# Gdevops

# 全球敏捷运维峰会

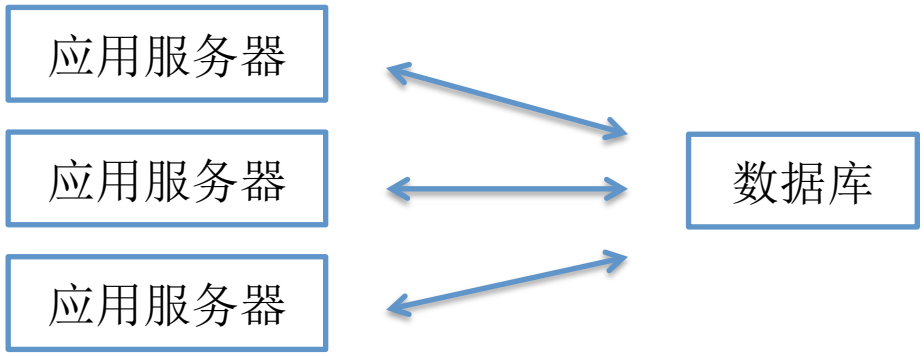
## 从零开始搭建百万每秒订单系统

演讲人：梁阳鹤

# 目录

# 订单系统关键组件

# 一个最简单的订单系统




3台应用服务器连1台数据库，数据库中有最核心的order表，一次正向订单处理流程如下：

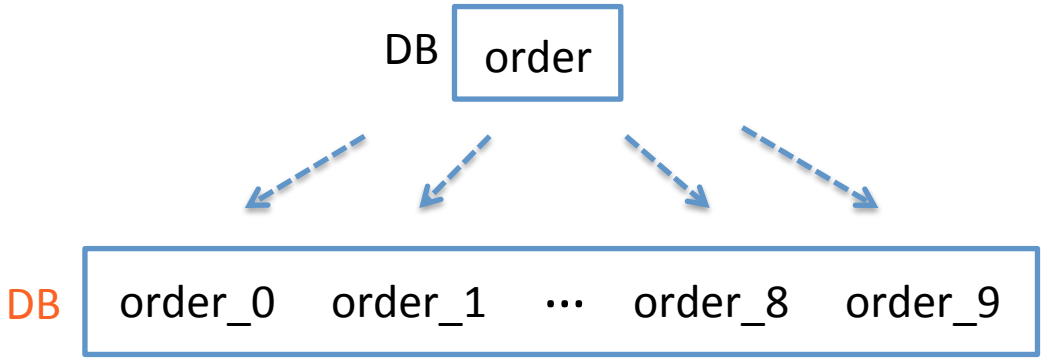
1. 下单操作，应用服务器向数据库order表插入订单，订单状态为未支付
2. 调起支付平台完成支付
3. 支付成功，应用服务器将订单状态修改为已支付

