PB级企业电商离线数仓项目实战【上】（讲师回灯）

项目背景

人类正从IT时代走向DT(Data Technology)时代。在DT时代，人们比以往任何时候更能收集到更丰富的数据。IDC 的报告显示：预计到2020年，全球数据总量将超过40ZB（相当于40 万亿GB），这一数据量是2011年的22倍。正在呈“爆炸式”增长的数据，其潜在的巨大价值有待发掘。

如果不能对这些数据进行有序、有结构地分类组织和存储，不能有效利用并发掘它，继而产生价值，那么它同时也成为一场“灾难”。无序、无结构的数据犹如堆积如山的垃圾，给企业带来的是令人咋舌的高额成本。

日益丰富的业态，也带来了各种各样、纷繁复杂的数据需求。 如何有效地满足企业决策层、管理层、员工、商家、合作伙伴等多样化的需求，提高他们对数据使用的满意度，是数据服务和数据产品需要面对的挑战。

- 如何建设高效的数据模型和体系，使数据易用，避免重复建设和数据不一致性，保证数据的规范性；
- 如何提供高效易用的数据开发工具；
- 如何做好数据质量保障；
- 如何有效管理和控制日益增长的存储和计算消耗，保证数据服务的稳定，保证其性能；

这些都给大数据系统的建设提出了更多的要求。

这里介绍的电商离线数据仓库项目，正是为了满足不断变化的业务需求，实现系统的高度扩展性、灵活性以及数据展现的高性能而设计的。整个项目的讲解分为以下几个部分：

- 上半部分
    - 数据仓库理论
    - 电商离线数据仓库设计
    - 会员活跃度分析
    - 广告分析
- 下半部分
    - 核心交易分析
    - 任务调度

- 血缘关系和数据管理
- 数据质量监控
- 即席查询

第一部分 数据仓库理论

第1节 数据仓库

1.1 什么是数据仓库

1988年，为解决全企业集成问题，IBM公司第一次提出了信息仓库（Information Warehouse）的概念。数据仓库的基本原理、技术架构以及分析系统的主要原则都已确定，数据仓库初具雏形。

1991年Bill Inmon（比尔·恩门）出版了他的第一本关于数据仓库的书《Building the Data Warehouse》，标志着数据仓库概念的确立。书中指出，**数据仓库(Data Warehouse)是一个面向主题的(Subject Oriented)、集成的(Integrated)、相对稳定的(Non-Volatile)、反映历史变化的(Time Variant)数据集合，用于支持管理决策(Decision-Making Support)**。该书还提供了建立数据仓库的指导意见和基本原则。凭借着这本书，Bill Inmon被称为数据仓库之父。

1.2 数据仓库四大特征

- 面向主题的
- 集成的
- 稳定的
- 反映历史变化的

面向主题的

与传统数据库**面向应用**进行数据组织的特点相对应，数据仓库中的数据是**面向主题**进行组织的。

什么是主题呢?

- 主题是一个抽象的概念，是较高层次上企业信息系统中的数据综合、归类并进行分析利用的抽象
- 在逻辑意义上，它是对应企业中某一宏观分析领域所涉及的分析对象

面向主题的数据组织方式，就是在较高层次上对分析对象的数据的一个完整、一致的描述，能完整、统一地刻划各个分析对象所涉及的企业的各项数据，以及数据之间的联系。所谓较高层次是相对面向应用的数据组织方式而言的，是指按照主题进行数据组织的方式具有更高的数据抽象级别。

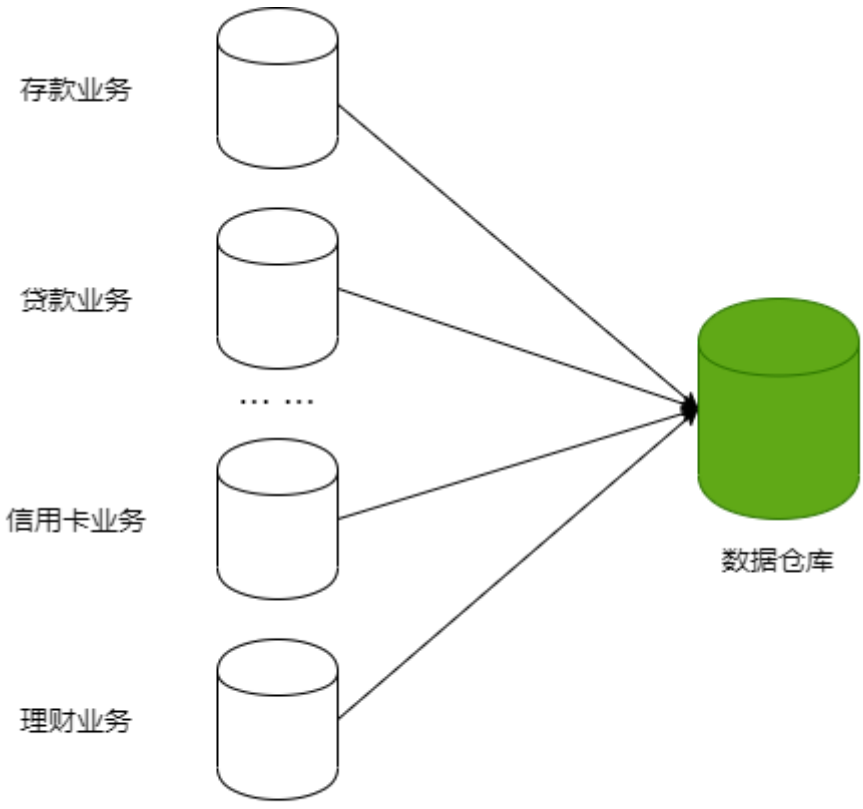
例如销售情况分析就是一个分析领域，那么数据仓库的分析主题可以是“销售分析”。

集成的

数据仓库的数据是从原有的分散的多个数据库、数据文件、用户日志中抽取来的，数据来源可能既有内部数据又有外部数据。操作型数据与分析型数据之间差别很大：

- 数据仓库的每一个主题所对应的源数据，在原有的各分散数据库中有重复和不一致的地方，且来源于不同的联机系统的数据与不同的应用逻辑捆绑在一起
- 数据仓库中的数据很难从原有数据库系统直接得到。数据在进入数据仓库之前，需要经过统一与综合

数据仓库中的数据是为分析服务的，而分析需要多种广泛的不同数据源以便进行比较、鉴别，数据仓库中的数据会从多个数据源中获取，这些数据源包括多种类型数据库、文件系统以及Internet网上数据等，它们通过数据集成而形成数据仓库中的数据。



客户价值分析，需要整合多个业务数据库

稳定的

数据仓库数据反映的是一段相当长的时间内历史数据的内容，是不同时点的数据库快照的集合，以及基于这些快照进行统计、综合和重组的导出数据。

数据稳定主要是针对应用而言。数据仓库的用户对数据的操作大多是数据查询或比较复杂的挖掘，一旦数据进入数据仓库以后，一般情况下被较长时间保留。数据经加工和集成进入数据仓库后是极少更新的，通常只需要定期的加载和更新。

反映历史变化的

数据仓库包含各种粒度的历史数据。数据仓库中的数据可能与某个特定日期、星期、月份、季度或者年份有关。虽然数据仓库不会修改数据，但并不是说数据仓库的数据是永远不变的。数据仓库的数据也需要更新，以适应决策的需要。数据仓库的数据随时间的变化表现在以下几个方面：

- 数据仓库的数据时限一般要远远长于操作型数据的数据时限
- 业务系统存储的是当前数据，而数据仓库中的数据是历史数据
- 数据仓库中的数据是按照时间顺序追加的，都带有时间属性

1.3 数据仓库作用

整合企业业务数据，建立统一的数据中心；

产生业务报表，了解企业的经营状况；

为企业运营、决策提供数据支持；

可以作为各个业务的数据源，形成业务数据互相反馈的良性循环；

分析用户行为数据，通过数据挖掘来降低投入成本，提高投入效果；

开发数据产品，直接或间接地为企业盈利；

1.4 数据仓库与数据库的区别

数据库与数据仓库的区别实际讲的是 OLTP 与 OLAP 的区别。

OLTP（On-Line Transaction Processing 联机事务处理），也称面向交易的处理系统。主要针对具体业务在数据库系统的日常操作，通常对少数记录进行查询、修改。用户较为关心操作的响应时间、数据的安全性、完整性和并发支持的用户数等问题。传统的数据库系统作为数据管理的主要手段，主要用于操作型处理。

OLAP（On-Line Analytical Processing 联机分析处理），一般针对某些主题的历史数据进行分析，支持管理决策。

数据仓库的出现，并不是要取代数据库：

- 数据仓库主要用于解决企业级的数据分析问题或者说管理和决策
- 数据仓库是为分析数据而设计，数据库是为捕获和存储数据而设计
- 数据仓库是面向分析，面向主题设计的，即信息是按主题进行组织的，属于分析型；数据库是面向事务设计的，属于操作型

  数据仓库在设计是有意引入数据冗余（目的是为了提高查询的效率），采用反范式的方式来设计；数据库设计是尽量避免冗余（第三范式），一般采用符合范式的规则来设计
- 数据仓库较大，数据仓库中的数据来源于多个异构的数据源，而且保留了企业的历史数据；数据库存储有限期限、单一领域的业务数据

数据仓库的出现，并不是要取代数据库：

- 数据库是面向事务的设计，数据仓库是面向主题设计的
- 数据库存储有限期限的业务数据，数据仓库存储的是企业历史数据
- 数据库设计尽量避免冗余，数据存储设计满足第三范式，但是便于进行数据分析。数据仓库在设计时有意引入冗余，依照分析需求，分析维度、分析指标进行设计
- 数据库是为捕获数据而设计，数据仓库是为分析数据而设计

以银行业务为例。数据库是事务系统的数据平台，客户在银行做的每笔交易都会写入数据库，被记录下来，这里，可以简单地理解为用数据库记账。数据仓库是分析系统的数据平台，它从事务系统获取数据，并做汇总、加工，为决策者提供决策的依据。比如，某银行某分行一个月发生多少交易，该分行当前存款余额是多少。如果存取款多，消费交易多，那么该地区就有必要设立ATM了。

银行的交易量是巨大的，通常以百万甚至千万次来计算。事务系统是实时的，这就要求时效性，客户存一笔钱需要几十秒是无法忍受的，这就要求数据库只能存储很短一段时间的数据。而分析系统是事后的，它要提供关注时间段内所有的有效数据。这些数据是海量的，汇总计算起来也要慢一些，但是，只要能够提供有效的分析数据就达到目的了。

数据仓库是在数据库已经大量存在的情况下，为了进一步挖掘数据资源、为了决策需要而产生的，它决不是所谓的大型数据库。

对比内容	数据库	数据仓库
数据内容	近期值、当前值	历史的、归档的数据
数据目标	面向业务操作	面向管理决策、面向分析（主题）
数据特性	动态频繁更新	静态、不能直接更新；定时添加数据
数据结构	高度结构化、满足第三范式	简单的、冗余的、满足分析的
使用频率	高	低
数据访问量	访问量大；每次访问的数据量少	访问量小；每次访问的数据量大
对响应时间的要求	高	低（不敏感）

1.5 数据集市

数据仓库（DW）是一种反映主题的全局性数据组织。但全局性数据仓库往往太大，在实际应用中将它们按部门或业务分别建立反映各个子主题的局部性数据组织，即数据集市（Data Mart），有时也称它为部门数据仓库。

数据集市：是按照主题域组织的数据集合，用于支持部门级的数据分析与决策。如在商品销售的数据仓库中可以建立多个不同主题的数据集市：

- 商品采购数据集市
- 商品库存数据集市
- 商品销售数据集市

数据集市仅仅是数据仓库的某一部分，实施难度大大降低，并且能够满足企业内部部分业务部门的迫切需求，在初期获得了较大成功。但随着数据集市的不断增多，这种架构的缺陷也逐步显现。企业内部独立建设的数据集市由于遵循不同的标准和建设原则，以致多个数据集市的数据混乱和不一致，形成众多的数据孤岛。

企业发展到一定阶段，出现多个事业部，每个事业部都有各自数据，事业部之间的数据往往都各自存储，各自定义。每个事业部的数据就像一个个孤岛一样无法(或者极其困难)和企业内部的其他数据进行连接互动。这样的情况称为数据孤岛，简单说就是数据间缺乏关联性，彼此无法兼容。

第2节 数据仓库建模方法

数据模型就是数据组织和存储方法，它强调从业务、数据存取和使用角度合理存储数据。有了适合业务和基础数据存储环境的模型，能获得以下好处：

- 性能：良好的数据模型能帮助我们快速查询所需要的数据，减少数据的I/O吞吐
- 成本：良好的数据模型能极大地减少不必要的数据冗余，也能实现计算结果复用，极大地降低大数据系统中的存储和计算成本
- 效率：良好的数据模型能极大地改善用户使用数据的体验，提高使用数据的效率
- 质量：良好的数据模型能改善数据统计口径的不一致性，减少数据计算错误的可能性

大数据系统需要数据模型方法来帮助更好地组织和存储数据，以便在性能、成本、效率和质量之间取得最佳平衡。

2.1 ER模型

数据仓库之父Bill Inmon提出的建模方法是从全企业的高度设计一个3NF模型，用实体关系(Entity Relationship, ER)模型描述企业业务，在范式理论上符合3NF。数据仓库中的3NF与OLTP系统中的3NF 的区别在于，它是站在企业角度面向主题的抽象，而不是针对某个具体业务流程的实体对象关系的抽象。其具有以下几个特点：

- 需要全面了解整个企业业务和数据
- 实施周期非常长
- 对建模人员的能力要求非常高

采用ER模型建设数据仓库模型的出发点是整合数据，将各个系统中的数据以整个企业角度按主题进行相似性组合和合并，并进行一致性处理，为数据分析决策服务，但是并不能直接用于分析决策。其建模步骤分为三个阶段：

- 高层模型：一个高度抽象的模型，描述主要的主题以及主题间的关系，用于描述企业的业务总体概况
- 中层模型：在高层模型的基础上，细化主题的数据项
- 物理模型(也叫底层模型)：在中层模型的基础上，考虑物理存储，同时基于性能和平台特点进行物理属性的设计，也可能做一 些表的合并、分区的设计等

2.2 维度模型

维度模型是数据仓库领域的Ralph Kimball大师所倡导的，他的《数据仓库工具箱》是数据仓库工程领域最流行的数据仓库建模经典。

维度建模从分析决策的需求出发构建模型，为分析需求服务，重点关注用户如何更快速地完成需求分析，同时具有较好的大规模复杂查询的响应性能。其典型的代表是星型模型，以及在一些特殊场景下使用的雪花模型。其设计分为以下几个步骤：

- 选择需要进行分析决策的业务过程。业务过程可以是：
    - 单个业务事件，比如交易的支付、退款等
    - 某个事件的状态，比如当前的账户余额等
    - 一系列相关业务事件组成的业务流程
- 选择数据的粒度。在事件分析中，我们要预判所有分析需要细分的程度，从而决定选择的粒度
- 识别维表。选择好粒度之后，就需要基于此粒度设计维表，包括维度属性，用于分析时进行分组和筛选
- 选择事实。确定分析需要衡量的指标

现代企业业务变化快、人员流动频繁、业务知识功底的不够全面，导致ER模型设计产出周期长。大多数企业实施数据仓库的经验说明：在不太成熟、快速变化的业务面前，构建ER模型的风险非常大，不太适合去构建ER模型。而维度建模对技术要求不高，快速上手，敏捷迭代，快速交付；更快速完成分析需求，较好的大规模复杂查询的响应性能。

第3节 数据仓库分层

数据仓库更多代表的是一种对数据的管理和使用的方式，它是一整套包括了数据建模、ETL（数据抽取、转换、加载）、作用调度等在内的完整的理论体系流程。数据仓库在构建过程中通常都需要进行分层处理。业务不同，分层的技术处理手段也不同。

分层的主要原因是在管理数据的时候，能对数据有一个更加清晰的掌控。详细来讲，主要有下面几个原因：

- **清晰的数据结构**

  每一个数据分层都有它的作用域，在使用表的时候能更方便地定位和理解。

- **将复杂的问题简单化**

将一个复杂的任务分解成多个步骤来完成，每一层只处理单一的问题，比较简单和容易理解。而且便于维护数据的准确性，当数据出现问题之后，可以不用修复所有的数据，只需要从有问题的地方开始修复。

- **减少重复开发**

  规范数据分层，开发一些通用的中间层数据，能够减少极大的重复计算。

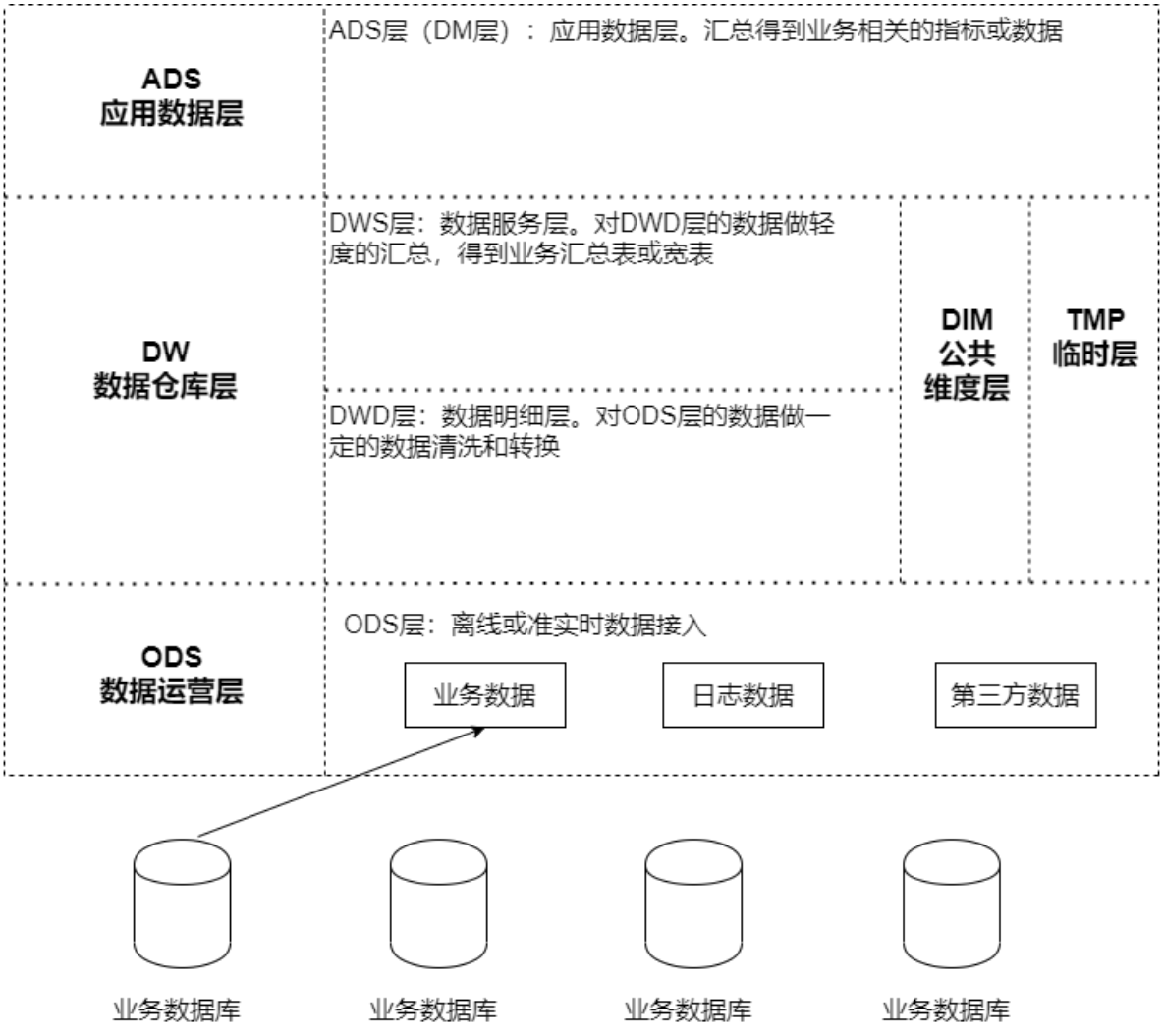
- **屏蔽原始数据的异常**

  屏蔽业务的影响，不必改一次业务就需要重新接入数据。

- **数据血缘的追踪**

  最终给业务呈现的是一个能直接使用业务表，但是它的来源很多，如果有一张来源表出问题了，借助血缘最终能够快速准确地定位到问题，并清楚它的危害范围。

数仓的常见分层一般为3层，分别为：数据操作层、数据仓库层和应用数据层（数据集市层）。当然根据研发人员经验或者业务，可以分为更多不同的层，只要能达到流程清晰、方便查数即可。



ODS（Operation Data Store 数据准备区）。数据仓库源头系统的数据表通常会**原封不动的存储一份**，这称为ODS层，也称为准备区。它们是后续数据仓库层加工数据的来源。ODS层数据的主要来源包括：

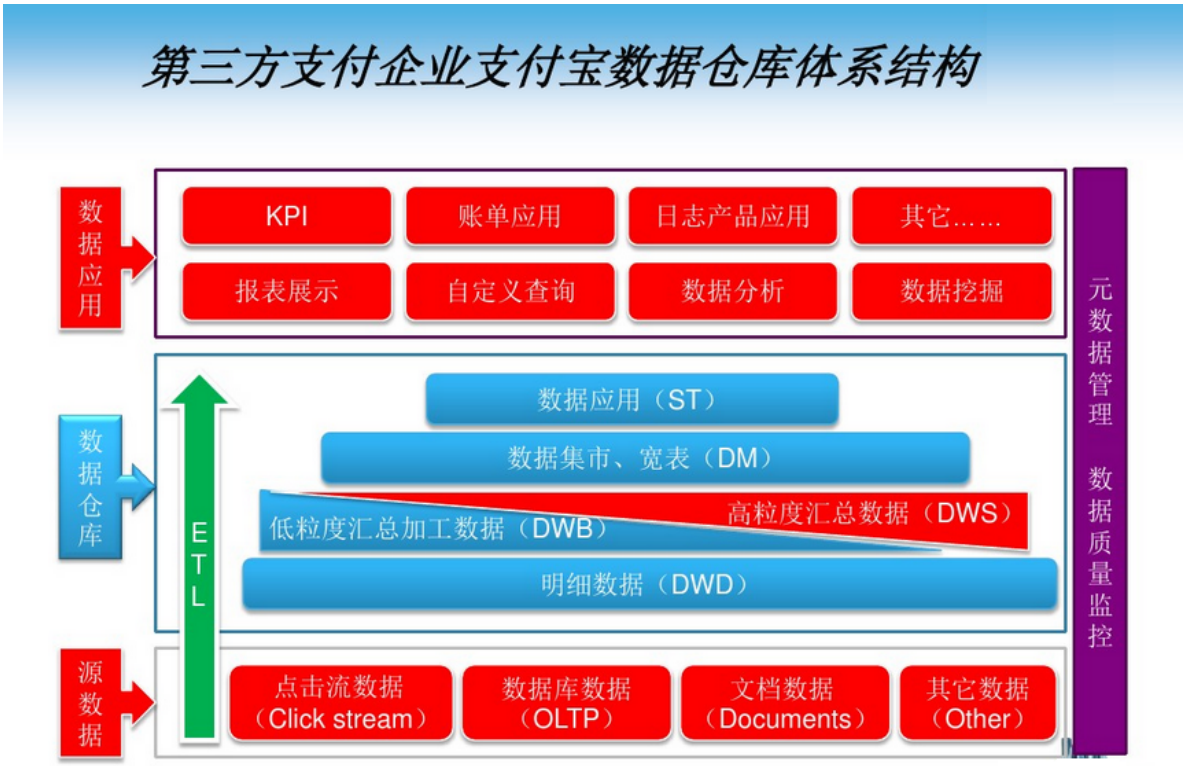
- 业务数据库。可使用DataX、Sqoop等工具来抽取，每天定时抽取一次；在实时应用中，可用Canal监听MySQL的 Binlog，实时接入变更的数据；
- 埋点日志。线上系统会打入各种日志，这些日志一般以文件的形式保存，可以用 Flume 定时抽取；
- 其他数据源。从第三方购买的数据、或是网络爬虫抓取的数据；

