

# 电信核心系统 应用MySQL数据库实践分享

吴宇星

福富软件

2016年7月

# About me

- 吴宇星
- 从前台到后台，从PHP到JAVA，从开发到架构，都折腾过、并在折腾着
- 2005年加入福富软件，一直从事电信CRM的研发设计
- 专注技术，找轮子、想轮子、造轮子

# 目录

- 系统背景分析
- 关键技术方案

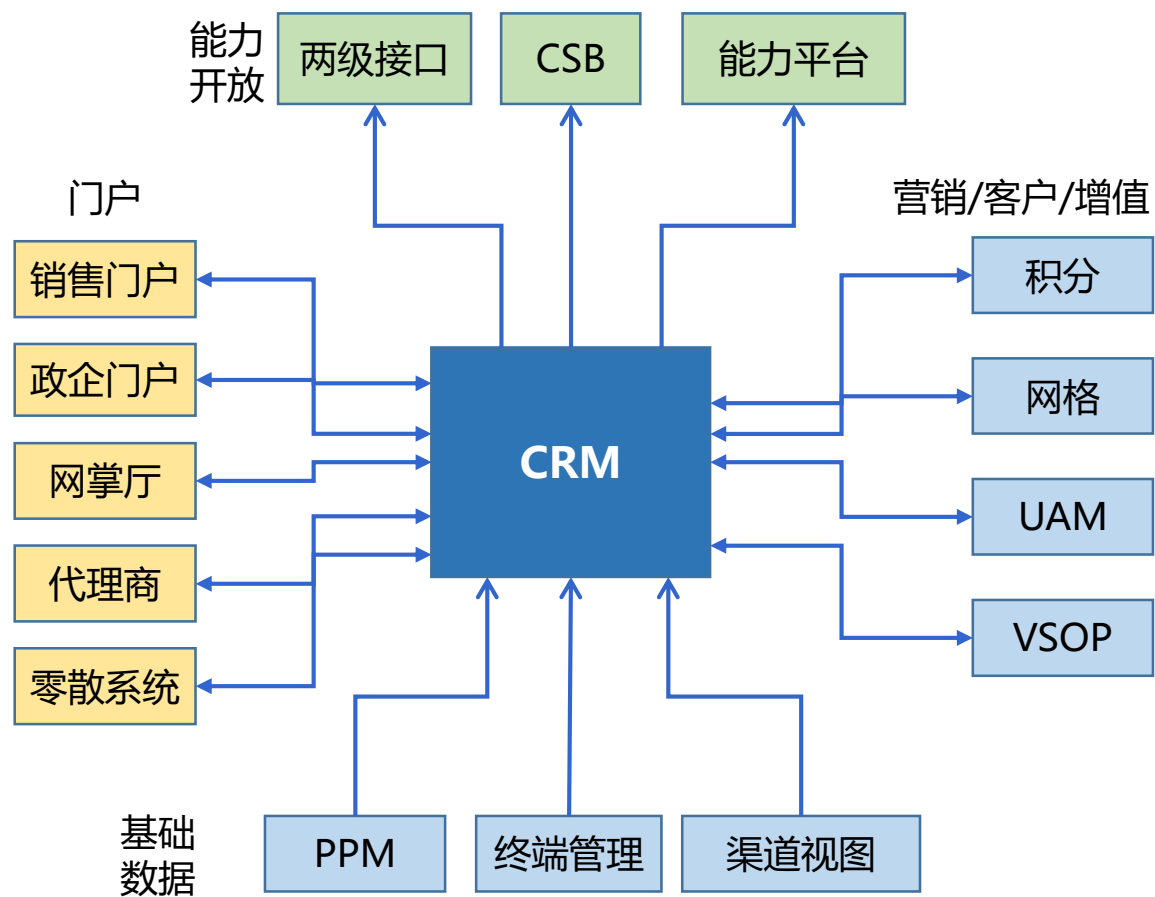
# 系统背景分析

应用系统介绍

传统架构介绍

我们的演进路方案

# 应用系统介绍



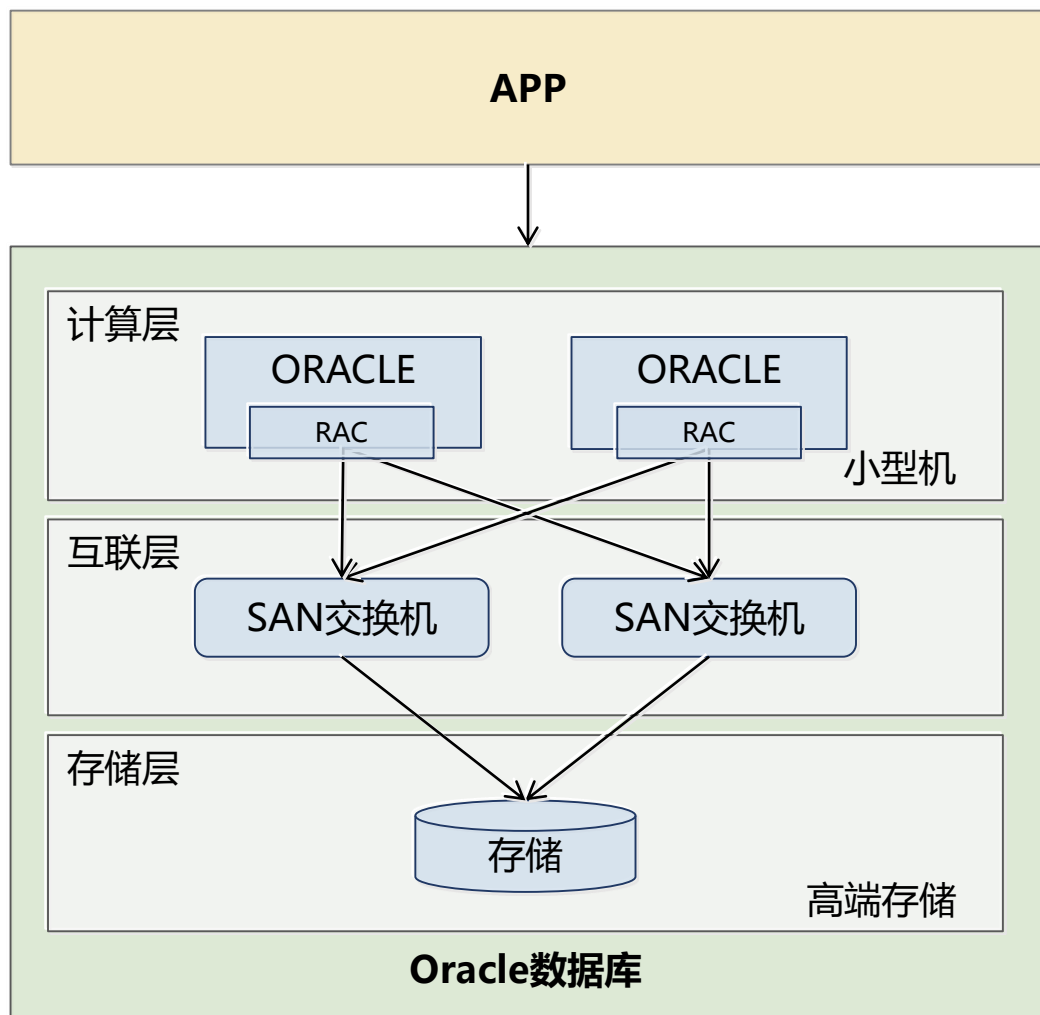
## □ 客户关系管理系统：

- 生产运营 **支撑系统**
- 包含：营销、订单受理、客户服务、投诉处理

## □ 系统特征：

- 高并发、大数据量、以 **联机事务** 处理为主的交易型业务系统
- **业务复杂、模型复杂**
- **外围渠道接口繁多**

# 传统架构介绍



## □ 架构特征：

- 典型的 **“IOE”** 架构，小型机+Oracle+EMC
- 系统**耦合高**
- 生产**运营要求高**，数据不允许丢

## □ 面临的问题：

- 数据库**压力大、系统运行性能不足**
- 数据库**横向扩展困难**，小型机CPU、内存无法扩容
- **无法快速支撑**业务需求

# 我们的演进方案

分阶段方式，循序渐近进行实施改造

技术验证

**数据双写和提供查询，验证分布式数据库技术的安全性和可靠性**

范围：订单、客户、账户、用户核心数据双写

模块试用

**改造部分模块读写走分布式数据库**

范围：对营销资源、客户投诉、集中受理台整体基于分布式数据库改造

全面推广

**CRM全系统整体改造**

范围：对CRM进行整体改造，读写都走分布式数据库

# 关键技术方案

关键技术问题和设计考虑因素

