

# / 高并发支付系统架构设计与实践

——梁阳鹤

2016.8.12

# 支付收银台

 收银台

订单编号: 201607181648496793  
请您在2016-07-18 16:58:49内完成支付, 以免逾期订单被取消

 支付宝  
推荐支付宝用户使用

 中信银行 (尾号:1266) 快捷支付  
使用银行卡支付, 一步付款

 北京银行 (尾号:4380) 快捷支付  
使用银行卡支付, 一步付款

 信用卡快捷支付  
信用卡快捷支付, 无需网银

 储蓄卡快捷支付  
储蓄卡快捷支付, 无需网银

... 其他支付方式

应付金额: 88 元

下一步

## 支付收银台所面临的问题

1. 用户与支付服务器建立快速的通讯网络
2. 读各种配置数据
3. 读用户绑卡信息
4. 对支付订单表的写入
5. 支持高并发抢购或秒杀

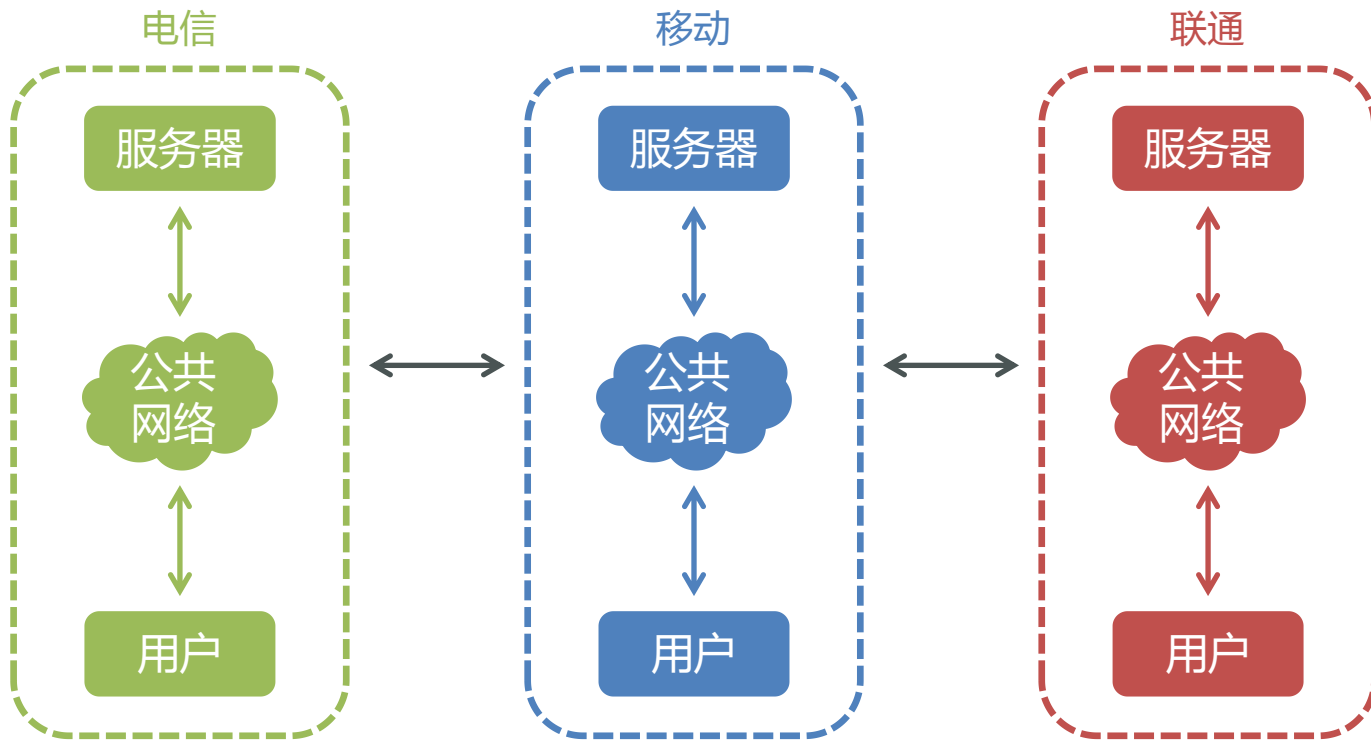
# 目录 CONTENTS

-  一 简单网络拓扑图
-  二 支付系统数据分级
-  三 本地内存与配置管理系统
-  四 分布式缓存
-  五 分布式数据库

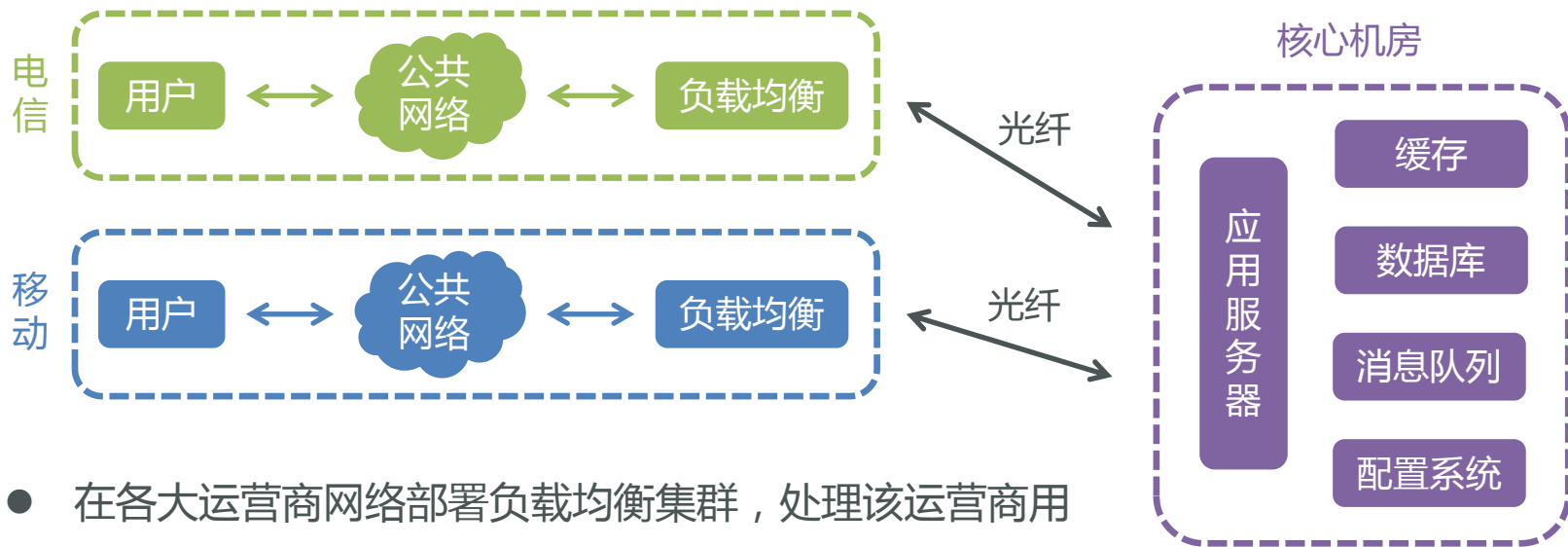
# 目录 CONTENTS

-  一 简单网络拓扑图
-  二 支付系统数据分级
-  三 本地内存与配置管理系统
-  四 分布式缓存
-  五 分布式数据库

# 运营商网络



## 简单网络拓扑图



- 在各大运营商网络部署负载均衡集群，处理该运营商用户的访问请求