DW（Data Warehouse 数据仓库层）。包含DWD、DWS、DIM层，由ODS层数据加工而成。主要完成数据加工与整合，建立一致性的维度，构建可复用的面向分析和统计的明细事实表，以及汇总公共粒度的指标。

- DWD（Data Warehouse Detail 细节数据层），是业务层与数据仓库的隔离层。以业务过程作为建模驱动，基于每个具体的业务过程特点，构建细粒度的明细层事实表。可以结合企业的数据使用特点，将明细事实表的某些重要维度属性字段做适当冗余，也即宽表化处理；
- DWS（Data Warehouse Service 服务数据层），基于DWD的基础数据，整合汇总成分析某一个主题域的服务数据。以分析的主题为建模驱动，基于上层的应用和产品的指标需求，构建公共粒度的汇总指标事实表；
- 公共维度层（DIM）：基于维度建模理念思想，建立一致性维度；
- TMP层：临时层，存放计算过程中临时产生的数据；

ADS（Application Data Store 应用数据层）。基于DW数据，整合汇总成主题域的服务数据，用于提供后续的业务查询等。

数据仓库层次的划分不是固定不变的，可以根据实际需求进行适当裁剪或者是添加。如果业务相对简单和独立，可以将DWD、DWS进行合并。



第4节 数据仓库模型

4.1 事实表与维度表

在数据仓库中，保存度量值的详细值或事实的表称为事实表。

事实数据表通常包含大量的行。事实数据表的主要特点是包含数字数据（事实），并且这些数字信息可以汇总，以提供有关单位作为历史的数据。事实表的粒度决定了数据仓库中数据的详细程度。

常见事实表：订单事实表

事实表的特点：表多（各种各样的事实表）；数据量大

事实表根据数据的粒度可以分为：事务事实表、周期快照事实表、累计快照事实表

维度表（维表）可以看作是用来分析数据的角度，纬度表中包含事实数据表中事实记录的特性。有些特性提供描述性信息，有些特性指定如何汇总事实数据表数据，以便为分析者提供有用的信息。

常见维度表：时间维度、地域维度、商品维度

小结：

- 事实表是关注的内容（如：销售额、销售量）

- 维表是观察事务的角度

4.2 事实表分类

1、事务事实表

事务事实表记录的事务层面的事实，保存的是最原子的数据，也称“原子事实表”。事务事实表中的数据在事务事件发生后产生，数据的粒度通常是每个事务一条记录。一旦事务被提交，事实表数据被插入，数据就不再进行更改，其更新方式为增量更新。

事务事实表的日期维度记录的是事务发生的日期，它记录的事实是事务活动的内容。用户可以通过事务事实表对事务行为进行特别详细的分析。

如：订单表

通过事务事实表，还可以建立聚集事实表，为用户提供高性能的分析。

2、周期快照事实表

周期快照事实表以具有规律性的、可预见的时间间隔来记录事实，时间间隔如每天、每月、每年等等。典型的例子如销售日快照表、库存日快照表等。它统计的是间隔周期内的度量统计，如历史至今、自然年至今、季度至今等等。

周期快照事实表的粒度是每个时间段一条记录，通常比事务事实表的粒度要粗，是在事务事实表之上建立的聚集表。周期快照事实表的维度个数比事务事实表要少，但是记录的事实要比事务事实表多。

如：商家日销售表（无论当天是否有销售发生，都记录一行）日期、商家名称、销售量、销售额

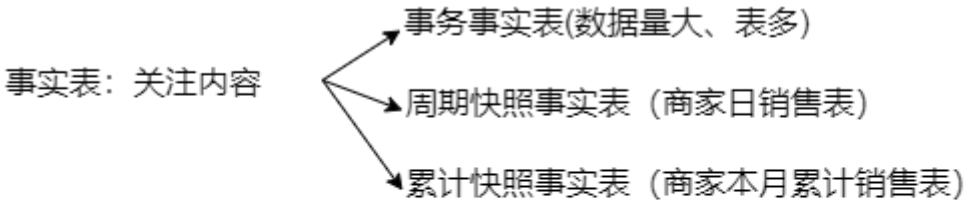
3、累积快照事实表

累积快照事实表和周期快照事实表有些相似之处，它们存储的都是事务数据的快照信息。但是它们之间也有着不同，周期快照事实表记录的确定的周期的数据，而累积快照事实表记录的不确定的周期的数据。

累积快照事实表代表的是完全覆盖一个事务或产品的生命周期的时间跨度，它通常具有多个日期字段，用来记录整个生命周期中的关键时间点。另外，它还会有一个用于指示最后更新日期的附加日期字段。由于事实表中许多日期在首次加载时是不知道的，所以必须使用代理关键字来处理未定义的日期，而且这类事实表在数据加载完后，是可以对它进行更新的，来补充随后知道的日期信息。

如：订货日期、预定交货日期、实际发货日期、实际交货日期、数量、金额、运费

如：商家本周、本月、本年累计销售表



维度表：观察事物的角度

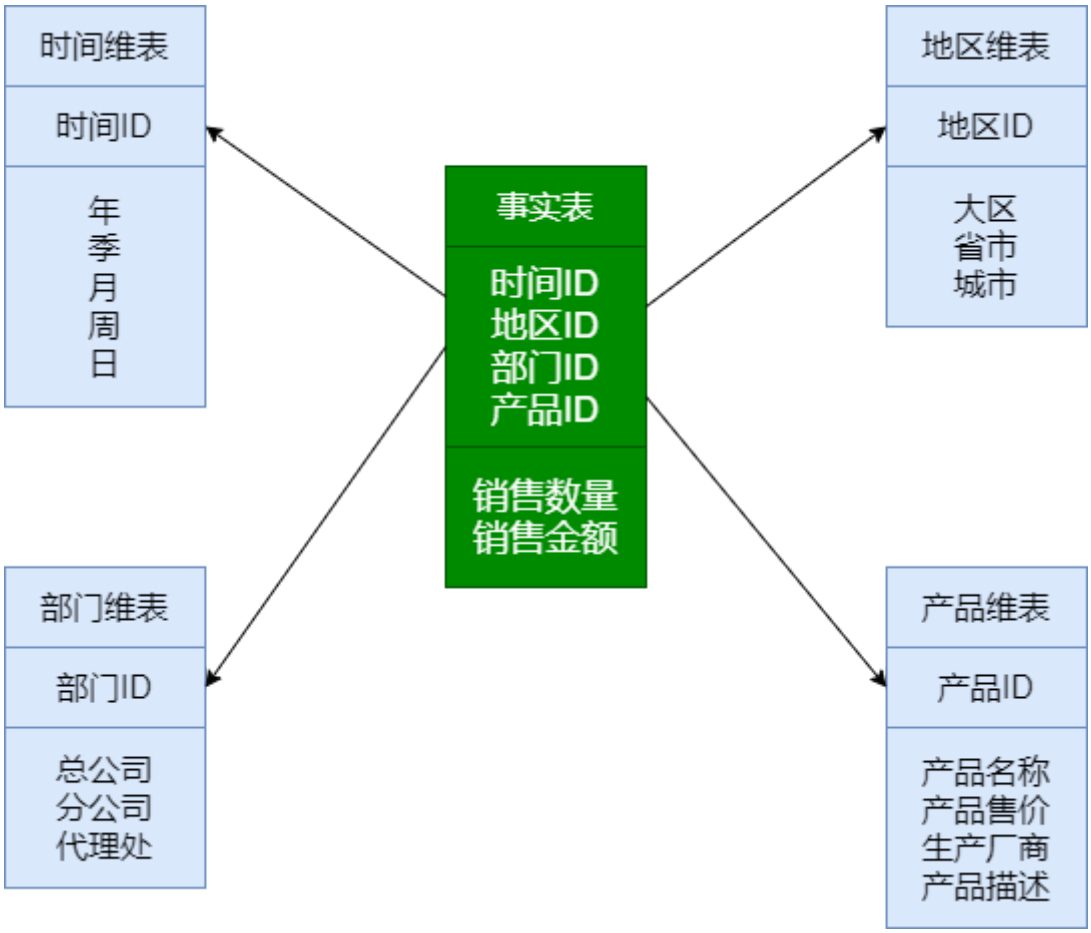
4.3 星型模型

星型模是一种多维的数据关系，它由一个事实表和一组维表组成；

事实表在中心，周围围绕地连接着维表；

事实表中包含了大量数据，没有数据冗余；

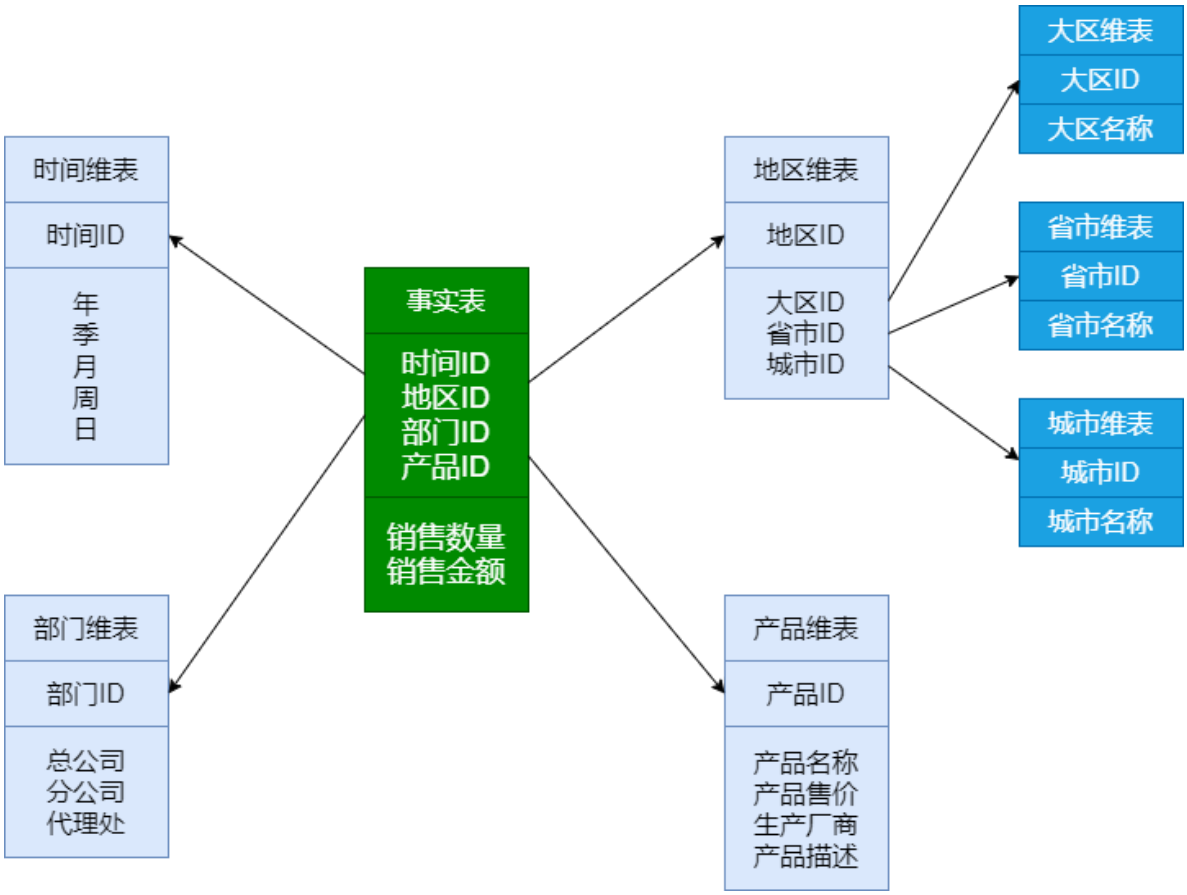
维表是逆规范化的，包含一定的数据冗余；



4.4 雪花模型

雪花模式是星型模型的变种，维表是规范化的，模型类似雪花的形状；

特点：雪花型结构去除了数据冗余。



星型模型存在数据冗余，所以在查询统计时只需要做少量的表连接，查询效率高；

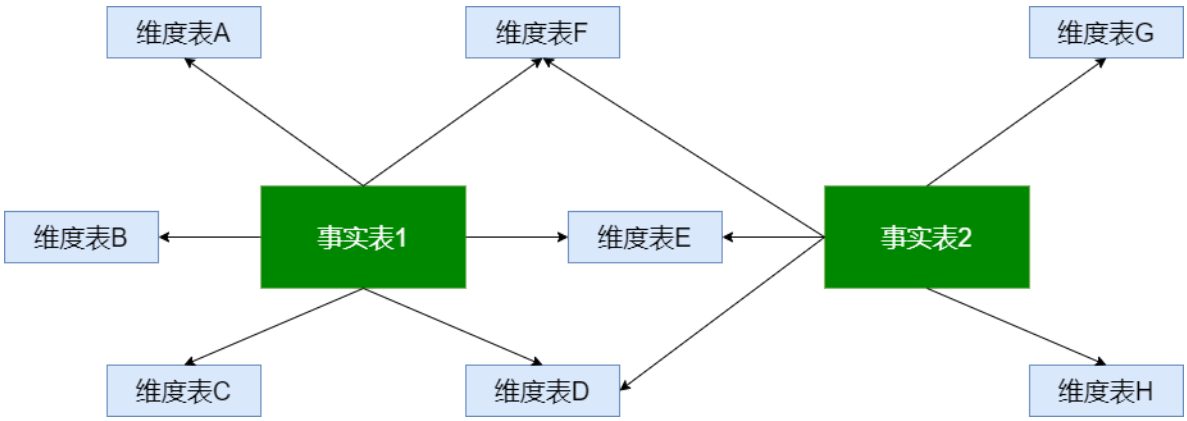
星型模型不考虑维表正规化的因素，设计、实现容易；

在数据冗余可接受的情况下，实际上使用星型模型比较多；

4.5 事实星座

数据仓库由多个主题构成，包含多个事实表，而维表是公共的，可以共享，这种模式可以看做星型模式的汇集，因而称作星系模式或者事实星座模式。

特点：公用维表



第5节 元数据

元数据（Metadata）是关于数据的数据。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。元数据就相当于所有数据的地图，有了这张地图就能知道数据仓库中：

- 有哪些数据
- 数据的分布情况
- 数据类型
- 数据之间有什么关系
- 哪些数据经常被使用，哪些数据很少有人光顾

在大数据平台中，元数据贯穿大数据平台数据流动的全过程，主要包括数据源元数据、数据加工处理过程元数据、数据主题库专题库元数据、服务层元数据、应用层元数据等。

业内通常把元数据分为以下类型：

- 技术元数据：库表结构、数据模型、ETL程序、SQL程序等
- 业务元数据：业务指标、业务代码、业务术语等
- 管理元数据：数据所有者、数据质量、数据安全等

第二部分 电商离线数仓设计

第1节 需求分析

近年来，中国的电子商务快速发展，交易额连创新高，电子商务在各领域的应用不断拓展和深化、相关服务业蓬勃发展、支撑体系不断健全完善、创新的动力和能力不断增强。电子商务正在与实体经济深度融合，进入规模性发展阶段，对经济社会生活的影响不断增大，正成为我国经济发展的新引擎。

中国电子商务研究中心数据显示，截止到 2012 年底，中国电子商务市场交易规模达7.85万亿人民币，同比增长 30.83%。其中，B2B 电子商务交易额 达 6.25 万亿，同比增长 27%。而 2011 年全年，中国电子商务市场交易额达 6 万亿人民币，同比增长 33%，占 GDP 比重上升到 13%；2012 年，电子商务占 GDP 的比重已经高达 15%。

电商行业技术特点

- 技术新
- 技术范围广
- 分布式
- 高并发、集群、负载均衡
- 海量数据
- 业务复杂
- 系统安全

电商业务简介

类似X东商城、X猫商城。电商网站采用商家入驻的模式，商家入驻平台提交申请，有平台进行资质审核，审核通过后，商家拥有独立的管理后台录入商品信息。商品经过平台审核后即可发布。网上商城主要分为：

- 网站前台。网站首页、商家首页、商品详细页、搜索页、会员中心、订单与支付相关页面、秒杀频道等；
- 运营商后台。运营人员的管理平台， 主要功能包括：商家审核、品牌管理、规格管理、模板管理、商品分类管理、商品审核、广告类型管理、广告管理、订单查询、商家结算等；
- 商家管理后台。入驻的商家进行管理的平台，主要功能包括：商品管理、订单查询统计、资金结算等功能；

数据仓库项目主要分析以下数据：

- 日志数据：启动日志、点击日志（广告点击日志）
- 业务数据库的交易数据：用户下单、提交订单、支付、退款等核心交易数据的分析

数据仓库项目分析任务：

- 会员活跃度分析主题

  每日新增会员数；每日、周、月活跃会员数；留存会员数、留存会员率

- 广告业务分析主题

  广告点击次数、广告点击购买率、广告曝光次数

- 核心交易分析主题

  订单数、成交商品数、支付金额

第2节 数据埋点

数据埋点，将用户的浏览、点击事件采集上报的一套数据采集的方法。

通过这套方法，能够记录到用户在App、网页的一些行为，用来跟踪应用使用的状况，后续用来进一步优化产品或是提供运营的数据支撑，包括访问数、访客数、停留时长、浏览数、跳出率。这样的信息收集可以大致分为两种：页面统计、统计操作行为。

在企业经营中，数据分析辅助决策是非常重要的一环，而埋点采集用户行为数据的工作则是基础中的基础。如果没有用户行为数据，经营分析将无从说起。埋点为数据分析提供基础数据，埋点工作流程可分为：

- 根据埋点需求完成开发（前端开发工程师 js）
- App或网页采用用户数据
- 数据上报服务器
- 数据的清洗、加工、存储（大数据工程师）
- 进行数据分析等到相应的指标（大数据工程师）

在以上过程中，涉及的相关人员可分以下几类：

- 埋点需求：数据产品经理，负责撰写需求文档，规定哪些区域、用户操作需要埋点

- 埋点采集：前端工程师，负责通过一套前端 js 代码对用户的请求事件上送至服务器
- 数据清洗、加工及存储：对埋点中数据缺失、误报等情况需要进行清洗，并通过一定的计算加工，输出业务分析所需要的结构化数据，最后将数据存储在数据仓库中
- 数据分析：在数据仓库中对数据进行整理，成业务关注的指标
- 前端展示：Java 开发

主流的埋点实现方法如下，主要区别是前端开发的工作量：

- 手动埋点：开发需要手动写代码实现埋点，比如页面ID、区域ID、按钮ID、按钮位置、事件类型（曝光、浏览、点击）等，一般需要公司自主研发的一套埋点框架
  - 优点：埋点数据更加精准
  - 缺点：工作量大，容易出错
- 无痕埋点：不用开发写代码实现的，自动将设备号、浏览器型号、设备类型等数据采集。主要使用第三方统计工具，如友盟、百度移动、魔方等
  - 优点：简单便捷
  - 缺点：埋点数据统一，不够个性化和精准

启动日志：


```
{
    "app_active": {
        "name": "app_active",
        "json": {
            "entry": "3",
            "action": "0",
            "error_code": "0"
        },
        "time": 1593553936325
    },
    "attr": {
        "area": "葫芦岛",
        "uid": "2F10092A192",
        "app_v": "1.1.12",
        "event_type": "common",
        "device_id": "1FB872-9A100192",
        "os_type": "0.7.0",
        "channel": "MA",
        "language": "chinese",
```

```
        "brand": "Huawei-4"
    }
}
```

事件日志（广告点击、收藏、点赞、消息通知、商品评论、商品详情页加载等事件）：

```
{
    "lagou_event": [{
        "name": "goods_detail_loading",
        "json": {
            "entry": "2",
            "goodsid": "0",
            "loading_time": "71",
            "action": "3",
            "staytime": "119",
            "showtype": "5"
        },
        "time": 1594804466872
    }, {
        "name": "notification",
        "json": {
            "action": "3",
            "type": "4"
        },
        "time": 1594775458428
    }, {
        "name": "ad",
        "json": {
            "duration": "19",
            "ad_action": "0",
            "shop_id": "46",
            "event_type": "ad",
            "ad_type": "2",
            "show_style": "1",
            "product_id": "9022",
            "place": "placeindex_right",
            "sort": "4"
        },
        "time": 1594779518872
    }, {
        "name": "favorites",
        "json": {
```

```
            "course_id": 2,
            "id": 0,
            "userid": 0
        },
        "time": 1594812897271
    }],
    "attr": {
        "area": "清远",
        "uid": "2F10092A77",
        "app_v": "1.1.7",
        "event_type": "common",
        "device_id": "1FB872-9A10077",
        "os_type": "0.8.4",
        "channel": "PQ",
        "language": "chinese",
        "brand": "iphone-2"
    }
}
```


第3节 数据指标体系

指标：对数据的统计值。如：会员数、活跃会员数、会员留存数；广告点击量；订单金额、订单数都是指标；

指标体系：将各种指标系统的组织起来，按照业务模型、标准对指标进行分类和分层；

没有数据指标体系的团队内数据需求经常表现为需求膨胀以及非常多的需求变更。每个人都有看数据的视角和诉求，然后以非专业的方式创造维度/指标的数据口径。数据分析人员被海量的数据需求缠住，很难抽离出业务规则设计好的解决方案，最终滚雪球似的搭建难以维护的数据仓库。