数据架构关键点方案

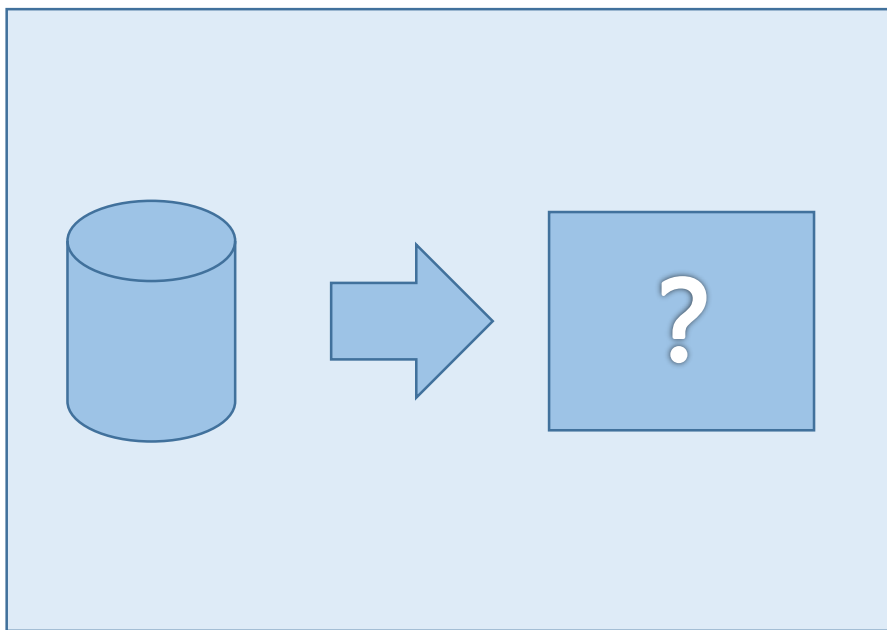
应用改造关键点方案

数据集成关键点方案

分布式数据运维工具介绍



# 关键技术挑战：难点及目标思路（1/2）



√ 应用系统、基础设施云化

× 数据存储分布式

## □ 数据量

- 总数据量：10T级别
- 总记录数量：200亿条级别
- 单表最大数量：15亿
- 单表超1亿数量：超30张

## □ 目标要求

- 数据量
- 性能
- 水平扩展

借鉴互联网应用的经验，采用分布式数据架构

# 关键技术挑战：主要考虑因素（1/2）

## 业务支撑

- 如果解决**数据统一共享**，无地域差异、简单快捷支撑全网业务的问题

## 版本研发

- 如何解决**开发效率**的问题。
- 如何解决架构演进可持续，且**代价相对较低**（如替换底层数据库，应用组件无需修改）。
- 如何解决跨库事务一致性问题
- 如何支撑高并发下的高效的数据访问

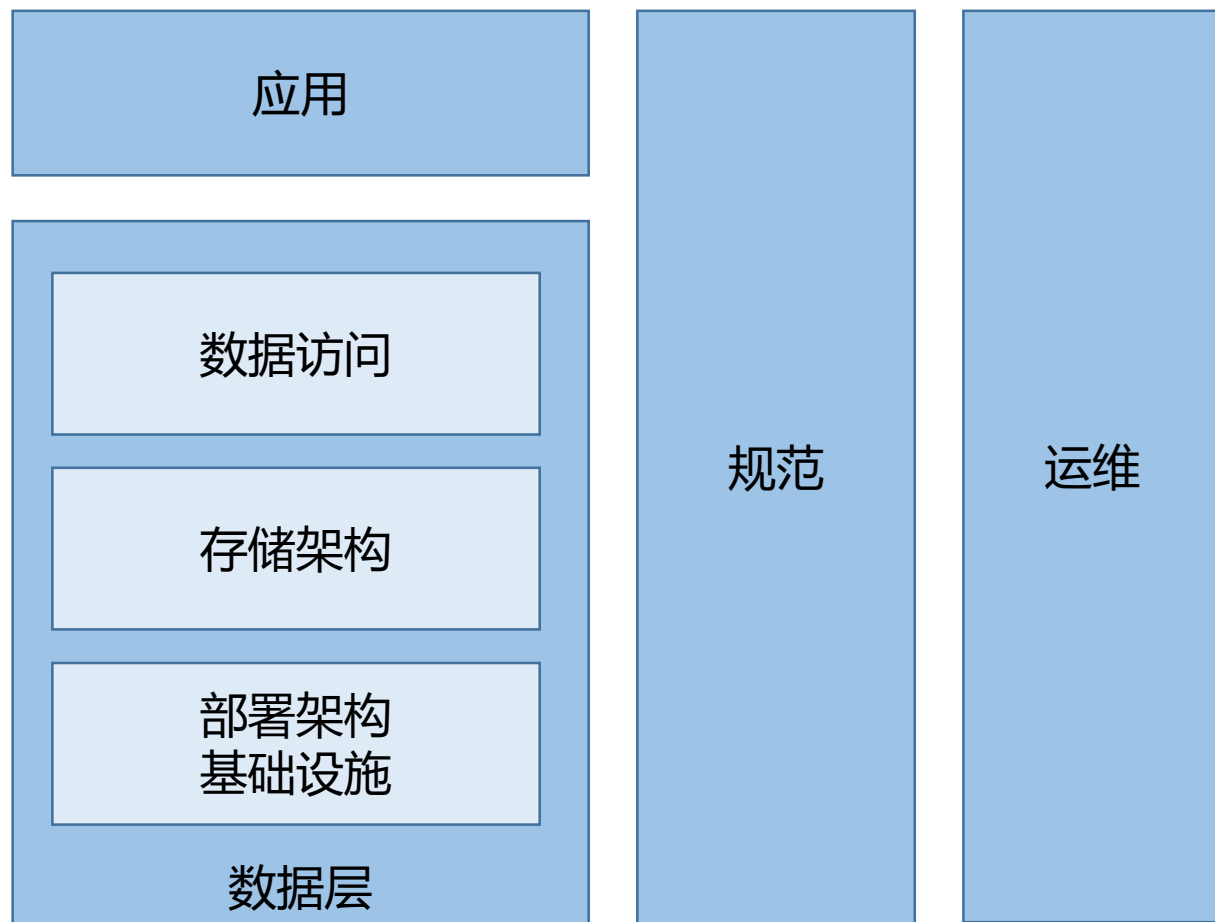
## 工程割接

- 如何解决**数据割接**高效问题

## 运营维护

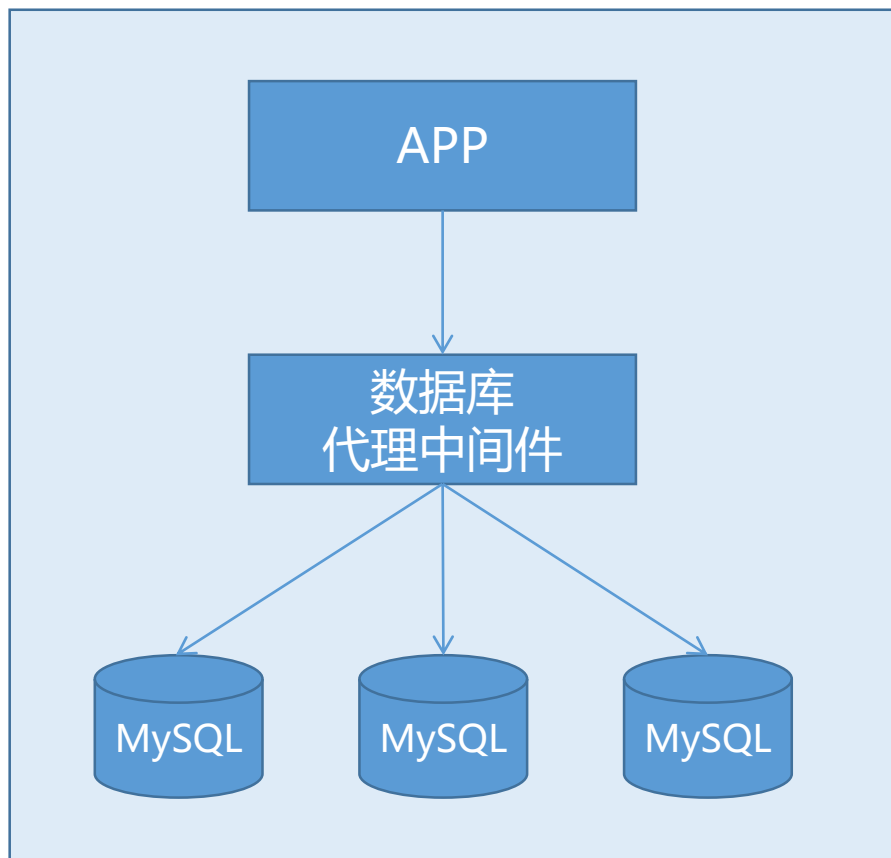
- 如何解决**在线扩容**问题
- 如何达到数据库的**高可用性**
- 如何做到**故障隔离**
- 如何解决故障问题处理过程中的**高效修复数据**

# 总体方案思路



- 综合考虑业务支撑效果、开发效率、工程割接、运营维护
- **不追求方案的高通用性**，针对关键业务场景进行分析制定解决
  - 考虑实现难度和代价，部分关键技术不追求适用范围的通用性，以解决电信行业内业务场景为出发点考虑。即适用性原则，不追求技术层面的“完美”
  - 通过关键业务场景的推演，来确认关键技术设计方案的可行性程度

