

4.1 MongoDB 应用场景及选型



扫码试看/订阅

《MongoDB 高手课》视频课程

MongoDB 数据库定位

- OLTP 数据库
- 原则上 Oracle 和 MySQL 能做的事情，MongoDB 都能做（包括 ACID 事务）
- 优点：横向扩展能力，数据量或并发量增加时候架构可以自动扩展
- 优点：灵活模型，适合迭代开发，数据模型多变场景
- 优点：JSON 数据结构，适合微服务/REST API

基于功能选择 MongoDB

	MongoDB	关系型数据库
亿级以上数据量	轻松支持	要努力一下，分库分表
灵活表结构	轻松支持	Entity Key /Value 表，关联查询比较痛苦
高并发读	轻松支持	需要优化
高并发写	轻松支持	需要优化
跨地区集群	轻松支持	需要定制方案
分片集群	轻松支持	需要中间件
地理位置查询	比较完整的地理位置	PG 还可以，其他数据库略麻烦
聚合计算	功能很强大	使用 Group By 等，能力有限
异构数据	轻松支持	使用 EKV 属性表
大宽表	轻松支持	性能受限

基于场景选择 MongoDB

移动/小程序
App

电商

内容管理

物联网

SaaS 应用

主机分流

实时分析

关系型迁移

移动应用

场景特点

- 基于 REST API / JSON
- 快速迭代，数据结构变化频繁
- 地理位置功能
- 爆发增长可能性
- 高可用

MongoDB 选型考量

- 文档模型可以支持不同的结构
- 原生地理位置功能
- 横向扩展能力支撑爆发增长
- 复制集机制快速提供高可用
- 摩拜单车 / Keep / ADP

商品信息

场景特点

- 商品信息包罗万象
- 商品的属性不同品类差异很大
- 数据库模式设计困难

MongoDB 选型考量

- 文档模型可以集成不同商品属性
- 可变模型适合迭代
- 京东商城 / 小红书 / GAP

内容管理

场景特点

- 内容数据多样，文本，图片，视频
- 扩展困难，数据量爆发增长

MongoDB 选型考量

- JSON 结构可以支持非结构化数据
- 分片架构可以解决扩展问题
- Adobe AEM / Sitecore

物联网

场景特点

- 传感器的数据结构往往是半结构化
- 传感器数量很大，采集频繁
- 数据量很容易增长到数亿到百亿

MongoDB 选型考量

- JSON 结构可以支持半结构化数据
- 使用分片能力支撑海量数据
- JSON 数据更加容易和其他系统通过 REST API 进行集成
- 华为 / Bosch / Mindsphere

SaaS应用

场景特点

- 多租户模式，需要服务很多客户
- 需求多变，迭代压力大
- 数据量增长快

MongoDB 选型考量

- 无模式数据库，适合快速迭代
- 水平扩展能力可以支撑大量用户增长
- ADP / Team ambition

主机分流

场景特点

- 金融行业传统采用 IBM 或者小机
- 传统瀑布开发模式流程长成本高
- 结构不易改变，难于适应新需求
- 根据某银行的统计，99% 的数据库操作为读流量
- 基于 MIPS 付费，读流量成本高

MongoDB 选型考量

- 使用实时同步机制，将数据同步出来到 MongoDB
- 使用 MongoDB 的高性能查询能力来支撑业务的读操作
- 相比于关系模型数据库，更加容易迁入数据并构建成 JSON 模型进行 API 服务

实时分析

场景特点

- 流数据计算
- 快速计算，秒级返回

MongoDB 选型考量

- 使用 MongoDB 缓存机制，可以利用内存计算加速
- 使用 MongoDB 聚合框架，实现分析功能
- 使用微分片架构的并发计算来大量缩减计算时间

关系型数据库替换

场景特点

- 基于 Oracle / MySQL/ SQLServer 的历史应用
- 数据量增长或者使用者变多以后性能变慢
- 分库分表需要应用配合
- 结构死板，增加新需求复杂困难

MongoDB 选型考量

- 高性能高并发的数据库性能
- 无需应用分库分表，集群自动解决扩容问题
- 动态模型适合快速开发
- 头条/网易/百度/东航/中国银行

4.2 MongoDB 典型案例（一）

案例一：客户360 世界500强保险公司

业务需求

跨国保险公司，来自60多个国家的9000多万用户，70多套业务系统。客户信息分散在多套系统里，希望构建一个客户360视图。

第一阶段支撑客服部门更好服务客户，减少客户等待时间。

第二阶段构建客户管理手机 App，自助管理所有保单。

为了达到这些目的，需要整个70+历史系统中的客户信息，通过唯一入口查询。

The screenshot displays a comprehensive customer information dashboard. At the top, there are search bars for 'Quick Search' and 'Contract key #' with a magnifying glass icon, and links for 'Advanced Search' and 'Logout'. The main header includes the company logo 'MetLife' and the slogan 'The Wall'. On the left, a sidebar lists categories: 'Life (3) ▾', 'TCA (1) ▾', and 'Confidence Level' with options 'High', 'Medium', and 'Low'. A small cartoon character of Snoopy is visible at the bottom of this sidebar. The central area shows a large speech bubble containing a policy number 'XXXXXXXX5456'. To the right is a detailed customer profile for 'Lucy Merryweather' (Owner), including SSN, DOB, Address, Phone numbers, Email, Product Type (Life), Contract Status (Active), Franchise Name (New England Financial), Group Name ([No Data]), Sales Agent (Tommy Topseller), and Agent ID (99B 560). Below the profile is a 'Transaction Details' section with a button to 'Close Transaction Details'. This section shows a summary for the last 60 days: 2 transactions. It lists two items: 'Self Service Address Change' (Completed, Source: Inbound, Channel: eService, System: BOSSLA, Received: January 20, 2012 @ 10:00 am) and 'Self Service Bene Change' (Completed, Source: Inbound, Channel: eService, System: BOSSLA, Received: June 1, 2011 @ 11:30 am). Each item has a 'View Documents (2)' link and a list of associated documents.

业务难点

- 来自60多个国家的9000多万用户，数据量大。
- 70多套不同的保险业务系统的数据需要汇集到一起。
- 已有系统在不断迭代，导致最终数据模型不断变化。
- 关系数据库的结构变化复杂性高，流程长，往往一个迭代未完成，源头系统又变化了。

关系型 vs MongoDB

使用关系数据库的尝试：

- 历时2年
- 前后分别使用2个不同的关系型数据库，
两个不同的厂商/团队
- 花费 \$2500万美元

结果：失败，schema 太复杂，无法有效的
把70套系统的 schema 融合成一套。

使用 MongoDB 的尝试：

- 动态数据模型轻松接收不同数据
- 7x24小时高可用

结果：2个星期做原型，3个月上线。

MongoDB 的数据模型

```
{
    "_id" : "UNIQUE Database ID",
    "CONCAT_KEY_ID" : "Auto Policy - LOB + Policy 3 + Policy Suffix",

    "PRTL_KEY" :
    {
        "AGRE_TYP_DSCR" : {"Life" | "Group" | "Annuity" | ...},
        "KEY_1_VAL" : {"Policy" | "Group_ID" | "Contract_ID" | ...},
        "KEY_2_VAL" : {"Franchise" | "Emp_ID" | "Franchise" | ...},
        "KEY_3_VAL" : {"Suffix" | "TBD" | "" | ...}
    },
    "COMMON_KEYS" :
    {
        "POL_NUM" : "Policy Number",
        "CNTRC_NUM" : "Contract Code - **THIS MAY BE CONSOLIDATED WITH POLICY NUMBER**",
        "POL_SFX_NUM" : "A Suffix Number to Uniquely Identify Policies",
        "SRV_CD" : "Used to identify policies",
        "SEQ_NUM" : "A Policy Sequence Number",
        "GRP_NM" : "Group Name, INST Only",
        "SRC_GRP_NUM" : "Group Number as represented in INST source systems",
        "GRP_NUM" : "Common Group Number across INST currently managed in CDF",
        "SUBDIV_ID" : "Common Sub Division Number across INST currently managed in CDF",
        "CLM_BR_ID" : "Branch Number across INST currently managed in CDF",
        "CVR_TYP_DSCR" : "Coverage Type represents product for INST currently housed in UIS"
    },
    "DENTAL_POLICY":
    {
        "PLAN_TYPE": "PPO | PMO",
        "ORTHONTICS_INCLUDED": "YES|NO"
    },
    "AUTO_POLICY":
    {
        "YEAR": "2015",
        "MAKER": "BMW",
        "MODEL": "X5"
    }
}
```

这个子文档仅在牙医保单内存在

这个子文档仅在汽车保单内存在

MongoDB 的数据模型

```
"PARTY" :  
[  
  {  
    "ROLE_DSCR" : "Owner",  
    "OWN_ROLE_DSCR" : "Need to define ... values to indicate Primary or Secondary",  
    "TYP_CD" : "Individual (NEED TO DEFINE TYPE CODES.. Person, Organization, etc",  
  
    "FULL_AGRF_PARTY_HASH_ID" : "identification string generated based on the value of the source record. determining if records have changed",  
    "RQR_MTCH_FLD_HASH_ID" : "identification string generated based on the value of the concatenated fields identified for use in match",  
    "ROOT_ORG_NM" : "Standardized Organization Name",  
    "ROOT_FRST_NM" : "Standardized Party First Name",  
    "ROOT_MID_1_NM" : "Standardized Party Middle Name",  
    "STD_GEN_OF_NM" : "Standardized generation of name used for matching",  
  },  
,  
  
"AGNT" :  
[  
  {  
    "ROLE_DSCR" : "Service Agent is planned for phase 1. Other roles like Writing Agent or Manager exist.",  
    "FULL_NM" : "Name - First Middle Last of the agent that services the policy",  
    "ID" : "DAI",  
    "TEL_NUM" : "Phone Number",  
  },  
  {  
    "ROLE_DSCR" : "Service Agent is planned for phase 1. Other roles like Writing Agent or Manager exist.",  
    "FULL_NM" : "Name - First Middle Last of the agent that services the policy",  
    "ID" : "DAI",  
    "TEL_NUM" : "Phone Number",  
  }  
]
```

MongoDB 的数据模型

保单：13张关联表 => 1张保单表：让存储、查询变得更简单和灵活。

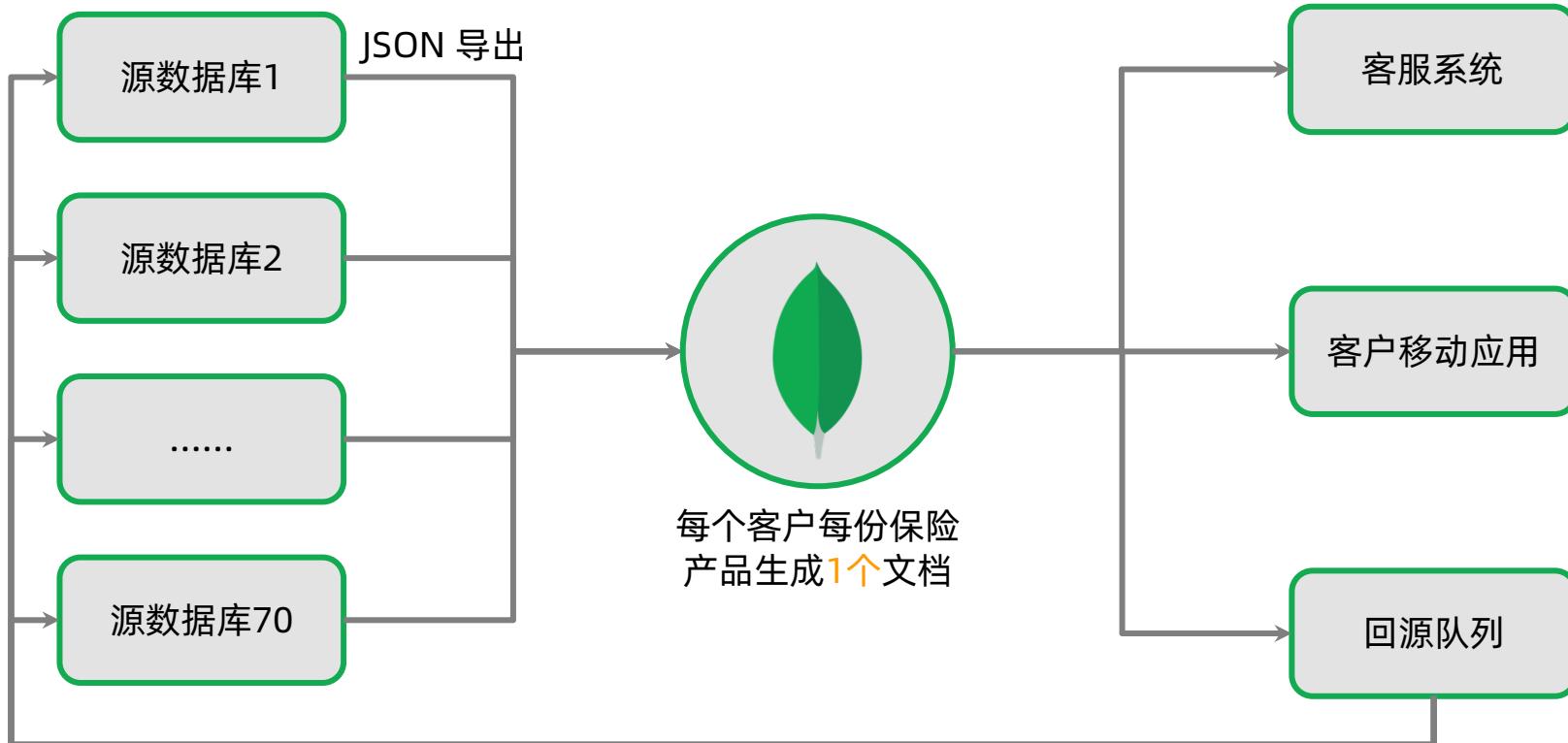
现有模型

No.	表名
1	TBL_POLICY_ORPHAN
2	TBL_POLICY_ORPHAN_NEW
3	TBL_POLICY_ORPHAN_N1
4	RPT_ACTIVESECTION_ORPHAN
5	TBL_QG_BNKPOLICY
6	CRM_SPREREC
7	P10I_MD_RISKCON
8	P10I_MD_PERSON
9	P10I_MD_PHONE
10	P10I_MD_SPREREC
11	P10I_MD_EMPNO
12	P10I_MD_ADDRESS
13	P10I_TD_PREREC

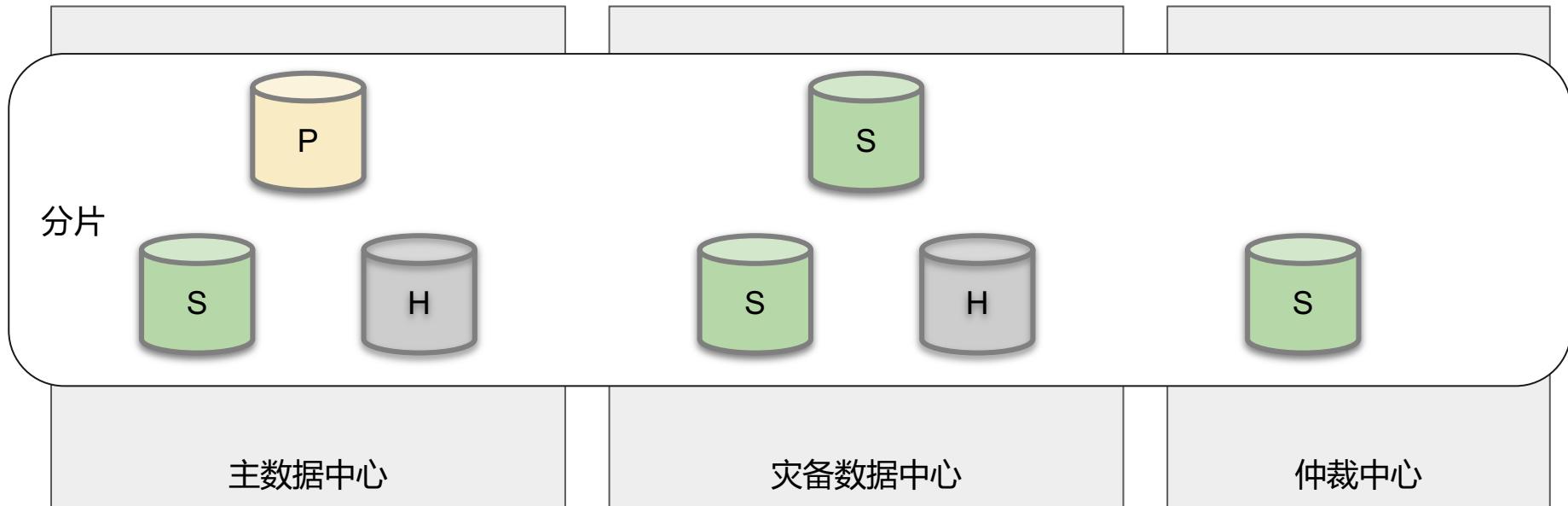
MongoDB模型

```
{  
    POLICYNO: "XXXXXX",  
    CLASSCODE: "XXXXXX",  
    TYPEID: 10101,  
    APID: "XXXXXXXXXX",  
    ORPHAN: 1  
    AIDTYPE : "XX",  
    GPOLICYNO: "XXXXXX",  
    DESKPAY : "X",  
    RENEWID : "X",  
    RENEWDATE : ISODate("XXXX-XX-XX"),  
    BEGDATE : ISODate("XXXX-XX-XX"),  
    STOPDATE : ISODate("XXXX-XX-XX"),  
    POLIST : "X",  
    REASON : "xxxxxxxxxxxx xxxx" ,  
    ..  
    ..  
    PersonList: [...],  
    PhoneList : [... ],  
    AddressList : [...] }
```

系统架构



系统架构



案例小结

此案例利用了 MongoDB 的以下特性：

- 反范式的数据模型使得复杂数据整合成为可能
- 高可用（本地和跨机房）
 - 故障发生时应用可以自动切换到正常的节点上
 - 可以在秒级时间内完成故障转移，使得用户体验得到保证

