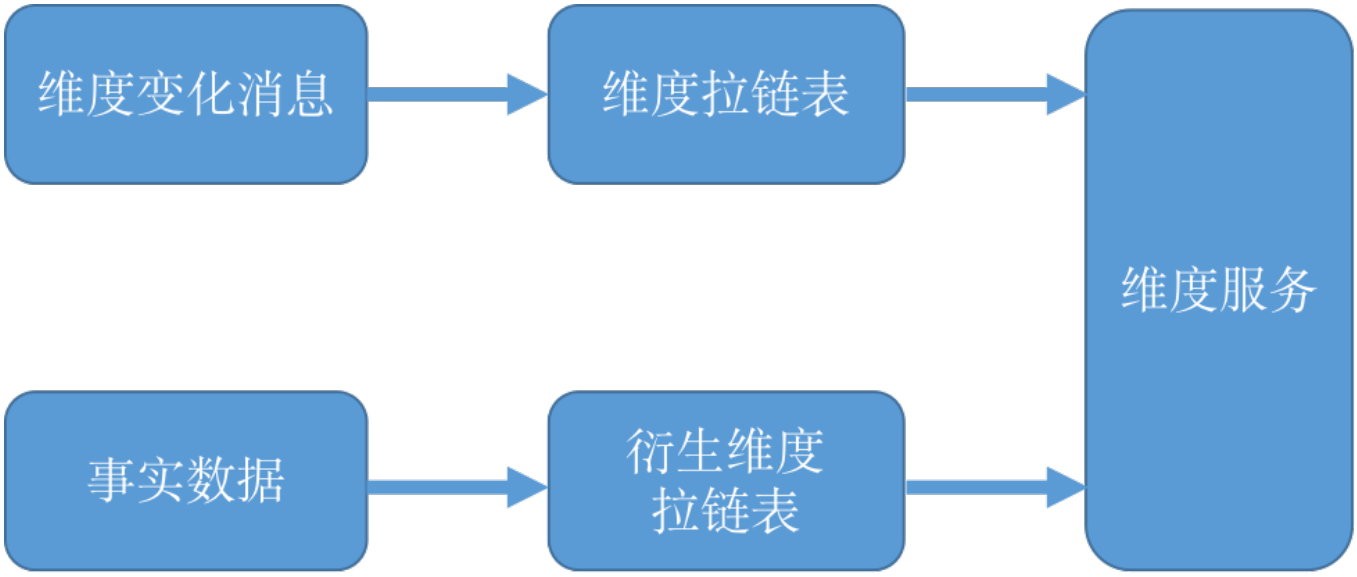
这些数据实际上我们采用的方法就是直接可以通过离线仓库里面会有对应的维表，然后通过一个同步作业把它加载到缓存中来进行访问。还有一些维度数据创建得会很快，可能会不断有新的数据创建出来，但是一旦创建出来，其实也就不再会变了。

比如说，美团上开了一家新的门店，门店所在的城市名字等这些固定的属性，其实可能很长时间都不会变，取最新的那一条数据就可以了。这种情况下，我们会通过公司内部的一些公共服务，直接去访问当前最新的数据。最终，我们会包一个维度服务的这样一个概念来对用户进行屏蔽，具体是从哪里查询相关细节，通过维度服务即可关联具体的维度信息。

- 变化频率高的维度

第二类是一些变化频率较高的数据。比如常见的病人心脑科的状态变动，或者某一个商品的价格等。这些东西往往是会随着时间变化比较频繁，比较快。而对于这类数据，我们的处理方案就稍微复杂一点。首先对于像价格这

样变化比较频繁的这种维度数据，会监听它的变化。比如说，把价格想象成维度，我们会监听维度价格变化的消息，然后构建一张价格变换的拉链表。



一旦建立了维度拉链表，当一条数据来的时候，就可以知道，在这个数据某一时刻对应的准确的维度是多少，避免了由于维度快速的变化导致关联错维度的问题。

另一类如新老客这维度，于我们而言其实是一种衍生维度，因为它本身并不是维度的计算方式，是用该用户是否下过单来计算出来的，所以它其实是用订单数据来算出来的一个维度。

所以类似订单数的维度，我们会在 DW 层建立一些衍生维度的计算模型，然后这些计算模型输出的其实也是拉链表，记录下一个用户每天这种新老客的变化程度，或者可能是一个优质用户的变化的过程。由于建立拉链表本身也要关联维度，所以可以通过之前分组 key 的方式来保障不乱序，这样还是将其当做一个不变的维度来进行关联。