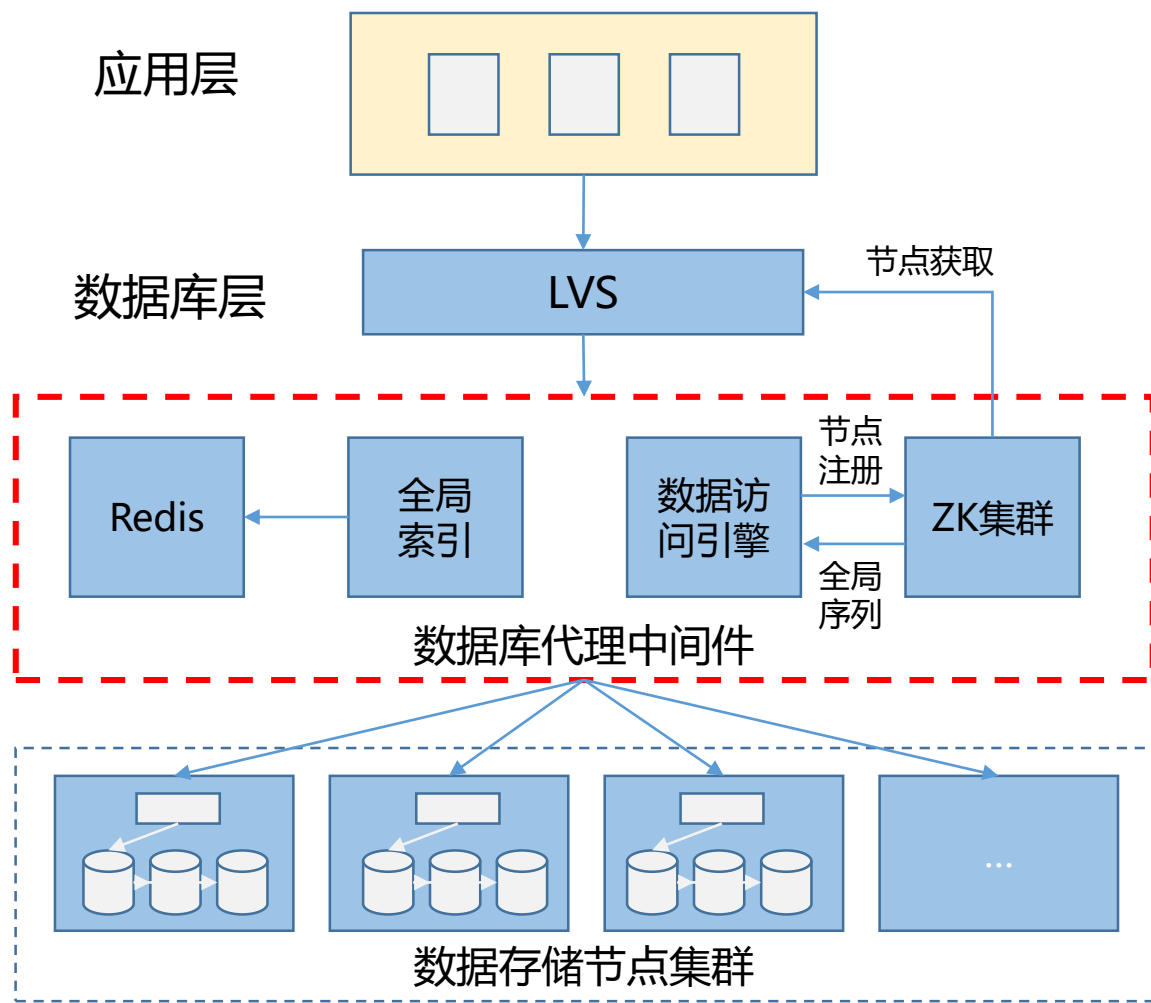
# 数据架构关键点：数据代理层-总体介绍（1/4）



基于MySQL开源协议，采用数据库代理方式，形成分布式数据库中间件解决方案，主要包含：

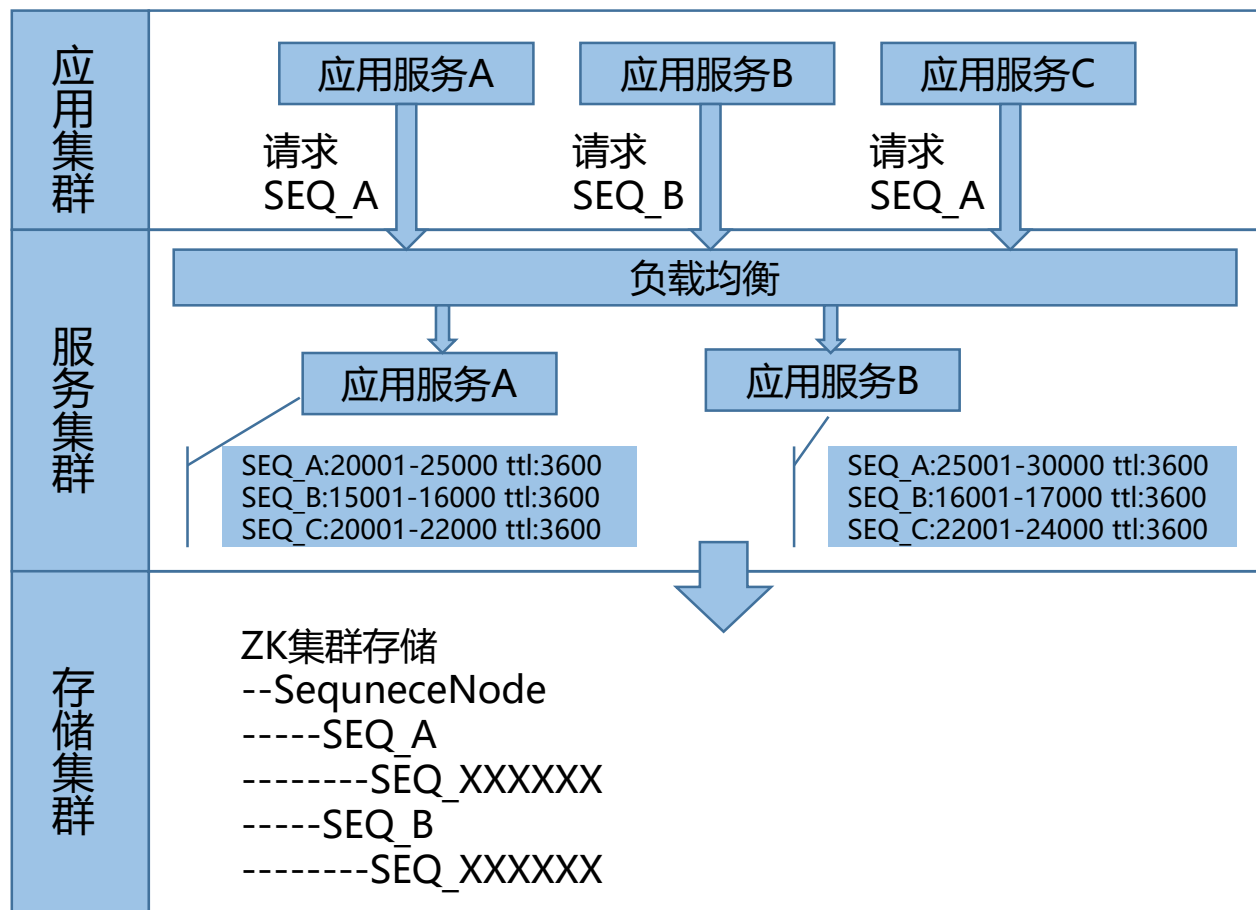
- ❑ Sever：统一接收应用层的数据请求，实现“SQL解析”、“执行引擎”、“数据路由”、“数据汇聚”和“连接管理”等功能。
  - **数据访问层**：主要负责关系型数据分片、路由、读写；
  - **全局索引**：跨分片数据按照关键字索引，支撑业务数据快速定位；
  - **全局序列**：提供全局序列号获取功能。
- ❑ Client API：
  - MySQL协议，**通过Hint实现部分特殊控制功能**，如：路由查看、指定数据节点等能力。

# 数据架构关键点：数据代理层-调用及关键点（2/4）



- ❑ 应用不直接访问数据库，**通过LVS访问**数据库代理中间件（Proxy），LVS提供VIP接入，支持主备切换
- ❑ Proxy通过ZK注册活动节点，**LVS通过ZK发现Proxy活动节点**进行负载分发，集群中的节点是对等的，只要有一个节点能提供服务，整个集群即可用
- ❑ 全局索引数据存储在分布式缓存（redis）中
- ❑ 全局序列数据存储在ZK中

# 数据架构关键点：数据代理层-全局序列方案（3/4）



全局序列 ( ID )

数据库被切分到多个物理结点上，无法依赖数据库自身的主键生成机制，需要实现全局序列生成功能。

## □ 主要原则：

- 1、提供统一的API用于获取全局序列号
- 2、全局序列能够保持**单调稳定增长，不能跳跃**
- 3、全局序列**不能与区域关联**

## □ 实现方案：

采用两层服务提供sequence能力，底层采用**ZK保存**sequence持久化值；上层采用多台服务器组成集群，并通过**预读sequence**段的方式保存在内存中，通过集群负载均衡方式对外提供服务。

