2万字总结阿里大数据之路-数据管理篇（建议收藏）

早安❤❤❤，Baby们，大家好，我是唐三少。上期给大家带来《万字总结阿里大数据之路-数据技术上篇（建议收藏）》，还有昨天发的一篇《2万字总结阿里大数据之路-数据模型中篇（建议收藏）》，反响不错，今天继续给大家带来阿里大数据之路的最后一篇，欢迎大家品读，上号！！！

第1章 元数据

1.1 元数据概述

1.1.1 元数据定义

元数据打通了源数据、数据仓库、数据应用，记录了数据从产生到消费的全过程。元数据主要记录数据仓库中模型的定义、各层级间的映射关系、监控数据仓库的数据状态及 ETL 的任务运行状态。

元数据按用途的不同分为两类：技术元数据（ Technical Metadata) 和业务元数据（Business Metadata ）

- 技术元数据：是存储关于数据仓库系统技术细节的数据，是用于开发和管理数据仓库使用的数据。

  分布式计算系统存储元数据，如表、列、分区等信息。记录了表的表名。分区信息、责任人信息、文件大小、表类型，生命周期，以及列的字段名、字段类型、字段备注、是否是 分区字段等信息。

  分布式计算系统运行元数据，如 MaxCompute 上所有作业运行等 信息：类似于 Hive 的 Job 日志，包括作业类型、实例名称、输入输出、 SQL 、运行参数、执行时间、最细粒度的 FuxiInstance (MaxCompute 中 MR 执行的最小单元）执行信息等。

  数据开发平台中数据同步、计算任务、任务调度等信息，包括数据同步的输入输出表和字段，以及同步任务本身的节点信息：计算任务主要有输入输出、任务本身的节点信息；任务调度主要有任务的依赖类型、依赖关系等，以及不同类型调度任务的运行日 志等。