通过这种方式来建立拉链表相对麻烦，所以实际上建议利用一些外部组件的功能。实际操作的时候，我们使用的是 Hbase。HBase 本身支持数据多版本的，而且它能记录数据更新的时间戳，取数据的时候，甚至可以用这个时间戳来做索引。

所以实际上只要把数据存到 HBase 里，再配合上 mini-versions ，就可以保证数据不会超时死掉。上面也提到过，整个实时数仓有一个大原则，不处理离线数仓能处理的过程。相当于处理的过程，只需要处理三天以内的数据，所以还可以通过配置 TTL 来保证 HBase 里的这些维度可以尽早的被淘汰掉。因为很多天以前的维度，实际上也不会再关联了，这样就保证维度数据不会无限制的增长，导致存储爆炸。

■ 维度数据使用

处理维度数据之后，这个维度数据怎么用?

利用 UDTF 关联

通过使用开发 UDTF ， 利用 LATERAL TABLE 进行关联。

通过重新生成 SQL

解析sql 识别维表，以及维表中的字段将原查询转化为：原表.flatmap(维表)

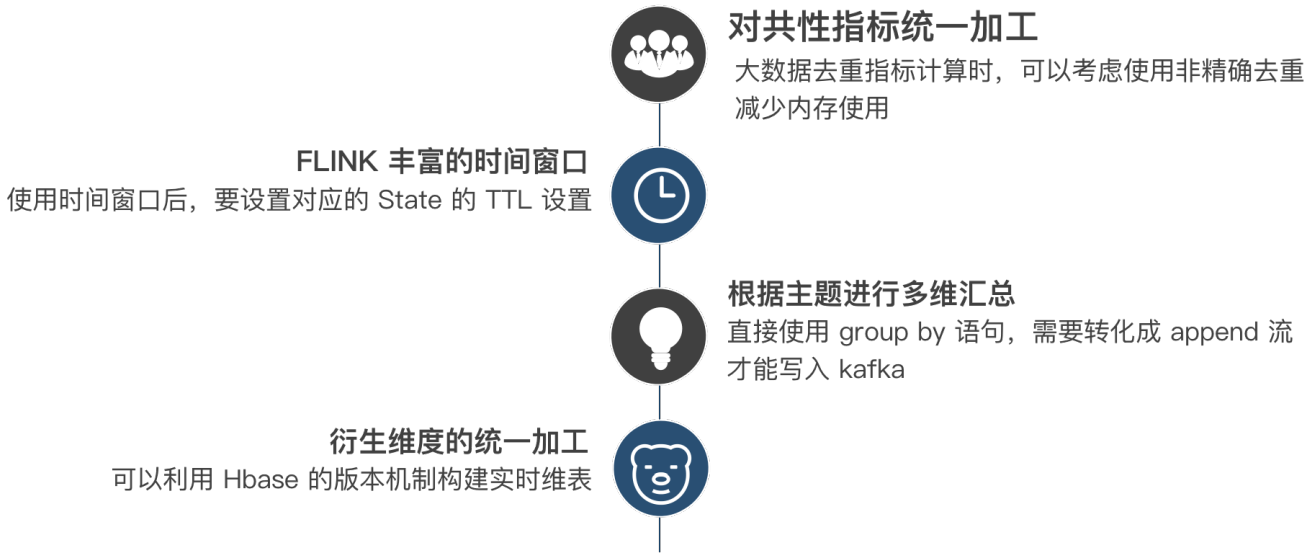
第一种方案，也是最简单的方案，就是使用 UDTF 关联。其实就是写一个 UDTF 去查询上面提到的维度服务，具体来讲就是用 LATERAL TABLE 关键词来进行关联，内外关联都是支持的。

另外一种方案就是通过解析 SQL ，识别出关联的维表以及维表中的字段，把它原本的查询进行一次转化为原表.flatmap (维表)，最后把整个操作的结果转换成一张新的表来完成关联操作。

但是这个操作要求使用者有很多周边的系统来进行配合，首先需要能解析 SQL ，同时还能识别文本，记住所有维表的信息，最后还要可以执行 SQL 转化，所以这套方案适合一些已经有成熟的基于 Flink SQL 的 SQL开发框架的系统来使用。如果只是单纯的写封装的代码，建议还是使用 UDTF 的方式来进行关联会非常的简单，而且效果也是一样的。