# 数据架构关键点：数据代理层-路由、索引、汇聚（4/4）

## □ 数据路由

- 传入分片ID情况下，通过**分片ID计算出实际存储节点**，直接执行并返回结果

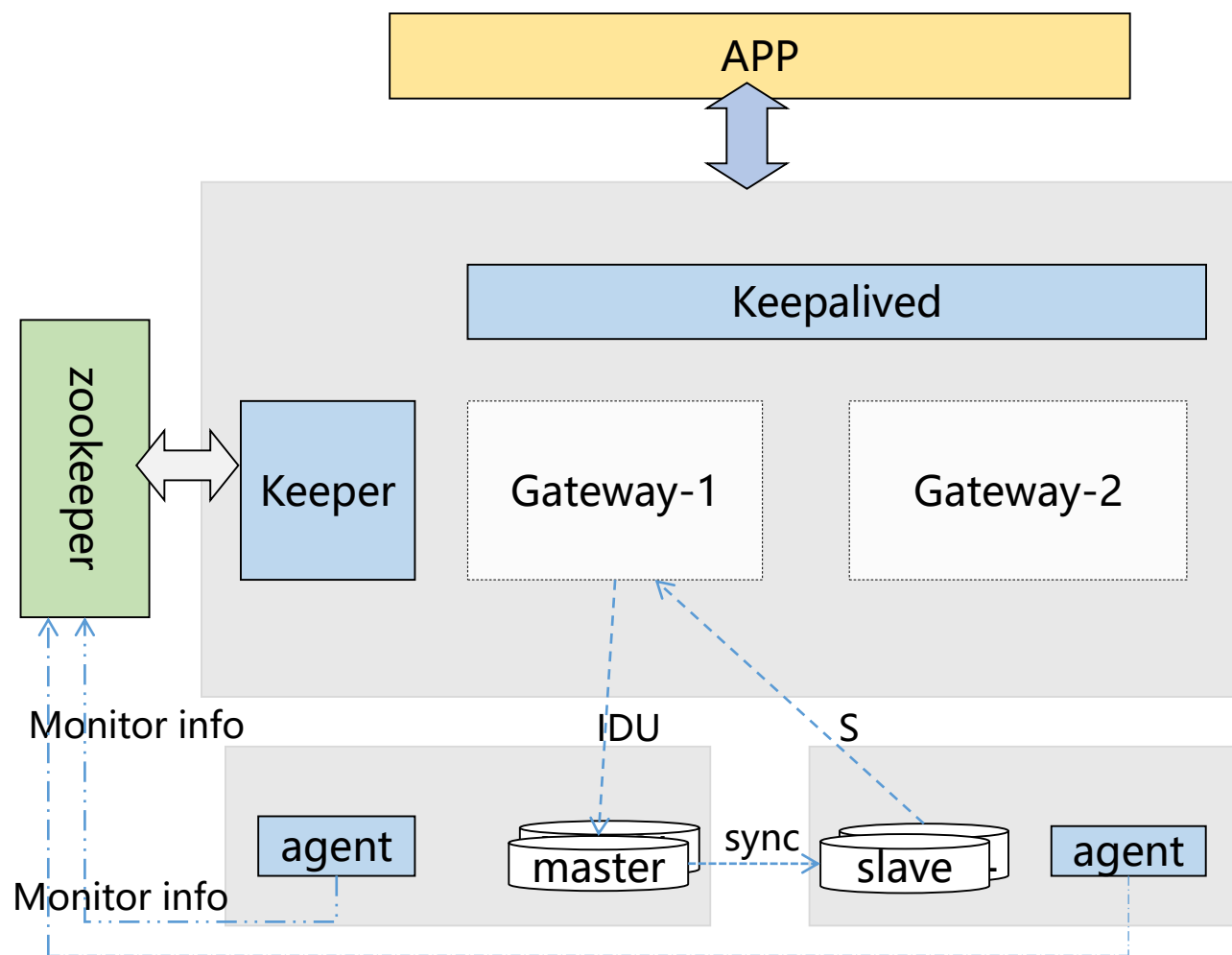
## □ 全局索引

- **未传入分片ID情况下**，通过全局索引定位分片。

## □ 数据汇聚

- 全局索引以及SQL条件不能定位具体某个分片时，**将SQL分发给多个数据库，再统一汇聚数据库执行结果。**
- 跨分片数据汇聚，需要解决跨节点的distinct，count，order by，group by以及聚合函数问题，通过多次查询的方式再计算，实现跨分片数据汇聚。
- 使用数据汇聚功能需要严格限制，数据汇聚功能只能应用在SQL的执行时间要求不是太高的场景。

# 数据架构关键点：数据库高可用方案



## □ 数据高可用：

- 故障自动切换功能，在主库发生故障时及时切换到从库保证高可用
- 高性能、高一致性的数据强同步方案
- 读写分离功能，提高数据并发访问能力

## □ 原则

- 追求一致性
- 主备切换

## □ 其他方案：

- Mysql cluster
- 双主配SAN存储
- 双主配DRBD存储
- 双主多从模式—Mysql半同步复制



# 数据架构关键点：数据存储-原则（1/3）

数据存储表类型：全局表、单库表、分片表

## 1. 聚集性

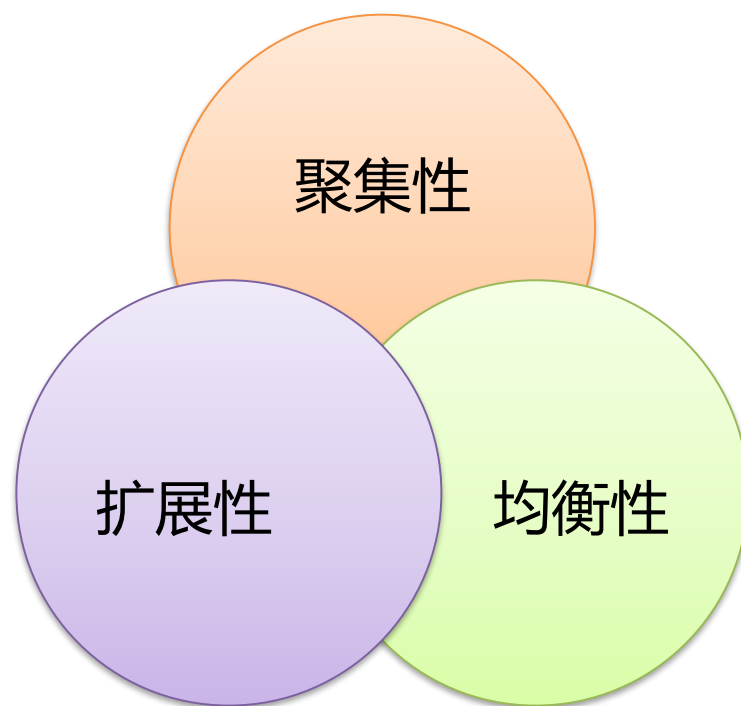
领域驱动设计里的聚集，把关系紧密的表放在一起，可在一定程度上规避跨分片JOIN、分布式事务，频繁调用分片索引。

## 2. 扩展性

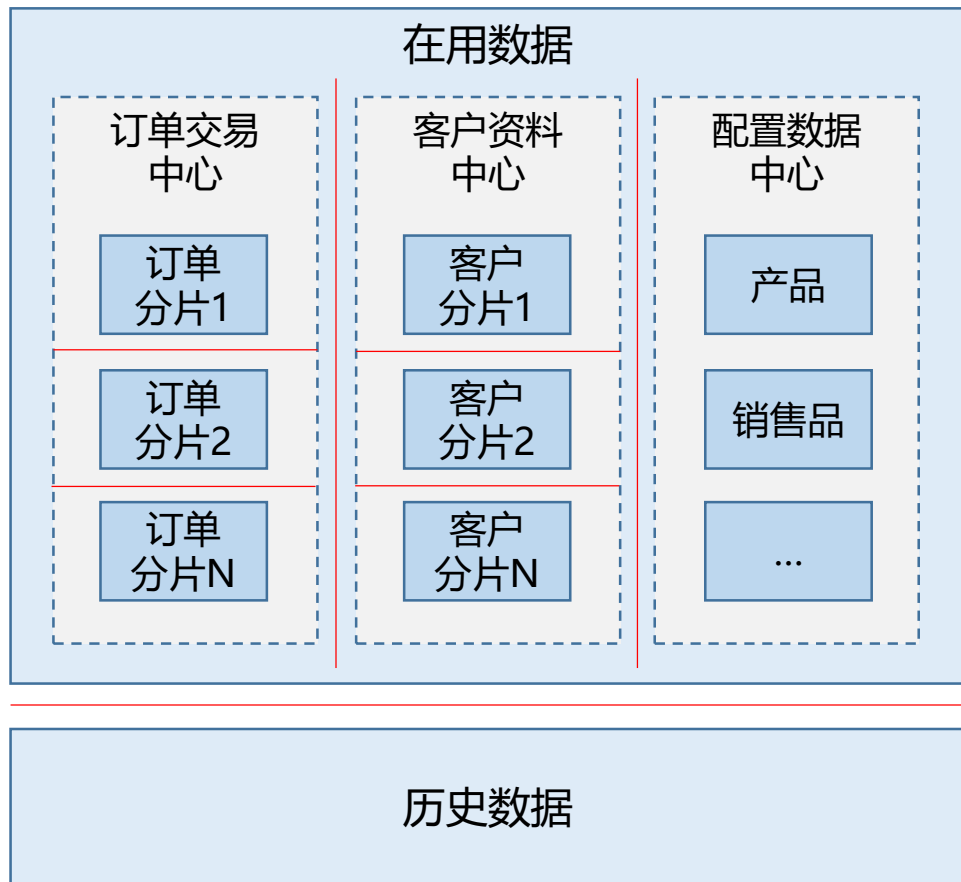
具备良好的扩展性，增加分片节点时，避免数据迁移，且已经达到存储上限的节点不再写入数据。

## 3. 均衡性

能均匀的分布数据读写，避免“热点”问题。



# 数据架构关键点：数据存储-方案（2/3）



## □ 总体方案

- **第一层，按时间**：分为生产在用表 + 历史数据库。
- **第二层，按功能子域**：分为客户资料中心、订单中心、营销资源中心、配置中心。
- **第三层，按ID散列**：对于客户资料中心、订单交易中心和营销资源中心这三类实体，分别根据客户ID、订单ID和营销资源ID散列存储，配置数据中心，作为公共数据，全部分片冗余存储。