下单QPS：7000

# 目录

# 物理分表

物理分表指的是将表拆成N张分表，在进行增删改查时，根据分表字段计算最终目标表，并在此表上进行操作




**优点：**

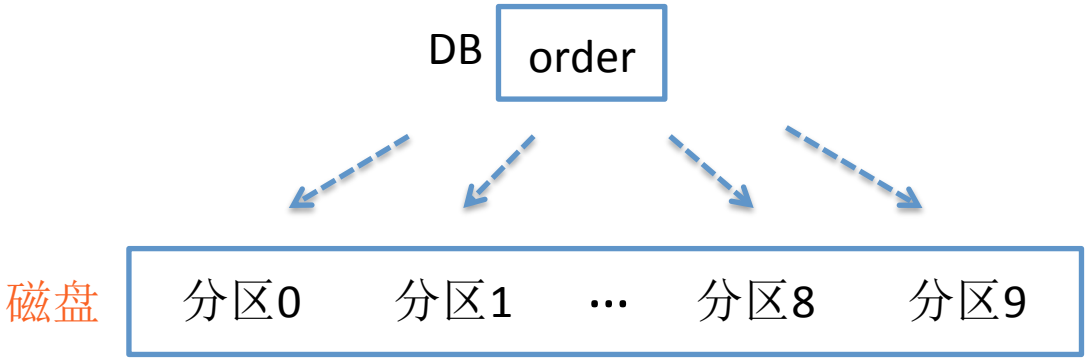
- ✓ 减缓单表写入压力
- ✓ 减小单表容量

**不足：**

- ✓ 增加编码复杂度
- ✓ 有些查询需要合并数据

# 逻辑分表

逻辑分表也称分区表，对外所有的数据还在一个表中，但物理存储数据根据一定的规则存放在不同的文件中




**优点：**

- ✓ 减缓单表写入压力
- ✓ 对外透明，还是一张表

