

# 贝贝百亿级服务架构 及可用性保障实践

郁佳杰 @ 贝贝网

技术架构未来

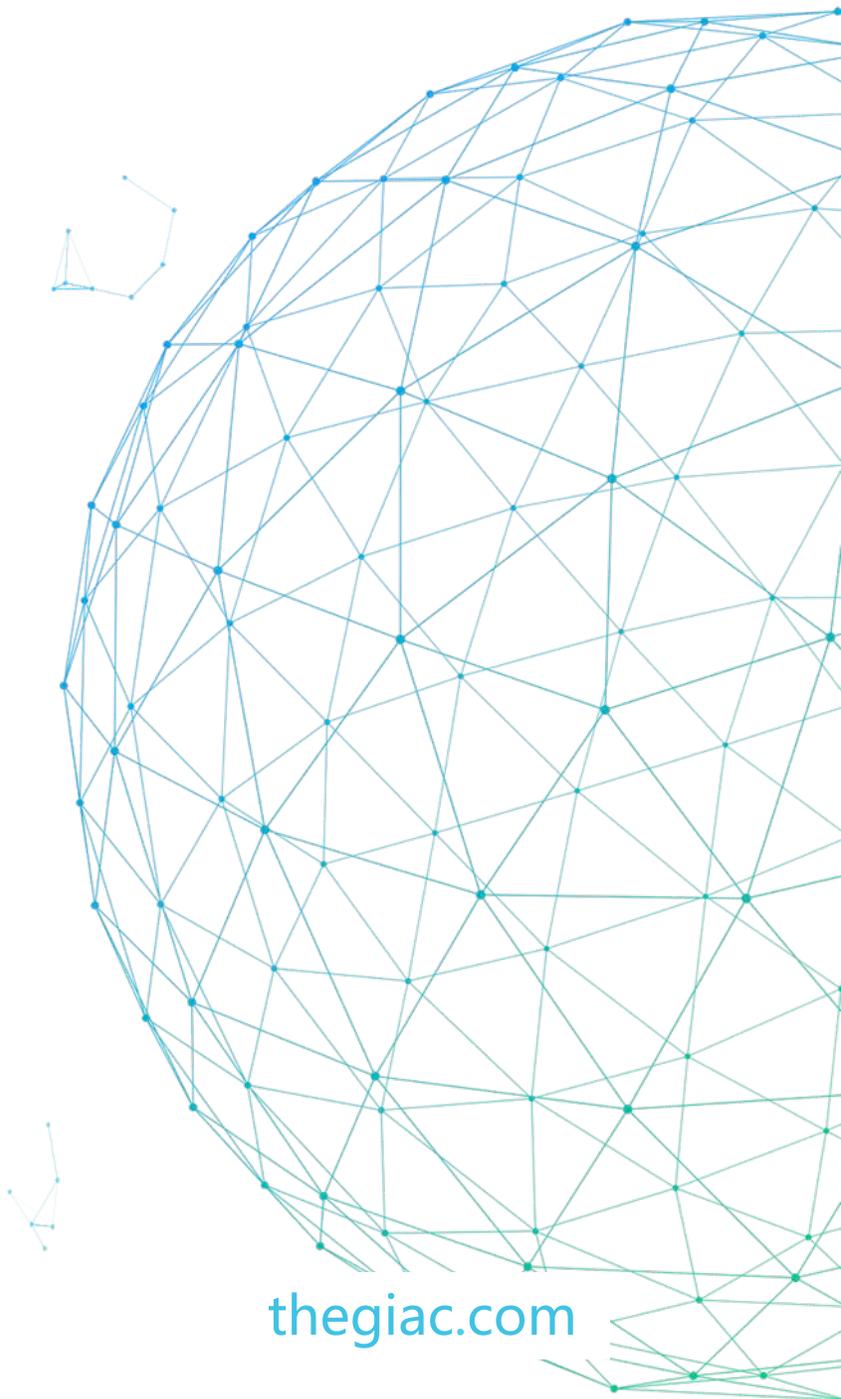


## 关于我

郁佳杰，贝贝联合创始人

- 2009年加入阿里，旺铺平台重构工作
- 2011年创业：米折网
- 2014年连续创业：贝贝网

**GIAC** | BEIJING  
Dec.12.16-17



[thegiacy.com](http://thegiacy.com)

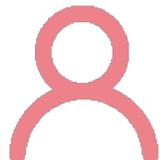
# 关于贝贝



5000万用户



5000+品牌



1300名员工



自营仓面积  
10万平米



千万级  
MAU

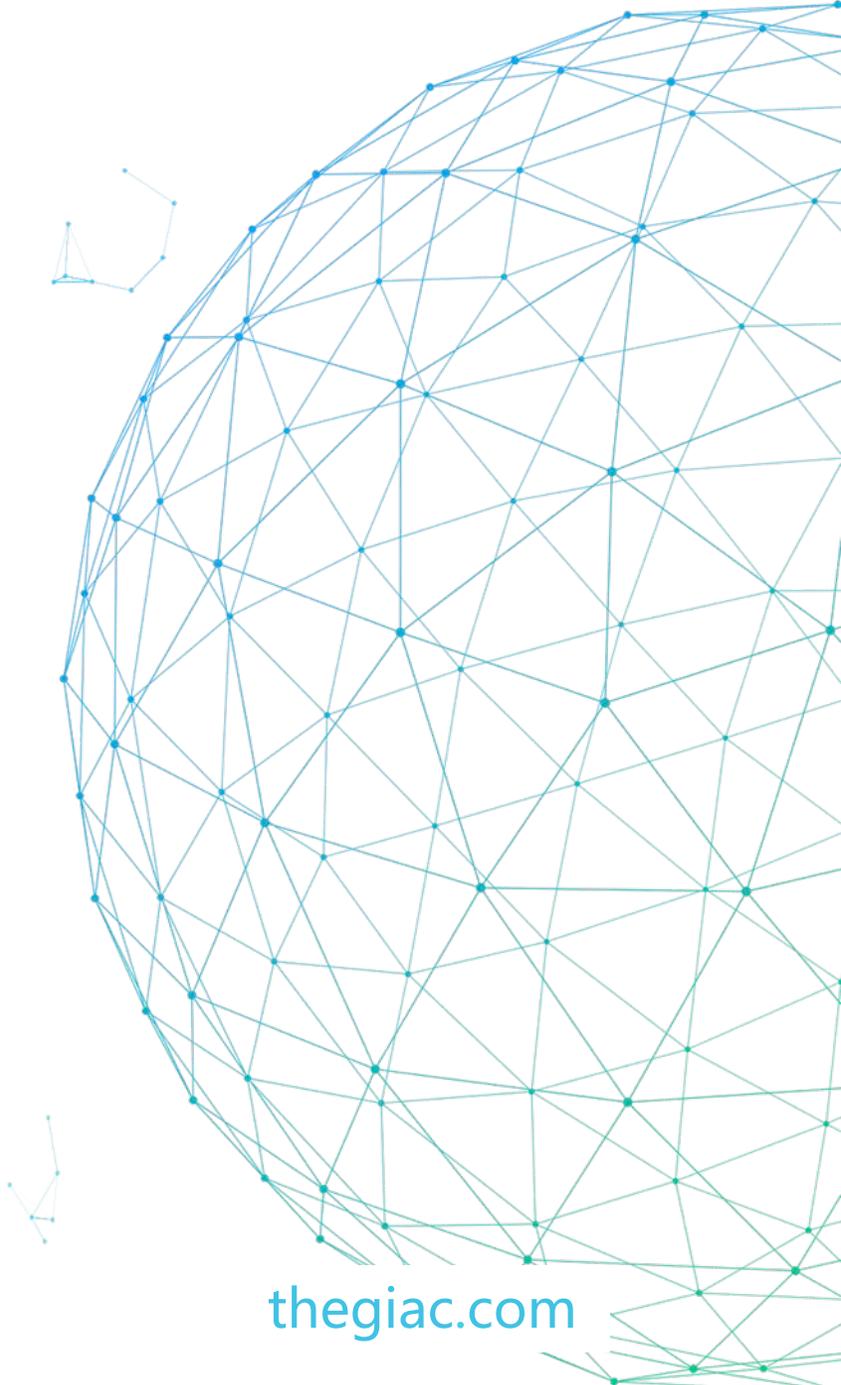


规模化  
连续盈利



**GIAC** | BEIJING  
Dec.12.16-17

[thegiacc.com](http://thegiacc.com)

