实时数仓在有赞的实践

作者：小君，部门：技术中台/数据中台

前言

随着实时技术的不断发展和商家实时应用场景的不断丰富，有赞在实时数仓建设方面做了大量的尝试和实践。本文主要分享有赞在建设实时数仓过程中所沉淀的经验，内容包括以下五个部分：

- 建设背景
- 应用场景
- 方案设计
- 项目应用
- 未来展望

建设背景

- 实时需求日趋迫切

产品需求和内部决策对于数据实时性的要求越来越迫切，需要实时数仓的能力来赋能。传统离线数仓的数据时效性是T+1，调度频率以天为单位，无法支撑实时场景的数据需求。即使能将调度频率设置成小时，也只能解决部分时效性要求不高的场景，对于实效性要求很高的场景还是无法优雅的支撑。因此实时使用数据的问题必须得到有效解决。

- 实时技术日趋成熟

实时计算框架已经经历了三代发展，分别是：Storm、SparkStreaming、Flink，计算框架越来越成熟。一方面，实时任务的开发已经能通过编写SQL的方式来完成，在技术层面能很好地继承离线数仓的架构设计思想；另一方面，有赞在线数据开发平台所提供的功能对实时任务开发、调试、运维的支持也日渐趋于成熟，开发成本逐步降低，有助于去做这件事。

应用场景

- 实时BI看板

通过有赞BI工具基于实时数仓创建实时数据集，使用数据集配置柱状图、线图、饼图等图表来呈现实时汇总数据。目前BI工具所支持接入的实时数据源有Druid、MySQL。

- 实时OLAP

实时数仓基于Druid和ClickHouse等OLAP数据库，给用户提供实时数据分析能力。

- 实时数据服务

通过有赞统一数据服务平台OneServices将实时数仓沉淀的实时指标以通用接口的方式提供给业务方使用。目前OneServices所支持接入的实时数据源有MySQL、Kylin、Druid、ClickHouse、Tidb。

- 实时监控告警

通过实时接入应用程序运行的日志数据，经过结构化处理后，加工成预警指标，将异常信息及时推送。

- 实时推荐

通过实时数仓构建用户实时行为特征，提供给算法进行模型训练，给用户推送更加实时的个性化内容。

方案设计

该部分主要介绍实时数仓的方案设计，内容包括：分层设计、ETL设计、数据验证、数据恢复。

分层设计

传统离线数仓的分层设计大家都很熟悉，为了规范的组织和管理数据，层级划分会比较多，在一些复杂逻辑处理场景还会引入临时层落地中间结果以方便下游加工处理。实时数仓考虑到时效性问题，分层设计需要尽量精简，降低中间流程出错的可能性，不过总体而言，实时数仓还是会参考离线数仓的分层思想来设计。

实时数仓分层架构如下图所示 ：

- ODS（实时数据接入层）

ODS层，即实时数据接入层，通过数据采集工具收集各个业务系统的实时数据，对非结构化的数据进行结构化处理，保存原始数据，几乎不过滤数据；该层数据的主要来源有三个部分：第一部分是业务方创建的NSQ消息，第二部分是业务数据库的Binlog日志，第三部分是埋点日志和应用程序日志，以上三部分的实时数据最终统一写入Kafka存储介质中。

ODS层表命名规范：部门名称.应用名称.数仓层级_主题域前缀_数据库名/消息名

例如：接入业务库的Binlog
实时数仓表命名：deptname.appname.ods_subjectname_tablename

例如：接入业务方的NSQ消息
实时数仓表命名：deptname.appname.ods_subjectname_msgname

- DWS（实时明细中间层）

DWS层，即实时明细中间层，该层以业务过程作为建模驱动，基于每个具体的业务过程事件来构

建最细粒度的明细层事实表；比如交易过程，有下单事件、支付事件、发货事件等，我们会基于这些独立的事件来进行明细层的构建。在这层，事实明细数据同样是按照离线数仓的主题域来进行划分，也会采用维度建模的方式组织数据，对于一些重要的维度字段，会做适当冗余。基于有赞实时需求的场景，重点建设交易、营销、客户、店铺、商品等主题域的数据。该层的数据来源于ODS层，通过FlinkSQL进行ETL处理，主要工作有规范命名、数据清洗、维度补全、多流关联，最终统一写入Kafka存储介质中。

