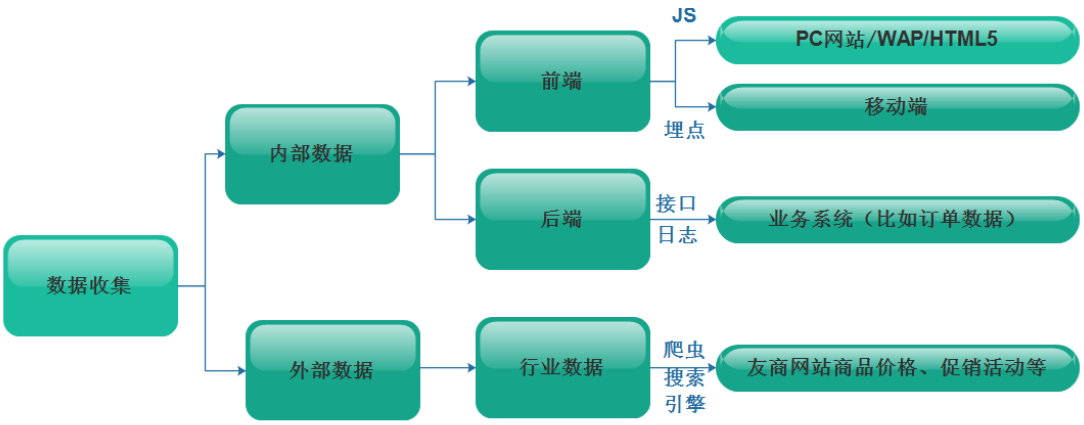
一、 课程计划

目录

二、 多彩缤纷数据源

典型的数据分析系统，要分析的数据种类其实是比较丰富的。依据来源可大体分为以下几个部分：



图：数据分析系统数据来源

1. 业务系统数据

业务系统产生的数据是不可忽视的，比如电商网站，大量的订单数据看似杂乱无章，实则蕴含潜在的商业价值，可以从中分析进而进行商业推广，产品推荐等。

另一角度来看，业务系统数据获取成本低、方式容易，属于公司内部范畴。业务系统的数据一般保存在关系型数据库当中。获取形式有：

接口调用：直接获取业务系统数据库的数据，但是要注意不能影响业务系统数据库的性能，比如大量获取数据增大数据库读数据压力。

数据库 dump:非高峰时段，或者在数据库从库上 dump 出全部数据。一般企业中会定时进行数据库的备份、导出工作，那么就可以共享使用这些数据。

比如 MySQL 数据库，使用 mysqldump 工具就可以进行数据库的导出。

```
mysqldump -uroot -pPassword [database name] [dump file]
```

mysqldump 命令将数据库中的数据备份成一个文本文件。表的结构和表中的数据将存储在生成的文本文件中。

2. 爬虫数据

在进行网站数据分析的时候，除了内部数据之外，还有一部分数据是我们不能够忽视的。那就是所谓的外部数据。当然这是相对公司网站来说的。拥有了外部数据可以更好的帮助我们进行数据分析。

爬虫（Web crawler），是指一种按照一定的规则，自动地抓取万维网信息的程序或者脚本。它们被广泛用于互联网搜索引擎或其他类似网站，可以自动采集所有其能够访问到的页面内容，以获取或更新这些网站的内容和检索方式。

电子商务行业最初的爬虫需求来源于比价。这是某些电商网站的核心业务。大家如果买商品的时候，是一个价格敏感型用户的话，很可能会使用比价功能。毫无悬念，会使用爬虫技术来爬取所有相关电商的价格。

当然，这并不意味着大家喜欢被爬取。于是需要通过技术手段来做反爬虫。

三、 数据的管理

1. 数据文件管理

随着技术和业务的发展壮大，企业中产生的数据种类越来越多，数据量也越来越大。如何对数据进行有效的组织、存储、管理、检索、维护，将会显得越来越重要。在企业内部很多时候还涉及数据的跨部门存储与调用。因此，进行数据的管理就显得特别重要，也越来越受到企业的重视。

数据一般会以文件的形式存在，比如文本文件、视频文件、音频文件等。那么数据的管理就转化为对这些数据文件的管理。

文件管理的真谛在于方便保存和迅速提取，所有的数据文件将通过某种属性（比如业务、时间）分类被很好地组织起来，放在最能方便找到的地方。解决这个问题目前最理想的方法就是**分类管理**。

从每一个文件夹的建立，我们都要按照数据文件的属性，分为大大小小、多个层级的文件夹，建立合理的文件保存架构。此外所有的文件、文件夹，都要规范化地命名，并放入最合适的文件夹中。

当然企业中可不会像个人使用电脑文件资源管理器那样去管理文件，而是使用文件服务器去管理文件。在一些有条件的企业，会部署一个文件服务器，来统一管理文件。这样可以对企业管理带来如下好处：

定时集中对文件进行备份；

可以统一制定文件安全访问权限策略；

可以统一进行文件服务器防病毒管理。

常见的文件服务有以下这几种：

ftp 文件服务

Samba 文件服务

NFS 文件服务

1.1. FTP 文件服务

FTP 是一个文件传输的协议，采用 Client/Server 架构。用户可以通过各种不同的 FTP 客户端程序，借助 FTP 协议，来连接 FTP 服务器，以上传或者下载文件。它使用两个连接与客户端通信：

命令连接：用于传输文件管理类命令，在客户端连接后会始终在线；

数据连接：用于传输文件数据，此连接会按序创建。

Linux 中常用的 FTP 客户端软件有 lftp, ftp, lftpget, wget, curl 等。Windows 中可以使用浏览器，资源管理器或 Filezilla 等软件。

1.2. NFS 文件服务

NFS 是 Network File System 的缩写，即网络文件系统。它允许网络中的计算机之间通过 TCP/IP 网络共享资源。NFS 的基本原则是“容许不同的客户端及服务端通过一组 RPC 分享相同的文件系统”，它是独立于操作系统，容许不同硬件及操作系统的系统共同进行文件的分享。

NFS 在文件传送或信息传送过程中依赖于 RPC 协议。RPC，远程过程调用 (Remote Procedure Call) 是能使客户端执行其他系统中程序的一种机制。NFS 本身是没有提供信息传输的协议和功能的，但 NFS 却能让我们通过网络进行资料的分享，原因就在与 RPC,可以说 NFS 本身就是使用 RPC 的一个程序。可以这么理解 RPC 和 NFS 的关系：NFS 是一个文件系统，而 RPC 是负责信息的传输。