  数据质量和运维相关元数据，如任务监控、运维报警、数据质量、故障等信息，包括任务监控运行日志、告警配置及运行日志、故障信息等。

- 业务元数据：从业务角度描述了数据仓库中的数据，它提供了介于使用者和实际系统之间的语义层，使得不懂计算机技术的业务人员也能够 “读懂”数据仓库中的数据。

1.1.2 元数据价值

元数据有重要的应用价值，是数据管理、数据内容、数据应用的基础；

- 在数据管理方面为集团数据提供在计算、存储、成本、质量、安全、模型等治理领域上的数据支持。

  例如在计算上可以利用元数据查找超长运行节点，对这些节点进行专项治理，保障基线产出时间。

- 在数据内容方面为集团数据进行数据域、数据主题、业务属性等的提取和分析提供数据素材。

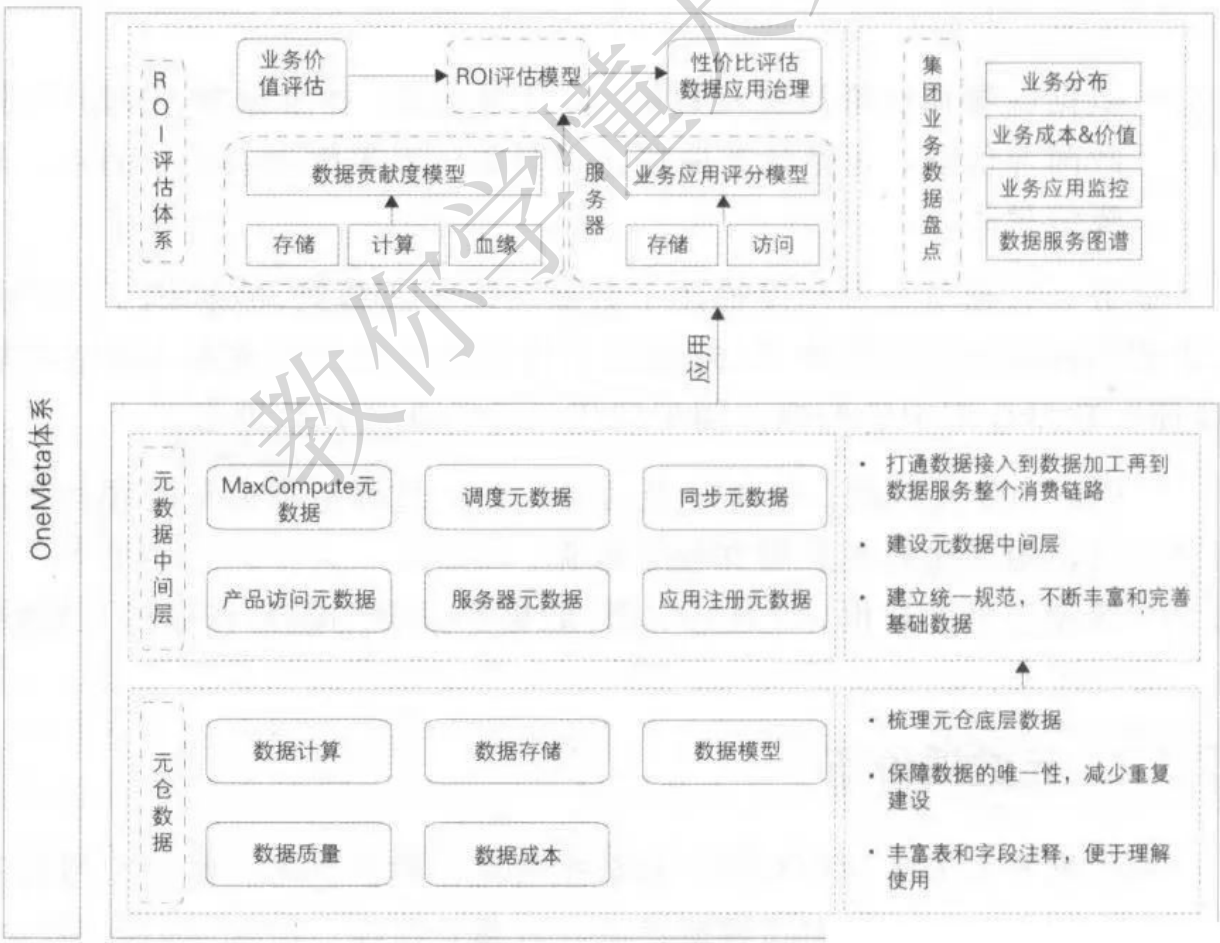
  例如可以利用元数据构建知识图谱，给数据打标签，清楚地知道现在有哪些数据。

- 在数据应用方面打通产品及应用链路，保障产品 数据准确、及时产出。

  例如打通 MaxCompute 和应用数据，明确数据资产等级，更有效地保障产品数据。

1.1.3 统一元数据体系建设

元数据的质量直接影响到数据管理的准确性，如何把元数据建设好将起到至关重要的作用。元数据建设的目标是打通数据接入到加工，再到数据消费整个链路，规范元数据体系与模型，提供统一的元数据服务出口，保障元数据产出的稳定性和质量。



统一元数据体系建设思路图

1.2 元数据应用

价值：数据驱动决策，数字化运营

- 通过数据驱动的方法，我们能够判断趋势，从而展开有效行动，帮助自己发现问 题，推动创新或解决方案的产生
- 对于数据使用者，可以通过元数据让其快速找到所需要的数据；
- 对于ETL工程师，可以通过元数据指导其进行模型设计、任务优化和任务下线等各种日常 ETL工作；
- 对于运维工程师，可以通过元数据指导 其进行整个集群的存储、计算和系统优化等运维工作

1.2.1 Data Profile

核心思路：为纷繁复杂的数据建立一个脉络清晰的血缘图谱。通过图计算、标签传播算法等技术，系统化、自动化地对计算与存储平台上的数 据进行打标、整理、归档，实际承担的是为元数据“画像”的任务，开发了四类标签：

- 基础标签：针对数据的存储情况、访问情况、安全等级等进行打标签。
- 数仓标签：针对数据是增量还是全量、是否可再生、数据的生命周期来进行标签化处理。
- 业务标签：根据数据归属的主题域、产品线、业务类型为数据打上不同的标签。
- 潜在标签：这类标签主要是为了说明数据潜在的应用场景，比如 社交、媒体、广告、电商、金融等。

1.2.2 元数据门户

- 元数据门户致力打造一站式的数据管理平台、高效的一体化数据市场
- “前台”产品为数据地图，定位消费市场，实现检索数据、理解数据等“找数据”需求
- “后台”产品为数据管理，定位于一站式数据管理，实现成本管理、安全管理、质量管理等。

1.2.3 应用链路分析

通过应用链路分析，产出表级血缘、字段血缘和表的应用血缘。其中表级血缘主要有两种计算方式：

- 一种是通过 MR 任务日志进行解析；
- 一种是根据任务依赖进行解析。

常见的应用链路分析应用主要有影响分析、重要性分析、下线分析、链路分析、寻根溯源、故障排查等

1.2.4 数据建模

通过元数据驱动的数据仓库模型建设，可以在一定程度上解决此问题，提高数据仓库建模的数据化指导，提升建模效率。

- 表的基础元数据，包括下游情况、查询次数、关联次数、聚合次数、产出时间等。
- 表的关联关系元数，包括关联表、关联类型、关联字段、关联次数等。
- 表的字段的基础元数据，包括字段名称、字段注释、查询次数、关联次数、聚合次数、过滤次数等。

- 其中查询指 SQL 的 SELECT，关联指SQL 的 JOIN，聚合指 SQL 的 GROUP BY，过滤指 SQL 的WHERE。

星形模型设计中，使用元数据信息有：

- 基于下游使用中关联次数大于某个阈值的表或查询次数大于某个阈值的表等元数据信息，筛选用于数据模型建设的表。
- 基于表的字段元数据，如字段中的时间字段、字段在下游使用中的过滤次数等，选择业务过程标识字段。
- 基于主从表的关联关系、关联次数，确定和主表关联的从表。
- 基于主从表的字段使用情况，如字段的查询次数、过滤次数、关联次数、聚合次数等，确定哪些字段进入目标模型。

1.2.5 驱动 ETL 开发

驱动 ETL 开发示意图

第2章 计算管理

2.1 系统优化

2.1.1 HBO

（History-Based Optimizer，基于历史的优化器）

在任务稳定的情况下，可以考虑基于任务的历史执行情况进行资源评估，即采用HBO

- 提高 CPU 利用率
- 提高内存利用率
- 提高 Instance 并发数
- 降低执行时长

针对“大促”这类数据量暴涨的场景， HBO 也增加了根据数据量动态调整 Instance 数的功能，主要依据 Map 的数据量增长情况进行调整。

2.1.2 CBO

基于代价的优化器，根据收集的统 计信息来计算每种执行方式的代价，进而选择最优的执行方式。

引人了重新排序 Join（JoinReorder）和自动MapJoin（AutoMapJoin ）优化规则等，同时基于 Volcano 模型的优 化器会尽最大的搜索宽度来获取最优计划

