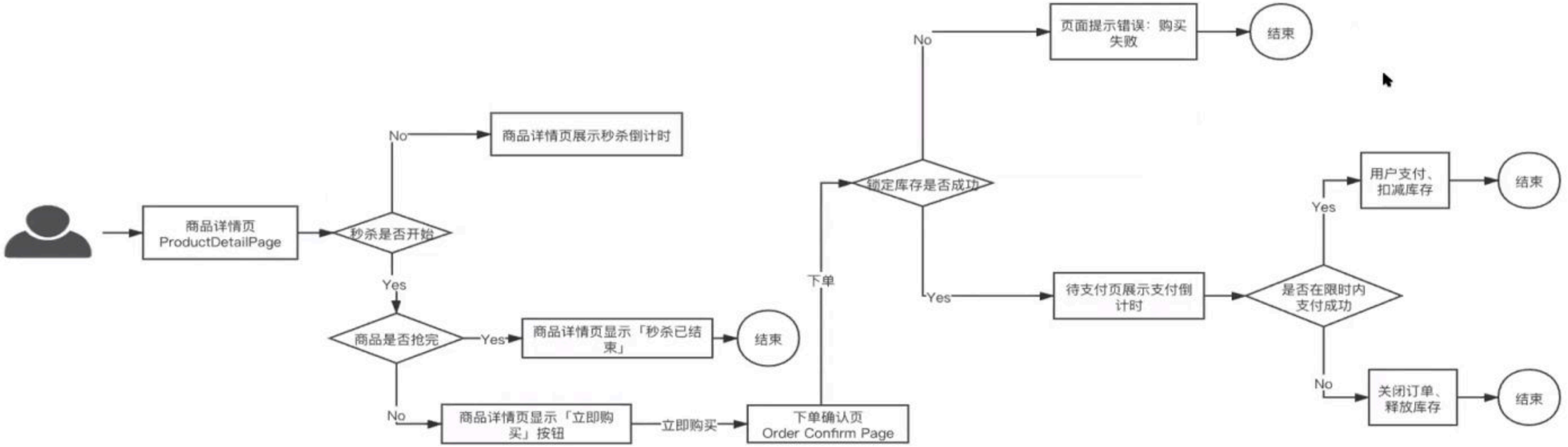
秒杀系统

Outline

- 业务需求
- 架构原则
- 技术方案
  - 动静态分离
  - 二八原则：热点数据
  - 流量削峰
  - 减库存
  - 高可用
  - 其它业务需求的实现
- 方案对比

电商平台

- 用户功能
  - 登录注册
  - 商品管理
  - 商品展示
  - 秒杀
- 基础模块
  - 用户
  - 商品
  - 订单
  - 库存
  - 支付


商品详情页
ProductDetailPage
秒杀是否开始
No
商品详情页展示秒杀倒计时
Yes
商品是否抢完
Yes
商品详情页显示「秒杀已结束」
结束
No
商品详情页显示「立即购买」按钮
立即购买
下单确认页
Order Confirm Page
下单
锁定库存是否成功
No
页面提示错误：购买失败
结束
Yes
待支付页展示支付倒计时
是否在限时内支付成功
Yes
用户支付、扣减库存
结束
No
关闭订单、释放库存
结束

秒杀流程

利益相关者

干系人	问题分类	业务出现的问题	设计要求
用户	体验较差	秒杀开始，系统瞬间承受平时数十倍甚至上百倍的流量，直接宕掉	高性能
		用户下单后却付不了款，显示商品已经被其他人买走了	一致性
商家	商品超卖	100 件商品，却出现 200 人下单成功，成功下单买到商品的人数远远超过活动商品数量的上限	一致性
	资金受损	竞争对手通过恶意下单的方式将活动商品全部下单，导致库存清零，商家无法正常售卖	高可用
		秒杀器猖獗，黄牛通过秒杀器扫货，商家无法达到营销目的	高可用
平台运维人员	风险不可控	系统的其它与秒杀活动不相关的模块变得异常缓慢，业务影响面扩散	高可用
	拖垮网站	在线人数创新高，核心链路涉及的上下游服务从前到后都在告警	高性能
		库存只有一份，所有请求集中读写同一个数据，DB 出现单点	高性能

