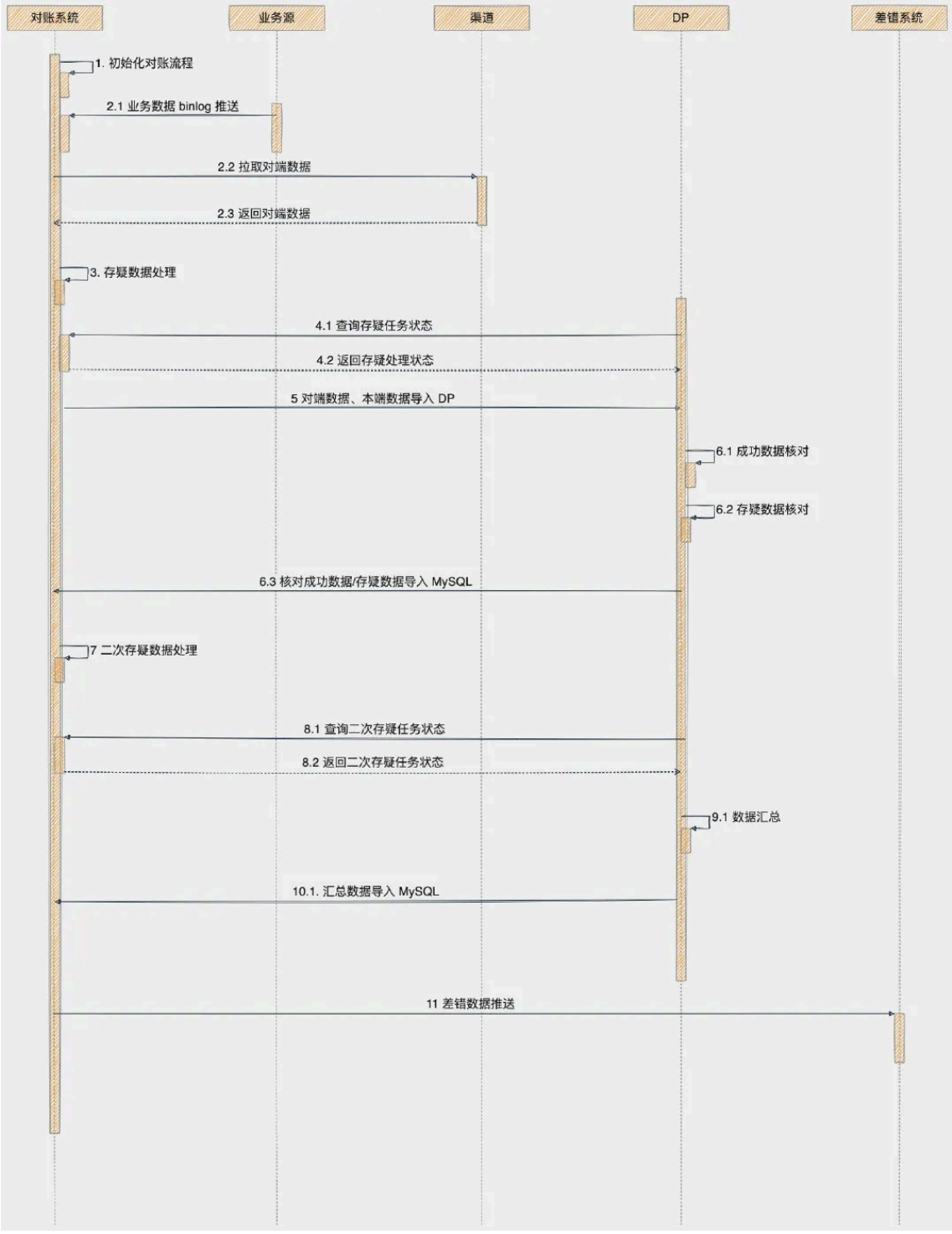
千万级支付对账系统怎么玩（上篇）？

原创 楼下小黑哥 小黑十一点半 2022年01月04日 08:40

上篇文章聊到了对账系统业务逻辑以及千万数据集对账系统存在的难点，这篇文章就来聊下千万级数据集下对账系统实现方案。

首先我们先来看下对账整体时序图，先有个印象：



下面整篇文章将会围绕上面时序图开始讲解，由于文章篇幅过长，所以文章将会拆分成上下两部分。

数据平台

上次文章中提到，千万级数据需要使用 Hive，Spark等相关大数据技术，这就离不开大数据平台的技术支持。

简单聊下我们这边大数据平台DP （Data Platform），它提供用户大数据离线任务开发所需要的环境、工具以及数据，具有入口统一性、一站式、简化hadoop本身的复杂性、数据安全等特点。

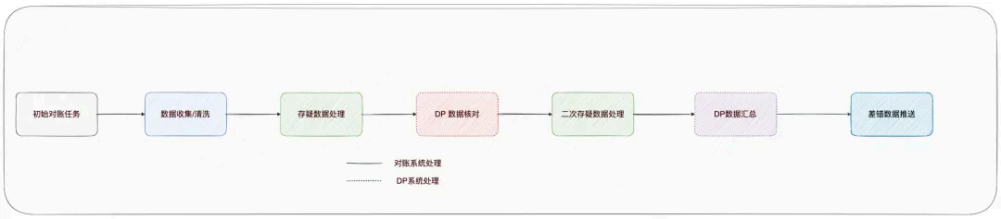
DP 平台提供功能如下：

- 数据双向离线同步，MySQL 与 Hive 互相同步
- 大数据离线计算，支持SQL（SparkSQL/HiveSQL/Presto)形式处理各类的数据清洗、转化、聚合操作，也支持使用MapReduce、Spark等形式，处理比较复杂的计算场景
- 即时的SQL查询，允许用户即时的执行SQL、查看执行的日志和结果数以及进行结果数据的可视化分析
- 数据报表

那本篇文章不会涉及具体的大数据技术相关的实现细节,相关原理（主要是咱也不会~），主要聊下对账系统如何联合 DP 平台实现完整数据对账方案。