- 各运营商的负载均衡集群到核心机房的应用服务器使用短距离光纤
- 应用服务器到各个系统或组件的实时调用均使用长连接

# 目录 CONTENTS

一 简单网络拓扑图

**二 支付系统数据分级**

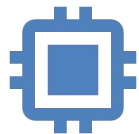
三 本地内存与配置管理系统

四 分布式缓存

五 分布式数据库

## / 不同存储系统的特点对比

本地内存



特点

- 无延时
- 容量小
- 不能持久化

缓存



特点

- 延时低
- 容量大
- 能持久化，但不使用

数据库



特点

- 延时高
- 容量大
- 能持久化

为追求高性能与高并发，尽可能使用本地内存，缓存次之，数据库最后



# 支付收银台数据分级

配置数据



特点

- 数据量小
- 需要持久化
- 读多写少



数据库 + 本地内存

用户绑卡数据



特点

- 数据量大
- 需要持久化
- 读多写少



数据库 + 分布式缓存

支付订单数据



特点

- 数据量大
- 需要持久化
- 写多



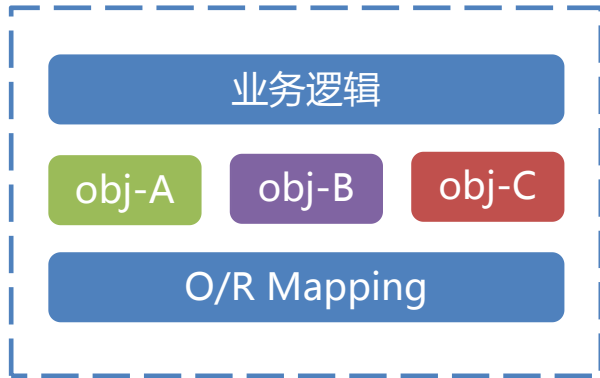
分布式数据库

# 目录 CONTENTS

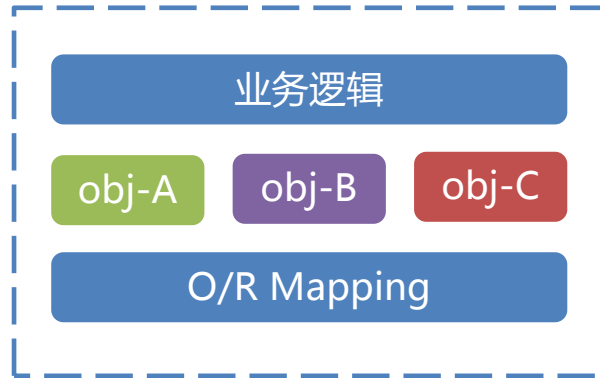
- 一 简单网络拓扑图
- 二 支付系统数据分级
- 三 本地内存与配置管理系统**
- 四 分布式缓存
- 五 分布式数据库