1.3. Samba 文件服务

SMB（Server Messages Block，信息服务块）是一种在局域网上共享文件和打印机的一种通信协议，它为局域网内的不同计算机之间提供文件及打印机等资源的共享服务。

Samba 是一组软件包,在 Linux 和 UNIX 系统上实现 SMB 协议的一个免费软件。Linux 操作系统提供了 Samba 服务，Samba 服务为两种不同的操作系统架起了一座桥梁，使 Linux 系统和 Windows 系统之间能够实现互相通信，这使得 Windows/Linux/Unix 间可以自由的进行文件共享。

2. 文件管理规范

为了更快速，更准确，更规范的进行数据文件管理，企业一般都会去制定相应的管理规范。从而使各方都按照这个规范去进行文件的存储、读取。规范着重于文件命名规则，以及一些校验性文件的描述。

例子：FTP 服务进行跨部门文件共享的相关规范。以数据库数据文件导出至 ftp 文件服务器共享为例，目录下会存在以下 3 种格式文件。仅供参考。

2.1. 接口新增数据文件

正常数据：文件类型标示_源系统数据库类型简写.源系统数据库.表名称_数据日期_重传序号.lzo

如正常数据为：

增量（上次导出之后的新数据）：

i_s.Peking.orders_20130711_000.lzo

全量（表中所有的数据）：

a_s.Peking.orders_20130711_000.lzo

2.2. 接口控制校验文件

正常数据：增全量标示_源系统数据库类型简写.源系统数据库.表名称_数据日期_重传序号.md5

如正常数据为：

增量:

i_s.peking.orders_20130711_000.md5

全量：

a_s.peking.orders_20130711_000.md5

控制校验文件的存在意义在于标识数据的完整性校验，预防上传丢失导致其他使用者获取不完整数据。

2.3. 接口表结构文件

正常数据：增全量标示_源系统数据库类型简写.源系统数据库.表名称_数据日期_重传序号.xml

如正常数据为：

增量:

i_s.peking.orders_20130711_000.xml

全量：

a_s.peking.orders_20130711_000.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<table type="hbase" database="credit" code="cds_courier_active">
<field type="string" code="pin"/>
<field type="string" code="d_seq_no"/>
<field type="string" code="d_cust_id"/>
<field type="string" code="d_mobile"/>
<field type="string" code="d_active_stcd"/>
<field type="string" code="d_active_result"/>
<field type="string" code="d_credit_amt"/>
<field type="string" code="d_active_tm"/>
<field type="string" code="d_seq_no_interface"/>
<field type="string" code="d_apply_channel"/>
<field type="string" code="d_apply_type"/>
<field type="string" code="d_oder_id"/>
<field type="string" code="d_name"/>
<field type="string" code="d_approve_times"/>
<field type="string" code="d_product_type"/>
</table>
```

表结构文件存在意义在于：便于数据的使用人员快速的了解本批次数据的大致内容，也方便后续回头对数据进行检测时作为依照。

3. 数据质量检测

数据质量是保证数据应用的基础，它的评估标准主要包括四个方面：完整性、一致性、准确性、及时性。评估数据是否达到预期设定的质量要求，就可以通过这四个方面来进行判断。

完整性指的是数据信息是否存在缺失的状况，数据缺失的情况可能是整个数据记录缺失，也可能是数据中某个字段信息的记录缺失。不完整数据的价值就会大大降低，也是数据质量最为基础的一项评估标准。

数据质量的完整性比较容易去评估，一般可以通过数据统计中的记录值和唯一值进行评估。例如，网站日志日访问量就是一个记录值，平时的日访问量在1000 左右，突然某一天降到 100 了，需要检查一下数据是否存在缺失了。再例如，网站统计地域分布情况的每一个地区名就是一个唯一值，我国包括了 32 个省和直辖市，如果统计得到的唯一值小于 32，则可以判断数据有可能存在缺失。

一致性是指数据是否遵循了统一的规范，数据集合是否保持了统一的格式。数据质量的一致性主要体现在数据记录的规范和数据是否符合逻辑。规范指的是，一项数据存在它特定的格式，例如手机号码一定是 11 位的数字，IP 地址一定是由 4 个 0 到 255 间的数字加上.组成的。逻辑指的是，多项数据间存在着固定的逻辑关系，例如 PV 一定是大于等于 UV 的，跳出率一定是在 0 到 1 之间的。

准确性是指数据记录的信息是否存在异常或错误。存在准确性问题的数据不仅仅只是规则上的不一致。最为常见的数据准确性错误如乱码。其次，异常的大或者小的数据也是不符合条件的数据。数据质量的准确性可能存在于个别记录，也可能存在于整个数据集，例如数量级记录错误。这类错误则可以使用最大值和最小值的统计量去审核。

一般数据都符合正态分布的规律，如果一些占比少的数据存在问题，则可以通过比较其他数量少的数据比例，来做出判断。

及时性是指数据从产生到可以查看的时间间隔，也叫数据的延时时长。及时性对于数据分析本身要求并不高，但如果数据分析周期加上数据建立的时间过长，就可能导致分析得出的结论失去了借鉴意义。

四、 数据仓库

1. 数据仓库的基本概念

数据仓库，英文名称为Data Warehouse，可简写为DW或DWH。数据仓库的目的是构建面向分析的集成化数据环境，为企业提供决策支持（Decision Support）。它出于分析性报告和决策支持目的而创建。

数据仓库本身并不“生产”任何数据，同时自身也不需要“消费”任何的数据，数据来源于外部，并且开放给外部应用，这也是为什么叫“仓库”，而不叫“工厂”的原因。

2. 数据仓库的主要特征

数据仓库是面向主题的（Subject-Oriented ）、集成的（Integrated）、非易失的（Non-Volatile）和时变的（Time-Variant ）数据集合，用以支持管理决策 。

2.1. 面向主题

传统数据库中，最大的特点是面向应用进行数据的组织，各个业务系统可能是相互分离的。而数据仓库则是面向主题的。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。

操作型处理（传统数据）对数据的划分并不适用于决策分析。而基于主题组织的数据则不同，它们被划分为各自独立的领域，每个领域有各自的逻辑内涵但互不交叉，在抽象层次上对数据进行完整、一致和准确的描述。一些主题相关的数据通常分布在多个操作型系统中。

2.2. 集成性

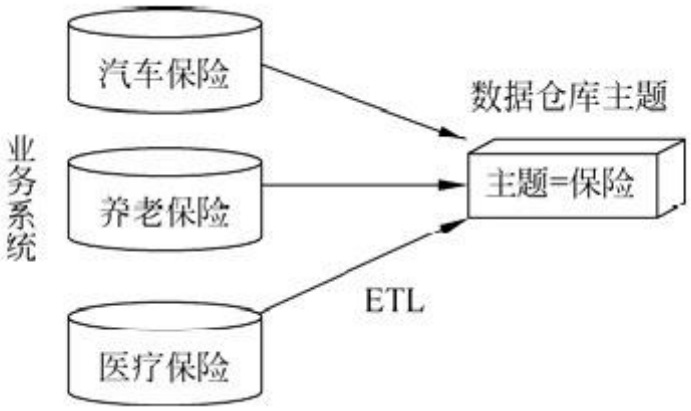
通过对分散、独立、异构的数据库数据进行抽取、清理、转换和汇总便得到了数据仓库的数据，这样保证了数据仓库内的数据关于整个企业的一致性。

数据仓库中的综合数据不能从原有的数据库系统直接得到。因此在数据进入数据仓库之前，必然要经过统一与综合，这一步是数据仓库建设中最关键、最复杂的一步，所要完成的工作有：

（1）要统一源数据中所有矛盾之处，如字段的同名异义、异名同义、单位不统一、字长不一致，等等。

（2）进行数据综合和计算。数据仓库中的数据综合工作可以在从原有数据库抽取数据时生成，但许多是在数据仓库内部生成的，即进入数据仓库以后进行综合生成的。