对账系统概览

开头的时序图，我们可以看到整个对账过程设计好几个业务流程，那在这里对账系统内部将会维护一个流程状态机，当前一个流程处理结束之后，下一个流程才能被触发。



由于当前对账系统实现方案，涉及对账系统与 DP 平台，对账系统目前没办法调用 DP 平台触发任务，但是 DP 平台可以通过通过 HTTP 接口调用对账系统。

所以当前流程触发的方式使用的是定时任务的方案，每个流程有一个单独的定时任务。

对账系统内的定时任务触发的时候，将会判断当前流程是否已经到达执行条件，即判断一下当前任务的状态。

每个定时任务触发时间人为设置的时候，岔开一两分钟，防止同时运行。

DP 平台使用自带调度任务，对账系统无法控制 DP 任务的运行。

DP 平台定时任务可以通过运行 Scala 脚本代码，调用对账系统提供 HTTP 查询接口，通过这种方式判断当前流程是否已经到达执行条件。

下面详细解释一下每个流程。

初始化对账任务

对账系统依靠对账任务记录推动流转，目前每天凌晨将会初始化生成对账任务记录，后续任务流转就可以从这里开始。

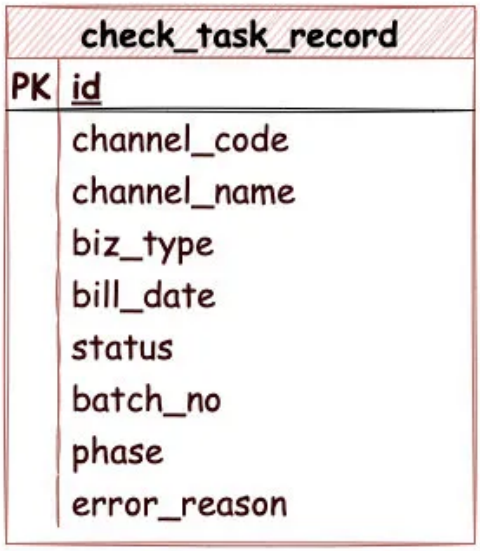
对账系统维护一张对账核对规则表：

对账核对规则表关键字段含义如下：

- channel_code 渠道编码，每个支付渠道将会分配一个唯一渠道编码，例如微信，支付宝
- biz_type 业务类型，例如支付，退款，提现等
- status 是否生效