因为这些特性的存在，使得项目成功。

案例二：主机分流（主机下行）

国内四大行之一

业务需求

为提升用户体验，该银行要在手机银行APP中支持实时账户交易历史查询。涉及的数据包括：

- 借记卡交易历史。
- 信用卡交易历史。
- 后续还将支持股票、基金账户等。

对这些交易历史进行整合，使用户可以看到自己账户的交易历史全貌。涉及的交易数据量：

- 约6000万交易数据/天，结息日达到~5亿/天。
- 历史存量数据3年，超过600亿。
- 核心系统（主机）支持这样的流量非常困难，成本太高。

方案选择：主机分流

关系型数据库：

- 超大量数据需要巨大数量的 DB 实例。
- 考虑高可用，成本极高。
- 分库分表造成开发难度大幅上升。
- 整合不同账户数据时表结构差异大，合并困难。

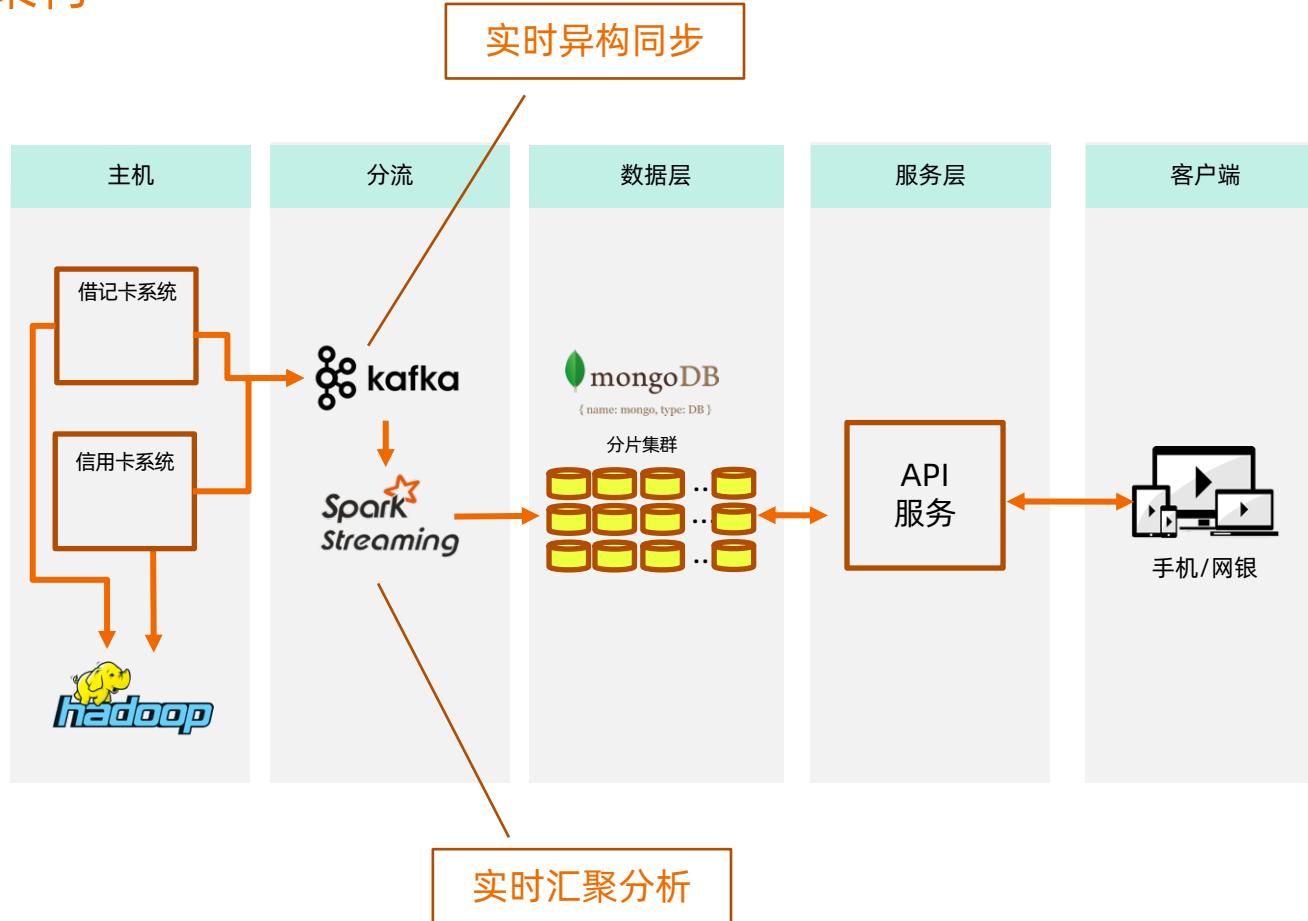
结果：评审期被否定

使用 MongoDB 的尝试：

- 动态数据模型轻松接收不同数据。
- 水平扩展解决大数据量问题。
- 7x24小时可用。

结果：成功上线

系统架构



案例小结

主机分流架构特点：

- 处理接近每秒1万的交易数据
- 总量数百亿
- 查询平均性能数毫秒

本案例中利用了 MongoDB 的以下特性：

- 灵活数据模型：使数据整合更为容易
- 弹性扩展：使得海量数据 + 低延迟查询成为可能

4.3 MongoDB 典型案例（二）

案例三：MySQL 迁移 顶级互联网公司

业务场景

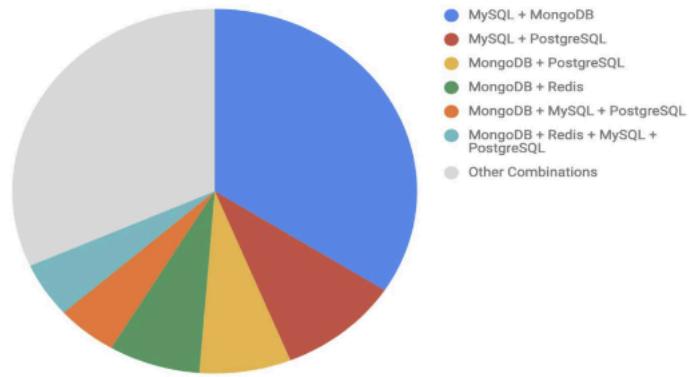
- 旗下多个 App
- DAU 7 亿
- 42万+服务器
- 每天新增30PB数据（包含非结构化）
- 每日线上变更 6000+
- 标准数据方案
MySQL + Redis + 对象存储
- 结构化和半结构化在MySQL
- MySQL：数万台服务器

痛点

- 数据库变更需要团队配合，对6000+/天的频繁变更造成很大阻碍
- MySQL 集群本身扩容比较困难
- 中间件的约束较多
- 迭代速度受影响

新的方向

MySQL + MongoDB: 34.15%
MySQL + PostgreSQL: 9.76%
MongoDB + PostgreSQL: 7.32%
MongoDB + Redis: 7.32%
MySQL + MongoDB + PostgreSQL: 4.88%
MySQL + MongoDB + PostgreSQL + Redis: 4.88%



- MongoDB 无须中间件，改善变更效率
- 集群扩容容易
- 结构灵活，迭代快

MongoDB 业务场景

在线 TP 业务

- 数据模型多变，新增 collection 比较常见
- 低时延和少毛刺
- 请求量大

中台元数据管理

- 中台系统，schema 需要很多嵌入式文档
- 数据量大，点查为主

LBS 地理位置

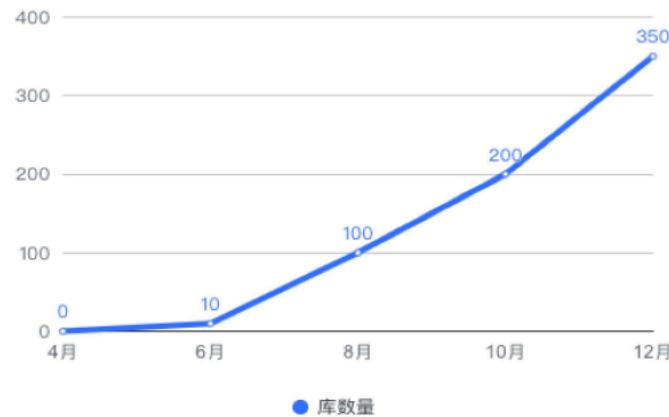
- 计算密集，数据点小，但是量大
- 写入更新异常频繁

游戏

- 游戏日志写入量大，查询量一般
- 在线分析需求

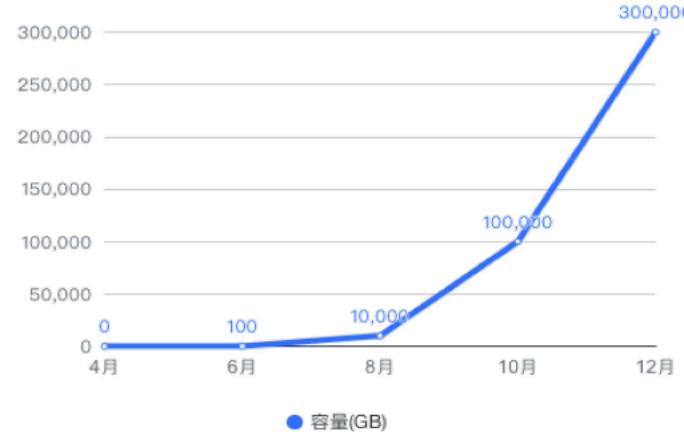
MongoDB 使用现状

用户库增长曲线



350 Apps

容量增长曲线



300TB

案例四：海量数据存储 顶级互联网公司



国内最大搜索厂商

亿级用户网盘应用

挑战

网盘通讯录，短信存储管理

网盘文件元数据管理

用户操作日志

数亿用户体量

MySQL集群无法支撑性能的诉求

解决方案

迁移到MongoDB分片集群

快速增长时期每三个月扩容一次

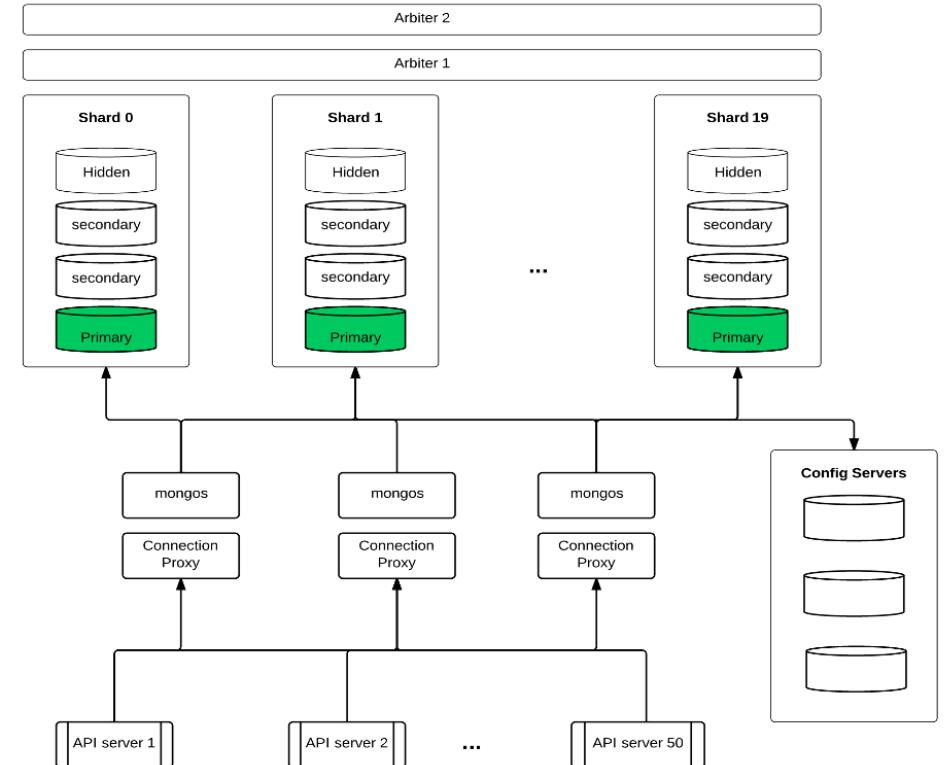
系统2012年上线运行至今

有力支撑业务的持续发展，目前支撑100多个业务

2PB+ 的数据存储量

100多个分片

技术架构



架构特点

- 分片集群
- 片键: userid
- 并发5000/s/节点
- 通过自建proxy限流

硬件规范:

- CPU: 8 Core
- RAM: 48G
- Storage: 2T SSD, RAID 0



世界顶级网络设备生产商

电商平台重构

挑战

电商平台承担数百亿美元的订单

网页性能带来糟糕的用户体验：页面显示需要超过5秒

平台无法满足市场业务需求的速度，无法及时发布新功能

传统架构难以用上云架构和DevOps的红利

解决方案

整个电商平台数据库从Oracle整体迁移到MongoDB

14个关系型表及合成1个集合

60个索引优化到7个

减少了12万行代码

3-5秒的页面刷新时间降低到小于1秒



世界顶级航旅服务商

亿级数据量实时分析

挑战

服务 124 个航空公司

每天处理16亿预订相关的请求

需要为乘客提供一个个性化的预订体验，需要做大量的预先聚合计算

解决方案

100TB的数据存储在大型MongoDB集群

32 台物理机，288个微分片

一个计算任务在288个微分片上同时执行，大幅度提高分析计算的效率

大型分析场景响应时间在数秒内完成



世界顶级数据治理厂商

自然模型主数据存储

挑战

主数据管理需要整合企业多套业务系统的核心数据

传统方式使用关系模型和第三范式

很多精力花在业务逻辑到物理模型的转换，再从物理模型转化为业务逻辑的过程中

客户的数据量很大的时候大量关联导致性能问题

解决方案

使用MongoDB来存储主数据

JSON模型比较自然的表现业务的逻辑

查询和写入性能较之于Oracle 方案高了20倍

4.4 关系型数据库迁移

从基于关系型数据库应用迁移到 MongoDB 的理由

- 高并发需求 (数千 - 数十万 ops) , 关系型数据库不容易扩展
- 快速迭代 - 关系型模式太严谨
- 灵活的 JSON 模式
- 大数据量需求
- 地理位置查询
- 多数据中心跨地域部署

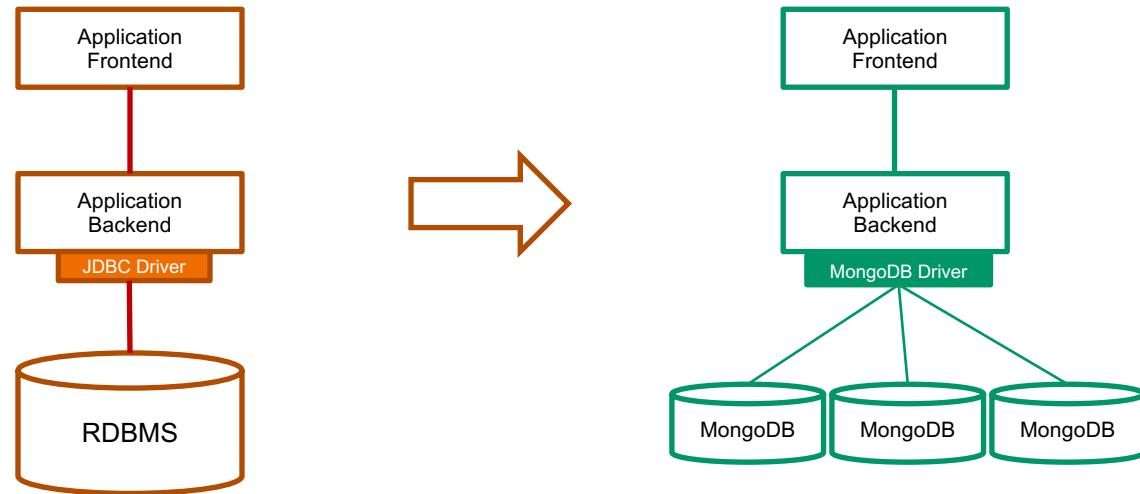
应用迁移难度

- 关系型到关系型 - 相对简单
 - Oracle -> MySQL, Oracle - PostgreSQL
- 关系型到文档型 - 相对复杂
 - Oracle -> MongoDB
- 需要考虑:
 - 总体架构 (从单体式到分布式)
 - 模式设计 (从关系模型到文档模型)
 - SQL 语句 / 存储过程 / JDBC / ORM
 - 数据迁移 (如何处理已有数据?)

总体架构

从单体到分布式，需要考虑：

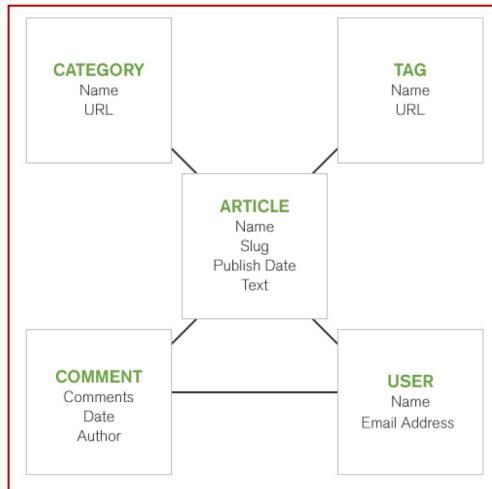
- 3x 的计算资源
- 3x 的存储资源
- 网络



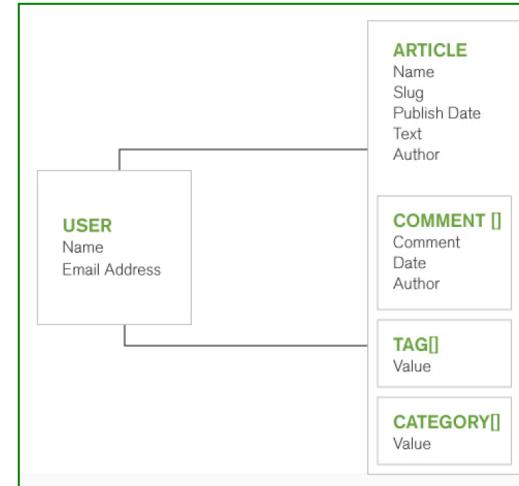
模式设计

针对已有关系模型，考虑如何用文档模型进行设计

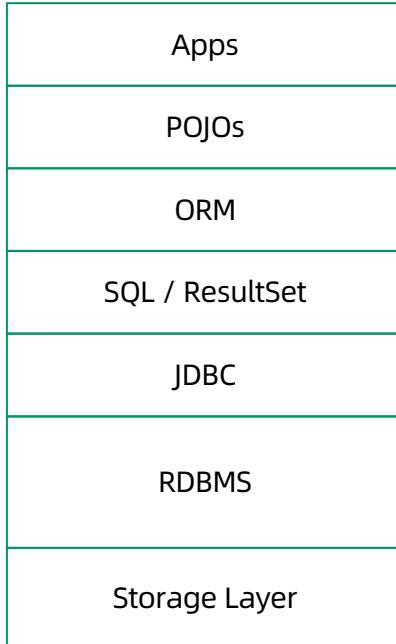
5个关系型表



2 个集合 (MongoDB)

