万字总结阿里大数据之路-数据技术篇（建议收藏）

大家好，我是唐三少，今天给大家带来此篇欢迎大家品读。给自己一个大嘴巴子，又话多了，下面开始上号。

目录

一、日志采集

本章主要介绍数据采集中的日志采集部分，阿里巴巴的日志采集体系方案包括两大体系：

- Aplus.JS是Web端（ 基于浏览器）日志采集技术方案；
- UserTrack是APP端（无线客户端）日志采集技术方案。

本章从浏览器的页面日志采集、无线客户端的日志采集以及我们遇到的日志采集挑战三块内容来阐述阿里巴巴的日志采集经验。

1.1 浏览器的页面日志采集

（1）页面浏览（展现）日志采集

顾名思义，页面浏览日志是指当一个页面被浏览器加载呈现时采集的日志。此类日志是最基础的互联网日志，也是目前所有互联网产品的两大基本指标：页面浏览量（Page View，PV）和访客数（UniqueVisitors，UV）的统计基础。页面浏览日志是目前成熟度和完备度最高，同时也是最具挑战性的日志采集任务，我们将重点讲述此类日志的采集。

（2）页面交互日志采集

当页面加载和渲染完成之后，用户可以在页面上执行各类操作。随着互联网前端技术的不断发展，用户可在浏览器内与网页进行的互动已经丰富到只有想不到没有做不到的程度，互动设计都要求采集用户的互动行为数据，以便通过量化获知用户的兴趣点或者体验优化点。交互日志采集就是为此类业务场景而生的。

除此之外，还有一些专门针对某些特定统计场合的日志采集需求，如专门采集特定媒体在页面被曝光状态的曝光日志、用户在线状态的实时监测等，但在基本原理上都脱胎于上述两大类。

1.2 无线客户端的日志采集

众所周知，日志采集多是为了进行后续的数据分析。

移动端的数据采集，一是为了服务于开发者，协助开发者分析各类设备信息；二是为了帮助各APP更好地了解自己的用户，了解用户在APP上的各类行为，帮助各应用不断进行优化，提升用户体验。

无线客户端的日志采集采用采集SDK来完成，在阿里巴巴内部，多使用名为UserTrack的SDK来进行无线客户端的日志采集。无线客户端的日志采集和浏览器的日志采集方式有所不同，移动端的日志采集根据不同的用户行为分成不同的事件，“事件”为无线客户端日志行为的最小单位。基于常规的分析，UserTrack（UT）把事件分成了几类，常用的包括页面事件（同前述的页面浏览）和控件点击事件（同前述的页面交互）等。

对事件进行分类的原因，除了不同事件的日志触发时机、日志内容和实现方式有差异之外，另一方面是为了更好地完成数据分析。在常见的业务分析中，往往较多地涉及某类事件，而非全部事件；故为了降低后续处理的复杂性，对事件进行分类尤为重要。要更好地进行日志数据分析，涉及很多方面的内容，如需要处理Hybrid应用，实现H5和Native日志的统一；又如识别设备，保证同一设备上各应用获取到的设备信息是唯一的。除此之外，对于采集到的数据如何上传，以及后续又如何合理处理等，每个过程都值得我们进行深入的研究和探索。

1.3 日志采集的挑战案例

对于目前的互联网行业而言，互联网日志早已跨越初级的饥饿阶段（大型互联网企业的日均日志收集量均以亿为单位计量），反而面临海量日志的淹没风险。各类采集方案提供者所面临的主要挑战已不是日志采集技术本身，而是如何实现日志数据的结构化和规范化组织，实现更为高效的下游统计计算，提供符合业务特性的数据展现，以及为算法提供更便捷、灵活的支持等方面。