每次对接新的支付渠道，对账配置规则需要新增核对规则。

初始化对账定时任务将会查找核对规则表中所有的生效的配置规则，依次生成当天的对账任务记录：



对账任务记录部分字段与核对规则表含义一样，不再赘述，其他字段含义如下：

- bill_date 账期，一般 D 日对账任务核对 D-1 数据，所以账期为 D-1 日
- batch_no 对账批次，生成规则如下：账期+渠道编码+ 001
- phase,当前对账任务处于阶段，根据上面对账流程可以分为
  - 初始化
  - 数据收集

  - 存疑处理
  - 数据核对
  - 二次存疑处理
  - 数据汇总
  - 差错数据推送
- error_reason 错误原因

初始化对账任务结束之后，对账任务流程推动到第二阶段，数据收集。

数据收集

数据收集阶段，收集两端待核对的数据，为后面的数据核对任务提供核对数据。

数据收集阶段分为两部分：

- 本端数据收集，即自己方产生的支付数据
- 对端数据收集，即三方渠道产生支付数据

本端数据收集

本端数据，是自己业务产生的支付数据，这些数据原本存在各个业务的数据库中。

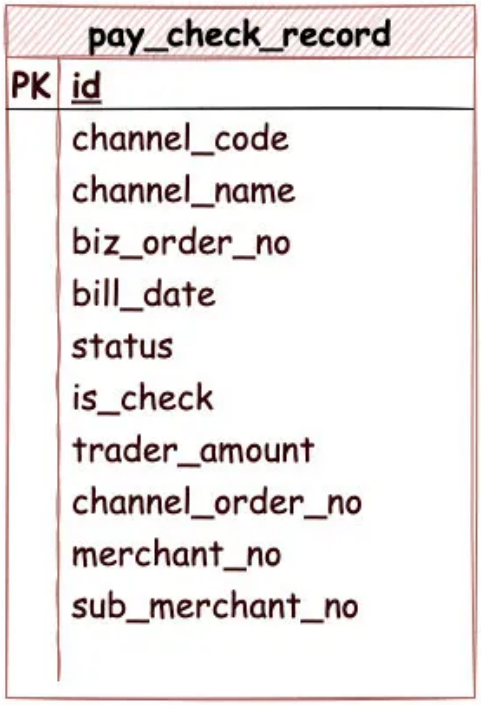
对账系统获取这些支付数据，一般有两种方式：

- 查询，对账系统主动拉取
- 推送，对账系统监听获取数据

查询数据方式上篇文章也聊到过，数据量小的情况下，没什么问题。一旦数据量变大，查询效率就会变低。

所以这里我们采用推送的方式，对账系统监听各个业务数据表 binlog，每当业务数据发生变动，对账系统就可以接受到 binlog 消息。

对账系统接受到 binlog 消息，将会判断当前消息是否需要过滤，是否已经支付成功等等，满足条件之后，binlog 消息将会插入本端数据表中，表结构如下：



本端记录表关键字段含义如下：

- channel_code 渠道编码，每个支付渠道将会分配一个唯一渠道编码，例如微信，支付宝
- biz_order_no 本端支付流水号
- bill_date 账期
- status 状态
- is_check 对账状态，0-未核对，1-已核对
- trade_amount 支付金额
- channel_order_no 三方渠道支付单号
- merchant_no 商户号
- sub_merchant_no 子商户号