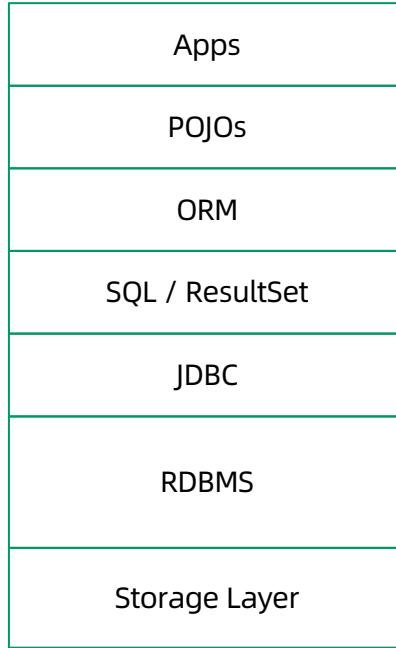


迁移的主战场



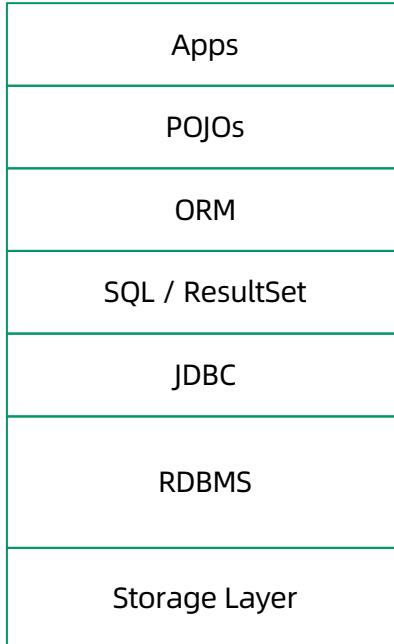
- 存储过程
- 运维工具、脚本
- 权限设置
- 数据库监控备份及恢复

硬件存储



典型的关系型数据库部署在 SAN 上
MongoDB 支持 SAN, 但是使用本地存储
可以最大化的提高性能

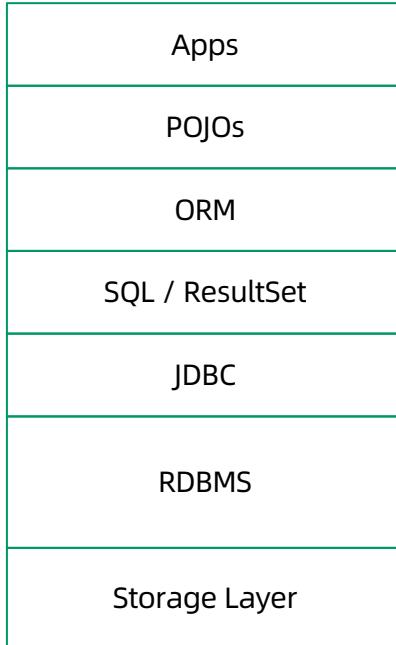
JDBC



MongoDB 没有原生态 JDBC, 而是采用自带的驱动程序:

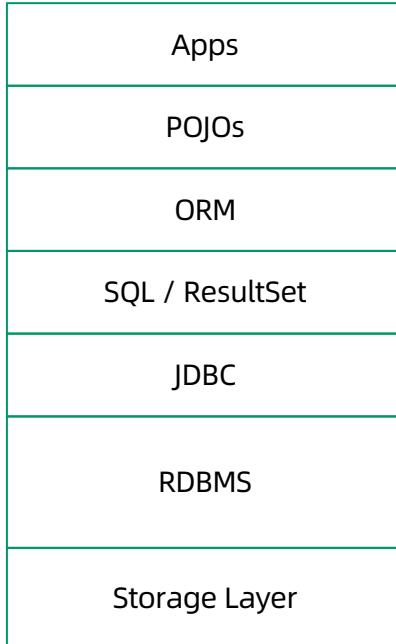
- 自带连接池管理
- 事务支持

SQL



MongoDB 不支持SQL的增删改，结
果集也不是 ResultSet

ORM



ORM: Object Relational Mapping
转换关系型到POJO对象模型

不需要，但是可以有ODM

ODM: Object Document Model

Spring Data
mongoose

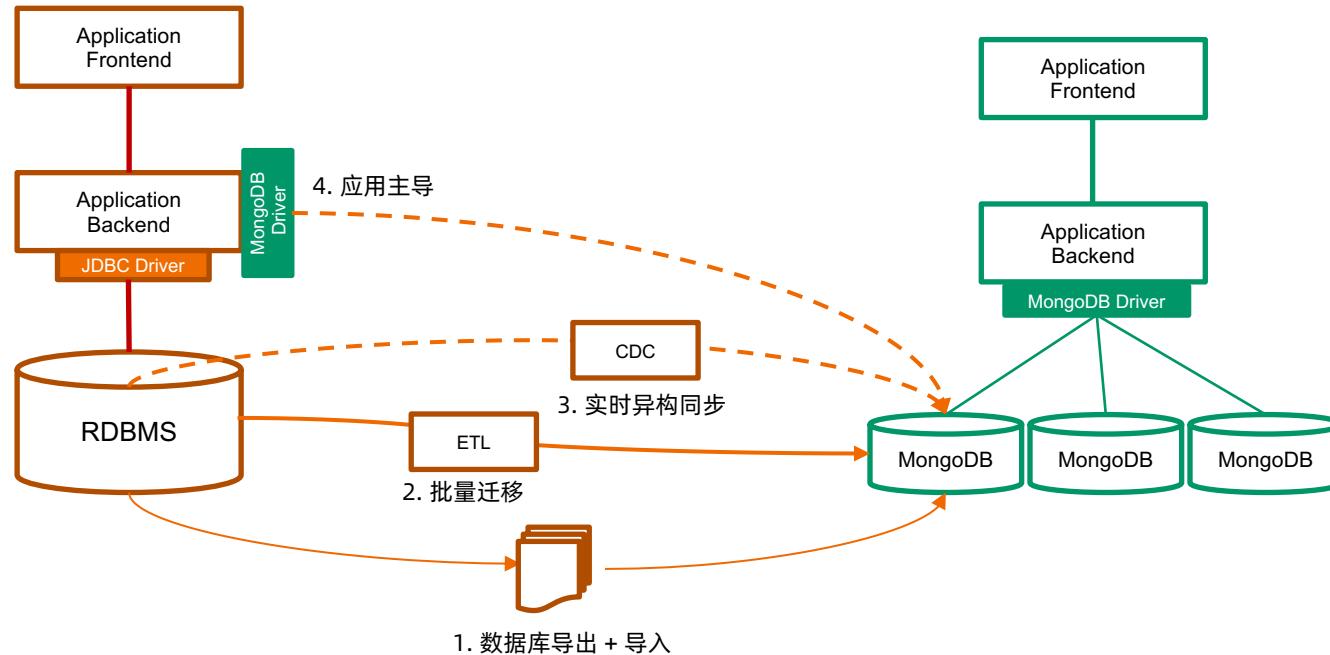
数据迁移

- 迁移时应用往往已经上线相当长一段时间，如何迁移这些数据到 MongoDB？
 - 数据库导出+导入
 - 批量迁移工具
 - 实时同步工具
 - 应用主导迁移

4.5 数据迁移方式及工具

数据迁移

如何迁移已有数据到 MongoDB?



1. 数据库导出导入

步骤：

- 停止现有的基于 RDBMS 的应用
- 使用 RDBMS 的数据库导出工具，将数据库表导出到 CSV 或者 JSON（如 mysqldump）
- 使用 mongoimport 将 CSV 或者 JSON 文件导入 MongoDB 数据库
- 启动新的 MongoDB 应用

备注：

- 适用于一次性数据迁移
- 需要应用/数据库下线，较长的下线时间

数据库导出导入: mysql - mongo

```
# mysqldump inventory -hxxxx -uroot -p -T mysql-files  
  
# cat customers.txt  
  
"1001","Sally","Thomas","sally.thomas@acme.com"  
"1002","George","Bailey",gbailey@foobar.com  
  
# mongoimport -d xxx -c customers --type=csv --headerline customers.txt  
# mongoimport -d xxx -c products --type=csv --headerline products.txt  
# mongoimport -d xxx -c orders --type=csv --headerline orders.txt
```

2. 批量同步

步骤：

- 安装同步工具（如 Kettle / Talend）
- 创建输入源（关系型数据库）
- 创建输出源（MongoDB）
- 编辑数据同步任务
- 执行



备注：

- 适用批量同步，定期更新，特别是每晚跑批的场景
- 支持基于时间戳的增量同步，需要源表有合适的时间戳支持
- 对源库有较明显的性能影响，不宜频繁查询
- 不支持实时同步

3. 实时同步

步骤：

- 安装实时同步工具（如Informatica / Tapdata）
- 创建输入源（关系型数据库）
- 创建输出源（MongoDB）
- 编辑实时数据同步任务
- 执行



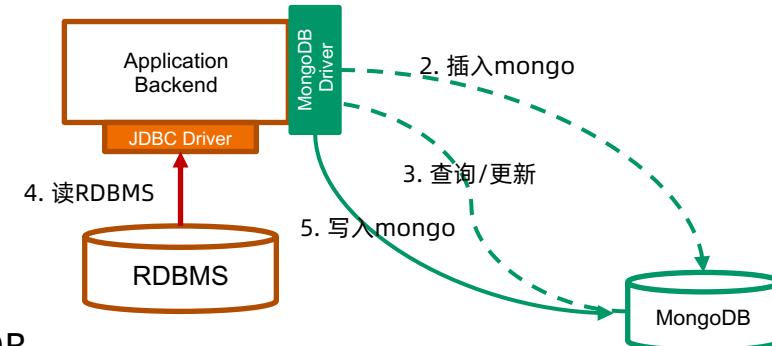
备注：

- 基于源库的日志文件解析机制，可以实现秒级数据的同步
- 对源库性能影响较小
- 可以支持应用的无缝迁移

4. 应用主导迁移

步骤：

1. 升级已有应用支持 MongoDB
2. 数据插入请求直接进入 MongoDB
3. 数据查询和更新请求首先定向到 MongoDB
4. 如果记录不存在，从 RDBMS 读出来并写入到 MongoDB
5. 重复步骤3
6. 当步骤4在限定时间段（一星期、一个月）没有被调用，认为迁移完成



备注：

- 需要研发团队配合，有一定开发和测试量
- 为保证不遗漏数据，仍然先要执行一次批量同步

数据迁移方式比较

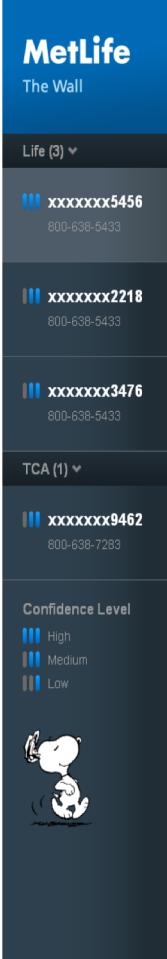
方法	技术实现方式	特点	适用场景
数据库导出导入	数据库自带工具 自己编写简单脚本	简单方便 一次型迁移全量数据	基于已有数据的新应用 旧应用更新换代，采用重写方式
批量同步	Talend Pentaho 或使用脚本程序	适用 SQL，批量查询出需要数据， 写到目标端 无需代码开发	
实时同步	Informatica Oracle Golden Gate Tapdata	基于数据库日志解析并实时写入到 目标端 准实时同步 无需代码开发	无缝切换 老应用不下线，和新应用并存 主机下行 关系库加速 实时数据平台
应用迁移	在已有应用中增加对 MongoDB 的支持	需要较多的开发和测试支持	旧应用更新换代，采用修改升级原有应用方式

4.6 Oracle 迁移实战

保险公司单一视图的需求

跨国保险公司，来自60多个国家的9000多万用户，70多套业务系统。客户信息分散在多套系统里，希望构建一个客户360视图。

为了达到这些目的，需要整个70+历史系统中的客户与产品信息，通过唯一入口查询



MetLife
The Wall

- Life (3) ▾
- ||||| xxxxxxxx5456
800-638-5433
- ||||| xxxxxxxx2218
800-638-5433
- ||||| xxxxxxxx3476
800-638-5433
- TCA (1) ▾
- ||||| xxxxxxxx9462
800-638-7283

Confidence Level

- ||||| High
- ||||| Medium
- ||||| Low



Quick Search Contract key # Advanced Search

The Wall

Policy # XXXXXXXX5456

Customer Service Phone Number 800-638-5433

Logout

Lucy Merryweather

Owner

SSN	Phone
XXX-XX-XXXX	(819) 555-7702
DOB	Alternate Phone
August 10, 1973	(819) 555-2231
Address	Email
109 West 93rd Avenue Lincoln, OK 07882	lkmerry@gmail.com

Product Type

Life

Contract Status

Active

Franchise Name

New England Financial

Group Name

[No Data]

Group Number

[No Data]

Sales Agent

Tommy Topseller

Agent ID

99B 560

Close Transaction Details

Transaction Details All Transactions ▾ All Service Channels ▾ All Sources ▾

Last 60 days • 2 transactions

 Self Service Address Change

Status: Completed | Source: Inbound | Channel: eService | System: BOSSLA

▼ View Documents (2)

Change Confirmation (Mailed) • January 24, 2012 @ 4:31 pm

Change Request • January 20, 2012 @ 9:50 am

Received: January 20, 2012 @ 10:00 am

 Self Service Bene Change

Status: Completed | Source: Inbound | Channel: eService | System: BOSSLA

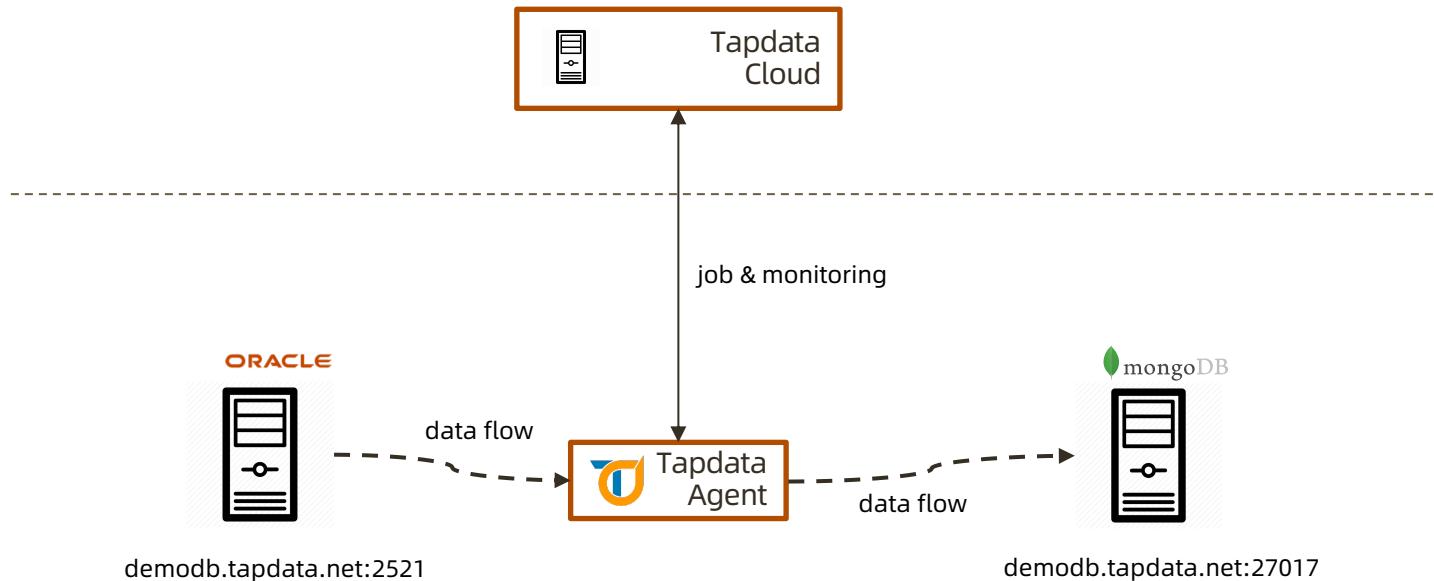
▼ View Documents (2)

Change Confirmation (Mailed) • June 3, 2011 @ 6:03 pm

Change Request • June 1, 2011 @ 11:20 am

Received: June 1, 2011 @ 11:30 am

测试环境



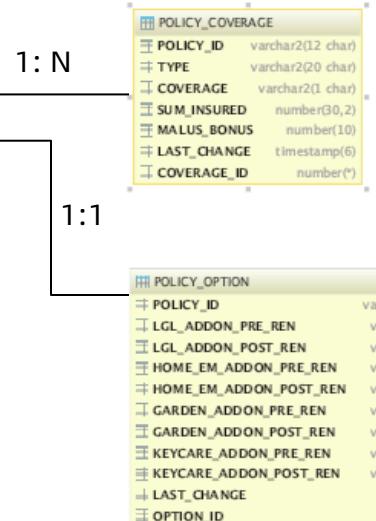
步骤

1. 比较 Oracle 关系模型和目标 JSON 数据模型
2. 注册&下载 Tapdata Agent
3. 创建 Oracle 连接 和 MongoDB 连接
4. 模型转换设计和同步任务编辑
5. 运行任务
6. 检查结果

1. Oracle 关系模型 -> 目标 JSON 模型

POLICY	
POLICY_ID	varchar2(12 char)
CUSTOMER_ID	varchar2(12 char)
QUOTE_DAY	date
COVER_START	date
LAST_ANN_PREMIUM_GROSS	number(10,2)
POLICY_STATUS	varchar2(100 char)
LAST_CHANGE	timestamp(6)

POLICY_RISK	
POLICY_ID	varchar2(12 char)
RSK_CLASSIF_BLDG	number(10)
RSK_CLASSIF_PRSNL	number(10)
APPR_ALARM	varchar2(1 char)
APPR_LOCKS	varchar2(1 char)
BEDROOMS	number(10)
ROOF_CNSTRCTN	number(10)
WALL_CONSTRCTN	number(10)
FLOODING	varchar2(1 char)
LISTED	number(10)
MAX_DAYS_UNOCC	number(10)
NEIGH_WATCH	varchar2(1 char)
OCC_STATS	varchar2(2 char)
OWNERSHIP_TYPE	number(10)
PAYING_GUESTS	number(10)
PROP_TYPE	number(10)
SAFE_INSTALLED	varchar2(1 char)
YEARBUILT	number(10)
LAST_CHANGE	timestamp(6)

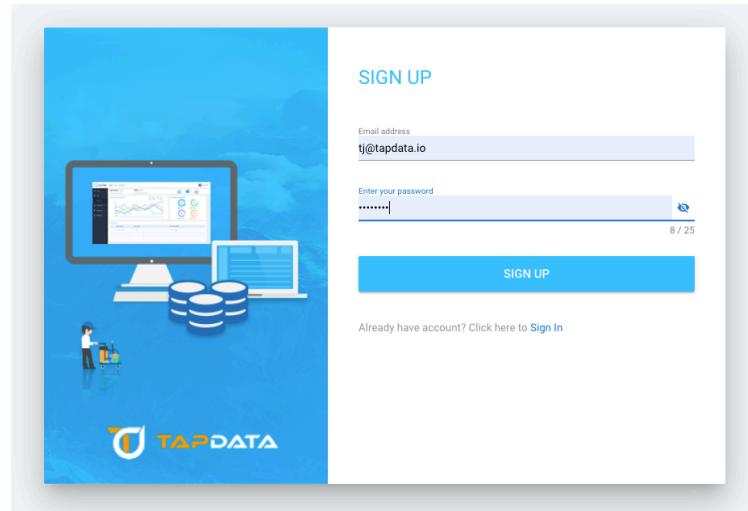


```

_id: ObjectId("5e47f3d2704d02e838758c1b")
POLICY_ID: "P000000002"
COVER_START: 1999-09-01 03:23:40.000
CUSTOMER_ID: "C000060613"
LAST_ANN_PREMIUM_GROSS: 308
LAST_CHANGE: 2019-07-04 09:13:55.740
POLICY_STATUS: "Live"
QUOTE_DAY: 1999-08-01 03:23:40.000
POLICY_COVERAGE: Array
  ▼ 0: Object
    SUM_INSURED: 1000000
    MALUS_BONUS: 6
    POLICY_ID: "P000000002"
    TYPE: "BUILDING"
    COVERAGE: "Y"
    LAST_CHANGE: 2019-07-04 17:13:55.740
  ▼ 1: Object
    SUM_INSURED: 50000
    MALUS_BONUS: 7
    POLICY_ID: "P000000002"
    TYPE: "PERSONAL_ITEMS"
    COVERAGE: "Y"
    LAST_CHANGE: 2019-07-04 17:13:55.740
  ▼ POLICY_OPTION: Object
    GARDEN_ADDON_POST_REN: "N"
    GARDEN_ADDON_PRE_REN: "N"
    HOME_EM_ADDON_POST_REN: "N"
    HOME_EM_ADDON_PRE_REN: "N"
    KEYCARE_ADDON_POST_REN: "N"
    KEYCARE_ADDON_PRE_REN: "N"
    LAST_CHANGE: 2019-07-04 17:13:55.740
    LGL_ADDON_POST_REN: "Y"
    LGL_ADDON_PRE_REN: "Y"
    POLICY_ID: "P000000002"
  
```

2. 注册下载 Tapdata Agent

- <http://cloud.tapdata.net>
- 注册免费账号
- 下载客户端
- 支持数据库：MongoDB, Oracle, MySQL , SQLServer, DB2, PostgreSQL, GaussDB, GBase
- 秒级同步，多表合一能力



3. DEMO

- 创建 Oracle 连接 和 MongoDB 连接
- 模型转换设计和同步任务编辑
- 运行任务
- 检查结果

小结

- 大部分的企业应用运行在 Oracle 为代表的关系型数据库
- 运行了数十年的业务往往有：1) 性能瓶颈 2) 支持创新困难 的特点
- 直接替换这些业务难度很大，风险很高
- 使用实时同步工具，将数据同步到 MongoDB，快速构建新应用，是一种低风险高成效的应用模式。

4.7 MongoDB + Spark

什么是 Spark ?

