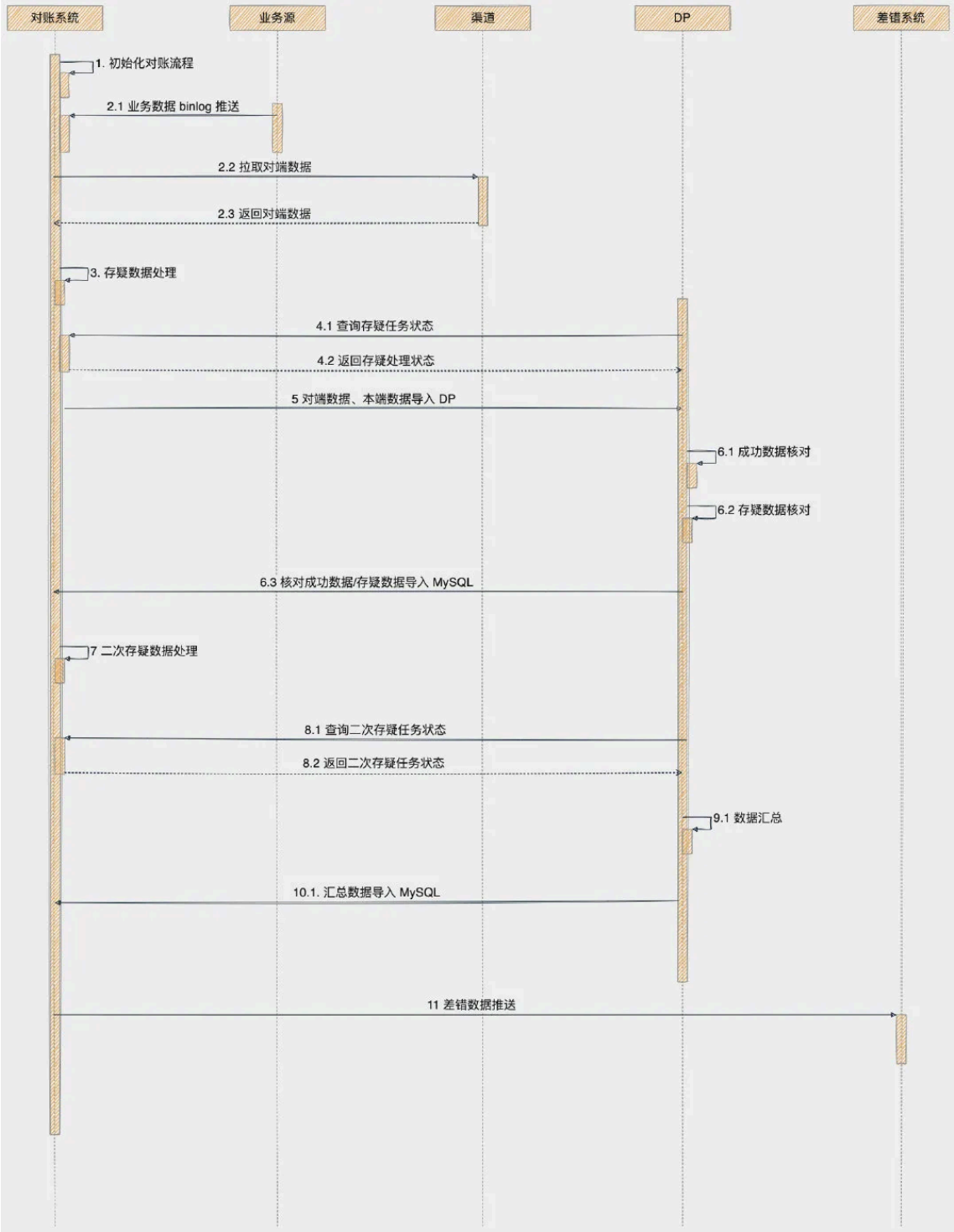
千万级支付对账系统怎么玩（下篇）？

原创 楼下小黑哥 小黑十一点半 2022年01月10日 08:39

上篇文章我们讲到对账系统收集数据的流程，下面我们再来讲下数据核对的流程。

这里再放一下支付对账系统整个流程，忘记的同学可以看这个图片再回忆一下。



数据导入DP

在 DP 核对之前，我们需要将对账系统收集的数据，从 MySQL 导入 DP Hive 表中。

DP 任务调度开始，DP 平台定时检测对账系统提供 HTTP 接口，判断本次存疑流程是否处理完成。

如果完成，自动触发将数据从 MYSQL 导入 DP Hive 表中。

数据导入之后，将会开始 DP 核对规程。这个过程就是整个对账流程最关键的部分，这个流程核对两端数据，检查两端是否存在差异数据。

DP 核对

数据导入结束，DP 平台开始核对数据，这个过程分为两个核对任务：

- 成功数据核对
- 存疑数据核对

成功数据核对

成功数据核对任务，核对的目的是为了核对出本端与对端支付单号与金额一致的数据。

这里的核对任务使用了 Hive SQL ，整个 SQL 如下所示：

```
---- A
CREATE TABLE IF NOT EXISTS dp.pay_check_success (
    `batch_no` bigint comment '批次号',
    `merchant_no` string comment '三方商户号',
    `sub_merchant_no` string comment '三方子商户号',
    `biz_id` string comment '对账业务关联字段',
    `biz_amount` bigint comment '金额',
    `biz_date` string comment '业务日期',
    `biz_type` int comment '业务类型',
    `status` int comment '状态标识',
    `remark` string comment '备注',
    `create_time` string comment '创建时间',
    `update_time` string comment '修改时间',
    `trade_date` int comment '订单交易日期',
    `channel_code` int comment '渠道类型'
);

----B
insert
    overwrite table dp.pay_check_success
select
    tb1.batch_no as batch_no,
    tb1.merchant_no as merchant_no,
    tb1.sub_merchant_no as sub_merchant_no,
```

```
    tb1.biz_id as biz_id,
    tb1.biz_amount as biz_amount,
    tb1.biz_date as biz_date,
    tb1.biz_type as biz_type,
    tb1.status as status,
    tb1.remark as remark,
    tb1.trade_date as trade_date,
    tb1.channel_code as channel_code
from
    (
        select
            tb2.batch_no as batch_no,
            tb1.merchant_no as merchant_no,
            tb1.sub_merchant_no as sub_merchant_no,
            tb1.biz_order_no as biz_id,