# 使用本地内存读配置数据

应用服务器



应用服务器



开发人员

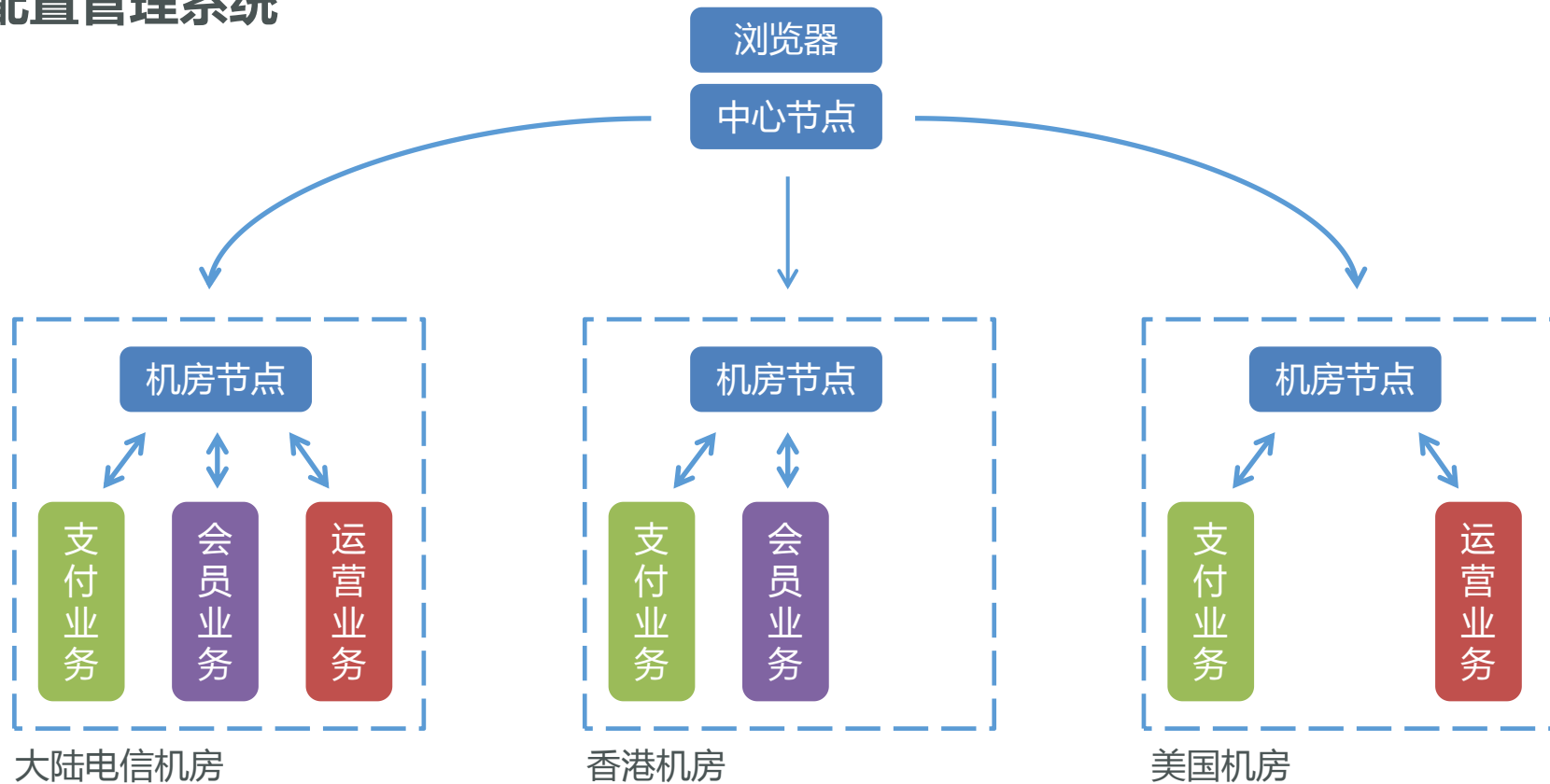


修改配置数据



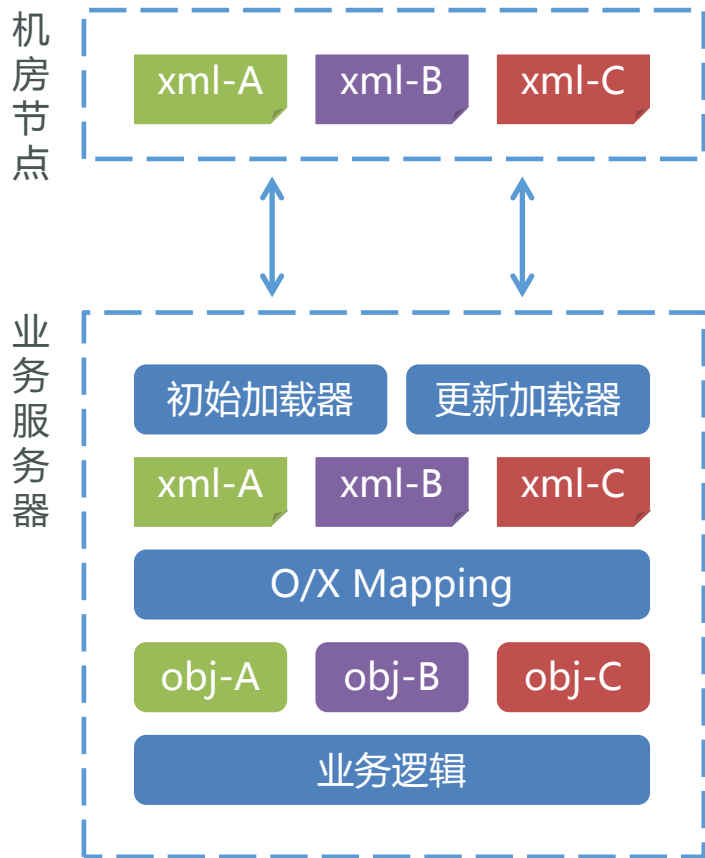
数据库

# 配置管理系统



- 配置数据使用xml格式
- 配置数据缓存在内存
- 支持实时更新
- 支持全球化部署

# 机房节点与业务服务器实时数据同步



更新加载器如何监听配置数据被更新？

更新加载器每隔3秒发起rpc调用，参数如下

xml-A, MD5(xml-A)

xml-B, MD5(xml-B)

xml-C, MD5(xml-C)