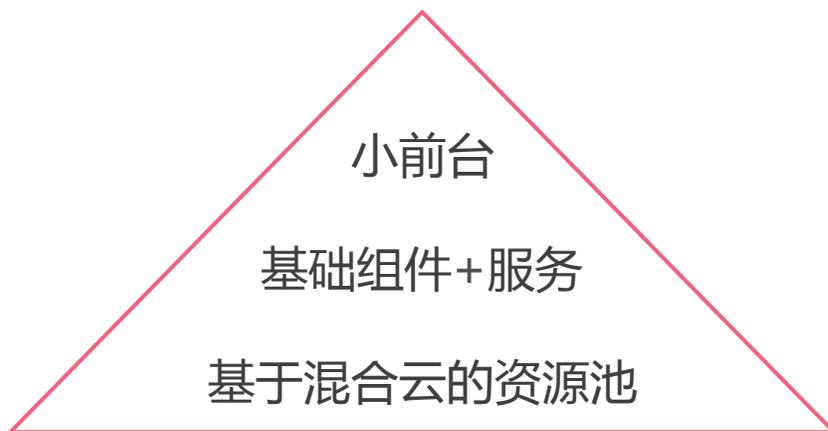


# 提纲

- 一、从0到1，初创期的技术选型
- 二、野蛮生长下的服务架构演进
- 三、双11技术保障实践

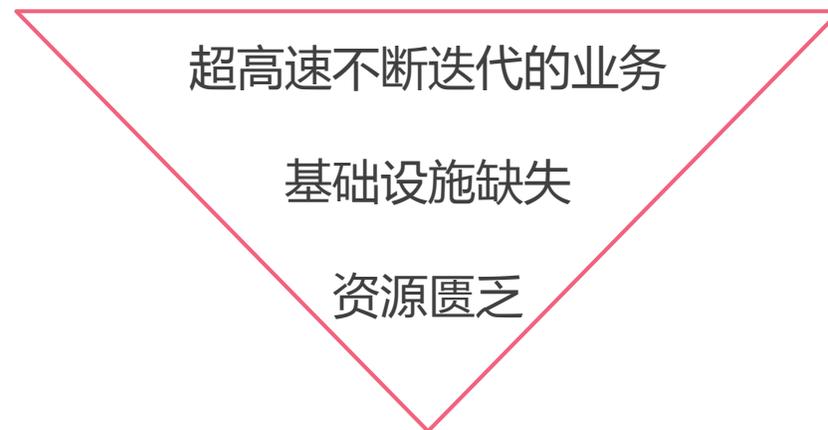
# 从0到1，初创期的技术选型

丰满的理想



VS

骨感的现实



# 怎么玩？

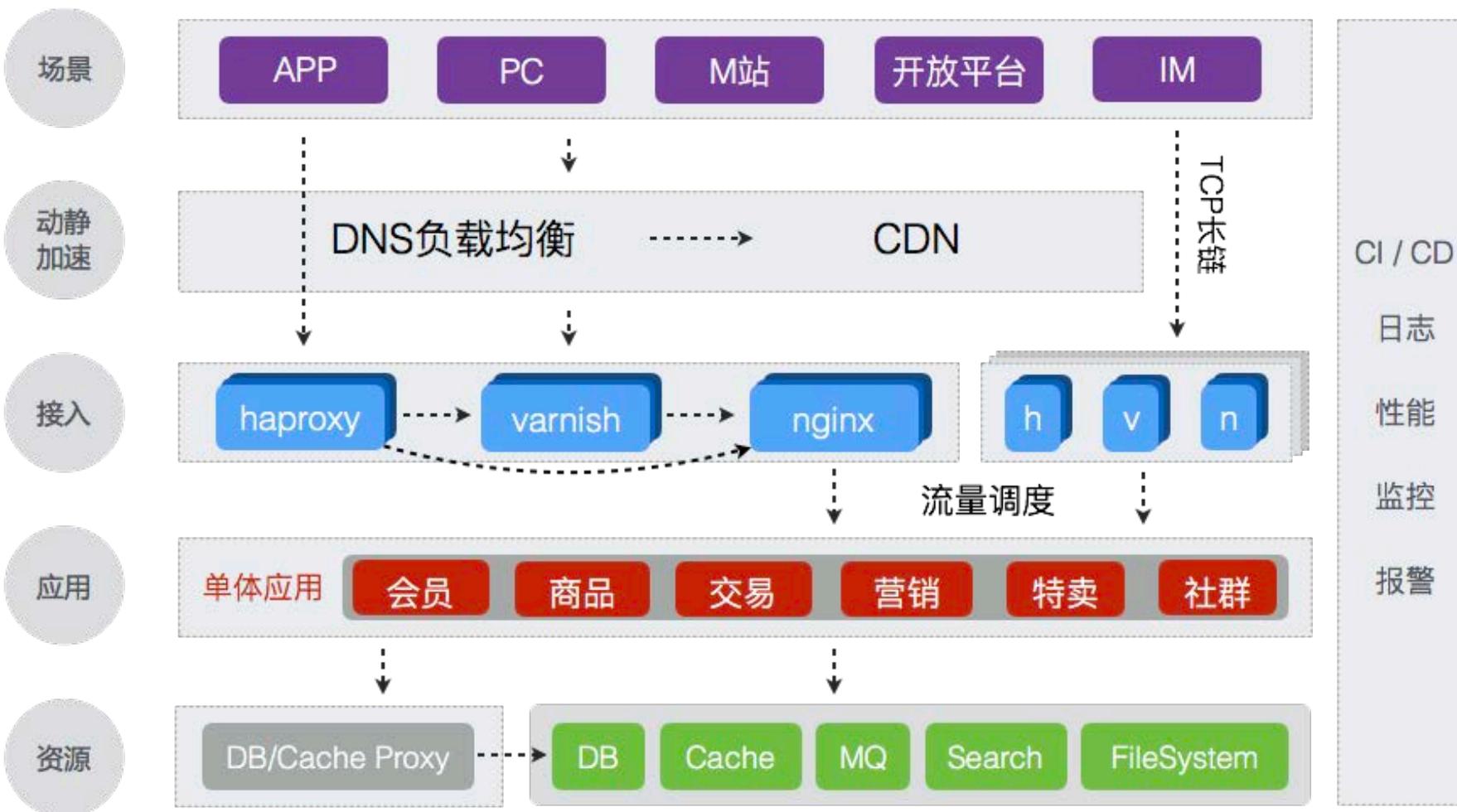
- 收起技术洁癖和猎奇心：
  - 最小代价交付用户价值
- 出来混，有些东西可以晚一点再还：
  - 重业务架构
  - 轻平台架构