Apache Spark™ is a fast and general engine for large-scale data processing.

通用，快速，大规模数据处理引擎

Spark SQL

Streaming

Machine Learning

Java, Python, Scala, R

内部循环数据流程

全内存计算

可比Map Reduce 快100倍

横向扩展，集团作战

HDFS原生支持

我能用它做什么？

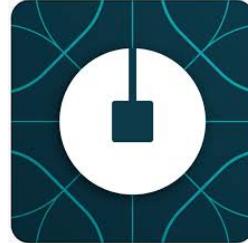
个性化



产品推荐



流处理

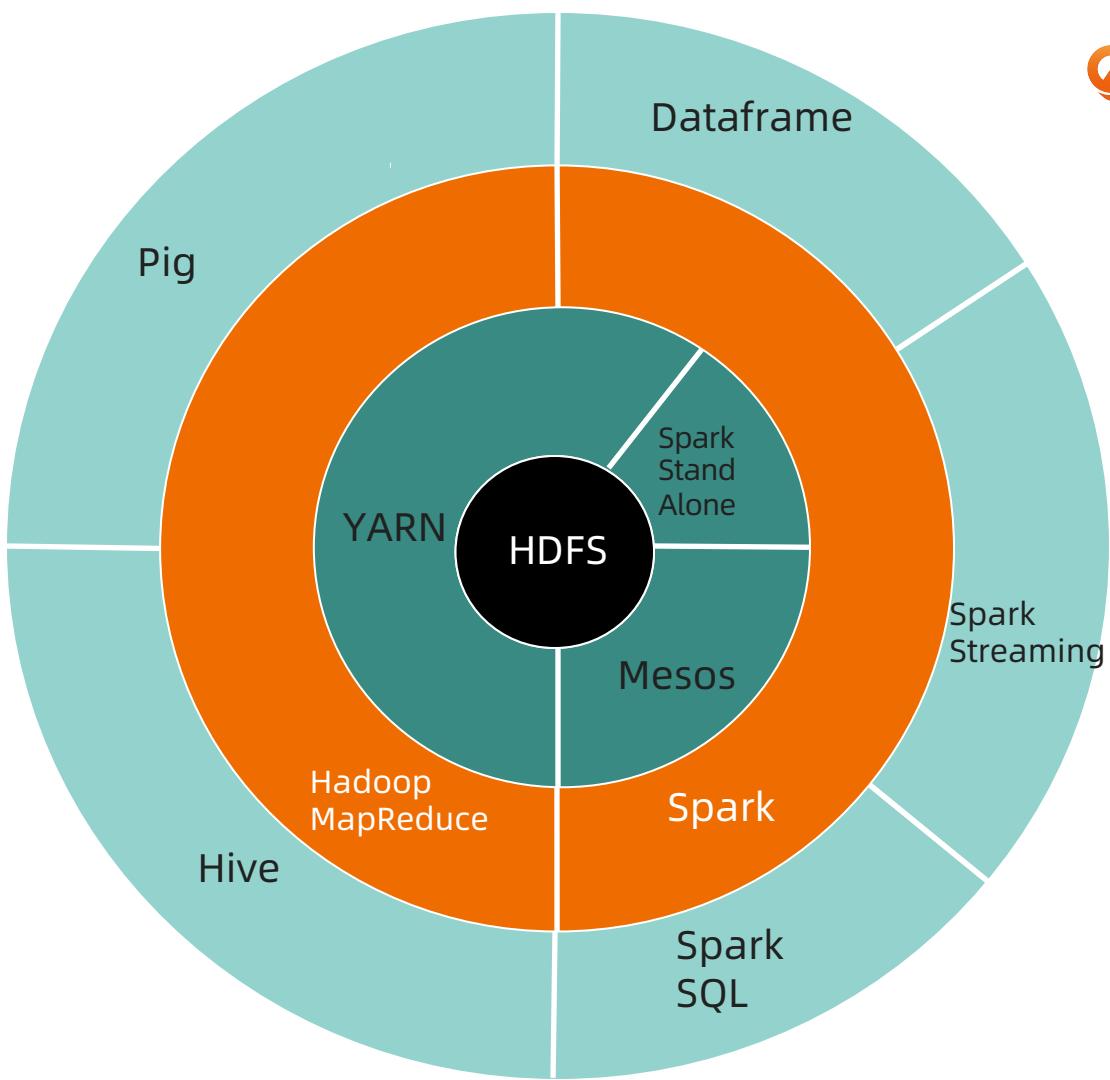


商业智能



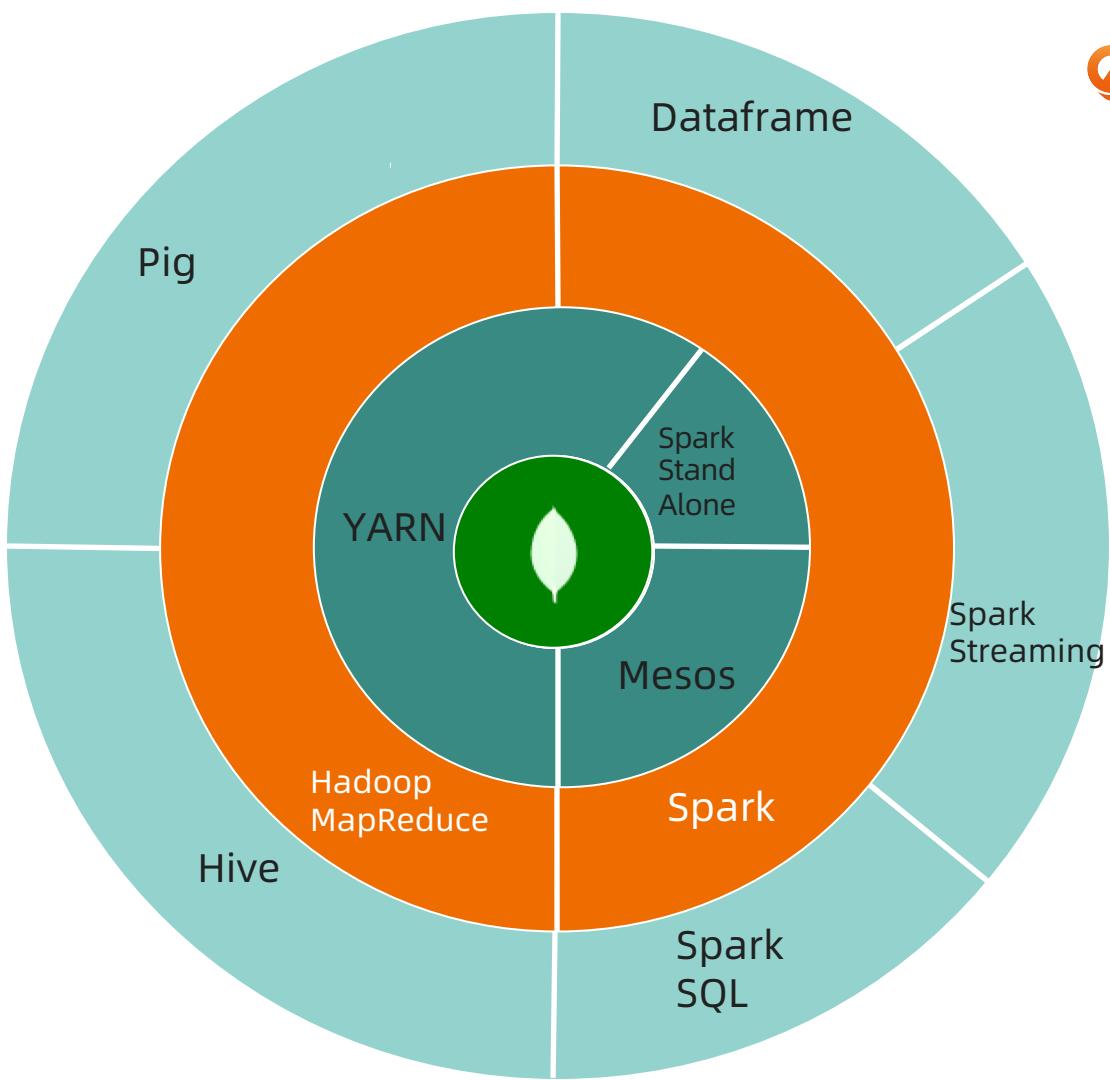
Spark 生态系统

程序接口
计算引擎
资源管理
数据存储



Spark 生态系统

程序接口
计算引擎
资源管理
数据存储



HDFS vs. MongoDB

	HDFS	MongoDB
共同点	横向扩展，支持 TB-PB 级数据量 低成本，x86 数据自动多份复制 支持非结构化数据	
	粗颗粒度存储	细颗粒度，结构化存储
差异点	无索引	支持使用索引，更加快速
	一次写入，多次读	读写混合
	非交互式，离线 分钟级 SLA	交互式，实时在线 毫秒级 SLA

一个日志的例子



2016-07-31.log



2016-07-30.log



...

2016-07-01.log



索引 { error: 1 }

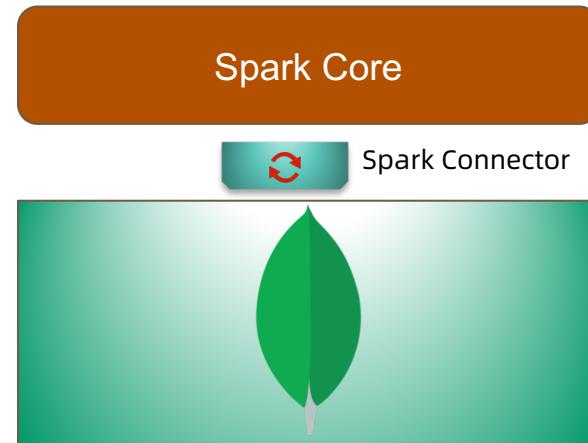
```
{ ts: 2016-07-31 23:50:50, host: xyz, error:404, .body: {} ...}  
{ ts: 2016-07-31 23:49:23, host: def, error:019, .body: {} ...}  
{ ts: 2016-07-31 23:49:22, host: xyz, error:null, body: {} ...}  
...  
{ ts: 2016-07-01 02:04:12, host: abc, error: 500, body: {} ...}
```

使用场景	HDFS	MongoDB
7月1日到31日所有页面的点击量统计	OK	OK
每日HTTP 404错误日志数量统计	低效：需要扫描所有文件行	可利用索引 秒级响应
对日志行增加自定义字段 保存分析结果	不支持	OK

Spark Connector

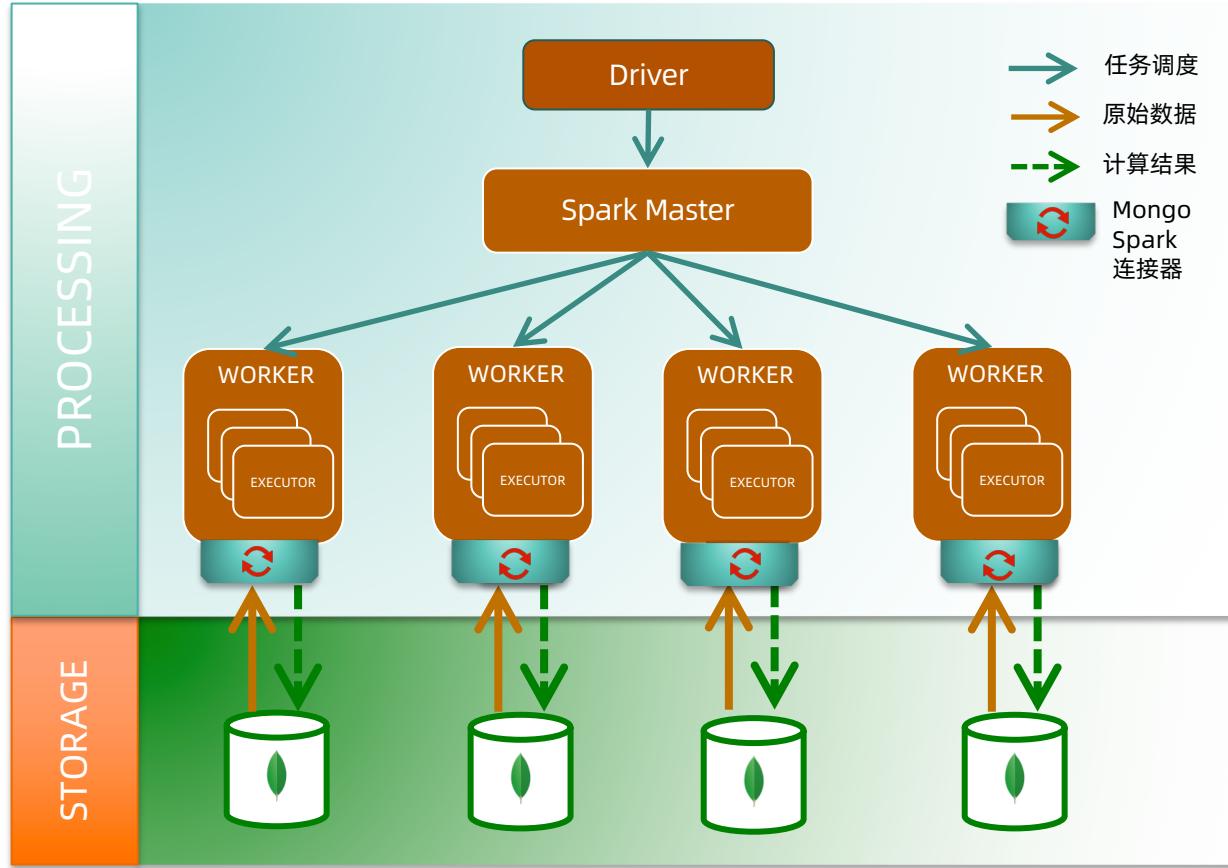
让 Spark 连接 MongoDB 非常简单，使用 Spark Connector。

开发者：MongoDB 公司

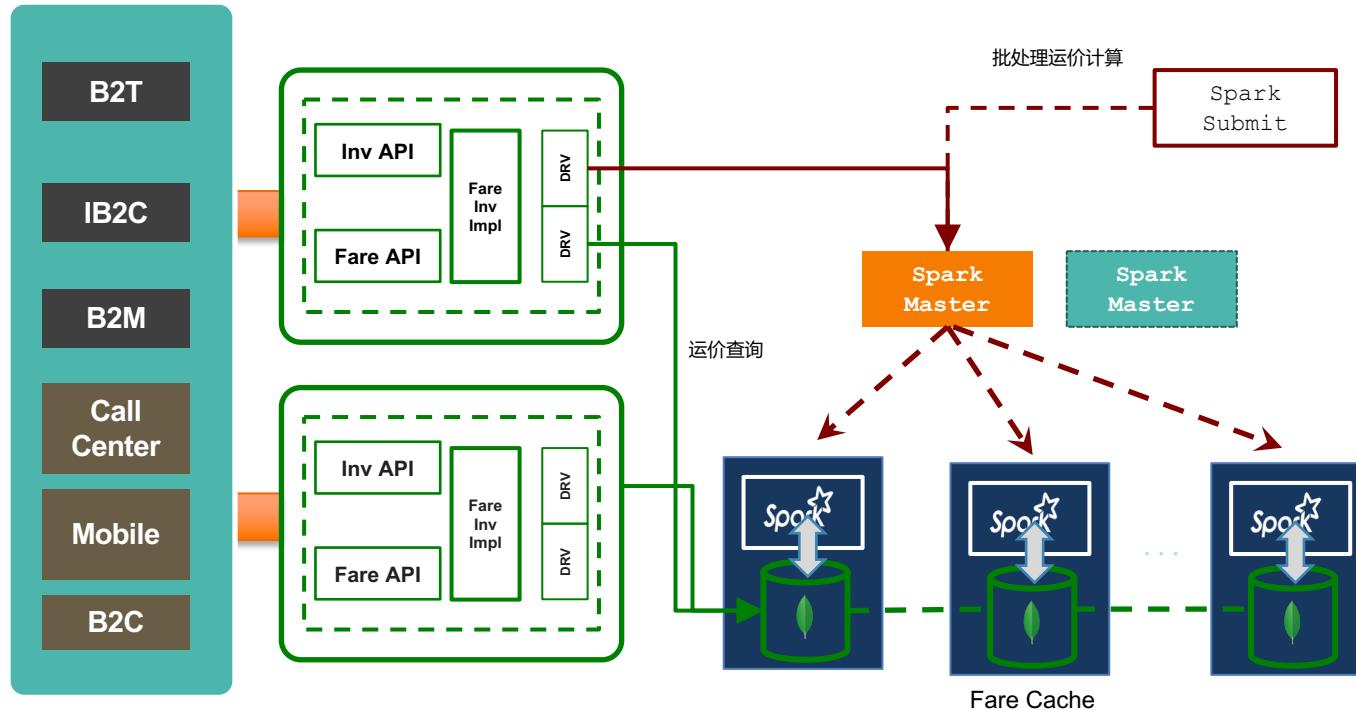


<https://github.com/mongodb/mongo-spark>

Spark + MongoDB 架构

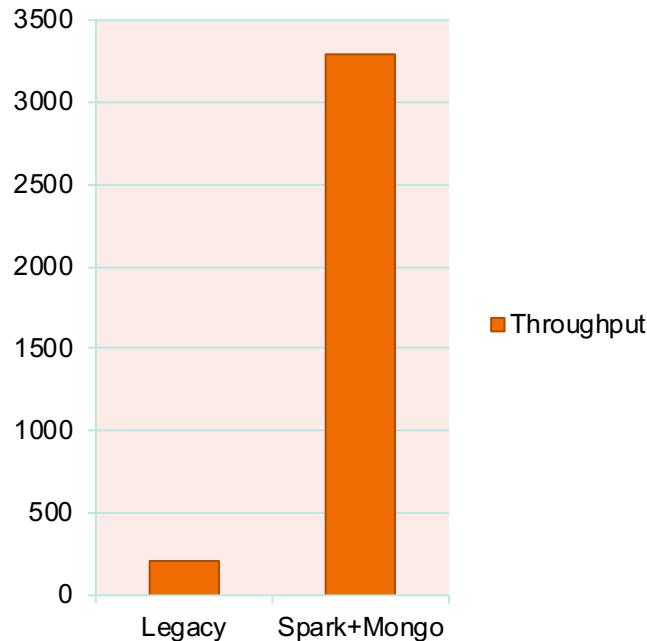


Spark + MongoDB 运价计算及查询案例

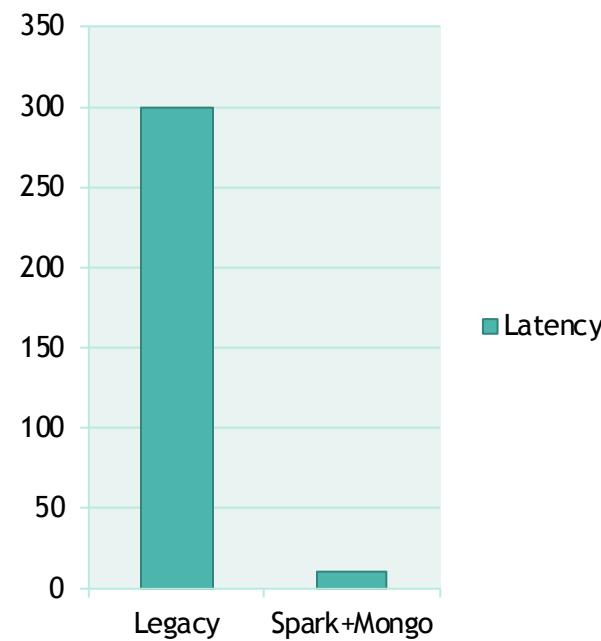


处理能力和响应时间比较

单节点处理能力 (QPS)