            tb1.trader_amount as biz_amount,
            '${DP_1_DAYS_AGO_Ymd}' as biz_date,
            '0' as status,
            '' as remark,
            '${DP_1_DAYS_AGO_Ymd}' as trade_date,
            tb1.channel_code as channel_code
        from
            dp.pay_check_record tb1
            inner join dp.pay_check_channel_record tb2 on tb1.biz_order_no = tb2.biz_order_no
            and tb1.trader_amount = tb2.trader_amount
            and tb1.channel_code = tb2.channel_code
        where
            tb1.is_check = 0
            and tb2.is_check = 0
            and tb1.bill_date = '${DP_1_DAYS_AGO_Ymd}'
            and tb2.bill_date = '${DP_1_DAYS_AGO_Ymd}'
            and tb1.is_filter = 0
    ) tb1
```

整个 SQL 分为两部分，第一部分将会在 DP 中创建一张 pay_check_success，记录核对成功的数据。

第二部分，将核对成功的数据插入上面创建的 pay_check_success 表中。

查找核对成功的数据 SQL 如下：

```
        select
            tb2.batch_no as batch_no,
            tb1.merchant_no as merchant_no,
            tb1.sub_merchant_no as sub_merchant_no,
            tb1.biz_order_no as biz_id,
```

```
        tb1.trader_amount as biz_amount,
        '${DP_1_DAYS_AGO_Ymd}' as biz_date,
        '0' as status,
        '' as remark,
        '${DP_1_DAYS_AGO_Ymd}' as trade_date,
        tb1.channel_code as channel_code
    from
        dp.pay_check_record tb1
        inner join dp.pay_check_channel_record tb2 on tb1.biz_order_no = tb2.biz_order_no
        and tb1.trader_amount = tb2.trader_amount
        and tb1.channel_code = tb2.channel_code
    where
        tb1.is_check = 0
        and tb2.is_check = 0
        and tb1.bill_date = '${DP_1_DAYS_AGO_Ymd}'
        and tb2.bill_date = '${DP_1_DAYS_AGO_Ymd}'
        and tb1.is_filter = 0
```