当所有数据的MD5值相等，说明配置数据没有变化

当某个或多个数据的MD5值不等，返回最新的配置数据并映射到对象中

# 目录 CONTENTS

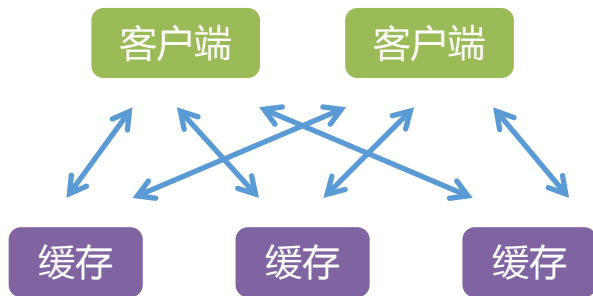
- 一 简单网络拓扑图
- 二 支付系统数据分级
- 三 本地内存与配置管理系统
- 四 分布式缓存**
- 五 分布式数据库

# 分布式缓存简介

什么是分布式缓存？

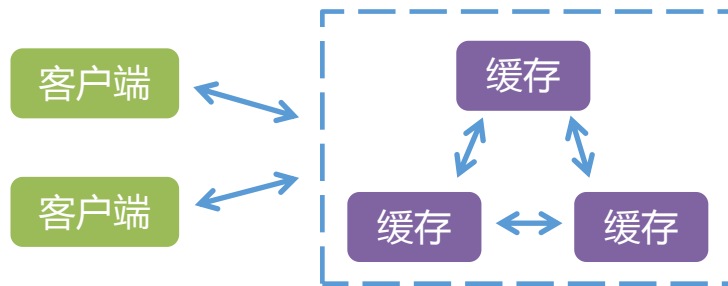
由多个缓存节点，通过一定算法，构成一整套高可用可扩展的缓存集群

客户端实现分布式缓存



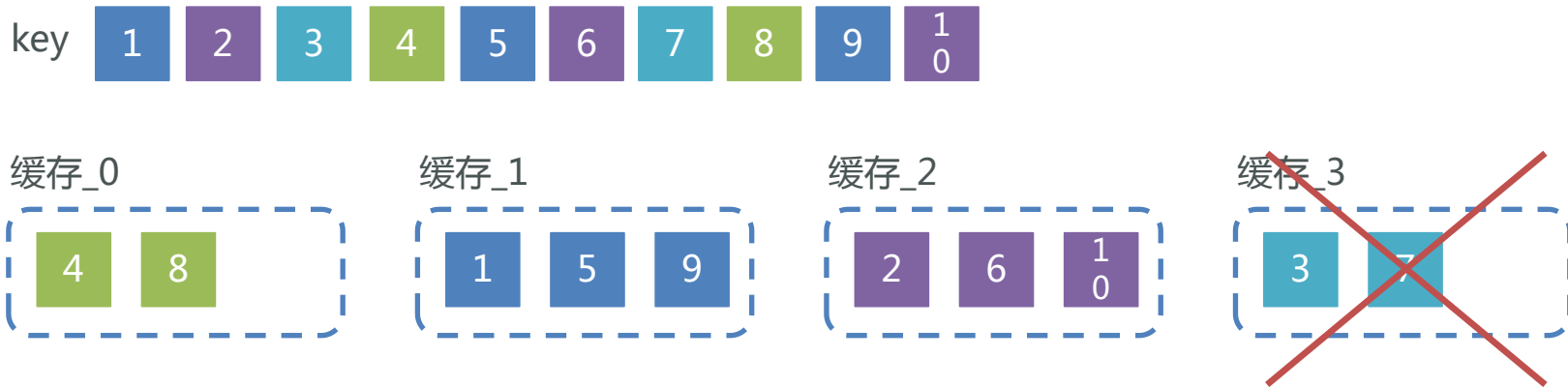