# 数据架构关键点：数据存储-Oracle迁到MySQL存储模型方案（3/3）

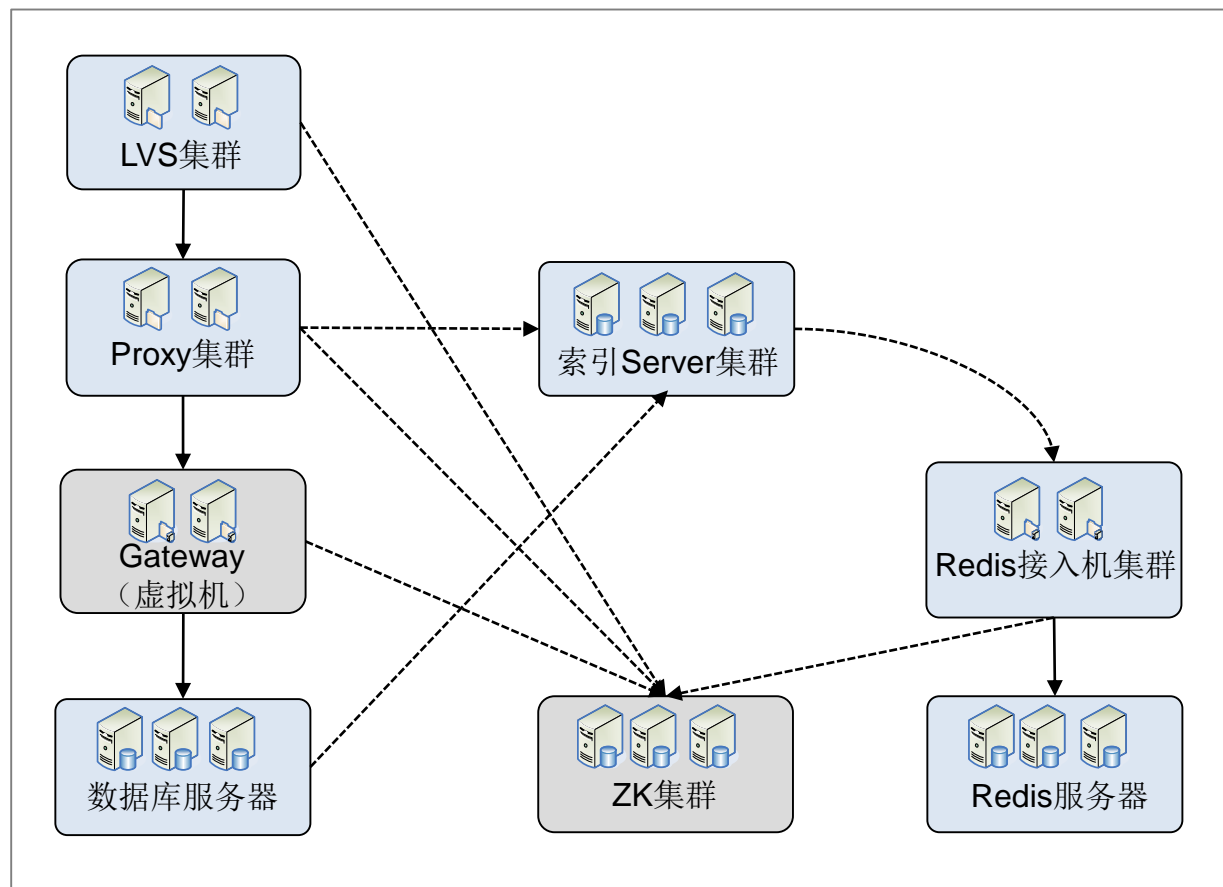
## 1、表、字段不变：分布式数据库的表、字段与传统Oracle保持一致，特殊点处理如下

- 类型映射：number对应bigint，varchar2对应varchar，date对应datetime，Oracle中的clob类型对应MySQL的text
- Oracle中的BLOB类型调整：部分调整到小文件系统，部分转换为string类型存储到mysql text字段（拆表）

## 2、分布式数据库增加分片字段

- 实例分片表统一增加Sharding\_id字段用于存储分片键
- 部分关系表冗余存储，并额外增加SHARDING\_ID\_OPPOSITE字段，用于存储对端的分片键值
- 支撑根据业务需要，在实例分片关系表上扩展对端的业务号码、区域等字段，方便查询使用

# 数据架构关键点：物理部署架构



## □ 数据存储节点：

- 物理机器数量：24台
- MySQL实例：每台2个，共48个
- 一主两备，形成16组数据存储节点
- 数据库数：每个MySQL实例4个

## □ 数据库高可用gateway

- 虚拟机：配套32台

## □ 数据库 Proxy：

- 物理机：4台

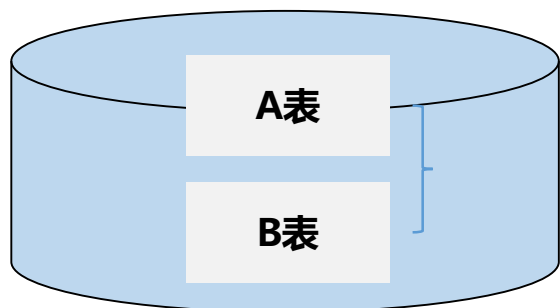
## □ 其他：

- ZK：公共

# 应用改造关键点：关联语句拆分

跨表数据关联改造：根据业务的特点判断查询的表数据是否放在同一个分库，并进行业务逻辑调整。

关联表在同一分库的情况：**查询的库表使用同一个分库字段进行查询**



Select ...  
From A, B  
Where ...

直接关联查询

例如：根据用户找出用户属性。直接在用户所在分库中进行资料表和资料属性表的关联。

关联表不在同一分库的情况：**查询的库表使用不同的分库字段进行查询**



Select ...  
From C  
Where ...



内存数据C'



Select ...  
From D  
Where ...

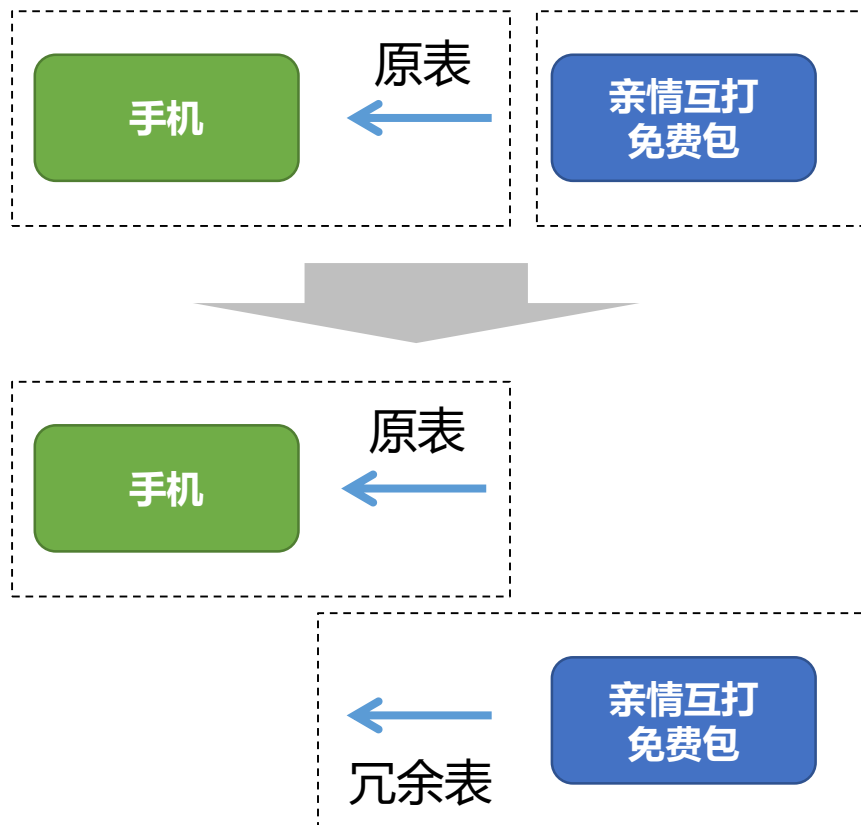


内存数据D'

由应用在内存中做关联

例如：根据销售品实例找其下所有用户的属性。需要先到销售品分库中找出销售品实例下的用户列表，再到各用户分库中找出用户属性。这种情况的难点在于要应用自己做数据关联处理。

# 应用改造关键点：冗余-关系类数据（1/3）



- 关系类数据冗余，根据业务逻辑要求，关联原表或者关联冗余表进行查询

# 应用改造关键点：冗余-业务索引（2/3）

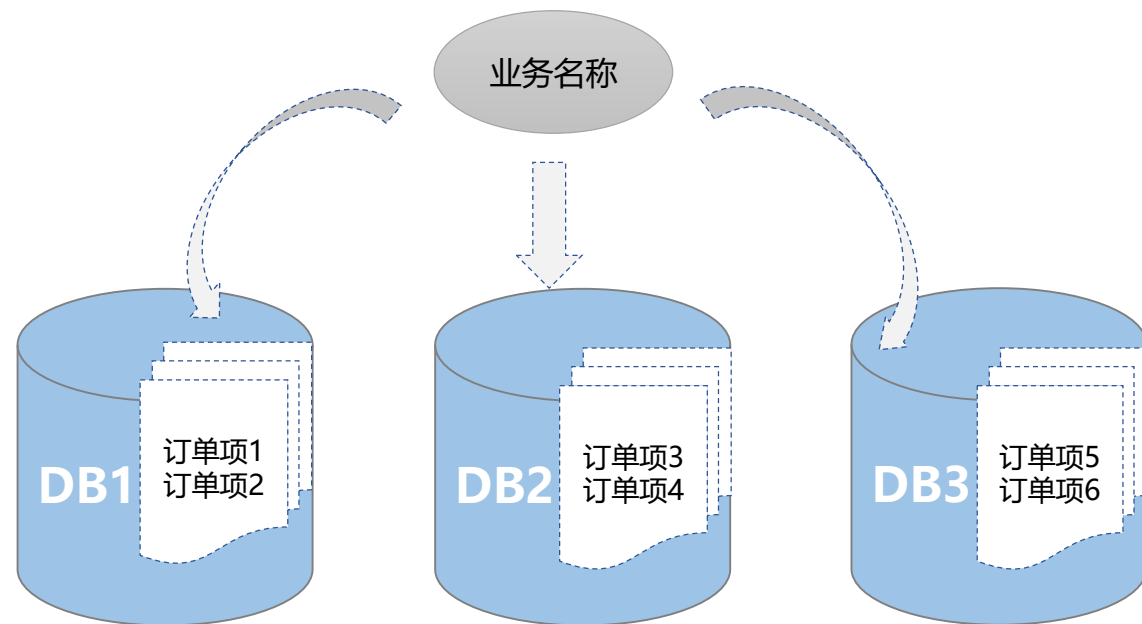
在数据分布部署的情况下，查询会变得复杂

## □ 业务场景

- 根据证件号码查询订单项列表
- 根据业务名称+客户名称查询订单项列表
- 根据客户名称查询订单项列表
- 根据客户编号查询订单项列表

## □ 问题描述

- Oracle，所有数据在一个数据库中，可以通过数据库索引和多表关联查询完成相应的查询需求。
- 在数据库分布部署之后，数据分散在多个数据库实例中，多表关联和数据库索引不能提供相应的帮助。