DWS层表命名规范：部门名称.应用名称.数仓层级_主题域前缀_数仓表命名

例如：实时事件A的中间层
实时数仓表命名：deptname.appname.dws_subjectname_tablename_eventnameA

例如：实时事件B的中间层
实时数仓表命名：deptname.appname.dws_subjectname_tablename_eventnameB

- DIM（实时维表层）

DIM层，即实时维表层，用来存放维度数据，主要用于实时明细中间层宽化处理时补全维度使用，目前该层的数据主要存储于HBase中，后续会基于QPS和数据量大小提供更多合适类型的存储介质。

DIM层表命名规范：应用名称_数仓层级_主题域前缀_数仓表命名

例如：HBase存储，实时维度表
实时数仓表命名：appname_dim_tablename

- DWA（实时汇总层）

DWA层，即实时汇总层，该层通过DWS层数据进行多维汇总，提供给下游业务方使用，在实际应用过程中，不同业务方使用维度汇总的方式不太一样，根据不同的需求采用不同的技术方案去实现。第一种方式，采用FlinkSQL进行实时汇总，将结果指标存入HBase、MySQL等数据库，该种方式是我们早期采用的方案，优点是实现业务逻辑比较灵活，缺点是聚合粒度固化，不易扩展；第二种方式，采用实时OLAP工具进行汇总，该种方式是我们目前常用的方案，优点是聚合粒度易扩展，缺点是业务逻辑需要在中间层预处理。

DWA层表命名规范：应用名称_数仓层级_主题域前缀_聚合粒度_数据范围

例如：HBase存储，某域当日某粒度实时汇总表
实时数仓表命名：appname_dwa_subjectname_aggname_daily

- APP（实时应用层）

APP层，即实时应用层，该层数据已经写入应用系统的存储中，例如写入Druid作为BI看板的实时数据集；写入HBase、MySQL用于提供统一数据服务接口；写入ClickHouse用于提供实时OLAP服务。因为该层非常贴近业务，在命名规范上实时数仓不做统一要求。

实时ETL

实时数仓ETL处理过程所涉及的组件比较多，接下来盘点构建实时数仓所需要的组件以及每个组件的应用场景。如下图所示：

具体实时ETL处理流程如下图所示：

3.2.1 维度补全

创建调用Duboo接口的UDF函数在实时流里补全维度是最便捷的使用方式，但如果请求量过大，对Duboo接口压力会过大。在实际应用场景补全维度首选还是关联维度表，但关联也存在一定概率的丢失问题，为了弥补这种丢失，可以采用Duboo接口调用兜底的方式来补全。伪代码如下：

```
create function call_dubbo as 'XXXXXXX';
create function get_json_object as 'XXXXXXX';

case
   when cast( b.column as bigint) is not null
      then cast( b.column as bigint)
         else cast(coalesce(cast(get_json_object(call_dubbo('clusterUrl'
                                                ,'serviceName'
                                                ,'methodName'
                                                ,cast(concat('[',cast(a.column as varchar),']') as varchar)
                                                ,'key'
                                                )
                                    ,'rootId')
                                as bigint)
                           ,a.column)
                     as bigint)  end
```

3.2.2 幂等处理

实时任务在运行过程中难免会遇到执行异常的情况，当任务异常重启的时候会导致部分消息重新发送和消费，从而引发下游实时统计数据不准确，为了有效避免这种情况，可以选择对实时消息流做幂等处理，当消费完一条消息，将这条消息的Key存入KV，如果任务异常重启导致消息重新发送的时候，先从KV判断该消息是否已被消费，如果已消费就不再往下发送。伪代码如下：

```
create function idempotenc as 'XXXXXXX';

insert into table
select
   order_no
from
   (
      select
         a.orderNo                                  as  order_no
       , idempotenc('XXXXXXX', coalesce( order_no, '') )  as  rid
      from
         table1
   ) t
where
   t.rid = 0;
```

