QCon+ 案例研习社

腾讯在新一代数据湖技术Iceberg上的探索与实践

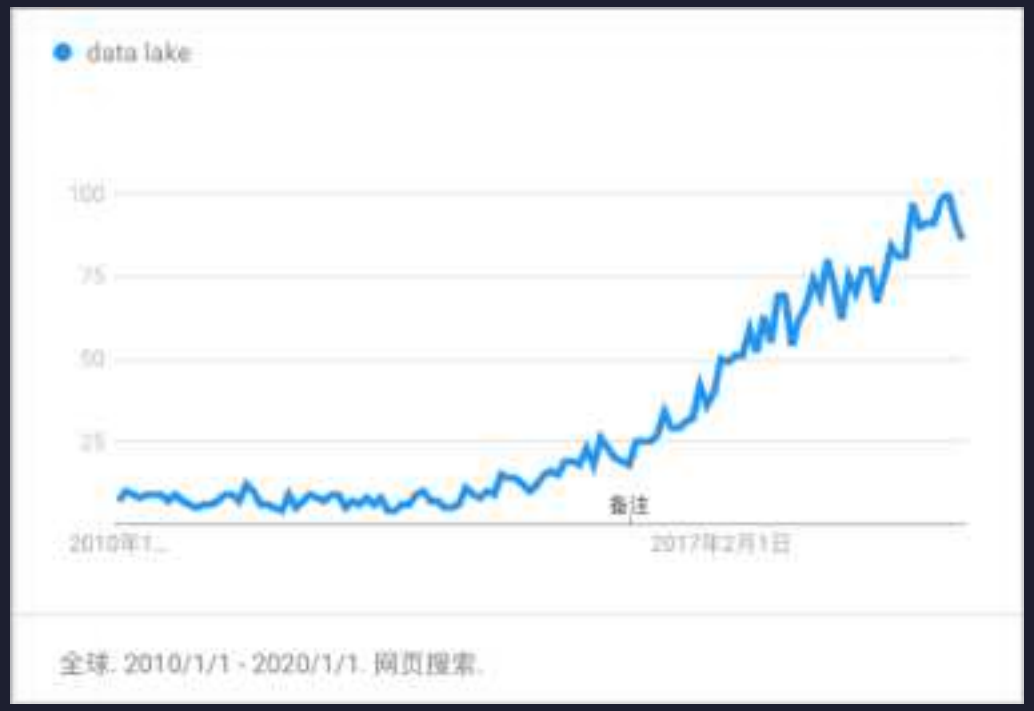
邵赛赛

自我介绍

- 腾讯大数据数据湖负责人
- Apache Member，Apache Spark PMC，Apache Livy PPMC
- 曾就职于Hortonworks，Intel，多年开源大数据从业经验

目录

数据湖发展的现状



- 从广义上看数据湖系统主要包括数据湖存储和数据湖分析
- 现有数据湖技术主要由云厂商推动，包括基于对象存储的数据湖存储及在其之上的分析套件
  - 基于对象存储（S3, WASB）的数据湖存储技术，如Azure ADLS, AWS Lake Formation等
  - 以及运行在其上的分析工具，如AWS EMR, Azure HDInsight, RStudio等等

