第三章 数据同步

数据同步技术含义：**不同系统间的数据流转**，有多种不同的应用场景。

应用场景：

- 同类型不同集群数据库之间的数据同步
- 主数据库与备份数据库之间的数据备份
- 主系统与子系统之间的数据更新
- 不同地域、不同数据库类型之间的数据传输交换

大数据系统中的数据同步

- **数据从业务系统同步进入数据仓库**
- **数据从数据仓库同步进入数据服务或数据应用**

一、数据同步基础

源业务系统的数据类型：

- 关系型数据库的结构化数据：MySQL、Oracle等，数据存储在数据库表中
- 非关系型数据库的非结构化数据：HBase、MongoDB等，数据存储在数据库表中
- 文件系统的结构化或非结构化据：阿里云对象存储OSS、文件存储NAS等，数据以文件形式进行存储

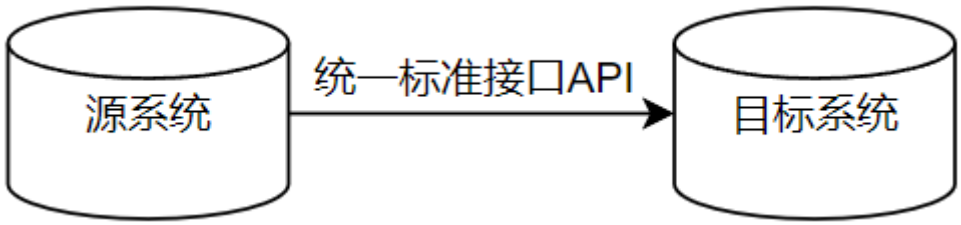
数据同步需要针对不同的数据类型及业务场景选择不同的同步方式。

同步方式可以分为三种：

- 直连同步
- 数据文件同步
- 数据库日志解析同步

1.1 直连同步

通过定义好的规范接口API直连业务数据库，如JDBC规定了统一规范的标准接口。适合操作型业务系统的数据同步。

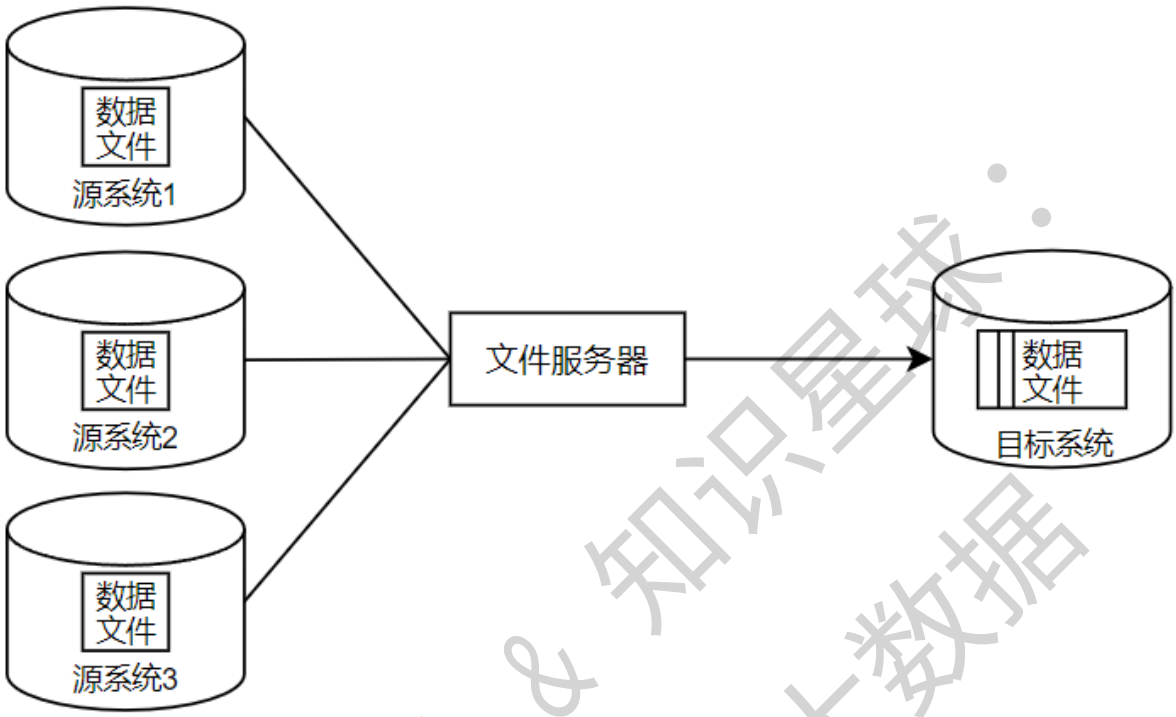


直连同步优缺点：

- 优点：配置简单，容易实现。