业务的快速上线是第一要素  
简单粗暴，实现低成本管理

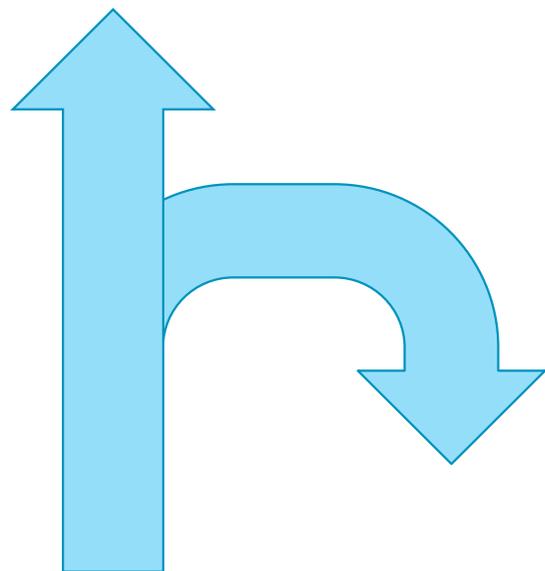
- “PHP是最好的语言”
- 做好动静分离，缓存就是银弹
- 专注业务逻辑，不做重资产

# 早期LNMP架构

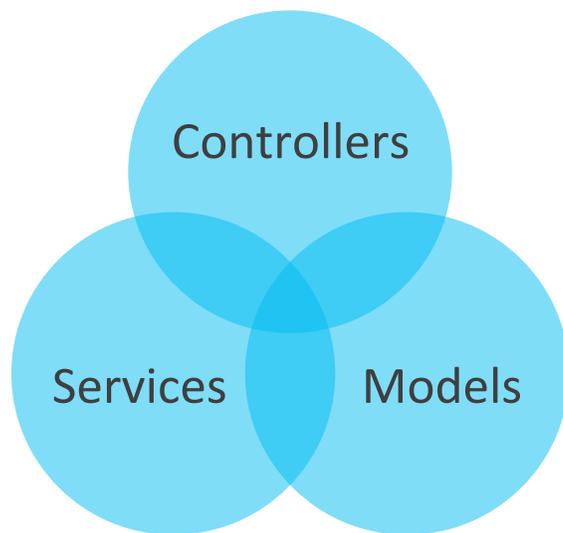


# 早期面临的问题

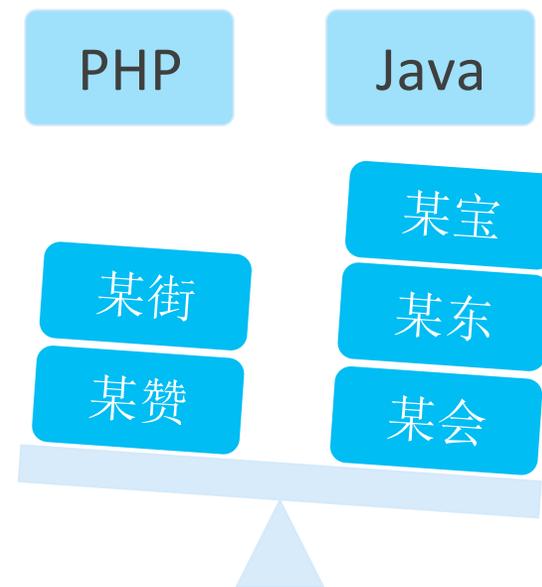
基础业务无法复用  
应用难拆分  
大工程



业务边界不清晰  
代码复杂  
难维护 ( 熵 )