可以设置规则白名单（使用哪些优化规则）、黑名单（关闭哪些优化规则）

Optimizer 会提供谓词下推（ PredicatePushDown ）优化，主要目的是尽量早地进行谓词过滤，以减少后续操作的数据量，提高性能。但需要注意的是：

- UDF：对于UDF是否下推，优化器做了限制，不会任意下推这种带有用户意图的函数，主要是因为不同用户书写的函数含义不一样，不可以一概而论。
- 不确定函数：对于不确定函数，优化器也不会任意下推，比如 sample 函数，如果用户将其写在 where 子句中，同时语句存在Join，则优化器是不会下推到 TableScan 的
- 隐式类型转换：书写 SQL 语句时，应尽量避免 Join Key 存在隐式类型转换。

2.2 任务优化

2.2.1 Map 倾斜

在 Map 端读数据时，由于读人数据的文件大小分布不均匀，因此 会导致有些 MapInstance 读取并且处理的数据特别多，而有些 Map Instance 处理的数据特别少，造成 Map 端长尾；

上游表文件的大小特别不均匀，并且小文件特别多，导致当前表 Map 端读取的数据分布不均匀，引起长尾，手段有二：

- 通过对 上游合并小文件 + 调节本节点的小文件 的参数来进行优化
- 通过“distribute by rand(”会将 Map 端分发后的数据重新按照随 机值再进行一次分发

Map 端长尾的根本原因是由于读入的文件块的数据分布不均匀，再 加上 UDF函数性能、Join、聚合操作等，导致读人数据量大的 Map lnstance 耗时较长。在开发过程中如果遇到 Map 端长尾的情况，首先考 虑如何让Map Instance读取的数据量足够均匀，然后判断是哪些操作导致Map Instance 比较慢，最后考虑这些操作是否必须在 Map 端完成， 在其他阶段是否会做得更好。

2.2.2 Join 倾斜

因为数据倾斜导致长尾的现象比较普遍，严重影响任务的执行时 间，尤其是在“双 11”等大型活动期间，长尾程度比平时更严重。比 如某些大型店铺的 PV 远远超过一般店铺的PV，当用浏览日志数据和 卖家维表关联时，会按照卖家 ID 进行分发

- MapJoin 方案：Join 倾斜时，如果某路输入比较小，则可以采用MapJoin避免倾斜；但是MapJoin的使用有限制，必须是Join中的从表比较小才可用
- Join 因为空值导致长尾：将空值处理成随机值
- Join 因为热点值导致长尾：先将热点 key 取出，对于主表数据用热点key切分成热点数据和非热点数据两部分分别处理，最后合并。

2.2.3 Reduce 倾斜

Reduce端产生长尾的主要原因就是key 的数据分布不均匀

- 对同一个表按照维度对不同的列进行Count Distinct操作，造成 Map 端数据膨胀，从而使得下游的 Join 和 Reduce出现链路上的长尾。

- Map 端直接做聚合时出现 key 值分布不均匀，造成 Reduce 端长尾 对热点 key 进行单独处理，然后通过“UnionAll”合并

- 动态分区数过多时可能造成小文件过多，从而引起Reduce 端长尾 把符合不同条件的数据放到不同的分区
  解决小文件过多参数：set odps.sql.reshuffle.dynamicpt=true;

- 多个 Distinct 同时出现在一段 SQL 代码中时，数据会被分发多次， 不仅会造成数据膨胀 N 倍，还会把长尾现象放大 N 倍（常见） 提前Group By，消除 Distinct，即分别把指标 GroupBy 到“原始表的数据粒度”，然后再进行 Join 操作
  当出现的 Distinct 个数不多、表的数据量也不是很大、表的数据分布较均匀时，不使用 Multi Distinct 的计算效果也是可以接受的

第3章 存储和成本管理

3.1 数据压缩

针对3份副本的压缩方案：archive 压缩方法，存储比约为 1:3 提高到 1:1.5

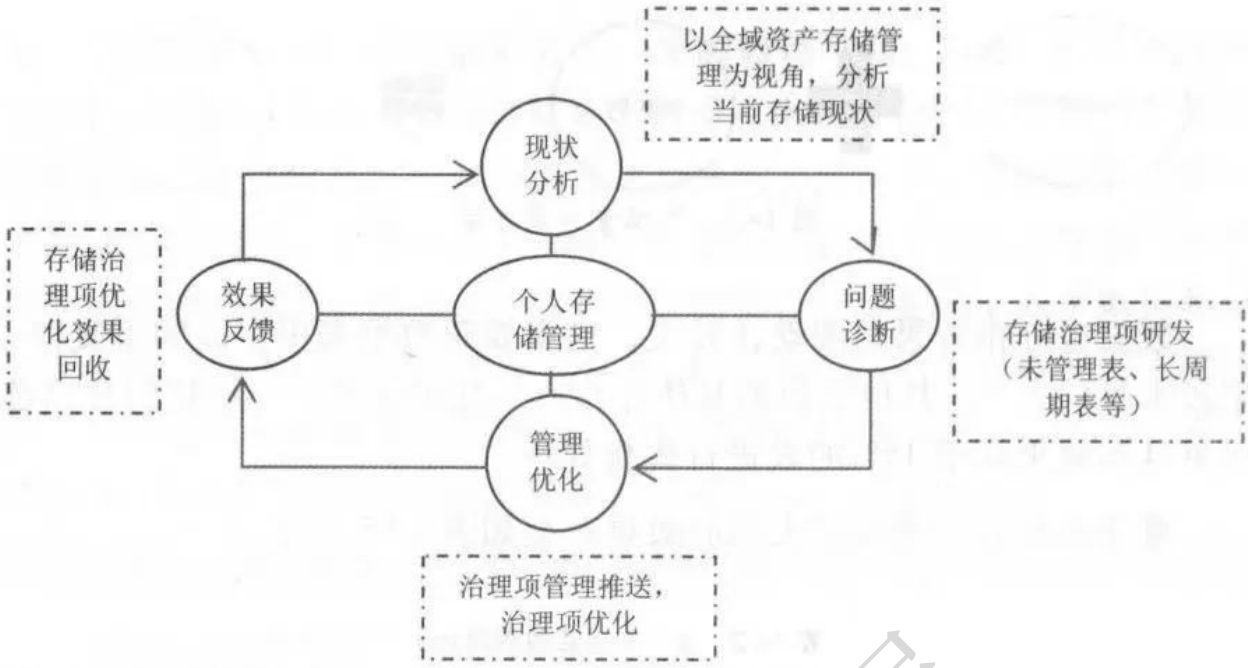
- 恢复数据块的时间将要比原来的方式更长，读的性能会有一 定的损失
- 应用在冷备数据与日志 数据的压缩存储上。