> 上述 SQL 存在一些 DP 平台系统变量。DP_1_DAYS_AGO_Ymd 代表当前日期的前一天

主要逻辑非常简单，利用 sql 内连接查询的功能，可以查找单号,金额,渠道编码一致的数据。

成功数据核对任务结束，将会把刚才在 DP 中创建的 pay_check_success 同步回对账系统的 MYSQL 数据库中。

存疑数据核对

存疑数据核对任务，核对的目的是为了核对出本端与对端支付单号或金额不一致的数据。

这些数据将会当做存疑数据，这些数据将会在第二阶段存疑数据处理。

这里的核对任务也是使用了 Hive SQL ，整个 SQL 跟上面比较类似，SQL 如下所示：

```
CREATE TABLE IF NOT EXISTS dp.check_dp_buffer_record (
    `biz_id` string comment '订单号',
    `order_type` string comment '订单类型 0本端订单 1渠道订单',
    `bill_date` int comment '对账日期',
    `biz_type` int comment '业务类型',
    `channel_code` int comment '渠道类型',
    `amount` string comment '金额',
    `merchant_no` string comment '商户号',
    `sub_merchant_no` string comment '三方子商户号',
    `trade_date` int comment '交易日期',
```

```
    `create_time` string comment '创建时间',
    `update_time` string comment '修改时间'
);

insert
    overwrite table dp.check_dp_buffer_record
select
    tb1.biz_id as biz_id,
    tb1.order_type as order_type,
    tb1.bill_date as bill_date,
    tb1.biz_type as biz_type,
    tb1.channel_code as channel_code,
    tb1.amount as amount,
    tb1.merchant_no as merchant_no,
    tb1.sub_merchant_no as sub_merchant_no,
    tb1.trade_date as trade_date,
    '${DP_0_DAYS_AGO_Y_m_d_HMS}',
    '${DP_0_DAYS_AGO_Y_m_d_HMS}'
FROM
    (
        select
            tb1.biz_order_no as biz_id,
            0 as order_type,
            tb1.bill_date as bill_date,
            10 as biz_type,
            tb1.channel_code as channel_code,
            tb1.trade_amount as amount,
            tb1.merchant_no as merchant_no,
            tb1.sub_merchant_no as sub_merchant_no,
            '${DP_1_DAYS_AGO_Ymd}' as trade_date
        FROM
            (
                select
                    biz_order_no,
                    bill_date,
                    channel_code,
                    trade_amount,
                    merchant_no,
                    sub_merchant_no
                from
                    ods.pay_check_record
                where
                    and bill_date = '${DP_1_DAYS_AGO_Ymd}'
                    and is_filter = 0
                    and is_check = 0
            ) tb1
            LEFT JOIN (
                select
                    biz_order_no,
                    trade_amount,
                    channel_code
                from
                    ods.pay_check_channel_record
                where
                    and bill_date = '${DP_1_DAYS_AGO_Ymd}'
                    and is_check = 0
            ) tb2 ON tb1.biz_order_no = tb2.biz_order_no
            and tb1.trade_amount = tb2.trade_amount
            and tb1.channel_code = tb2.channel_code
```

```
        where
            tb2.biz_order_no IS NULL
        union
        select
            tb1.biz_order_no as biz_id,
            1 as order_type,
            tb1.bill_date as bill_date,
            10 as biz_type,
            tb1.channel_code as channel_code,
            tb1.trade_amount as amount,
            tb1.merchant_no as merchant_no,
            tb1.sub_merchant_no as sub_merchant_no,
            '${DP_1_DAYS_AGO_Ymd}' as trade_date
        FROM
            (
                select
                    biz_order_no,
                    bill_date,
                    channel_code,
                    trade_amount,
                    merchant_no,
                    sub_merchant_no
                from
                    ods.pay_check_chnnel_bill
                where
                    and bill_date = '${DP_1_DAYS_AGO_Ymd}'
                    and is_check = 0
            ) tb1
            LEFT JOIN (
                select
                    biz_order_no,
                    channel_code,
                    trade_amount
                from
                    ods.pay_check_record
                where
                    and bill_date = '${DP_1_DAYS_AGO_Ymd}'
                    and is_filter = 0
                    and is_check = 0
            ) tb2 ON tb1.biz_order_no = tb2.biz_order_no
            and tb1.trade_amount = tb2.trade_amount
            and tb1.channel_code = tb2.channel_code
        where
            tb2.biz_order_no IS NULL
    ) tb1;
```