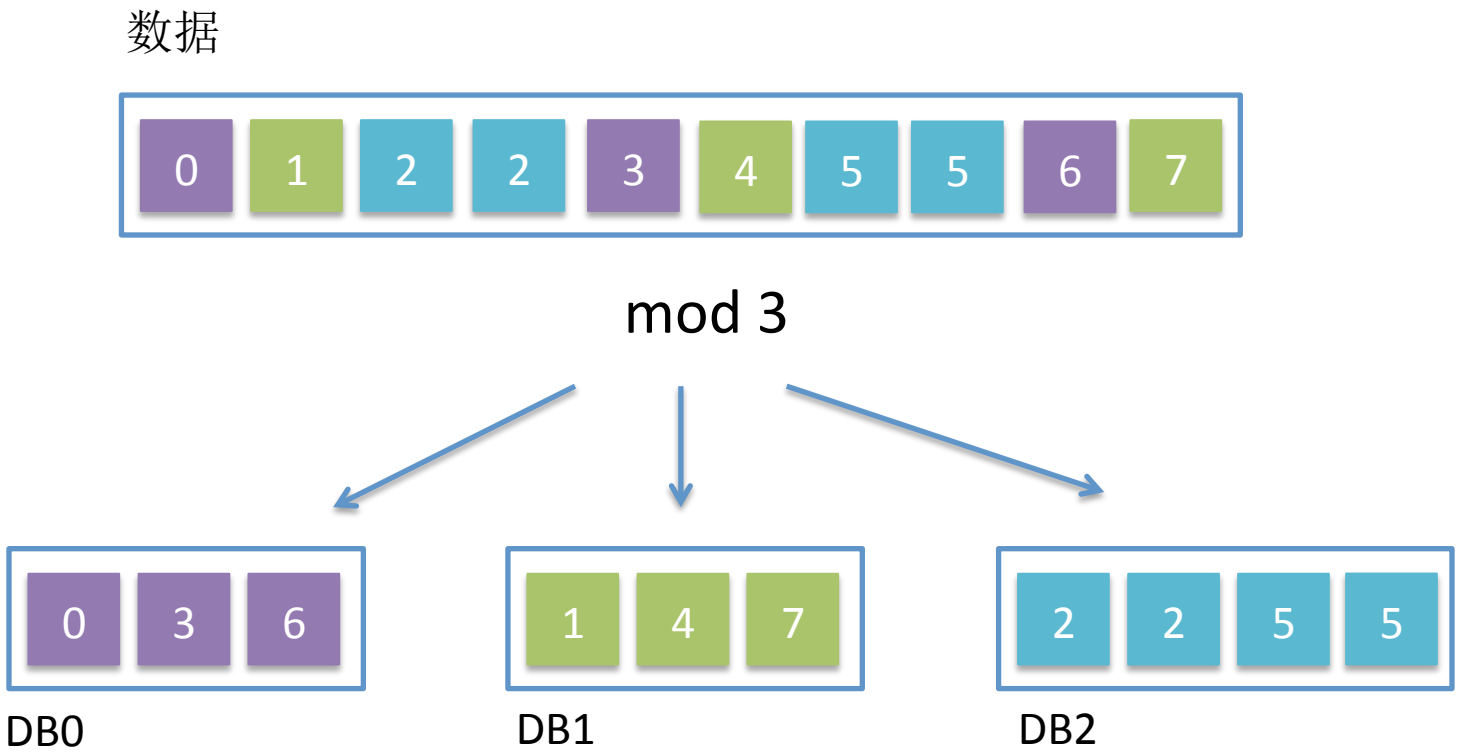
**不足：**

- ✓ 单表数据量不会减小
- ✓ 性能提升比物理分表略低

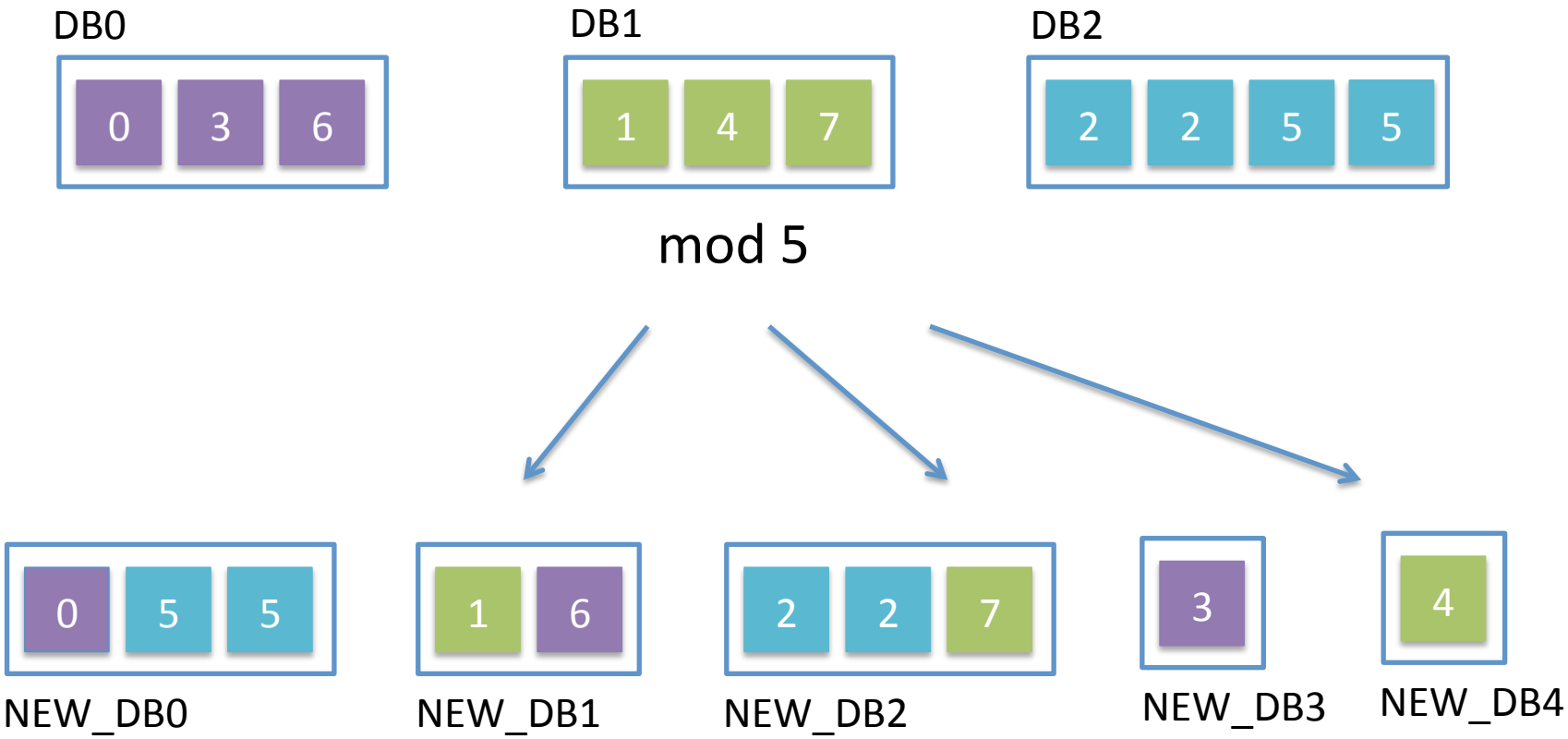
# 目录

# 简单取模分库

简单取模分库指的是将分库字段模N，得到序号0到N-1共N个数字，并将数据放到对应序号的数据库中



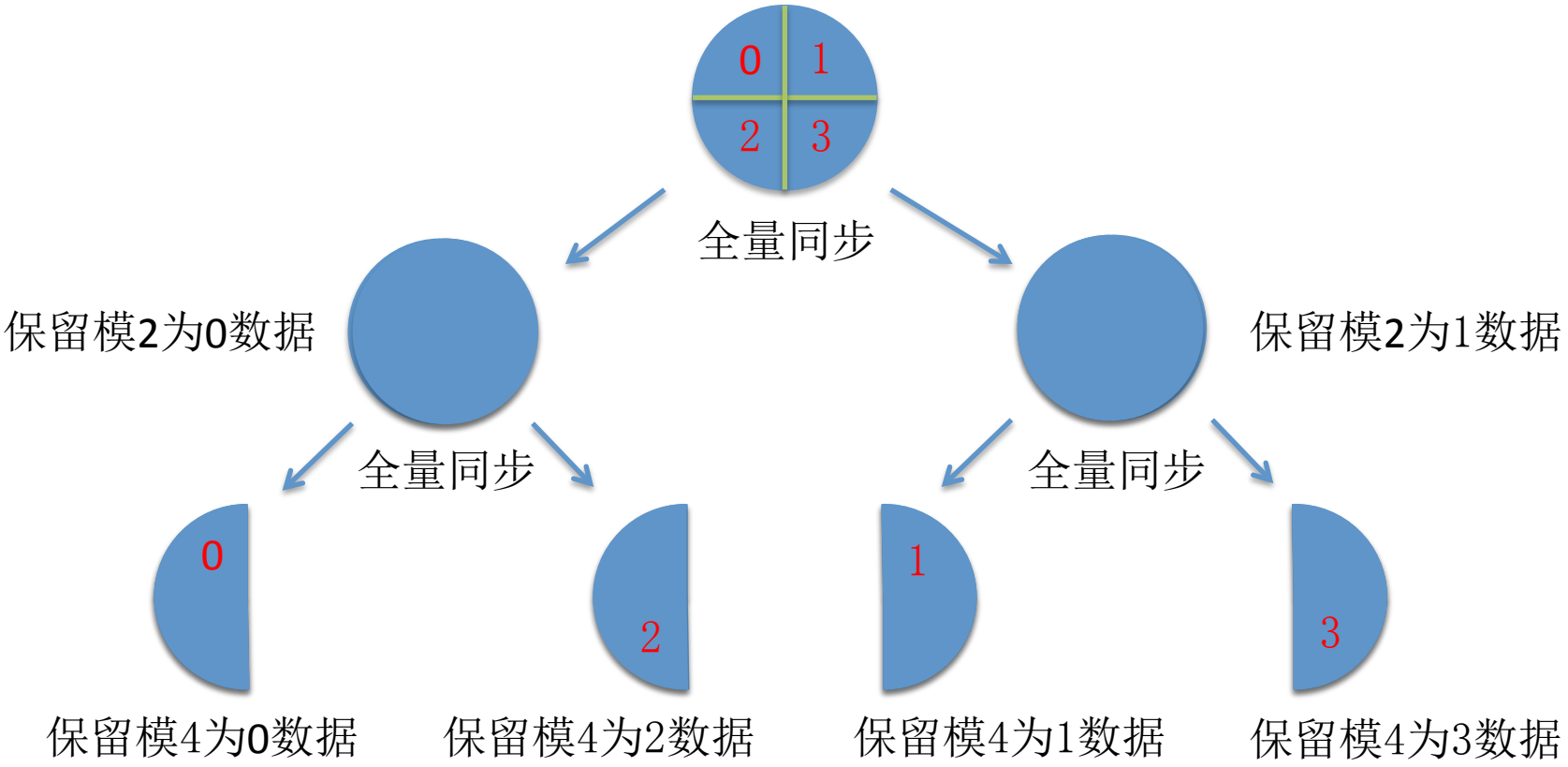
# 简单取模分库扩容



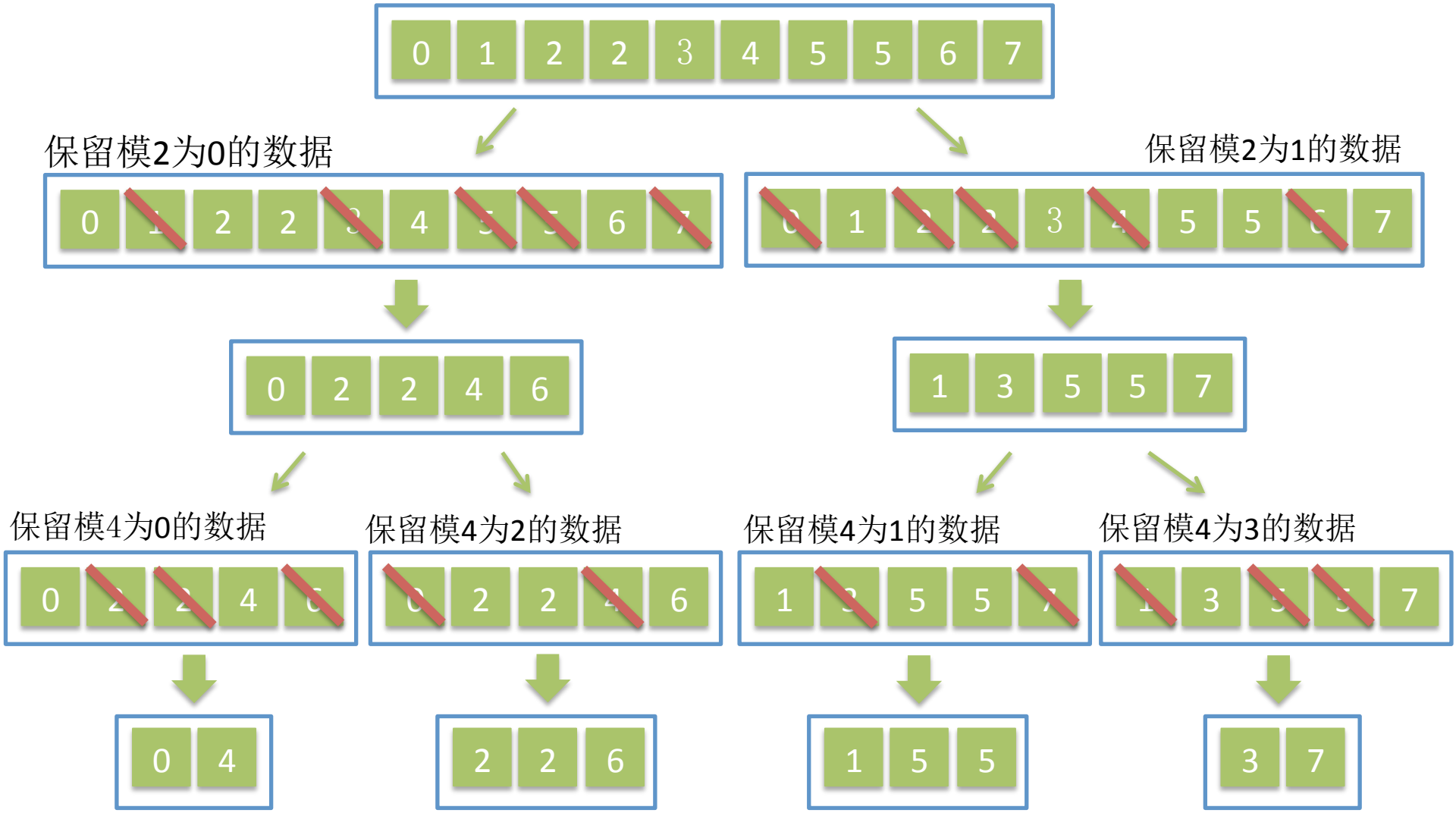