- 各缓存之间保持独立，不通讯
- 客户端根据key计算数据在哪个缓存中

服务端实现分布式缓存



- 各缓存之间保持通讯
- 客户端直接将数据放入缓存集群中

## 客户端实现分布式缓存 - 简单取模哈希



4个缓存正常运行时

- 数据按模4的方式将数据放入缓存中
- 按模4的方式从缓存中读取数据，缓存全部命中

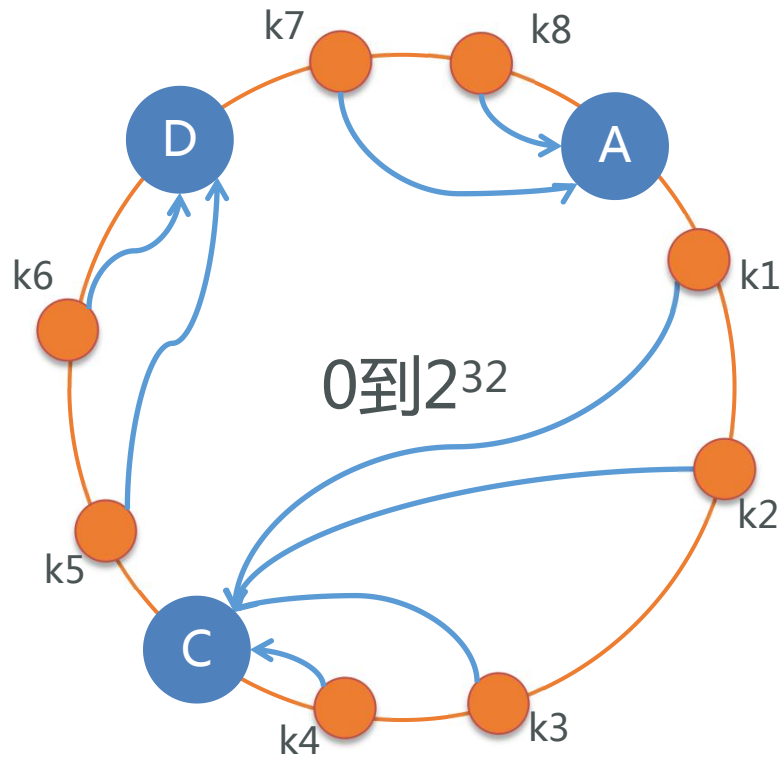
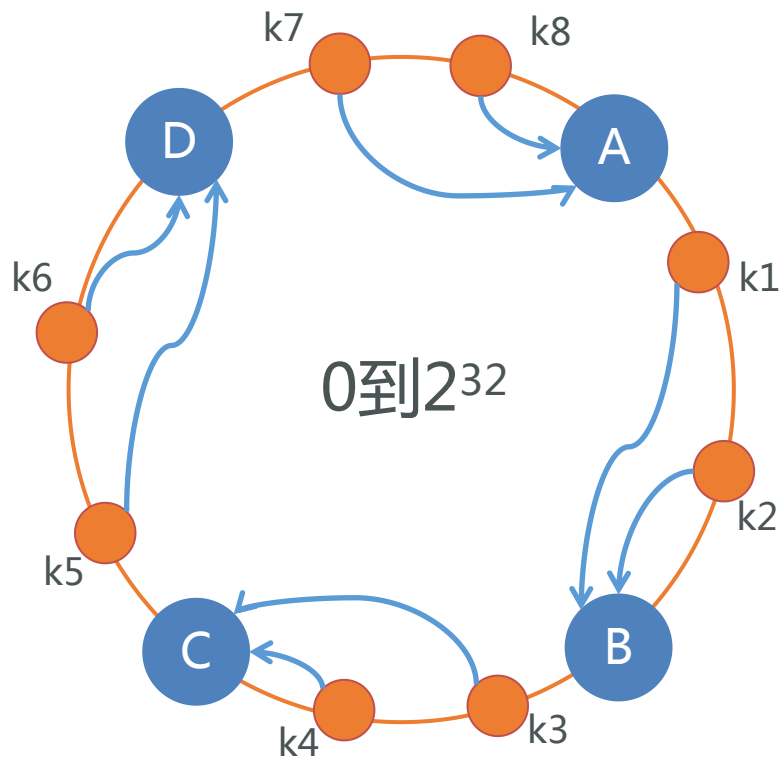
去掉缓存3虽然实际只丢失了20%的数据，但由于使用了简单取模哈希，客户端却认为丢失80%的数据！！