查询响应时间



4.8 MongoDB + Spark 连接实战

场景

- 假设我们有以下数据：
 - name: 用户名；
 - address: 住址；
 - birthday: 出生日期；
 - favouriteColor: 最喜欢的颜色；
- 我们想按照月份统计每个月出生的人中，喜欢的人数最多的颜色是什么。

选择合适的版本

Spark Connector 要求使用的 MongoDB 为 2.6 以上。为了在使用过程中不出现兼容性问题，请按照以下版本对应关系使用（更多详情请参考[文档](#)）：

Spark Connector	Spark	Scala	JDK
2.4.x	2.4.x	2.12.x或2.11.x	Java 8以上
2.3.x	2.3.x	2.11.x	Java 8以上
2.2.x	2.2.x	2.11.x	Java 8以上
2.1.x	2.1.x	2.11.x	Java 8以上
2.0.x	2.0.x	2.11.x	Java 6以上
1.1.x	1.6.x	2.11.x或2.10.x*	Java6以上

* Spark Connector 1.1 针对 Scala 2.11.x 和 2.10.x 有不同的二进制包，使用时请注意。

环境配置

- JDK 8:
<https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
- Maven3: <http://maven.apache.org/install.html>

环境配置

- 配置文件： /src/main/resources/config.properties
 - input: 用于计算的源 MongoDB 集合；
 - output: 计算后用于存储输出结果的集合；
- 编译运行：

```
mvn install  
mvn package  
java -jar target/SparkConnectorDemo-1.0-SNAPSHOT-jar-with-dependencies.jar
```

基本配置

```
SparkSession spark = SparkSession.builder()  
    .master("local")  
    .appName("MongoSparkConnectorIntro")  
    .config("spark.mongodb.input.uri", "mongodb://127.0.0.1/demo.Person")  
    .config("spark.mongodb.output.uri", "mongodb://127.0.0.1/demo.Output")  
    .getOrCreate();  
  
JavaSparkContext jsc = new JavaSparkContext(spark.sparkContext());
```

- master: 指定 Spark 的 Master 节点地址。没有独立 Spark 集群的时候可以指定为 local（仅用于测试目的）；
- spark.mongodb.input.uri: 来源 MongoDB 连接字符串
 - demo: 来源数据库名；
 - Person: 来源集合名；
- spark.mongodb.output.uri: 目标 MongoDB 连接字符串
 - demo: 输出数据库名；
 - Output: 输出集合名；

更多配置选项请参考文档：<https://docs.mongodb.com/spark-connector/master/configuration/>

ReadConfig 和 WriteConfig

- 在使用 Spark Connector 的时候，很多 API 中会出现可选的 ReadConfig 和 WriteConfig，它们可以提供读和写方面的定制化修改。例如对于读操作：
 - 配置一个不同于 spark.mongodb.input.uri 中指定的数据库和集合；
 - 配置读取操作的 ReadConcern；
 - 配置读取要使用的 ReadPreference；
 - 更多选项请参考：[ReadConfig](#)；
- 同理，对于写操作也有一些可定制选项，例如：
 - 配置一个不同于 spark.mongodb.output.uri 中指定的数据库和集合；
 - 配置写操作的 WriteConcern；
 - 批量写时是否保证顺序；
 - 更多选项请参考：[WriteConfig](#)；

数据加载和输出

```
List pipeline = Arrays.asList(  
    addFields(new Field("month", new Document("$month", "$birthday")))  
);  
  
JavaMongoRDD<Document> mgoRdd = MongoSpark.load(jsc, rc)  
    .withPipeline( pipeline );  
  
// 各种Spark运算  
  
MongoSpark.save(rdd, writeConfig);
```

- `MongoSpark.load`: 从来源 MongoDB 中加载数据；
- `withPipeline`: 使用指定的 Aggregation 管道对集合中的数据进行预处理。所有 Aggregation 运算符都支持；
- `MongoSpark.save`: 将结果集输出到配置的 MongoDB 中；

条件输出

- 上面讲到的 `MongoSpark.save` 使用了最简单的方式将结果输出到目标集合中：
 - 如果 `rdd` 中的 `_id` 在目标集合中存在，则替换那条记录；
 - 如果 `rdd` 中的 `_id` 在目标集合中不存在，则新增一条记录；
- 如果想根据其他字段更新数据怎么办？

条件输出

```
final WriteConfig writeConfig = WriteConfig.create(jsc).withOptions(new HashMap<String, String>());  
final ReadConfig readConfig = ReadConfig.create(jsc).withOptions(new HashMap<String, String>());  
mgoRdd.foreachPartition(iterator -> {  
    MongoConnector mc = MongoConnector.create(writeConfig.asJavaOptions());  
    mc.withCollectionDo(readConfig, Document.class, collection -> {  
        while(iterator.hasNext()) {  
            Document doc = iterator.next();  
            collection.replaceOne(eq("user_id", "12345678"), doc);  
        }  
        return null;  
    });  
});
```

- `readConfig/writeConfig`: 定义一些读写配置，例如 `readConcern`, `writeConcern`, 或重定义输入输出集合等；
- `foreachPartition`: 在各个分区遍历 rdd 中的内容，`iterator` 中即为每条数据；
- `withCollectionDo`: 从 Spark Connector 获得 `MongoCollection`；

Spark Demo 代码

<https://github.com/geektime-geekbang/geektime-mongodb-course/tree/master/spark-demo>

小结

- MongoDB 只是一个存储，Spark 里面的计算操作基本不变，只有数据输入输出要使用特定 API。
- MongoDB 的优势是可以更加快速，更加大量的为 Spark 提供原始数据。

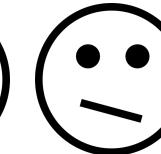
4.9 MongoDB SQL 套接件

MongoDB 支持 SQL 方式查询吗？

- No



```
select count(*) from User  
where favoriteColor='purple'
```



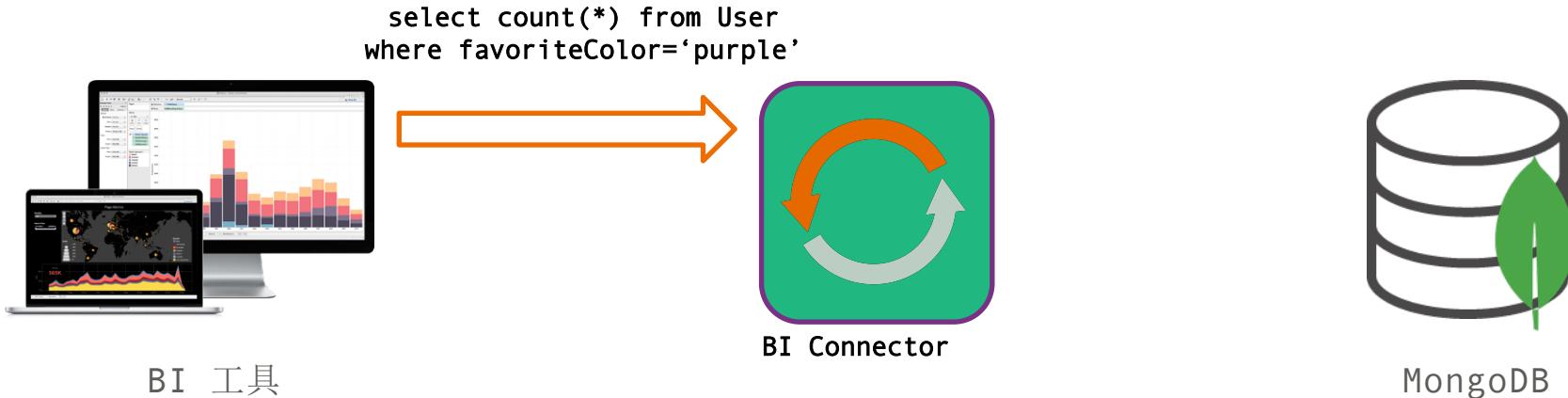
我不懂



MongoDB

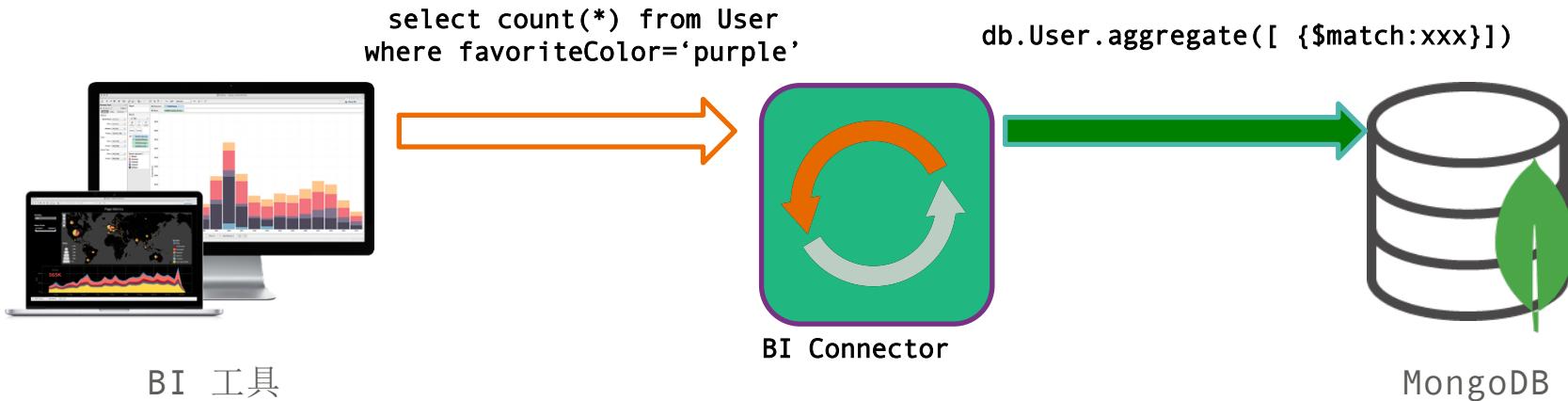
MongoDB 支持 SQL 方式查询吗？

- Yes, 通过一个转换套件



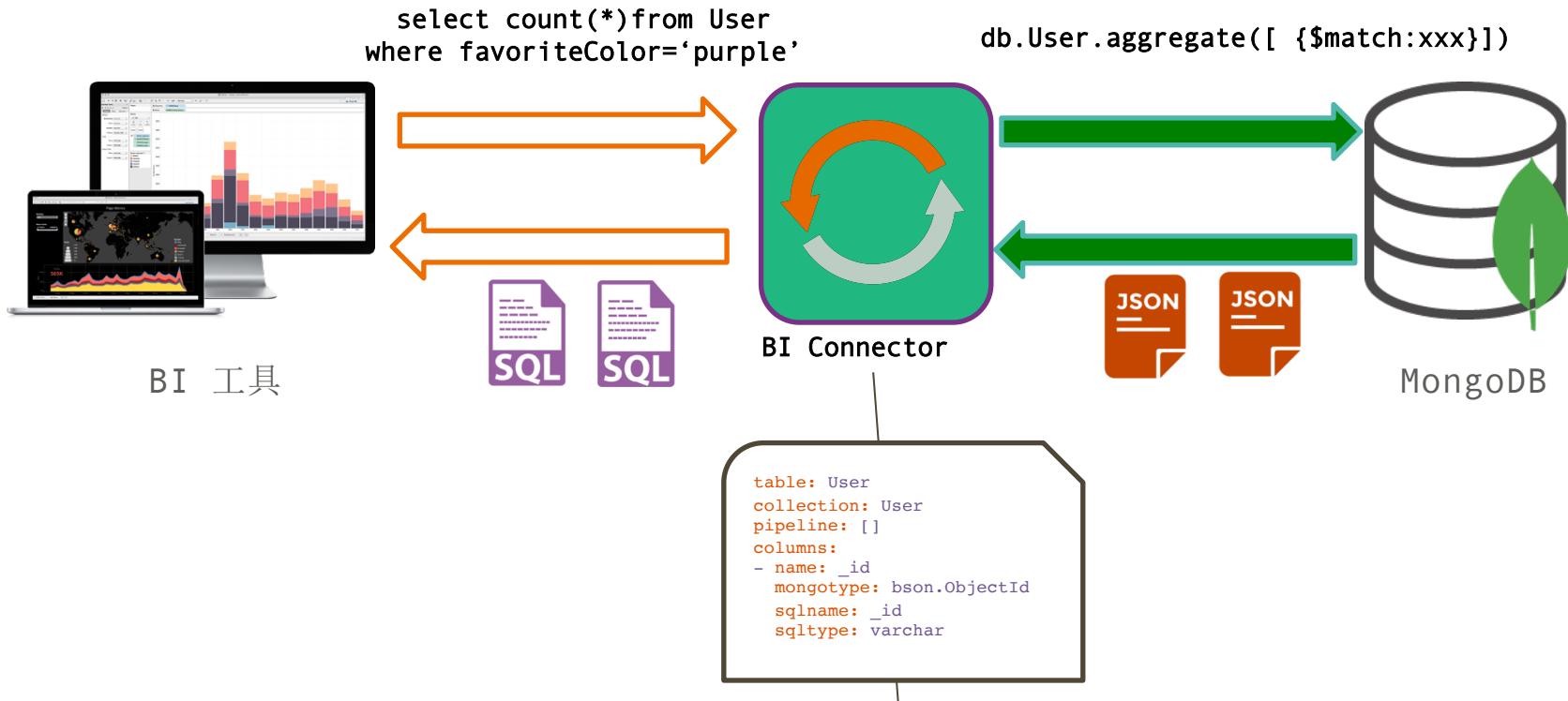
MongoDB 支持 SQL 方式查询吗？

- Yes, 通过一个转换套件



MongoDB 支持 SQL 方式查询吗？

- Yes, 通过一个转换套件



文档型到关系型映射定义
由工具自动生成
可以手工修改

映射文件 (DRDL)

```
table: User
collection: User
pipeline: []
columns:
- name: _id
  mongotype: bson.ObjectId
  sqlname: _id
  sqltype: varchar
- name: favoriteColor
  mongotype: string
  sqlname: favoriteColor
  sqltype: varchar
- name: address
  mongotype: string
  sqlname: address
  sqltype: varchar
```

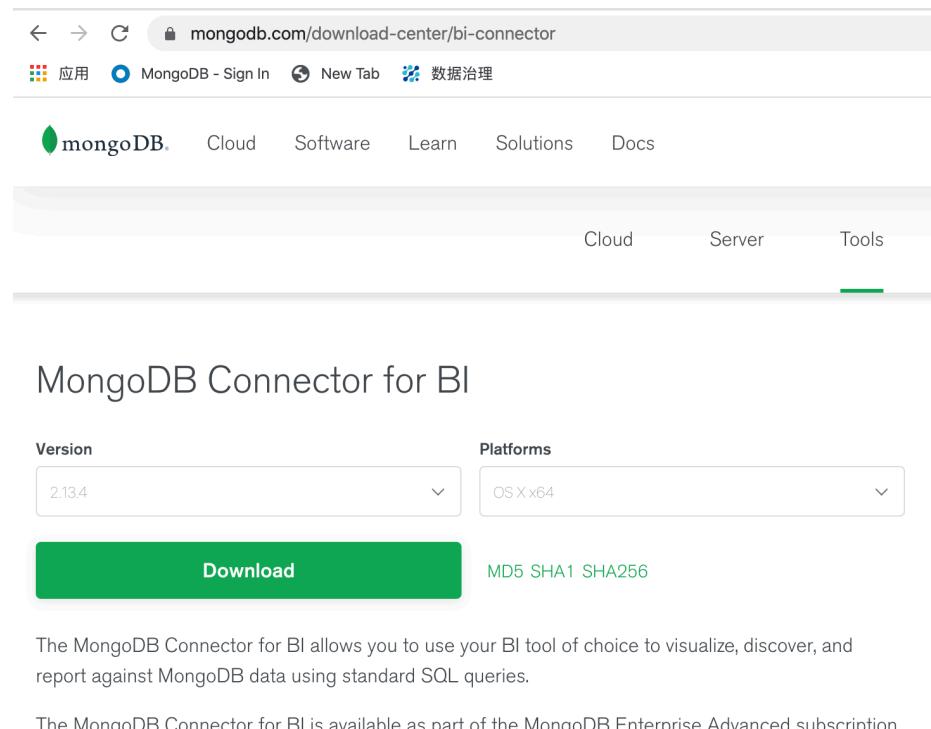
可修改表名

可隐藏字段

可修改字段名

关于 BI 套接件

- 独立于 MongoDB 运行的服务
- MongoDB 企业版组件
- (免费仅限开发环境)
- MySQL 协议兼容，可以使用 MySQL 客户端连接



The screenshot shows a web browser displaying the MongoDB download center at mongodb.com/download-center/bi-connector. The page header includes the MongoDB logo and navigation links for Cloud, Software, Learn, Solutions, and Docs. Below the header, there are dropdown menus for Version (set to 2.13.4) and Platforms (set to OS X x64). A large green "Download" button is prominently displayed. To the right of the button, there are download checksums: MD5, SHA1, and SHA256. Below the download section, a brief description states: "The MongoDB Connector for BI allows you to use your BI tool of choice to visualize, discover, and report against MongoDB data using standard SQL queries." Another paragraph notes: "The MongoDB Connector for BI is available as part of the MongoDB Enterprise Advanced subscription, which features the most comprehensive support for MongoDB and the best SLA".