- 缺点：对源系统的性能影响较大，大批量数据同步时会降低甚至拖垮业务系统的性能，不适合从业务系统到数据仓库系统的同步。

1.2 数据文件同步

通过约定好的文件编码、大小、格式等，直接将源系统的生成数据的文本文件通过专门的文件服务器（如FTP服务器）传输到目标系统，然后加载到目标数据库系统中。



优点：当数据源包含多个异构的数据库系统时，用这种方式比较简单、实用。另外，日志类数据通常是以文本文件形式存在的，也适合使用数据文件同步方式。

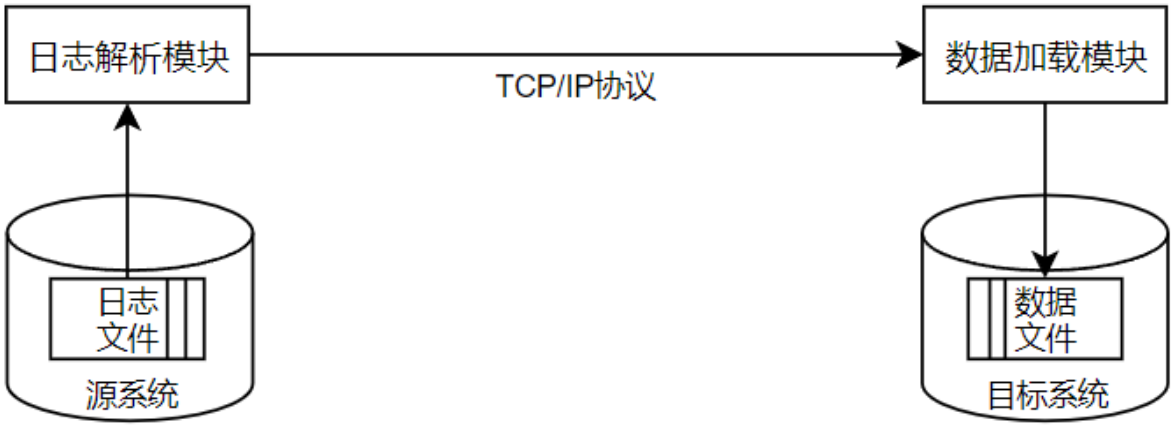
注意：

- 通过文件服务器上传、下载可能会造成丢包或错误，为了确保数据文件同步的完整性，需要有一个效验文件，用于验证数据同步的准确性。
- 在数据文件从源系统中生成时，可以增加压缩和加密功能，提高文件的传输效率和安全性。

1.3 数据库日志解析同步

大多数主流数据库都已经实现了使用日志文件进行系统恢复，因为日志文件信息足够丰富，而且数据格式也很稳定，完全可以通过解析日志文件获取发生变更的数据，从而满足增量数据同步的需求。

数据库中日志文件信息丰富、数据格式稳定，保存了数据记录的变更（增、删、改），当出现故障时，可以通过日志文件进行系统恢复。所以可以通过读取数据库的日志文件，收集变化的数据信息，并判断日志中的变更是否属于被收集的对象，将其解析到目标数据文件中。