缓存\_3宕机，只有3个缓存提供服务

- 按模3的方式从缓存中读取数据，1和2缓存命中，3到10缓存丢失

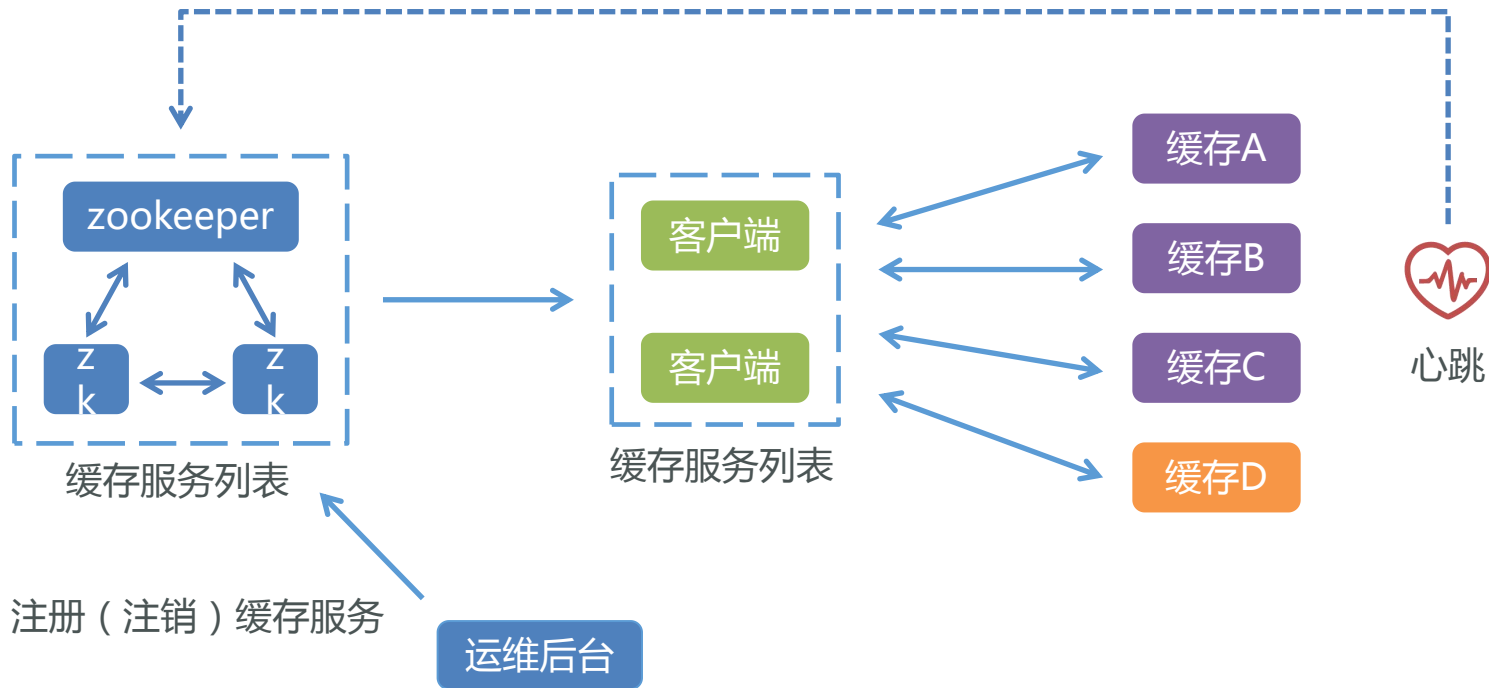


## 客户端实现分布式缓存 - 一致性哈希

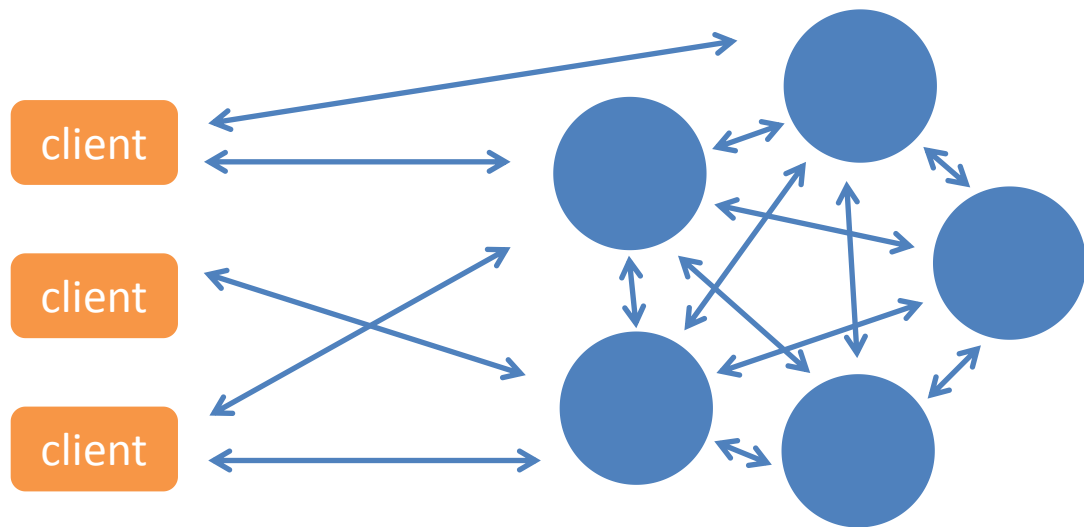


带虚拟节点的一致性哈希

# 一致性哈希+zkookeeper



## 服务端实现分布式缓存 - redis



[redis.io/topics/cluster-tutorial](https://redis.io/topics/cluster-tutorial)

[redis.io/topics/cluster-spec](https://redis.io/topics/cluster-spec)

# 目录 CONTENTS

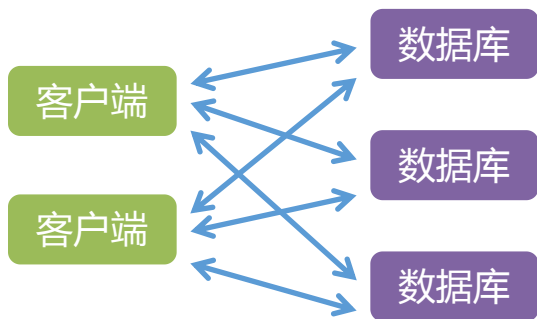
- 一 简单网络拓扑图
- 二 支付系统数据分级
- 三 本地内存与配置管理系统
- 四 分布式缓存
- 五 分布式数据库**