架构原则

- 4要1不要
  - 数据尽量少
  - 请求数要尽量少
  - 路径要尽量短
  - 依赖要尽量少
  - 不要有单点

动静态分离

动静态分离

- 静态数据缓存到离用户近的地方
  - 重新设计秒杀商品页面，不使用网站原来的商品详细页面，页面内容静态化，用户请求不需要经过应用服务，直接访问Cache（用户浏览器里、CDN上或者在服务端）
- 直接缓存HTTP连接
- 谁来缓存数据
  - Java层
  - Web服务器层

动态数据分离

- URL唯一化
  - 缓存的key
- 分离浏览者相关的因素
- 分离时间因素
- 异步化低于因素
- 去掉Cookie
  - Web服务器Varnish可以通过命令去掉Cookie。在缓存的静态数据中不含有Cookie

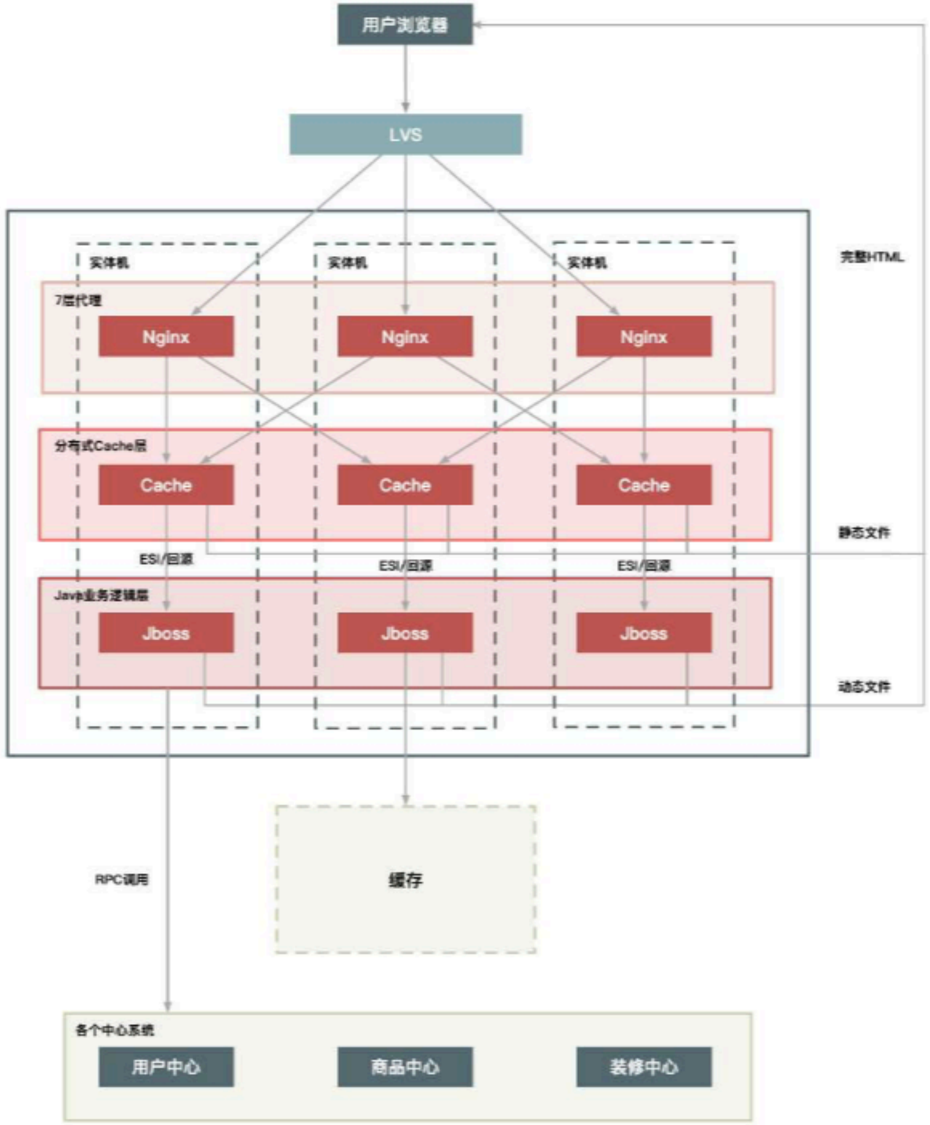
动态内容的处理

- ESI（Edge Side Includes）
  - 在Web代理服务器上做动态内容请求，并将请求插入到静态页面中。
- CSI（Client Side Includes）
  - 单独发起一个异步JavaScript请求，以向服务端获取动态内容。