✓ 简单取模分库在扩容时需要进行行级数据迁移

# 表级数据迁移模型

表级数据迁移是指在数据库扩容时，能直接通过主从同步等简单的全表复制操作完成数据迁移扩容，不需用脚本按行迁移数据



# 表级数据迁移扩容实例

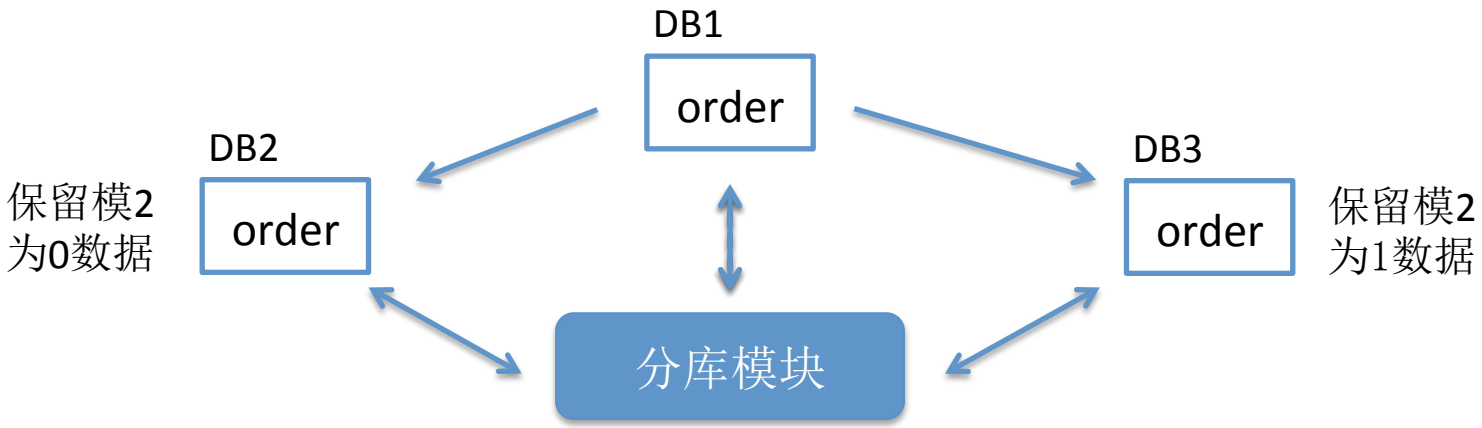

0 1 2 2 3 4 5 5 6 7
保留模2为0的数据
保留模2为1的数据
0 2 2 4 6
1 3 5 5 7
保留模4为0的数据
保留模4为2的数据
保留模4为1的数据
保留模4为3的数据
0 4
2 2 6
1 5 5
3 7