非电商主流技术栈  
方案成熟度低  
人才短缺





# 提纲

- 一、从0到1，初创期的技术选型
- 二、野蛮生长下的服务架构演进  
接入层、服务化、数据层
- 三、双11技术保障实践

# 接入层

## HAProxy的痛点

- 基于配置文件，动态变更能力弱
- 操作风险高，故障恢复时间长
- 无法实现灵活的限流功能
- 监控不方便



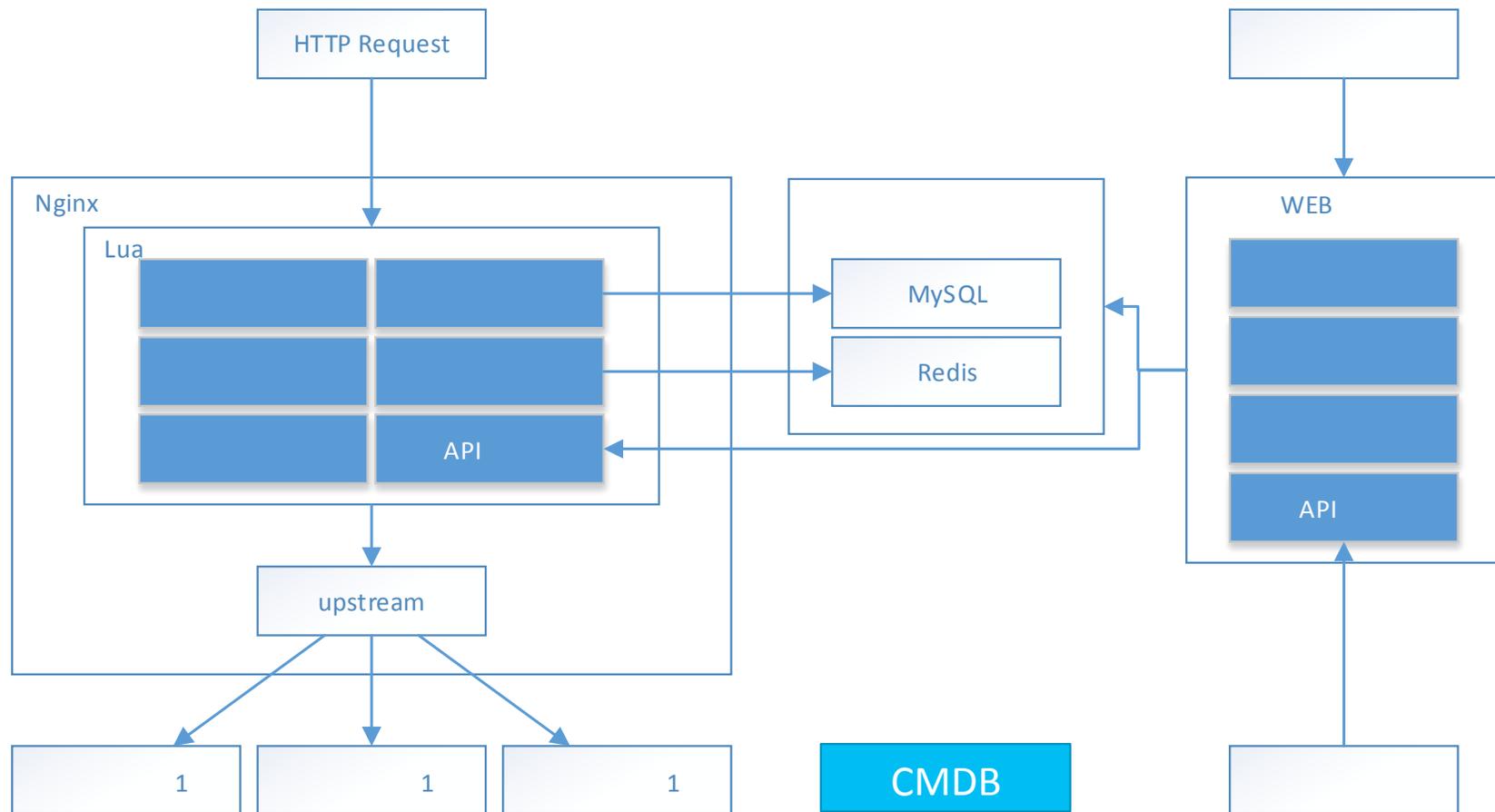
**Kongfu** based on OpenResty

配置管理：秒级全网生效

规则引擎：前缀/后缀匹配算法，平衡配置复杂度和灵活性

限流模块：简单的request、backend级别限流、以及复杂的业务规则限流

监控模块：请求量、错误率、规则执行等的实时统计





