# 1.1数据湖概念

## 1.1.1 什么是数据湖

数据湖是一个集中式的存储库，允许你以任意规模存储多个数据源，所有结构化和非结构化数据，可以按照原样存储数据，无需对数据进行结构化处理，并运行不同类型的分析对数据进行加工，例如：大数据处理、实时分析、机器学习，以指导做好更好地决策

## 1.1.2 大数据为什么需要数据湖

当前基于Hive的离线数据仓库已经非常成熟，在传统的离线数据仓库中对记录级别的数据进行更新是非常麻烦的，需要对待更新的数据所属的整个分区，甚至是整个表进行全面覆盖才行，由于离线数仓多级逐层加工的架构设计，数据更新时也需要从贴源层开始逐层反应到后续的派生表中去。

随着实时计算引擎的不断发展以及业务对于实时报表的产出需求不断膨胀，业界最近几年就一直聚焦并探索于实时数仓建设。根据数仓架构演变过程，在Lambda架构中含有离线处理与实时处理两条链路，其架构图如下：

图1

Kappa架构可以称为真正的实时数仓，目前在业界最常用实现是Flink + Kafka，然而基于Kafka+Flink的实时数仓方案也有几个非常明显的缺陷，所以在目前很多企业中实时数仓构建中经常使用混合架构，没有实现所有业务都采用Kappa架构中实时处理实现，Kappa架构缺陷如下：

1. Kafka无法支持海量数据存储。对于海量数据量数据线来说，Kafka一般只能存非常短时间的数据，比如最近一周，甚至最近一天。
2. Kafka无法支持高效的OLAP查询，大多数业务都希望能在DWD\DWS层支持缺席查询的，但是Kafka无法非常友好地支持这样的需求。
3. 无法复用目前已经非常成熟的基于离线数仓的数据血缘、数据质量管理体系，需要重新实现一套数据血缘、数据质量管理体系。
4. Kafka不支持updat/upsert，目前Kafka仅支持append。

为解决Kappa结构的痛点问题，业界最主流是采用“流批一体”方式，这里流批一体可以理解为批和流使用SQL同一处理，也可以理解为处理框架的统一，例如：Spark、Flink，但这里更重要指的是存储层上的统一，只要存储层上做到“批流一体”就可以解决以上Kappa遇到的各种问题。数据湖技术可以很好的实现存储层面上的“批流一体”，这就是为什么大数据中需要数据湖的原因。

## 1.1.3 数据湖和数据仓库的区别

数据仓库与数据湖主要的区别在于如下两点：

* 存储数据类型

数据仓库是存储数据，进行建模，存储的是结构化数据；数据湖以其本源格式保存大量原始数据，包括结构化的、半结构化的和结构化的数据，主要是由原始的、混乱的、非结构化的数据组成。在需要数据之前，没有定义数据结构和需求。

* 数据处理模式

在我们可以加载到数据仓库中的数据，我们首先需要定义好它，这叫做写时模式（Schema-On-Write）。而对于数据湖，您只需加载原始数据，然后，当您准备使用数据时，就给它一个定义，这叫做读模式（Schema-On-Read）。这是两种截然不同的数据处理方法。因为数据湖是在数据使用时再定义模型结构，因此提高了数据模型定义的灵活性，可满足更多不同上层业务的高效率分析诉求

图2

# 1.2什么是Hudi

Apache Hudi 是一个Data Lakes的开源方案，Hudi是Hadoop Update And Increamentals的简写，它是由Uber开发并开源的Data Lakes解决方案。Hudi能够基于HDFS之上管理大型分析数据基，可以对数据进行插入、更新、增量消费等操作，主要目的是高效减少摄取过程中的数据延时。

Hudi非常轻量级，可以作为lib与Spark、Flink进行集成，Hudi官网http://hudi.apache.org/

图3

Hudi基于Parquet列式存储与Avro行式存储，同时避免创建小文件，实际高效率低延时的数据访问。在HDFS数据集上提供插入更新、增量拉取、全量拉取。Hudi具有如下特点：

* 快速upsert，可插入索引。
* 以原子方式操作数据并具有回滚功能。
* 写入器和查询之间的快照隔离。
* 用于数据恢复的savepoint保存点。Hudi通过Savepoint来实现数据恢复
* 管理文件大小，使用统计数据布局。
* 行和列数据的异步压缩。
* 时间轴元数据跟踪血缘。