DEMO

下载并安装MongoDB BI Connector

启动 mongosqld 进程

```
mongosqld --addr localhost:3307
```

```
--mongo-uri mongodb://localhost:27017
```

```
-v --logpath sql.log
```

使用mysql 客户端去连接 3307 端口

DEMO

```
select * from orders;
```

```
select first_name,product_id, quantity from orders o left join customers c on o.customer_id =  
c.customer_id;
```

```
select count(*), customer_id from orders group by customer_id;
```

MongoDB BI 套接件小结

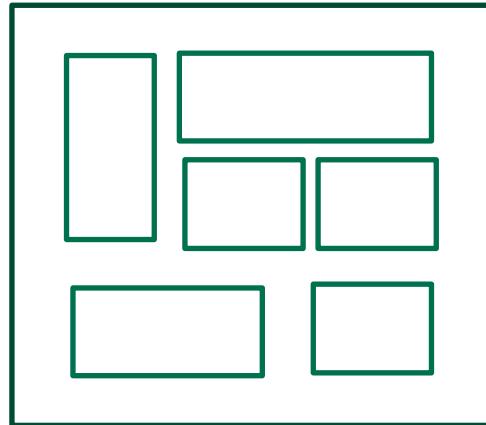
- 一个官方提供 SQL 解析器
- 只支持查询，不支持增删改
- 因为经过一个额外节点，性能和原生聚合框架相比有损耗
- 复杂的 SQL 查询，功能上支持，但是性能上建议要根据实际情况压测

4.10 MongoDB 与微服务

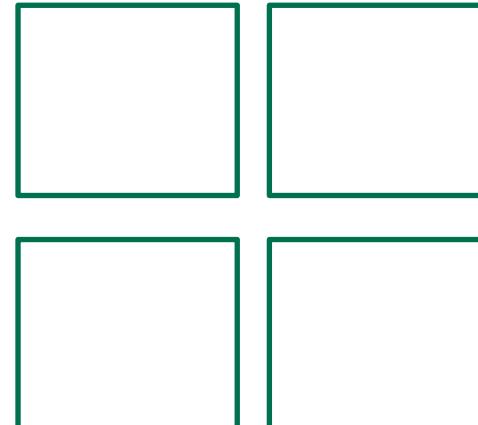
微服务 vs SOA vs 单体应用

The microservices architectural style is an approach to develop a single application as a suite of **small services**, each running in its **own process** and communicating with lightweight mechanisms, often **an HTTP resource API**. These services are built around **business capabilities** and **independently deployable** by **fully automated deployment machinery**...

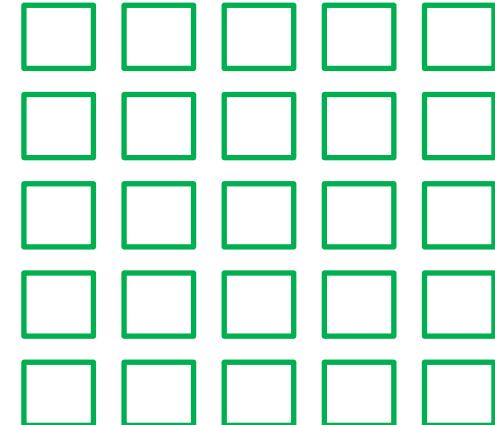
-- Martin Fowler



单体应用



SOA



Microservice

微服务的优势



开发速度快



变化响应快

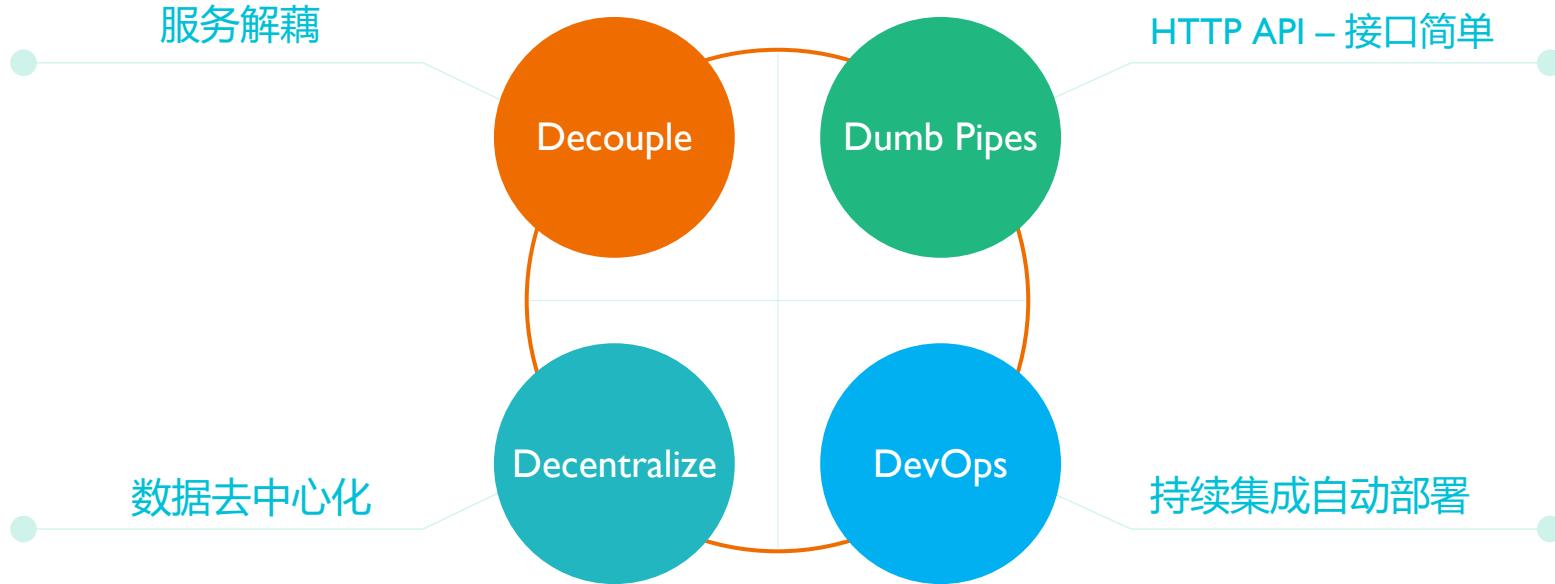


易维护



扩容简单

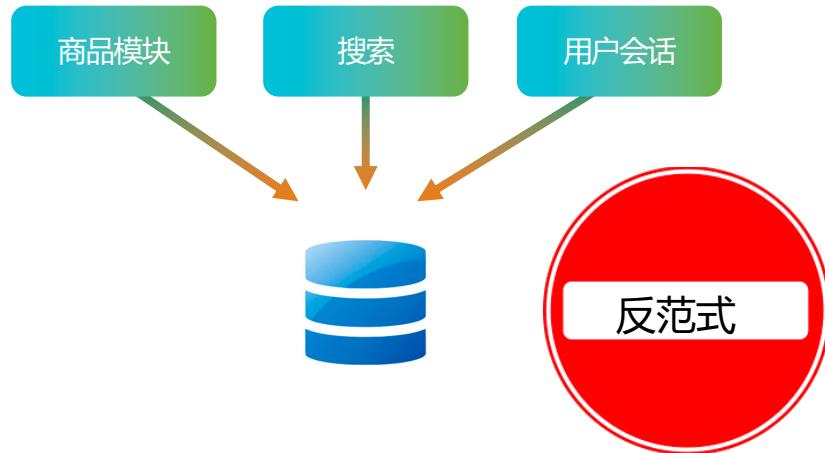
微服务架构设计要素



微服务的数据架构设计考量点

- 一服一库还是多服一库
- 混合持久化还是多模数据库
- 扩容便捷性

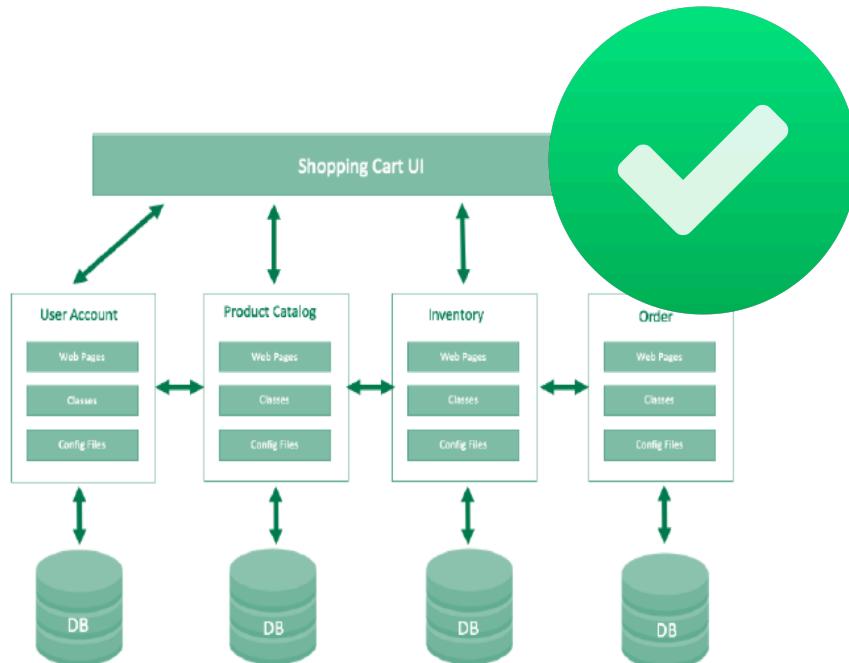
多个微服务共享一个数据库



一个(逻辑) 数据库为所有微服务使用

- 单点故障，一个性能问题可能拖垮整个服务集群
- 容易引起强关联，不利解耦
- 难以为某一个微服务单独扩容

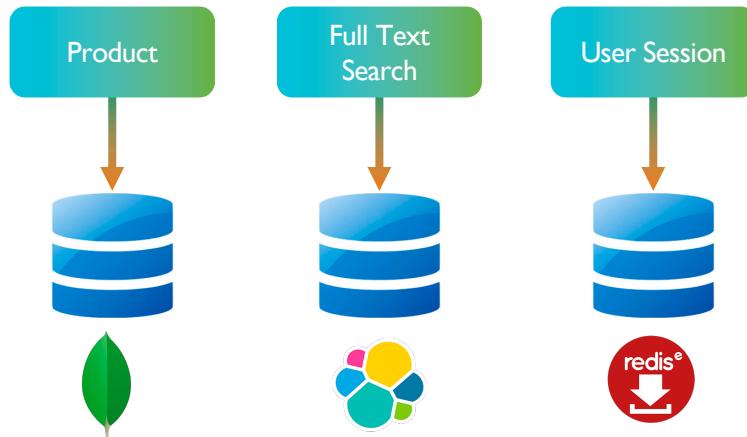
关键微服务使用自己专用的数据库



- 每个微服务使用一个逻辑库
- 数据库变动时候不影响其他服务

混合持久化

多态存储



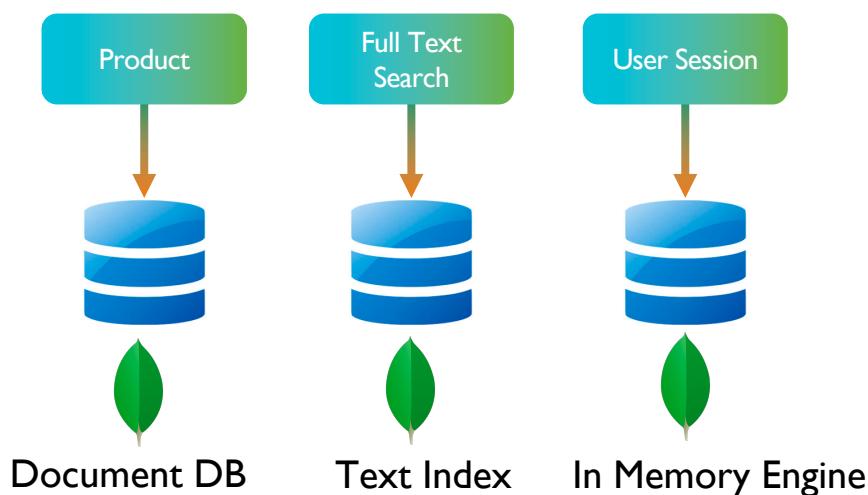
为关键的服务提供最合适的数据库



学习，管理和硬件成本

多模数据库

多模架构



一种数据库，多种模式



一种技术，学习及管理简单

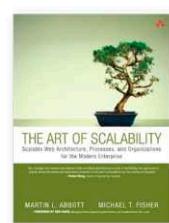
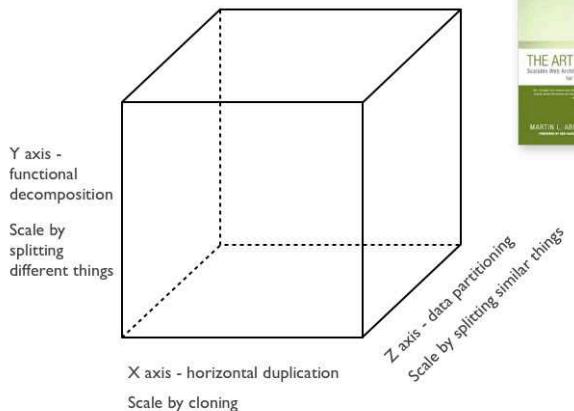


可以单独优化

是否可以快速扩容

The Scaling Cube

3 dimensions to scaling



- **X-axis** 水平扩展应用
- **Y-axis** 微服务化
- **Z-axis** 数据扩容

MongoDB 的扩容能力

Elastic scalability 弹性伸缩

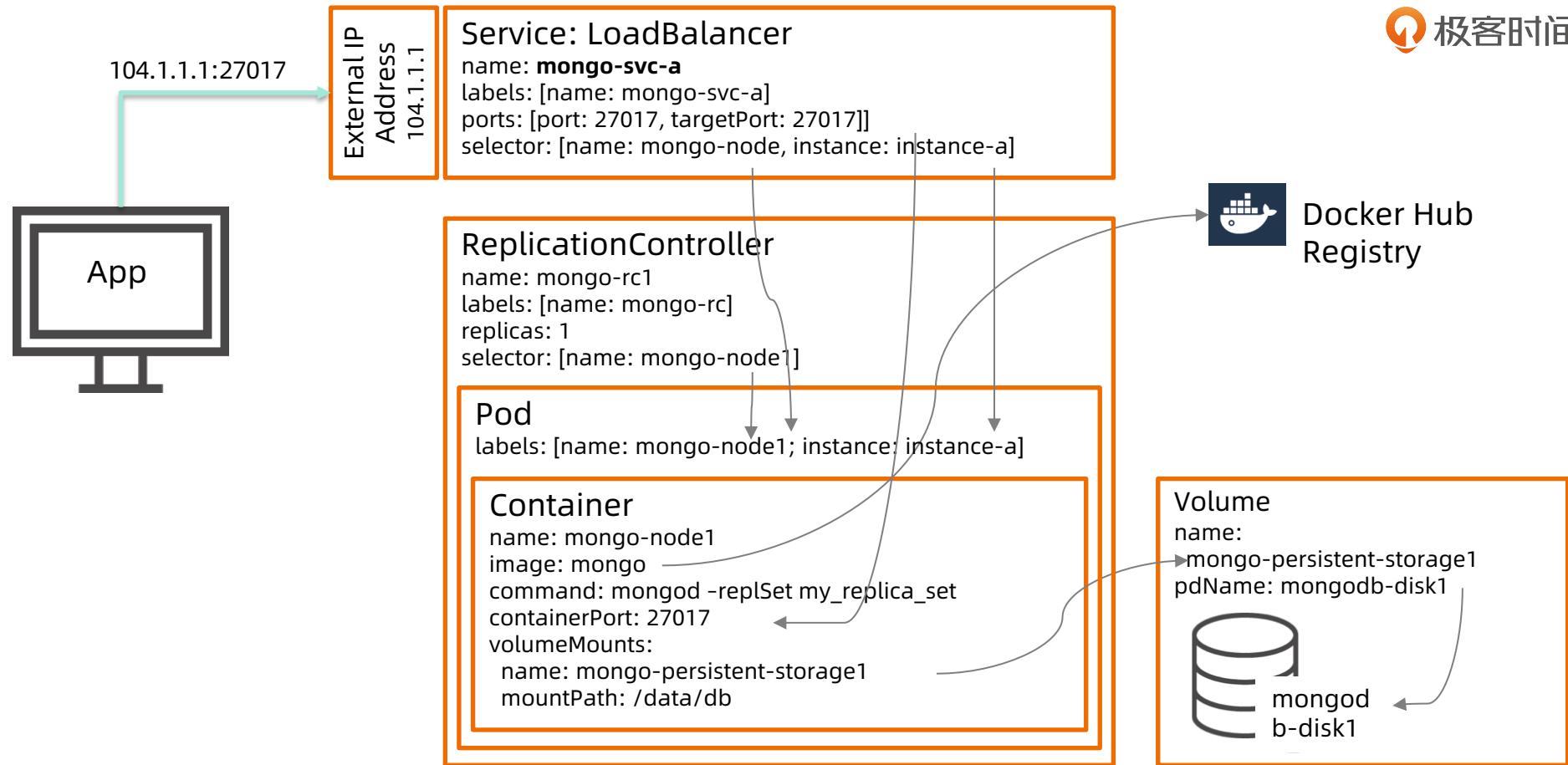
Auto Balancing 自动均衡

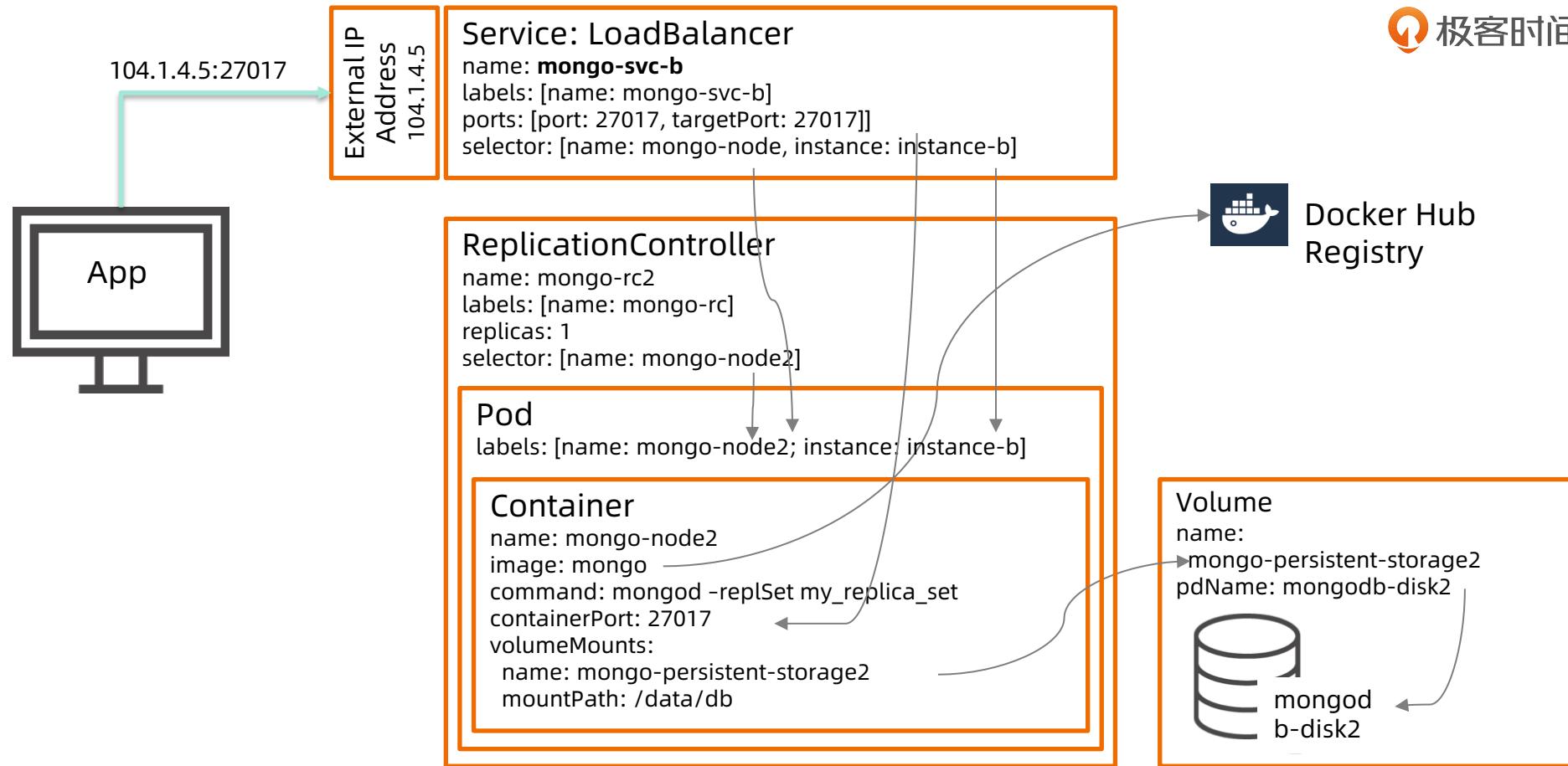
MongoDB 容器化部署



MongoDB 是一个有状态的服务，在容器化部署时候要特别注意

- 复制集节点要能够互相通讯：配置的时候要使用服务名，或者固定的服务 IP 地址
- 使用 Persistent Volume 或类似的长久存储
- 使用 Ops Manager 进行集群管理（而不是 K8S/OpenShift）