3.2 数据重分布

- 基于列存储，每个表的数据 分布不同，插人数据的顺序不一样，会导致压缩效果有很大的差异，因 此通过修改表的数据重分布，避免列热点，将会节省一定的存储空间。
- 主要通过修改 distributeby 和 sort by字段的方法进行数据重分布
- 一般会筛选出重分布效果高于15%的表进行优化处理

3.3 存储治理项优化

- 优化项有未管理表、空表、 最近62天未访问表、数据无更新无任务表、数据无更新有任务表、开 发库数据大于 100GB 且无访问表、长周期表等



存储治理项优化主要流程图

3.4 生命周期管理

生命周期管理的根本目的就是用最少的存储成本来满足最大的业务需求，使数据价值最大化。

3.4.1 生命周期管理策略

1. 周期性删除策略
2. 彻底删除策略
3. 永久保留策略
4. 极限存储策略
5. 冷数据管理策略
6. 增量表 merge 全量表策略：交易增量数据，使用 订单创建日期或者订单结束日期作为分区，同时将未完结订单放在最大 分区中，对于存储，一个订单在表里只保留一份；对于用户使用，通过 分区条件就能查询某一段时间的数据。

3.4.2 通用的生命周期管理矩阵

1. 历史数据等级划分

- PO：非常重要的主题域数据和非常重要的应用数据，具有不可恢 复性，如交易、日志、集团 KPI 数据、 IPO 关联表。
- P1：重要的业务数据和重要的应用数据，具有不可恢复性，如重 要的业务产品数据。
- P2：重要的业务数据和重要的应用数据，具有可恢复性，如交易 线 ETL 产生的中间过程数据。
- P3：不重要的业务数据和不重要的应用数据，具有可恢复性，如 某些 SNS 产品报表。

生命周期管理矩阵

		P0	P1	P2	P3
ODS 层	事件型流水表（增量表）	永久保留	3 年	365 天	180 天
	事件型镜像表（增量表）	永久保留	3 年	365 天	180 天
	维表（全量表）	33 天+极限存储	33 天+极限存储	33 天+极限存储	33 天+极限存储
	Merge 全量表	2 天	2 天	2 天	2 天
	普通全量表	3 年	365 天	365 天	180 天
	新同步全量表	3 天	3 天	3 天	3 天

续表

		P0	P1	P2	P3
DWD 层	事件型流水表（增量表）	永久保留	3 年	365 天	180 天
	事件型镜像表（增量表）	永久保留	3 年	365 天	180 天
	维表（全量表）	33 天+极限存储	33 天+极限存储	33 天+极限存储	33 天+极限存储
	普通全量表	3 年	365 天	365 天	180 天
DWS 层	各粒度数据	永久保留	3 年	3 年	3 年
临时存储区	ETL 临时表	7 天	3 天	3 天	3 天
	TT 临时数据	7 天	7 天	7 天	7 天
应用层	运营报表	永久保留	—	—	—
	对外数据	7 年	—	—	—
	内部产品	3 年	—	—	—

3.5 数据成本计量

将数据成本定义为存储成本、计算成本和扫描成本三个部分，能够很好地体现出数据在加工链路中的 上下游依赖关系

- 扫描成本：对上游数据表的扫描
- 存储成本：计量数据表消耗的存储资源
- 计算成本：计量数据计算过程中的 CPU 消耗

3.6 数据使用计费

- 根据3.5，分为计算付费、存储付费和扫描付费
- 通过成本计量，可以比较合理地评估出数据加工链路中的成本，从成本的角度
  反映出在数据加工链路中是否存在加工复杂、链路过长、依赖不合理等问题，

间接辅助数据模型优化，提升数据整合效率

- 通过数据使用计费，可以规范下游用户的数据使用方法，提升数据使用效率，从而为业务提供优质的数据服务

第4章 数据质量

4.1 数据质量保障原则

如何评估数据质量的好坏，业界有不同的标准，阿里主要从4个方面进行评估：完整性、准确性、一致性、及时性；

1. 完整性

数据完整性是数据最基础的保障；

- 完整性：指数据的记录和信息是否完整，是否存在缺失的情况；
- 数据缺失：主要包括记录的缺失和记录中某个字段信息的缺失；

  记录的丢失：如，交易中每天只发订单数都在 100 万笔左右，如果某天支付订单突然下降到 1 万笔，很可能是记录丢失了；
  记录中字段的丢失：如，订单的商品 ID、卖家 ID 都是必然存在的，这些字段的空值个数肯定是 0，一旦大于 0 就违背了完整性约束；