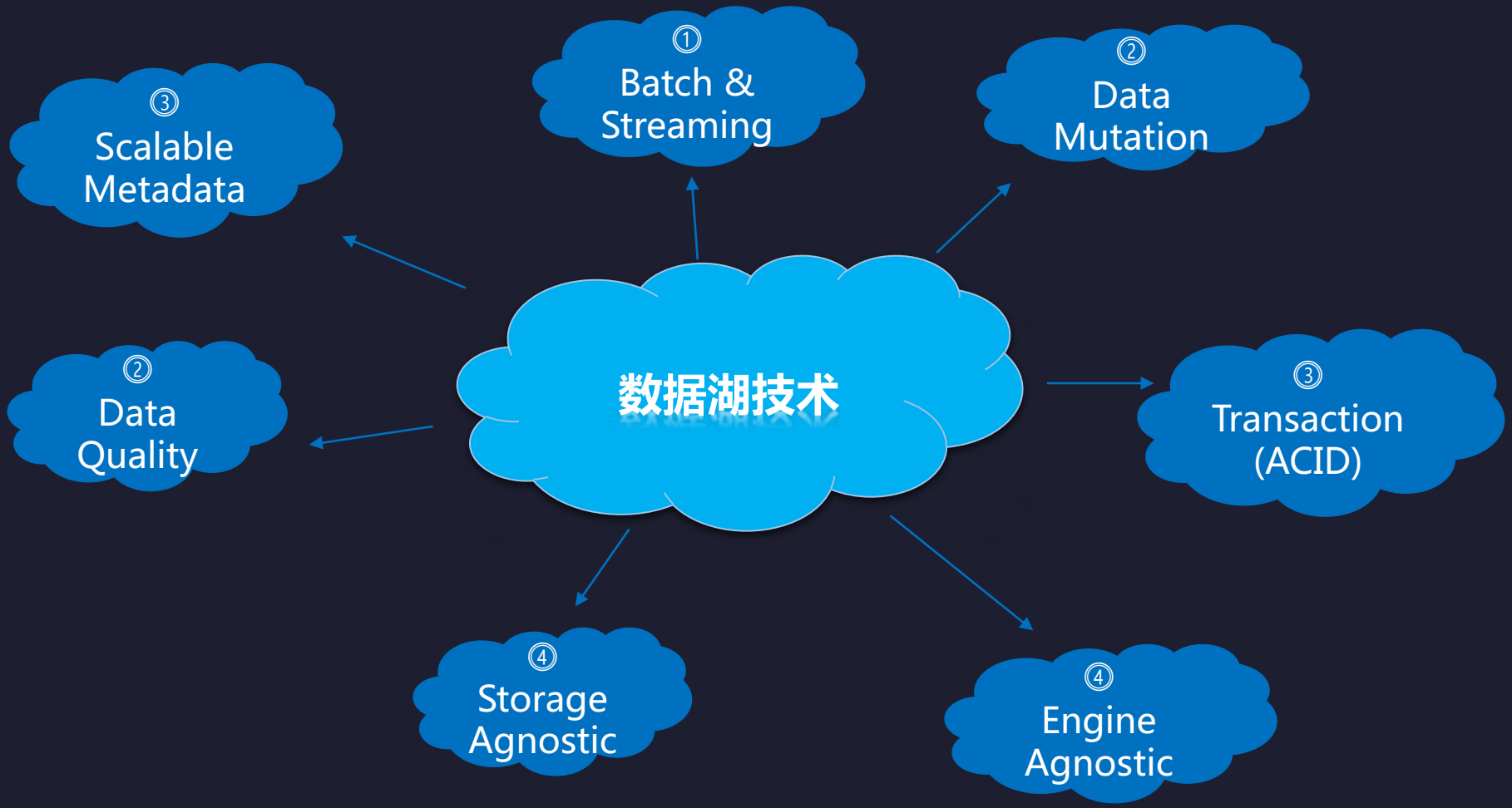
业界趋势

- 构建**统一、高效的数据存储**以满足不同数据处理场景的需求已成为趋势
  - ETL作业和OLAP分析 — 高性能的结构化存储，分布式能力
  - 机器学习训练和推理 — 海量的非结构存储，容器挂载能力

- 通用数仓(Hive，Spark)在向**数据湖分析泛化**，而数仓则向高性能架构演进(Snowflake)

目录

数据湖技术需要具备的能力



数据湖技术三剑客

- 构建于存储格式之上的**数据组织方式**
- 提供ACID能力，提供一定的**事务特性和并发能力**
- 提供**行级别的数据修改能力**
- 确保schema的**准确性**，提供一定的schema**修改能力**