# 表级数据迁移扩容流程

我们假设一开始分库模块连接唯一数据库DB1，现在我们要将数据库扩容到两台，下面时操作流程：

1. 将DB2和DB3做为DB1从库，全量同步DB1中的order表
2. 断开同步，将DB2和DB3做为主库（DB1可选择是否停止写入）
3. 分库模块采用模2算法连接DB2和DB3，不再连接DB1
4. 删除DB2和DB3中order表的冗余数据

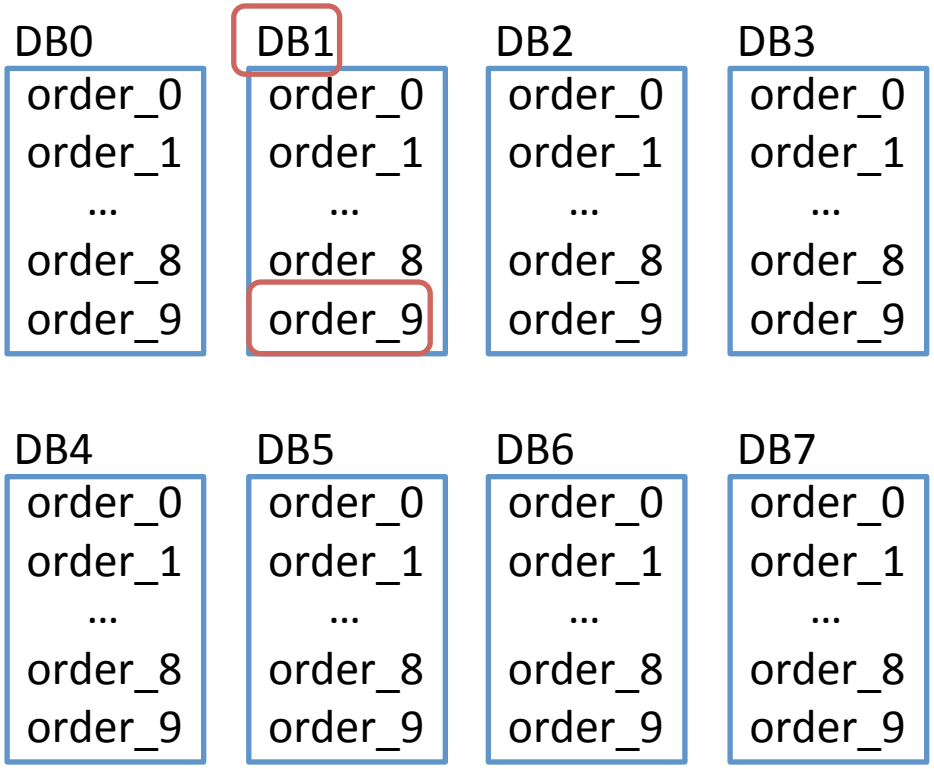
合并操作



# 目录

# 用户ID维度订单表集群

数据结构




## 算法

- ✓ 用户ID简写为uid
- ✓ 数据库编号 = (uid / 10) mod 8
- ✓ 表编号 = uid mod 10

## 举例

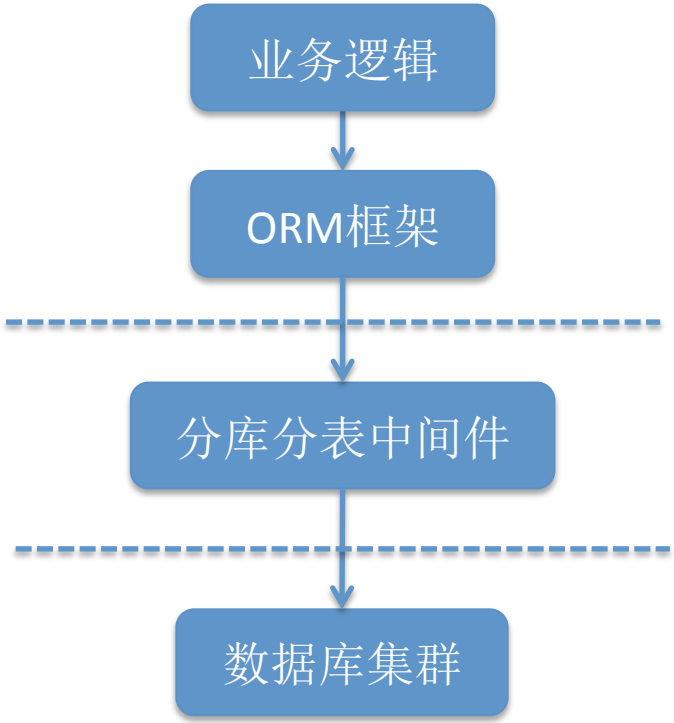
- ✓ 当uid=2019时
- ✓ 数据库编号 = 1
- ✓ 表编号 = 9