上面展示的支付记录表结构，根据业务类型不同，本端其实还有退款记录表，提现记录表等。

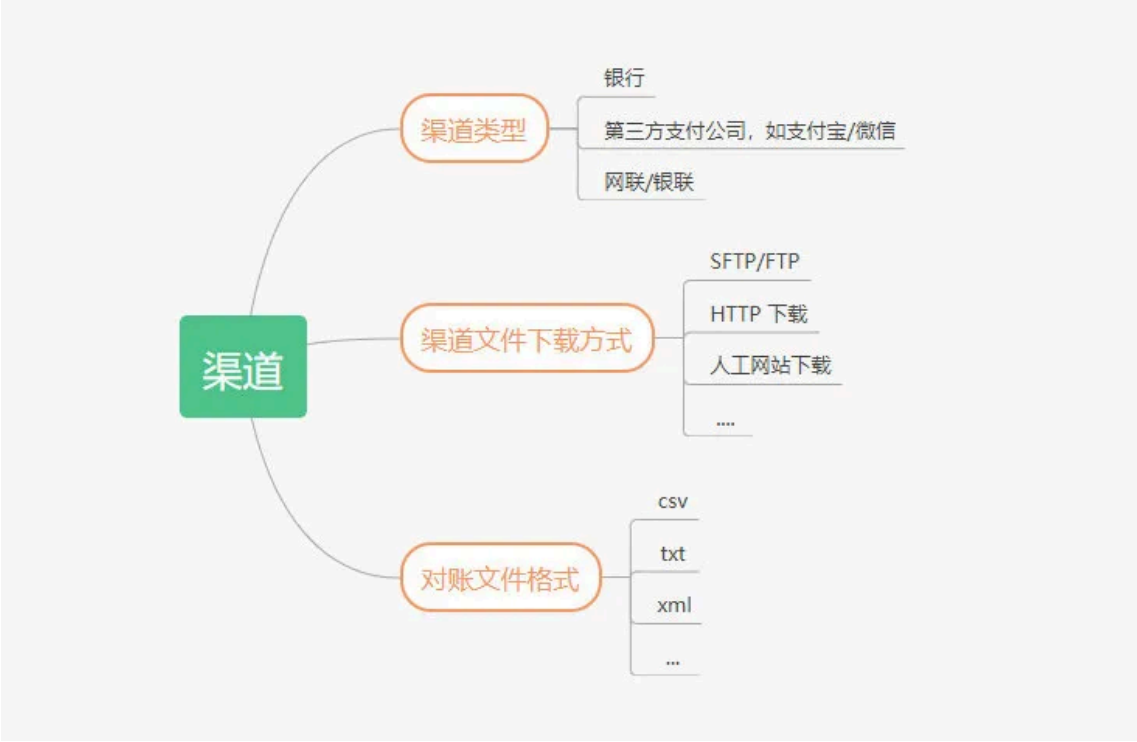
这里设计的时候，实际上也可以将所有业务数据放在一张表中，然后根据业务类型字段区分。

对端数据收集

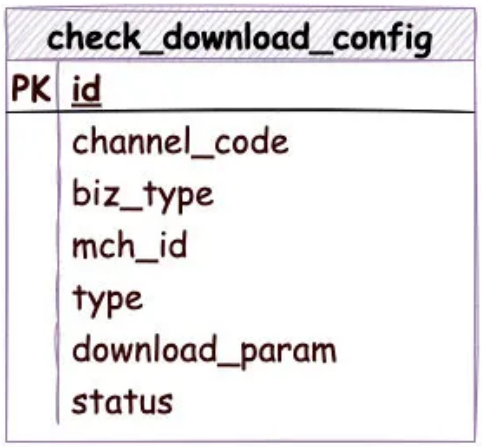
对端数据，就是第三方支付渠道产生支付数据，一般 D 日产生交易之后，D+1 日第三方渠道将会生成一个对账文件。