# 服务化

## 痛点一：异构的历史遗留系统

- 采用什么通信协议
- 如何进行统一的服务治理



## BSF PHP服务化扩展

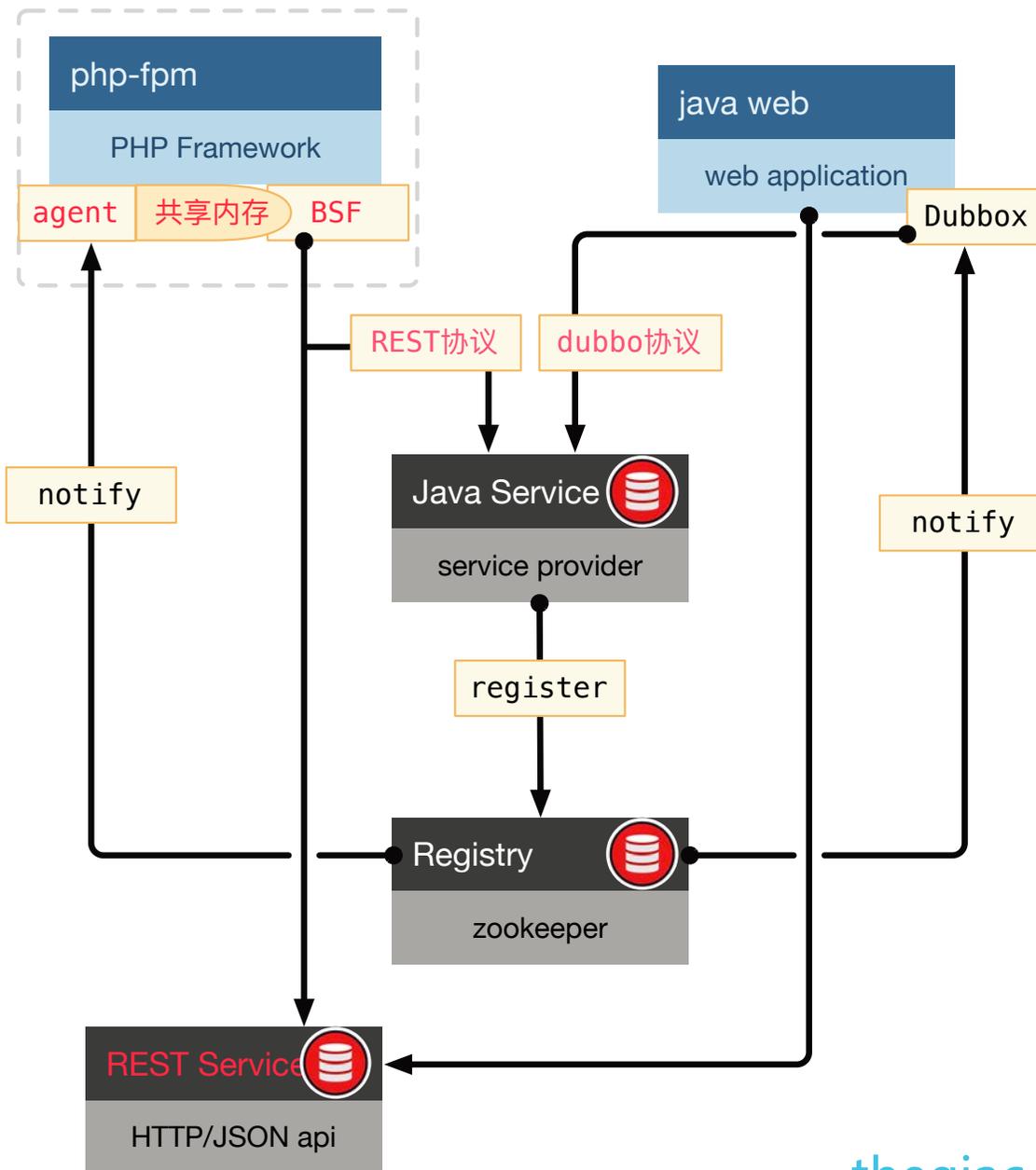
协议：REST & Hessian

服务注册与发现：zookeeper

负载均衡：预热问题

自动重试、容错

对应用透明



# 服务化

## 痛点二：复杂的单体应用

- 如何进行服务下沉和重构



# 服务化

## 痛点三：依赖错综复杂

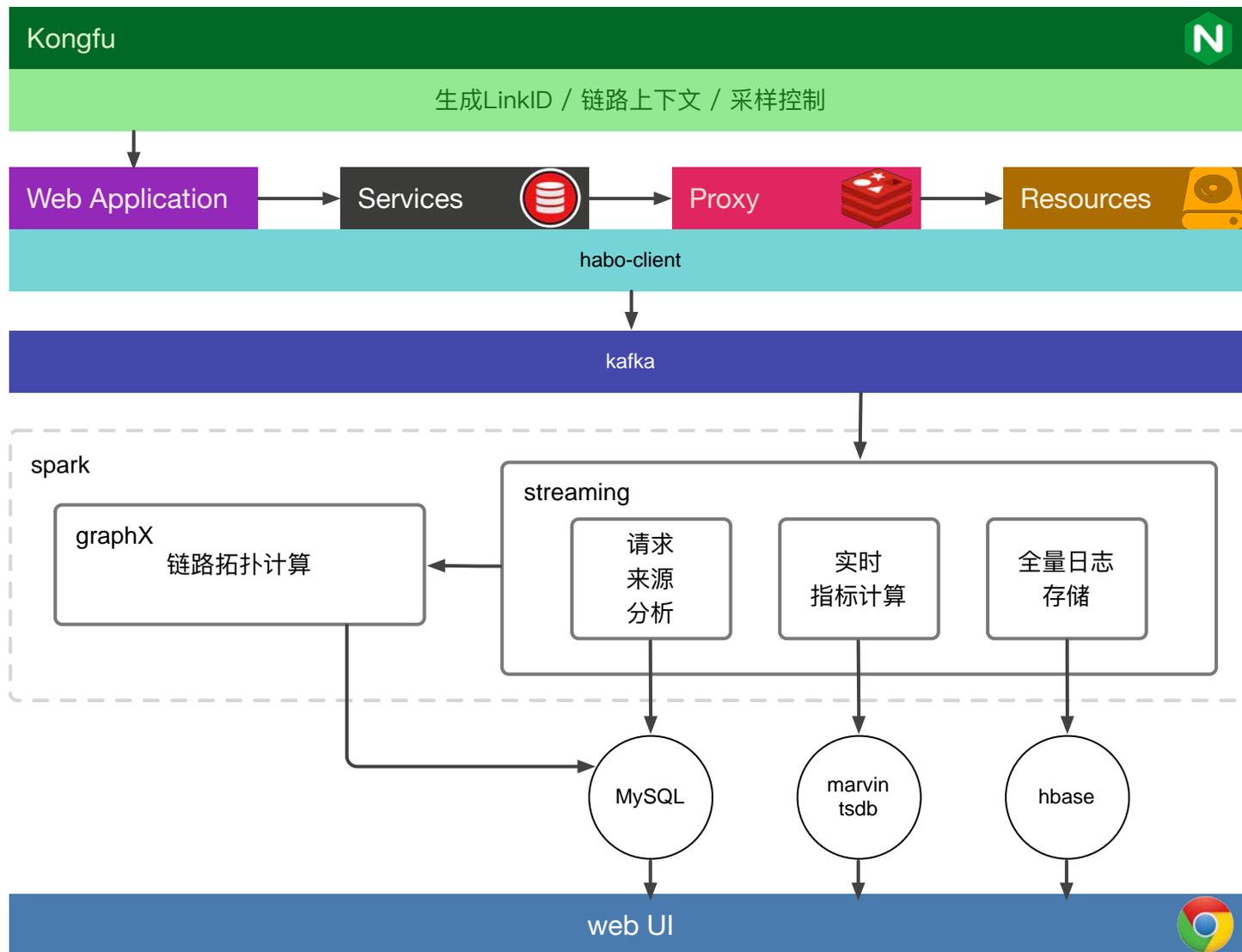
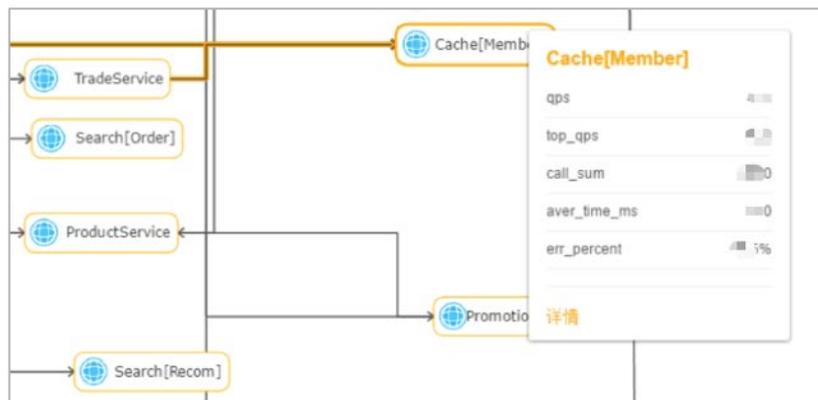
- 如何进行分布式调用链跟踪