## 实施

- ✓ 分库分表中间件
- ✓ 本地分库分表

# 分库分表中间件

分库分表中间件是应用服务器与数据库中间的一层服务，该服务一般会伪装成数据库接收来自应用服务器的SQL，并通过解析SQL完成数据库路由和表拆分




select xx from order where uid = 2019

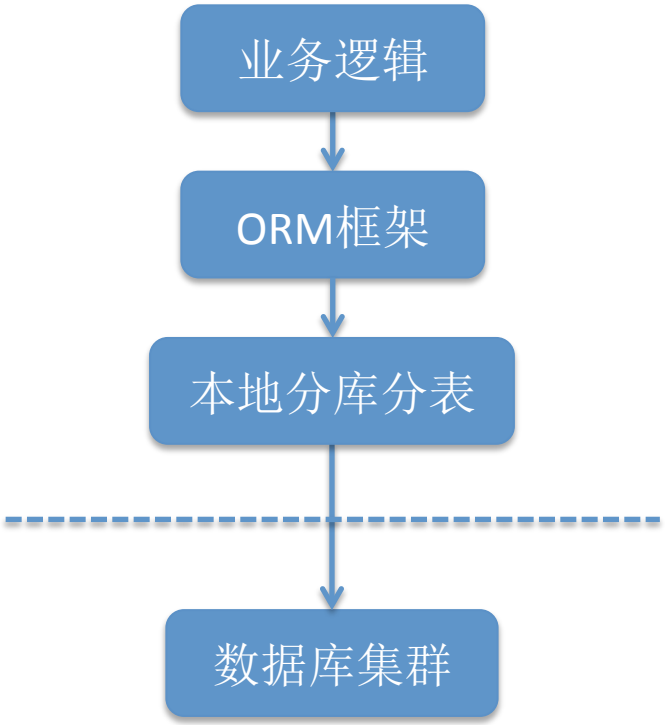
路由到DB1

SQL修改为：

select xx from order_9 where uid = 2019

# 本地分库分表

本地分库分表是指在应用服务器解析SQL，直接完成数据库路由和表拆分，应用服务器和数据库直连




select xx from order where uid = 2019

路由到DB1

SQL修改为：

select xx from order_9 where uid = 2019

# 中间件VS本地

## 分库分表中间件

优点：

- ✓ 对应用层完全透明
- ✓ 层次分明，责任清晰

不足：

- ✓ 性能损耗15%到20%
- ✓ 引入新服务，稳定性降低

## 本地分库分表

优点：

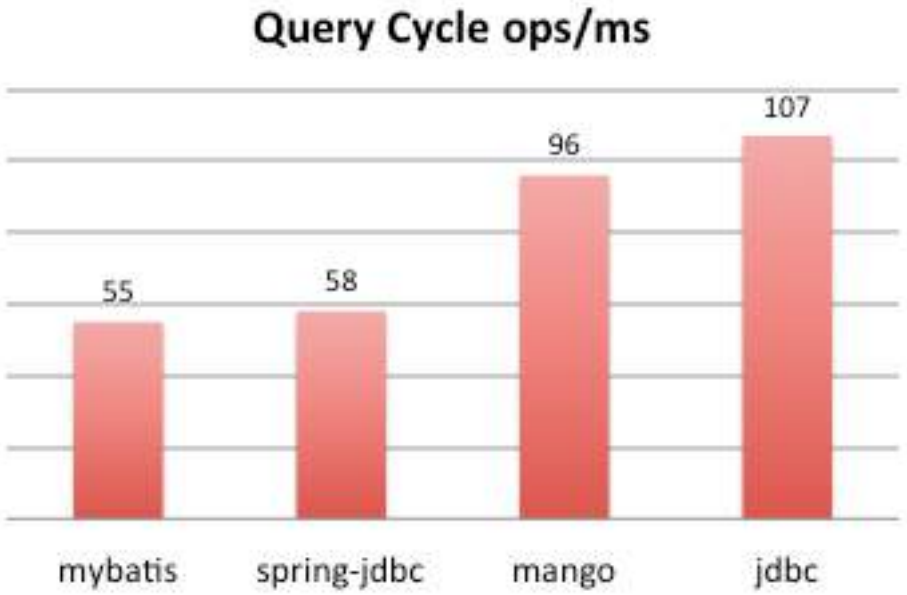
- ✓ 性能几乎无损耗
- ✓ 不引入新服务，简单稳定

不足：

- ✓ 应用层编码复杂度加大

# 分布式ORM框架Mango

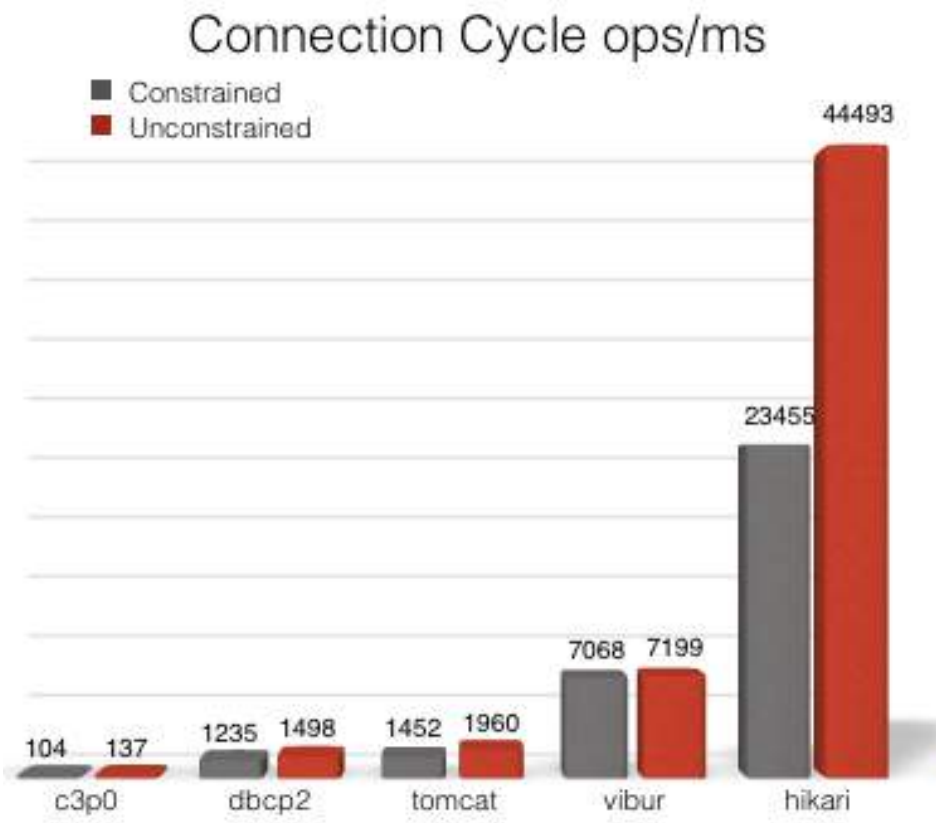
- 超高性能
- 金融级稳定
- 使用简单
- 支持分库分表