这里介绍两个最典型的场景和阿里巴巴所采用的解决方案。

- 日志分流与定制处理
- 采集与计算一体化设计

二、数据同步

2.1 数据同步基础

数据同步的三种方式：

- 数据直抽。
- 数据文件同步。
- 数据库日志解析同步。

2.2 数据同步策略

2.2.1 批量数据同步

- 数据类型统一采用字符串类型（中间状态）。
- DataX对不同的数据源提供插件，将数据从数据源读出并转换为中间状态存储。
- 传输过程全内存操作，不读写磁盘，也没有进程间通信。

2.2.2 实时数据同步

- 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。
- 日志数据——> 日志交换中心——> 订阅了该数据的数据仓库。
- 针对订阅功能，可以支持主动、被动订阅，订阅端自动负载均衡。数据消费者自己把握消费策略。可以订阅历史数据，随意设置订阅位置。并具有属性过滤功能。

2.3 数据同步问题

2.3.1 分库分表处理

建立了一个中间层的逻辑表来整合分库分表。使得外部访问中间层的时候，与访问单库单表一样简洁。中间层介于应用持久层和JDBC驱动之间。

2.3.2 高效同步和批量同步

统一管理不同源数据库的元数据信息，强化版的元数据管理平台，要求数据同步配置透明化。通过库名和表名即可通过元数据管理平台唯一定位，再由自动化的数据同步平台完成建表、配置任务、发布、测试的一键化处理。