数据验证

由于实时数仓的数据是无边界的流，相比于离线数仓固定不变的数据更难验收。基于不同的场景，我们提供了2种验证方式，分别是：抽样验证与全量验证。如图3.3所示

- 抽样验证方案

该方案主要应用在数据准确性验证上，实时汇总结果是基于存储在Kafka的实时明细中间层计算而来，但Kafka本身不支持按照特定条件检索，不支持写查询语句，再加上消息的无边界性，统计结果是在不断变化的，很难寻找参照物进行比对。鉴于此，我们采用了持久化消息的方法，将消息落盘到TiDB存储，基于TiDB的能力对落盘的消息进行检索、查询、汇总。编写固定时间边界的测试用例与相同时间边界的业务库数据或者离线数仓数据进行比对。通过以上方式，抽样核心店铺的数据进行指标准确性验证，确保测试用例全部通过。

- 全量验证方案

该方案主要应用在数据完整性和一致性验证上，在实时维度表验证的场景使用最多。大体思路：将存储实时维度表的在线HBase集群中的数据同步到离线HBase集群中，再将离线HBase集群中的数据导入到Hive中，在限定实时维度表的时间边界后，通过数据平台提供的数据校验功能，比对实时维度表与离线维度表是否存在差异，最终确保两张表的数据完全一致。

数据恢复

实时任务一旦上线就要求持续不断的提供准确、稳定的服务。区别于离线任务按天调度，如果离线任务出现Bug，会有充足的时间去修复。如果实时任务出现Bug，必须按照提前制定好的流程，严格按照步骤执行，否则极易出现问题。造成Bug的情况有非常多，比如代码Bug、异常数据Bug、实时集群Bug，如下图展示了修复实时任务Bug并恢复数据的流程。

项目应用

项目一：视频号直播间实时数据统计

微信视频号的用户数已超过2亿，是微信生态内较大的流量入口，视频号直播成为一个重要的线上消费场景。目前，有赞已接入了小程序交易组件，直播时借助小程序的商品交易能力挂载有赞商品进行直播卖货。商家在直播过程中以及直播结束后都需要实时的关注直播的交易数据，根据直播间的实时数据反馈来动态调整经营策略。

本次项目涉及实时指标主要有支付订单数、支付订单金额、支付商品件数、支付人数、支付新客数等，汇总粒度有直播间粒度、商品粒度等。由于直播间的开始时间和结束时间存在跨天的情况，无法按照自然天实时预聚合；对于已经结束的直播计划商家需要能查看历史效果数据。基于以上的产品需求，对于存在跨天聚合的场景，我们采用ClickHouse存明细的方式进行实时汇总；对于支持查看历史效果数据，我们采用实时数据+离线数据的方式去组装最新的全量数据。
由于离线分区数据在凌晨跑批更新的时候存在执行和替换的时间，如何确保每时每刻服务都能查到最全的明细数据，这是实时数据+离线数据
方案设计的关键。我们基于ClickHouse设计两种类型的分区：一种是实时分区，realtime_yyyymmdd；另一种是离线分区，offline_yyyymmdd

- 当现在是T日时，见Step1

最新全量数据 = T日实时数据 + (T-1)日离线数据

- 当现在是T+1日时 (T日离线数据正在执行中还未产出)，见Step2

最新全量数据 = T日实时数据 + （T+1)日实时数据 + (T-1)日离线数据

- 当现在是T+1日时 (T日离线数据已产出)，见Step3

最新全量数据 = （T+1)日实时数据 + T日离线数据

具体过程如下图，何时从Step2切换到Step3，我们通过服务层代码判断，使用此方案能确保应用端接口在每时每刻的查询都是最新的全量数据。对于实时分区和离线分区分别都配置4天有效期，过期失效的历史分区会自动清理。如图下图所示

关于新老客户数，这类指标在计算前涉及与历史数据进行比较，所以势必要维护一张动态更新的维度表，在计算新老客户数前，流入进来的消息先与动态维表进行关联，当消息的支付时间比维表的支付时间大则为老客数据；当消息无法关联上维表里的数据时则为新客数据，随后立即更新HBase维度表，当该用户再次支付所产生的消息则统计为老客数。伪代码如下：

```
insert into table
select
   ...
 , case
      when b.rowkey is not null
         then b.paid_order_first_time
            else a.paid_order_time
               end
from
   table1 a
left join
   table2 for SYSTEM_TIME as of a.proctime as b
on
   a.column1 = b.rowkey
```