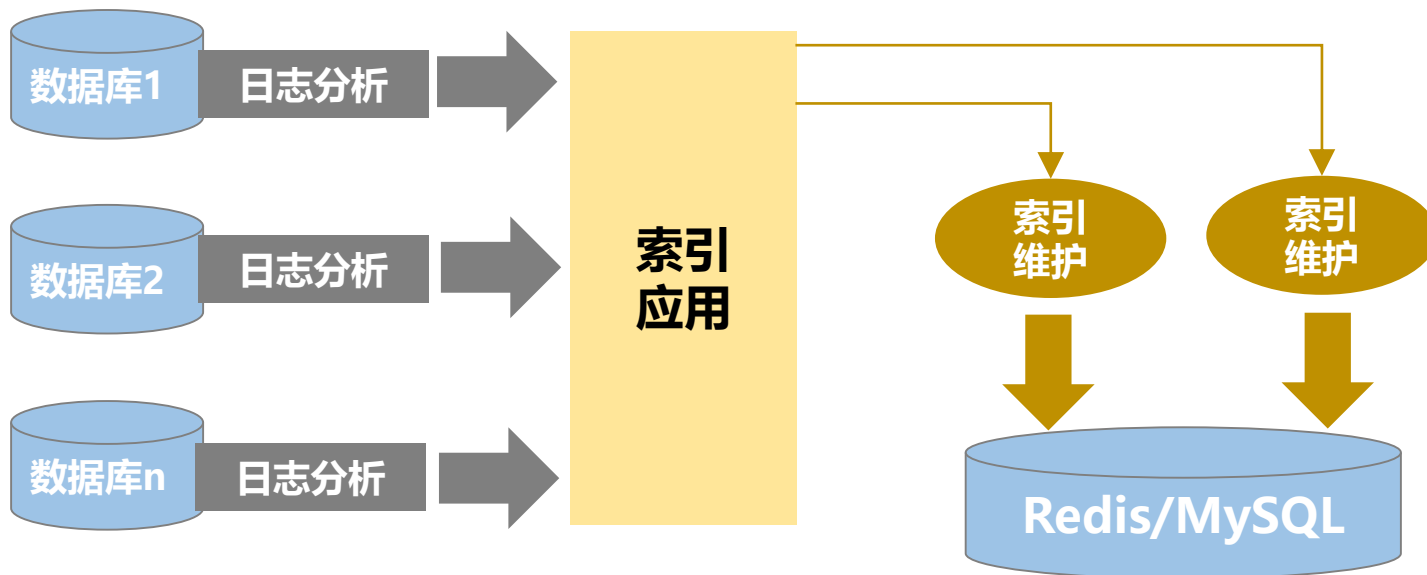


# 应用改造关键点：冗余数据更新（3/3）

生产过程中，数据会频繁的变化，需要在数据发生变化后能实时的更新业务索引和冗余数据。

应用处理：业务表及冗余表同时处理

□ 逻辑同步处理



□ binlog获取

□ 更新索引



# 应用改造关键点：分布式事务解决方案

## 1. 事务一致性场景

从业务场景分析，需要考虑事务一致性业务场景例子：

- 客户、订购业务，订单提交；
- 客户、订购业务，订单流程处理、竣工；
- 订单归档，历史数据到历史库；

## 2. 从事务完成数据时效要求，需要考虑事务一致性类型如下：

- 立即生效
- 快速生效  
允许时延：秒级、分钟级别
- 较长时间时延  
允许时延：小时级别或以上

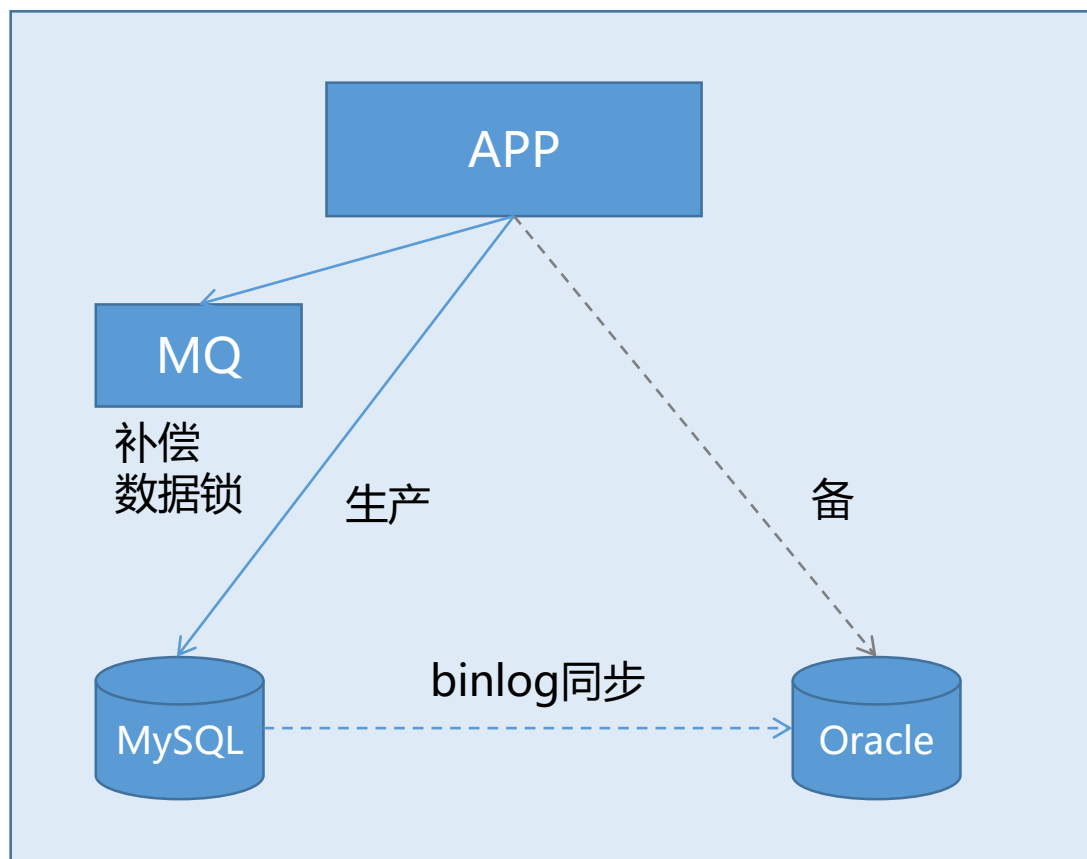
## 3. 方案

- 调整业务实现逻辑，将跨数据节点的事务**调整**为单数据节点事务
- 将分布式事务分解为具有**幂等性**子执行事务

## 4. 其他选择：

- XA
- 消息中间件

# 应用改造关键点：应用升级保障机制



初期应用升级保障机制

**假设MySQL数据库无法提供服务，判断无法快速恢复的情况下，应用能够在线切换到备用的Oracle提供服务**

- ❑ 应用数据双写：同时写MQ和MySQL事务提交
- ❑ 数据同步：
  - 正常：binlog
  - 补偿：消息
- ❑ 数据锁：
  - MQ->Redis
  - 应用SDK判断
- ❑ 在线数据源切换：
  - 支持Oracle、MySQL两两组合
  - 在线切换，SQL语法转换