2.3.3 增量与全量同步的合并

全外连接与insert overwrite代替merge与update。

采用分区，每天保持一个最新的全量版本，每个版本仅保留较短的时间周期如3天至一周。

方式为当天的增量数据与前一天的全量数据合并，生成当天的全量数据。

2.3.4 数据同步性能

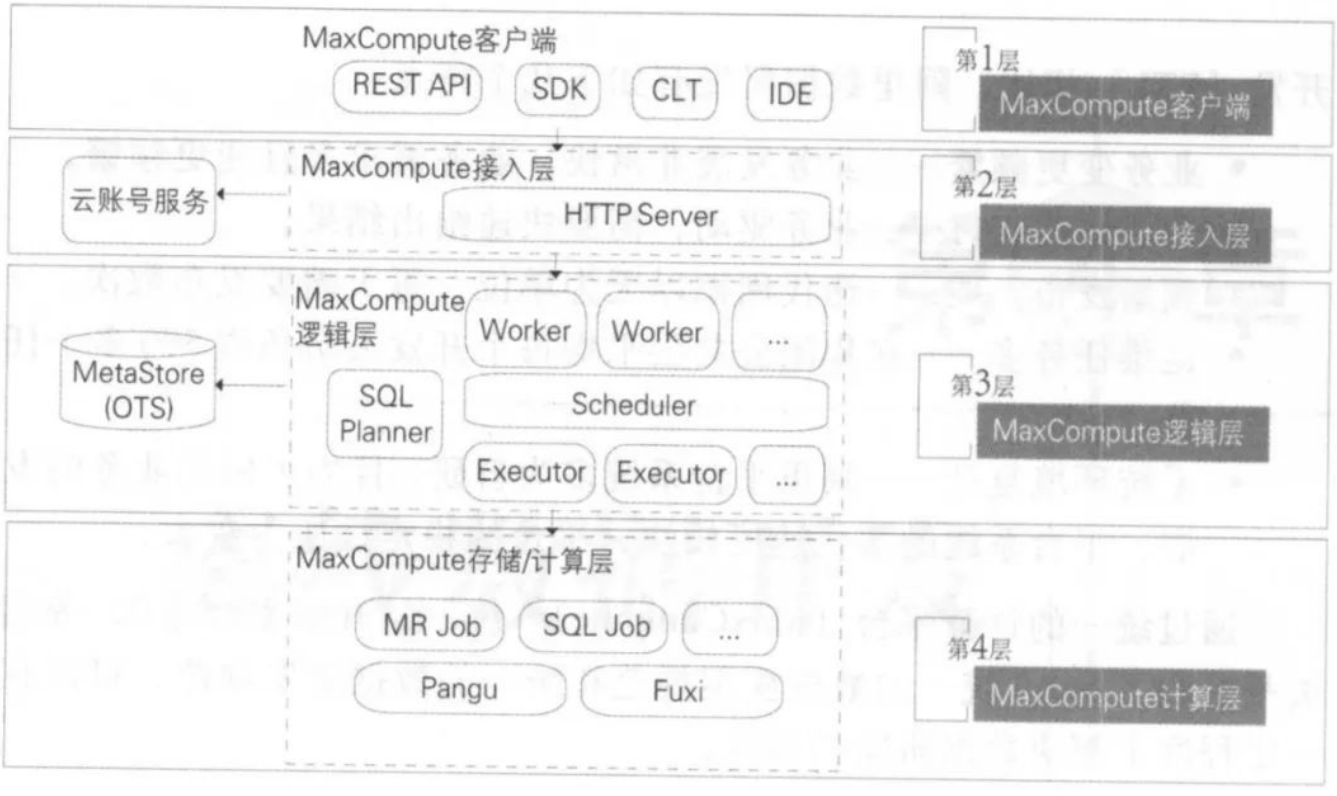
2.3.5 数据漂移

常见于0点时分左右，数据按照日期划分跨天的问题。冗余获取0点左右的数据，根据多种时间字段来排序去重，重新划分和补录数据。

三、离线数据开发

3.1 统一计算平台

- 承担数据导入、存储、各式计算。
- 从上至下分为客户端、接入层、逻辑层、计算层。



MaxCompute 体系架构图

（1）在逻辑层有 Worker Sc heduler Executor 三个角色：

- Worker 处理所有的阻STful 请求，包括用户空间（ Project ）管理操作、资源（ Resource ）管理操作、作业管理等，对于 SQLDMLMR 等需要启动 MapReduce 的作业，会生成 MaxCompute Instance（类似于 Hive 中的 Job），提交给 Scheduler 一步处理。
- Scheduler负责 MaxCompute Instance 的调度和拆解，并向计算层的计算集群询问