2. 准确性

- 准确性：指数据汇总记录的信息和数据是否准确，是否存在异常或者错误的信息；

  准确：数据表中记录的信息与业务过程中真实发生的事实要一致；
  如何判断是否准确：卡点监控 —— 制定相应规则，根据根校验数据，符合规则的数据则认为是准确的；
  如，一笔订单如果出现确认收货金额为负值，或者下单时间在公司成立之前，或者订单没有买家信息等，这些必然是有问题的；

3. 一致性

- 一致性：一般体现在跨度很大的数据仓库体系中，如阿里的数据仓库，内部有很多业务数据仓库分支，对于同一份数据，必须保证一致性；

  一致：也就是指多个业务数据仓库间的公共数据，必须在各个数据仓库中保持一致；
  如，用户 ID，从在线业务库加工到数据仓库，再到各个消费节点，必须都是同一种类型，长度也需要保持一致；
  所以，在阿里建设数据仓库时，才有了公共层的加工，以确保数据的一致性；

4. 及时性

- 及时性：指数据要能及时产出；

  主要体现在数据应用上，要及时产出给到需求方；

- 一般决策支持分析师希望当天就能看到前一天的数据，而不是等三五天才能看到某一个数据分析结果；否则就已失去了数据及时性的价值；

如，阿里“双 11”的交易大屏数据，就要做到秒级；

4.2 数据质量方法概述

阿里的数据质量建设体系：

数据质量建设方法

1. 消费场景知晓

- 功能：分析解决消费场景知晓的问题；
- 方法：通过数据资产等级和基于元数据的应用链路，来分析解决消费场景知晓的问题；

  确定数据资产等级：根据应用的影响程度，确定数据资产的等级；

- 过程：

  根据数据链路血缘，将资产等级上推至各数据生产加工的各个环节，确定链路上所有涉及数据的资产等级，以及在各个加工环节上根据资产等级的不同所采取不同的处理方式；

2. 数据生产加工各个环节卡点校验

主要对两部分的数据卡点校验：在线系统和离线系统数据生产加工各个环节的卡点校验；

- 在线系统：OLTP（On - Line Transaction Processing，联机事务处理）系统；

  在线系统生产加工各环节卡点校验：

  1.根据资产等级的不同，当对应的业务系统变更时，决定是否将变更通知下游；

2.对于高资产等级的业务，当出现新业务数据时，是否纳入统计中，需要卡掉审批；

- 离线系统：OLAP（On - Line Analytical Processing，联机分析处理）系统；

  离线系统生产加工各环节卡点校验：

  主要包括：代码开发、测试、发布、历史或错误数据回刷等环节的卡点校验；
  代码开发阶段、发布前的测试阶段
  针对数据资产等级的不同，对校验的要求有所不同；

3. 风险点监控

风险点监控：主要针对在数据运行过程中可能出现的数据质量和时效等问题进行监控；

主要对两个方面进行风险点监控：

- 在线数据的风险点监控：

  主要针对在线系统日常运行产出的数据进行业务规则的校验；
  主要使用 “实时业务检测平台 BCP（Biz Check Platform）”；

- 离线数据的风险点监控：

  主要是针对离线系统日常运行产出的数据，进行数据质量监控和时效性监控；
  DQC：监控数据质量；
  摩萨德：监控数据时效性；

4. 质量衡量

- 对质量的衡量：

  事前的衡量：如 DQC 覆盖率；
  事后的衡量：
  跟进质量问题，确定质量问题原因、责任人、解决情况等，并用于数据质量的复盘，避免类似事件再次发生；
  根据质量问题对不同等级资产的影响程度，确定其是属于低影响的事件还是具有较大影响的故障；

- 质量分：综合事前和事后的衡量数据进行打分；

5. 质量配套工具

- 针对数据质量的各个方面，都有相关的工具进行保证，以提高效能；

4.2.1 消费场景知晓

- 消费场景知晓的问题：

  数据研发工程师难以确认几百 PB 的数据是否都是重要的？是否都要进行保障？
  是否有一些数据已经过期了？是否所有需要都要精确的进行质量保障？

- 解决方案：数据资产等级方案；

- 产出：

  根据数据产品和应用的影响程度，给数据产品和应用划分资产等级，并打标处

理；
根据数据链路血缘，将资产等级上推至各数据生产加工的各个环节，确定链路上所有涉及数据的资产等级，情打标处理；（等级标签与对应的数据产品 / 应用一致）

1. 数据资产等级定义

背景：针对阿里庞大的数据仓库，数据的规模已经达到 EB 级，对于这么大的数据量，如果一概而论势必会造成精力无法集中、保障无法精确；

五个数据等级，不同性质的重要性一次降低：

- 毁灭性质

即，数据一旦出错，将会引起重大资产损失，面临重大受益损失，造成重大公共风险；

- 全局性质

即，数据直接或间接用于集团业务和效果的评估、重要平台的运维、对外数据产品的透露、影响用户在阿里系网站的行为等；

- 局部性质

即，数据直接或间接用于内部一般数据产品或者运营 / 产品报告，如果出现问题会给事业部或业务线造成影响，或者造成工作效率损失；

- 一般性质

即，数据主要用于小二的日常数据分析，出现问题几乎不会带来影响或者影响很小；

- 未知性质

不能明确说出数据的应用场景，则标注为未知；

1. 对于不同的数据资产等级，使用英文 Asset 进行标记：

毁灭性质：A1 等级；
全局性质：A2 等级；
局部性质：A3 等级；
一般性质：A4 等级；
未知性质：A5 等级；
重要程度：A1 > A2 > A3 > A4 > A5；