- 建立指标体系实际上是与需求方达成一致。能有效遏制不靠谱的需求，让需求变得有条例和体系化；
- 指标体系是知道数据仓库建设的基石。稳定而且体系化的需求，有利于数据仓库方案的优化，和效率提升；

由产品经理牵头、与业务、IT方协助，制定的一套能从维度反应业务状况的一套待实施框架。在建立指标体系时，要注重三个选取原则：准确、可解释、结构性。

- 准确：核心数据一定要理解到位和准确，不能选错；

- 可解释：所有指标都要配上明确、详细的业务解释。如日活的定义是什么，是使用了App、还是在App中停留了一段时间、或是收藏或购买购买了商品；
- 结构性：能够充分对业务进行解读。如新增用户只是一个大数，还需要知道每个渠道的新增用户，每个渠道的新增转化率，每个渠道的新增用户价值等。

在建立指标体系之前，先了解一下指标的构成，在工作过程中遇见的指标多为派生性指标。指标的构成如下所示：

- 基础指标 + [修饰词] + 时间段
- 修饰词是可选的；基础指标和时间段是必须的
- 基础指标是不可拆分的指标，如：交易额、支付金额、下单数
- 修饰词多是某种场景的表现，如：通过搜索带来的交易等
- 时间段即为一个时间周期，如：双十一期间，618活动期间等

三者叠加在一起就形成业务上常用的指标（这些指标也是派生指标），如：双11这一天通过搜索带来的交易额、双11这一天的交易额。同样，像此类日活、月活、次日留存、日转化率等都属于派生指标。

在筛选完合理指标后，就要着手建立对应的指标体系。主要分为四个步骤：理清业务阶段和需求、确定核心指标、对指标进行维度的拆解、指标的落地；

1、厘清业务阶段及需求

企业的发展往往分为三个阶段：创业期、上升期、成熟发展期，不同的阶段关注的核心指标也是不同的。

- 业务前期，最关注用户量，此时的指标体系应该紧密围绕用户量的提升来做各种维度的拆解
- 业务中期，除了关注用户量的走势大小，更加重要的是优化当前的用户量结构，比如看用户留存，如果留存偏低，那就需要进一步分析查找原因
- 成熟发展期，更多关注的就是产品变现能力和市场份额，要关注收入指标、各种商业化模式的的收入，同时做好市场份额和竞品的监控，以防止新起势力抢占份额等

2、确定核心指标

这个阶段最重要的是找到正确的核心指标。

例：某款产品的日活口径是打开APP，而且日活量不小，而且稳定上升。然而分析时发现，打开APP的用户中，5秒跳出率高达25%，这是非常不健康的，那么当前的核心指标日活实际上已经有了问题，更加好的核心指标应该是停留时长大于5秒的用户数。

每个APP的核心指标都不太一样，一定要花时间去考虑这件事。就像XX头条APP，它的日活和留存指标一定非常高，但仅关注这种指标肯定是不对的，它的真正核心指标绝对不是单纯的日活和留存。

3、核心指标维度拆解

核心指标的波动必然是某种维度的波动引起，要监控核心指标，本质上还是要监控维度核心指标。

在分析"进入APP用户数"指标时，要关注渠道转化率，分析用户从哪里来；同时用户通过哪种方式打开的，如通过点击桌面图标、点击通知栏、点击Push等；

在分析"停留时长大于5秒占比"指标时，要重点关注停留时长的分布，停留1秒 -- 5秒的用户各有多少，具体分布情况；停留大于5秒的用户特征和行为特性是怎么样的情况；停留小于5秒的用户特征等；

电商平台注重交易额，在真正达成交易之前，用户要打开APP、选择商品、确认订单、支付订单等整个交流漏斗模型。每一个环节的关键指标都可以通过公式的形式进行拆解，在根据拆解公式逐个分析对应的影响因素。

4、指标宣贯、存档、落地

在完成整个指标体系搭建后，要告知所有相关业务人员。一方面为下一步工作做铺垫，另一方面是为了让所有相关人员知晓已完成，以防甩锅；

对指标口径的业务逻辑进行详细的描述并存档，只有明确、清晰的定义才能明白指标的具体含义；

就是建立核心指标的相关报表，实际工作中，报表会在埋点前建好的，这样的话一旦版本上线就能立刻看到数据，而且也比较容易发现问题。

整个指标体系的搭建主要是由产品经理主导完成的，业务人员需要配合产品经理选择并确认指标，这也是在建立之初最重要的一点。

第4节 总体架构设计

4.1、技术方案选型

```
框架选型
软件选型
服务器选型
集群规模的估算
```

框架选型

Apache / 第三方发行版（CDH / HDP / Fusion Insight）

Apache社区版本优点：

- 完全开源免费
- 社区活跃
- 文档、资料详实

缺点：

- 复杂的版本管理
- 复杂的集群安装
- 复杂的集群运维
- 复杂的生态环境

第三方发行版本（CDH / HDP / Fusion Insight）

Hadoop遵从Apache开源协议，用户可以免费地任意使用和修改Hadoop。正因如此，市面上有很多厂家在Apache Hadoop的基础上开发自己的产品。如Cloudera的CDH，Hortonworks的HDP，华为的Fusion Insight等。这些产品的优点是：

- 主要功能与社区版一致
- 版本管理清晰。比如Cloudera，CDH1，CDH2，CDH3，CDH4等，后面加上补丁版本，如CDH4.1.0 patch level 923.142
- 比 Apache Hadoop 在兼容性、安全性、稳定性上有增强。第三方发行版通常都经过了大量的测试验证，有众多部署实例，大量的运用到各种生产环境
- 版本更新快。如CDH每个季度会有一个update，每一年会有一个release
- 基于稳定版本Apache Hadoop，并应用了最新Bug修复或Feature的patch
- 提供了部署、安装、配置工具，大大提高了集群部署的效率，可以在几个小时内部署好集群
- 运维简单。提供了管理、监控、诊断、配置修改的工具，管理配置方便，定位问题快速、准确，使运维工作简单，有效

CDH：最成型的发行版本，拥有最多的部署案例。提供强大的部署、管理和监控工具。国内使用最多的版本；拥有强大的社区支持，当遇到问题时，能够通过社区、论坛等网络资源快速获取解决方法；

HDP：100%开源，可以进行二次开发，但没有CDH稳定。国内使用相对较少；

Fusion Insight：华为基于hadoop2.7.2版开发的，坚持分层，解耦，开放的原则，得益于高可靠性，在全国各地政府、运营商、金融系统有较多案例。

软件选型

数据采集：**DataX、Flume**、Sqoop、Logstash、Kafka

数据存储：**HDFS**、HBase

数据计算：**Hive、MapReduce、Tez**、Spark、Flink

调度系统：**Airflow**、azkaban、Oozie

元数据管理：**Atlas**

数据质量管理：**Griffin**

即席查询：**Impala**、Kylin、ClickHouse、Presto、Druid

其他：**MySQL**

框架、软件尽量不要选择最新的版本，选择半年前左右稳定的版本。

产品	版本
Hadoop	2.9.2
Hive	2.3.7
Flume	1.9
DataX	3.0
Airflow	1.10
Atlas	1.2.0
Griffin	0.4.0
Impala	impala-2.3.0-cdh5.5.0
MySQL	5.7

服务器选型

选择物理机还是云主机

机器成本考虑：物理机的价格 > 云主机的价格

运维成本考虑：物理机需要有专业的运维人员；云主机的运维工作由供应商完成，运维相对容易，成本相对较低；

集群规模规划

如何确认集群规模（假设：每台服务器20T硬盘，128G内存）

可以从计算能力(CPU、 内存)、**存储量**等方面着手考虑集群规模。

假设：

1、每天的日活用户500万，平均每人每天有100条日志信息

2、每条日志大小1K左右

3、不考虑历史数据，半年集群不扩容

4、数据3个副本

5、离线数据仓库应用

需要多大集群规模?

要分析的数据有两部分：日志数据+业务数据

每天日志数据量：500W * 100 * 1K / 1024 / 1024 = 500G

半年需要的存储量：500G * 3 * 180 / 1024 = 260T

通常要给磁盘预留20-30%的空间（这里取25%）： 260 * 1.25 = 325T

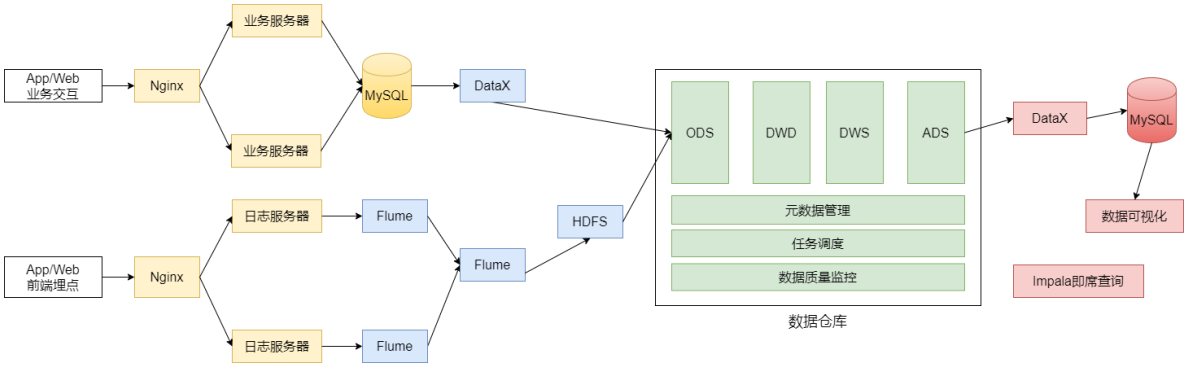
数据仓库应用有1-2倍的数据膨胀（这里取1.5）：500T

需要大约25个节点

其他未考虑因素：数据压缩、业务数据

以上估算的生产环境。实际上除了生产环境以外，还需要开发测试环境，这也需要一定数据的机器。

4.2、系统逻辑架构



4.3、开发物理环境

```
5台物理机；500G数据盘；32G内存；8个core
```

```
[root@hadoop5 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        20G   14G  6.5G  68% /
devtmpfs         16G     0   16G   0% /dev
tmpfs            16G     0   16G   0% /dev/shm
tmpfs            16G  218M   16G   2% /run
tmpfs            16G     0   16G   0% /sys/fs/cgroup
/dev/vdb        500G   96G  405G  20% /data
tmpfs           3.2G     0  3.2G   0% /run/user/0
tmpfs           3.2G     0  3.2G   0% /run/user/1000
[root@hadoop5 ~]# grep MemTotal /proc/meminfo
MemTotal:       32740424 kB
[root@hadoop5 ~]# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             8
NUMA node(s):          1
```

系统盘20G

数据盘500G

内存32G

1物理CPU，8个核

	hadoop1	hadoop2	hadoop3	hadoop4	hadoop5
NameNode	√				
SecondaryNameNode		√			
DataNode	√	√	√	√	√
ResourceManager	√				
DataManager	√	√	√	√	√
Hive	√	√			√
HiveServer2					√
Flume		√			
DataX		√			
Airflow		√			
Atlas		√			
Griffin		√			
Impala	√	√	√	√	√
MySQL		√			

关于数据集的说明：

1、在开发过程中使用小规模数据集

2、模块测试使用真实的数据集（数据量大）

3、在做项目期间根据自己实际情况使用不同的数据量（建议使用小规模的数据集）

4.4、数据仓库命名规范

```
1 数据库命名
    命名规则：数仓对应分层
    命名示例：ods / dwd / dws/ dim / temp / ads

2 数仓各层对应数据库
    ods层 -> ods_{业务线|业务项目}
    dw层  -> dwd_{业务线|业务项目} + dws_{业务线|业务项目}
    dim层 -> dim_维表
    ads层 -> ads_{业务线|业务项目} (统计指标等)
    临时数据 -> temp_{业务线|业务项目}
备注：本项目未采用

3 表命名（数据库表命名规则）
    * ODS层：
    命名规则：ods_{业务线|业务项目}_[数据来源类型]_{业务}

    * DWD层：
    命名规则：dwd_{业务线|业务项目}_{主题域}_{子业务}

    * DWS层：
    命名规则：dws_{业务线|业务项目}_{主题域}_{汇总相关粒度}_{汇总时间周期}

    * ADS层：
    命名规则：ads_{业务线|业务项目}_{统计业务}_{报表form|热门排序topN}

    * DIM层：
    命名规则：dim_{业务线|业务项目|pub公共}_{维度}
```

创建数据库：

```
create database if not exists ods;
create database if not exists dwd;
create database if not exists dws;
create database if not exists ads;
create database if not exists dim;
create database if not exists tmp;
```

第三部分 电商分析之--会员活跃度

第1节 需求分析

会员数据是后期营销的很重要的数据。网店会专门针对会员进行一系列营销活动。

电商会员一般门槛较低，注册网站即可加入。有些电商平台的高级会员具有时效性，需要购买VIP会员卡或一年内消费额达到多少才能成为高级会员。

计算指标：

新增会员：每日新增会员数

活跃会员：每日，每周，每月的活跃会员数

会员留存：1日，2日，3日会员留存数、1日，2日，3日会员留存率

指标口径业务逻辑：

会员：以设备为判断标准，每个独立设备认为是一个会员。Android系统通常根据IMEI号，IOS系统通常根据OpenUDID 来标识一个独立会员，每部移动设备是一个会员；

活跃会员：打开应用的会员即为活跃会员，暂不考虑用户的实际使用情况。一台设备每天多次打开计算为一个活跃会员。在自然周内启动过应用的会员为周活跃会员，同理还有月活跃会员；

会员活跃率：一天内活跃会员数与总会员数的比率是日活跃率；还有周活跃率（自然周）、月活跃率（自然月）；

新增会员：第一次使用应用的用户，定义为新增会员；卸载再次安装的设备，不会被算作一次新增。新增用户包括日新增会员、周（自然周）新增会员、月（自然月）新增会员；

留存会员与留存率：某段时间的新增会员，经过一段时间后，仍继续使用应用认为是留存会员；这部分会员占当时新增会员的比例为留存率。

已知条件：

1、明确了需求

2、输入：启动日志（OK）、事件日志

3、输出：新增会员、活跃会员、留存会员

4、日志文件、ODS、DWD、DWS、ADS（输出）

下一步作什么?

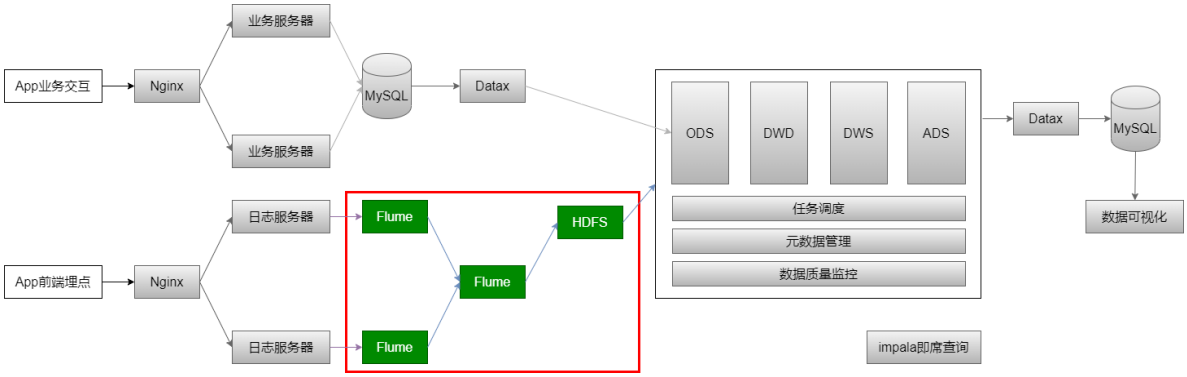
数据采集：日志文件 => Flume => HDFS => ODS

第2节 日志数据采集

原始日志数据（一条启动日志）

```
2020-07-30 14:18:47.339 [main] INFO  com.lagou.ecommerce.AppStart
- {"app_active":{"name":"app_active","json":
{"entry":"1","action":"1","error_code":"0"},"time":1596111888529}
,"attr":{"area":"泰
安","uid":"2F10092A9","app_v":"1.1.13","event_type":"common","dev
ice_id":"1FB872-
9A1009","os_type":"4.7.3","channel":"DK","language":"chinese","br
and":"iphone-9"}}
```

数据采集的流程：



选择Flume作为采集日志数据的工具：

- Flume 1.6
  - 无论是Spooling Directory Source、Exec Source均不能很好的满足动态实时收集的需求
- Flume 1.8+

- 提供了一个非常好用的 Taildir Source
- 使用该source，可以监控多个目录，对目录中新写入的数据进行实时采集

2.1、taildir source配置

taildir Source的特点：

- 使用正则表达式匹配目录中的文件名
- 监控的文件中，一旦有数据写入，Flume就会将信息写入到指定的Sink
- 高可靠，不会丢失数据
- 不会对跟踪文件有任何处理，不会重命名也不会删除
- 不支持Windows，不能读二进制文件。支持按行读取文本文件

taildir source配置

```
a1.sources.r1.type = TAILDIR
a1.sources.r1.positionFile =
/data/lagoudw/conf/startlog_position.json
a1.sources.r1.filegroups = f1
a1.sources.r1.filegroups.f1 = /data/lagoudw/logs/start/.*log
```

- **positionFile**

  配置检查点文件的路径，检查点文件会以 json 格式保存已经读取文件的位置，解决断点续传的问题

- **filegroups**

  指定filegroups，可以有多个，以空格分隔（taildir source可同时监控多个目录中的文件）

- **filegroups.**

  配置每个filegroup的文件绝对路径，文件名可以用正则表达式匹配

2.2、hdfs sink配置

```
a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path = /user/data/logs/start/%Y-%m-%d/
a1.sinks.k1.hdfs.filePrefix = startlog.

# 配置文件滚动方式（文件大小32M）
a1.sinks.k1.hdfs.rollSize = 33554432
```

```
a1.sinks.k1.hdfs.rollCount = 0
a1.sinks.k1.hdfs.rollInterval = 0
a1.sinks.k1.hdfs.idleTimeout = 0
a1.sinks.k1.hdfs.minBlockReplicas = 1

# 向hdfs上刷新的event的个数
a1.sinks.k1.hdfs.batchSize = 100

# 使用本地时间
a1.sinks.k1.hdfs.useLocalTimeStamp = true
```

HDFS Sink 都会采用滚动生成文件的方式，滚动生成文件的策略有：

- 基于时间。hdfs.rollInterval 30秒
- 基于文件大小。hdfs.rollSize 1024字节
- 基于event数量。hdfs.rollCount 10个event
- 基于文件空闲时间。hdfs.idleTimeout 0
- 0，禁用
- minBlockReplicas。默认值与 hdfs 副本数一致。设为1是为了让 Flume 感知不到hdfs的块复制，此时其他的滚动方式配置（时间间隔、文件大小、events数量）才不会受影响

2.3、Agent的配置

```
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# taildir source
a1.sources.r1.type = TAILDIR
a1.sources.r1.positionFile =
/data/lagoudw/conf/startlog_position.json
a1.sources.r1.filegroups = f1
a1.sources.r1.filegroups.f1 = /data/lagoudw/logs/start/.*log

# memorychannel
a1.channels.c1.type = memory
a1.channels.c1.capacity = 100000
a1.channels.c1.transactionCapacity = 2000

# hdfs sink
a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path = /user/data/logs/start/%Y-%m-%d/
```

```
a1.sinks.k1.hdfs.filePrefix = startlog.

# 配置文件滚动方式（文件大小32M）
a1.sinks.k1.hdfs.rollSize = 33554432
a1.sinks.k1.hdfs.rollCount = 0
a1.sinks.k1.hdfs.rollInterval = 0
a1.sinks.k1.hdfs.idleTimeout = 0
a1.sinks.k1.hdfs.minBlockReplicas = 1

# 向hdfs上刷新的event的个数
a1.sinks.k1.hdfs.batchSize = 1000

# 使用本地时间
a1.sinks.k1.hdfs.useLocalTimeStamp = true

# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

/data/lagoudw/conf/flume-log2hdfs.conf

2.4、Flume的优化配置

1、启动agent

```
flume-ng agent --conf-file /data/lagoudw/conf/flume-
log2hdfs1.conf -name a1 -Dflum
e.roog.logger=INFO,console
```

2、向 /data/lagoudw/logs/ 目录中放入日志文件，报错：

java.lang.OutOfMemoryError: GC overhead limit exceeded

```
[root@hadoop2 conf]# ps -ef | grep flume
root     53828 46007  1 17:01 pts/2    00:00:14 /opt/apps/jdk1.8.0_231/bin/java -Xmx20m -Dflume.root.log
ger=INFO,console -cp /opt/apps/flume-1.9/lib/*:/opt/apps/hadoop-2.9.2/etc/hadoop:/opt/apps/hadoop-2.9.2/
share/hadoop/common/lib/*:/opt/apps/hadoop-2.9.2/share/hadoop/common/*:/opt/apps/hadoop-2.9.2/share/hado
op/hdfs:/opt/apps/hadoop-2.9.2/share/hadoop/hdfs/lib/*:/opt/apps/hadoop-2.9.2/share/hadoop/hdfs/*:/opt/a
pps/hadoop-2.9.2/share/hadoop/yarn:/opt/apps/hadoop-2.9.2/share/hadoop/yarn/lib/*:/opt/apps/hadoop-2.9.2
/share/hadoop/yarn/*:/opt/apps/hadoop-2.9.2/share/hadoop/mapreduce/lib/*:/opt/apps/hadoop-2.9.2/share/ha
doop/mapreduce/*:/opt/apps/hadoop-2.9.2/contrib/capacity-scheduler/*.jar:/opt/apps/hive-2.3.7/lib/* -Dja
va.library.path=:/opt/apps/hadoop-2.9.2/lib/native org.apache.flume.node.Application --conf-file conf/fl
ume-log2hdfs.conf -name a1
```

缺省情况下 Flume jvm堆最大分配20m，这个值太小，需要调整。

3、解决方案：在 $FLUME_HOME/conf/flume-env.sh 中增加以下内容

```
export JAVA_OPTS="-Xms4000m -Xmx4000m -
Dcom.sun.management.jmxremote"

# 要想使配置文件生效，还要在命令行中指定配置文件目录
flume-ng agent --conf /opt/apps/flume-1.9/conf --conf-file
/data/lagoudw/conf/flume-log2hdfs1.conf -name a1 -
Dflume.roog.logger=INFO,console
```

Flume内存参数设置及优化：

- 根据日志数据量的大小，Jvm堆一般要设置为4G或更高
- -Xms -Xmx 最好设置一致，减少内存抖动带来的性能影响

存在的问题：Flume放数据时，使用本地时间；不理会日志的时间戳

2.5、自定义拦截器

前面 Flume Agent 的配置使用了本地时间，可能导致数据存放的路径不正确。

要解决以上问题需要使用自定义拦截器。

agent用于测试自定义拦截器。netcat source =>logger sink