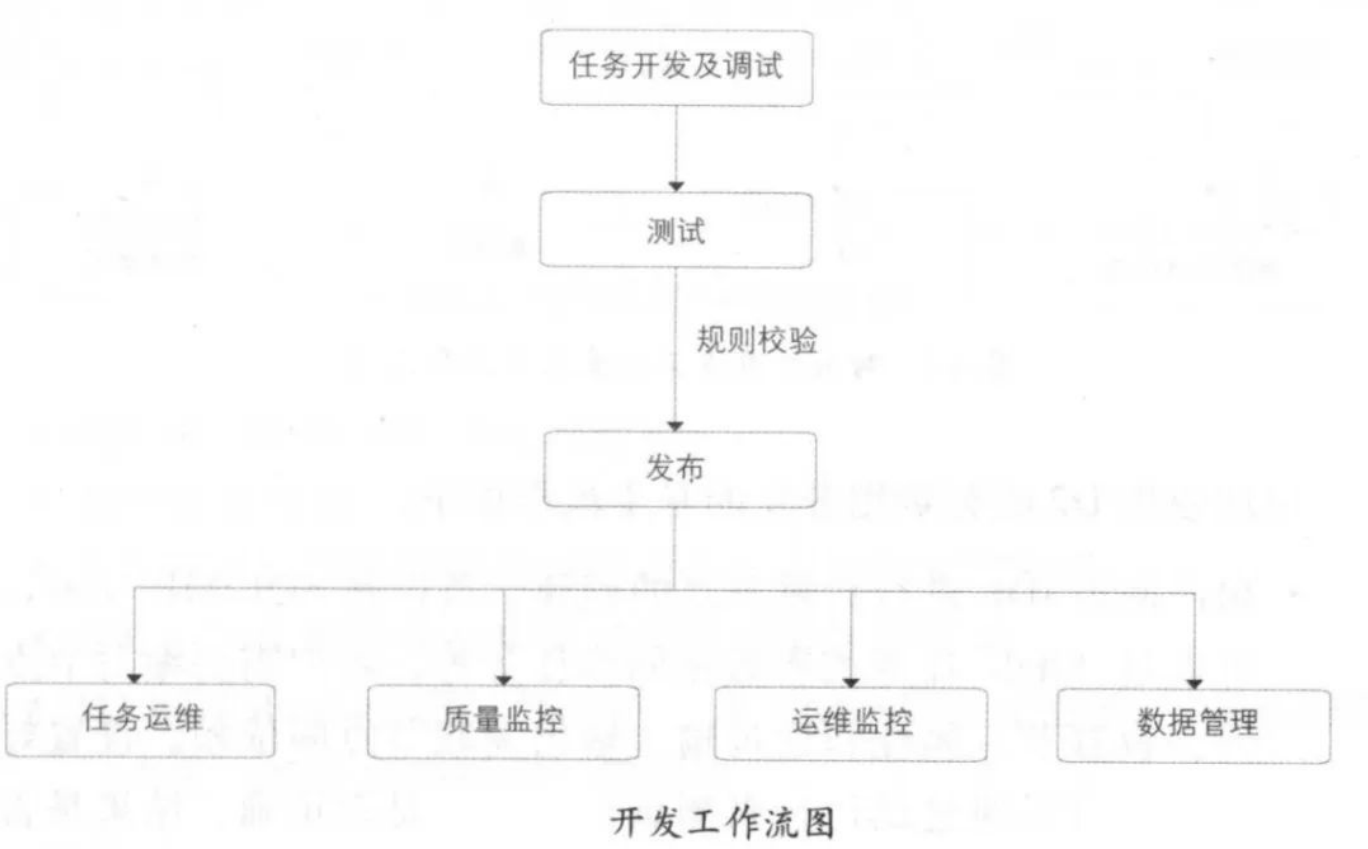
资源占用情况以进行流控。

- Executor 负责 MaxCompute Instance 的执行，向计算层的计算集群提交真正的计算任务。

（2）MaxCompute 能保证数据的正确性，如对数据的准确性要求非常高的蚂蚁金服小额贷款业务，就运行于Max Compute 平台之上。

3.2 统一开发平台

包括开发调试与发布平台、代码质量控制平台、数据质量监控平台、测试平台。



开发工作流图

两张图对应起来看：

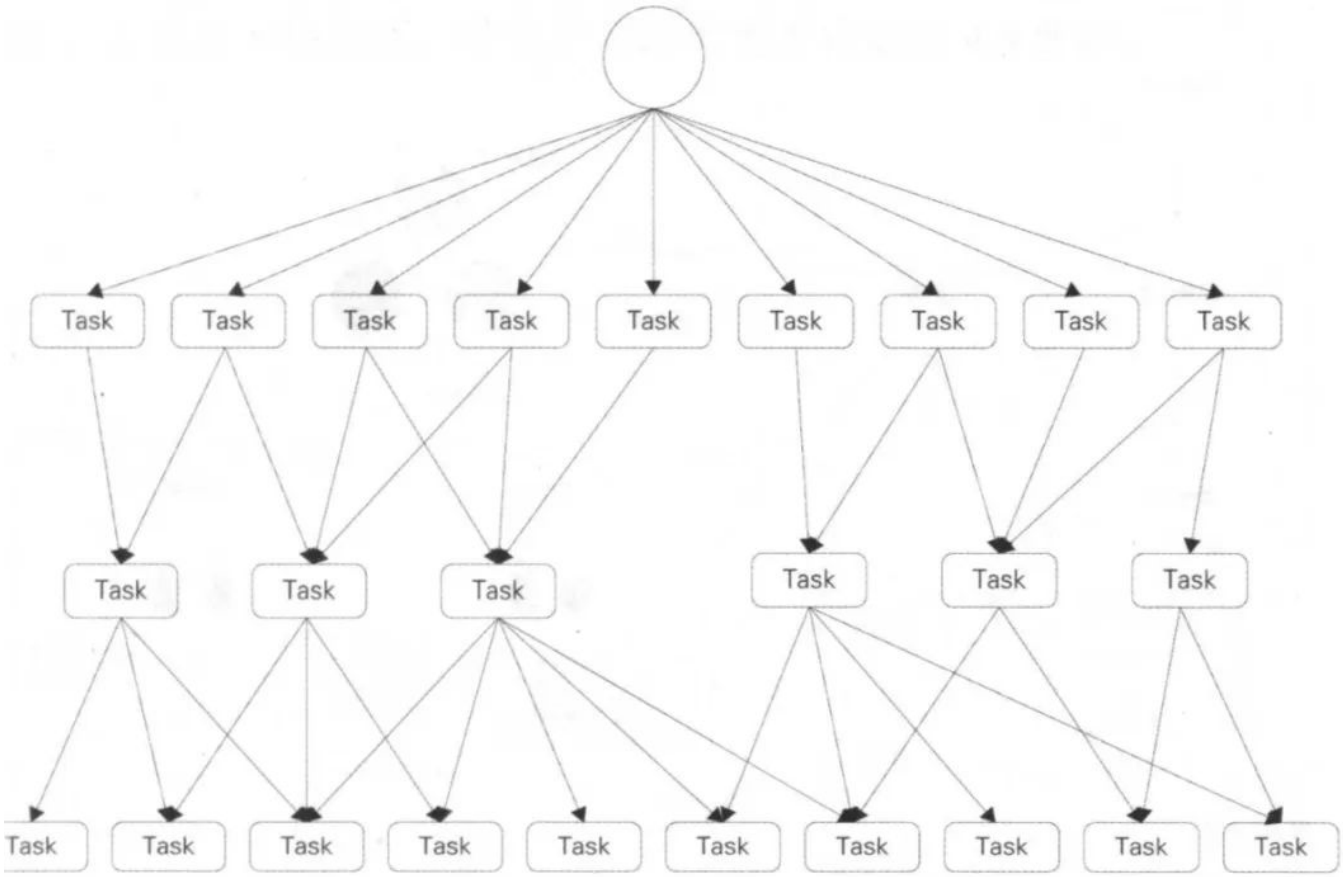
(1) 在彼岸：多路分支进行测试和完成数据脱敏(将敏感数据模糊化)

(2) SQLSCAN：对用户的SQL进行规范，检查代码的规范性

(3) 开发平台(D2)发布系统：实现和用户的IDE对接，用户可以通过IDE在D2上创建工作节点。

(4) DQC：清洗和监控数据，接收到到的数据与规则库对比，监控相关数据的可用性和对无用的数据进行清洗。

3.3 任务调度系统



调度任务关系依赖图

- 调度系统分为调度引擎和执行引擎。
- 状态机分为工作流状态机与任务状态机，工作流包含待提交、已创建、正在执行、成功、失败等各个工作节点；而任务状态则是在工作流之下的一系列状态，例如执行中的等待状态。
- 通过事件驱动，生成调度实例，在两种状态机之间切换执行调度，根据状态的不同也在调度引擎和执行引擎之间切换。

3.4 特点

- 依赖管理。自动识别SQL的输入输出表，自动关联依赖的任务。
- 周期调度。会根据资源和上游依赖的情况，自动调整具体执行时间。
- 手动运行。基于自动发布，可以在开发平台中开发脚本，发布到生产后手工调度。

四、实时技术

4.1 流式技术架构

架构分为数据采集、数据处理、数据存储、数据服务四部分。

4.1.1 数据采集

- 从数据源采集数据均已文件的形式保存，通过监控文件内容的变化，使用数据大小的限制和间隔时间阈值的限制来共同决定采集的频率。
- 将数据落到数据中间件之后，可由流计算平台来订阅数据。