为何 MongoDB 适合微服务

- 易扩展，符合微服务的易扩展特性
- 模式灵活，迭代快，符合微服务架构的理念
- 多模数据库， 可以为不同的微服务提供业务

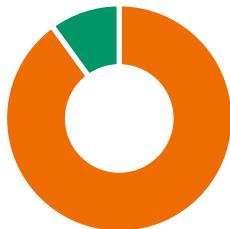
4.11 MongoDB 与数据中台

我们处于一个数据时代



数据即价值

DATA == VALUE



- 89%的CEO认为企业数据是无形资产
- 11%的CEO认为企业数据没有太多价值



关注并衡量企业数据的价值

认为数据是可估值企业资产

良好的目录管理&定义

数据没有太多价值

直接通过API经济将数据变现

Gartner 报告

现代企业的数据痛点

数据
孤岛

系统
重复

业务
开发
低效

某航空公司的乘客相关系统：门户，电商，常旅客，地勤，柜服，客服，营销等十来套。这些相互隔离的系统都和客户信息相关，但是相互隔离。

某半导体制造企业一套软件重复部署了52套，因为关系型数据库库不能支撑性能要求，所以每上一个新的业务就需要重复部署一套系统

某教育系统不断需要构建一些基于教师和学生的业务场景，但是每个业务都需要大量的前端开发，不仅成本难以控制，周期也难以控制。

数据孤岛：高成本，低效率



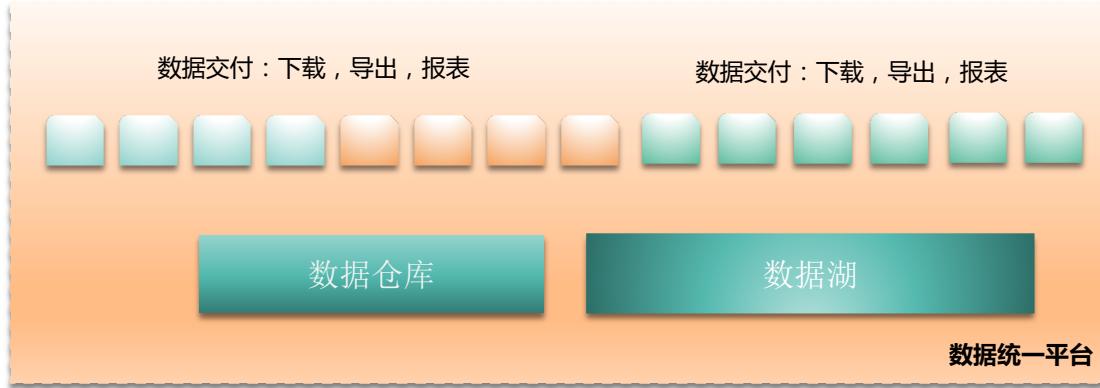
数据孤岛的形成根因

- 基于部门业务特性构建IT系统
- 80年代数据库技术无法扩展

影响：

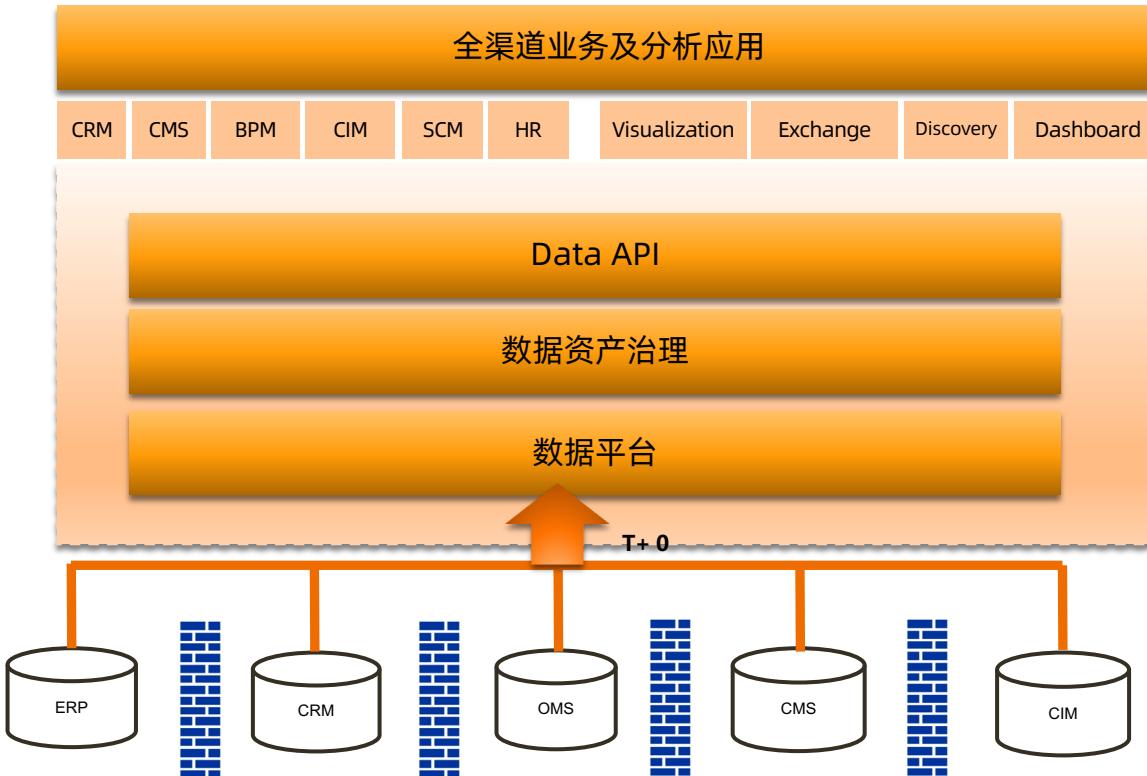
- 业务系统落地困难，需要花很多时间进行数据适配
- 数据不完整，影响客户体验
- 资源浪费严重，重复建设成本高

已有数仓及大数据方案之不足



- 并非为打通系统为主要目标，仍然以某个数据业务为核心
- 无数据资产管理，难以起到企业内有效共享
- 数据不及时，多为批量导入，价值受限
- 交付方式多为批量，难以满足API式交互需求

新型解决方案：面向业务的数据中台



- 打通孤岛系统为主要目标
- 结合数据资产管理，充分实现企业内数据有效共享
- 数据 T+0 进入平台，支撑分析型和实时型业务

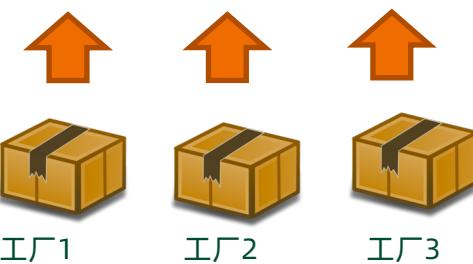
京东商城的例子



3. 半天内从下单到送货上门，实现最佳用户体验



2. 智能货仓自动分拣调配



1. 将优秀的商品提前运输到仓库

数据中台的一个定义

- 以打通部门或数据孤岛的**统一数据平台**为基础，构建统一
- **数据资产体系**，并以**API**服务方式为全渠道业务(**分析+应用**)提供即时交付能力的企业级数据架构

数据中台技术需求 (一)

模块	关键能力	备注
数据存储系统	横向扩展能力	中台需要具有能够收纳企业所有业务系统数据的能力
	灵活数据模型	中台数据模型多为整合多个源系统，并且需要不断支撑新型需求，需具有灵活建模的能力
	高并发低延迟响应能力	中台支持交互式应用，并有可能直接穿透到客户，需提供毫秒级数据访问能力及高连接数能力
	同城高可用及异地备份	中台支持的为前端业务系统，必须具有 24x7, 99.9% 的高可用能力，以及异地热备的能力
	数据安全	存储加密，传输加密，字段加密，LDAP 认证，鉴权，稽核
数据同步汇聚工具	批量同步及导入能力	能够把已有业务数据一次性或定期方式导入到中台
	数据库实时同步能力	以 CDC 方式，在 3~5 秒延迟内将数据从源生产数据库同步中台存储系统，保证最佳用户体验
	数据库及其他数据源支持	DB2, Oracle, PG, SQLServer, DW, Hadoop, CSV, Legacy 及 API 接口等等
	断点续断传机制	系统中断后可以从中断继续，不会丢失数据更新
	异构数据模型整合能力	支持不同源系统不同结构数据模型在同步过程中同时进行模型转换，如转换 JSON 格式
数据治理及开发	数据目录及元数据管理	需提供一个可自定义数据目录的管理能力，有效组织中台内众多的数据类型。支持修改描述，搜寻等功能
	数据建模	支持在中台内进行按照业务需求动态建模，包括新建模型，多表合并或关联合并
	数据开发	支持在中台内进行数据的一些处理及计算，如转换栏位类型，栏位增强，数据合并等
	数据质量管理	支持定义数据规则并对违规数据进行统计，检查及修订等
	数据匹配去重	中台需提供唯一数据 ID 能力。来自不同源系统的同一个数据实体（如客户）需能够进行匹配及去重

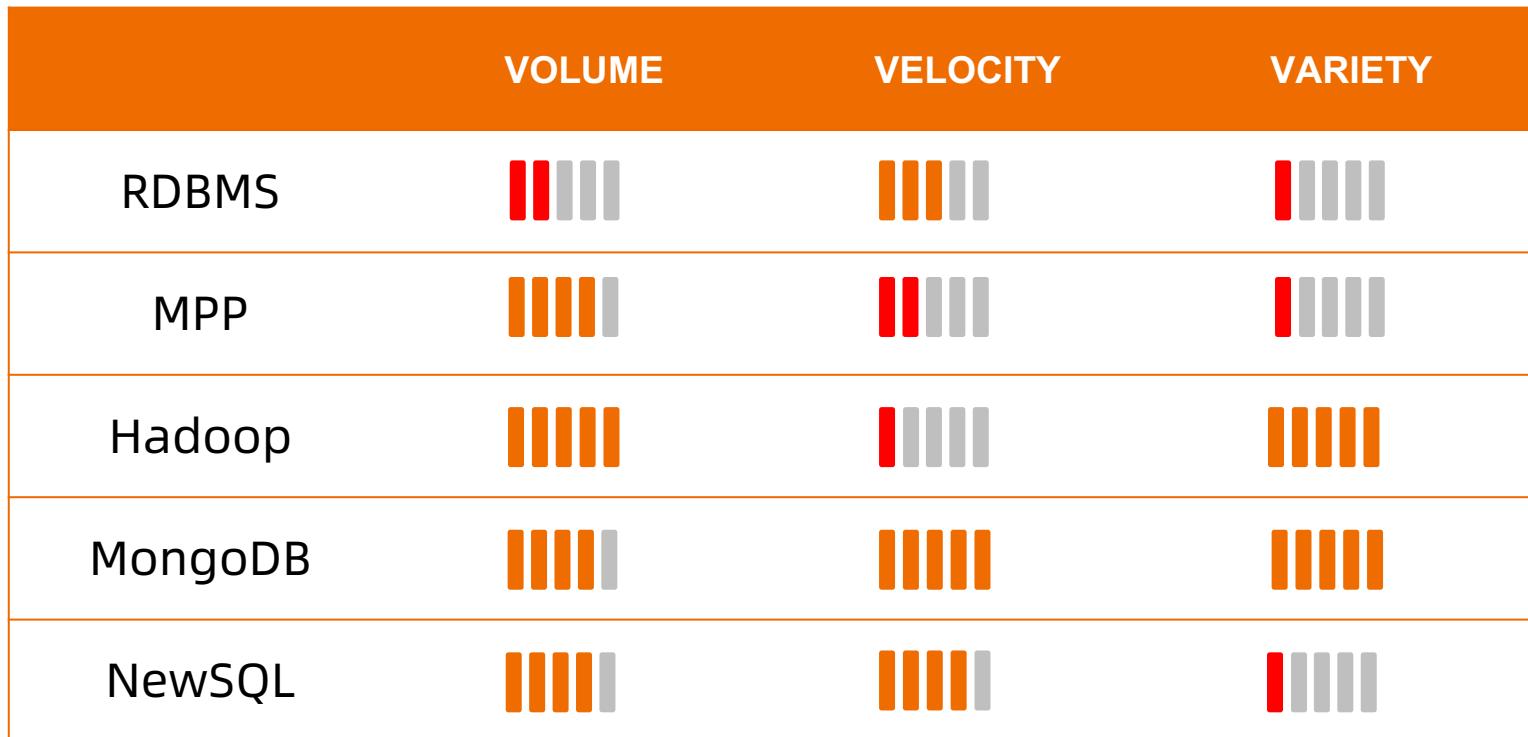
数据中台技术需求 (二)

模块	关键能力	备注
数据交换及发布	无代码 REST API 快速发布能力	中台的数据模型需要能够即时的以 API 方式发布出去
	REST API 订制能力	可按照需求进行级及列级的过滤
	API 文档及测试	提供工具让用户了解 API 的使用方式并进行测试
	SQL 计算接口	允许让 BI 及报表用户以 SQL 方式来查询数据
	横向扩展及高可用	能够随着使用量的增加和进行能力扩展
	大数据计算接口	提供 Hadoop/Spark 数据计算框架的对接能力, 能够直接与其对接提供数据进行数据运算并收集计算结果
	流计算接口	提供 Kafka 或类似的流处理计算框架的对接能力, 能够向 kafka 以 producer 方式提供数据或者以 consumer 方式消费数据
系统管理能力	可视化任务设计	通过 UI 进行数据开发任务的设计及调整
	任务调度及监控	提供任务调度及任务运行状况实时监控, 了解数据同步或者处理进度
	日志管理	系统运行日志监控及搜寻
	告警机制	异常事件如任务中断即时报警
	用户权限管理	创建、修改中台管理用户, 角色及权限配置等
	数据备份及恢复	数据的即时备份及指定时间点恢复能力
	集群管理及监控	中台系统集群的部署管理, 运行状况监控等。

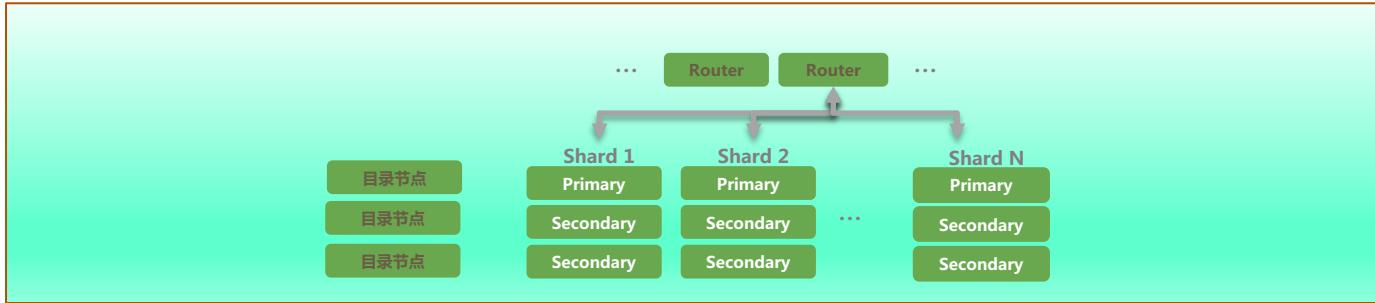
数据中台技术模块

数据平台	ETL/CDC	数据治理	数据服务
Hadoop / HDFS / Hive	Kafka	Apache Atlas	Spring
Teradata	Kettle	Informatica	Kong
MongoDB	Flink	Erwin	Kafka
Greenplum	Spark ETL	Oracle	Loopback
MySQL Cluster	Talend	WhereHow	Mulesoft
Oracle	Informatica		CA
Transwarp	Golden Gate		APIGee
Elastic Search	Tapdata		Tapdata

数据平台选型 - Gartner 3V 标准



MongoDB 作为中台内数据平台方案的技术优势



横向扩展能力

- TB – PB 数据量支持
- 无需下线自动扩展

支持多种数据模型

- 结构化和非结构化
- 模型变动灵活

高性能高并发

- 毫秒级响应
- 高并发促销场景

多云跨中心部署

- 支持各种云
- 多中心混合云部署

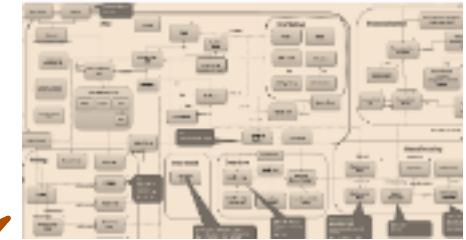
4.12 MongoDB 数据中台案例

客户业务场景

- 近百年珠宝老店，业务遍布大陆、港、台、澳地区。
- 数百家门店，百亿级年营收。
- 十多套线上及线下销售，会员系统，库存盘点，仓库管理等系统。
- 技术栈基于 Oracle。

旧数据模式的挑战

- 大陆业务发展迅速，市场部门需求层出不穷，IT部门不断扩大团队，引入更有效率的技术方案，仍无法跟上速度。
- 传统的数据库设计晦涩难懂，仅有极少数核心技术人员有能力掌握，严重影响新业务开发上线
- 公司策略是向海外包括美国/东南亚扩张，需要能够快速搭建技术系统以配合业务发展