对账系统需要从对端提供的对账文件获取对端数据。

渠道的对账文件，下载方式，文件类型存在很大的差异，每次接入新的支付渠道，这里需要经过新的开发。



对端数据这里维护了一张渠道下载配置表，对端数据收集的时候将会获取所有可用配置：

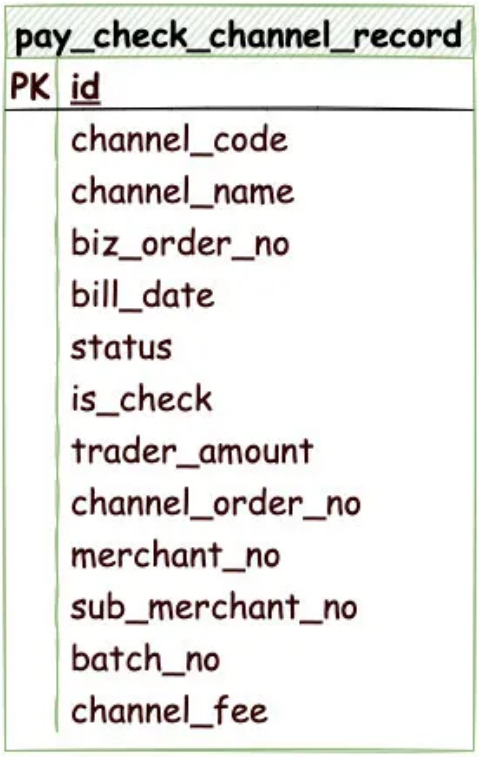


渠道下载配置表关键字段含义如下：

- mch_id 三方渠道分配的商户号
- type 下载类型
  - FTP
  - SFTP
  - HTTP
- download_param 下载的配置参数，比如 ftp 的地址，登录密码，下载地址等。

对账文件下载成功之后，需要根据文件类型进行解析，最后转化自己的需要的对账数据入库。

对端数据表结构如下：



上面关键字段与本端记录表类似，额外新增字段:

- channel_fee 渠道手续费，用于统计渠道收的手续费

同样渠道记录表根据根据业务类型也分为退款渠道记录表，提现渠道记录表等，同样也可以合并成一张表，根据业务类型区分。

对端数据收集阶段，由于拉取三方渠道的对账文件，那有时候渠道端存在异常，将会导致对账文件下载延迟，从而导致其他任务也出现的相应的延迟。

这一点是整个对账流程中，相对不可控的问题。我们需要在对账流程设计中考虑这一点。

对账文件下载解析成功入库之后，对账流程将会流转到下一个流程存疑数据处理。

存疑数据处理

讲解这个流程之前，先给大家解释一下什么是存疑数据？

正常支付过程中，会存在一个两边账期不一致的问题，比如说本端数据支付时间是 2021 年 12 月 28 日 23 点 59 分 59 秒，那么本端认为这笔支付交易账期是 2021 年 12 月 28 日。

然而这笔支付发送给三方渠道之后，三方渠道支付成功的时间已经是 2021 年 12 月 29 日 0 点 0 分 2 秒，三方渠道支付账期记为2021 年 12 月 29 日。

这种情况下我们这边记录账期是2021 年 12 月 28 日，但是第三方渠道这笔记录是 2021 年 12 月 29 日，所以 2021 年 12 月 28 日对账单上没有这笔支付记录，这就导致一笔差异数据（一端有/一端无）的情况。

上面这种情况就是典型因为日切问题导致差异。

但是我们知道 2021 年 12 月 29 日对账单上肯定会包含这笔，所以我们可以先把这笔差异数据挂起，当做存疑数据，等到2021 年 12 月 29 日账期对账的时候，对方账单包含这笔，当天就能核对成功，这就解决这笔差异数据。