下图说明一个保险公司综合数据的简单处理过程，其中数据仓库中与“保险”主题有关的数据来自于多个不同的操作型系统。这些系统内部数据的命名可能不同，数据格式也可能不同。把不同来源的数据存储到数据仓库之前，需要去除这些不一致。



图：数据仓库的数据集成

2.3. 非易失性（不可更新性）

操作型数据库主要服务于日常的业务操作，使得数据库需要不断地对数据实时更新，以便迅速获得当前最新数据，不至于影响正常的业务运作。在数据仓库中只要保存过去的业务数据，不需要每一笔业务都实时更新数据仓库，而是根据

商业需要每隔一段时间把一批较新的数据导入数据仓库。

数据仓库的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。

数据非易失性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。数据仓库中一般有大量的查询操作，但修改和删除操作很少。因此，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。

2.4. 时变性

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。数据仓库的目的是通过分析企业过去一段时间业务的经营状况，挖掘其中隐藏的模式。虽然数据仓库的用户不能修改数据，但并不是说数据仓库的数据是永远不变的。分析的结果只能反映过去的情况，当业务变化后，挖掘出的模式会失去时效性。因此数据仓库的数据需要更新，以适应决策的需要。从这个角度讲，数据仓库建设是一个项目，更是一个过程 。数据仓库的数据随时间的变化表现在以下几个方面。

（1）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。

（2）操作型系统存储的是当前数据，而数据仓库中的数据是历史数据。

（3）数据仓库中的数据是按照时间顺序追加的，它们都带有时间属性。

3. 数据仓库与数据库区别

数据库与数据仓库的区别实际讲的是 OLTP 与 OLAP 的区别。

操作型处理，叫联机事务处理 OLTP（On-Line Transaction Processing,），也可以称面向交易的处理系统，它是针对具体业务在数据库联机的日常操作，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。

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

首先要明白，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。
- 数据库一般存储业务数据，数据仓库存储的一般是历史数据。
- 数据库设计是尽量避免冗余，一般针对某一业务应用进行设计，比如一张简单的 User 表，记录用户名、密码等简单数据即可，符合业务应用，但是不符合分析。数据仓库在设计是有意引入冗余，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。

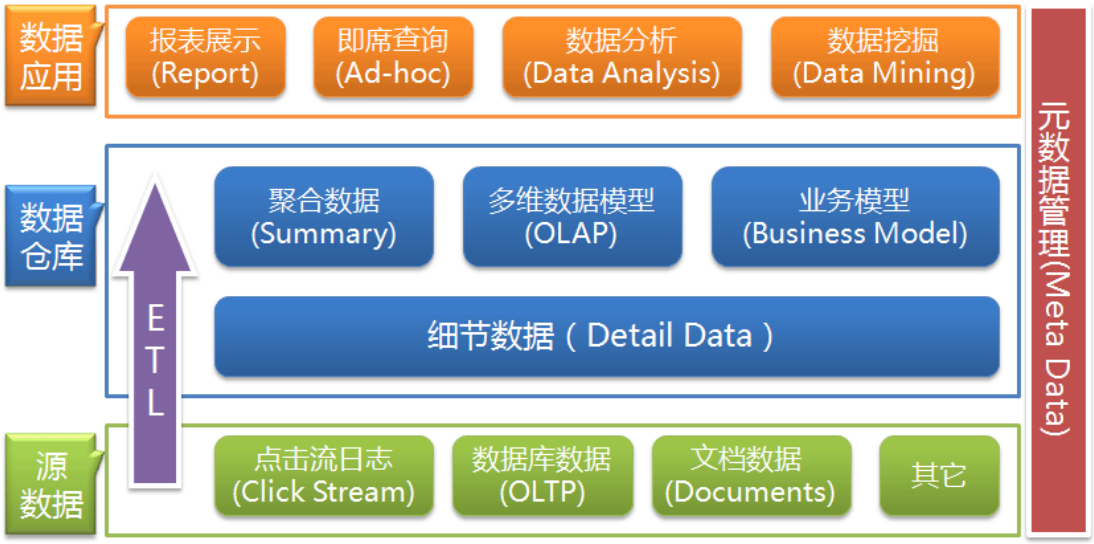
以银行业务为例。数据库是事务系统的数据平台，客户在银行做的每笔交易都会写入数据库，被记录下来，这里，可以简单地理解为用数据库记账。数据仓库是分析系统的数据平台，它从事务系统获取数据，并做汇总、加工，为决策者提供决策的依据。比如，某银行某分行一个月发生多少交易，该分行当前存款余额是多少。如果存款又多，消费交易又多，那么该地区就有必要设立 ATM 了。

显然，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”。

4. 数据仓库分层架构

按照数据流入流出的过程，数据仓库架构可分为三层——**源数据、数据仓库、数据应用。**



数据仓库的数据来源于不同的源数据，并提供多样的数据应用，数据自下而上流入数据仓库后向上层开放应用，而数据仓库只是中间集成化数据管理的一个平台。

- 源数据层（ODS）：此层数据无任何更改，直接沿用外围系统数据结构和数据，不对外开放；为临时存储层，是接口数据的临时存储区域，为后一步的数据处理做准备。
- 数据仓库层（DW）：也称为细节层，DW 层的数据应该是一致的、准确的、干净的数据，即对源系统数据进行了清洗（去除了杂质）后的数据。
- 数据应用层（DA 或 APP）：前端应用直接读取的数据源；根据报表、专题分析需求而计算生成的数据。

数据仓库从各数据源获取数据及在数据仓库内的数据转换和流动都可以认为是 ETL（抽取 Extra，转化 Transfer，装载 Load）的过程，ETL 是数据仓库的流水线，也可以认为是数据仓库的血液，它维系着数据仓库中数据的新陈代谢，而数据仓库日常的管理和维护工作的大部分精力就是保持 ETL 的正常和稳定。

为什么要对数据仓库分层？

用空间换时间，通过大量的预处理来提升应用系统的用户体验（效率），因此数据仓库会存在大量冗余的数据；不分层的话，如果源业务系统的业务规则发生变化将会影响整个数据清洗过程，工作量巨大。

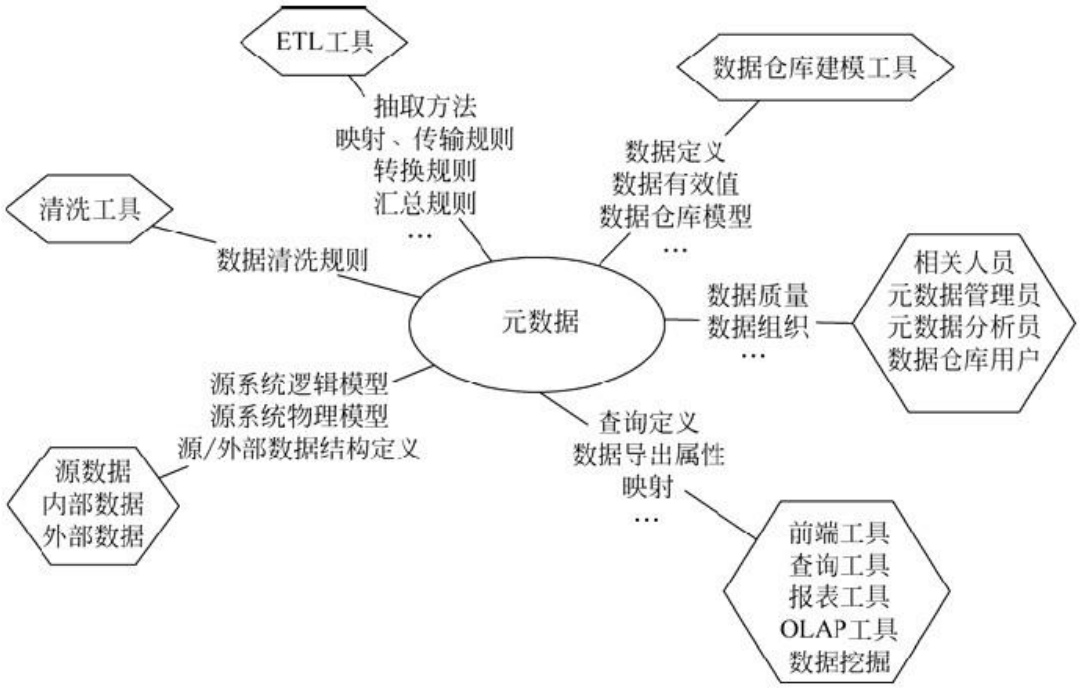
通过数据分层管理可以简化数据清洗的过程，因为把原来一步的工作分到了多个步骤去完成，相当于把一个复杂的工作拆成了多个简单的工作，把一个大的黑盒变成了一个白盒，每一层的处理逻辑都相对简单和容易理解，这样我们比较容易保证每一个步骤的正确性，当数据发生错误的时候，往往我们只需要局部调整某个步骤即可。

5. 数据仓库元数据管理

元数据（Meta Date），主要记录数据仓库中模型的定义、各层级间的映射关系、监控数据仓库的数据状态及 ETL 的任务运行状态。一般会通过元数据资料库（Metadata Repository）来统一地存储和管理元数据，其主要目的是使数据仓库的设计、部署、操作和管理能达成协同和一致。

元数据是数据仓库管理系统的重要组成部分，元数据管理是企业级数据仓库中的关键组件，贯穿数据仓库构建的整个过程，直接影响着数据仓库的构建、使用和维护。