数据湖技术的比较

	Delta Lake (open source)	Apache Iceberg	Apache Hudi
Transaction (ACID)	Y	Y	Y
MVCC	Y	Y	Y
Time travel	Y	Y	Y
Schema Evolution	Y	Y	Y
Data Mutation	Y (update/delete/merge into)	WIP	Y (upsert)
Streaming	Sink and source for spark struct streaming	Sink and source (WIP) for Spark struct streaming, Flink (WIP)	DeltaStreamer HiveIncrementalPuller
File Format	Parquet	Parquet, ORC, AVRO	Parquet
Compaction/Cleanup	Manual	API available (Spark Action)	Manual and Auto
Integration	DSv1, Delta connector	DSv2 InputFormat	DSv1 InputFormat
Multiple language support	Scala/Java/Python	Java/Python	Java/Python
Storage Abstraction	Y	Y	N
API dependency	Spark-bundled	Native/Engine bundled	Spark-bundled
Data ingestion	Spark, Presto, Hive	Spark, Flink, Hive	DeltaStreamer

目录

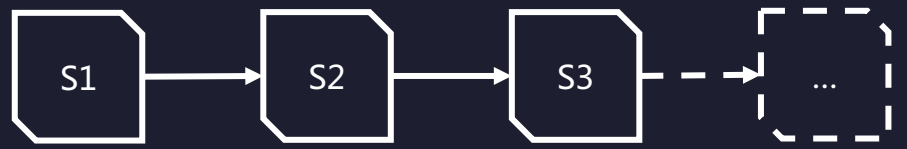
Apache Iceberg

Apache Iceberg is an open table format for huge analytic datasets.

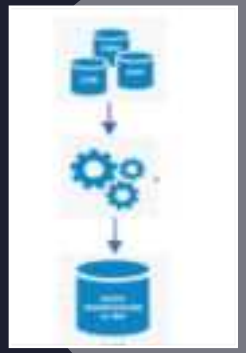
Iceberg adds tables to Presto and Spark that use a high-performance format that works just like a SQL table.

核心思想：在时间轴上跟踪表的所有变化

- 快照（snapshot）表示表数据文件的一个完整集合。
- 每次更新操作会生成一个新的快照。



为什么选择Iceberg



优化数据入库流程

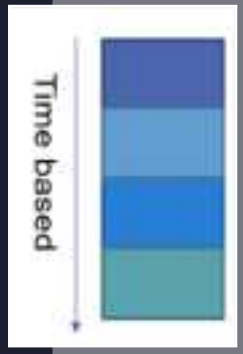
- Iceberg提供ACID事务能力，上游数据写入即可见，不影响当前数据处理任务，这大大简化了ETL
- Iceberg提供upsert/merge into 能力，可以极大地缩小数据入库延迟

支持更多的分析引擎

- 优秀的内核抽象使之不绑定于特定引擎，目前在支持的有Spark, Flink, Presto, Hive.
- Iceberg提供了java native API，不用特定引擎也可以访问Iceberg表

统一数据存储和灵活的文件组织

- 提供了基于流式的增量计算模型和基于批处理的全量表计算模型，批任务和流任务可以使用相同的存储模型(HDFS, OZONE)，数据不再孤立。
- Iceberg支持隐藏分区和分区进化，方便业务进行数据分区策略更新
- 支持Parquet, ORC, Avro行存列存兼顾