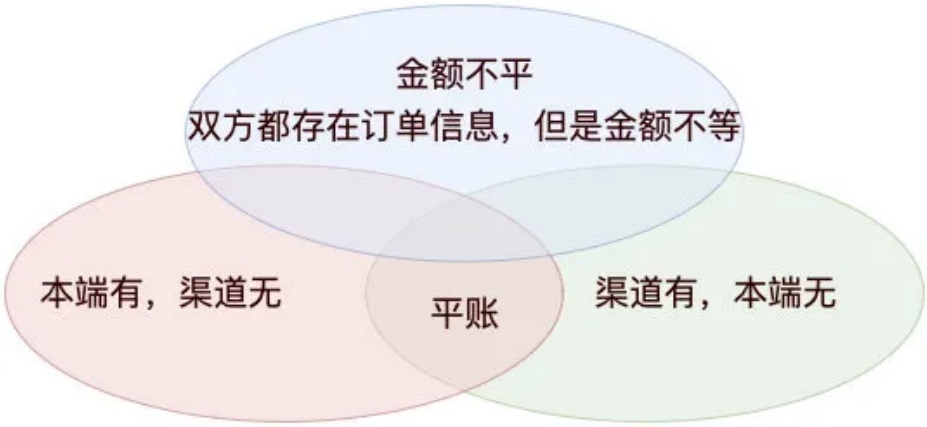
所以说存疑数据，就跟其字面意思一样，当这笔数据当前处理不了的时候，那就现放着，不做定论，过一天我再尝试处理一下。

除了上面日切问题导致的差异数据以外，还有一些情况：

- 网络问题，导致两边订单状态不一致。
- 测试环境与生产环境共用一个三方渠道商户号，测试环境产生的交易出现在对账单里

存疑数据分为三种类型：

- 本端有，渠道无，即本端存在订单信息，渠道账单记录没有订单信息，可能是日切导致的问题
- 渠道有，本端无，即本端不存在订单信息，渠道端账单记录却有订单信息，可能是测试环境与生产环境共用渠道参数
- 金额不平，即双方都存在订单信息，但是双方订单金额不一致



了解完存疑数据的定义，我们再来看下存疑数据处理的流程。

存疑数据将会由下面的流程中产生，这里先来看下存疑表结构：

关键字段如下：

- batch_no 批次号
- biz_id 业务单号
- biz_amount 金额
- status 0-未处理，1-已处理
- biz_date 账期
- biz_type 业务类型
- channel_code 渠道类型
- delayed_times 延迟天数
- merchant_no 商户号
- sub_merchant_no 子商户号
- buffer_type 存疑类型，0-本端存疑，1-渠道存疑

存疑处理过程将会捞起所有存疑表中还未处理的存疑数据，根据存疑类型反向查找对账数据表。例如：

- 渠道存疑（第一天对账，本端有，渠道无），查找对端数据
- 本端存疑（第一天对账，本端无，渠道有），查找本端数据

查找对端/本端数据，都是根据支付流水号加业务类型查找定位。

如果在本端/对端数据中找到，这里还需要再对比一下金额：

- 如果金额不相等，代表单号相同，但是金额不等，将这笔移动到支付差异表
- 如果金额相等，代表这两笔核平，存疑表将这笔数据更新为核对成功，本端/对端数据更新为对账成功

上面这一步比较重要，因为下面对账核对过程主要核对要素是支付流水号+支付金额，通过这种方式收集单片账是无法知道是因为单号不存在，还是因为金额不存在原因，具体流程可以看下下面核对流程。

如果在本端/对端数据还是找不到,那就根据渠道配置的存疑规则，如果当前已经存疑的天数大于配置渠道存疑天数，则将数据直接移动到差错表。

如果存疑天数小于当前渠道配置天数，那就不要管，继续保存在存疑表，等待下一天存疑数据处理。

一般来说，日切导致的数据，存疑一天，就可以解决。但是有些渠道可能是 T+1 在对账，这种情况需要配置的存疑天数就要长一点了。

本地存疑数据处理结束之后，下面就要开始 DP 数据处理。

总结

上篇文章主要聊了对账流程前半部分，这几个流程主要是为了后续 DP 平台核对收集业务数据。

这里存疑流程处理比较关键，不熟悉对账流程的同学，这里需要重点关注下。

下篇文章主要讲下 DP 平台对账数据如何核对，敬请期待。

#对账系统 4　系统架构设计 5　支付 3

#对账系统 · 目录

上一篇
支付对账系统序章：千万级数据对账怎么这么难？

下一篇
千万级支付对账系统怎么玩（下篇）？