优点：数据库日志读取操作是在操作系统层面完成，不需要通过数据库，因此不会给源系统带来性能影响。

缺点：

- 数据延迟。例如，业务系统做批量补录可能会使数据更新量超出系统处理峰值，导致数据延迟。
- 投人较大。需要在源数据库与目标数据库之间部署一个系统实时抽取数据。
- 数据漂移和遗漏。通常是指增量表的同一个业务日期数据中包含前一天或后一天凌晨附近的数据或者丢失当天的变更数据。

二、阿里数据仓库的同步方式

数据仓库的特性之一是**集成**，**将不同的数据来源、不同形式的数据整合在一起**。

阿里数仓数据同步特点：

- 数据来源多样化。除了结构化数据，还有大量非结构化数据
- 数据量非常大。阿里（书上提到的）每天需要同步的数据量达到PB级别

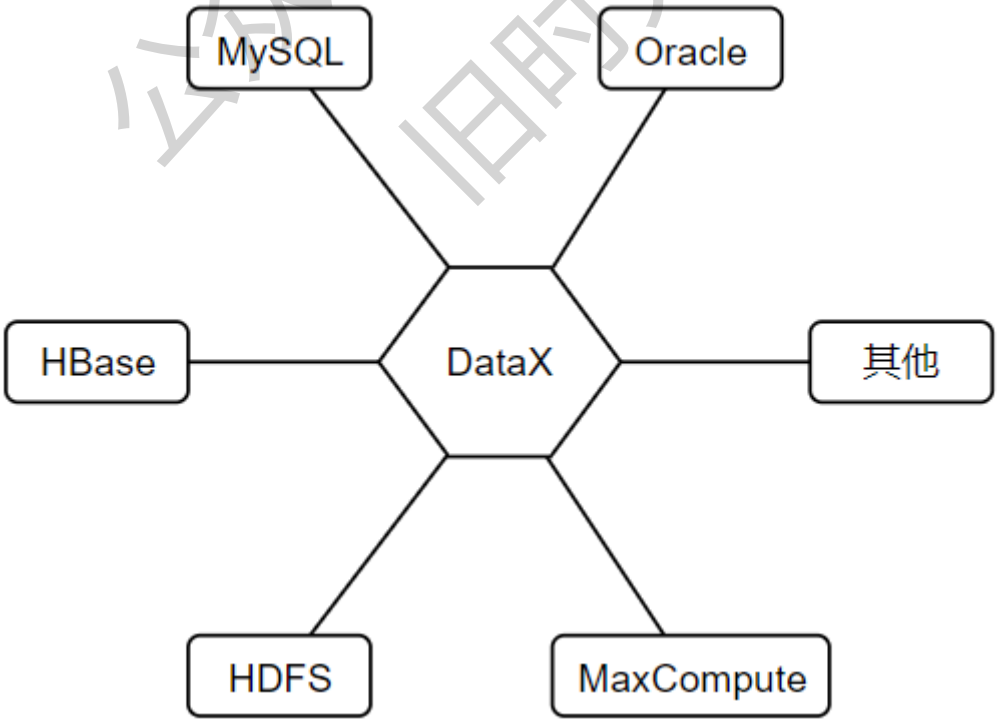
2.1 批量数据同步

对于离线类型的数据仓库应用，需要将不同的数据源批量同步到数据仓库，以及将经过数据仓库处理的结果数据定时同步到业务系统。

市场上数据库种类繁多，不同数据库的数据类型都略有不同，而数据仓库是集成各类数据源的地方，所以数据类型必须是统一的。

阿里通过DataX将各类源数据库系统的数据类型统一转换为字符串类型的方式，实现数据格式的统一。

对于不同的数据源，DataX通过插件的形式提供支持，开发者可以在极短的时间内开发一个插件以快速支持新的数据库或文件系统，将数据从数据源读出并转换为中间状态，同时维护好数据的传输、缓存等工作。