```
# a1是agent的名称。source、channel、sink的名称分别为：r1 c1 k1
a1.sources = r1
a1.channels = c1
a1.sinks = k1

# source
a1.sources.r1.type = netcat
a1.sources.r1.bind = hadoop2
a1.sources.r1.port = 9999
a1.sources.r1.interceptors = i1
a1.sources.r1.interceptors.i1.type =
cn.lagou.dw.flume.interceptor.CustomerInterceptor$Builder

# channel
a1.channels.c1.type = memory
a1.channels.c1.capacity = 10000
a1.channels.c1.transactionCapacity = 100
```

```
# sink
a1.sinks.k1.type = logger

# source、channel、sink之间的关系
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

自定义拦截器的原理：

1、自定义拦截器要集成Flume 的 Interceptor

2、Event 分为header 和 body（接收的字符串）

3、获取header和body

4、从body中获取"time":1596382570539，并将时间戳转换为字符串 "yyyy-MM-dd"

5、将转换后的字符串放置header中

自定义拦截器的实现：

1、获取 event 的 header

2、获取 event 的 body

3、解析body获取json串

4、解析json串获取时间戳

5、将时间戳转换为字符串 "yyyy-MM-dd"

6、将转换后的字符串放置header中

7、返回event

```
    <properties>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.apache.flume</groupId>
            <artifactId>flume-ng-core</artifactId>
```

```
            <version>1.9.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.1.23</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.3.2</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <descriptorRefs>
                        <descriptorRef>jar-with-
dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
                <executions>
                    <execution>
                        <id>make-assembly</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

```
package cn.lagou.dw.flume.interceptor;
```

```
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
import com.google.common.base.Strings;
import org.apache.commons.compress.utils.Charsets;
import org.apache.flume.Context;
import org.apache.flume.Event;
import org.apache.flume.interceptor.Interceptor;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class CustomerInterceptor implements Interceptor {
    private static DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd");

    @Override
    public void initialize() {

    }

    @Override
    public Event intercept(Event event) {
        // 获得body的内容
        String eventBody = new String(event.getBody(),
Charsets.UTF_8);
        // 获取header的内容
        Map<String, String> headerMap = event.getHeaders();

        final String[] bodyArr = eventBody.split("\\s+");

        try {
            String jsonStr = bodyArr[6];
            if (Strings.isNullOrEmpty(jsonStr)) {
                return null;
            }

            // 将 string 转成 json 对象
            JSONObject jsonObject = JSON.parseObject(jsonStr);
            String timestampStr = jsonObject.getString("time");
```

```
            // 将 timestamp 转为 时间日期类型(格式：yyyy-MM-dd)
            long timeStamp = Long.parseLong(timestampStr);
            String date =
formatter.format(LocalDateTime.ofInstant(Instant.ofEpochMilli(tim
eStamp), ZoneId.systemDefault()));

            headerMap.put("logtime", date);
            event.setHeaders(headerMap);
        } catch (Exception e) {
            headerMap.put("logtime", "unknown");
            event.setHeaders(headerMap);
        }

        return event;
    }

    @Override
    public List<Event> intercept(List<Event> events) {
        List<Event> out = new ArrayList<>();
        for (Event event : events) {
            Event outEvent = intercept(event);
            if (outEvent != null) {
                out.add(outEvent);
            }
        }
        return out;
    }

    @Override
    public void close() {

    }

    public static class Builder implements Interceptor.Builder {
        @Override
        public Interceptor build() {
            return new CustomerInterceptor();
        }

        @Override
        public void configure(Context context) {
        }
    }
}
```

将程序打包，放在 flume/lib目录下；

启动Agent测试

```
flume-ng agent --conf /opt/apps/flume-1.9/conf --conf-file
conf/flume-log2hdfs1.conf -name a1 -
Dflume.root.logger=INFO,console
```

2.6、采集启动日志(使用自定义拦截器)

1、定义配置文件

```
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# taildir source
a1.sources.r1.type = TAILDIR
a1.sources.r1.positionFile =
/data/lagoudw/conf/startlog_position.json
a1.sources.r1.filegroups = f1
a1.sources.r1.filegroups.f1 = /data/lagoudw/logs/start/.*log
a1.sources.r1.interceptors = i1
a1.sources.r1.interceptors.i1.type =
cn.lagou.dw.flume.interceptor.CustomerInterceptor$Builder

# memorychannel
a1.channels.c1.type = memory
a1.channels.c1.capacity = 100000
a1.channels.c1.transactionCapacity = 2000

# hdfs sink
a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path = /user/data/logs/start/dt=%{logtime}/
a1.sinks.k1.hdfs.filePrefix = startlog.

# 配置文件滚动方式（文件大小32M）
a1.sinks.k1.hdfs.rollSize = 33554432
a1.sinks.k1.hdfs.rollCount = 0
a1.sinks.k1.hdfs.rollInterval = 0
a1.sinks.k1.hdfs.idleTimeout = 0
a1.sinks.k1.hdfs.minBlockReplicas = 1
```

```
# 向hdfs上刷新的event的个数
a1.sinks.k1.hdfs.batchSize = 1000

# 使用本地时间
# a1.sinks.k1.hdfs.useLocalTimeStamp = true

# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

修改：

- 给source增加自定义拦截器
- 去掉本地时间戳 `a1.sinks.k1.hdfs.useLocalTimeStamp = true`
- 根据header中的logtime写文件

2、启动服务

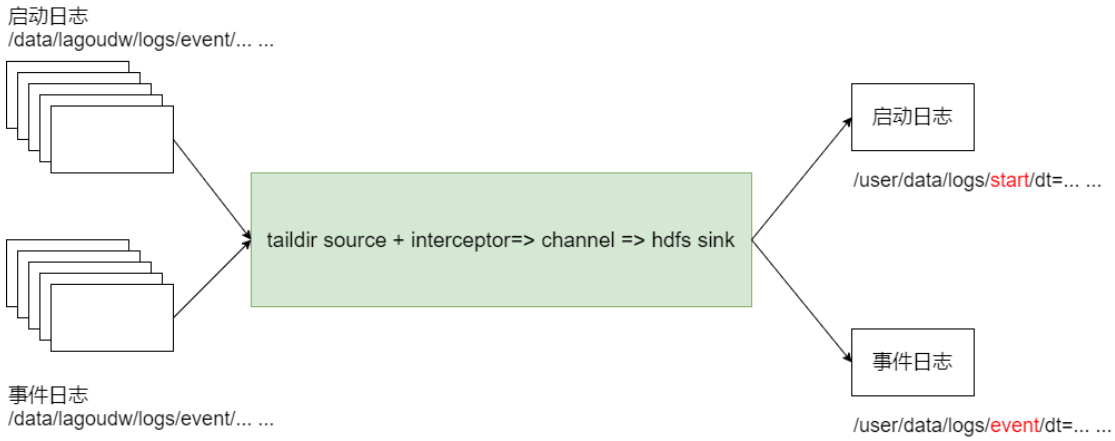
```
# 测试
flume-ng agent --conf /opt/apps/flume-1.9/conf --conf-file
/data/lagoudw/conf/flume-log2hdfs2.conf -name a1  -
Dflume.root.logger=INFO,console
```

3、拷贝日志

4、检查HDFS文件

2.7、采集启动日志和事件日志

本系统中要采集两种日志：启动日志、事件日志，不同的日志放置在不同的目录下。要想一次拿到全部日志需要监控多个目录。



总体思路

1、taildir监控多个目录

2、修改自定义拦截器，不同来源的数据加上不同标志

3、hdfs sink 根据标志写文件

Agent配置

```
a1.sources = r1
a1.sinks = k1
a1.channels = c1

# taildir source
a1.sources.r1.type = TAILDIR
a1.sources.r1.positionFile =
/data/lagoudw/conf/startlog_position.json
a1.sources.r1.filegroups = f1 f2
a1.sources.r1.filegroups.f1 = /data/lagoudw/logs/start/.*log
a1.sources.r1.headers.f1.logtype = start
a1.sources.r1.filegroups.f2 = /data/lagoudw/logs/event/.*log
a1.sources.r1.headers.f2.logtype = event

# 自定义拦截器
a1.sources.r1.interceptors = i1
a1.sources.r1.interceptors.i1.type =
cn.lagou.dw.flume.interceptor.LogTypeInterceptor$Builder

# memorychannel
a1.channels.c1.type = memory
a1.channels.c1.capacity = 100000
a1.channels.c1.transactionCapacity = 2000
```

```
# hdfs sink
a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path = /user/data/logs/%{logtype}/dt=%{logtime}/
a1.sinks.k1.hdfs.filePrefix = startlog

a1.sinks.k1.hdfs.fileType = DataStream

# 配置文件滚动方式（文件大小32M）
a1.sinks.k1.hdfs.rollSize = 33554432
a1.sinks.k1.hdfs.rollCount = 0
a1.sinks.k1.hdfs.rollInterval = 0
a1.sinks.k1.hdfs.idleTimeout = 0
a1.sinks.k1.hdfs.minBlockReplicas = 1

# 向hdfs上刷新的event的个数
a1.sinks.k1.hdfs.batchSize = 1000

# 使用本地时间
# a1.sinks.k1.hdfs.useLocalTimeStamp = true

# Bind the source and sink to the channel
a1.sources.r1.channels = c1
a1.sinks.k1.channel = c1
```

- `filegroups`

指定filegroups，可以有多个，以空格分隔（taildir source可同时监控多个目录中的文件）

- `headers.<filegroupName>.<headerKey>`

给event增加header key。不同的filegroup，可配置不同的value

自定义拦截器

编码完成后打包上传服务器，放置在$FLUME_HOME/lib 下

```
package cn.lagou.dw.flume.interceptor;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONArray;
import com.alibaba.fastjson.JSONObject;
import org.apache.commons.compress.utils.Charsets;
import org.apache.flume.Context;
import org.apache.flume.Event;
import org.apache.flume.event.SimpleEvent;
import org.apache.flume.interceptor.Interceptor;
```

```
import org.junit.Test;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class LogTypeInterceptor implements Interceptor {
    @Override
    public void initialize() {

    }

    @Override
    // 逐条处理event
    public Event intercept(Event event) {
        // 获取 event 的 body
        String eventBody = new String(event.getBody(),
Charsets.UTF_8);

        // 获取 event 的 header
        Map<String, String> headersMap = event.getHeaders();

        // 解析body获取json串
        String[] bodyArr = eventBody.split("\\s+");

        try{
            String jsonStr = bodyArr[6];

            // 解析json串获取时间戳
            String timestampStr = "";
            JSONObject jsonObject = JSON.parseObject(jsonStr);

            if (headersMap.getOrDefault("logtype",
"").equals("start")){
                // 取启动日志的时间戳
                timestampStr =
jsonObject.getJSONObject("app_active").getString("time");
            } else if (headersMap.getOrDefault("logtype",
"").equals("event")) {
                // 取事件日志第一条记录的时间戳
```

```
                JSONArray jsonArray =
jsonObject.getJSONArray("lagou_event");
                if (jsonArray.size() > 0){
                    timestampStr =
jsonArray.getJSONObject(0).getString("time");
                }
            }

            // 将时间戳转换为字符串 "yyyy-MM-dd"
            // 将字符串转换为Long
            long timestamp = Long.parseLong(timestampStr);
            DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd");

            Instant instant = Instant.ofEpochMilli(timestamp);
            LocalDateTime localDateTime =
LocalDateTime.ofInstant(instant, ZoneId.systemDefault());
            String date = formatter.format(localDateTime);

            // 将转换后的字符串放置header中
            headersMap.put("logtime", date);
            event.setHeaders(headersMap);
        }catch (Exception e){
            headersMap.put("logtime", "Unknown");
            event.setHeaders(headersMap);
        }

        return event;
    }

    @Override
    public List<Event> intercept(List<Event> events) {
        List<Event> lstEvent = new ArrayList<>();

        for (Event event: events){
            Event outEvent = intercept(event);
            if (outEvent != null) {
                lstEvent.add(outEvent);
            }
        }

        return lstEvent;
    }

    @Override
```

```
    public void close() {

    }

    public static class Builder implements Interceptor.Builder {
        @Override
        public Interceptor build() {
            return new LogTypeInterceptor();
        }

        @Override
        public void configure(Context context) {

        }
    }

    @Test
    public void startJunit(){
        String str = "2020-08-02 18:19:32.959 [main] INFO
com.lagou.ecommerce.AppStart - {\"app_active\":
{\"name\":\"app_active\",\"json\":
{\"entry\":\"1\",\"action\":\"0\",\"error_code\":\"0\"},\"time\":
1596342840284},\"attr\":{\"area\":\"大庆
\",\"uid\":\"2F10092A2\",\"app_v\":\"1.1.15\",\"event_type\":\"co
mmon\",\"device_id\":\"1FB872-
9A1002\",\"os_type\":\"2.8\",\"channel\":\"TB\",\"language\":\"ch
inese\",\"brand\":\"iphone-8\"}}";
        Map<String, String> map = new HashMap<>();
        // new Event
        Event event = new SimpleEvent();
        map.put("logtype", "start");
        event.setHeaders(map);
        event.setBody(str.getBytes(Charsets.UTF_8));

        // 调用interceptor处理event
        LogTypeInterceptor customerInterceptor = new
LogTypeInterceptor();
        Event outEvent = customerInterceptor.intercept(event);

        // 处理结果
        Map<String, String> headersMap = outEvent.getHeaders();
        System.out.println(JSON.toJSONString(headersMap));
    }

    @Test
```

```
    public void eventJunit(){
        String str = "2020-08-02 18:20:11.877 [main] INFO
com.lagou.ecommerce.AppEvent - {\"lagou_event\":
[{\"name\":\"goods_detail_loading\",\"json\":
{\"entry\":\"1\",\"goodsid\":\"0\",\"loading_time\":\"93\",\"acti
on\":\"3\",\"staytime\":\"56\",\"showtype\":\"2\"},\"time\":15963
43881690},{\"name\":\"loading\",\"json\":
{\"loading_time\":\"15\",\"action\":\"3\",\"loading_type\":\"3\",
\"type\":\"1\"},\"time\":1596356988428},
{\"name\":\"notification\",\"json\":
{\"action\":\"1\",\"type\":\"2\"},\"time\":1596374167278},
{\"name\":\"favorites\",\"json\":
{\"course_id\":1,\"id\":0,\"userid\":0},\"time\":1596350933962}],
\"attr\":{\"area\":\"长治
\",\"uid\":\"2F10092A4\",\"app_v\":\"1.1.14\",\"event_type\":\"co
mmon\",\"device_id\":\"1FB872-
9A1004\",\"os_type\":\"0.5.0\",\"channel\":\"QL\",\"language\":\"
chinese\",\"brand\":\"xiaomi-0\"}}";
        Map<String, String> map = new HashMap<>();
        // new Event
        Event event = new SimpleEvent();
        map.put("logtype", "event");
        event.setHeaders(map);
        event.setBody(str.getBytes(Charsets.UTF_8));

        // 调用interceptor处理event
        LogTypeInterceptor customerInterceptor = new
LogTypeInterceptor();
        Event outEvent = customerInterceptor.intercept(event);

        // 处理结果
        Map<String, String> headersMap = outEvent.getHeaders();
        System.out.println(JSON.toJSONString(headersMap));
    }
}
```

测试

启动Agent，拷贝日志，检查HDFS文件

```
# 清理环境
rm -f /data/lagoudw/conf/startlog_position.json
rm -f /data/lagoudw/logs/start/*.log
```

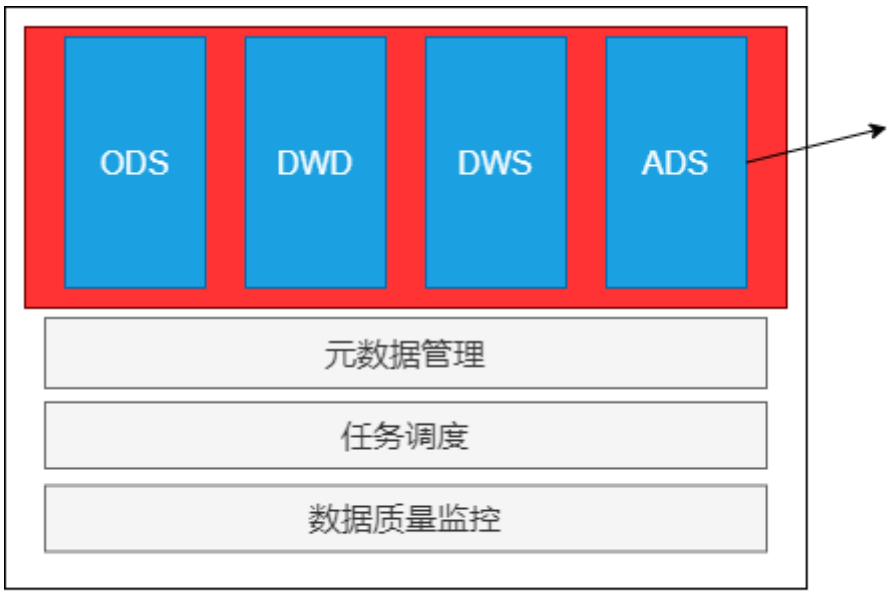
```
rm -f /data/lagoudw/logs/event/*.log

# 启动 Agent
flume-ng agent --conf /opt/apps/flume-1.9/conf --conf-file
/data/lagoudw/conf/flume-log2hdfs3.conf -name a1  -
Dflume.root.logger=INFO,console

# 拷贝日志
cd /data/lagoudw/logs/source
cp event0802.log ../event/
cp start0802.log ../start/

# 检查HDFS文件
hdfs dfs -ls /user/data/logs/event
hdfs dfs -ls /user/data/logs/start

# 生产环境中用以下方式启动Agent
nohup flume-ng agent --conf /opt/apps/flume-1.9/conf --conf-file
/data/lagoudw/conf/flume-log2hdfs3.conf -name a1  -
Dflume.root.logger=INFO,LOGFILE > /dev/null 2>&1 &
```

- nohup，该命令允许用户退出帐户/关闭终端之后继续运行相应的进程
- /dev/null，代表linux的空设备文件，所有往这个文件里面写入的内容都会丢失，俗称黑洞
- 标准输入0，从键盘获得输入 /proc/self/fd/0
- 标准输出1，输出到屏幕（控制台） /proc/self/fd/1
- 错误输出2，输出到屏幕（控制台） /proc/self/fd/2
- `>/dev/null` 标准输出1重定向到 /dev/null 中，此时标准输出不存在，没有任何地方能够找到输出的内容
- `2>&1` 错误输出将会和标准输出输出到同一个地方
- `>/dev/null 2>&1` 不会输出任何信息到控制台，也不会有任何信息输出到文件中

2.8 日志数据采集小结

- 使用taildir source 监控指定的多个目录，可以给不同目录的日志加上不同header
- 在每个目录中可以使用正则匹配多个文件
- 使用自定义拦截器，主要功能是从json串中获取时间戳，加到event的header中
- hdfs sink使用event header中的信息写数据（控制写文件的位置）
- hdfs文件的滚动方式（基于文件大小、基于event数量、基于时间）
- 调节flume jvm内存的分配

第3节 ODS建表和数据加载



数据仓库

ODS层的数据与源数据的格式基本相同。

创建ODS层表：

```
use ODS;
create external table ods.ods_start_log(
`str` string)
comment '用户启动日志信息'
partitioned by (`dt` string)
location '/user/data/logs/start';

-- 加载数据的功能(测试时使用)
alter table ods.ods_start_log add partition(dt='2020-08-02');
alter table ods.ods_start_log drop partition (dt='2020-08-02');
```

加载启动日志数据：

script/member_active/ods_load_log.sh

可以传参数确定日志，如果没有传参使用昨天日期

```
#! /bin/bash

APP=ODS
hive=/opt/apps/hive-2.3.7/bin/hive
```

```
# 可以输入日期，如果未输入日期取昨天的时间
if [ -n "$1" ]
then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

# 定义要执行的SQL
sql="
alter table "$APP".ods_start_log add partition(dt='$do_date');
"

$hive -e "$sql"
```

第4节 json数据处理

数据文件中每行必须是一个完整的 json 串，一个 json串 不能跨越多行。

Hive 处理json数据总体来说有三个办法：

- 使用内建的函数get_json_object、json_tuple
- 使用自定义的UDF
- 第三方的SerDe

4.1、使用内建函数处理

get_json_object(string json_string, string path)

返回值：String

说明：解析json字符串json_string，返回path指定的内容；如果输入的json字符串无效，那么返回NUll；函数每次只能返回一个数据项；

json_tuple(jsonStr, k1, k2, ...)