具体过程如图下图所示：

以上是在项目开发过程中涉及的部分技术方案，项目上线后的效果如下图：

项目二：销售员实时分析

销售员是有赞推出的一款可帮助商家拓宽推广渠道的应用营销工具。商家可通过制定推广计划招

募买家加入推广队伍，并在其成功推广后给予一定佣金奖励，以此给店铺带来更多的传播和促进销量提升。销售员分析功能可以让商家直观的看到销售员的经营数据，从而精细化运用销售员，提升管理效率。

本次项目涉及实时指标主要有今日新增销售员数、今日新增下级销售员数、今日销售员成交金额、今日成交客户数、今日支出佣金等。以上指标都是当日实时指标，不用考虑离线场景，所以使用纯实时数据部分就能支撑。结合业务数据量级我们选型使用MySQL存储实时预聚合的数据，后续业务方通过OneServices提供的接口来获取数据进行实时展示。

部分代码如下图所示：

下面这张图有赞OneServices所提供的能力，通过配置的方式，将底层在线存储引擎映射成Dubbo接口，提供上层应用访问。


```
public class Request implements Serializable {

    //请求唯一标识,填写uuid即可
    String requestId;

    //接入应用
    String caller;

    //秘钥
    String secret;

    //API名称
    String apiName;

    //请求参数
    Map params;

    //是否开启debug模式,如果开启,将返回更多的信息
    Boolean debugMode = false;

    //系统调用相关补充信息,可不填
    Map additional;
}
```

以上是在项目开发过程中涉及的部分技术方案，项目上线后的效果如下图：

项目三：消费者机器人服务实时统计

有赞消费者机器人服务占整体服务量的比重已较高，但目前只能查看离线数据，较为滞后，无法针对随时爆发的线上问题及时发现并布防，因此需要开发实时BI看板，以便于实时监控各个来源的流入以及转人工情况，便于发现异常点，及时止损，降低转人工率。

本次项目涉及的实时指标主要有问题流入量、机器人会话量、机器人转人工会话量、直连人工会话量等。实时数据源涉及客户问题入口记录表和机器人会话表，通过监听Binlog的方式实时采集数据，在机器人会话中存储了问题编码，但通过问题编码无法确定问题来源类型，因此必须要将以上两个数据流通过双流JOIN的方式连接在一起，才能区分哪些是消费者机器人所服务的问题以及会话。

流A是客户问题流入的消息流，流B是基于客户问题机器人进行自动回复的会话消息流。客户问题产生会先于机器人服务会话，会话存在有效期，超过有效期问题未结束重算一次新的服务会话。问题会话量的统计周期是一天，超过一天即使问题未结束也不再计算会话量。以上需求是一个明显的双流JOIN场景，如图4.3.1所示，我们采用FlinkSQL提供的Interval JOIN功能。首先A流的服务ID必须等于B流的服务ID，其次A流的创建时间满足以下表达式 a.created_time between b.created_time and b.created_time + INTERVAL '1' DAY ，伪代码如下：

```
from
  (
    select
      key
     , column
     , created_time
    from
      table1
  ) a
inner join
  (
    select
      key
     , created_time
    from
      table2
  ) b
on
  a.column = b.column
where
```

a.created_time between b.created_time and b.created_time + INTERVAL '1' DAY

出于性能和存储的考虑，要将过期的数据清除，通过所配置的INTERVAL参数，将过期的STATE自动清除。

在机器人会话表中如果客户多次关闭会话接着再次打开会话（在会话有效期内），在数据库层面会生成多条记录，但是会话编码不变。如果不对数据去重会导致实时统计结果偏高，但是精确去重对性能消耗较大，会影响到数据产出的时效性，与需求方沟通后，决定采用非精确去重，在使用Druid接入Kafka后，对去重字段使用hyperUnique
算法处理，损失一定精度的准确性，确保实时汇总数据产出的时效性。

如下图所示：

以上是在项目开发过程中涉及的部分技术方案，项目上线后的效果如下图所示：

未来展望

在实时数仓方面，我们未来有几个重点事情：

- 实时数仓主题域建设覆盖面更广，支撑更多的业务应用
- 建立实时数仓价值评估体系，量化投入与产出
- 推进在线平台能力的完善，优化实时任务血缘、简化参数配置、本地代码调试等提效的功能