应用级、组件级透明

采样控制

链路形态分析/实时指标

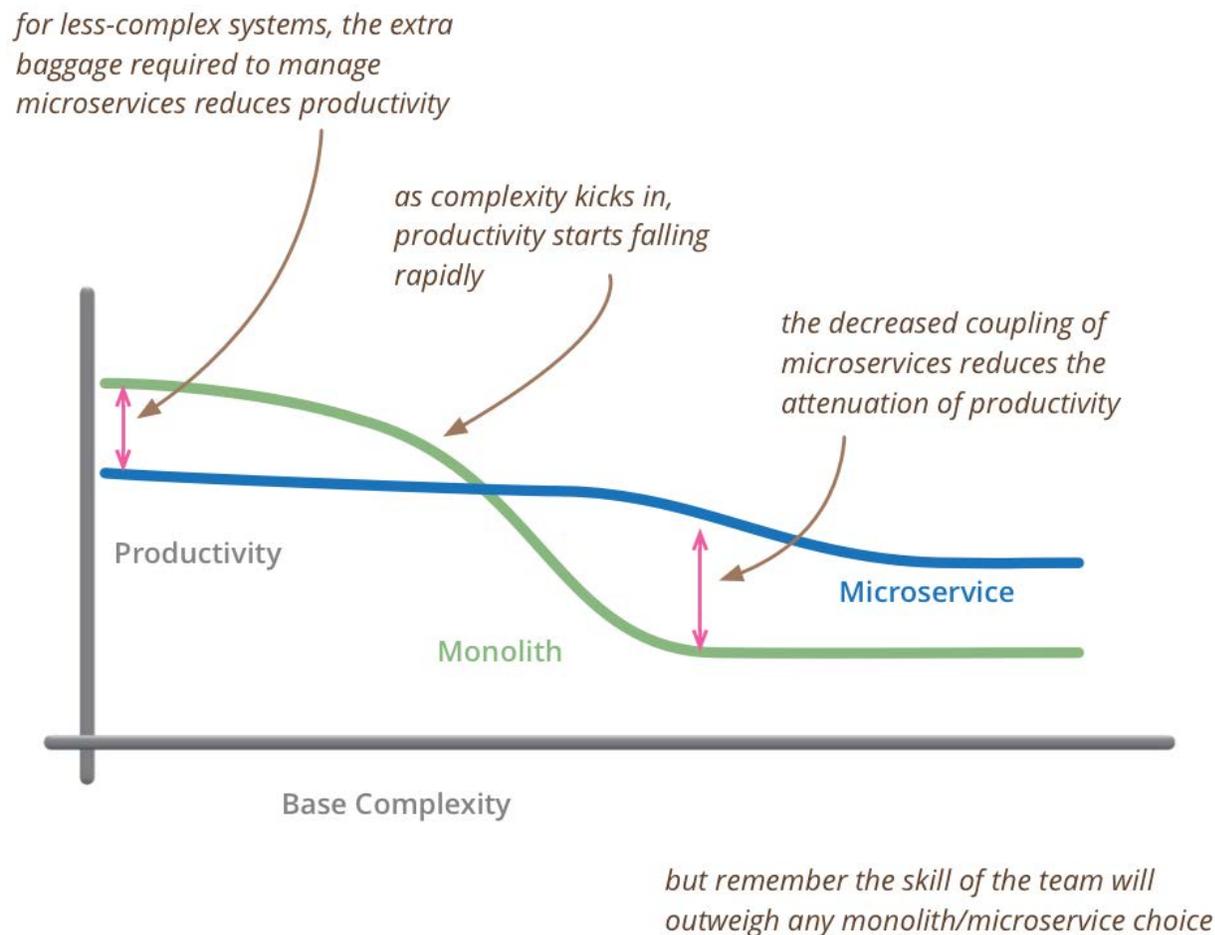
性能退化监控



# 服务化

## 痛点四：服务化不是纯技术问题

- 事关组织和团队的解耦
- 服务化是有额外成本的
- 组织和系统架构之间存在映射关系
  - 团队是分布式的
  - 互相之间存在依赖
  - 并行开发



# 数据层

- 单库，大量单表过十亿
- 存储空间不足
- TPS/QPS能力受限
- 无法水平扩展



**Cobarx** 分布式数据库中间件

透明的垂直/水平拆分能力

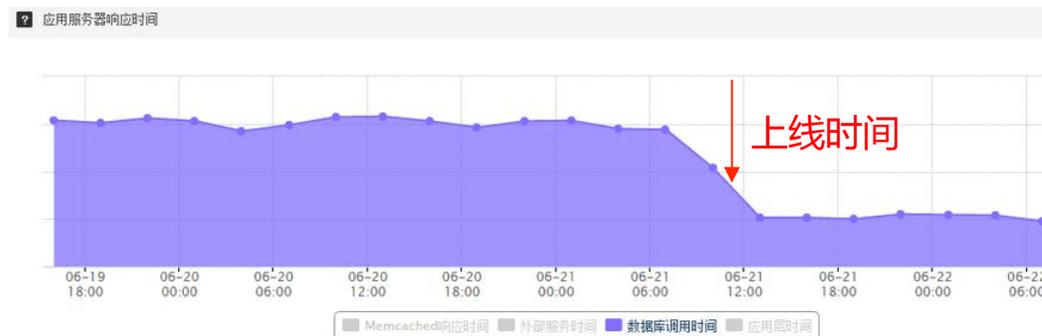
动态读写分离和从库LB

SQL黑/白名单

监控统计和HA强化

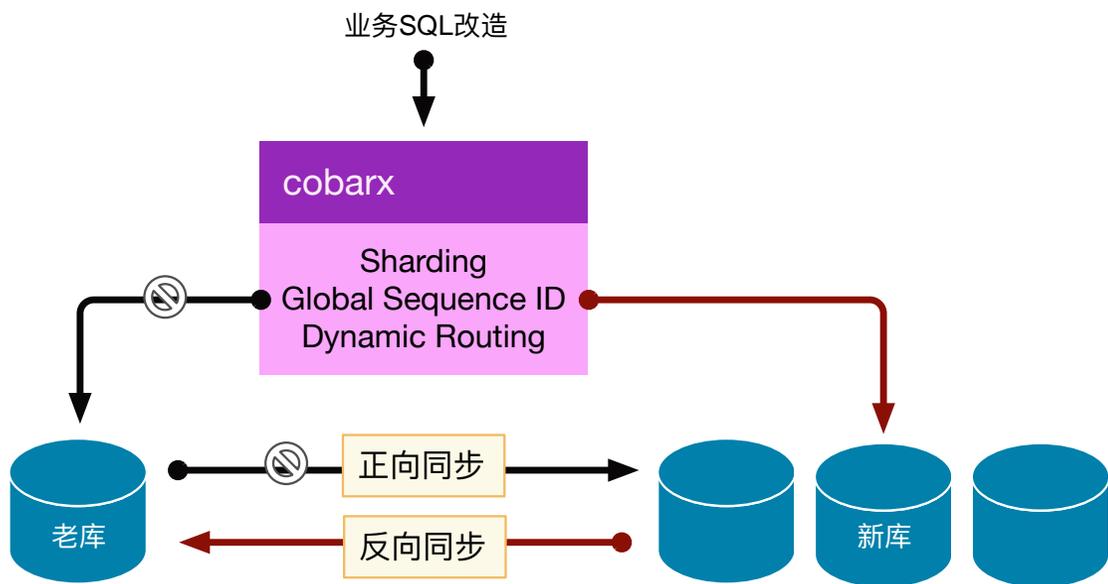
WEB化管理

特性	Cobar	TDDL	Mycat	Altas	Cobarx
透明proxy	Y	N	Y	Y	Y
读写分离	N	Y	Y	Y	Y
垂直分库	Y	Y	Y	Y	Y
水平分表	Y	Y	Y	Y	Y
全局ID	N	Y	N	Y	Y
动态管理	Y	Y	Y	N	Y
监控和管理	完善	丰富	完善	不完善	丰富



# 数据拆分

垂直分库**完全透明**，水平分表**少量业务改造**



1. 完成首次数据迁移
2. 正向同步期定期校验数据一致性
3. 切预发布只读流量验证业务
4. 老库只读，proxy进行秒级切换
5. 开启反向同步，确保随时可回滚
6. 核心库保持老库至少三个月可用