# 分布式数据库简介

什么是分布式数据库？

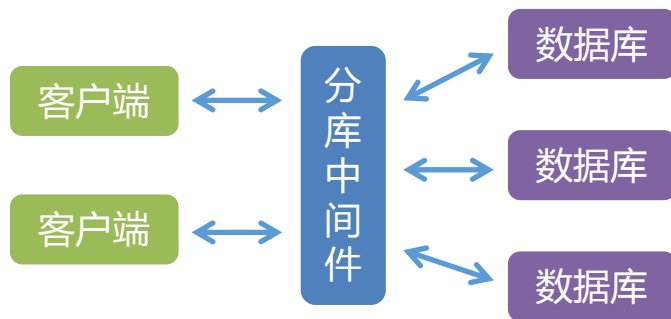
由多个数据库，通过一定算法，构成一整套高可用可扩展的数据库集群

客户端分库



- 分库算法在客户端实现
- 不引入新服务无性能损耗

使用分库中间件分库

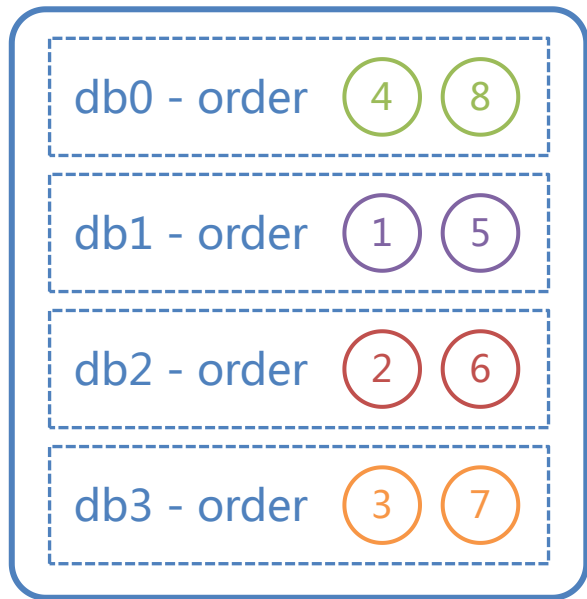


- 分库算法在分库中间件实现
- 引入中间件服务性有一定性能损耗

## 一个简单实用的分库算法

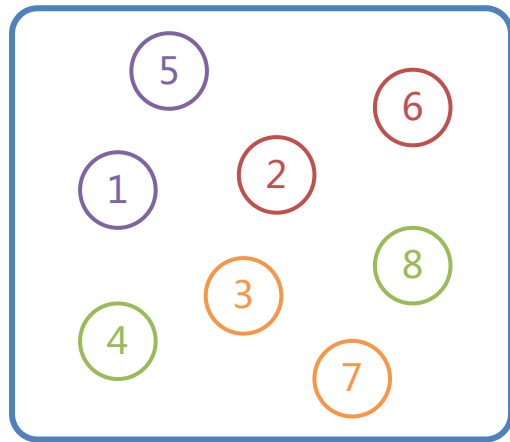
- 分库后数据分布均匀
- 扩容时数据迁移便捷

MySQL



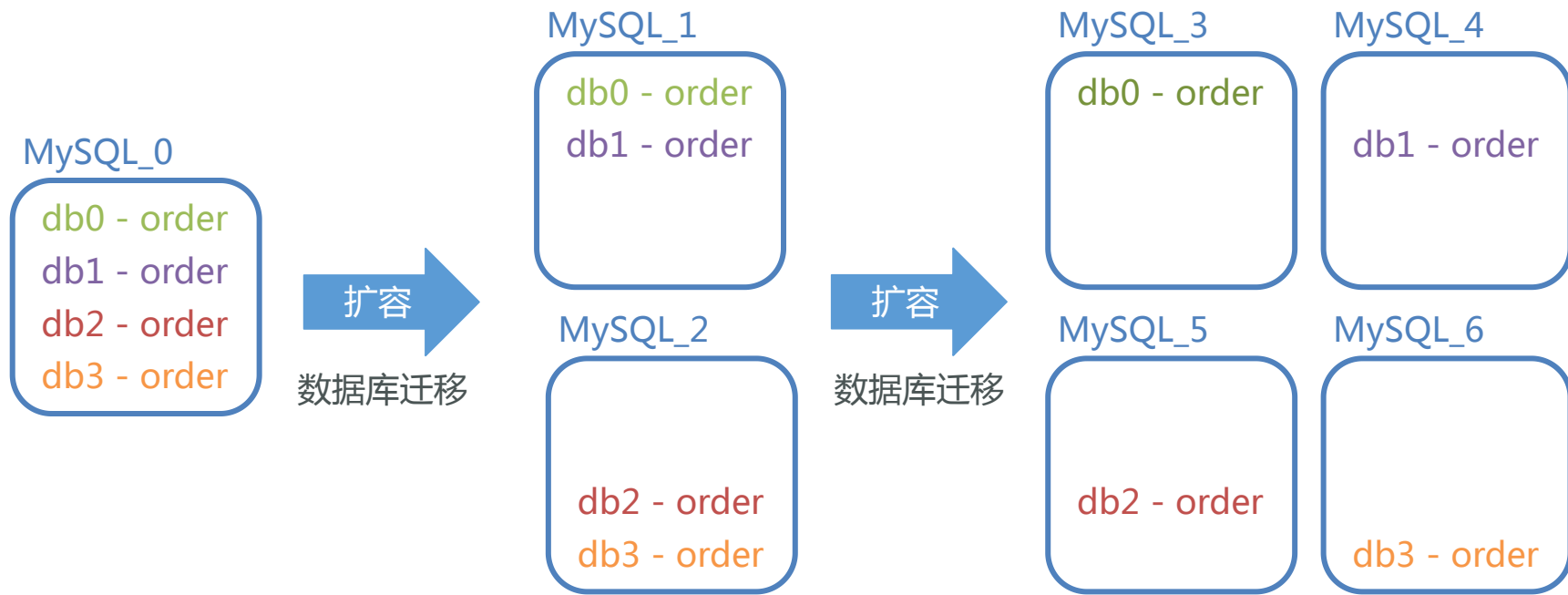
← 用户ID mod 4

用户ID



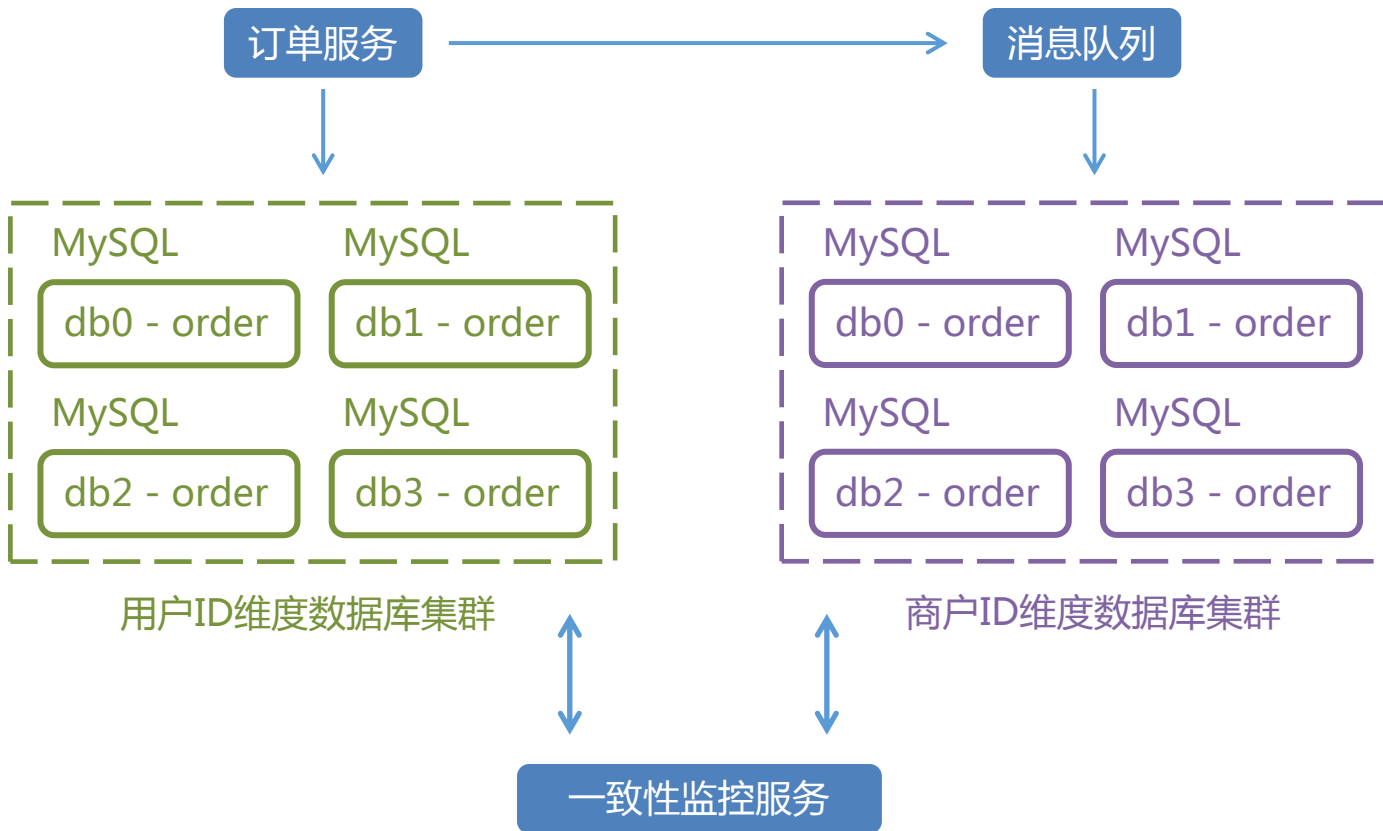