返回值：所有的输入参数、输出参数都是String；

说明：参数为一组键k1，k2，。。。。。和json字符串，返回值的元组。该方法比get_json_object高效，因此可以在一次调用中输入多个键；

explode，使用explod将Hive一行中复杂的 array 或 map 结构拆分成多行。

测试数据：

```
user1;18;male;{"id": 1,"ids": [101,102,103],"total_number": 3}
user2;20;female;{"id": 2,"ids": [201,202,203,204],"total_number":
4}
user3;23;male;{"id": 3,"ids":
[301,302,303,304,305],"total_number": 5}
user4;17;male;{"id": 4,"ids": [401,402,403,304],"total_number":
5}
user5;35;female;{"id": 5,"ids": [501,502,503],"total_number": 3}
```

建表加载数据：

```
CREATE TABLE IF NOT EXISTS jsont1(
username string,
age int,
sex string,
json string
)
row format delimited fields terminated by ';';

load data local inpath '/data/lagoudw/data/weibo.json' overwrite
into table jsont1;
```

json的处理：

```
-- get 单层值
select username, age, sex, get_json_object(json, "$.id") id,
get_json_object(json, "$.ids") ids,
get_json_object(json, "$.total_number") num
from jsont1;

-- get 数组
select username, age, sex, get_json_object(json, "$.id") id,
get_json_object(json, "$.ids[0]") ids0,
get_json_object(json, "$.ids[1]") ids1,
get_json_object(json, "$.ids[2]") ids2,
get_json_object(json, "$.ids[3]") ids3,
```

```
get_json_object(json, "$.total_number") num
from jsont1;

-- 使用 json_tuple 一次处理多个字段
select json_tuple(json, 'id', 'ids', 'total_number')
  from jsont1;

-- 有语法错误
select username, age, sex, json_tuple(json, 'id', 'ids',
'total_number')
  from jsont1;

-- 使用 explode + lateral view
-- 在上一步的基础上，再将数据展开
-- 第一步，将 [101,102,103] 中的 [ ] 替换掉
-- select "[101,102,103]"
-- select "101,102,103"
select regexp_replace("[101,102,103]", "\\[|\\]", "");

-- 第二步，将上一步的字符串变为数组
select split(regexp_replace("[101,102,103]", "\\[|\\]", ""),
",");

-- 第三步，使用explode + lateral view 将数据展开
select username, age, sex, id, ids, num
  from jsont1
lateral view json_tuple(json, 'id', 'ids', 'total_number') t1 as
id, ids, num;

with tmp as(
select username, age, sex, id, ids, num
  from jsont1
lateral view json_tuple(json, 'id', 'ids', 'total_number') t1 as
id, ids, num
)
select username, age, sex, id, ids1, num
  from tmp
lateral view explode(split(regexp_replace(ids, "\\[|\\]", ""),
",")) t1 as ids1;
```

小结：json_tuple 优点是一次可以解析多个json字段，对嵌套结果的解析操作复杂;

4.2、使用UDF处理

自定义UDF处理json串中的数组。自定义UDF函数：

- 输入：json串、数组的key
- 输出：字符串数组

pom文件增加依赖

```
        <dependency>
            <groupId>org.apache.hive</groupId>
            <artifactId>hive-exec</artifactId>
            <version>2.3.7</version>
            <scope>provided</scope>
        </dependency>
```

```
package cn.lagou.dw.hive.udf;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONArray;
import com.alibaba.fastjson.JSONException;
import com.alibaba.fastjson.JSONObject;
import com.google.common.base.Strings;
import org.apache.hadoop.hive.ql.exec.UDF;
import org.junit.Test;

import java.util.ArrayList;

public class ParseJsonArray extends UDF {
    public ArrayList<String> evaluate(String jsonStr, String
arrKey){
        if (Strings.isNullOrEmpty(jsonStr)) {
            return null;
        }

        try{
            JSONObject object = JSON.parseObject(jsonStr);
            JSONArray jsonArray = object.getJSONArray(arrKey);
            ArrayList<String> result = new ArrayList<>();
            for (Object o: jsonArray){
                result.add(o.toString());
            }
```

```
            return result;
        } catch (JSONException e){
            return null;
        }
    }

    @Test
    public void JunitParseJsonArray(){
        String str = "{\"id\": 1,\"ids\":
[101,102,103],\"total_number\": 3}";
        String key = "ids";
        ArrayList<String> evaluate = evaluate(str, key);
        System.out.println(JSON.toJSONString(evaluate));
    }
}
```

使用自定义 UDF 函数：

```
-- 添加开发的jar包（在Hive命令行中）
add jar /data/lagoudw/jars/cn.lagou.dw-1.0-SNAPSHOT-jar-with-
dependencies.jar;

-- 创建临时函数。指定类名一定要完整的路径，即包名加类名
create temporary function lagou_json_array as
"cn.lagou.dw.hive.udf.ParseJsonArray";

-- 执行查询
-- 解析json串中的数组
select username, age, sex, lagou_json_array(json, "ids") ids
  from jsont1;

-- 解析json串中的数组，并展开
select username, age, sex, ids1
  from jsont1
lateral view explode(lagou_json_array(json, "ids")) t1 as ids1;

-- 解析json串中的id、num
select username, age, sex, id, num
  from jsont1
lateral view json_tuple(json, 'id', 'total_number') t1 as id,
num;
```

```
-- 解析json串中的数组，并展开
select username, age, sex, ids1, id, num
  from jsont1
lateral view explode(lagou_json_array(json, "ids")) t1 as ids1
lateral view json_tuple(json, 'id', 'total_number') t1 as id,
num;
```

4.3、使用SerDe处理

序列化是对象转换为字节序列的过程；反序列化是字节序列恢复为对象的过程；

对象的序列化主要有两种用途：

- 对象的持久化，即把对象转换成字节序列后保存到文件中
- 对象数据的网络传送

SerDe 是Serializer 和 Deserializer 的简写形式。Hive使用Serde进行行对象的序列与反序列化。最后实现把文件内容映射到 hive 表中的字段数据类型。SerDe包括 Serialize/Deserilize 两个功能：

- Serialize把Hive使用的java object转换成能写入HDFS字节序列，或者其他系统能识别的流文件
- Deserilize把字符串或者二进制流转换成Hive能识别的java object对象

Read : HDFS files => InputFileFormat => <key, value> => Deserializer => Row object

Write : Row object => Seriallizer => <key, value> => OutputFileFormat => HDFS files

常见：https://cwiki.apache.org/confluence/display/Hive/DeveloperGuide#DeveloperGuide-HiveSerDe

Hive本身自带了几个内置的SerDe，还有其他一些第三方的SerDe可供选择。

```
create table t11(id string)
stored as parquet;

create table t12(id string)
stored as ORC;

desc formatted t11;
desc formatted t12;
```

LazySimpleSerDe（默认的SerDe）

ParquetHiveSerDe

OrcSerde

对于纯 json 格式的数据，可以使用 JsonSerDe 来处理。

```
{"id": 1,"ids": [101,102,103],"total_number": 3}
{"id": 2,"ids": [201,202,203,204],"total_number": 4}
{"id": 3,"ids": [301,302,303,304,305],"total_number": 5}
{"id": 4,"ids": [401,402,403,304],"total_number": 5}
{"id": 5,"ids": [501,502,503],"total_number": 3}
```

```
create table jsont2(
id int,
ids array<string>,
total_number int
)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe';

load data local inpath '/data/lagoudw/data/json2.dat' into table
jsont2;
```

各种json格式处理方法小结：

1、简单格式的json数据，使用get_json_object、json_tuple处理

2、对于嵌套数据类型，可以使用UDF

3、纯json串可使用JsonSerDe处理更简单

第5节 DWD层建表和数据加载

```
2020-08-02 18:19:32.966 [main] INFO  com.lagou.ecommerce.AppStart
- {"app_active":{"name":"app_active","json":
{"entry":"1","action":"1","error_code":"0"},"time":1596309585861}
,"attr":{"area":"绍
兴","uid":"2F10092A10","app_v":"1.1.16","event_type":"common","de
vice_id":"1FB872-
9A10010","os_type":"3.0","channel":"ML","language":"chinese","bra
nd":"Huawei-2"}}
2020-08-02
```

主要任务：ODS（包含json串） => DWD

json数据解析，丢弃无用数据（数据清洗），保留有效信息，并将数据展开，形成每日启动明细表。

5.1、创建DWD层表

```
use DWD;
drop table if exists dwd.dwd_start_log;
CREATE TABLE dwd.dwd_start_log(
`device_id` string,
`area` string,
`uid` string,
`app_v` string,
`event_type` string,
`os_type` string,
`channel` string,
`language` string,
`brand` string,
`entry` string,
`action` string,
`error_code` string
)
PARTITIONED BY (dt string)
STORED AS parquet;
```

表的格式：parquet、分区表

5.2、加载DWD层数据

script/member_active/dwd_load_start.sh

```
#! /bin/bash

source /etc/profile

# 可以输入日期，如果未输入日期取昨天的时间
if [ -n "$1" ]
then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

# 定义要执行的SQL
sql="
with tmp as(
select split(str, ' ')[7] line
  from ods.ods_start_log
 where dt='$do_date'
)
insert overwrite table dwd.dwd_start_log
partition(dt='$do_date')
select get_json_object(line, '$.attr.device_id'),
get_json_object(line, '$.attr.area'),
get_json_object(line, '$.attr.uid'),
get_json_object(line, '$.attr.app_v'),
get_json_object(line, '$.attr.event_type'),
get_json_object(line, '$.attr.os_type'),
get_json_object(line, '$.attr.channel'),
get_json_object(line, '$.attr.language'),
get_json_object(line, '$.attr.brand'),
get_json_object(line, '$.app_active.json.entry'),
get_json_object(line, '$.app_active.json.action'),
get_json_object(line, '$.app_active.json.error_code')
  from tmp;
"

hive -e "$sql"
```

日志文件 =》 Flume =》 HDFS =》 ODS =》 DWD

ODS =》 DWD；json数据的解析；数据清洗

下一步任务：DWD（会员的每日启动信息明细） => DWS（如何建表，如何加载数据）

活跃会员 ===> 新增会员 ===> 会员留存

第6节 活跃会员

活跃会员：打开应用的会员即为活跃会员；

新增会员：第一次使用应用的用户，定义为新增会员；

留存会员：某段时间的新增会员，经过一段时间后，仍继续使用应用认为是留存会员；

活跃会员指标需求：每日、每周、每月的活跃会员数

DWD：会员的每日启动信息明细（会员都是活跃会员；某个会员可能会出现多次）

DWS：每日活跃会员信息(关键)、每周活跃会员信息、每月活跃会员信息

每日活跃会员信息 ===> 每周活跃会员信息

每日活跃会员信息 ===> 每月活跃会员信息

ADS：每日、每周、每月活跃会员数（输出）

```
ADS表结构：
daycnt  weekcnt monthcnt    dt
```

备注：周、月为自然周、自然月

处理过程：

1、建表（每日、每周、每月活跃会员信息）

2、每日启动明细 ===> 每日活跃会员

3、每日活跃会员 => 每周活跃会员；每日活跃会员 => 每月活跃会员

4、汇总生成ADS层的数据

6.1、创建DWS层表

```
use dws;
drop table if exists dws.dws_member_start_day;
create table dws.dws_member_start_day
```

```
(
`device_id` string,
`uid` string,
`app_v` string,
`os_type` string,
`language` string,
`channel` string,
`area` string,
`brand` string
) COMMENT '会员日启动汇总'
partitioned by(dt string)
stored as parquet;

drop table if exists dws.dws_member_start_week;
create table dws.dws_member_start_week(
`device_id` string,
`uid` string,
`app_v` string,
`os_type` string,
`language` string,
`channel` string,
`area` string,
`brand` string,
`week` string
) COMMENT '会员周启动汇总'
PARTITIONED BY (`dt` string)
stored as parquet;

drop table if exists dws.dws_member_start_month;
create table dws.dws_member_start_month(
`device_id` string,
`uid` string,
`app_v` string,
`os_type` string,
`language` string,
`channel` string,
`area` string,
`brand` string,
`month` string
) COMMENT '会员月启动汇总'
PARTITIONED BY (`dt` string)
stored as parquet;
```

6.2、加载DWS层数据

script/member_active/dws_load_member_start.sh

```
#! /bin/bash

source /etc/profile

# 可以输入日期，如果未输入日期取昨天的时间
if [ -n "$1" ]
then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

# 定义要执行的SQL
# 汇总得到每日活跃会员信息，每日数据汇总得到每周、每月数据
sql="
insert overwrite table dws.dws_member_start_day
partition(dt='$do_date')
select device_id,
concat_ws('|', collect_set(uid)),
concat_ws('|', collect_set(app_v)),
concat_ws('|', collect_set(os_type)),
concat_ws('|', collect_set(language)),
concat_ws('|', collect_set(channel)),
concat_ws('|', collect_set(area)),
concat_ws('|', collect_set(brand))
  from dwd.dwd_start_log
 where dt='$do_date'
group by device_id;

-- 汇总得到每周活跃会员
insert overwrite table dws.dws_member_start_week
partition(dt='$do_date')
select device_id,
concat_ws('|', collect_set(uid)),
concat_ws('|', collect_set(app_v)),
concat_ws('|', collect_set(os_type)),
concat_ws('|', collect_set(language)),
concat_ws('|', collect_set(channel)),
concat_ws('|', collect_set(area)),
concat_ws('|', collect_set(brand)),
date_add(next_day('$do_date', 'mo'), -7)
```

```
  from dws.dws_member_start_day
 where dt >= date_add(next_day('$do_date', 'mo'), -7)
   and dt <= '$do_date'
group by device_id;

-- 汇总得到每月活跃会员
insert overwrite table dws.dws_member_start_month
partition(dt='$do_date')
select device_id,
concat_ws('|', collect_set(uid)),
concat_ws('|', collect_set(app_v)),
concat_ws('|', collect_set(os_type)),
concat_ws('|', collect_set(language)),
concat_ws('|', collect_set(channel)),
concat_ws('|', collect_set(area)),
concat_ws('|', collect_set(brand)),
date_format('$do_date', 'yyyy-MM')
  from dws.dws_member_start_day
 where dt >= date_format('$do_date', 'yyyy-MM-01')
   and dt <= '$do_date'
group by device_id;
"

hive -e "$sql"
```

注意shell的引号

ODS => DWD => DWS（每日、每周、每月活跃会员的汇总表）

6.3、创建ADS层表

计算当天、当周、当月活跃会员数量

```
drop table if exists ads.ads_member_active_count;
create table ads.ads_member_active_count(
`day_count`   int COMMENT '当日会员数量',
`week_count`  int COMMENT '当周会员数量',
`month_count` int COMMENT '当月会员数量'
) COMMENT '活跃会员数'
partitioned by(dt string)
row format delimited fields terminated by ',';
```

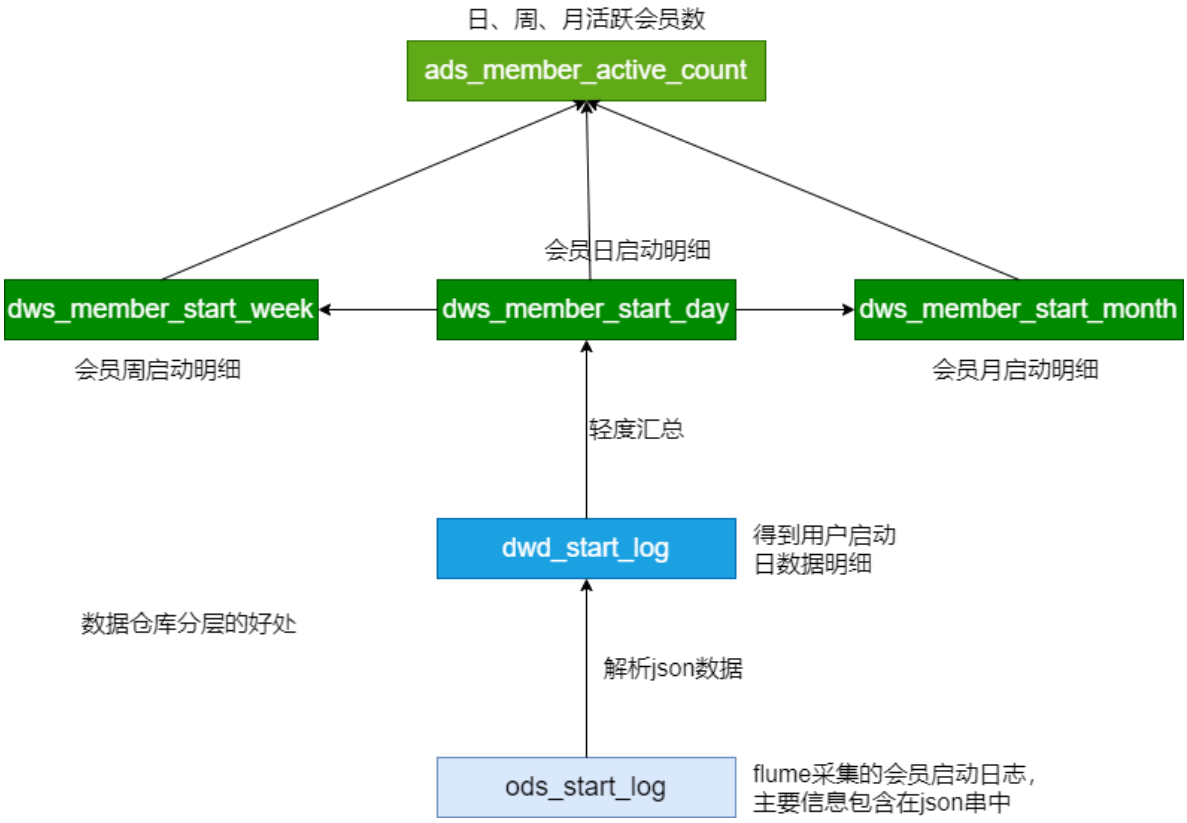
6.4、加载ADS层数据

script/member_active/ads_load_member_active.sh

```
#!/bin/bash

source /etc/profile

if [ -n "$1" ] ;then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

sql="
with tmp as(
select 'day' datelabel, count(*) cnt, dt
  from dws.dws_member_start_day
 where dt='$do_date'
group by dt
union all
select 'week' datelabel, count(*) cnt, dt
  from dws.dws_member_start_week
 where dt='$do_date'
group by dt
union all
select 'month' datelabel, count(*) cnt, dt
  from dws.dws_member_start_month
 where dt='$do_date'
group by dt
)
insert overwrite table ads.ads_member_active_count
partition(dt='$do_date')
select sum(case when datelabel='day' then cnt end) as day_count,
       sum(case when datelabel='week' then cnt end) as
week_count,
       sum(case when datelabel='month' then cnt end) as
month_count
  from tmp
group by dt;
"

hive -e "$sql"
```

```
#!/bin/bash

source /etc/profile

if [ -n "$1" ] ;then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

sql="
insert overwrite table ads.ads_member_active_count
partition(dt='$do_date')
select daycnt, weekcnt, monthcnt
  from (select dt, count(*) daycnt
          from dws.dws_member_start_day
         where dt='$do_date'
        group by dt
        ) day join
        (select dt, count(*) weekcnt
           from dws.dws_member_start_week
          where dt='$do_date'
        group by dt
        ) week on day.dt=week.dt
        join
        (select dt, count(*) monthcnt
           from dws.dws_member_start_month
          where dt='$do_date'
        group by dt
        ) month on day.dt=month.dt;
"

hive -e "$sql"
```

6.5、小结



脚本执行次序：

```
ods_load_startlog.sh
dwd_load_startlog.sh
dws_load_member_start.sh
ads_load_member_active.sh
```

第7节 新增会员

留存会员：某段时间的新增会员，经过一段时间后，仍继续使用应用认为是留存会员；

新增会员：第一次使用应用的用户，定义为新增会员；卸载再次安装的设备，不会被算作一次新增。

新增会员先计算 => 计算会员留存

需求：每日新增会员数

08-02：

DWD：会员每日启动明细（95-110）；所有会员的信息（1-100）？？？

- 新增会员：101-110
- 新增会员数据 + 旧的所有会员的信息 = 新的所有会员的信息（1-110）

08-03：

DWD：会员每日启动明细（100-120）；所有会员的信息（1-110）

- 新增会员：111-120
- 新增会员数据 + 旧的所有会员的信息 = 新的所有会员的信息（1-120）

计算步骤：

- 计算新增会员
- 更新所有会员信息

改进后方法：

- 在所有会员信息中增加时间列，表示这个会员是哪一天成为新增会员
- 只需要一张表：所有会员的信息（id，dt）
- 将新增会员 插入 所有会员表中

案例：如何计算新增会员

```
-- 日启动表 => DWS
drop table t1;
create table t1(id int, dt string)
row format delimited fields terminated by ',';
load data local inpath '/data/lagoudw/data/t10.dat' into table
t1;
4,2020-08-02
5,2020-08-02
6,2020-08-02
7,2020-08-02
8,2020-08-02
9,2020-08-02

-- 全量数据 => DWS
drop table t2;
create table t2(id int, dt string)
row format delimited fields terminated by ',';
load data local inpath '/data/lagoudw/data/t2.dat' into table t2;
```

```
1,2020-08-01
2,2020-08-01
3,2020-08-01
4,2020-08-01
5,2020-08-01
6,2020-08-01

-- 找出 2020-08-02 的新用户
select t1.id, t1.dt, t2.id, t2.dt
  from t1 left join t2 on t1.id=t2.id
 where t1.dt="2020-08-02";

select t1.id, t1.dt
  from t1 left join t2 on t1.id=t2.id
 where t1.dt="2020-08-02"
   and t2.id is null;

-- 将找到 2020-08-02 新用户数据插入t2表中
insert into table t2
select t1.id, t1.dt
  from t1 left join t2 on t1.id=t2.id
 where t1.dt="2020-08-02"
   and t2.id is null;

-- 检查结果
select * from t2;

-- t1 加载 2020-08-03 的数据
14,2020-08-03
15,2020-08-03
16,2020-08-03
17,2020-08-03
18,2020-08-03
19,2020-08-03
load data local inpath '/data/lagoudw/data/t3.dat' into table t1;

-- 将找到 2020-08-03 新用户数据插入t2表中
insert into table t2
select t1.id, t1.dt
  from t1 left join t2 on t1.id=t2.id
 where t1.dt="2020-08-03"
   and t2.id is null;

-- 检查结果
select * from t2;
```

7.1、创建DWS层表

```
drop table if exists dws.dws_member_add_day;
create table dws.dws_member_add_day
(
`device_id` string,
`uid` string,
`app_v` string,
`os_type` string,
`language` string,
`channel` string,
`area` string,
`brand` string,
`dt` string
)  COMMENT '每日新增会员明细'
stored as parquet;
```

7.2、加载DWS层数据

script/member_active/dws_load_member_add_day.sh

```
#!/bin/bash

source /etc/profile

if [ -n "$1" ]
then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

sql="
insert into table dws.dws_member_add_day
select t1.device_id,
t1.uid,
t1.app_v,
t1.os_type,
t1.language,
t1.channel,
t1.area,
```

```
t1.brand,
'$do_date'
  from dws.dws_member_start_day t1 left join
dws.dws_member_add_day t2
    on t1.device_id=t2.device_id
 where t1.dt='$do_date'
   and t2.device_id is null;
"

hive -e "$sql"
```

7.3、创建ADS层表

```
drop table if exists ads.ads_new_member_cnt;
create table ads.ads_new_member_cnt
(
`cnt` string
)
partitioned by(dt string)
row format delimited fields terminated by ',';
```

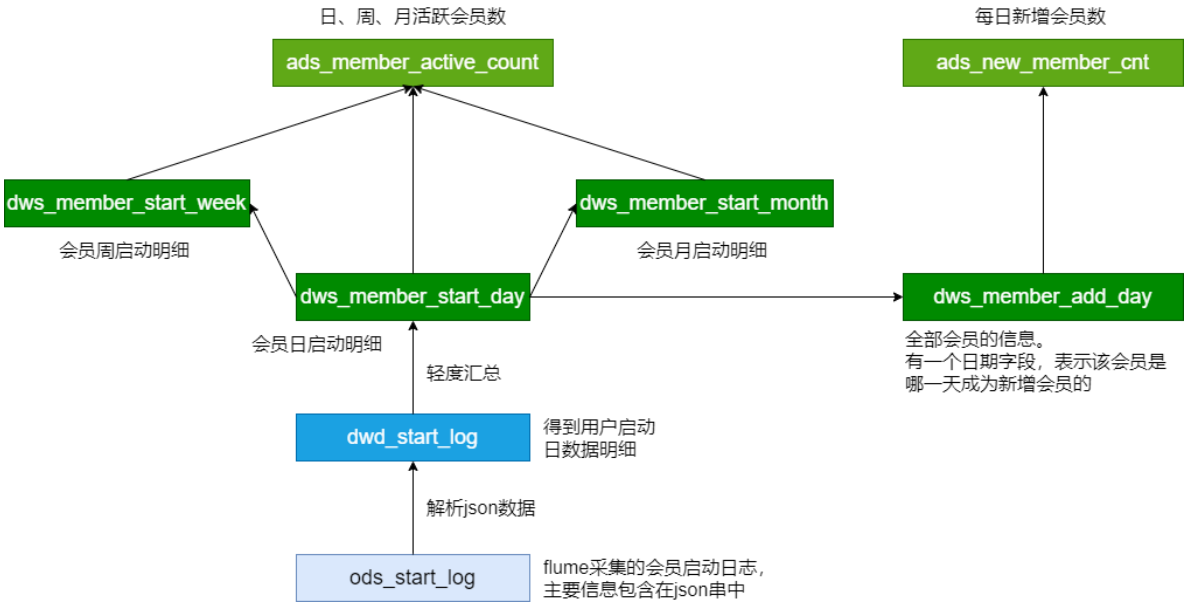
7.4、加载ADS层数据

script/member_active/ads_load_member_add.sh

```
#!/bin/bash

source /etc/profile

if [ -n "$1" ] ;then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

sql="
insert overwrite table ads.ads_new_member_cnt
partition (dt='$do_date')
select count(1)
  from dws.dws_member_add_day
 where dt = '$do_date'
```

```
"

hive -e "$sql"
```