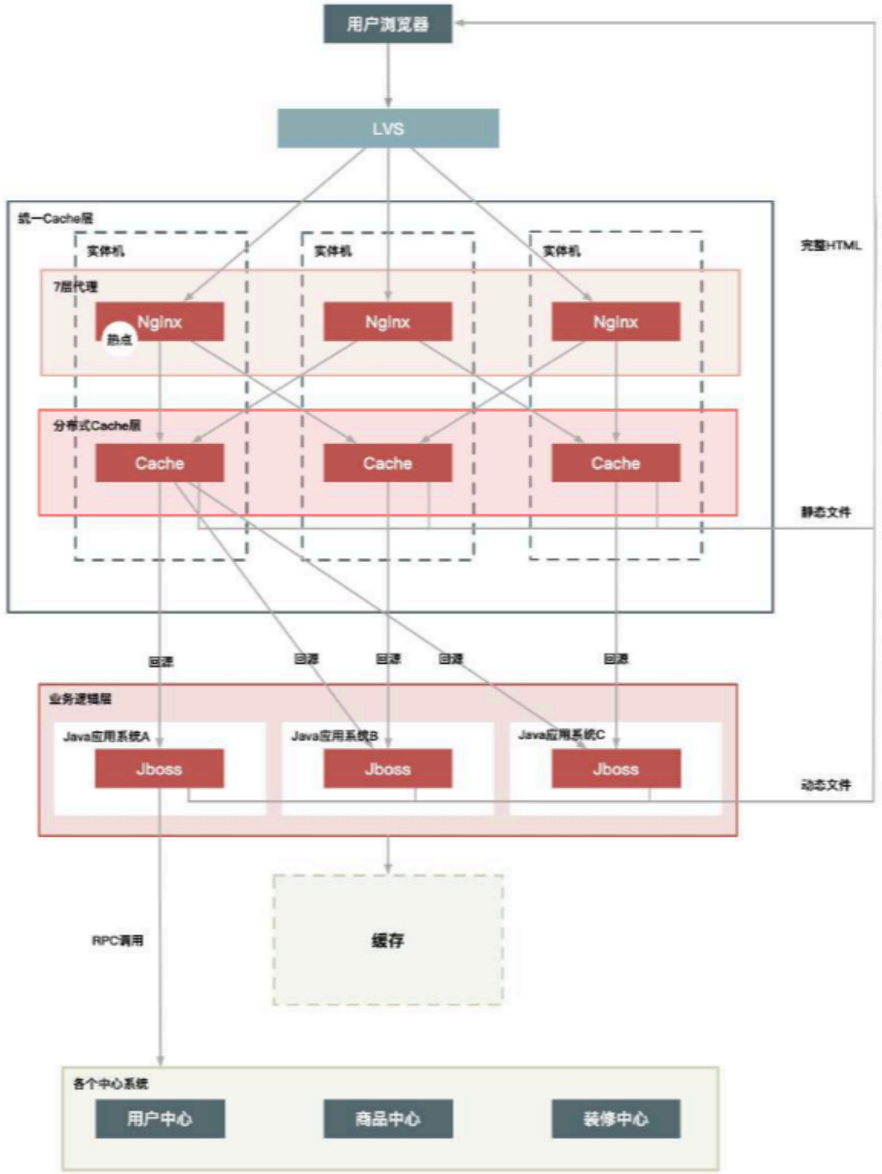
部署方案

- 实体机单机部署
- 统一Cache层
- 上CDN

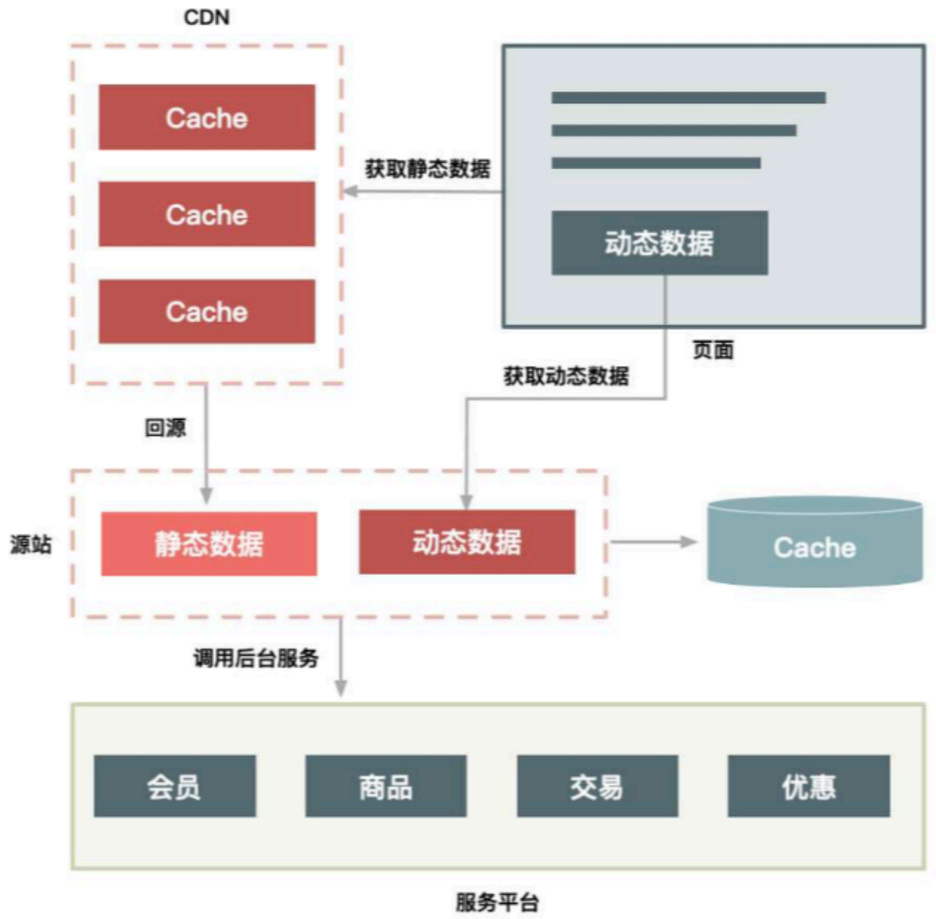
实体单机


用户浏览器
LVS
实体机
实体机
实体机
7层代理
Nginx
Nginx
Nginx
分布式Cache层
Cache
Cache
Cache
ESI/回源
ESI/回源
ESI/回源
Java业务逻辑层
Jboss
Jboss
Jboss
完整HTML
静态文件
动态文件
RPC调用
缓存
各个中心系统
用户中心
商品中心
装修中心

统一Cache


用户浏览器
LVS
统一Cache层
实体机
实体机
实体机
完整HTML
7层代理
Nginx
Nginx
Nginx
热点
分布式Cache层
Cache
Cache
Cache
静态文件
回源
回源
回源
回源
回源
业务逻辑层
Java应用系统A
Java应用系统B
Java应用系统C
Jboss
Jboss
Jboss
动态文件
RPC调用
缓存
各个中心系统
用户中心
商品中心
装修中心