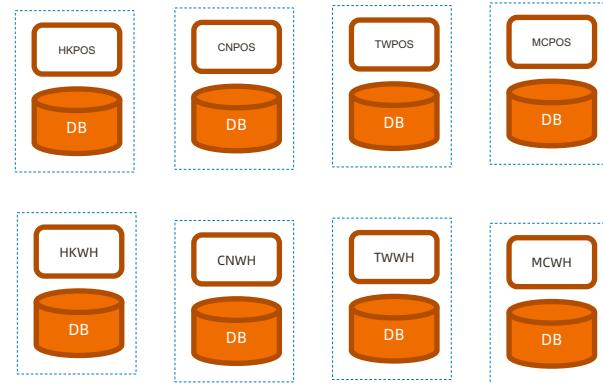


入职6个月，才算对数据库的结构设计搞得比较清楚，开始能做一些调整。

- 某技术经理

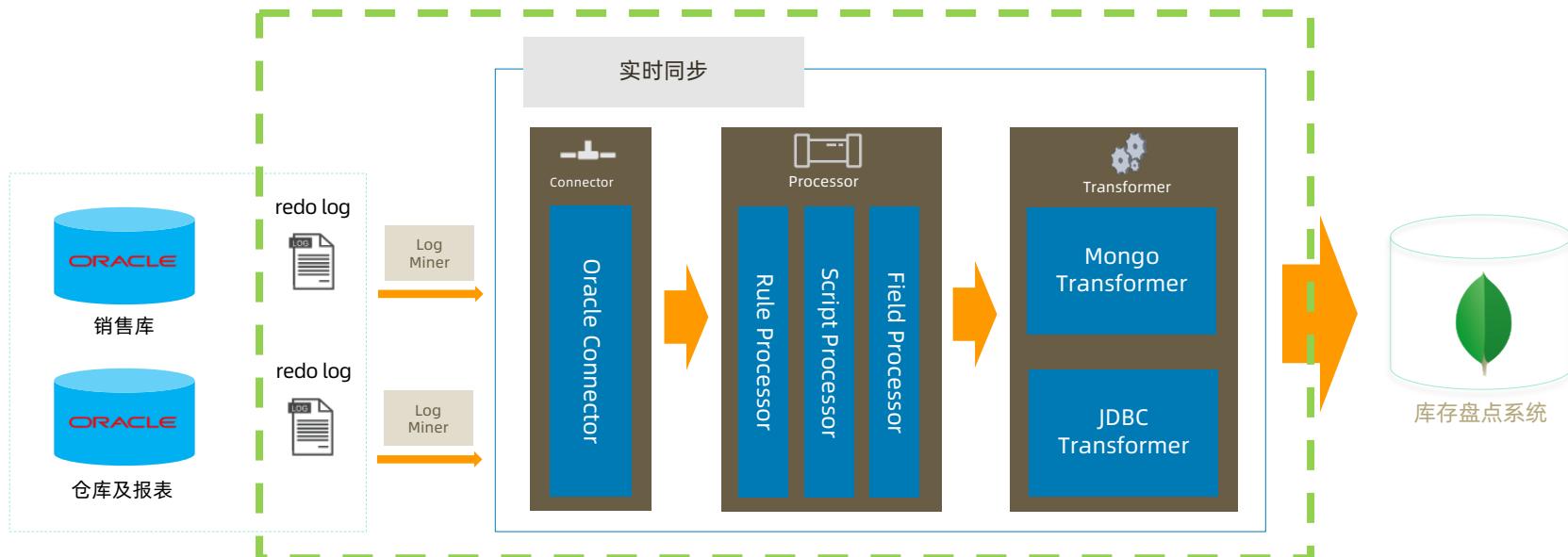
数据孤岛及重复的系统

- 每个地区有自己的销售和库存系统，产品和销售数据在很多套 Oracle 系统。数据通过队列系统交互，易错难追踪
- 难以快速提供一个准确的库存及销售，影响业务部门决策
- 构建一些数据 API 为前端业务提供支持的时候，需要进行繁琐的数据对接和整合，然后再开发

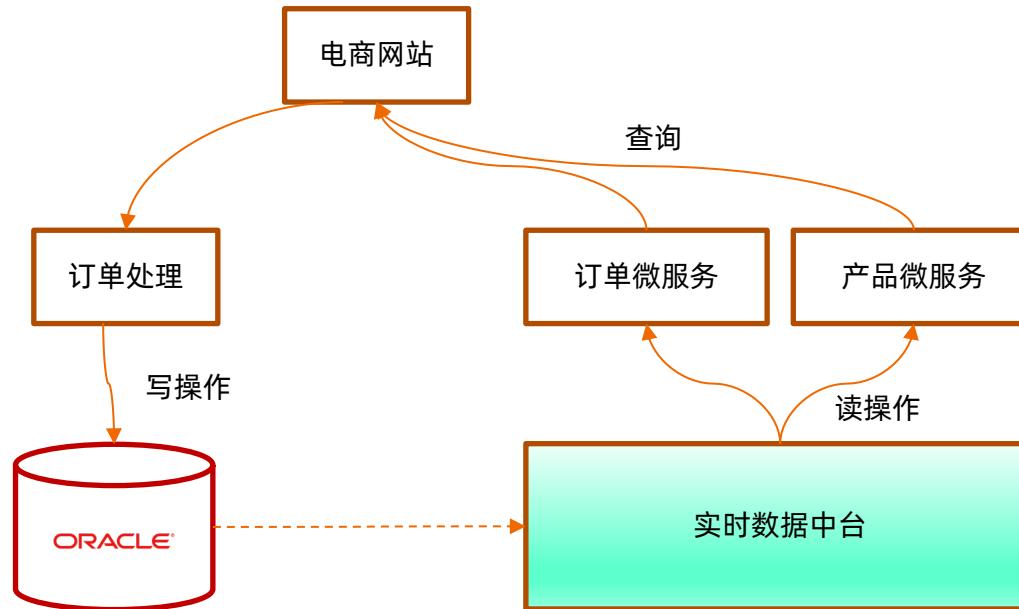


- 维护上的重复
- 数据复制的成本
- 数据使用的体验

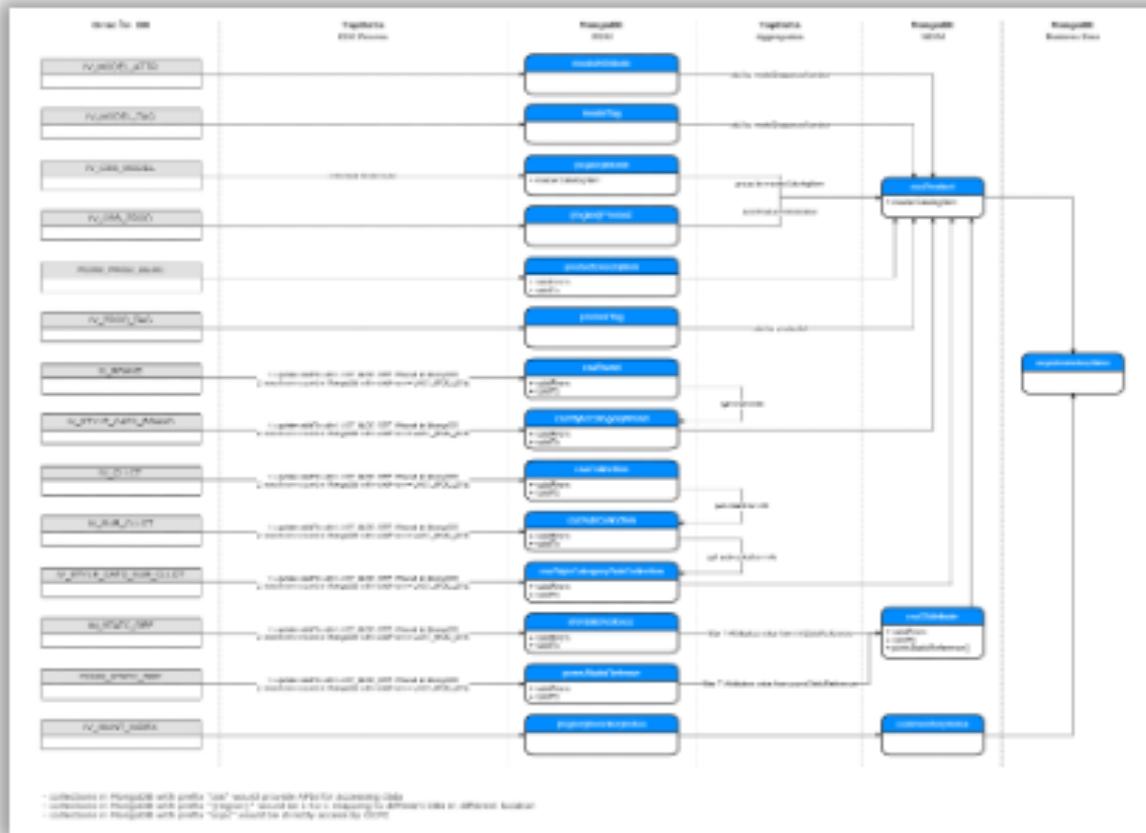
MongoDB: 开始只是一两个孤立的应用



用 MongoDB 来搭建中台第一步：读写分离模式支撑查询业务



传统关系模型 vs 中台业务模型：



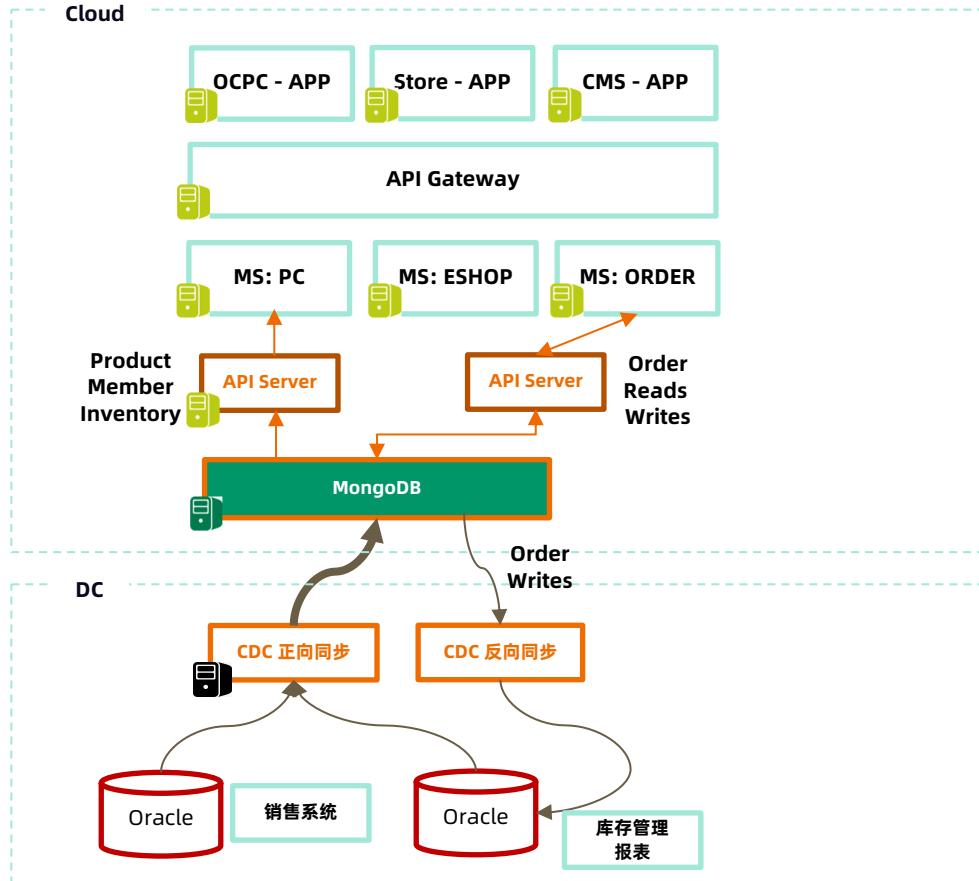
- Oracle Schema:

40+ 张表

- MongoDB中台:

4个业务模型

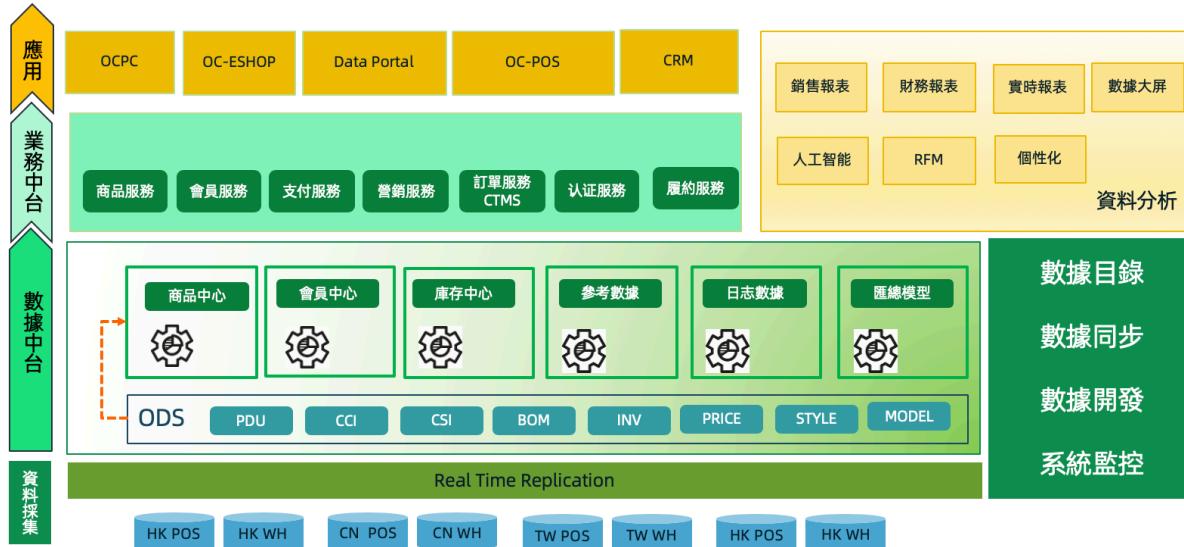
MongoDB 中台建设第二步：支撑写入



说明

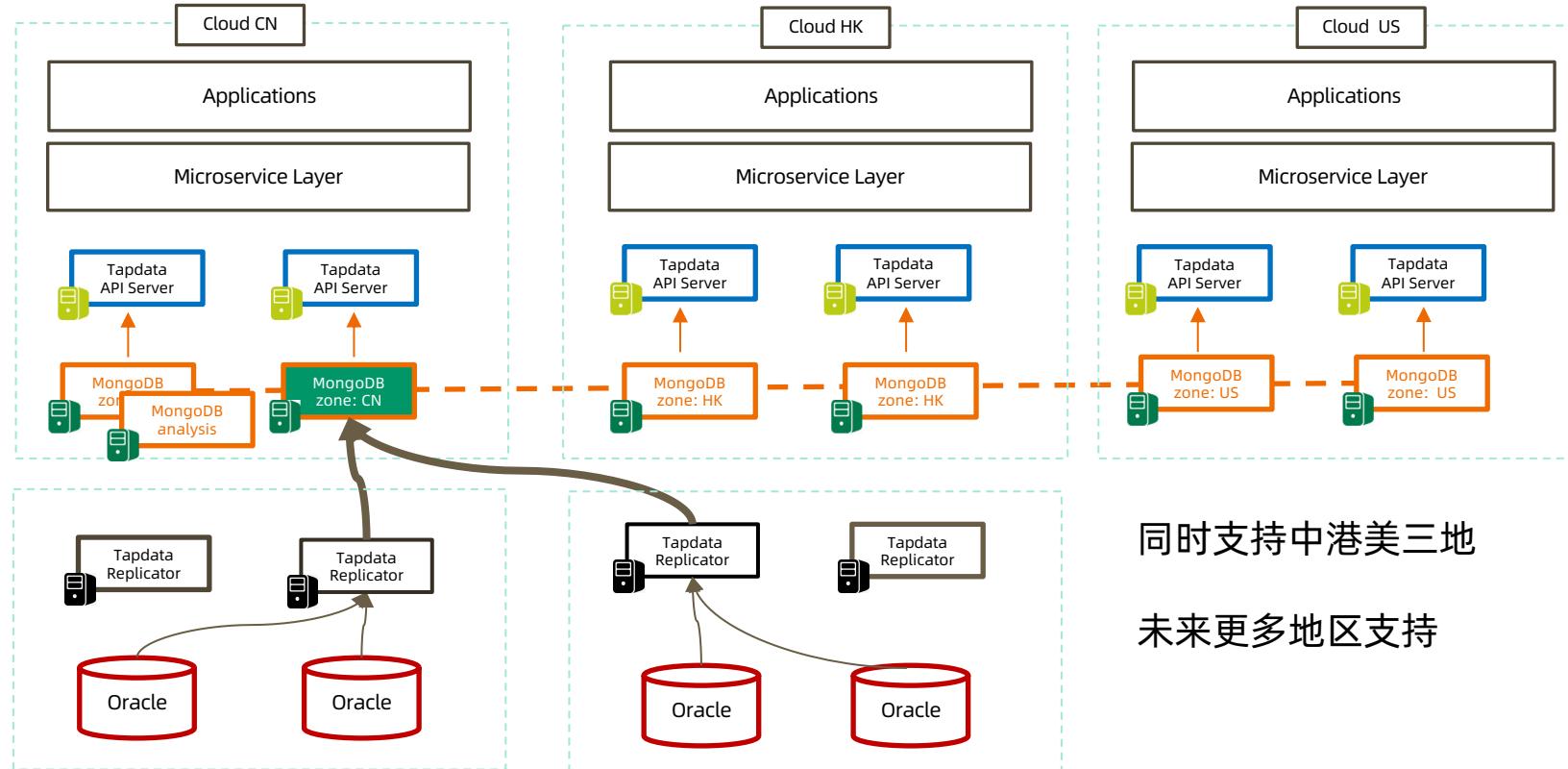
- 使用实时同步技术，将 Oracle 内所有主数据（产品，会员等）和交易（订单，库存，日志）数据复制到 MongoDB。
- 利用的 MongoDB 的灵活模型快速构建新业务需要的业务模型。
- 通过中台提供的 REST API 为微服务提供商商品查询，检索及订单处理能力
- 订单微服务接受订单，直接调用中台 API 修改库存。
- 中台将库存修改信息通过反向 CDC 发送到需要的原业务库，比如报表系统或者订单通知系统

第三阶段：作为 System of Records 的主数据平台



- 连接企业数据孤岛
- 构建统一数据资产体系：
商品，会员，订单等
- 企业主数据的更新统一在
中台发生，保证中台数据
的权威性，一致性和可靠
性

技术架构图



搭建企业数据中台的价值

能力沉淀与复用

知识与能力沉淀到中台
数据重用能力显著增强

快速开发

新业务不再受老系统牵制
快速开发快速迭代

更新的技术栈

高可用，易扩容
从单体式到分布式
多中心部署

从 IT 价值

到企业价值



更好的支持市场和营销
增加营收



降低运营成本
提升运营利润



改善用户体验



扫码试看/订阅

《MongoDB 高手课》视频课程