- 支持读写分离

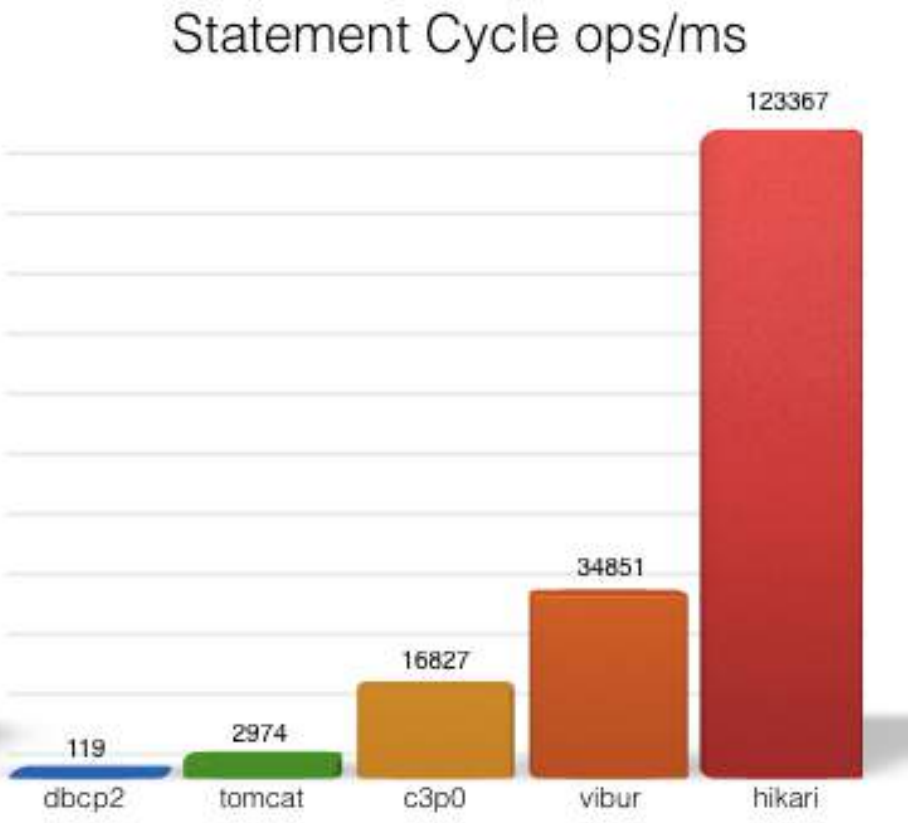



Mango主页：mango.jfaster.org

乐视集团支付分库分表开源实现：github.com/liangyanghe/letv-payorder

# 数据库连接池HikariCP




Github地址：github.com/brettwooldridge/HikariCP

# 目录

# Snowflake算法

Snowflake算法是由Twitter开源，用于生成64位全局唯一ID的算法


**时间戳**：当前时间，粒度为毫秒

**机器号**：不同服务器分配唯一机器编号

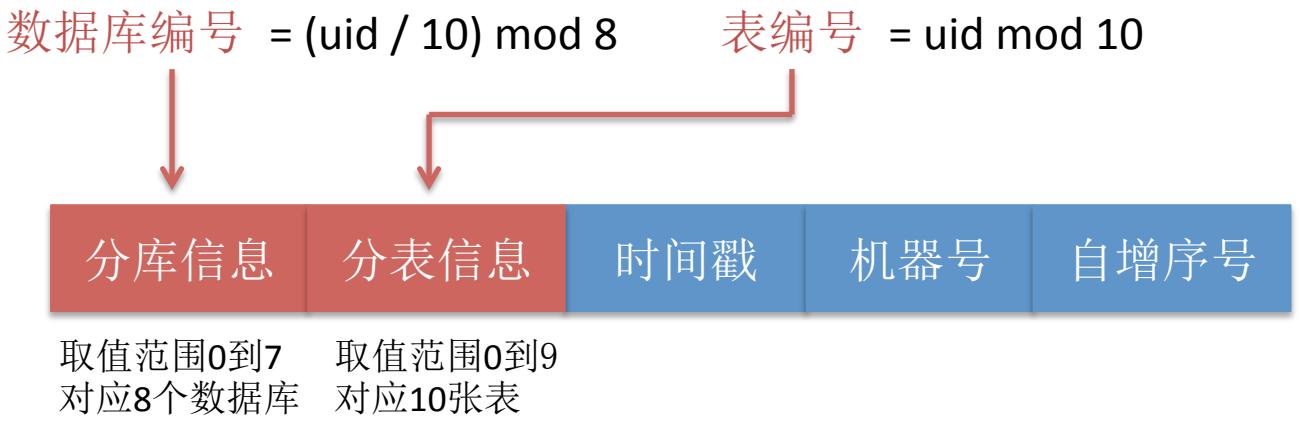
**自增序号**：同一毫秒多次请求，自增此序号

✓ 服务器时间一致　　✓ 无严格序列性　　✓ 默认使用到2082年

Snowflake相关：github.com/twitter/snowflake

# 订单ID添加分库分表信息

我们会在订单ID上添加分库分表信息，这样在没有uid时，我们也能根据订单ID查询订单信息




## 根据订单ID查询时

- ✓ 实际数据库编号 = 分库信息
- ✓ 实际表编号 = 分表信息

## 问题

当扩容到16个数据库时，根据8个序号，无法确定数据在哪个数据库中

# 分库精度冗余

分库精度冗余用于解决数据库扩容时订单ID中分库信息精度丢失的问题，假设有$2^N$个数据库，我们在计算分库信息时不模$2^N$，而是模$2^M$（M>N），这样在数据库扩容到$2^M$时都不会出现分库精度丢失