7.5、小结



调用脚本次序：

```
dws_load_member_add_day.sh
ads_load_member_add.sh
```

第8节 留存会员

留存会员与留存率：某段时间的新增会员，经过一段时间后，仍继续使用应用认为是留存会员；这部分会员占当时新增会员的比例为留存率。

需求：1日、2日、3日的会员留存数和会员留存率

30	**31**	**1**	**2**	
		10W新会员	3W	1日留存数
	20W		5W	2日留存数
30W			4W	3日留存数

10W新会员：dws_member_add_day（dt=08-01）明细

3W：特点 在1号是新会员，在2日启动了（2日的启动日志）
dws_member_start_day

8.1、创建DWS层表

```
-- 会员留存明细
drop table if exists dws.dws_member_retention_day;
create table dws.dws_member_retention_day
(
`device_id` string,
`uid` string,
`app_v` string,
`os_type` string,
`language` string,
`channel` string,
`area` string,
`brand` string,
`add_date` string  comment   '会员新增时间',
`retention_date` int comment '留存天数'
)COMMENT '每日会员留存明细'
PARTITIONED BY (`dt` string)
stored as parquet;
```

8.2、加载DWS层数据

script/member_active/dws_load_member_retention_day.sh

```
#!/bin/bash

source /etc/profile

if [ -n "$1" ] ;then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

sql="
insert overwrite table dws.dws_member_retention_day
partition(dt='$do_date')
(
select t2.device_id,
       t2.uid,
```

```
       t2.app_v,
       t2.os_type,
       t2.language,
       t2.channel,
       t2.area,
       t2.brand,

       t2.dt add_date,
       1
  from dws.dws_member_start_day t1 join dws.dws_member_add_day t2
on t1.device_id=t2.device_id
 where t2.dt=date_add('$do_date', -1)
   and t1.dt='$do_date'

union all

select t2.device_id,
       t2.uid,
       t2.app_v,
       t2.os_type,
       t2.language,
       t2.channel,
       t2.area,
       t2.brand,
       t2.dt add_date,
       2
  from dws.dws_member_start_day t1 join dws.dws_member_add_day t2
on t1.device_id=t2.device_id
 where t2.dt=date_add('$do_date', -2)
   and t1.dt='$do_date'

union all

select t2.device_id,
       t2.uid,
       t2.app_v,
       t2.os_type,
       t2.language,
       t2.channel,
       t2.area,
       t2.brand,
       t2.dt add_date,
       3
  from dws.dws_member_start_day t1 join dws.dws_member_add_day t2
on t1.device_id=t2.device_id
```

```
 where t2.dt=date_add('$do_date', -3)
   and t1.dt='$do_date'
);
"

hive -e "$sql"
```

return code 2 from org.apache.hadoop.hive.ql.exec.mr.MapRedTask

一般是内部错误

1、找日志（hive.log【简略】 / MR的日志【详细】）

hive.log ===> 缺省情况下 /tmp/root/hive.log （hive-site.conf）

MR的日志 ===> 启动historyserver、日志聚合 + SQL运行在集群模式

2、改写SQL

```
#!/bin/bash

source /etc/profile

if [ -n "$1" ] ;then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

sql="
drop table if exists tmp.tmp_member_retention;
create table tmp.tmp_member_retention as
(
select t2.device_id,
       t2.uid,
       t2.app_v,
       t2.os_type,
       t2.language,
       t2.channel,
       t2.area,
       t2.brand,

       t2.dt add_date,
       1
```

```
  from dws.dws_member_start_day t1 join dws.dws_member_add_day t2
on t1.device_id=t2.device_id
 where t2.dt=date_add('$do_date', -1)
   and t1.dt='$do_date'

union all

select t2.device_id,
       t2.uid,
       t2.app_v,
       t2.os_type,
       t2.language,
       t2.channel,
       t2.area,
       t2.brand,
       t2.dt add_date,
       2
  from dws.dws_member_start_day t1 join dws.dws_member_add_day t2
on t1.device_id=t2.device_id
 where t2.dt=date_add('$do_date', -2)
   and t1.dt='$do_date'

union all

select t2.device_id,
       t2.uid,
       t2.app_v,
       t2.os_type,
       t2.language,
       t2.channel,
       t2.area,
       t2.brand,
       t2.dt add_date,
       3
  from dws.dws_member_start_day t1 join dws.dws_member_add_day t2
on t1.device_id=t2.device_id
 where t2.dt=date_add('$do_date', -3)
   and t1.dt='$do_date'
);

insert overwrite table dws.dws_member_retention_day
partition(dt='$do_date')
select * from tmp.tmp_member_retention;
"
```

```
hive -e "$sql"
```

8.3、创建ADS层表

```
-- 会员留存数
drop table if exists ads.ads_member_retention_count;
create table ads.ads_member_retention_count
(
  `add_date`        string  comment '新增日期',
  `retention_day`   int comment '截止当前日期留存天数',
  `retention_count` bigint comment  '留存数'
)  COMMENT '会员留存数'
partitioned by(dt string)
row format delimited fields terminated by ',';

-- 会员留存率
drop table if exists ads.ads_member_retention_rate;
create table ads.ads_member_retention_rate
(
  `add_date`          string comment '新增日期',
  `retention_day`     int comment '截止当前日期留存天数',
  `retention_count`   bigint comment  '留存数',
  `new_mid_count`     bigint comment '当日会员新增数',
  `retention_ratio`   decimal(10,2) comment '留存率'
) COMMENT '会员留存率'
partitioned by(dt string)
row format delimited fields terminated by ',';
```

8.4、加载ADS层数据

script/member_active/ads_load_member_retention.sh

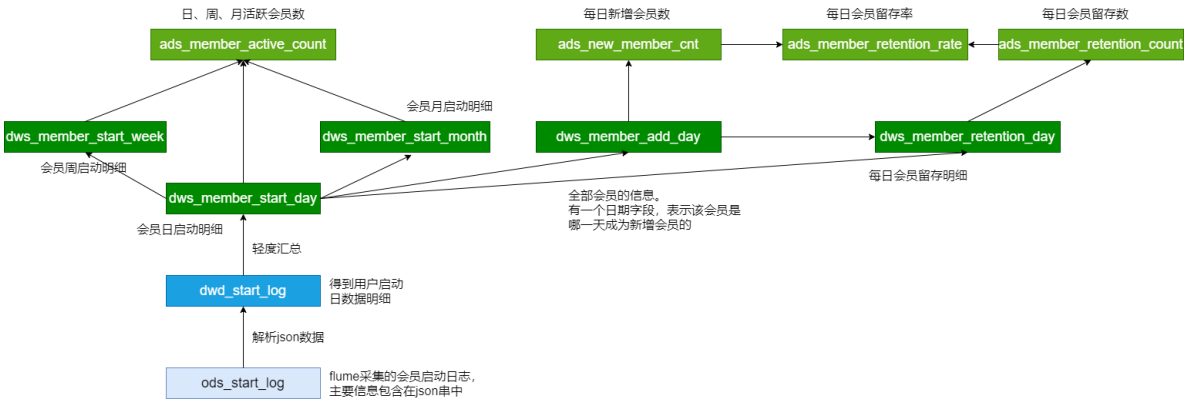
```
#!/bin/bash

source /etc/profile

if [ -n "$1" ] ;then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi
```

```
sql="
insert overwrite table ads.ads_member_retention_count
partition (dt='$do_date')
select add_date, retention_date,
       count(*) retention_count
  from dws.dws_member_retention_day
 where dt='$do_date'
group by add_date, retention_date;

insert overwrite table ads.ads_member_retention_rate
partition (dt='$do_date')
select t1.add_date,
       t1.retention_day,
       t1.retention_count,
       t2.cnt,
       t1.retention_count/t2.cnt*100
  from ads.ads_member_retention_count t1 join
ads.ads_new_member_cnt t2 on t1.dt=t2.dt
 where t1.dt='$do_date';
"

hive -e "$sql"
```

备注：最后一条SQL的连接条件应为：t1.add_date=t2.dt。在10.4 节中有详细说明。

8.5、小结

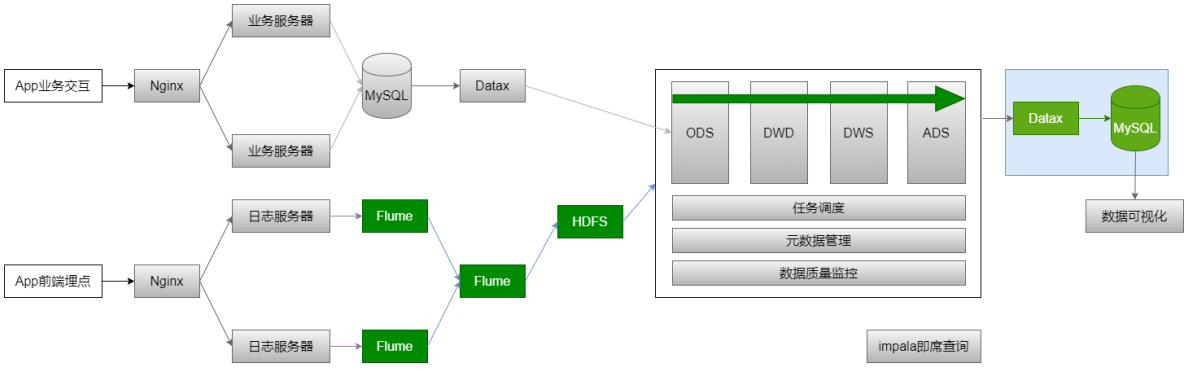
会员活跃度--活跃会员数、新增会员、留存会员



脚本调用次序：

```
# 加载ODS / DWD 层采集
ods_load_startlog.sh
dwd_load_startlog.sh

# 活跃会员
dws_load_member_start.sh
ads_load_member_active.sh

# 新增会员
dws_load_member_add_day.sh
ads_load_member_add.sh

# 会员留存
dws_load_member_retention_day.sh
ads_load_member_retention.sh
```

第9节 Datax 数据导出



基本概念及安装参见《附录一 DataX快速入门》

ADS有4张表需要从数据仓库的ADS层导入MySQL，即：Hive => MySQL

```
ads.ads_member_active_count
ads.ads_member_retention_count
ads.ads_member_retention_rate
ads.ads_new_member_cnt
```

```
-- MySQL 建表
-- 活跃会员数
create database dwads;
drop table if exists dwads.ads_member_active_count;
create table dwads.ads_member_active_count(
`dt`          varchar(10) COMMENT '统计日期',
```

```
`day_count`   int COMMENT '当日会员数量',
`week_count`  int COMMENT '当周会员数量',
`month_count` int COMMENT '当月会员数量',
primary key (dt)
);

-- 新增会员数
drop table if exists dwads.ads_new_member_cnt;
create table dwads.ads_new_member_cnt
(
`dt`  varchar(10) COMMENT '统计日期',
`cnt` string,
primary key (dt)
);

-- 会员留存数
drop table if exists dwads.ads_member_retention_count;
create table dwads.ads_member_retention_count
(
  `dt`              varchar(10) COMMENT '统计日期',
  `add_date`        string  comment '新增日期',
  `retention_day`   int comment '截止当前日期留存天数',
  `retention_count` bigint comment  '留存数',
  primary key (dt)
)  COMMENT '会员留存情况';

-- 会员留存率
drop table if exists dwads.ads_member_retention_rate;
create table dwads.ads_member_retention_rate
(
  `dt`               varchar(10) COMMENT '统计日期',
  `add_date`         string comment '新增日期',
  `retention_day`    int comment '截止当前日期留存天数',
  `retention_count`  bigint comment  '留存数',
  `new_mid_count`    bigint comment '当日会员新增数',
  `retention_ratio`  decimal(10,2) comment '留存率',
  primary key (dt)
) COMMENT '会员留存率';
```

导出活跃会员数(ads_member_active_count)

export_member_active_count.json

hdfsreader => mysqlwriter

```
{
    "job": {
        "setting": {
            "speed": {
                "channel": 1
            }
        },
        "content": [{
            "reader": {
                "name": "hdfsreader",
                "parameter": {
                    "path":
"/lagou/hive/warehouse/ads.db/ads_member_active_count/dt=$do_date
/*",
                    "defaultFS": "hdfs://hadoop1:9000",
                    "column": [{
                            "type": "string",
                            "value": "$do_date"
                        }, {
                            "index": 0,
                            "type": "string"
                        },
                        {
                            "index": 1,
                            "type": "string"
                        },
                        {
                            "index": 2,
                            "type": "string"
                        }
                    ],
                    "fileType": "text",
                    "encoding": "UTF-8",
                    "fieldDelimiter": ","
                }

            },
            "writer": {
                "name": "mysqlwriter",
                "parameter": {
                    "writeMode": "replace",
                    "username": "hive",
                    "password": "12345678",
                    "column":
["dt","day_count","week_count","month_count"],
```

```
                    "preSql": [
                        ""
                    ],
                    "connection": [{
                        "jdbcUrl":
"jdbc:mysql://hadoop2:3306/dwads?
useUnicode=true&characterEncoding=utf-8",
                        "table": [
                            "ads_member_active_count"
                        ]
                    }]
                }
            }
        }]
    }
}
```

执行命令:

```
python datax.py -p "-Ddo_date=2020-08-02"
/data/lagoudw/script/member_active/t1.json
```

export_member_active_count.sh

```
#!/bin/bash
JSON=/data/lagoudw/script/member_active
source /etc/profile

if [ -n "$1" ] ;then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

python $DATAX_HOME/bin/datax.py -p "-Ddo_date=$do_date"
$JSON/export_member_active_count.json 2020-08-02
```

第10节 高仿日启动数据测试

数据采集 => ODS => DWD => DWS => ADS> MySQL

活跃会员、新增会员、会员留存

DAU： Daily Active User（日活跃用户）

MAU： monthly active user（月活跃用户）

假设App的DAU在1000W左右，日启动数据大概 1000W 条；

测试3天的数据：7月21日、7月22日、7月23日。1000W条数据约3.5G+，每条记录约370字节。

10.1、Hive on MR测试

选择 7月21日 的启动日志进行测试

1、使用 flume 采集数据（采集3天的数据）

修改flume的参数：1G滚动一次；加大channel缓存；加大刷新 hdfs 的缓存

```
# 配置文件滚动方式（文件大小1G）
a1.sinks.k1.hdfs.rollSize = 1073741824

a1.channels.c1.capacity = 500000
a1.channels.c1.transactionCapacity = 20000

# 向hdfs上刷新的event个数
a1.sinks.k1.hdfs.batchSize = 10000
```

```
# 清理工作。删除元数据文件、日志、hdfs等文件
rm -f /data/lagoudw/conf/startlog_position.json
rm -rf /data/lagoudw/logs/start/*
hdfs dfs -rm -r -f /user/data/logs/start/dt=2020-07-21

# 启动flume
flume-ng agent --conf /opt/apps/flume-1.9/conf --conf-file
/data/lagoudw/conf/flume-log2hdfs4.conf -name a1 -
Dflume.root.logger=INFO,console

# 写日志
```

```
java -cp data-generator-1.1-SNAPSHOT-jar-with-dependencies.jar
com.lagou.ecommerce.AppStart 1 10000000 2020-07-21 >
/data/lagoudw/logs/start/start0721.log

java -cp data-generator-1.1-SNAPSHOT-jar-with-dependencies.jar
com.lagou.ecommerce.AppStart 6000000 16000000 2020-07-22 >
/data/lagoudw/logs/start/start0722.log

java -cp data-generator-1.1-SNAPSHOT-jar-with-dependencies.jar
com.lagou.ecommerce.AppStart 8000000 18000000 2020-07-23 >
/data/lagoudw/logs/start/start0723.log

# 检查 hdfs 文件是否到达
hdfs dfs -ls /user/data/logs/start/dt=2020-07-21
```

1个文件大小3.5G，时间4分钟左右

2、执行脚本

```
SCRIPT_HOME=/data/lagoudw/script/member_active

# 加载 ODS 层数据（文件与表建立关联）
sh $SCRIPT_HOME/ods_load_startlog.sh 2020-07-21

# 加载 ODS 层数据（解析json数据）
sh $SCRIPT_HOME/dwd_load_startlog.sh 2020-07-21

# number of mappers: 14; number of reducers: 0
# 是一个 map-only 的 Task；只对输入的数据做格式上的转换，没有聚合操作（即没有reduce task）
# mapred.max.split.size=256M；3.5G / 256M = 14 Mapper Task

# 活跃会员
sh $SCRIPT_HOME/dws_load_member_start.sh 2020-07-21
# number of mappers: 3; number of reducers: 2
# 调整了task的内存分配(根据实际情况分配)
# 任务执行时间：3.5 + 3 + 3 = 10分钟
# ODS(Text) => 14个map => DWD(parquet) => 小文件合并 256M切分 => 3 map

sh $SCRIPT_HOME/ads_load_member_active.sh 2020-07-21

# 新增会员
sh $SCRIPT_HOME/dws_load_member_add_day.sh 2020-07-21
```

```
sh $SCRIPT_HOME/ads_load_member_add.sh 2020-07-21

# 会员留存
sh $SCRIPT_HOME/dws_load_member_retention_day.sh 2020-07-21
sh $SCRIPT_HOME/ads_load_member_retention.sh 2020-07-21
```

相关表：

```
select count(*) from ods.ods_start_log where dw='2020-07-21';

select count(*) from dwd.dwd_start_log where dw='2020-07-21';

select count(*) from dws.dws_member_start_day where dw='2020-07-21';
select count(*) from dws.dws_member_start_week where dw='2020-07-21';
select count(*) from dws.dws_member_start_month where dw='2020-07-21';
select count(*) from dws.dws_member_add_day where dw='2020-07-21';
select count(*) from dws.dws_member_retention_day where dw='2020-07-21';

select count(*) from ads.ads_member_active_count where dw='2020-07-21';
select count(*) from ads.ads_new_member_cnt where dw='2020-07-21';
select count(*) from ads.ads_member_retention_count where dw='2020-07-21';
select count(*) from ads.ads_member_retention_rate where dw='2020-07-21';
```

遇到的问题：

Error: Java heap space

原因：内存分配问题

解决思路：给map、reduce task分配合理的内存；map、reduce task处理合理的数据

现在情况下map task分配了多少内存？使用的是缺省参数每个task分配200M内存【mapred.child.java.opts】

每个节点：8 core / 32G；mapred.child.java.opts = 3G

```
        <property>
                <name>mapred.child.java.opts</name>
                <value>-Xmx3072m</value>
        </property>
```

调整map个数：

mapred.max.split.size=256000000

调整reduce个数：

hive.exec.reducers.bytes.per.reducer

hive.exec.reducers.max

10.2、Tez简介及安装

参见附录二 Hive on Tez

10.3、Hive on Tez测试

07-22(新增600W) / 0723(新增200W)：1000W条左右，

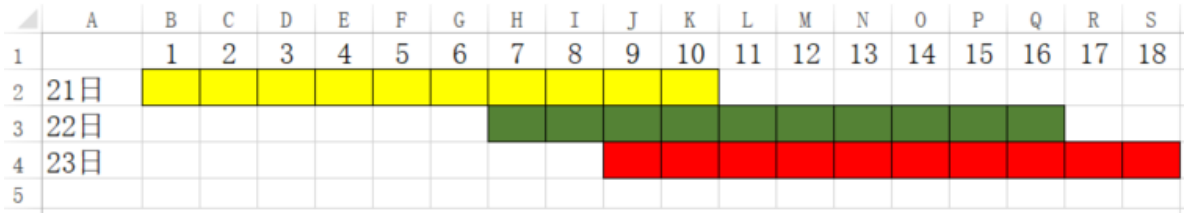
执行脚本

```
SCRIPT_HOME=/data/lagoudw/script/member_active

# 加载 ODS 层数据（文件与表建立关联）
sh $SCRIPT_HOME/ods_load_startlog.sh 2020-07-22

# 加载 ODS 层数据（解析json数据）
sh $SCRIPT_HOME/dwd_load_startlog.sh 2020-07-22

# 活跃会员
sh $SCRIPT_HOME/dws_load_member_start.sh 2020-07-22
sh $SCRIPT_HOME/ads_load_member_active.sh 2020-07-22

# 新增会员
sh $SCRIPT_HOME/dws_load_member_add_day.sh 2020-07-22
sh $SCRIPT_HOME/ads_load_member_add.sh 2020-07-22

# 会员留存
```

```
sh $SCRIPT_HOME/dws_load_member_retention_day.sh 2020-07-22
sh $SCRIPT_HOME/ads_load_member_retention.sh 2020-07-22
```

```
SCRIPT_HOME=/data/lagoudw/script/member_active

# 加载 ODS 层数据（文件与表建立关联）
sh $SCRIPT_HOME/ods_load_startlog.sh 2020-07-23

# 加载 ODS 层数据（解析json数据）
sh $SCRIPT_HOME/dwd_load_startlog.sh 2020-07-23

# 活跃会员
sh $SCRIPT_HOME/dws_load_member_start.sh 2020-07-23
sh $SCRIPT_HOME/ads_load_member_active.sh 2020-07-23

# 新增会员
sh $SCRIPT_HOME/dws_load_member_add_day.sh 2020-07-23
sh $SCRIPT_HOME/ads_load_member_add.sh 2020-07-23

# 会员留存
sh $SCRIPT_HOME/dws_load_member_retention_day.sh 2020-07-23
sh $SCRIPT_HOME/ads_load_member_retention.sh 2020-07-23
```

10.4、会员留存率的计算



看图说话：
23日1日留存。22日新增会员(600W)，在23日有600W会员登录了；留存率：100%
23日2日留存。21日新增会员(1000W)，在23日有200W会员登录了；留存率：20%
22日1日留存。21日新增会员(1000W)，在22日有400W会员登录了；留存率：40%