■ 汇总层的建设

> 在建设实时数仓的汇总层的时候，跟离线的方案其实会有很多一样的地方。



第一点是对于一些共性指标的加工，比如说 pv、uv、交易额这些运算，我们会在汇总层进行统一的运算。另外，在各个脚本中多次运算，不仅浪费算力，同时也有可能会算错，需要确保关于指标的口径是统一在一个固定的模型里面的。本身 Flink SQL 已经其实支持了非常多的计算方法，包括这些 count distinct 等都支持。

值得注意的一点是，它在使用 count distinct 的时候，他会默认把所有的要去重的数据存在一个 state 里面，所以当去重的基数比较大的时候，可能会吃掉非常多的内存，导致程序崩溃。这个时候其实是可以考虑使用一些非精确系统的算法，比如说 BloomFilter 非精确去重、 HyperLogLog 超低内存去重方案，这些方案可以极大的减少内存的使用。

第二点就是 Flink 比较有特色的一个点，就是 Flink 内置非常多的这种时间窗口。Flink SQL 里面有翻滚窗口、滑动窗口以及会话窗口，这些窗口在写离线 SQL 的时候是很难写出来的，所以可以开发出一些更加专注的模型，

甚至可以使用一些在离线开发当中比较少使用的一些比较小的时间窗口。

比如说，计算最近10分钟的数据，这样的窗口可以帮助我们建设一些基于时间趋势图的应用。但是这里面要注意一点，就是一旦使用了这个时间窗口，要配置对应的 TTL 参数，这样可以减少内存的使用，提高程序的运行效率。另外，如果 TTL 不够满足窗口的话，也有可能会导致数据计算的错误。

第三点，在汇总层进行多维的主题汇总，因为实时仓库本身是面向主题的，可能每一个主题会关心的维度都不一样，所以我们会在不同的主题下，按照这个主题关心的维度对数据进行一些汇总，最后来算之前说过的那些汇总指标。但是这里有一个问题，如果不使用时间窗口的话，直接使用 group by ，它会导致生产出来的数据是一个 retract 流，默认的 kafka 的 sink 它是只支持 append 模式，所以在这里要进行一个转化。

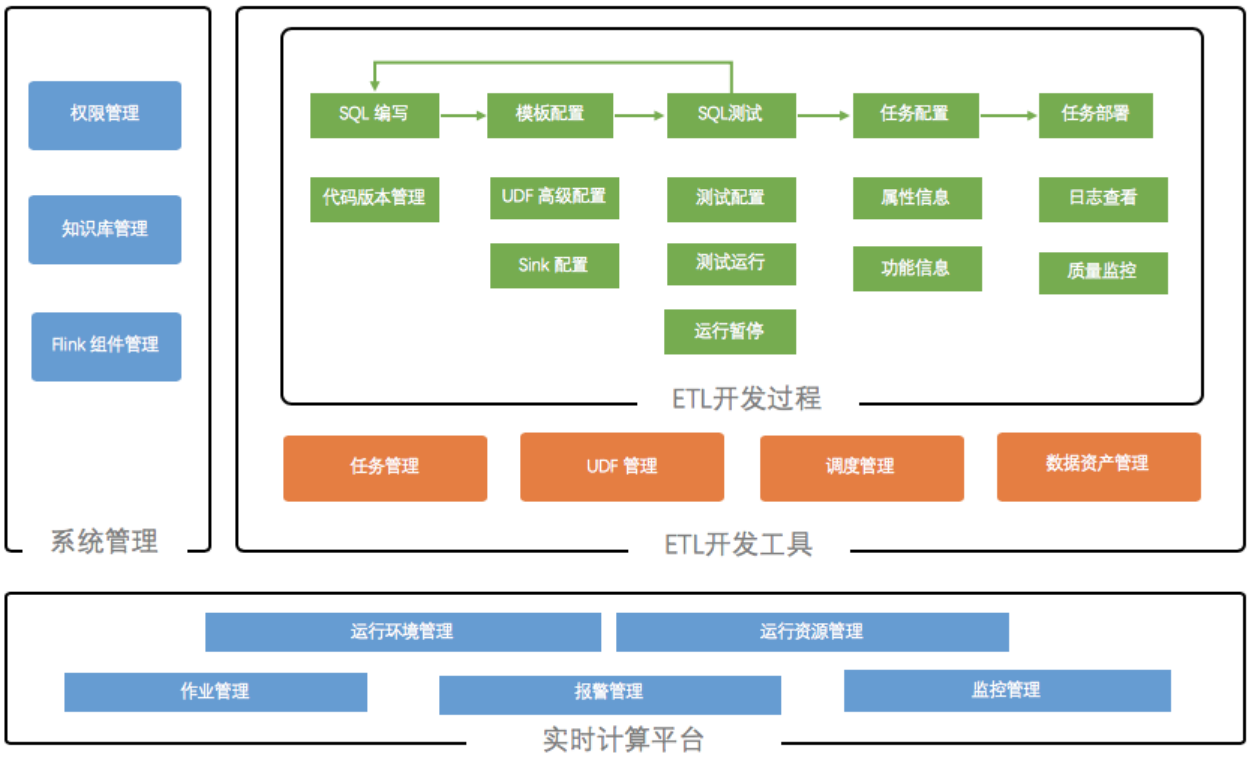
如果想把这个数据写入 kafka 的话，需要做一次转化，一般的转化方案实际上是把撤回流里的 false 的过程去掉，把 true 的过程保存起来，转化成一个 append stream ，然后就可以写入到 kafka 里了。