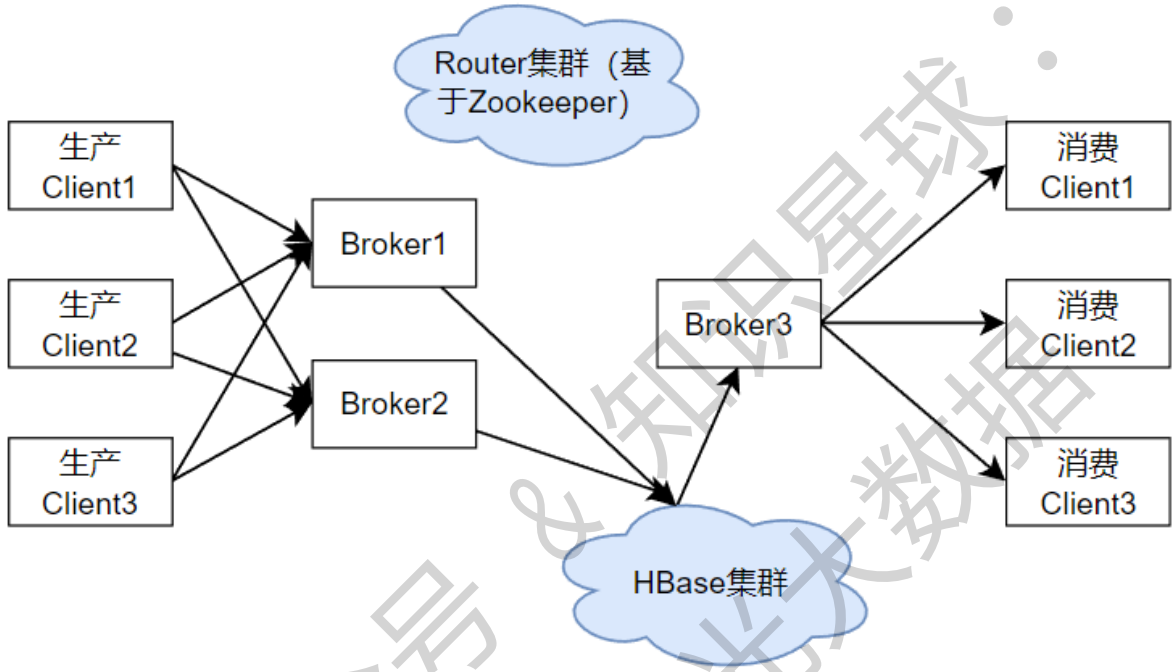


2.2 实时数据同步

日志类数据都是实时产生的，所以需要尽快将日志以数据流的方式不间断地同步到数据仓库。或者对业务系统产生的数据进行实时处理，比如天猫“双11”的数据大屏，对所产生的交易数据需要实时汇总，实现秒级的数据刷新。

这类数据是通过解析MySQL的binlog日志来实时获得增量的数据更新，并通过消息订阅模式来实现数据的实时同步。具体来说，就是建立一个体制数据交换中心，通过专门的模块从每台服务器源源不断地读取日志数据，或者解析业务数据库系统的binlog或归档日志，将增量数据以数据流的方式不断同步到日志交换中心，然后通知所有订阅了这些数据的数据仓库系统来获取。

阿里是通过TimeTunnel（TT）消息中间件（具有高性能、实时性、顺序性、高可靠性、高可用性、可扩展性等特点）来实现实时数据同步的。TT是一种基于生产者、消费者和Topic消息标识的消息中间件，将消息数据持久化到 HBase 的高可用、分布式数据交互系统。