整个 SQL 分为两部分，第一部分将会在 DP 中创建一张 check_dp_buffer_record，记录核对差异的的数据。

第二部分，将核对差异的数据插入上面创建的 check_dp_buffer_record 表中。

查找差异数据较为麻烦，需要分成两部分收集：：

- 本端单边账，即本端存在数据，但是对端不存在数据
- 渠道端单边账，即对端存在数据，本端不存在数据

两边数据查找到之后，使用 SQL union 功能，将两端数据联合。

我们先来看下本端单边张的逻辑的：

```
select
    tb1.biz_order_no as biz_id,
    0 as order_type,
    tb1.bill_date as bill_date,
    10 as biz_type,
    tb1.channel_code as channel_code,
    tb1.trade_amount as amount,
    tb1.merchant_no as merchant_no,
    tb1.sub_merchant_no as sub_merchant_no,
    '${DP_1_DAYS_AGO_Ymd}' as trade_date
FROM
    (
        select
            biz_order_no,
            bill_date,
            channel_code,
            trade_amount,
            merchant_no,
            sub_merchant_no
        from
            ods.pay_check_record
        where
            and bill_date = '${DP_1_DAYS_AGO_Ymd}'
            and is_filter = 0
            and is_check = 0
    ) tb1
    LEFT JOIN (
        select
            biz_order_no,
            trade_amount,
            channel_code
        from
            ods.pay_check_channel_record
        where
            and bill_date = '${DP_1_DAYS_AGO_Ymd}'
            and is_check = 0
    ) tb2 ON tb1.biz_order_no = tb2.biz_order_no
    and tb1.trade_amount = tb2.trade_amount
    and tb1.channel_code = tb2.channel_code
where
    tb2.biz_order_no IS NULL
```

SQL 看起来比较复杂，实际逻辑可以简化为下面 SQL：

```
select
    *
from
    innerTab t1
    LEFT JOIN channelTab t2 ON t1.biz_order_no = t2.biz_order_no
    and t1.trade_amount = t2.trade_amount
    and t1.channel_code = t2.channel_code
where
    t2.biz_order_no is null;
```

这里主要利用 SQL 左连接的功能，本端数据 left join 渠道数据，如果渠道单号不存在，则认为本端数据存在，渠道数据不存在，当然也有可能是两端数据都存在，但是金额不相等。

这种情况记为本端数据存疑，orderType 为 0。

渠道端单边账收集逻辑：

```
select
    tb1.biz_order_no as biz_id,
    1 as order_type,
    tb1.bill_date as bill_date,
    10 as biz_type,
    tb1.channel_code as channel_code,
    tb1.trade_amount as amount,
    tb1.merchant_no as merchant_no,
    tb1.sub_merchant_no as sub_merchant_no,
    '${DP_1_DAYS_AGO_Ymd}' as trade_date
FROM
    (
        select
            biz_order_no,
            bill_date,
            channel_code,
            trade_amount,
            merchant_no,
            sub_merchant_no
        from
            ods.pay_check_chnnel_bill
        where
            and bill_date = '${DP_1_DAYS_AGO_Ymd}'
            and is_check = 0
    ) tb1
    LEFT JOIN (
        select
            biz_order_no,
            channel_code,
            trade_amount
        from
            ods.pay_check_record
        where
            and bill_date = '${DP_1_DAYS_AGO_Ymd}'
            and is_filter = 0
            and is_check = 0
    ) tb2 ON tb1.biz_order_no = tb2.biz_order_no
```

```
    and tb1.trade_amount = tb2.trade_amount
    and tb1.channel_code = tb2.channel_code
where
    tb2.biz_order_no IS NULL
```