- 构建数据仓库的主要步骤之一是 ETL。这时元数据将发挥重要的作用，它定义了源数据系统到数据仓库的映射、数据转换的规则、数据仓库的逻辑结构、数据更新的规则、数据导入历史记录以及装载周期等相关内容。数据抽取和转换的专家以及数据仓库管理员正是通过元数据高效地构建数据仓库。
- 用户在使用数据仓库时，通过元数据访问数据，明确数据项的含义以及定制报表。
- 数据仓库的规模及其复杂性离不开正确的元数据管理，包括增加或移除外部数据源，改变数据清洗方法，控制出错的查询以及安排备份等。



元数据可分为技术元数据和业务元数据。技术元数据为开发和管理数据仓库的 IT 人员使用，它描述了与数据仓库开发、管理和维护相关的数据，包括数据源信息、数据转换描述、数据仓库模型、数据清洗与更新规则、数据映射和访问权限等。而业务元数据为管理层和业务分析人员服务，从业务角度描述数据，包括商务术语、数据仓库中有什么数据、数据的位置和数据的可用性等，帮助业务人员更好地理解数据仓库中哪些数据是可用的以及如何使用。

由上可见，元数据不仅定义了数据仓库中数据的模式、来源、抽取和转换规则等，而且是整个数据仓库系统运行的基础，元数据把数据仓库系统中各个松散的组件联系起来，组成了一个有机的整体。

五、 Apache Hive

1. Hive 简介

1.1. 什么是 Hive

Hive 是基于 Hadoop 的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供类 SQL 查询功能。

本质是将 SQL 转换为 MapReduce 程序。

主要用途：用来做离线数据分析，比直接用 MapReduce 开发效率更高。

1.2. 为什么使用 Hive

直接使用 Hadoop MapReduce 处理数据所面临的问题：

人员学习成本太高

MapReduce 实现复杂查询逻辑开发难度太大

使用 Hive ：

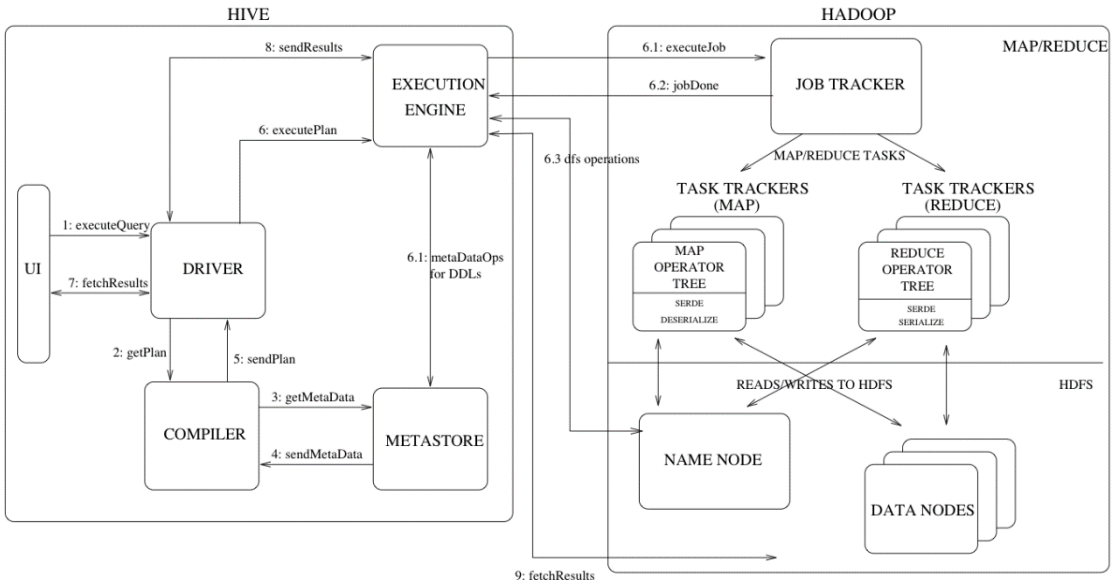
操作接口采用类 SQL 语法，提供快速开发的能力

避免了去写 MapReduce，减少开发人员的学习成本

功能扩展很方便

2. Hive 架构

2.1. Hive 架构图



2.2. Hive 组件

用户接口：包括 CLI、JDBC/ODBC、WebGUI。其中，CLI(command line interface)为 shell 命令行；JDBC/ODBC 是 Hive 的 JAVA 实现，与传统数据库 JDBC 类似；WebGUI 是通过浏览器访问 Hive。

元数据存储：通常是存储在关系数据库如 mysql/derby 中。Hive 将元数据存储在数据库中。Hive 中的元数据包括表的名字，表的列和分区及其属性，表的属性（是否为外部表等），表的数据所在目录等。

解释器、编译器、优化器、执行器:完成 HQL 查询语句从词法分析、语法分析、编译、优化以及查询计划的生成。生成的查询计划存储在 HDFS 中，并在随后有 MapReduce 调用执行。

2.3. Hive 与 Hadoop 的关系

Hive 利用 HDFS 存储数据，利用 MapReduce 查询分析数据。

3. Hive 与传统数据库对比

hive 用于海量数据的离线数据分析。

hive 具有 sql 数据库的外表，但应用场景完全不同，hive 只适合用来做批量数据统计分析。

更直观的对比请看下面这幅图：

	Hive	RDBMS
查询语言	HQL	SQL
数据存储	HDFS	Raw Device or Local FS
执行	MapReduce	Excutor
执行延迟	高	低
处理数据规模	大	小
索引	0.8版本后加入位图索引	有复杂的索引

4. Hive 数据模型

Hive 中所有的数据都存储在 HDFS 中，没有专门的数据存储格式

在创建表时指定数据中的分隔符，Hive 就可以映射成功，解析数据。

Hive 中包含以下数据模型：

db：在 hdfs 中表现为 hive.metastore.warehouse.dir 目录下一个文件夹

table：在 hdfs 中表现所属 db 目录下一个文件夹

external table：数据存放位置可以在 HDFS 任意指定路径

partition：在 hdfs 中表现为 table 目录下的子目录

bucket：在 hdfs 中表现为同一个表目录下根据 hash 散列之后的多个文件

5. Hive 安装部署

Hive 安装前需要安装好 JDK 和 Hadoop。配置好环境变量。

根据元数据存储的介质不同，分为下面两个版本，其中 derby 属于内嵌模式。实际生产环境中则使用 mysql 来进行元数据的存储。

内置 derby 版：

解压 hive 安装包

bin/hive 启动即可使用

缺点：不同路径启动 hive，每一个 hive 拥有一套自己的元数据，无法共享

mysql 版：

解压、修改配置文件

vi conf/hive-site.xml

配置 Mysql 元数据库信息

详细安装步骤查看《Hive 安装手册》

六、 Hive 基本操作

1. DDL 操作

1.1. 创建表

建表语法

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
  [(col_name data_type [COMMENT col_comment], ...)]
  [COMMENT table_comment]
  [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
  [CLUSTERED BY (col_name, col_name, ...)
  [SORTED BY (col_name [ASC|DESC], ...)] INTO num_buckets BUCKETS]
  [ROW FORMAT row_format]
  [STORED AS file_format]
  [LOCATION hdfs_path]
```