新增会员：21日、22日新增会员信息

```
select * from ads.ads_new_member_cnt limit 10;          => t2
cnt     dt
9799962 2020-07-21
6200027 2020-07-22
2000000 2020-07-23

select * from ads.ads_member_retention_count limit 100;   => t1
add_date  retention_day retention_count  dt
2020-07-21      1       3799973 2020-07-22      9799962 2020-07-21
2020-07-21      2       1799973 2020-07-23      9799962 2020-07-21
2020-07-22      1       6200027 2020-07-23      6200027 2020-07-22

select t1.*, t2.*
  from ads.ads_member_retention_count t1 join ads.ads_new_member_cnt t2 on
t1.add_date=t2.dt
 where t1.dt='2020-07-23';
```

script/member_active/ads_load_member_retention.sh（已修改Bug）

```
#!/bin/bash

source /etc/profile

if [ -n "$1" ] ;then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

sql="
insert overwrite table ads.ads_member_retention_count
partition (dt='$do_date')
select add_date, retention_date,
       count(*) retention_count
  from dws.dws_member_retention_day
 where dt='$do_date'
group by add_date, retention_date;

insert overwrite table ads.ads_member_retention_rate
```

```
partition (dt='$do_date')
select t1.add_date,
       t1.retention_day,
       t1.retention_count,
       t2.cnt,
       t1.retention_count/t2.cnt*100
  from ads.ads_member_retention_count t1 join
ads.ads_new_member_cnt t2 on t1.add_date=t2.dt
 where t1.dt='$do_date';
"

hive -e "$sql"
```

修改后的代码（计算留存率）：

```
select t1.*, t2.*
  from ads.ads_member_retention_count t1 join
ads.ads_new_member_cnt t2 on
t1.add_date=t2.dt
 where t1.dt='2020-07-23';
```

备注：主要改的是连接条件。将连接条件改为：t1.add_date=t2.dt

```
SCRIPT_HOME=/data/lagoudw/script/member_active
sh $SCRIPT_HOME/ads_load_member_retention.sh 2020-07-23
```

第四部分 电商分析之--广告业务

互联网平台通行的商业模式是利用免费的基础服务吸引凝聚大量用户，并利用这些用户资源开展广告或其他增值业务实现盈利从而反哺支撑免费服务的生存和发展。广告收入不仅成为互联网平台的重要收入之一，更决定了互联网平台的发展程度。

电商平台本身就汇聚了海量的商品、店铺的信息，天然适合进行商品的推广。对于电商和广告主来说，广告投放的目的无非就是吸引更多的用户，最终实现营销转化。因此非常关注不同位置广告的曝光量、点击量、购买量、点击率、购买率。

第1节 需求分析

事件日志数据样例：


```
{
    "lagou_event": [{
        "name": "goods_detail_loading",
        "json": {
            "entry": "3",
            "goodsid": "0",
            "loading_time": "80",
            "action": "4",
            "staytime": "68",
            "showtype": "4"
        },
        "time": 1596225273755
    }, {
        "name": "loading",
        "json": {
            "loading_time": "18",
            "action": "1",
            "loading_type": "2",
            "type": "3"
        },
        "time": 1596231657803
    }, {
        "name": "ad",
        "json": {
            "duration": "17",
            "ad_action": "0",
            "shop_id": "786",
            "event_type": "ad",
            "ad_type": "4",
            "show_style": "1",
            "product_id": "2772",
            "place": "placeindex_left",
            "sort": "0"
        },
        "time": 1596278404415
    }, {
        "name": "favorites",
        "json": {
            "course_id": 0,
            "id": 0,
            "userid": 0
        },
        "time": 1596239532527
```

```
    }, {
        "name": "praise",
        "json": {
            "id": 2,
            "type": 3,
            "add_time": "1596258672095",
            "userid": 8,
            "target": 6
        },
        "time": 1596274343507
    }],
    "attr": {
        "area": "拉萨",
        "uid": "2F10092A86",
        "app_v": "1.1.12",
        "event_type": "common",
        "device_id": "1FB872-9A10086",
        "os_type": "4.1",
        "channel": "KS",
        "language": "chinese",
        "brand": "xiaomi-2"
    }
}
```


采集的信息包括：

- 商品详情页加载：goods_detail_loading
- 商品列表：loading
- 消息通知：notification
- 商品评论：comment
- 收藏：favorites
- 点赞：praise
- 广告：ad
    - action。用户行为；0 曝光；1 曝光后点击；2 购买
    - duration。停留时长
    - shop_id。商家id
    - event_type。"ad"
    - ad_type。格式类型；1 JPG；2 PNG；3 GIF；4 SWF
    - show_style。显示风格，0 静态图；1 动态图
    - product_id。产品id
    - place。广告位置；首页=1，左侧=2，右侧=3，列表页=4
    - sort。排序位置

需求指标：

1、点击次数统计(分时统计)

曝光次数、不同用户id数、不同用户数

点击次数、不同用户id数、不同用户数

购买次数、不同用户id数、不同用户数

2、转化率-漏斗分析

点击率 = 点击次数 / 曝光次数

购买率 = 购买次数 / 点击次数

3、活动曝光效果评估：

行为(曝光、点击、购买)、时间段、广告位、产品，统计对应的次数

时间段、广告位、商品，曝光次数最多的前N个

第2节 事件日志采集

1、启动Flume Agent（适当的修改参数，128M滚动一次）

```
# 启动flume
flume-ng agent --conf /opt/apps/flume-1.9/conf --conf-file
/data/lagoudw/conf/flume-log2hdfs3.conf -name a1 -
Dflume.root.logger=INFO,console
```

2、生成数据（文件大小约640M，100W条事件日志）

```
cd /data/lagoudw/jars

java -cp data-generator-1.1-SNAPSHOT-jar-with-dependencies.jar
com.lagou.ecommerce.AppEvent 1000000 2020-08-02 >
/data/lagoudw/logs/event/events0802.log
```

3、数据采集完成后，检查HDFS结果

```
hdfs dfs -ls /user/data/logs/event
```

第3节 ODS层建表和数据加载

```
drop table if exists ods.ods_log_event;
CREATE EXTERNAL TABLE ods.ods_log_event(`str` string)
PARTITIONED BY (`dt` string)
STORED AS TEXTFILE
LOCATION '/user/data/logs/event';
```

/data/lagoudw/script/advertisement/ods_load_event_log.sh

```
#!/bin/bash

source /etc/profile

if [ -n "$1" ]
then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

sql="
alter table ods.ods_log_event add partition (dt='$do_date');
"

hive -e "$sql"
```

第4节 DWD层建表和数据加载

ODS：分区；事件的主要信息在json串中(json数组)，公共信息在另外一个json串中；

ODS => **解析json，从json串中，提取jsonArray数据；将公共信息从json串中解析出来** => 所有事件的明细

所有事件的明细，包括：

- 分区
- 事件(json串)
- 公共信息字段

所有事件的明细 => **广告json串解析** => 广告事件的明细

广告事件的明细：

- 分区
- 广告信息字段
- 公共信息字段

4.1、DWD层建表

```
-- 所有事件明细
drop table if exists dwd.dwd_event_log;
CREATE EXTERNAL TABLE dwd.dwd_event_log(
`device_id` string,
`uid` string,
`app_v` string,
`os_type` string,
`event_type` string,
`language` string,
`channel` string,
`area` string,
`brand` string,

`name` string,
`event_json` string,
`report_time` string)
PARTITIONED BY (`dt` string)
stored as parquet;

-- 与广告点击明细
drop table if exists dwd.dwd_ad;
CREATE TABLE dwd.dwd_ad(
  `device_id` string,
  `uid` string,
  `app_v` string,
  `os_type` string,
```

```
  `event_type` string,
  `language` string,
  `channel` string,
  `area` string,
  `brand` string,
  `report_time` string,
  `duration` int,
  `ad_action` int,
  `shop_id` int,
  `ad_type` int,
  `show_style` smallint,
  `product_id` int,
  `place` string,
  `sort` int,
  `hour` string
)
PARTITIONED BY (`dt` string)
stored as parquet;
```

4.2、事件json串解析

内建函数、UDF、SerDe（json是所有的信息）

详细内容参见 第三部分 电商分析之--会员活跃度 => 第4节 json数据处理 => 使用UDF（处理jsonArray）

```
package cn.lagou.dw.hive.udf;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONArray;
import com.alibaba.fastjson.JSONException;
import com.alibaba.fastjson.JSONObject;
import com.google.common.base.Strings;
import org.apache.hadoop.hive.ql.exec.UDF;
import org.junit.Test;

import java.util.ArrayList;

public class ParseJsonArray extends UDF {
    public ArrayList<String> evaluate(String jsonStr) {
        // 传入空字符串，返回null
        if (Strings.isNullOrEmpty(jsonStr)){
            return null;
```

```
        }

        try{
            // 获取jsonArray
            JSONArray jsonArray = JSON.parseArray(jsonStr);
            ArrayList<String> lst = new ArrayList<>();
            for(Object o: jsonArray) {
                lst.add(o.toString());
            }

            return lst;
        }catch (JSONException e){
            return null;
        }
    }

    @Test
    public void JunitParseJsonArray() {
        String jsonStr = "
[{\"name\":\"goods_detail_loading\",\"json\":
{\"entry\":\"1\",\"goodsid\":\"0\",\"loading_time\":\"93\",\"acti
on\":\"3\",\"staytime\":\"56\",\"showtype\":\"2\"},\"time\":15963
43881690},{\"name\":\"loading\",\"json\":
{\"loading_time\":\"15\",\"action\":\"3\",\"loading_type\":\"3\",
\"type\":\"1\"},\"time\":1596356988428},
{\"name\":\"notification\",\"json\":
{\"action\":\"1\",\"type\":\"2\"},\"time\":1596374167278},
{\"name\":\"favorites\",\"json\":
{\"course_id\":1,\"id\":0,\"userid\":0},\"time\":1596350933962}]"
;
        ArrayList<String> result = evaluate(jsonStr);
        System.out.println(result.size());
        System.out.println(JSON.toJSONString(result));
    }
}
```

4.3、DWD层数据加载

主要功能：解析json串；得到全部的事件日志

/data/lagoudw/script/advertisement/dwd_load_event_log.sh

```
#!/bin/bash
```

```
source /etc/profile

if [ -n "$1" ] ;then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

sql="
use dwd;
add jar /data/lagoudw/jars/cn.lagou.dw-1.0-SNAPSHOT-jar-with-
dependencies.jar;
create temporary function json_array as
'cn.lagou.dw.hive.udf.ParseJsonArray';

with
tmp_start as
(
  select split(str, ' ')[7] as line
    from ods.ods_log_event
   where dt='$do_date'
)

insert overwrite table dwd.dwd_event_log
PARTITION (dt='$do_date')
select
device_id,
uid,
app_v,
os_type,
event_type,
language,
channel,
area,
brand,

get_json_object(k,'$.name') as name,
get_json_object(k,'$.json') as json,
get_json_object(k,'$.time') as time
from
(
    select
    get_json_object(line,'$.attr.device_id') as device_id,
    get_json_object(line,'$.attr.uid') as uid,
    get_json_object(line,'$.attr.app_v')  as app_v,
```

```
    get_json_object(line,'$.attr.os_type') as os_type,
    get_json_object(line,'$.attr.event_type') as event_type,
    get_json_object(line,'$.attr.language') as language,
    get_json_object(line,'$.attr.channel') as channel,
    get_json_object(line,'$.attr.area') as area,
    get_json_object(line,'$.attr.brand') as brand,
    get_json_object(line,'$.lagou_event') as lagou_event
from tmp_start
) A lateral view explode(json_array(lagou_event)) B as k
"

hive -e "$sql"
```

从全部的事件日志中获取广告点击事件：

/data/lagoudw/script/advertisement/dwd_load_ad_log.sh

```
#!/bin/bash

source /etc/profile

if [ -n "$1" ] ;then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

sql="
insert overwrite table dwd.dwd_ad
PARTITION (dt='$do_date')
select
    device_id,
    uid,
    app_v,
    os_type,
    event_type,
    language,
    channel,
    area,
    brand,
    report_time,
    get_json_object(event_json,'$.duration') ,
    get_json_object(event_json,'$.ad_action') ,
    get_json_object(event_json,'$.shop_id') ,
```

```
    get_json_object(event_json,'$.ad_type'),
    get_json_object(event_json,'$.show_style'),
    get_json_object(event_json,'$.product_id'),
    get_json_object(event_json,'$.place'),
    get_json_object(event_json,'$.sort'),

    from_unixtime(ceil(report_time/1000), 'HH')
from dwd.dwd_event_log
where dt='$do_date' and name='ad';
"

hive -e "$sql"
```

日志 => Flume => ODS => 清洗、转换 => 广告事件详细信息

第5节 广告点击次数分析

5.1 需求分析

广告：ad

- action。用户行为；0 曝光；1 曝光后点击；2 购买
- duration。停留时长
- shop_id。商家id
- event_type。"ad"
- ad_type。格式类型；1 JPG；2 PNG；3 GIF；4 SWF
- show_style。显示风格，0 静态图；1 动态图
- product_id。产品id
- place。广告位置；首页=1，左侧=2，右侧=3，列表页=4
- sort。排序位置

公共字段

分时统计：

曝光次数、不同用户id数（公共信息中的uid）、不同用户数(公共信息中的device_id)

点击次数、不同用户id数、不同用户数(device_id)

购买次数、不同用户id数、不同用户数(device_id)

DWD => DWS(不需要) => ADS；在某个分析中不是所有的层都会用到

5.2、创建ADS层表

```
drop table if exists ads.ads_ad_show;
create table ads.ads_ad_show(
    cnt bigint,
    u_cnt bigint,
    device_cnt bigint,
    ad_action tinyint,
    hour string
) PARTITIONED BY (`dt` string)
row format delimited fields terminated by ',';
```

5.3、加载ADS层数据

/data/lagoudw/script/advertisement/ads_load_ad_show.sh

```
#!/bin/bash

source /etc/profile

if [ -n "$1" ] ;then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

sql="
insert overwrite table ads.ads_ad_show
partition (dt='$do_date')
select count(1),
       count(distinct uid),
       count(distinct device_id),
       ad_action,
       hour
  from dwd.dwd_ad
 where dt='$do_date'
group by ad_action, hour
"
```

```
hive -e "$sql"
```

第6节 漏斗分析(点击率购买率)

6.1、需求分析

分时统计:

点击率 = 点击次数 / 曝光次数

购买率 = 购买次数 / 点击次数

6.2、创建ADS层表

```
drop table if exists ads.ads_ad_show_rate;
create table ads.ads_ad_show_rate(
    hour string,
    click_rate double,
    buy_rate double
) PARTITIONED BY (`dt` string)
row format delimited fields terminated by ',';
```

15075 15075 15075 0 01 2020-08-02 4349 4349 4349 1 01 2020-08-02 1245 1245 1245 2 01 2020-08-02

15075 4349 1245 01 2020-08-02

曝光 点击 购买 时间(HH)

行转列

```
-- 方法一
select sum(case when ad_action='0' then cnt end) show_cnt,
       sum(case when ad_action='1' then cnt end) click_cnt,
       sum(case when ad_action='2' then cnt end) buy_cnt,
       hour
  from ads.ads_ad_show
 where dt='2020-08-02' and hour='01'
group by hour ;
```

```
-- 方法二
select max(case when ad_action='0' then cnt end) show_cnt,
       max(case when ad_action='1' then cnt end) click_cnt,
       max(case when ad_action='2' then cnt end) buy_cnt,
       hour
  from ads.ads_ad_show
 where dt='2020-08-02' and hour='01'
group by hour ;
```

6.2、加载ADS层数据

/data/lagoudw/script/advertisement/ads_load_ad_show_rate.sh

```
#!/bin/bash

source /etc/profile

if [ -n "$1" ] ;then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

sql="
with tmp as(
select max(case when ad_action='0' then cnt end) show_cnt,
       max(case when ad_action='1' then cnt end) click_cnt,
       max(case when ad_action='2' then cnt end) buy_cnt,
       hour
  from ads.ads_ad_show
 where dt='$do_date'
group by hour
)
insert overwrite table ads.ads_ad_show_rate
partition (dt='$do_date')
select hour,
       click_cnt / show_cnt as click_rate,
       buy_cnt / click_cnt as buy_rate
  from tmp;
"

hive -e "$sql"
```

第7节 广告效果分析

7.1、需求分析

活动曝光效果评估：

行为(曝光、点击、购买)、时间段、广告位、商品，统计对应的次数

时间段、广告位、商品，曝光次数最多的前100个

###

7.2、创建ADS层表

```
drop table if exists ads.ads_ad_show_place;
create table ads.ads_ad_show_place(
    ad_action tinyint,
    hour string,
    place string,
    product_id int,
    cnt bigint
)PARTITIONED BY (`dt` string)
row format delimited fields terminated by ',';

drop table if exists ads.ads_ad_show_place_window;
create table ads.ads_ad_show_place_window(
    hour string,
    place string,
    product_id int,
    cnt bigint,
    rank int
)PARTITIONED BY (`dt` string)
row format delimited fields terminated by ',';
```

7.2、加载ADS层数据

/data/lagoudw/script/advertisement/ads_load_ad_show_page.sh

```
#!/bin/bash

source /etc/profile
```

```
if [ -n "$1" ] ;then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

sql="
insert overwrite table ads.ads_ad_show_place
partition (dt='$do_date')
select ad_action,
       hour,
       place,
       product_id,
       count(1)
  from dwd.dwd_ad
 where dt='$do_date'
group by ad_action, hour, place, product_id;
"

hive -e "$sql"
```

/data/lagoudw/script/advertisement/ads_load_ad_show_page_window.sh

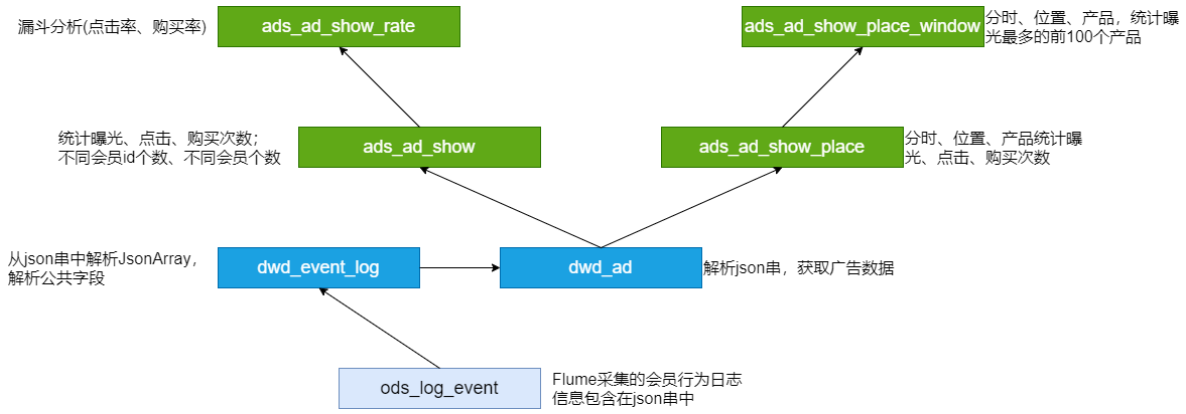
```
#!/bin/bash

source /etc/profile

if [ -n "$1" ] ;then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

sql="
insert overwrite table ads.ads_ad_show_place_window
partition (dt='$do_date')
select *
  from (
        select hour,
               place,
               product_id,
               cnt,
```

```
                row_number() over (partition by hour, place,
product_id order by cnt desc) rank
           from ads.ads_ad_show_place
          where dt='$do_date' and ad_action='0'
        ) t
 where rank <= 100
"

hive -e "$sql"
```

小结：分析简单，没有DWS层

Flume、json解析在会员分析讲解

第8节 广告分析小结



脚本调用次序:**

```
ods_load_event_log.sh
dwd_load_event_log.sh
dwd_load_ad_log.sh

ads_load_ad_show.sh
ads_load_ad_show_rate.sh
ads_load_ad_show_page.sh
ads_load_ad_show_page_window.sh
```

第9节 ADS层数据导出（DataX）

步骤：

- 在MySQL创建对应的表
- 创建配置文件（json）
- 执行命令，使用json配置文件；测试
- 编写执行脚本（shell）
- shell脚本的测试

1、MySQL 建表

```
drop table if exists dwads.ads_ad_show_place;
create table dwads.ads_ad_show_place(
    ad_action tinyint,
    hour varchar(2),
    place varchar(20),
    product_id int,
    cnt int,
    dt varchar(10)
);
```

2、创建配置文件

/data/lagoudw/script/advertisement/ads_ad_show_place.json

```
{
    "job":{
        "setting":{
            "speed":{
                "channel":1
            }
        },
        "content":[
            {
                "reader":{
                    "name":"hdfsreader",
                    "parameter":{

 "path":"/lagou/hive/warehouse/ads.db/ads_ad_show_place/dt=$do_da
te/*",
                        "defaultFS":"hdfs://hadoop1:9000",
                        "column":[
                            {
                                "index":0,
                                "type":"string"
```

```
                            },
                            {
                                "index":1,
                                "type":"string"
                            },
                            {
                                "index":2,
                                "type":"string"
                            },
                            {
                                "index":3,
                                "type":"string"
                            },
                            {
                                "index":4,
                                "type":"string"
                            },
                            {
                                "type":"string",
                                "value":"$do_date"
                            }
                        ],
                        "fileType":"text",
                        "encoding":"UTF-8",
                        "fieldDelimiter":","
                    }
                },
                "writer":{
                    "name":"mysqlwriter",
                    "parameter":{
                        "writeMode":"insert",
                        "username":"hive",
                        "password":"12345678",
                        "column":[
                            "ad_action",
                            "hour",
                            "place",
                            "product_id",
                            "cnt",
                            "dt"
                        ],
                        "preSql":[
                            "delete from ads_ad_show_place where
dt='$do_date'"
                        ],
```

```
                        "connection":[
                            {

 "jdbcUrl":"jdbc:mysql://hadoop2:3306/dwads?
useUnicode=true&characterEncoding=utf-8",
                                "table":[
                                    "ads_ad_show_place"
                                ]
                            }
                        ]
                    }
                }
            }
        ]
    }
}
```

3、执行命令(测试)

```
python /data/modules/datax/bin/datax.py -p "-Ddo_date=2020-08-02"
/data/lagoudw/script/advertisement/ads_ad_show_place.json
```

4、编写脚本

/data/lagoudw/script/advertisement/ads_ad_show_place.sh

```
#!/bin/bash

source /etc/profile
JSON=/data/lagoudw/script

if [ -n "$1" ] ;then
    do_date=$1
else
    do_date=`date -d "-1 day" +%F`
fi

python $DATAX_HOME/bin/datax.py -p "-Ddo_date=$do_date"
$JSON/advertisement/ads_ad_show_place.json
```