逻辑与本端单边账收集类似，渠道数据 left join 本端数据，如果本端单号不存在，则为渠道数据存在，本端数据不存在。当然也有可能是两端数据都存在，但是金额不相等。

这里记为渠道存疑数据，orderType 为 1

成功数据核对以及存疑数据核对结束，DP 平台将会自动把数据从 Hive 表中导入到 MYSQL。

数据导出结束，DP 平台将会调用对账系统的相关接口，通知对账系统 DP 核对流程结束。

DP 核对流程是整个对账流程核心流程，目前千万级数据的情况下，大概能在一个小时之内搞定。

DP 核对流程结束之后，对账系统开始下个流程-二次存疑数据处理

二次存疑数据处理

前面流程我们讲到存疑处理，为什么这里还需要二次存疑数据处理呢？

这因为 DP 核对存疑数据收集的过程，我们使用业务单号与金额去互相匹配，那如果不存在，有可能是因为两端数据有一端不存在，还有可能是因为两端数据数据都存在，但是金额却不相等。

DP 核对过程是无法区分出这两种情况，所以增加一个二次存疑数据处理流程，单独区分出这两类数据。

回到二次存疑数据处理流程，当天产生的所有存疑数据都从 DP 中导入到 check_dp_buffer_record 表。

二次存疑数据处理流程将会查找 check_dp_buffer_record 表所有未核对的记录，然后依次遍历。

遍历过程中将会尝试在 check_dp_buffer_record 表中查找相反方向的存疑数据。

这个可能不好理解，举个例子：

假如有一笔订单，本端是 100 元，渠道端是 10 元。这种情况两笔记录都会出现在 check_dp_buffer_record 表。

遍历到本端这笔的时候，这笔类型是本端存疑，type为 0。使用者本端单号从 check_dp_buffer_record 查找渠道端存疑（type 为 1）的数据。

上面的情况可以找到，证明这笔存疑数据其实是金额不相等，这里需要将数据移动到差错表。

那如果是正常一端缺失的数据，那自然去相反方向查找是找不到的，这种数据是正常存疑数据，移动内部存疑表。

对账系统二次存疑数据处理结束之后，开始下一个阶段数据汇总。

数据汇总

数据汇总阶段就是为了统计当天每个有多少成功功对账数据，多少存疑数据，统计结束通过看板给相关运营人员展示统计数据。

由于数据量大的问题，这里使用的是 DP 平台 Sprak 任务进行任务统计。

这里逻辑简单解释为，就是利用 Scala 脚本代码对数据进行相关求和，这里代码没有普遍性，就不展示具体的逻辑了。

差错数据推送

数据汇总结束之后，开始下一个阶段，差错数据推送给差错系统。

上面存疑数据处理的流程中转化的差错数据，当前存在对账系统内部差错数据表中。

目前我们差错数据是是另外一个差错系统单独处理，所以对账系统需要把差错数据表数据推送给差错系统。

这里的逻辑比较简单，查找所有待处理的差错数据，遍历发送 NSQ 消息给差错系统。

总结

千万级数据对账整个流程看起，其实相关操作流程都不是很难。

那我个人认为这里难点在于第一需要一套完整大数据平台体系，第二改变原有对账方式，思考如何将对账系统与大数据平台一起串起来。

希望这篇文章对正好碰到该类问题同仁起到相关帮助。

预告一下，上面我们讲的其实都是业务明细的对账，这一部分其实在财务领域被叫做，账账对账。

那实际上我们还需要核对当天应到资金与实到资金是否一致，下一篇文章我们聊聊这个。

支付 3 #对账系统 4 系统架构设计 5 支付那些事 15

支付 · 目录

上一篇 · 千万级支付对账系统怎么玩（上篇）？