说明：

1、CREATE TABLE 创建一个指定名字的表。如果相同名字的表已经存在，则抛出异常；用户可以用 IF NOT EXISTS 选项来忽略这个异常。

2、EXTERNAL 关键字可以让用户创建一个外部表，在建表的同时指定一个指向实际数据的路径（LOCATION）。

Hive 创建内部表时，会将数据移动到数据仓库指向的路径；若创建外部表，仅记录数据所在的路径，不对数据的位置做任何改变。在删除表的时候，内部表的元数据和数据会被一起删除，而外部表只删除元数据，不删除数据。

3、LIKE 允许用户复制现有的表结构，但是不复制数据。

CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name LIKE existing_table;

4、 ROW FORMAT DELIMITED

[FIELDS TERMINATED BY char]

[COLLECTION ITEMS TERMINATED BY char]

[MAP KEYS TERMINATED BY char]

[LINES TERMINATED BY char] | **SERDE** serde_name

[WITH SERDEPROPERTIES

(property_name=property_value, property_name=property_value,...)]

hive 建表的时候默认的分割符是'\001'，若在建表的时候没有指明分隔符，load 文件的时候文件的分隔符需要是'\001'；若文件分隔符不是'001'，程序不会报错，但表查询的结果会全部为'null'；

用 vi 编辑器 Ctrl+v 然后 Ctrl+a 即可输入'\001' ----------> ^A

SerDe 是 Serialize/Deserilize 的简称，目的是用于序列化和反序列化。

Hive 读取文件机制：首先调用 InputFormat（默认 TextInputFormat），返回一条一条记录(默认是一行对应一条记录)。然后调用 SerDe(默认 LazySimpleSerDe)的 Deserializer，将一条记录切分为各个字段（默认'\001'）。

Hive 写文件机制：将 Row 写入文件时，主要调用 OutputFormat、SerDe 的 Seriliazer，顺序与读取相反。

可通过 desc formatted 表名；进行相关信息查看。

当我们的数据格式比较特殊的时候，可以自定义 SerDe。

5、 PARTITIONED BY

在 hive Select 查询中一般会扫描整个表内容，会消耗很多时间做没必要的工作。有时候只需要扫描表中关心的一部分数据，因此建表时引入了 partition **分区**概念。

分区表指的是在创建表时指定的 partition 的分区空间。一个表可以拥有一个或者多个分区，每个分区以文件夹的形式单独存在表文件夹的目录下。表和列名不区分大小写。分区是以字段的形式在表结构中存在，通过 **describe table** 命令可以查看到字段存在，但是该字段不存放实际的数据内容，仅仅是分区的表示。

6、 **STORED AS** SEQUENCEFILE|TEXTFILE|RCFILE

如果文件数据是纯文本，可以使用 STORED AS TEXTFILE。如果数据需要压缩，使用 STORED AS SEQUENCEFILE。

TEXTFILE 是默认的文件格式，使用 **DELIMITED** 子句来读取分隔的文件。

6、**CLUSTERED BY INTO num_buckets BUCKETS**

对于每一个表（table）或者分，Hive 可以进一步组织成桶，也就是说桶是更为细粒度的数据范围划分。Hive 也是针对某一列进行桶的组织。Hive 采用对列值哈希，然后除以桶的个数求余的方式决定该条记录存放在哪个桶当中。

把表（或者分区）组织成桶（Bucket）有两个理由：

（1）获得更高的查询处理效率。桶为表加上了额外的结构，Hive 在处理有些查询时能利用这个结构。具体而言，连接两个在（包含连接列的）相同列上划分了桶的表，可以使用 Map 端连接 （Map-side join）高效的实现。比如 JOIN 操作。对于 JOIN 操作两个表有一个相同的列，如果对这两个表都进行了桶操作。那么将保存相同列值的桶进行 JOIN 操作就可以，可以大大较少 JOIN 的数据量。

（2）使取样（sampling）更高效。在处理大规模数据集时，在开发和修改查询的阶段，如果能在数据集的一小部分数据上试运行查询，会带来很多方便。

1.2. 修改表

增加分区：

```
ALTER TABLE table_name ADD PARTITION (dt='20170101') location
'/user/hadoop/warehouse/table_name/dt=20170101'; //一次添加一个分区

ALTER TABLE table_name ADD PARTITION (dt='2008-08-08', country='us') location
 '/path/to/us/part080808' PARTITION (dt='2008-08-09', country='us') location
 '/path/to/us/part080809';  //一次添加多个分区
```

删除分区

```
ALTER TABLE table_name DROP IF EXISTS PARTITION (dt='2008-08-08');
ALTER TABLE table_name DROP IF EXISTS PARTITION (dt='2008-08-08', country='us');
```

修改分区

```
ALTER TABLE table_name PARTITION (dt='2008-08-08') RENAME TO PARTITION (dt='20080808');
```

添加列

```
ALTER TABLE table_name ADD|REPLACE COLUMNS (col_name STRING);
```

注：ADD 是代表新增一个字段，新增字段位置在所有列后面(partition 列前)

REPLACE 则是表示替换表中所有字段。

修改列

```
test_change (a int, b int, c int);

ALTER TABLE test_change CHANGE a a1 INT; //修改 a 字段名

// will change column a's name to a1, a's data type to string, and put it after column b. The new
table's structure is: b int, a1 string, c int
ALTER TABLE test_change CHANGE a a1 STRING AFTER b;

// will change column b's name to b1, and put it as the first column. The new table's structure is:
b1 int, a ints, c int
ALTER TABLE test_change CHANGE b b1 INT FIRST;
```

表重命名

```
ALTER TABLE table_name RENAME TO new_table_name
```

1.3. 显示命令

show tables;

显示当前数据库所有表

show databases |schemas;

显示所有数据库

show partitions table_name;

显示表分区信息，不是分区表执行报错

show functions;

显示当前版本 hive 支持的所有方法

desc extended table_name;

查看表信息

desc formatted table_name;

查看表信息（格式化美观）

describe database database_name;

查看数据库相关信息

2. DML 操作

2.1. Load

在将数据加载到表中时，Hive 不会进行任何转换。加载操作是将数据文件移动到与 Hive 表对应的位置的纯复制/移动操作。

语法结构

```
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO
TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]
```

说明：

1、 filepath

相对路径，例如：project/data1

绝对路径，例如：/user/hive/project/data1

完整 URI，例如：hdfs://namenode:9000/user/hive/project/data1

filepath 可以引用一个文件（在这种情况下，Hive 将文件移动到表中），或者它可以是一个目录(在这种情况下,Hive 将把该目录中的所有文件移动到表中)。

2、 LOCAL

如果指定了 LOCAL， load 命令将在本地文件系统中查找文件路径。

load 命令会将 filepath 中的文件复制到目标文件系统中。目标文件系统由表的位置属性决定。被复制的数据文件移动到表的数据对应的位置。

如果没有指定 LOCAL 关键字，如果 filepath 指向的是一个完整的 URI，hive 会直接使用这个 URI。 否则：如果没有指定 schema 或者 authority，Hive 会使用在 hadoop 配置文件中定义的 schema 和 authority，fs.default.name 指定了 Namenode 的 URI。

3、 OVERWRITE

如果使用了 OVERWRITE 关键字，则目标表（或者分区）中的内容会被删除，然后再将 filepath 指向的文件/目录中的内容添加到表/分区中。

如果目标表(分区)已经有一个文件，并且文件名和 filepath 中的文件名冲突，那么现有的文件会被新文件所替代。

2.2. Insert

Hive 中 insert 主要是结合 select 查询语句使用，将查询结果插入到表中，例如：

insert overwrite table stu_buck

select * from student cluster by(Sno);

需要保证查询结果列的数目和需要插入数据表格的列数目一致.