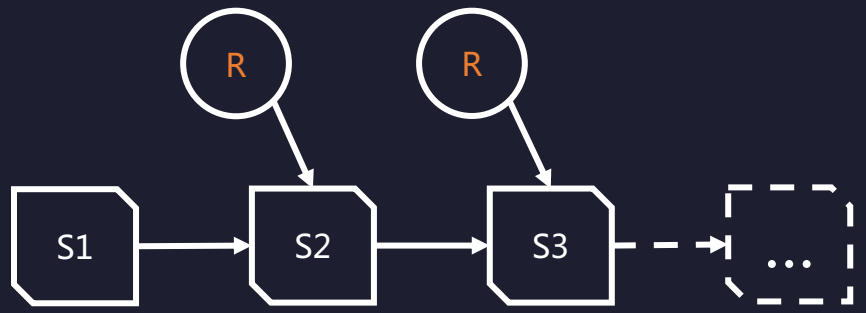
增量读取处理能力

- Iceberg支持通过流式方式读取增量数据
- Spark struct streaming 支持
- Flink table source支持

特性细节 – 快照设计方式

快照隔离

- 读操作仅适用当前已生成的快照
- 写操作会生成新的隔离快照，并在写完成后原子性提交

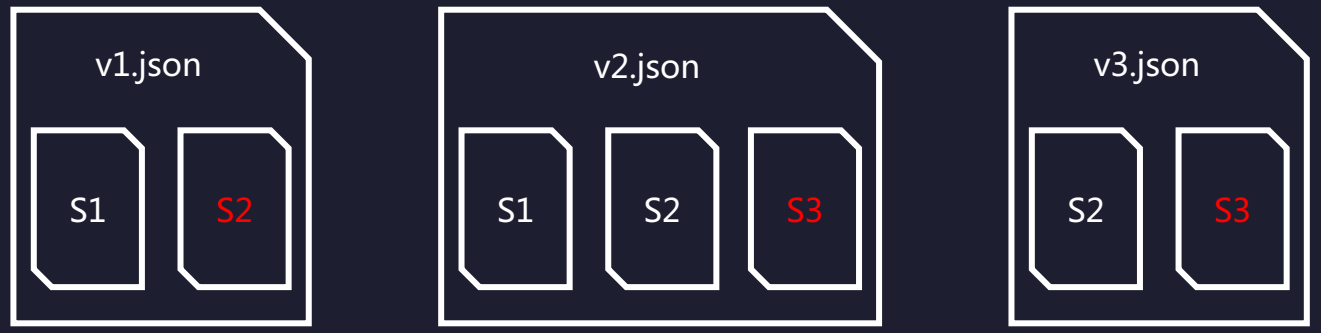


对于文件列表的所有修改都是原子操作

- 在分区中追加数据
- 合并或是重写分区

特性细节 – 元数据组织

- 实现基于快照的跟踪方式
  - 记录表的结构，分区信息，参数等
  - 跟踪老的快照以确保能够最终回收



- 表的元数据是不可修改的，并且始终向前迭代
- 当前的快照可以回退

特性细节 – 事务性提交

写操作必须

- 记录当前元数据的版本 — base version
- 创建新的元数据以及manifest文件
- 原子性地将base version替换为新的版本