如果一份数据出现在多个应用场景中，遵循就高原则；

2. 数据资产等级落地方法

需要解决的问题：对于如此庞大的数据量，如何给每一份数据都打上一个等级标签？

数据资产等级落地的方法 / 步骤：

数据流转过程

- 数据从业务系统中产生，经过同步工具进入数据仓库系统中，在数据仓库中进行一般意义上的清洗、加工、整合、算法、模型等一系列运算；
- 通过同步工具输出到数据产品中进行消费；

数据从业务系统到数据仓库再到数据产品，都是以表的形式体现的，流转过程如下图：

数据流转过程

同步到数据仓库（对应到阿里就是 MaxCompute 平台）中的都是业务数据库的原始表，主要用于承载业务需求，往往不能直接用于数据产品；（一般是 ODS 层的全量数据）

在数据产品中使用的都是经过数据仓库加工后的产出表；（根据需求 / 报表进行加工）

1.划分数据资产等级
2.根据数据流转过程，建立元数据，记录数据表与数据产品或者应用的对应关系；
3.根据影响程度，给数据产品和应用划分数据资产等级；
4.打标：依托元数据的上下游血缘，将整个消费链路打上某一类数据资产标签（也就是对消费链路数据打标）；
链路：指数据从业务系统到数据产品的流转过程；

总结：

通过上述步骤，就完成了数据资产等级的确认，给不同的数据定义了不同的重要程度，需要用到元数据的支撑；

4.2.2 数据加工过程卡点校验

目的：保障数据准确性、保障与离线数据的一致性；

1. 在线业务系统卡点校验（数据产出环节）

- 在线系统数据加工过程卡点校验，主要指在在线系统的数据生产过程中进行的卡点校验；
- 目的：保障与离线数据的一致性；
- 背景 / 问题：在线业务复杂多变，总是在不断变更，每一次变更都会带来数据的变化，因此需要做到两点：
  1、数据仓库需要适应着多变的业务发展，及时做到数据的准确性；
  2、需要高效的将在线业务的变更通知到离线数据仓库；阿里解决上述两个问题的方法：工具和人工双管齐下：既要在工具上自动捕捉每一次业务的变化，同时也要求开发人员在意识上自动进行业务变更通知；

2. 工具

发布平台：发送重大变更的通知；
通知内容：变更原因、变更逻辑、变更测试报告、变更时间等；
数据库平台：发送库表变更通知；
通知内容：变更原因、变更逻辑、变更测试报告、变更时间等；

3. 发布平台

功能：在业务进行重大变更时，订阅发布过程，然后给到离线开发人员，使其知晓此次变更的内容；
注：业务系统繁忙，日常发布变更数不胜数，并不是每一次业务变更都要只会离线业务，那样会造成不必要的浪费，而且影响在线业务迭代的效率；

订阅内容：针对全集团重要的高等级数据资产，整理出哪些变化会影响数据的加工，则订阅这些内容；
如，财报，这个自然是 A1 等级的资产，如果业务系统的改造会影响财报的计算，如约定好的计算口径被业务系统发布变更修改了，那么务必要告知离线业务，作为离线开发人员也必须主动关注这类发布变更信息；

卡点：发布平台集成了通知功能，针对重要的场景发布会进行卡点，确认通知后才能完成发布；

4. 数据库表的变化感知

无论是随着业务发展而做的数据库扩容还是表的 DDL 变化，都需要通知到离线开发人员；

DDL（(Data Definition Language）：数据库模式定义语言；用于描述数据库中要存储的现实世界实体的语言。

DDL 数据库模式定义语言是 SQL 语言（结构化查询语言）的组成部分；

例：CREATE DATABASE（创建数据库）、CREATE TABLE（创建表）；

DML（Data Manipulation Language）：数据操纵语言命令；使用户能够查询数据库以及操作已有数据库中的数据。

例：insert、delete、update、select 等都是 DML；

背景 / 问题：数据仓库在进行数据抽取时，采用的是 DataX 工具，可能限制了某个数据库表，如果发生数据库扩容或者迁移，DataX 工具是感知不到的，结果可能会导致数据抽取错漏，影响一系列的下游应用；

解决方法：通过数据库平台发送库表变更通知；

5. 开发人员

数据资产等级的上下游打通，同样也要将这个过程给到在线开发人员，使其知晓哪些是重要的核心数据资产，哪些暂时还只是作为内部分析数据使用；

要提高在线开发人员的意识，通过培训，将离线数据的诉求、离线数据的加工过程、数据产品的应用方式，告诉在线业务开发人员，使其意识到数据的重要性，了解数据的价值，同时也告知出错后果，使在线开发人员在完成业务目标时，也要注重数据的目标，做到业务端和数据端一致；

6. 离线系统卡点校验（数据离线加工环节）

背景 / 问题：数据从在线业务系统到数据仓库再到数据产品的过程中，需要在数据仓库这一层完成数据的清洗、加工；正是有了数据的加工，才有了数据仓库模型和数据仓库代码的建设；如何保障数据加工过程中的质量，是离线数据仓库保障数据质量的一个重要环节；