胆大心细，做好预案

# 小结与展望

- 协议优化（ 链接复用， TCP长链， 二进制 ）
- 异步、 并行
- 服务降级、 流控、 熔断



# 提纲

- 一、从0到1，初创期的技术选型
- 二、野蛮生长下的服务架构演进
- 三、双11技术保障实践

# 从商业目标到技术目标

公式1

请求量与交易额相关系数

平时大促全站请求量增长倍数

平时大促交易额增长倍数



双11全站请求量增长倍数

双11交易额增长倍数

公式2

DB TPS与交易额相关性

每交易DB写入次数



双11每秒交易数



双11DB每秒写入次数

# 从技术目标到技术方案

- 降级与解耦
- 秒杀场景优化
- 计数器场景优化
- 流控阈值计算
- 灾备容量及切换方案

# 降级与解耦

## 客户端降级

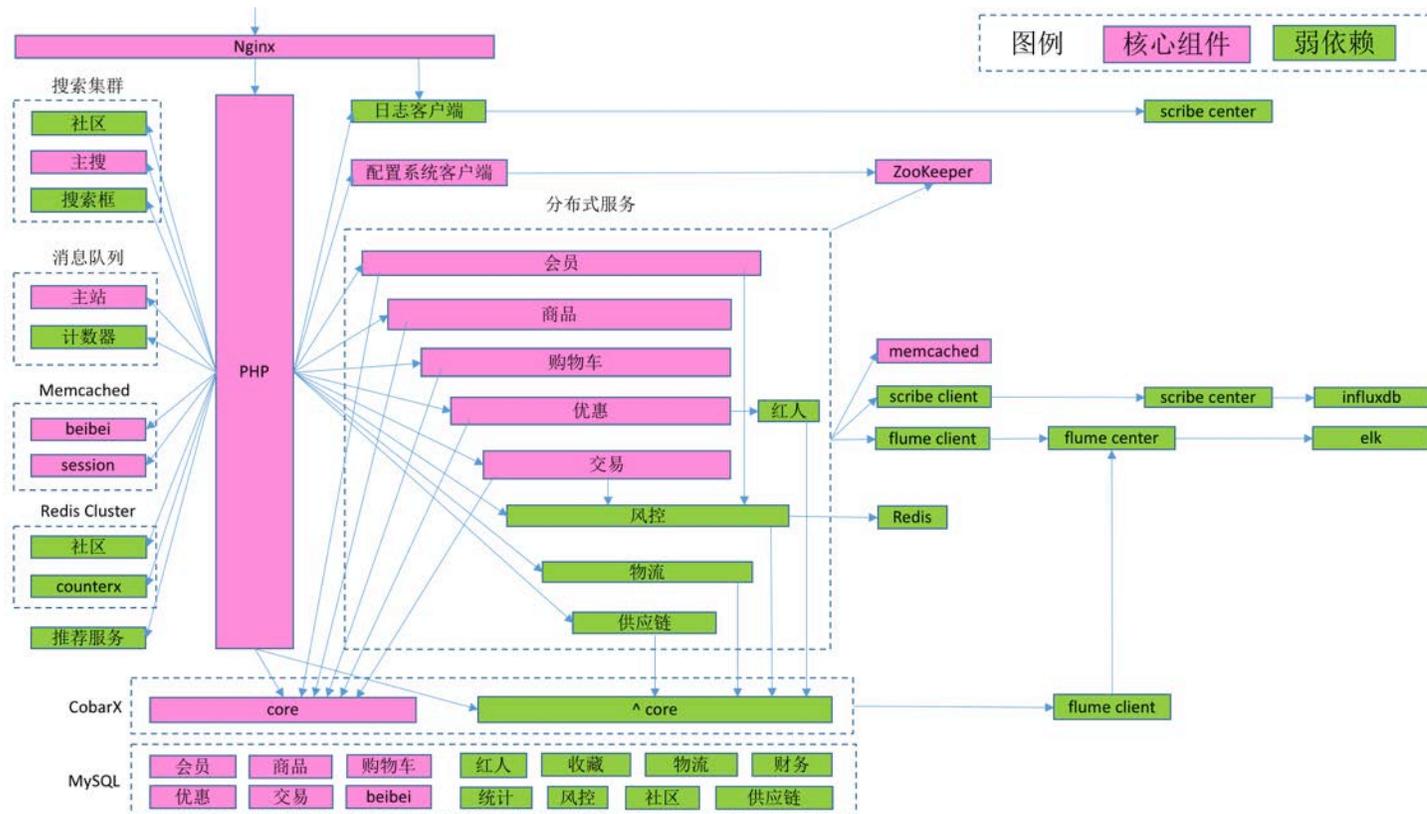
- 下发api blacklist
- 本地验证码削峰

## 服务端降级

- 缓存自动预热/延迟过期
- 业务流控+自动重试
- 弱依赖解除
- 随机拒绝服务

## 预先埋好开关

演练，演练，带上业务方演练！

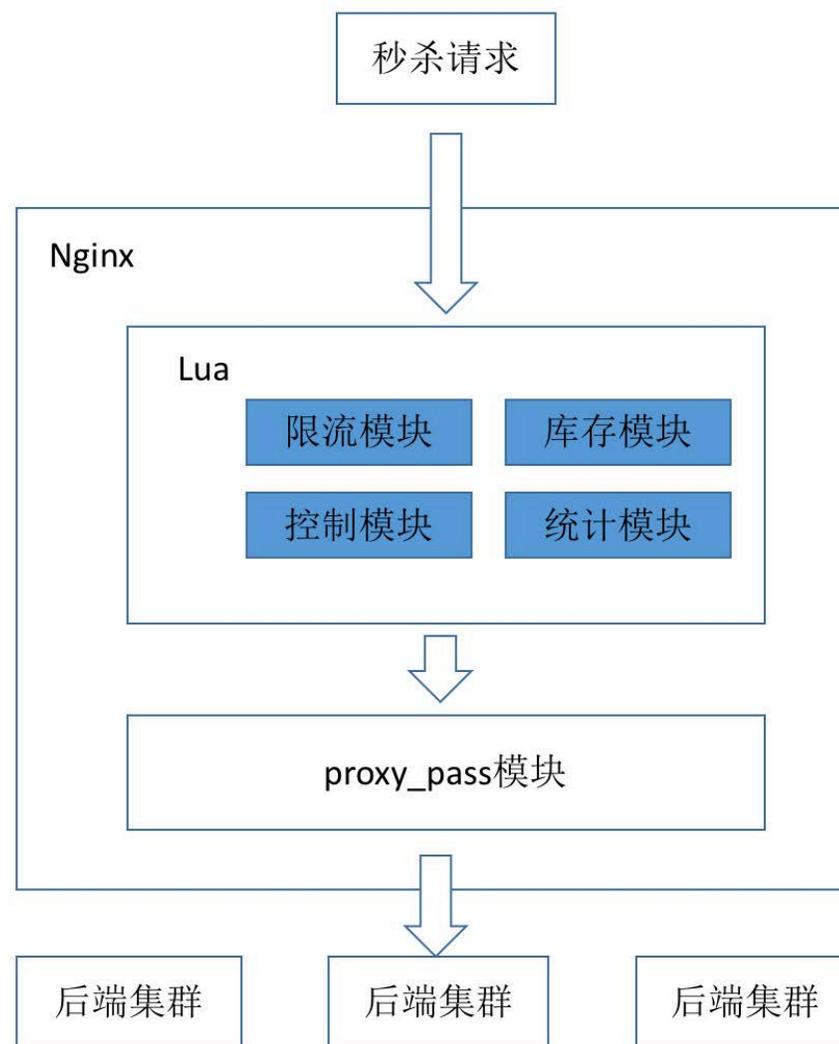
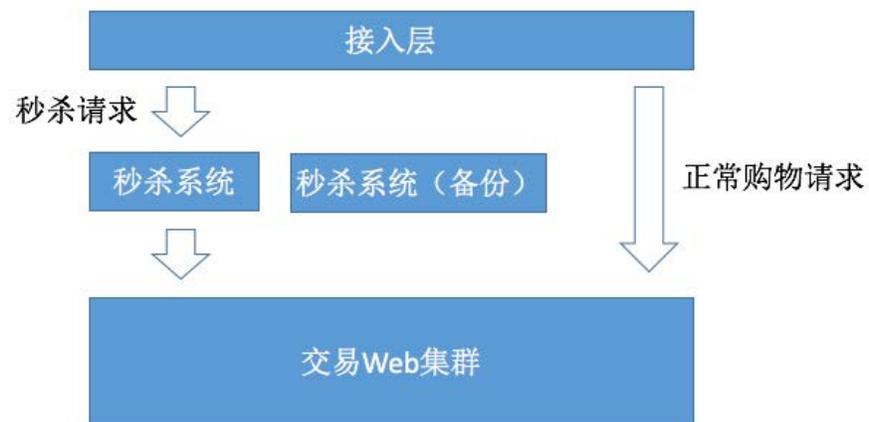