原子性替换保证了线性的历史

原子性替换需要依靠以下操作来保证

- 元数据管理器所提供的能力
- HDFS或是本地文件系统所提供的原子化的rename能力

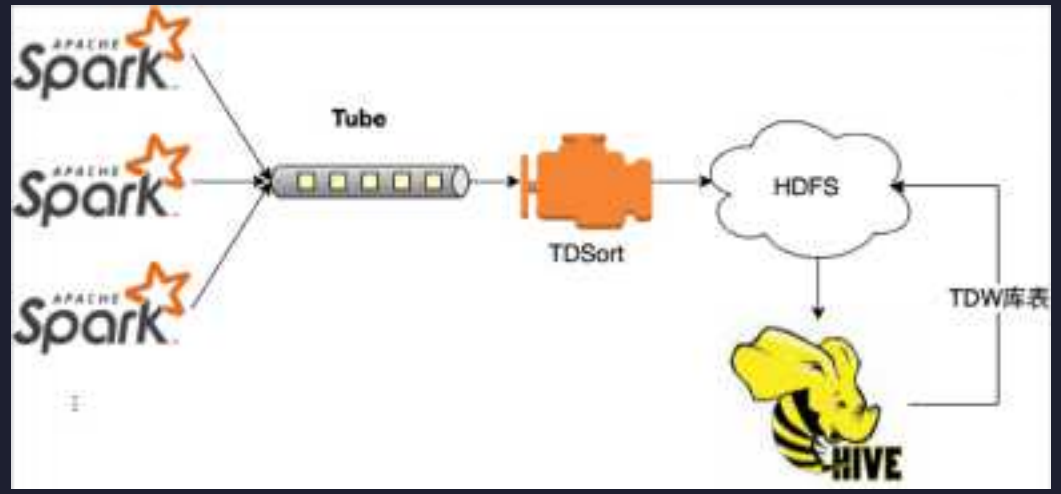
冲突解决 — 乐观锁

- 假定当前没有其他的写操作
- 遇到冲突则基于当前最新的元数据进行重试

目录

原有的数仓构建方案

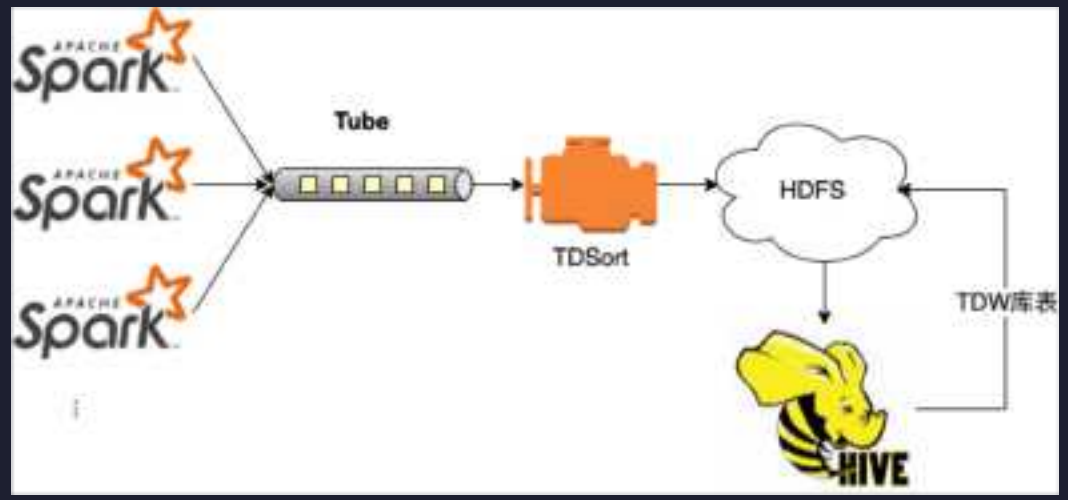
- 复杂的增量入库方案来保证exactly–once和数据去重
- 利用HDFS rename操作的原子性和复杂的命名规则来保证一致性、可见性
- 利用调度引擎来构建依赖关系，避免读写冲突

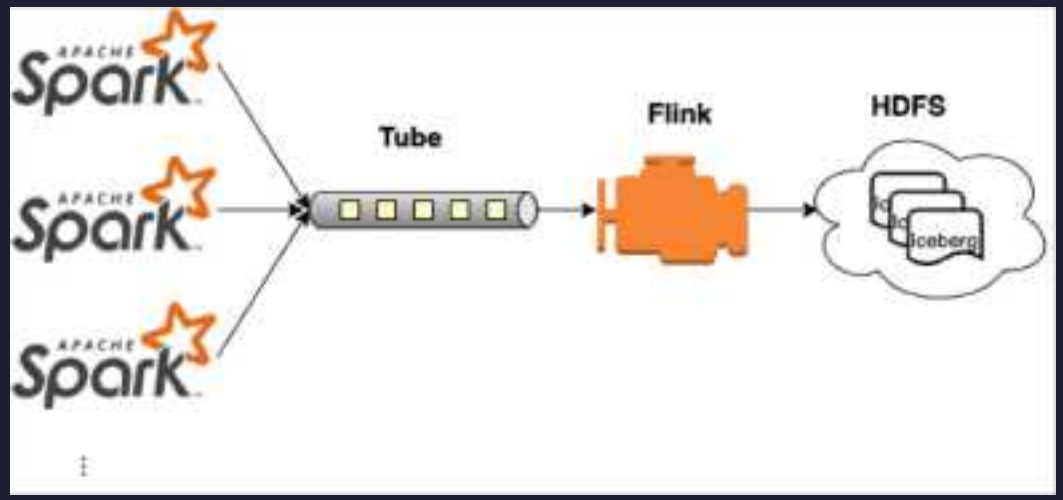


存在的问题

- 架构复杂，涉及多个不同系统的协调
- 架构的复杂性导致延迟明显，从数据产生到最终展示有较长的时延
- 需要支持exactly-once，并支持数据去重，导致入库方案异常复杂，增加运维难度