目的：保障数据加工过程中的质量（主要指数据的准确性）；

在两个环节进行卡点校验：

7. 代码提交时的卡点校验

背景 / 原因：数据研发人员素质不同，代码能力也有差异，代码质量难以得到高效保障；

解决方法：开发代码扫描工具 SQLSCAN，针对每一次提交上线的代码进行扫描，将风险点提取出来；

卡点方式：使用代码扫描工具 SQLSCAN，扫描代码提取风险点；

8. 任务发布上线时的卡点校验

为了保障线上数据的准确性，每一次变更都需要线下完成测试后在发布到线上环境中，线上测试通过后才算发布成功；

卡点方式：分别对任务（指变更的业务）发布上线前和上线后进行测试；

9. 发布上线前的测试：主要包括 Code Review 和回归测试；

- Code Review：是一种通过复查代码提高代码质量的过程；
- 回归测试：指修改了旧代码后，重新进行测试以确认修改没有引入新的错误或导致其他代码产生错误；

回归测试的目的：
保障新逻辑的正确；

保证不影响非此次变更的逻辑；

注：对于资产等级较高的任务变更发布，采用强阻塞的形式，必须通过在彼岸完成回归测试之后才允许发布；

10. 发布上线后的测试：在线上做 Dry Run 测试或者真是环境运行测试；

- Dry Run 测试：

不执行代码，仅运行执行计划，避免线上和线下环境不一致导致语法错误；

- 真实环境的运行测试：

使用真实数据进行测试；

- 节点变更或数据重刷新前的变更通知

通知内容：变更原因、变更逻辑、变更测试报告、变更时间等；
过程：
使用通知中心，将变更原因、变更逻辑、变更测试报告、变更时间等自动通知下游，下游对此次变更没有异议后，再按照约定时间执行发布变更，将变更对下游的影响降低至最低；

4.2.3 风险点监控

风险点监控：主要指针对数据在日常运行过程中容易出现的风险进行监控，并设置报警机制；
主要包括在线数据和离线数据运行风险点监控；

目的：保障数据的准确性；

1、 在线数据风险点监控

- 目的：减少了在线业务系统产生的脏数据，为数据准确性把第一道关；
  另外，减少用户错误信息的投诉，也减少了离线数据错误的回滚；

- BCP：阿里的实时业务检测平台；

- 思路 / 监控过程：在每一个业务系统中，当完成业务过程进行数据落库时，BCP 订阅一份相同的数据，根据提前设定好的业务规则，在 BCP 系统中进行逻辑校验，当校验不通过时，以报警的形式披露出来，给到规则订阅人，以完成数据的校对；

- BCP 的校验过程：
  获取数据源：用户在 BCP 平台订阅数据源，获取需要校验的数据源；
  编写规则：针对所订阅的数据源进行规则的编写，即校验的逻辑；

- 规则 / 逻辑：是至关重要的，是校验的核心，只有通过了这些规则，才认定该条记录是对的；
  如，针对 “订单拍下时间” 进行校验；逻辑：订单的拍下时间肯定不会大于当天的时间，也不会小于淘宝创立的时间；

配置告警：针对不同的规则配置不同的告警形式；

注：由于 BCP 的配置和运行成本较高，主要根据数据资产等级进行监控；

- 离线数据风险点监控

离线数据风险点监控主要包括对数据准确性和数据产出的及时性进行监控；

- 数据准确性监控

数据准确性是数据质量的关键，因此数据准确成为数据质量的重中之重，是所有离线系统加工时的第一保障要素；

方法：通过 DQC 进行数据准确性监控；

DQC（Data Quality Center，数据质量中心）：主要关注数据质量，通过配置数据质量校验规则，自动在数据处理任务过程中进行数据质量方面的监控；

注：监控数据质量并报警，其本身不对数据产出进行处理，需要报警接收人判断并决定如何处理；

监控方式：通过配置数据质量检验规则，自动在数据处理任务过程中进行监控；

监控规则：

强规则：会阻断任务的执行；

将任务置为失败状态，其下游任务将不会被执行；

弱规则：只告警而不会阻断任务的执行；

常见的 DQC 监控规则：主键监控、表数据量及波动监控、重要字段的非空监控、重要枚举字段的离散值监控、指标值波动监控、业务规则监控等；

规则配置：依赖数据资产等级确定监控规则；

DQC 检查其实也是运行 SQL 任务，只是这个任务是嵌套在主任务中的，一旦检查点太多自然就会影响整体的性能；因此还是依赖数据资产等级来确定规则的配置情况；

注：不同的业务会有业务规则的约束，这些规则来源于数据产品或者说消费的业务需求，有消费节点进行配置，然后上推到离线系统的起点进行监控，做到规则影响最小化；

- 数据及时性

在确保数据准确性的基础上，需要进一步让数据能够及时的提供服务；否则数据的价值将大幅度降低，甚至没有价值；

阿里的大部分离线任务：

一般以天为时间间隔，称为“天任务”，对于天任务，数据产品或者数据决策报表一般都要求在每天 9:00 甚至更早的时间产出；

为了确保前一天的数据完整，天任务是从零点开始运行的，由于计算加工的任务都是在夜里运行的，而要确保每天的数据能够按时产出，需要进行一系列的报警和优先级设置，使得重要的任务优先且正确的产出；

重要的任务：资产等级较高的业务；

- 任务优先级

对于 Map 任务和 Reduce 任务，调度是一个树形结构（RelNode 树），当配置了叶子节点（RelNode 节点）的优先级后，这个优先级会传递到所有上游节点，所以优先级的设置都是给到叶子节点，而叶子节点往往就是服务业务的消费节点；