如果查询出来的数据类型和插入表格对应的列数据类型不一致，将会进行转换，但是不能保证转换一定成功，转换失败的数据将会为 NULL。

可以将一个表查询出来的数据插入到原表中，结果相当于自我复制了一份数据。

Multi Inserts 多重插入:

from source_table

insert overwrite table tablename1 [partition (partcol1=val1,partclo2=val2)]

select_statement1

insert overwrite table tablename2 [partition (partcol1=val1,partclo2=val2)]

select_statement2..

Dynamic partition inserts 动态分区插入:

INSERT OVERWRITE TABLE tablename PARTITION (partcol1[=val1], partcol2[=val2] ...)

select_statement FROM from_statement

动态分区是通过位置来对应分区值的。原始表 select 出来的值和输出 partition 的值的关系仅仅是通过位置来确定的，和名字并没有关系。

导出表数据

语法结构

INSERT OVERWRITE [LOCAL] DIRECTORY directory1 SELECT ... FROM ...

multiple inserts:

FROM from_statement

INSERT OVERWRITE [LOCAL] DIRECTORY directory1 select_statement1

[INSERT OVERWRITE [LOCAL] DIRECTORY directory2 select_statement2] ...

数据写入到文件系统时进行文本序列化，且每列用^A 来区分，\n 为换行符。

2.3. Select

基本的 Select 操作

语法结构

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
JOIN table_other ON expr
[WHERE where_condition]
[GROUP BY col_list [HAVING condition]]
[CLUSTER BY col_list
  | [DISTRIBUTE BY col_list] [SORT BY| ORDER BY col_list]
]
[LIMIT number]
```

说明：

1、**order by** 会对输入做全局排序，因此只有一个 reducer，会导致当输入规模较大时，需要较长的计算时间。

2、**sort by** 不是全局排序，其在数据进入 reducer 前完成排序。因此，如果用 sort by 进行排序，并且设置 mapred.reduce.tasks>1，则 sort by 只保证每个 reducer 的输出有序，不保证全局有序。

3、**distribute by(**字段)根据指定字段将数据分到不同的 reducer，分发算法是 hash 散列。

4、**Cluster by**(字段) 除了具有 Distribute by 的功能外，还会对该字段进行排序。

如果 distribute 和 sort 的字段是同一个时，此时，cluster by = distribute by + sort by

3． Hive join

Hive 中除了支持和传统数据库中一样的内关联、左关联、右关联、全关联，还支持 LEFT SEMI JOIN 和 CROSS JOIN，但这两种 JOIN 类型也可以用前面的代替。

Hive 支持等值连接（a.id = b.id），不支持非等值(a.id>b.id)的连接，因为非等值连接非常难转化到 map/reduce 任务。另外，Hive 支持多 2 个以上表之间的 join。

写 join 查询时，需要注意几个关键点：

- **join 时，每次 map/reduce 任务的逻辑：**

reducer 会缓存 join 序列中除了最后一个表的所有表的记录，再通过最后一个表将结果序列化到文件系统。这一实现有助于在 reduce 端减少内存的使用量。实践中，应该把最大的那个表写在最后（否则会因为缓存浪费大量内存）。

- **LEFT，RIGHT 和 FULL OUTER 关键字用于处理 join 中空记录的情况**

```
SELECT a.val, b.val FROM a LEFT OUTER  JOIN b ON (a.key=b.key)
```

对应所有 a 表中的记录都有一条记录输出。输出的结果应该是 a.val，b.val，当 a.key=b.key 时，而当 b.key 中找不到等值的 a.key 记录时也会输出:

a.val，NULL

所以 a 表中的所有记录都被保留了；

“a RIGHT OUTER JOIN b”会保留所有 b 表的记录。

- **Join 发生在 WHERE 子句之前。**

如果你想限制 join 的输出，应该在 WHERE 子句中写过滤条件——或是在 join 子句中写。这里面一个容易混淆的问题是表分区的情况：

```
SELECT a.val, b.val FROM a
LEFT OUTER JOIN b ON (a.key=b.key)
WHERE a.ds='2009-07-07' AND b.ds='2009-07-07'
```

这会 join a 表到 b 表（OUTER JOIN），列出 a.val 和 b.val 的记录。WHERE 从句中可以使用其他列作为过滤条件。但是，如前所述，如果 b 表中找不到对应 a 表的记录，b 表的所有列都会列出 NULL，包括 ds 列。也就是说，join 会过滤 b 表中不能找到匹配 a 表 join key 的所有记录。这样的话，LEFT OUTER 就使得查询结果与 WHERE 子句无关了。解决的办法是在 OUTER JOIN 时使用以下语法：

```
SELECT a.val, b.val FROM a LEFT OUTER JOIN b
```

```
ON (a.key=b.key AND
    b.ds='2009-07-07' AND
    a.ds='2009-07-07')
```

这一查询的结果是预先在 join 阶段过滤过的，所以不会存在上述问题。这一逻辑也可以应用于 RIGHT 和 FULL 类型的 join 中。

- **Join 是不能交换位置的。**

  无论是 LEFT 还是 RIGHT join，都是左连接的。

```
SELECT a.val1, a.val2, b.val, c.val
FROM a
JOIN b ON (a.key = b.key)
LEFT OUTER JOIN c ON (a.key = c.key)
```

先 join a 表到 b 表，丢弃掉所有 join key 中不匹配的记录，然后用这一中间结果和 c 表做 join。

七、 Hive 参数配置

1. Hive 命令行

输入$HIVE_HOME/bin/hive –H 或者 –help 可以显示帮助选项：

说明：

1、 -i 初始化 HQL 文件。
2、 -e 从命令行执行指定的 HQL
3、 -f 执行 HQL 脚本
4、 -v 输出执行的 HQL 语句到控制台
5、 -p <port> connect to Hive Server on port number
6、 -hiveconf x=y Use this to set hive/hadoop configuration variables.

例如：