第四点，在汇总层会做一个比较重要的工作，就是衍生维度的加工。如果衍生维度加工的时候可以利用 HBase 存储，HBase 的版本机制可以帮助你更加轻松地来构建一个这种衍生维度的拉链表，可以帮助你准确的 get 到一个实时数据当时的准确的维度。

仓库质量保证

经过上面的环节，如果你已经建立好了一个仓库，你会发现想保证仓库的正常的运行或者是保证它高质量的运行，其实是一个非常麻烦的过程，它要比一线的操作复杂得多，所以我们在建设完仓库之后，需要建设很多的周边系统来提高我们的生产效率。

下面介绍一下我们目前使用的一些工具链系统，工具链系统的功能结构图如下图。



首先，工具链系统包括一个实时计算平台，主要的功能是统一提交作业和一些资源分配以及监控告警，但是实际上无论是否开发数仓，大概都需要这样的一个工具，这是开发 Flink 的基本工具。

对于我们来讲，跟数仓相关的主要工具有两块:

- 系统管理模块，这个模块实际上是我们的实时和离线是一起使用的。其中知识库管理模块，主要是用来记录模型中表和字段的一些信息，另外就是一些工单的解决方法也会维护进去。Flink 管理主要是用来管理一些我们公司自己开发的一些 Flink 相关的系统组件。
- 重点其实还是我们整个用来开发实时数仓 ETL 的一个开发工具。主要是如下几点：
- SQL 及 UDF 管理，管理 SQL 脚本和 UDF，以及对 UDF 进行配置。
- 任务日志查看和任务监控。
- 调度管理，主要是管理任务的重导和重传。
- 数据资产管理，管理实时和离线的元数据，以及任务依赖信息。