# 数据集成关键点：总体要求

## □数据迁移

不停业务系统**从集中式的ORACLE数据库迁移数据到分布式MySQL数据库集群。**

## □数据同步

**同其他系统之间的数据共享交换。**

## □数据稽核

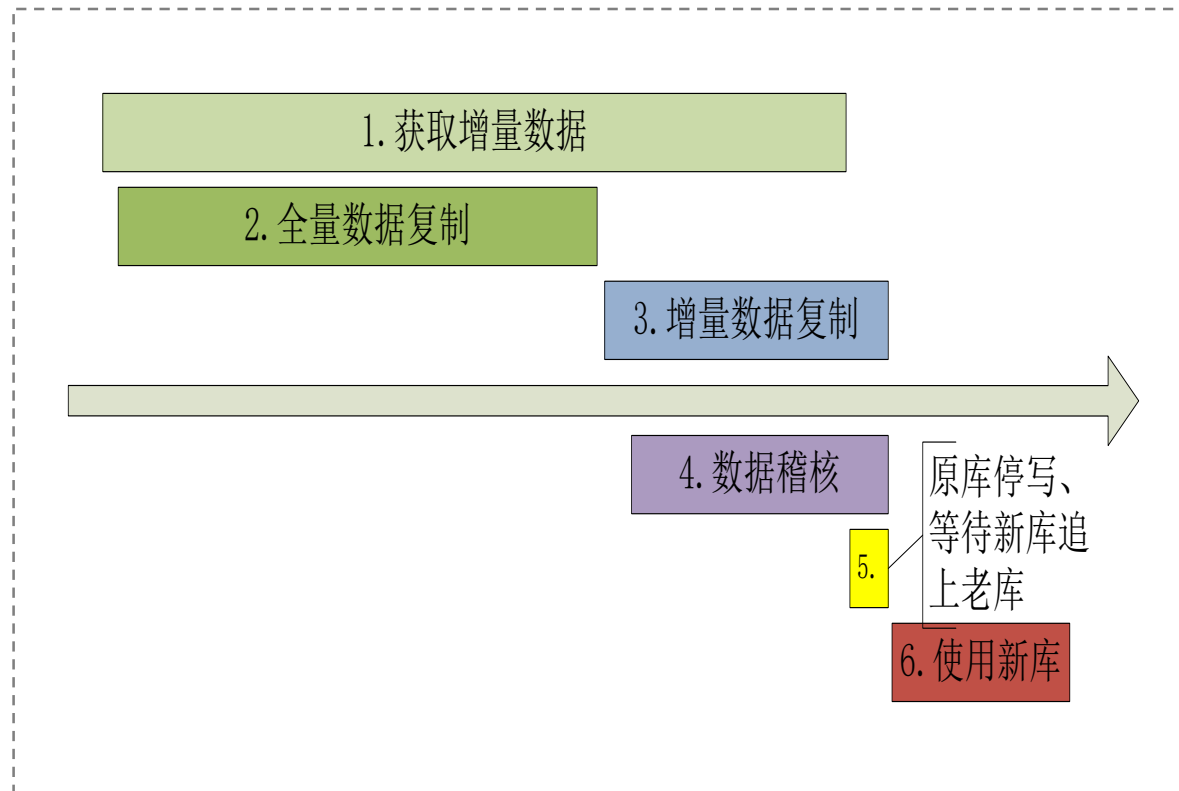
对集中式数据库和分布式数据库的数据进行比对、**验证数据的一致性**，。

# 数据集成关键点：在线数据割接

## □ 异构数据库在线迁移 ( Oracle、MySQL )

### □ 总体方案

- 基于 **ETL** 工具 Kettle 封装
- 通过 **全量数据迁移+增量数据迁移** 来满足不停业务系统在线数据迁移；
- 通过 **表分段** 方式满足断点续传功能；
- 通过多线程方式满足高效、**快速数据迁移**和复制。

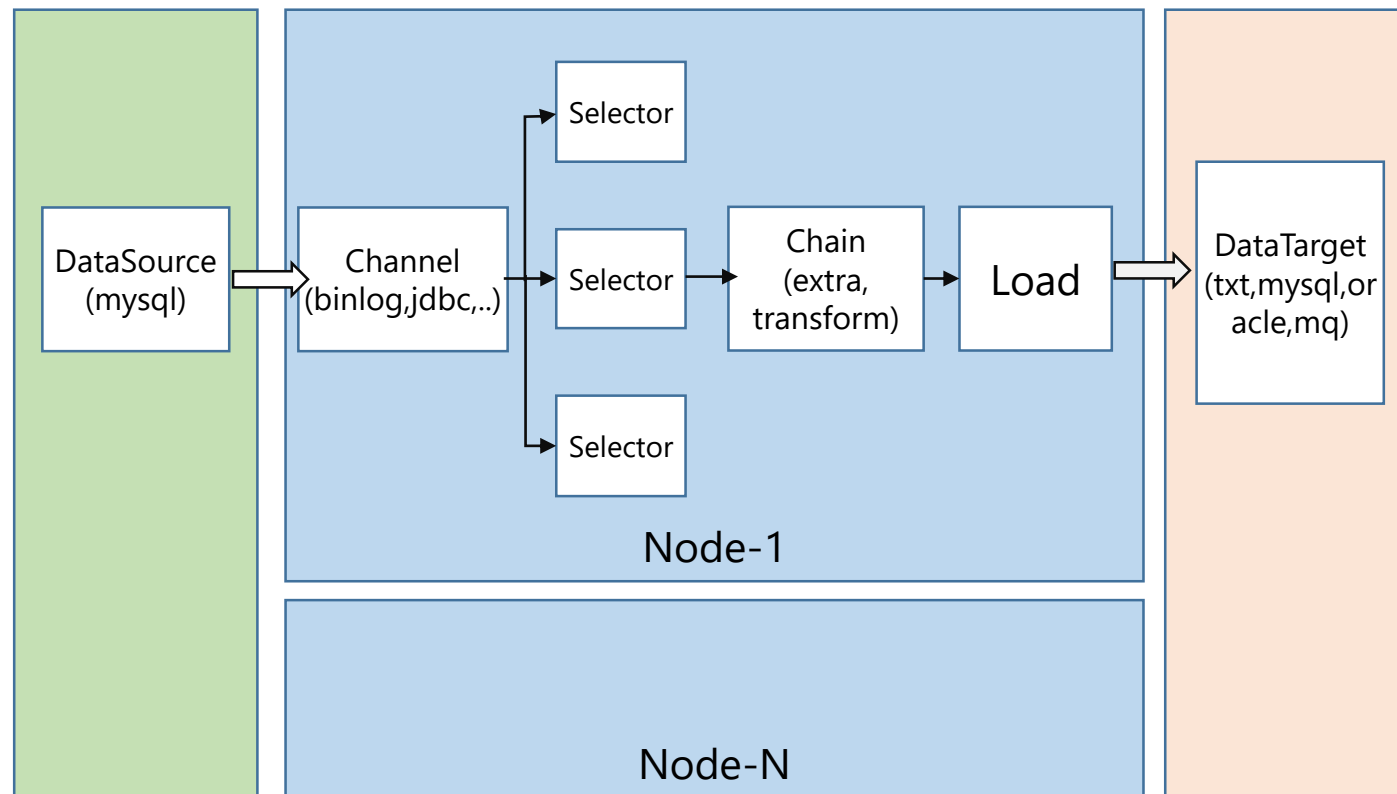


# 数据集成关键点：数据共享给外围系统

❑ 基于分布式MySQL数据库**增量日志**解析，提供增量数据订阅&消费。

## ❑ 总体方案

- 1、channel获取一批数据，将批次ID放到List，本批数据，放到一级本地队列；
- 2、selector从本地队列获取，将全数据（所有表）过滤为单任务（指定表）数据，并存储到本地队列；
- 3、extract从selector，获取对应批次数据，进行表等映射处理，保存到数据库、或者文件，如果需要FTP到远程服务器，触发传送事件。