4.1.2 数据处理

- SQL语义的流式数据分析能力。
- 流式处理的原理：多个数据入口、多个处理逻辑，处理逻辑可分为多个层级逐层执行。
- 数据倾斜：数据量非常大时，分桶执行。
- 去重处理：精确去重使用数据倾斜的方式，模糊去重使用哈希来减少内存占用。
- 事物处理：超时补发、每批数据自带ID、将内存数据备份到外部存储。

4.1.3 数据存储

- 实时系统要求数据存储满足多线程多并发以及毫秒级的低延时。
- 表名设计和rowkey设计遵循数据均衡分布、同一主维度的数据在同一张物理表。

4.2 流式数据模型

4.2.1 数据分层

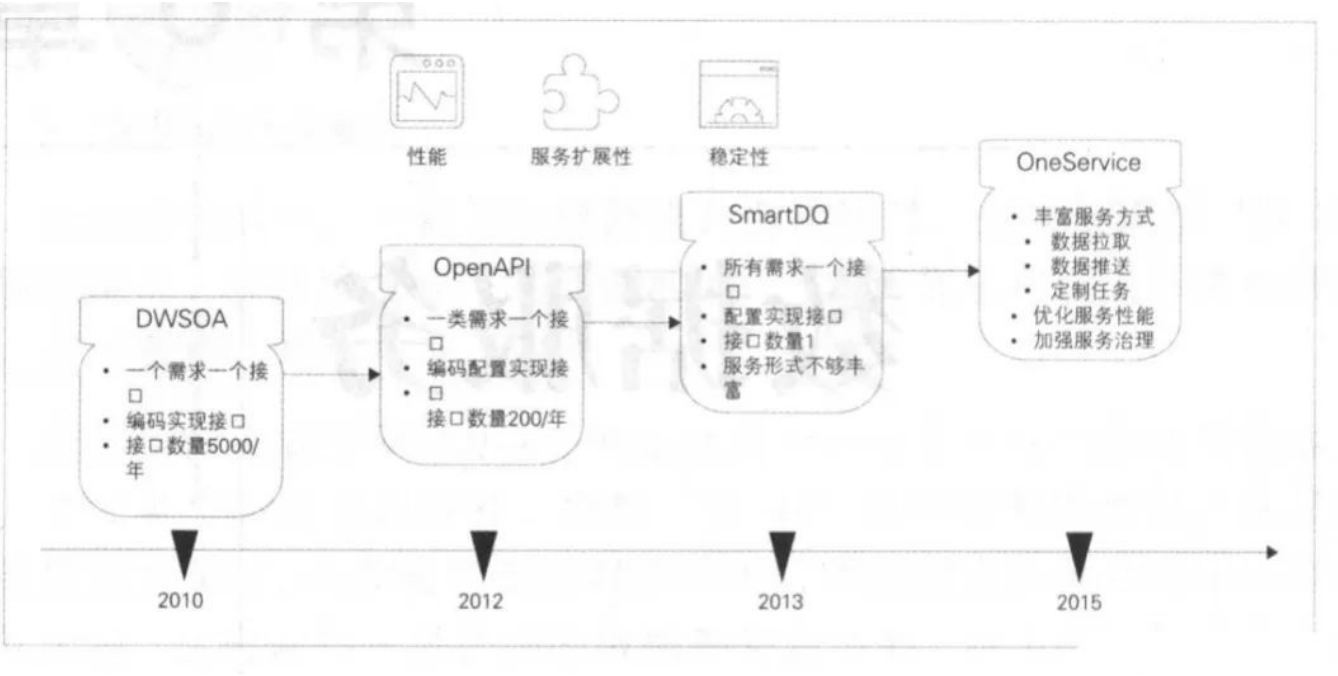
- ODS：直接从业务采集来的原始数据，包含所有业务的变更过程。存储于数据中间件。
- DWD：根据业务过程建模出来的事实明细。存储于数据中间件。
- DWS：各个维度的汇总指标，该维度是各个垂直业务线通用的，落地到存储系统。
- AWS：各个维度的汇总指标，适用于某个业务条线独有的维度，落地到存储系统。
- DIM：实时维表层，由离线的维表导入，离线处理。