```
$HIVE_HOME/bin/hive -e 'select * from tab1 a'
$HIVE_HOME/bin/hive -f /home/my/hive-script.sql
$HIVE_HOME/bin/hive -f hdfs://<namenode>:<port>/hive-script.sql
$HIVE_HOME/bin/hive -i /home/my/hive-init.sql
$HIVE_HOME/bin/hive -e 'select a.col from tab1 a'
    --hiveconf hive.exec.compress.output=true
    --hiveconf mapred.reduce.tasks=32
```

2. Hive 参数配置方式

Hive 参数大全：

https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties

开发 Hive 应用时，不可避免地需要设定 Hive 的参数。设定 Hive 的参数可以调优 HQL 代码的执行效率，或帮助定位问题。然而实践中经常遇到的一个问题是，为什么设定的参数没有起作用？这通常是错误的设定方式导致的。

对于一般参数，有以下三种设定方式：

配置文件 （全局有效）

命令行参数 （对 hive 启动实例有效）

参数声明 （对 hive 的连接 session 有效）

配置文件

用户自定义配置文件：$HIVE_CONF_DIR/hive-site.xml

默认配置文件：$HIVE_CONF_DIR/hive-default.xml

用户自定义配置会覆盖默认配置。

另外，Hive 也会读入 Hadoop 的配置，因为 Hive 是作为 Hadoop 的客户端启动的，Hive 的配置会覆盖 Hadoop 的配置。

配置文件的设定对本机启动的所有 Hive 进程都有效。

命令行参数

启动 Hive（客户端或 Server 方式）时，可以在命令行添加-hiveconf 来设定参数

例如：bin/hive -hiveconf hive.root.logger=INFO,console

设定对本次启动的 Session（对于 Server 方式启动，则是所有请求的 Sessions）有效。

参数声明

可以在 HQL 中使用 SET 关键字设定参数，这一设定的作用域也是 session 级的。

比如：

set hive.exec.reducers.bytes.per.reducer=<number> 每个 reduce task 的平均负载数据量

set hive.exec.reducers.max=<number> 设置 reduce task 数量的上限

set mapreduce.job.reduces=<number> 指定固定的 reduce task 数量

但是，这个参数在必要时<业务逻辑决定只能用一个 reduce task> hive 会忽略

上述三种设定方式的优先级依次递增。即参数声明覆盖命令行参数，命令行参数覆盖配置文件设定。注意某些系统级的参数，例如 log4j 相关的设定，必须用前两种方式设定，因为那些参数的读取在 Session 建立以前已经完成了。

八、 Hive 函数

1. 内置运算符

在 Hive 有四种类型的运算符：

- 关系运算符
- 算术运算符
- 逻辑运算符
- 复杂运算

内容较多，见《Hive 官方文档》或者《hive 常用运算和函数.doc》

2. 内置函数

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF

测试各种内置函数的快捷方法：

创建一个 dual 表

```
create table dual(id string);
```

load 一个文件（只有一行内容：内容为一个空格）到 dual 表

```
select substr('angelababy',2,3) from dual;
```

内容较多，见《Hive 官方文档》或者《hive 常用运算和函数.doc》

3. Hive 自定义函数和 Transform

当 Hive 提供的内置函数无法满足你的业务处理需要时，此时就可以考虑使用用户自定义函数（UDF：user-defined function）。

3.1. UDF 开发实例

新建 JAVA maven 项目

添加 hive-exec-1.2.1.jar 和 hadoop-common-2.7.4.jar 依赖（见参考资料）

1、写一个 java 类，继承 UDF，并重载 evaluate 方法

```
package cn.itcast.bigdata.udf
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;

public class Lower extends UDF{
    public Text evaluate(Text s){
        if(s==null){return null;}
        return new Text(s.toString().toLowerCase());
    }
}
```

2、打成 jar 包上传到服务器

3、将 jar 包添加到 hive 的 classpath

```
hive>add JAR /home/hadoop/udf.jar;
```

4、创建临时函数与开发好的 java class 关联

```
create temporary function tolowercase as 'cn.itcast.bigdata.udf.ToProvince';
```

5、即可在 hql 中使用自定义的函数 tolowercase ip

```
Select tolowercase(name),age from t_test;
```

3.2. Transform 实现（了解）

Hive 的 TRANSFORM 关键字***提供了在 SQL 中调用自写脚本的功能***

适合实现 Hive 中没有的功能又不想写 UDF 的情况

使用示例 1：下面这句 sql 就是借用了 weekday_mapper.py 对数据进行了处理.

```
add FILE weekday_mapper.py;

INSERT OVERWRITE TABLE u_data_new
SELECT
  TRANSFORM (movieid , rate, timestring,uid)
  USING 'python weekday_mapper.py'
  AS (movieid, rating, weekday,userid)
FROM t_rating;
```

其中 weekday_mapper.py 内容如下

```
#!/bin/python
import sys
import datetime

for line in sys.stdin:
  line = line.strip()
  movieid, rating, unixtime,userid = line.split('\t')
  weekday = datetime.datetime.fromtimestamp(float(unixtime)).isoweekday()
  print '\t'.join([movieid, rating, str(weekday),userid])
```

4. Hive 特殊分隔符处理（扩展）

hive 读取数据的机制：

首先用 InputFormat<默认是：org.apache.hadoop.mapred.TextInputFormat >的一个具体实现类读入文件数据，返回一条一条的记录（可以是行，或者是你逻辑中的“行”）

然后利用 SerDe<默认：org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe>的一个具体实现类，对上面返回的一条一条的记录进行字段切割。

Hive 对文件中字段的分隔符默认情况下只支持单字节分隔符，如果数据文件中的分隔符是多字符的，如下所示：

01||zhangsan

02||lisi

可用使用 RegexSerDe 通过正则表达式来抽取字段

```
drop table t_bi_reg;
create table t_bi_reg(id string,name string)
row format serde 'org.apache.hadoop.hive.serde2.RegexSerDe'
with serdeproperties(
'input.regex'='(.*)\\|\\|(.*)',
'output.format.string'='%1$s %2$s'
)
stored as textfile;

hive>load data local inpath '/root/hivedata/bi.dat' into table t_bi_reg;
hive>select * from t_bi_reg;
```

其中：

input.regex：输入的正则表达式

表示 || 左右两边任意字符被抽取为一个字段

output.format.string：输出的正则表达式

%1$s %2$s 则分别表示表中的第一个字段、第二个地段

注意事项：

a、使用 RegexSerDe 类时，所有的字段必须为 string

b、input.regex 里面，以一个匹配组，表示一个字段