如何利用Iceberg改造原有方案





- Iceberg格式是Hive/Spark兼容的可读写的表格式，可以直接使用Hive/Spark进行处理，无须再次将数据导入到数仓中
- Iceberg支持读写分离，写入并且commit后的数据下游立即可见，因此可以实时读取到新增的数据，降低整体时延

数据湖技术的出现和兴起

- 架构的演化催生新技术的产生
  - 简化整体架构
  - 降低端到端的延迟
  - 赋予事务型能力

Lambda Architecture　　Kappa Architecture　　CDC Architecture

基于Iceberg的数据湖分析系统

- 数据入湖
  - 利用Flink + Iceberg构建准实时数据接入层
- 湖上分析
  - 利用Flink，Structured Streaming进行增量数据处理
  - 利用Spark 3.0 SQL提供pluggable catalog和CRUD能力

Iceberg最佳实践一

Problem: Snapshot数量持续增长

- JSON metadata文件的大小持续增长，读写的时间变久
- 阻碍老的数据和元数据文件的删除

Solution #1: ExpireSnapshots

- Pros
  - 在常规的维护中能很好地工作
  - 无须底层分布式引擎的参与
  - 无须从底层存储中listing文件
- Cons
  - 某些corner case会很慢

Solution #2: ExpireSnapshotsAction

- Pros
  - 能够非常快速地删除上千个snapshot
  - 无须从底层存储中listing文件
- Cons
  - 需要底层分布式引擎的参与
  - 需要更多的资源

小贴士

- 定期删除snapshots（每1到2天）
- 保留足够多的历史以便数据恢复（5+天）
- 常规维护使用Table API
- 对于较重的case使用Actions API

Iceberg最佳实践二

Problem：遗留数据和元数据

- 由于作业的不正常结束造成没有commit的文件
- 在删除snapshot时候遗留的文件
- 遗留文件造成表的膨胀

Solution：RemoveOrphaneFilesAction

- Pros
  - 使用分布式的元数据表
  - 可以同时删除数据和元数据
- Cons
  - 需要使用代价高昂的listing操作
  - 需要依赖分布式引擎的参与

小贴士

- 不要频繁调用RemoveOrphaneFilesAction（1月1次）
- 在删除之前可以指定空的delete function来得到删除数据的预览

Iceberg最佳实践三

Problem：未优化的元数据

- 过多过小的manifest文件的产生
- 印象执行引擎对元数据的解析

Solution #1：在commit的时候合并manifest文件

- Pros
- 默认已经开启该功能
- 自动化地进行
- Cons
- 增加commit的时间
- Commit重试的代价变高

Solution #2：RewriteManifestsAction

- Pros
- 适合于PB级别的规模，每次写入需要操作多个分区的表
- Commit和重试尽可能保证轻量级
- Cons
- 需要用户或是系统手动触发

小贴士

- 除非行为是可预估的，在一般情况下禁止元数据聚合
  - commit.manifest-merge.enabled
- 如果作业执行变慢，根据需要进行元数据重写
- 开启snapshot id inheritance
  - compatibility.snapshot-id-inheritance.enabled

后续的规划

Iceberg内核能力提升

- 对于Merge On Read能力的支持
- 通用索引系统的实现

上下游适配的完善

- 更好地支持Spark DF，SQL
- 完善对Flink，Hive，Presto的支持

打造新一代的数据湖分析引擎

- 将Iceberg更好地融入的流式计算平台，数据分析平台
- 围绕Iceberg打造全新的入口，一站式的数据湖分析平台

总结

- 数据湖的趋势及现有的数据湖技术
- Iceberg的优势和技术细节
- 围绕Iceberg打造新一代的数据湖分析系统

THANKS

QCon+ 案例研习社