5、执行脚本

```
sh /data/lagoudw/script/advertisement/ads_ad_show_place.sh 2020-08-02
```

第10节 高仿日志数据测试

10.1、数据采集

- 1000W左右日活用户
- 按 30条日志 / 人天，合计3亿条事件日志
- 每条日志 650字节 左右
- 总数据量大概在180G
- 采集数据时间约2.5小时

1、清理环境

2、启动Flume：

```
nohup flume-ng agent --conf /opt/apps/flume-1.9/conf --conf-file /data/lagoudw/conf/flume-log2hdfs4.conf -name a1 -Dflume.root.logger=INFO,console &
```

日志文件很大，可以将hdfs文件滚动设置为10G甚至更大

3、写日志

```
cd /data/lagoudw/jars

nohup java -cp data-generator-1.1-SNAPSHOT-jar-with-dependencies.jar com.lagou.ecommerce.AppEvent 300000000 2020-08-03 > /data/lagoudw/logs/event/eventlog0803.log &
```

10.2、执行脚本

```
sh ods_load_event_log.sh 2020-08-03
sh dwd_load_event_log.sh 2020-08-03
sh dwd_load_ad_log.sh 2020-08-03

sh ads_load_ad_show.sh 2020-08-03
sh ads_load_ad_show_rate.sh 2020-08-03
sh ads_load_ad_show_page.sh 2020-08-03
sh ads_load_ad_show_page_window.sh 2020-08-03
```

附录

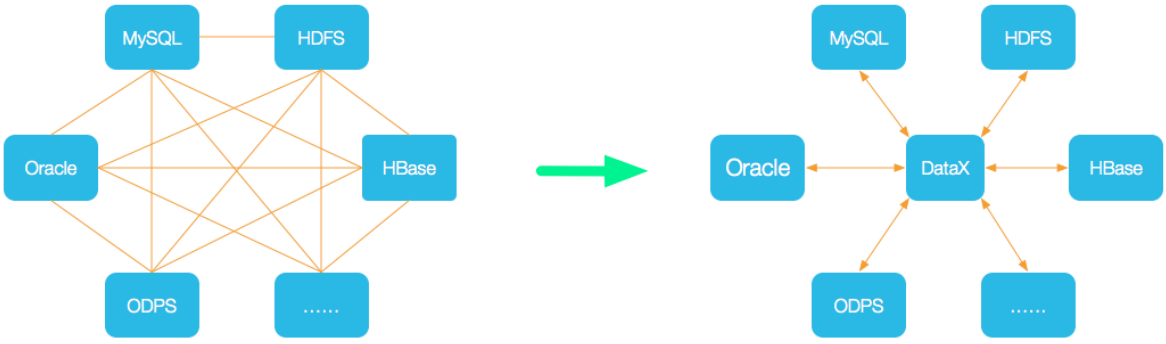
附录一 DataX快速入门

1.1、DataX概述及安装

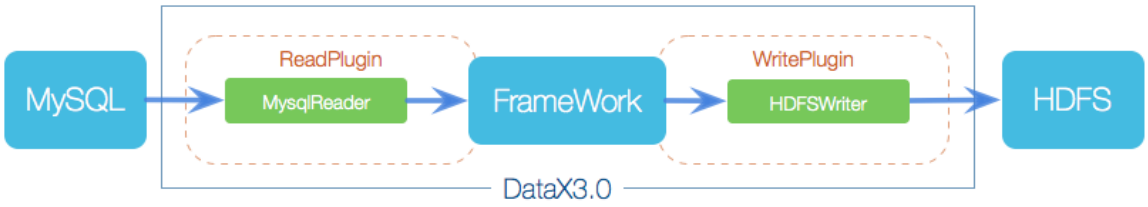
DataX 是阿里巴巴集团内被广泛使用的**离线数据**同步工具/平台，实现包括 MySQL、Oracle、SqlServer、Postgre、HDFS、Hive、ADS、HBase、TableStore(OTS)、MaxCompute(ODPS)、DRDS 等各种异构数据源之间高效的数据同步功能。

概述

为了解决异构数据源同步问题，DataX将复杂的网状的同步链路变成了星型数据链路，DataX作为中间传输载体负责连接各种数据源。当需要接入一个新的数据源的时候，只需要将此数据源对接到DataX，便能跟已有的数据源做到无缝数据同步。



DataX本身作为离线数据同步框架，采用Framework + plugin架构构建。将数据源读取和写入抽象成为Reader/Writer插件，纳入到整个同步框架中。

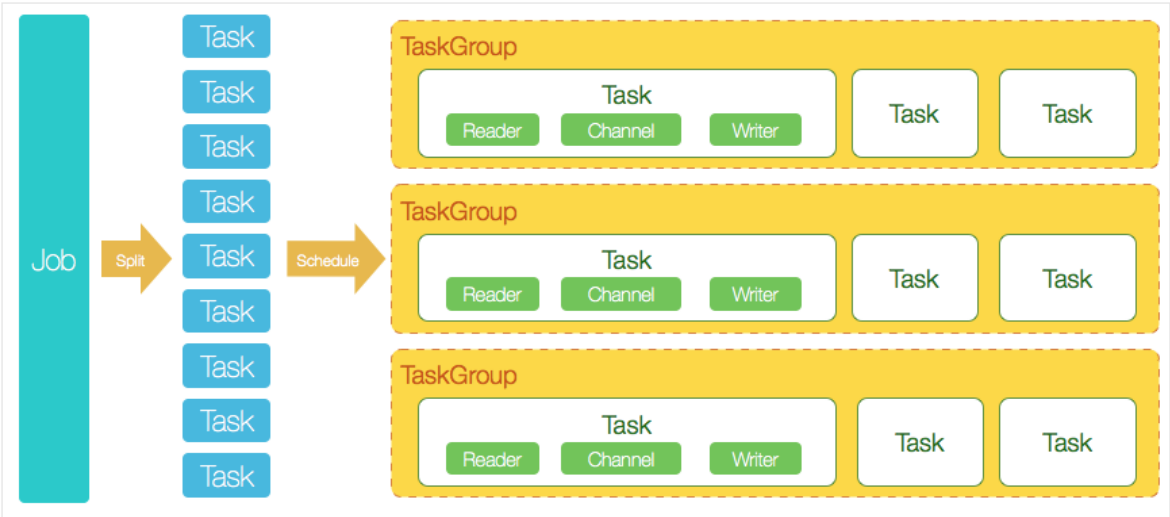


- Reader：数据采集模块，负责采集数据源的数据，将数据发送给Framework；
- Writer： 数据写入模块，负责不断向Framework取数据，并将数据写入到目的端；
- Framework：用于连接reader和writer，作为两者的数据传输通道，并处理缓冲，流控，并发，数据转换等核心技术问题。

经过几年积累，DataX目前已经有了比较全面的插件体系，主流的RDBMS数据库、NOSQL、大数据计算系统都已经接入。

DataX目前支持数据参见官网(https://github.com/alibaba/DataX/blob/master/introduction.md)。

DataX 3.0 开源版本支持单机多线程模式完成同步作业运行，本小节按一个DataX作业生命周期的时序图，从整体架构设计非常简要说明DataX各个模块相互关系。



核心模块

1. DataX完成单个数据同步的作业，称为Job。DataX接受到一个Job之后，将启动一个进程来完成整个作业同步过程。DataX Job模块是单个作业的中枢管理节点，承担了数据清理、子任务切分(将单一作业计算转化为多个子Task)、TaskGroup管理等功能。
2. DataX Job启动后，会根据不同的源端切分策略，将Job切分成多个小的Task(子任务)，以便于并发执行。**Task便是DataX作业的最小单元**，每一个Task都会负责一部分数据的同步工作。

3. 切分多个Task之后，DataX Job会调用Scheduler模块，根据配置的并发数据量，将拆分成的Task重新组合，组装成TaskGroup(任务组)。每一个TaskGroup负责以一定的并发运行完毕分配好的所有Task，**默认单个任务组的并发数量为5。**
4. 每一个Task都由TaskGroup负责启动，Task启动后，会固定启动Reader—>Channel—>Writer的线程来完成任务同步工作。
5. DataX作业运行起来之后， Job监控并等待多个TaskGroup模块任务完成，等待所有TaskGroup任务完成后Job成功退出。否则，异常退出，进程退出值非0。

DataX 3.0六大核心优势

- 可靠的数据质量监控
- 丰富的数据转换功能
- 精准的速度控制
- 强劲的同步性能
- 健壮的容错机制
- 极简的使用体验

详情见官网(https://github.com/alibaba/DataX/blob/master/introduction.md)

DataX安装配置

DataX官网：https://github.com/alibaba/DataX

前置条件：Linux、JDK(1.8以上，推荐1.8)、Python(推荐Python2.6.X)

DataX的安装比较简单基本上是开箱即用：

1、下载DataX工具包

http://datax-opensource.oss-cn-hangzhou.aliyuncs.com/datax.tar.gz

2、下载后解压至本地某个目录，进入bin目录，即可运行同步作业

配置环境变量 DATAX_HOME

```
$ cd  {YOUR_DATAX_HOME}/bin
$ python datax.py {YOUR_JOB.json}
$ python $DATAX_HOME/bin/datax.py $DATAX_HOME/job/job.json
```

自检脚本： python {YOUR_DATAX_HOME}/bin/datax.py {YOUR_DATAX_HOME}/job/job.json

3、测试

1.2、DataX使用案例

Reader插件和Writer插件

DataX3.0版本提供的Reader插件和Writer插件，每种读插件都有一种和多种切分策略:

```
"reader": {
    "name": "mysqlreader",      #从mysql数据库获取数据（也支持
sqlserverreader,oraclereader）
    "name": "txtfilereader",    #从本地获取数据
    "name": "hdfsreader",       #从hdfs文件、hive表获取数据
    "name": "streamreader",     #从stream流获取数据（常用于测试）
    "name": "httpreader",       #从http URL获取数据
    }
"writer": {
    "name":"hdfswriter",        #向hdfs,hive表写入数据
    "name":"mysqlwriter ",      #向mysql写入数据（也支持
sqlserverwriter,oraclewriter）
    "name":"streamwriter ",     #向stream流写入数据。（常用于测试）
    }
```

各种Reader插件、Writer插件的参考文档：https://github.com/alibaba/DataX

json配置文件模板

- 整个配置文件是一个job的描述;
- job下面有两个配置项，content和setting，其中content用来描述该任务的源和目的端的信息，setting用来描述任务本身的信息;
- content又分为两部分，reader和writer，分别用来描述源端和目的端的信息;
- setting中的speed项表示同时起几个并发执行该任务;

job的基本配置

```
{
    "job": {
        "content": [{
            "reader": {
                "name": "",
                "parameter": {}
            },
```

```
            "writer": {
                "name": "",
                "parameter": {}
            }
        }],
        "setting": {
            "speed": {},
            "errorLimit": {}
        }
    }
}
```

job Setting配置

```
{
    "job": {
        "content": [{
            "reader": {
                "name": "",
                "parameter": {}
            },
            "writer": {
                "name": "",
                "parameter": {}
            }
        }],
        "setting": {
            "speed": {
                "channel": 1,
                "byte": 104857600
            },
            "errorLimit": {
                "record": 10,
                "percentage": 0.05
            }
        }
    }
}
```

- job.setting.speed(流量控制)

  Job支持用户对速度的自定义控制，channel的值可以控制同步时的并发数，byte的值可以控制同步时的速度。

- job.setting.errorLimit(脏数据控制)

  Job支持用户对于脏数据的自定义监控和告警，包括对脏数据最大记录数阈值（record值）或者脏数据占比阈值（percentage值），当Job传输过程出现的脏数据大于用户指定的数量/百分比，DataX Job报错退出。

应用案例

Stream ===> Stream。stream reader/writer 都用于测试

```
{
    "job": {
        "content": [{
            "reader": {
                "name": "streamreader",
                "parameter": {
                    "sliceRecordCount": 10,
                    "column": [{
                            "type": "String",
                            "value": "hello DataX"
                        },
                        {
                            "type": "string",
                            "value": "DataX Stream To Stream"
                        },
                        {
                            "type": "string",
                            "value": "数据迁移工具"
                        }
                    ]
                }
            },
            "writer": {
                "name": "streamwriter",
                "parameter": {
                    "encoding": "GBK",
                    "print": true
                }
            }
        }],
        "setting": {
            "speed": {
                "channel": 2
            }
```

```
        }
    }
}
```

```
python $DATAX_HOME/bin/datax.py
/data/lagoudw/json/stream2stream.json
```

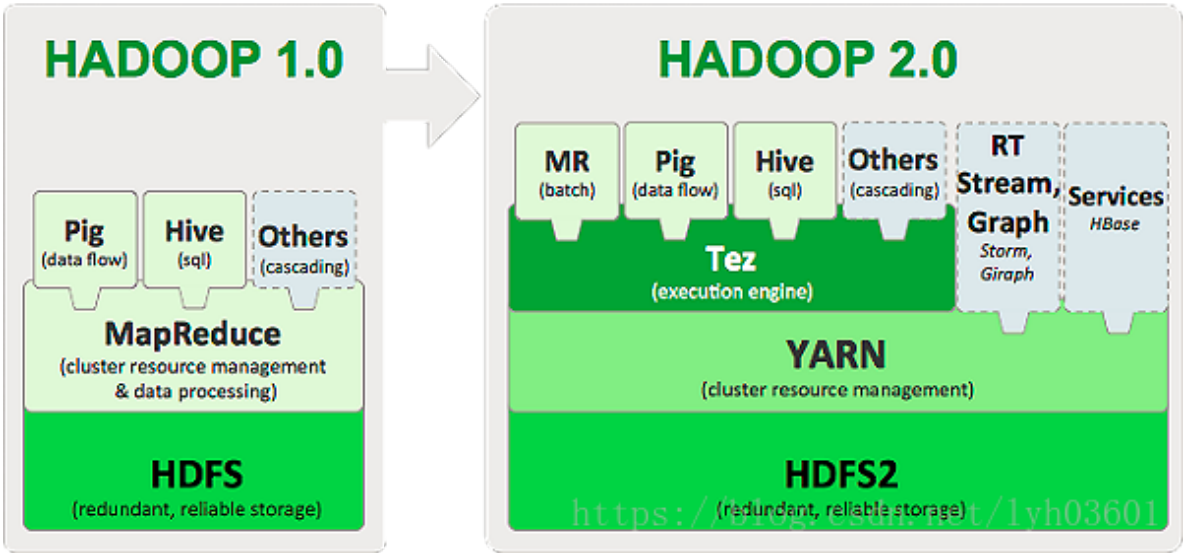
附录二 Hive on Tez

Hortonworks在2014年左右发布了Stinger Initiative，并进行社区分享，为的是让Hive支持更多SQL，并实现更好的性能。

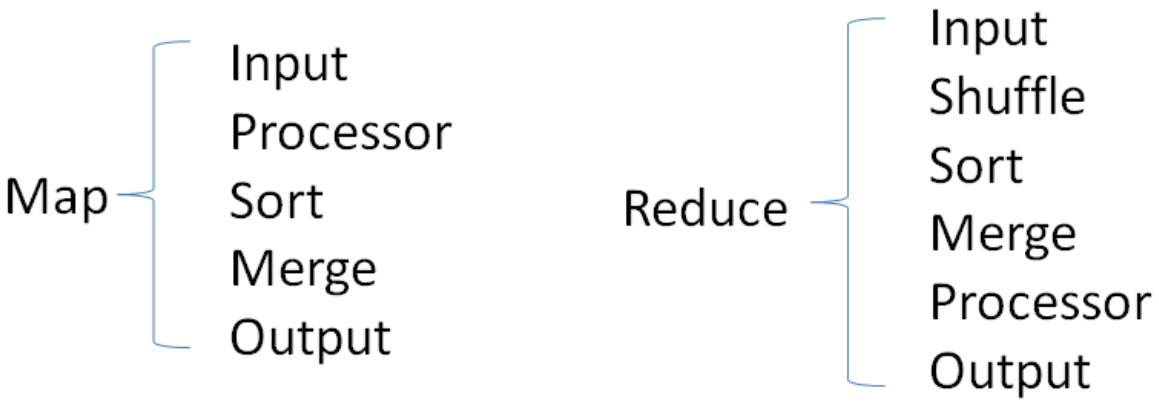
- 让Hive的查询功能更强大。增加类似OVER子句的分析功能，支持WHERE子查询，以及调整Hive的样式系统更多的符合标准的SQL模型；
- 优化Hive的请求执行计划，增加 Task 每秒处理记录的数量；
- 引入新的列式文件格式（ORC文件），提供一种更现代、高效和高性能的方式来储存Hive数据；
- 引入新的runtime框架——Tez，消除Hive的延迟以及吞吐量限制。Tez通过消除不必要的task、障碍同步和对HDFS的读写作业来优化Hive job；

2.1、Tez概述

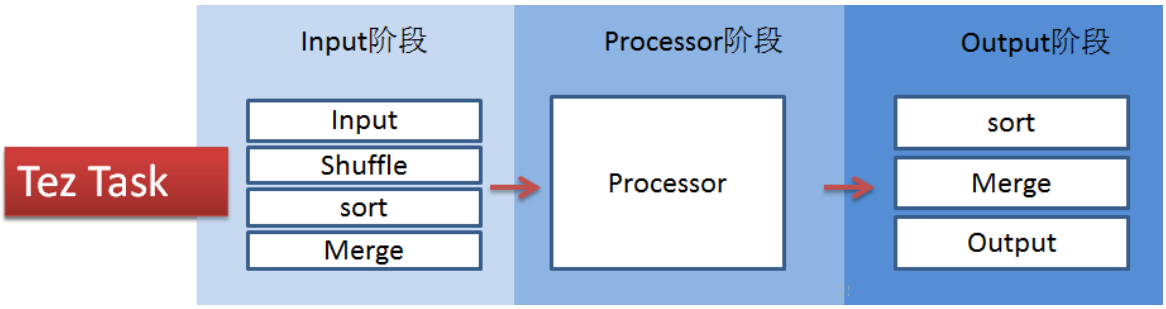
Tez是Apache开源的支持**DAG**（有向无环图）作业的计算框架，是支持Hadoop 2.x的重要引擎。它源于MapReduce框架，核心思想是将Map和Reduce两个操作进一步拆分，分解后的元操作可以任意灵活组合，产生新的操作，这些操作经过一些控制程序组装后，可形成一个大的DAG作业。



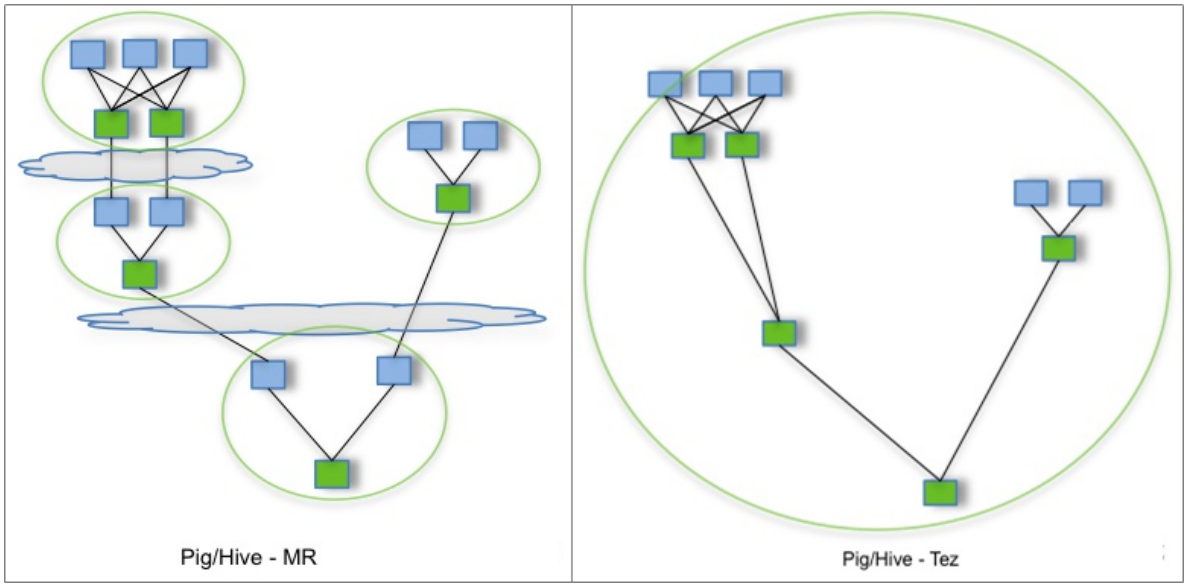
Tez将Map task和Reduce task进一步拆分为如下图所示：



Tez的task由Input、processor、output阶段组成，可以表达所有复杂的map、reduce操作，如下图：



Tez可以将多个有依赖的作业转换为一个作业（只需写一次HDFS，中间环节较少），从而大大提升DAG作业的性能。Tez已被Hortonworks用于Hive引擎的优化，经测试一般小任务比Hive MR 的2-3倍速度左右，大任务7-10倍左右，根据情况不同可能不一样。




```
Hive-on-MR is deprecated in Hive 2 and may not be available in
the future versions. Consider using a different execution engine
(i.e. spark, tez) or using Hive 1.X releases.
```

Tez+Hive仍采用 MapReduce 计算框架，但对DAG的作业依赖关系进行了裁剪，并将多个小作业合并成一个大作业，不仅减少了计算量，而且写HDFS次数也大大减少。

2.2、安装部署

1、下载软件包：apache-tez-0.9.2-bin.tar.gz

2、解压缩

```
tar -zxvf apache-tez-0.9.0-bin.tar.gz
cd apache-tez-0.9.0-bin/share
```

3、将tez的压缩包放到到hdfs上

```
hdfs dfs -mkdir -p /user/tez
hdfs dfs -put tez.tar.gz /user/tez
```

4、$HADOOP_HOME/etc/hadoop/ 下创建 tez-site.xml 文件，做如下配置：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <!-- 指定在hdfs上的tez包文件 -->
    <property>
        <name>tez.lib.uris</name>
        <value>hdfs://hadoop1:9000/user/tez/tez.tar.gz</value>
    </property>
</configuration>
```

保存后将文件复制到集群所有节点

5、增加客户端节点的配置(/etc/profile)

```
export TEZ_CONF_DIR=$HADOOP_CONF_DIR
export TEZ_JARS=/opt/apps/tez/*:/opt/apps/tez/lib/*
export HADOOP_CLASSPATH=$TEZ_CONF_DIR:$TEZ_JARS:$HADOOP_CLASSPATH
```

6、Hive设置Tez执行

```
hive> set hive.execution.engine=tez;
```

```
--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 11.53 s
--------------------------------------------------------------------------------
```

7、如果想默认使用Tez，可在$HIVE_HOME/conf目录下hive-site.xml 中增加

```
<property>
    <name>hive.execution.engine</name>
    <value>tez</value>
</property>
```