# 秒杀场景优化

秒杀的痛点：

- 瞬间高并发，冲击整个系统
- 占用有限的服务器资源，导致正常商品无法购买
- 不管怎么样，用户满意度都很低

解决思路：隔离、异步吞吐、特殊路由



# 计数器场景优化

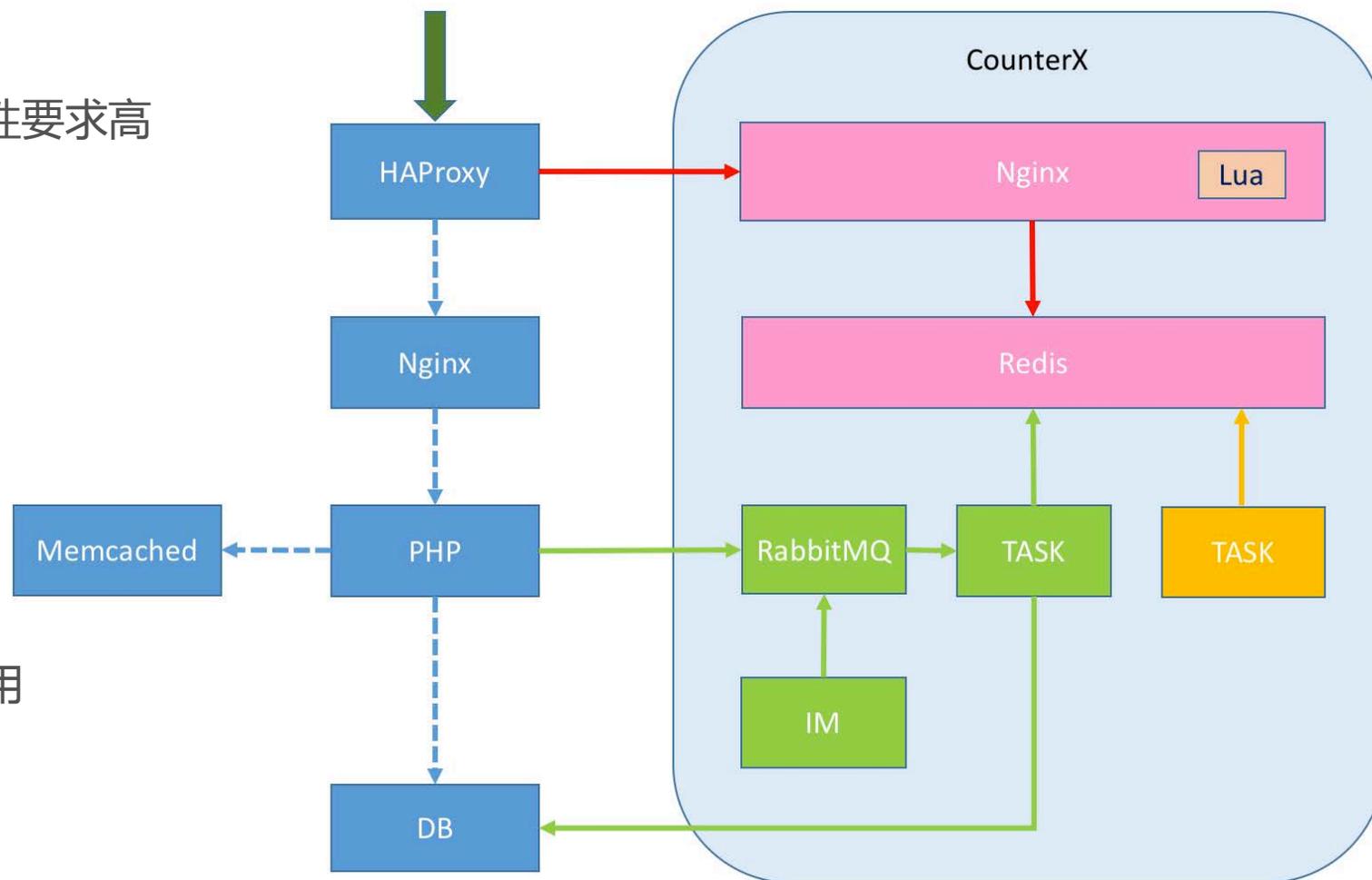
TOP接口，依赖复杂，实时性要求高

解决思路：

- 异步消息解耦
- 轻量级，独立系统
- 自修复，最终一致

效果：

- 单实例支撑每天百亿次调用
- RT降低一个数量级



# 接入层阈值计算

BASE理论：

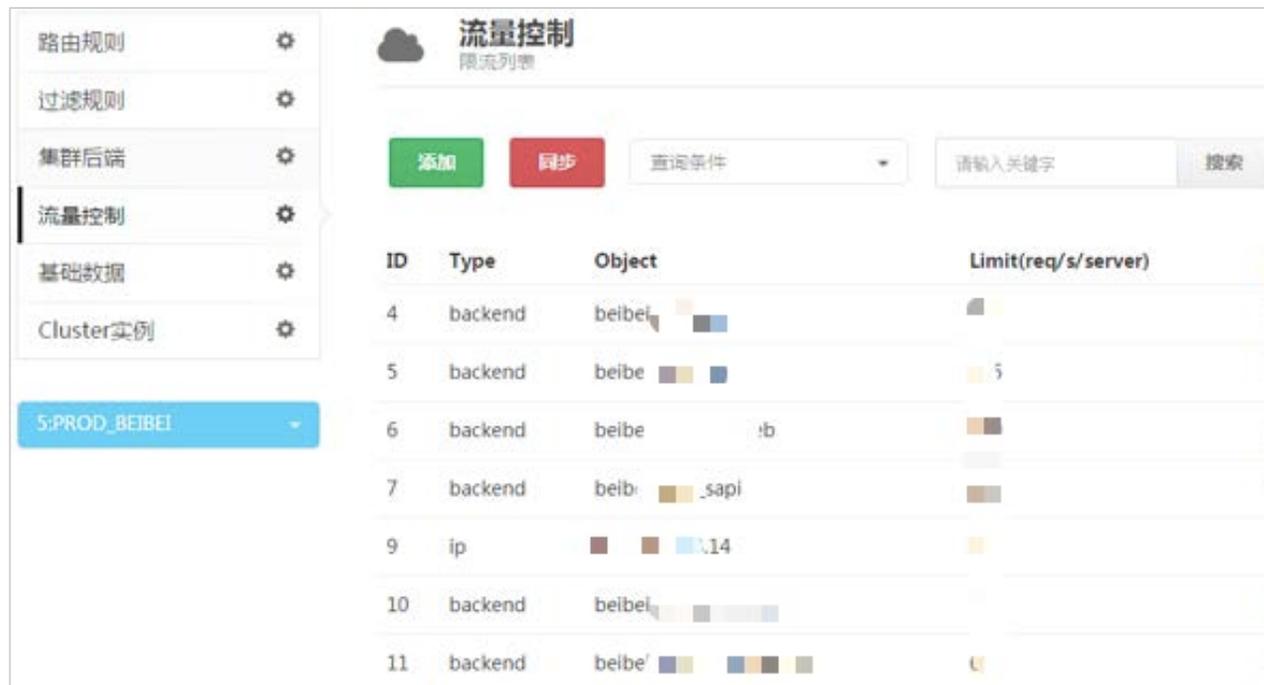
Basically Available (基本可用)

Soft state (软状态)

Eventually consistent (最终一致性)

在有限的资源内损失部分可用性

保护下游系统、保障核心链路QoS



The screenshot shows a web interface for traffic control. On the left is a navigation menu with items: 路由规则, 过滤规则, 集群后端, 流量控制 (selected), 基础数据, and Cluster实例. Below the menu is a dropdown menu showing 'S:PROD\_BEIBEI'. The main area is titled '流量控制 限流列表' and contains a table with columns: ID, Type, Object, and Limit(req/s/server). The table lists 11 items with various types and objects.

ID	Type	Object	Limit(req/s/server)
4	backend	beibel	
5	backend	beibe	5
6	backend	beibe	
7	backend	beibe_sapi	
9	ip	.14	
10	backend	beibel	
11	backend	beibe	



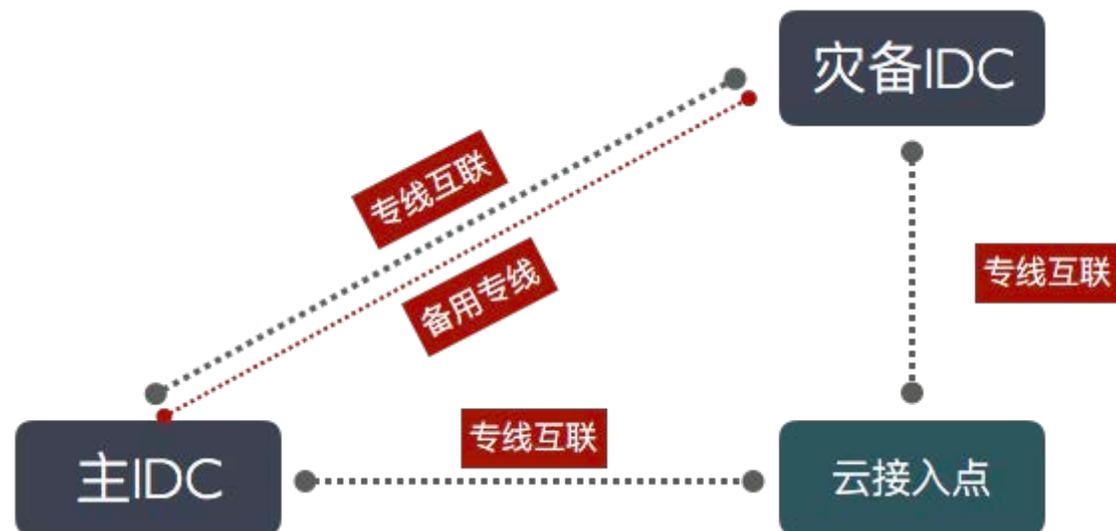
# 灾备容量及切换方案

灾备机房硬件不对等，遇到灾难怎么解？

容量底线：能支撑除去秒杀时刻外的流量

决策和经验：

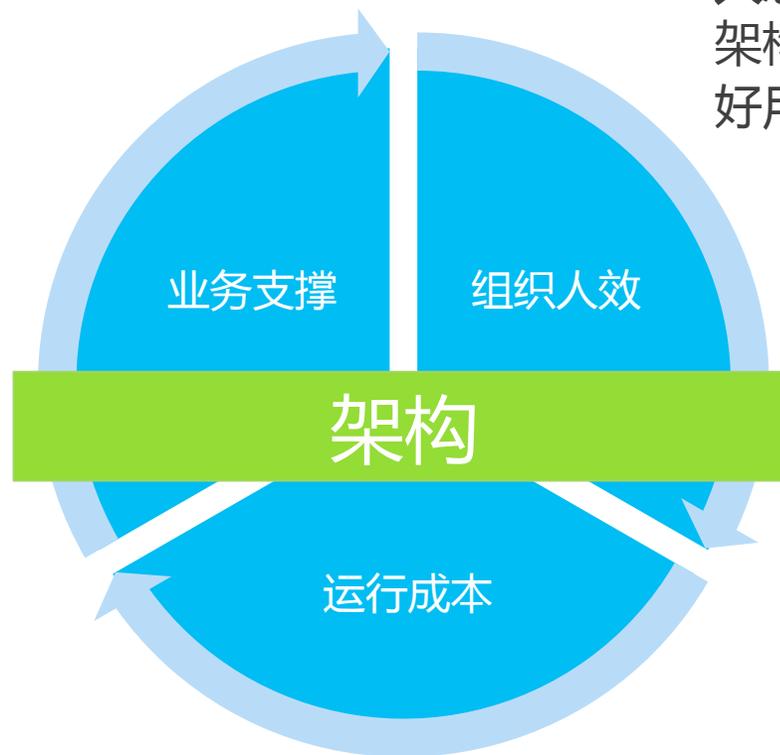
- 多机房同步应用发布及配置管理
- 日常调度少量流量跑只读场景
- 只部署核心应用，节省硬件成本
- 弱依赖通过配置中心默认解除
- 部分计算资源通过混合云弹性伸缩
- 接入层限流，保障最核心的业务
- 制定预案，定期真实切换演练



# 总结

架构价值从哪里来，到哪里去？

- 业务交付效率和质量
- 研发组织和人效
- 运行成本



人总会犯错  
架构解决的还有人和组织的问题  
好用并且用不烂的才是好架构

没有万金油  
架构是演进出来的  
适合自己的才是好架构

# Q&A