# 数据集成关键点：数据稽核方案

**定期稽核：**采用被动拉的模式，**通过设定一定的时间间隔收敛数据进行比对。**

## □ 场景

- 时间间隔无需很短
- 比对最终结果，无需了解实时的情况

## 模式一：定期被动拉的模式

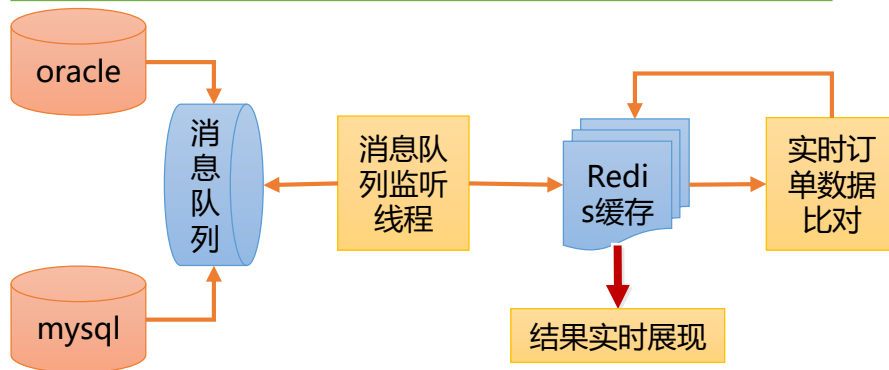


**实时稽核：**主动侦听各个环节的数据情况，进行**实时的数据比对、耗时计算、异常统计**等。

## □ 场景

- 要求实时比对数据
- 从整体的角度对各环节做比对

## 模式二：实时主动推的模式



# 数据运维关键点：批量执行、取数工具

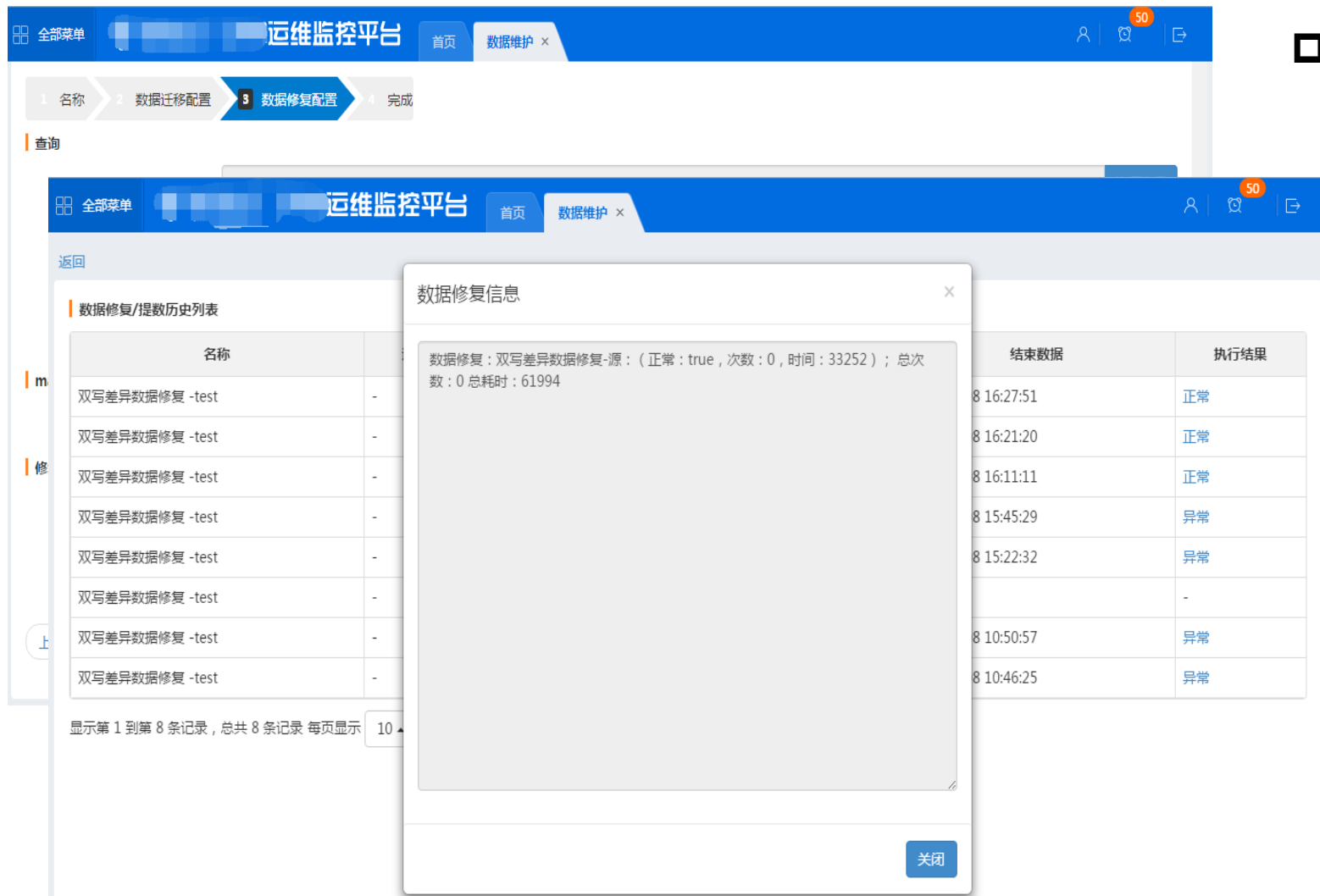
## ❑ 批量执行

- 应用（数据库）CMDB及执行工具
- 版本发布的**模型变更**、**数据脚本执行**
- 减少重复登陆、执行等操作

## ❑ 取数工具

- 省、市两级临时取数
- 周边系统临时取数
- 工单任务

# 数据运维关键点：PLSQL、存储过程替代工具



## 基于MySQL集群的数据运维工具

- 配置化、模版化
- 个案、批量
- 模型熟悉度：IT人员、非IT人员

便捷、可靠  
性不足



# 研发维护规范：研发规范

## □ 架构设计考虑因素

- 模型设计
- 读写分离

## □ 字段设计规范

- 字段类型越短越好
- 尽量不要使用default null
- 多表关联的字段，数据类型需保持绝对一致
- 当字段的类型为枚举型或布尔型时，建议使用tinyint类型
- 一般情况下不允许使用TEXT、BLOB，确实需要则拆分。
- 关于存储IP地址时字段类型的选择（可选）
- 关于存储时间字段类型的选择（可选）

## □ 数据表设计规范

- 统一使用InnoDB存储引擎
- 创建表时要明确定义主键
- 表设计时必须包含公共字段
- 每个数据表中字段数量尽可能少
- 每个数据表记录数尽可能少
- 创建表时，所有表名、字段名都需要添加注释。
- 表字符集统一使用utf8mb4，核对规则为utf8mb4\_bin。

## □ 索引设计使用规范

- 索引的数量要控制
- 主键准则
- 重要的SQL必须被索引
- 复合索引准则
- 索引禁忌
- 不使用外键
- 合理使用复合索引
- LIKE查询的索引问题，注意使用前缀索引

如何能够完全遵循？


# 研发维护规范：SQL规范

## □ 常见问题

- 执行计划错误
- 索引无效
- 参数类型错误
- 未使用绑定变量

## □ 我们的规范

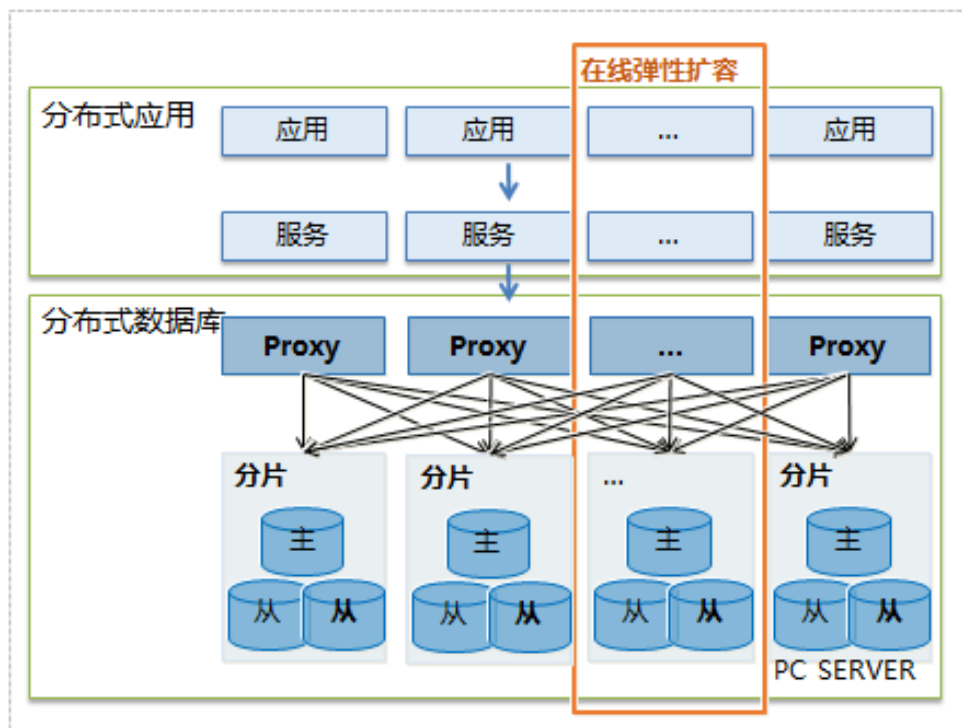
- 语句尽量简单，避免数据库做复杂运算，不用存储过程、函数
- Select一定要带上字段名，避免使用select \*
- 避免在条件语句中对索引列做运算或表达式
- 复杂查询拆分简单查询
- 避免数据类型隐式转换
- 尽量用 join 代替子查询
- 尽量少用or
- 尽量用 union all 代替 union
- 数据范围尽量早过滤



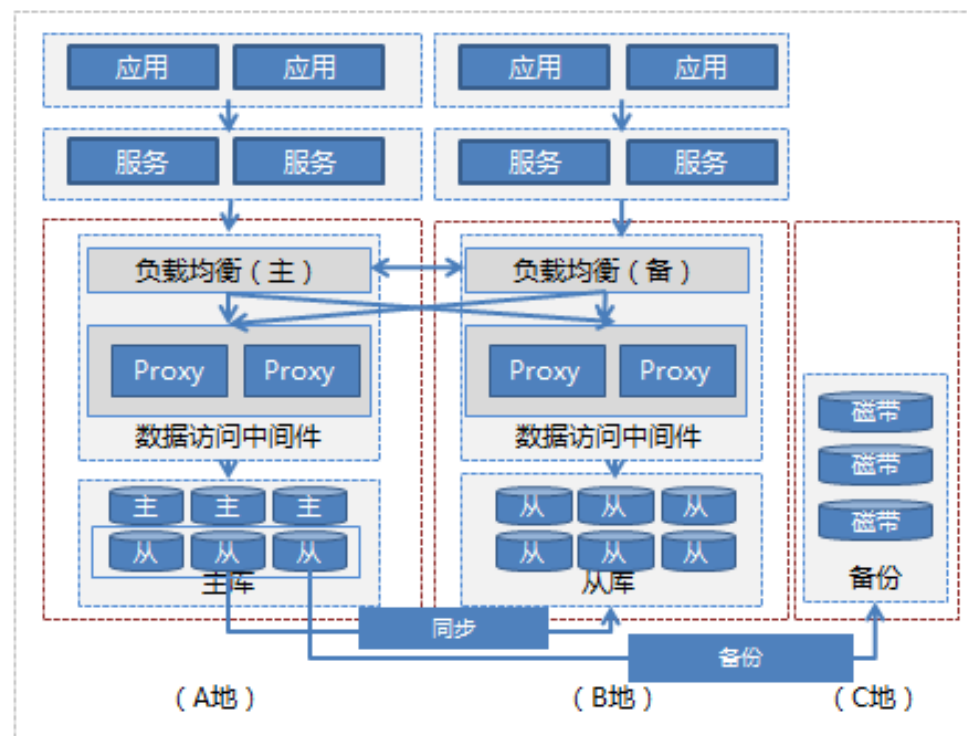
如何能够  
完全遵循？

# 继续前行...

实现资源动态按需调整，依据用户请求量和任务加载量**弹性扩缩容**



全链路无单点方案，实现系统**异地机房多活容灾**，保障业务不受“单地域灾难”影响



# 继续前行...

诚邀加盟，共同成长

- Web前端/J2EE后端开发
- H5/andriod/iOS开发
- 分布式技术开发
- 运维自动化

Email : wuyx@ffcs.cn



Thanks