4.2.2 多流关联

多个流关联时，只有能匹配上的数据会被输出到下游，否则存储到外部存储系统中，当有更新进来的时候，从外部存储系统中重新读取数据到内存，从已执行完成的部分继续执行。

五、数据服务

5.1 服务架构演进



5.1.1 DWSOA

- SOA服务
- 一个需求对应一到多个接口
- 复用性差、细微改动也要走全套上线流程

5.1.2 OpenAPI

- 为了减少接口数量，将相同维度的数据聚合成一张逻辑宽表，使用相同的接口。
- 维度不可控，随着维度数量的增长，关系映射的维护变得困难。

5.1.3 SmartDQ

- 为了减少维度，使用ORM框架封装了逻辑表，业务方使用SQL来查询数据，只需要关注逻辑表结构，对真实数据源和数据表不关心。
- 接口易上难下，即使一个接口也会绑定一批人。所以对外提供的数据服务接口一定要尽可能抽象，接口的数量要尽可能收敛，最后再保障服务质量的情况下，尽可能减少维护工作量。

5.1.4 OneService

- 但SQL只能满足简单的查询需求，不能解决复杂的业务逻辑。面对不同的场景，创建了多种服务类型。既有支持简单SQL查询的入口，又有面向负责业务逻辑的定制化插件入口。

5.2 性能优化

5.2.1 资源分配

- 剥离复杂的计算逻辑，将其交由数据共同层处理。
- 将Get类型与List类型的查询分为不同的线程池。
- 执行计划优化。比如拆分逻辑表的查询到不同的物理表去执行、将List类型的查

询改变为Get类型的查询等。

5.2.2 缓存优化

- 元数据缓存、逻辑表查询到物理表查询的映射缓存、查询结果缓存。

5.2.3 查询能力

- 合并离线数据查询与实时数据查询，在离线数据无法查到结果的时候即时切换到实时查询。
- 对于需要轮询的数据，采用推送代替轮询。当监听到数据有更新时，推送更新的数据。

六、数据挖掘

数据挖掘过程包括商业理解、数据准备、特征工程、模型训练、模型测试、模型部署、线上应用、效果反馈等环节。

数据中台分为特征层（Featural Data Mining Layer, FDM）、中间层、应用层(Application-oriented Data Mining Layer, ADM)，其中中间层分为个体中间层(Individual Data Mining Layer, IDM)、关系中间层(Relational Data Mining Layer, RDM)。

不同数据层的作用的区别：

- FDM层：用于存储在模型训练前常用的特征指标，并进行统一的清洗和去燥处理，提升机器学习特征工程环节的效率。
- IDM层：个体挖掘指标中间层，面向个体挖掘场景，用于存储通用性强的结果数据，主要包含商品、卖家、买家、行业等维度的个体数据挖掘的相关指标。
- RDM层：关系挖掘指标中间层，面向关系挖掘场景，用于存储通用性强的结果数据，主要包含商品间的相似关系、竞争关系、店铺间的相似关系、竞争关系等。
- ADM层：用来沉淀比较个性化偏应用的数据挖掘指标，比如用户偏好的类目，品牌等，这些数据已经过深度的加工处理，满足某一特点业务或产品的使用。

常见数据挖掘应用：

1.个体挖掘应用

- 用户画像
- 用户身份&同人识别
- 业务指标预测
- ID反作弊

2.关系挖掘应用

- 相似关系挖掘
- 竞争关系挖掘
- 推荐系统