三、数据同步遇到的问题与解决方案

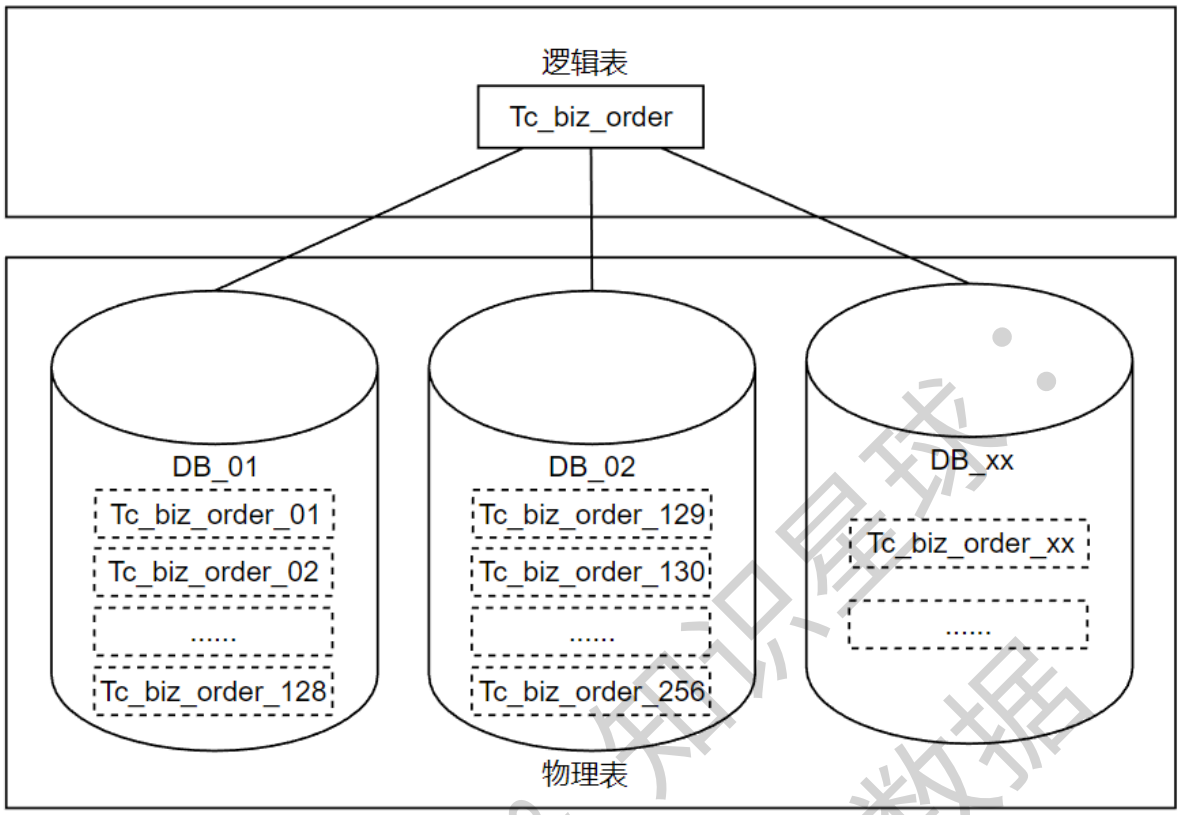
3.1 分库分表的处理

问题：业务系统需处理的数据量飞速增长，为了让系统具备灵活的扩展能力和高并发大数据量的处理能力，一些主流数据库采用了分布式分库分表方案来解决，但是这种设计也加大了同步处理的复杂度。

解决方案：通过建立中间状态的逻辑表来整合统一分库分表的访问。

阿里的TTDL（Taobao Distributed Data ayer）就是这样一个分布式数据库的访问引擎。

TDDL是在持久层框架之下、JDBC驱动之上的中间件，它与JDBC规范保持一致，有效解决了分库分表的规则引擎问题，实现了SQL析、规则计算、表名替换、选择执行单元并合并结果集的功能，同时解决了数据库表的读写分离、高性能主备切换的问题，实现了数据库配置信息的统一管理。