这里我们在单台MySQL上创建4个database，为以后扩容做准备

## 一个简单实用的分库算法之扩容



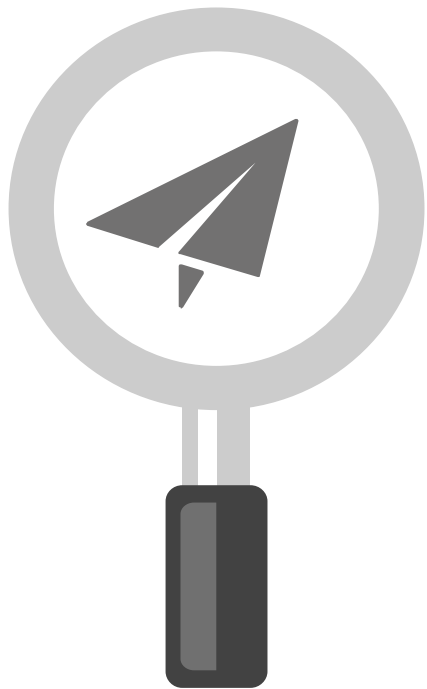
生产环境中一开始可以创建更多的database，以便扩容到更多数据库

# 数据冗余





## / 分库分表工具



**Mango**

[mango.jfaster.org](http://mango.jfaster.org)

**Mycat**

[www.mycat.org.cn](http://www.mycat.org.cn)

**Sharding JDBC**

[github.com/dangdangdotcom/sharding-jdbc](https://github.com/dangdangdotcom/sharding-jdbc)



# / Q&A

2016.8.12