# 1.3 Hudi概念术语

# 1.3.1 Timeline

Hudi数据湖可以维护很多张表，与Hive类似，数据存储在HDFS不同的目录结构中。Hudi维护了表在不同时刻执行的所有操作的Timeline，这有助于提供表的瞬时视图。

图4

* Timeline是Hudi用来管理提交（commit）的抽象，每个commit都绑定一个固定时间戳，分散到时间线上。在Timeline上，每个commit被抽象为一个HoodieInstant（hoodie瞬时时刻），一个instant记录了一次提交（commit）的行为、时间戳、和状态，也就是说每个HoodieInstant包含Action、Time、State三个部分，下面介绍每个HoodieInstant对应的这三个部分
* Instant Action

Instant Action指的是对Hudi表执行的操作类型，目前包括COMMITS、CLEANS、DELTA\_COMMIT、COMPACTION、ROLLBACK、SAVEPOINT这6种操作类型

1. Commits：表示一批记录原子性的写入到一张表中。
2. Cleans：清除表中不再需要的旧版本文件。
3. Delta\_commit：增量提交指的是将一批记录原子地写入MergeOnRead类型表，其中一些/所有数据都可以写入增量日志。
4. Compation：将行式文件转化为列式文件。
5. Rollback：Commits或者Delta\_commit执行不成功时回滚数据，删除期间产生的任意文件。
6. SavePoint：将文件组标记为”saved”，cleans执行时不会删除对应的数据。

* Instant Time

Instant Time表示一个时间戳，这个时间戳必须是按照Instant Action开始执行的时间顺序单调递增的。

* Instant State

Instant State表示在指定的时间点（Instant Time）对Hudi表执行操作（Instant Action）后，表所处的状态，目前包括REQUESTED（已调度但未初始化）、INFLIGHT（当前正在执行）、COMPLETED（操作执行完成）这3种状态。

Hudi的读写API通过Timeline的接口可以方便的在commit上进行条件筛选，对history和on-going的commit应用各种策略，快速筛选出需要操作的目标commit

下面结合官网中给出的例子理解下Timeline，例子场景是，在10：00~10:20之间，要对一个Hudi表执行Upsert操作，操作的频率大约是5分钟执行一次，每次操作执行完成，会看到对应这个Hudi表的Timeline上，有1系列Commit元数据生成。当满足一定条件时，会在指定的时刻对这些commit进行cleans和compaction操作，这两个操作都是在后台完全，其中在10：05之后执行了一次cleans操作，10:10之后执行了一次compaction操作。

图5

我们看到，从数据生成到最终到达Hudi系统，可能存在延时，乳上图数据大约在07:00、08：00、09:00时生成，数据到达大约延迟了分分别3、2、1小时多，最终生成commit的时间才是Upsert的时间，通过使用Timeline来管理，当增量查询10:00之后的最新数据时，可以非常高效的找到10:00之后发生过更新的文件，而不必根据延迟时间再去扫描更早时间文件，比如这里，就不需要扫描7:00、8:00或9:00这些时刻对应的文件（Buckets）。

# 1.3.2 文件格式及索引

Hudi将表组织HDFS上某个指定目录（basepath）下的目录结构，表被分成多个分区，分区是以目录的形式存在，每个目录下面会存在属于该分区的多个文件，类似Hive表，每个Hudi表分区通过一个分区路径（partitionpath）来唯一标识。

图6

在每个分区下面，通过文件分组（file groups）的方式来组织，每个分组对应一个唯一的文件ID。每个文件分组中包含多个文件分片（file slices）（一个新的base commit time）对应一个新的文件分片，实际就是一个新的数据版本），每个文件分片包含一个Base文件（\*.parquest）,这个文件是在执行commit/compaction操作的时候生成的，同时还生成了几个日志文件（\*.log\*）,日志文件中包含了从该Base文件生成以后执行的插入/更新操作。

Hudi采用MVCC设计，当执行COMPACTION操作时，会合并日志文件和Base文件，生成新的文件分片。Cleans操作会清理不用的/旧的文件分片，释放存储空间。

Hudi会通过记录Key与分区Path组成HoodieKey，即RecordKey+PartitionPath，通过将HoodieKey映射到前面提到的文件ID，具体其实是映射到file\_group/file\_id，这就是Hudi的索引。一旦记录的第一个版本被写入文件中，对应的Hoodie Key 就不会再改变了。

Hudi的Base