> 其实整个这条工具链，每个工具都有它自己特定的用场场景，下面重点讲解其中两个。

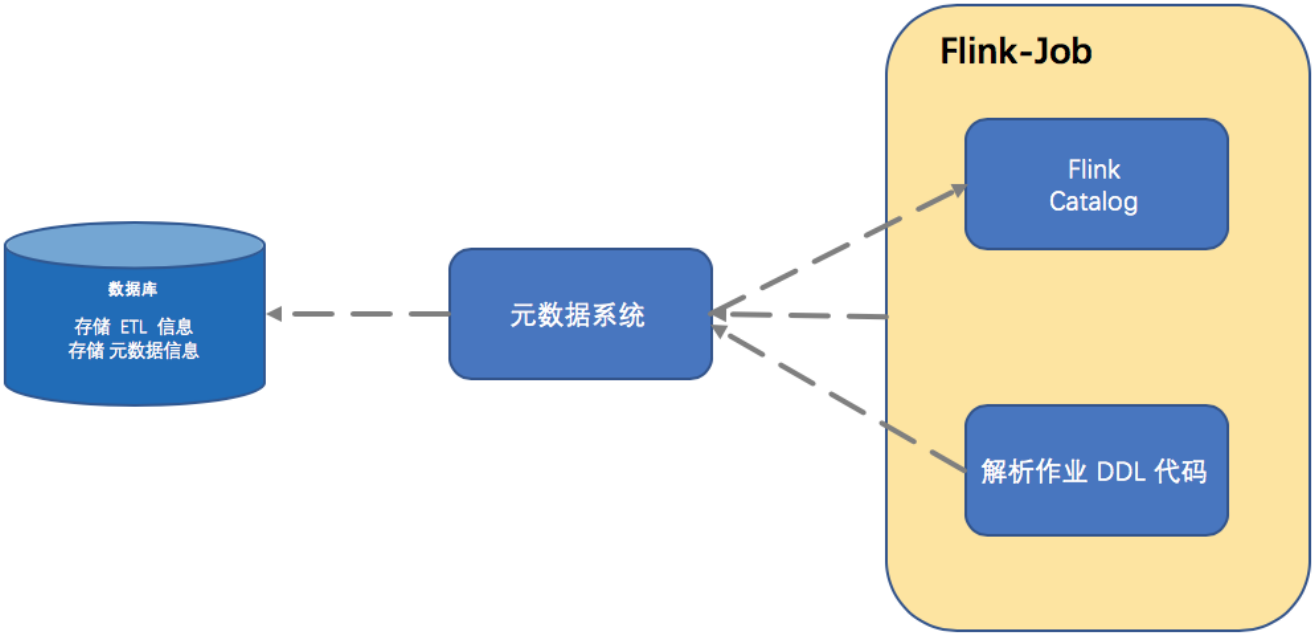
元数据与血缘管理

■ 元数据管理

我们在 Flink SQL 的开发过程中，每一个任务都要重新把元数据重新写一遍。因为 kafka 以及很多的缓存组件，如 Tair、Redis 都不支持元数据的管理，所以我们一定要尽早建设元数据管理系统。

■ 血缘管理

血缘其实对于实时数仓来讲比较重要，在上文中也提到过，在实时的作业的运维过程当中，一旦对自己的作业进行了修改，必须保证下游都是能够准确的解析新数据的这样一个情况。如果是依赖于这种人脑去记忆，比如说谁用我的销售表或者口头通知这种方式来讲的话，效率会非常的低，所以一定要建立一套就是血缘的管理机制。要知道到底是谁用了生产的表，然后上游用了谁的，方便大家再进行修改的时候进行周知，保证我们整个实时数仓的稳定。



元数据和血缘管理系统，最简单的实现方式大概分为以下三点：

- 通过元数据服务生成 Catalog

首先通过元数据系统，把元数据系统里的元数据信息加载到程序中来，然后生成 Flink Catalog 。这样就可以知道当前作业可以消费哪些表，使用哪些表。

- 解析 DDL 语句创建更新表

当作业进行一系列操作，最终要输出某张表的时候，解析作业里面关于输出部分的 DDL 代码，创建出新的元数据信息写入到元数据系统。

- 作业信息和运行状态写入元数据

作业本身的元数据信息以及它的运行状态也会同步到元数据系统里面来，让这些信息来帮助我们建立血缘关系。

最终的系统可以通过数据库来存储这些信息，如果你设计的系统没那么复杂，也可以使用文件来进行存储。重点是需要尽快建立一套这样的系统，不然在后续的开发和运维过程当中都会非常的痛苦。

数据质量验证

将实时数据写入 Hive，使用离线数据持续验证实时数据的准确性。

当建设完一个数仓之后，尤其是第一次建立之后，一定会非常怀疑自己数据到底准不准。在此之前的验证方式就是通过写程序去仓库里去查，然后来看数据对不对。在后续的建设过程中我们发现每天这样人为去对比太累了。

我们就采取了一个方案，把中间层的表写到 Hive 里面去，然后利用离线数据丰富的质量验证工具去对比离线和实时同一模型的数据差异，最后根据设定的阈值进行监控报警。这个方案虽然并不能及时的发现实时数据的问题，但是可以帮助你在上线前了解实时模型的准确程度。然后进行任务的改造，不断提高数据的准确率。另外这个方案还可以检验离线数据的准确性。

以上是美团点评基于 Flink 构建的实时数仓应用经验的分享，希望对大家有所帮助！点击「阅读原文」可回顾作者分享视频～