设置优先级：首先确定业务的资产等级，等级高的业务所对应的消费节点自然配置高优先级，一般业务则对应低优先级，确保高等级业务准时产出；

- 任务报警

任务报警和优先级类似，也是通过叶子节点传递；

任务在运行过程中难免会出错，因此要确保任务能够高效、平稳的执行，需要有一个监控报警系统，对于高优先级的任

务，一旦发现任务出错或者可能出现产出延迟，就要报警给到任务和业务 Owner；

摩萨德：阿里自主开发的监控报警系统；

- 摩萨德

摩萨德：离线任务的监控报警系统；是数据运维不可或缺的保障工具；

根据离线任务的运行情况实时决策是否告警、何时告警、告警方式、告警给谁等；

两个主要功能：强保障监控、自定义告警；

强保障监控

强保障监控是摩萨德的核心功能，是仅仅围绕运维目标即业务保障而设计的，只要在业务的预警时间受到威胁，摩萨德就一定会告警出来给到相关人员；

强保障监控主要包括：

监控范围：设置强保障业务的任务及其上游所有的任务都会被监控；

监控的异常：任务出错、任务变慢、预警业务延迟；

告警对象：默认是任务 Owner，也可以设置值班表到某一个人；

何时告警：根据业务设置的预警时间判断何时告警；

业务延迟预警和出错报警，都是根据 “产出预警时间“ 来判断的；

产出预警时间：摩萨德根据当前业务上所有任务最近 7 天运行的平均时间来推算当前业务所用的大概时间，来作为产出预警时间；

告警方式：根据业务的重要紧急程度，支持电话、短信、旺旺、邮件告警；

例：生意参谋业务（预警业务延迟）

资产等级及需求：定义的资产等级是 A2，要求早上 9:00 产出数据给到上架；

设置：给生意参谋业务定义一个强保障监控，业务产出时间是 9:00，业务预警时间是 7:00；

这里的预警时间是指，一旦摩萨德监控到当前业务的产出时间超出预警时间时，就会打电话给值班人员进行预警；

如，摩萨德推测生意参谋的产出时间要到 7:30，那么电话告警就出来了，由值班人员来判断如何加速产出；产出时间推算（预警判断，也就是产出延迟判断）：摩萨德是根据当前业务上所有任务最近 7 天运行的平均时间来推算的；虽然有误判的可能性，但是总体还是非常准确的，可以接受；

- 自定义监控

自定义监控是摩萨德比较轻量级的监控功能，用户可以根据自己的需求进行配置，主要包括：

1. 出错告警：可根据应用、业务、任务三个监控对象进行出错告警配置，监控对象出错即告警给到人 / Owner / 值班表；
2. 完成告警：可根据应用、业务、任务三个监控对象进行完成情况告警配置，监控对象完成即告警给到人 / Owner / 值班表；
3. 未完成告警：可根据应用、业务、任务三个监控对象进行未完成情况告警配置，监控对象未完成即告警给到人 / Owner / 值班表；

4. 周期性告警：针对某个周期的小时任务，如果在某个时间未完成，即告警给到人 / Owner / 值班表；

5. 超时告警：根据任务运行时长进行超时告警配置，任务运行超过指定时间即告警给到人 / Owner / 值班表；

- 摩萨德甘特图的服务

针对业务的运行情况，摩萨德会提供一天关键路径，即完成业务的最慢任务链路图；因为每个业务上游可能有成千上万个任务，所以这条关键路径对于业务链路优化来说非常重要；

4.2.4 质量衡量

保障数据仓库的数据质量，有很多方案，评价这些方案的优劣，需要一套度量指标：

- 数据质量起夜率

一般数据仓库的作业任务都是在夜晚进行，一旦出现问题就需要值班人员起夜进行处理；

起夜率：用每个月的起夜次数，作为衡量数据质量建设完善度的一个指标；

- 数据质量事件

数据质量事件：记录每一次数据质量的问题；

针对每一个数据质量问题，都记录一个数据质量事件：

功能：既用来衡量数据本身的质量，也用来衡量数据链路上下游的质量，是数据质量的一个重要度量指标；

1. 用来跟进数据质量问题的处理过程；
2. 用来归纳分析数据质量原因；
3. 根据数据质量原因来查缺补漏，既要找到数据出现问题的原因，也要针对类似问题给出后续预防方案；

- 数据质量故障体系

对于严重的数据质量事件，将升级为故障；

故障：指问题造成的影响比较严重，已经给公司带来资产损失或者公关风险；

背景：数据从采集到最后的消费，整个链路要经过几十个系统，任何一个环节出现问题，都会影响数据的产出，因此需要一种机制，能够将各团队绑在一起，目标一致，形成合力，故障体系在这个背景下应运而生；

故障体系中，一旦出现故障，就会通过故障体系，要求相关团队在第一时间跟进解决问题，消除影响；

1. 故障定义

首先识别出重要的业务数据，并注册到系统中，填写相关的业务情况，如技术负责人、业务负责人、数据应用场景、延迟或错误带来的影响、是否会发生资产损失等，完成后，会将这部分数据的任务挂到平台基线上，一旦延迟或错误，即自动生成故障单，形成故障；

2. 故障等级

故障发生后，会根据一定的标准判断故障等级，如故障时长、客户投诉量、资金损失等，将故障按 P1~ P4 定级，各团队会有故障分的概念，到年底会根据故障分情况来判断本年度的运维效果；

3. 故障处理

故障发生后，需要快速的识别故障原因，并迅速解决，消除影响；

在处理故障的过程中，会尽量将故障的处理进度通知到相关方，尽可能减少对业务的影响；

4. 故障 Review

故障 Review：即分析故障的原因、处理过程的复盘、形成后续解决的 Action，并且都会以文字的形式详细记录，对故障的责任进行归属，一般会责任到人；

注：对故障责任的判定，不是为了惩罚个人，而是通过对故障的复盘形成解决方案，避免问题再次发生；