我们假设N=3，M=6，那么目前共8个数据库，能保证扩容到64个数据库

冗余分库信息 = (uid / 10) mod 64


## 根据订单ID查询时

8个数据库时： 实际数据库编号 = 分库信息 mod 8

16个数据库时：实际数据库编号 = 分库信息 mod 16

64个数据库时：实际数据库编号 = 分库信息

# 目录

# 分库是否能无限扩容？

8个数据库支撑10万每秒订单写入

→ 128个数据库支撑100万每秒订单写入？

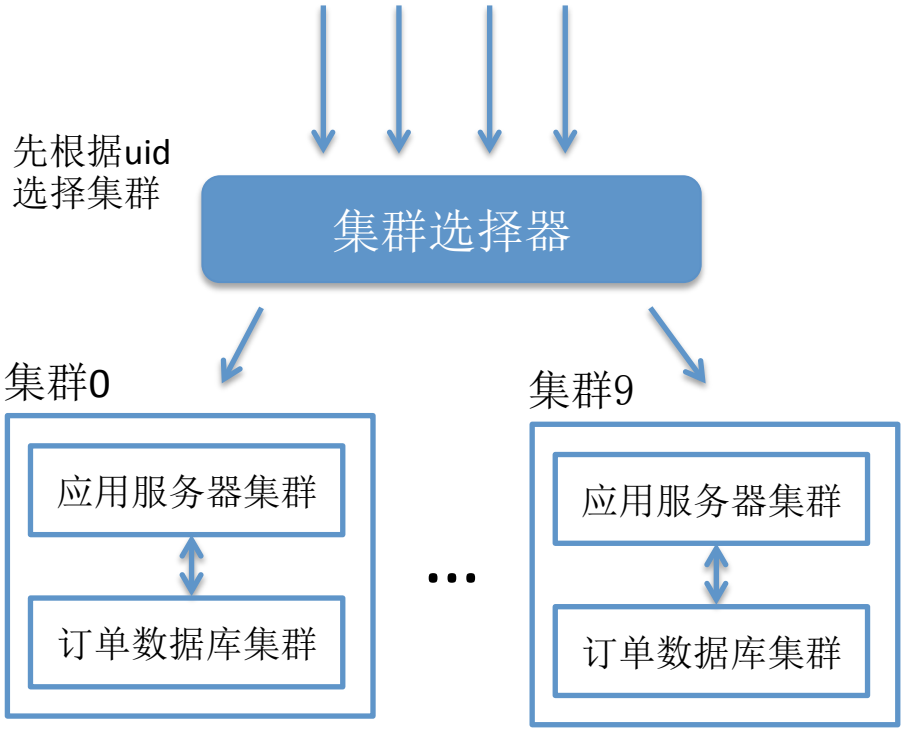
假设单台DB最大连接数为2000，1台DB与3台WEB为最佳搭配

| DB数量 | WEB数量 | 每台WEB到每台DB的最大连接数 |
|---|---|---|
| 1 | 3 | 400（最大能取到666） |
| 8 | 24 | 2000 / 24 = 83 |
| 128 | 384 | 2000 / 384 = 5 |

单台DB最大连接数限制是制约分库能无限扩容的主要原因

# 集群拆分模型

如果将分库分表看作二维拆分，那么集群拆分将是第三维度，我们可以使用分库分表搭建多个订单系统集群，每个集群数据独立，再根据用户ID的某些字段将请求路由到不同的集群




一种拆分策略

- ✓ 假设uid=XXXYZ
- ✓ 数字Z用于集群选择
- ✓ 数字Y用于分表
- ✓ 数字XXX用于分库

10个集群，每个集群支持10万，则总集群支持100万

# Gdevops

# 全球敏捷运维峰会

THANK YOU！