3.2 高效同步和批量同步

问题：

- 工作量大，相似且重复的操作会降低开发人员的工作热情
- 与数据需求方的沟通和流程成本加大

解决方案：

- 配置透明化，自动生成配置信息
- 简化操作步骤，建表、配置任务、发布、测试操作一键化处理
- 降低了数据同步的技能门槛，方便数据需求方获取和使用数据

阿里通过OneClick实现数据的一键化和批量化同步。一键完成DDL和DML生成、数据的冒烟测试以及在生产环境中测试等，大大降低了数据同步成本。

3.3 增量与全量同步的合并

问题：表的业务数据库越来越大，按周期全量同步的方式会影响处理效率，甚至不可能做到。

解决方案：每次只同步新变更的增量数据，然后与上一个同步周期获得的全量数据进行合并，从而获得最新版本的全量数据。简而言之，就是**增量同步**，然后**与原数据合并**。

当前流行的大数据平台基本都不支持 update 操作，现在比较推荐的方式是全外连接（full outer join）+ 数据全量覆盖重新加载（insert overwrite），例如日调度，则将当天的增量数据和前一天的全量数据做全外连接，重新加载最新的全量数据。

3.4 同步性能的处理

问题：

- 数据同步任务的总线程数达不到用户设置的首轮同步的线程数，可能影响数据同步效率
- 用户不清楚该如何设置首轮同步的线程数，可能导致CPU资源无法合理使用
- 同步控制器平等对待接收到的同步线程，导致重要的同步线程因得不到CPU资源而无法同步

上述问题总结下来，就是**数据同步任务运行不稳定**。

解决方案：

基于负载均衡思想的数据同步方案。通过目标数据库的元数据估算同步任务的总线程数，以及通过系统预先定义的期望同步速度估算首轮同步的线程数，同时通过数据同步任务的业务优先级决定同步线程的优先级，最终提升同步任务的执行效率和稳定性。

3.5 数据偏移的处理

问题：ODS层表的同一个业务日期数据中包含前一天或后一天凌晨附近的数据，或者丢失当天的变更数据。

ODS表按时间段来切分进行分区存储，通常的做法是按某些时间戳字段来切分，而实际上往往由于时间戳字段的准确性问题导致发生数据漂移。

通常，时间戳字段分为四类：

- 数据库表中标识数据记录更新时间：modified_time
- 数据库日志中标识数据记录更新时间：log_time
- 数据库表中记录业务发生时间：proc_time
- 标识数据记录被抽取到的时间：extract_time

理论上，上述几个时间应该是一致的，但实际生产中，会出现差异，可能的原因如下：

- 由于数据抽取需要时间，extract_time往往会晚于前三个时间
- 前台业务系统手工订正数据时未更新modified_time
- 由于网络或者系统压力问题，log_time或者modified_time会晚于proc_time

下列的一些做法，可能会导致数据漂移：

根据extract_time来获取数据，数据漂移问题最明显

根据modified_time限制，不更新modified_time导致数据一楼，或者凌晨时间产生的数据记录漂移到后一天

根据log_time限制，由于网络或者系统压力问题，会晚于proc_time，导致凌晨时间产生的数据记录漂移到后一天。

解决方案：

- 多获取后一天的数据，保障数据只多不少
    - 做法：具体的数据切分让下游根据自身不同的业务场景用不同的业务时间proc_time来限制
    - 缺点：存在一些数据误差。例如一个订单是当天支付的，但是第二天凌晨申请退款关闭了该订单，那么这条记录的订单状态会被更新，下游在统计支付订单状态时会出现错误。
- 通过多个时间戳字段限制时间，获取相对准确的数据
    - 做法：
        - 根据log_time分别冗余前一天最后15分钟的数据和后一天凌晨开始15分钟的数据，并用modified_time过滤非当天数据
        - 根据log_time获取后一天15分钟的数据，按照主键根据log_time做升序排列去重，根据主键去重获取最后状态变化的数据
        - 将前两步的结果数据做全外连接，通过限制业务时间proc_time来获取所需数据。

暮然 送你一张星球优惠券

「旧时光大数据」

立减

¥ 40

新人立减券

2023/12/31 12:00 后失效

知识星球

长按扫码领取优惠