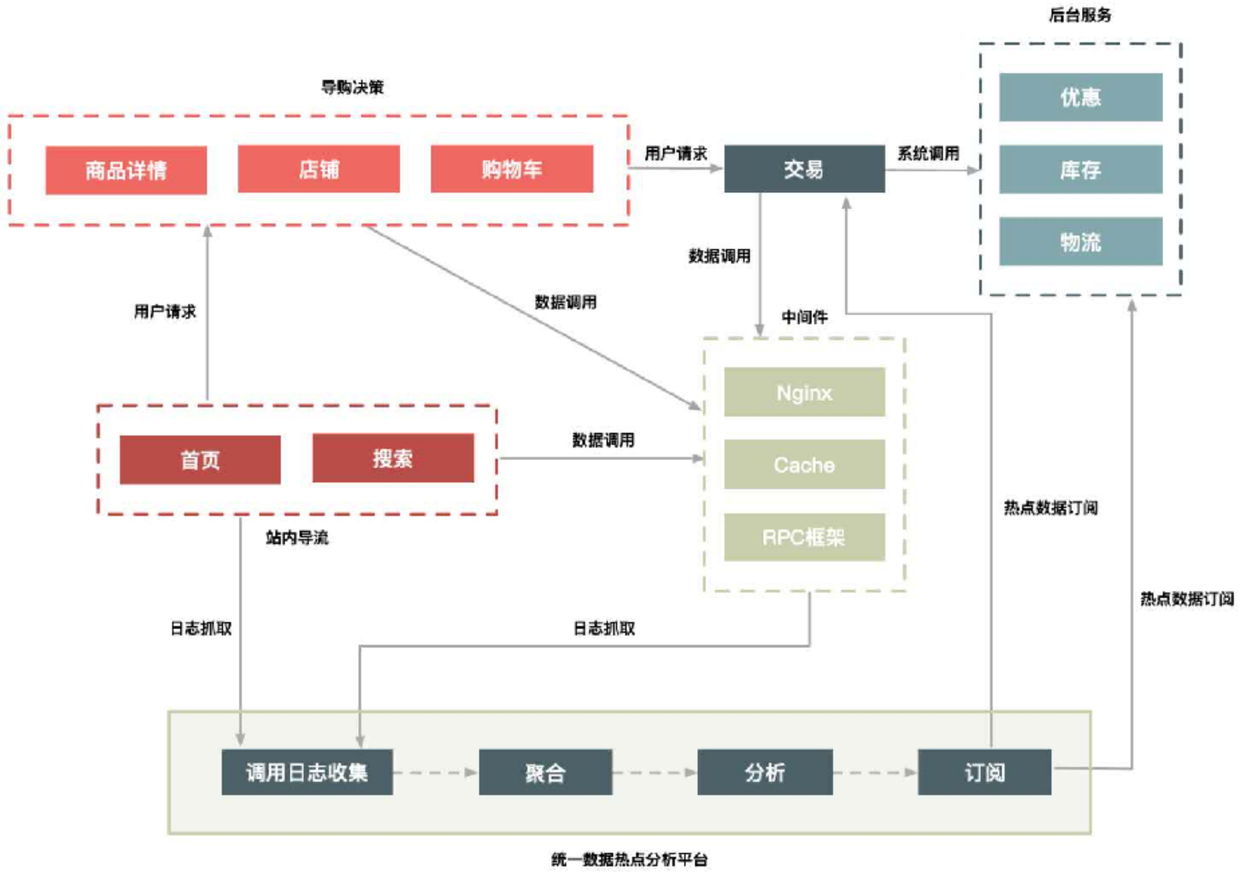
CDN


CDN
Cache
Cache
Cache
获取静态数据
动态数据
页面
获取动态数据
回源
源站
静态数据
动态数据
Cache
调用后台服务
会员
商品
交易
优惠
服务平台

二八原则：热点数据

热点数据

- 静态热点数据
  - 能够提前预测
- 动态热点数据
  - 监控发现突发热点


后台服务
导购决策
商品详情
店铺
购物车
用户请求
交易
系统调用
优惠
库存
物流
数据调用
用户请求
数据调用
中间件
Nginx
首页
搜索
数据调用
Cache
RPC框架
热点数据订阅
站内导流
日志抓取
日志抓取
热点数据订阅
调用日志收集
聚合
分析
订阅
统一数据热点分析平台

动态热点发现系统

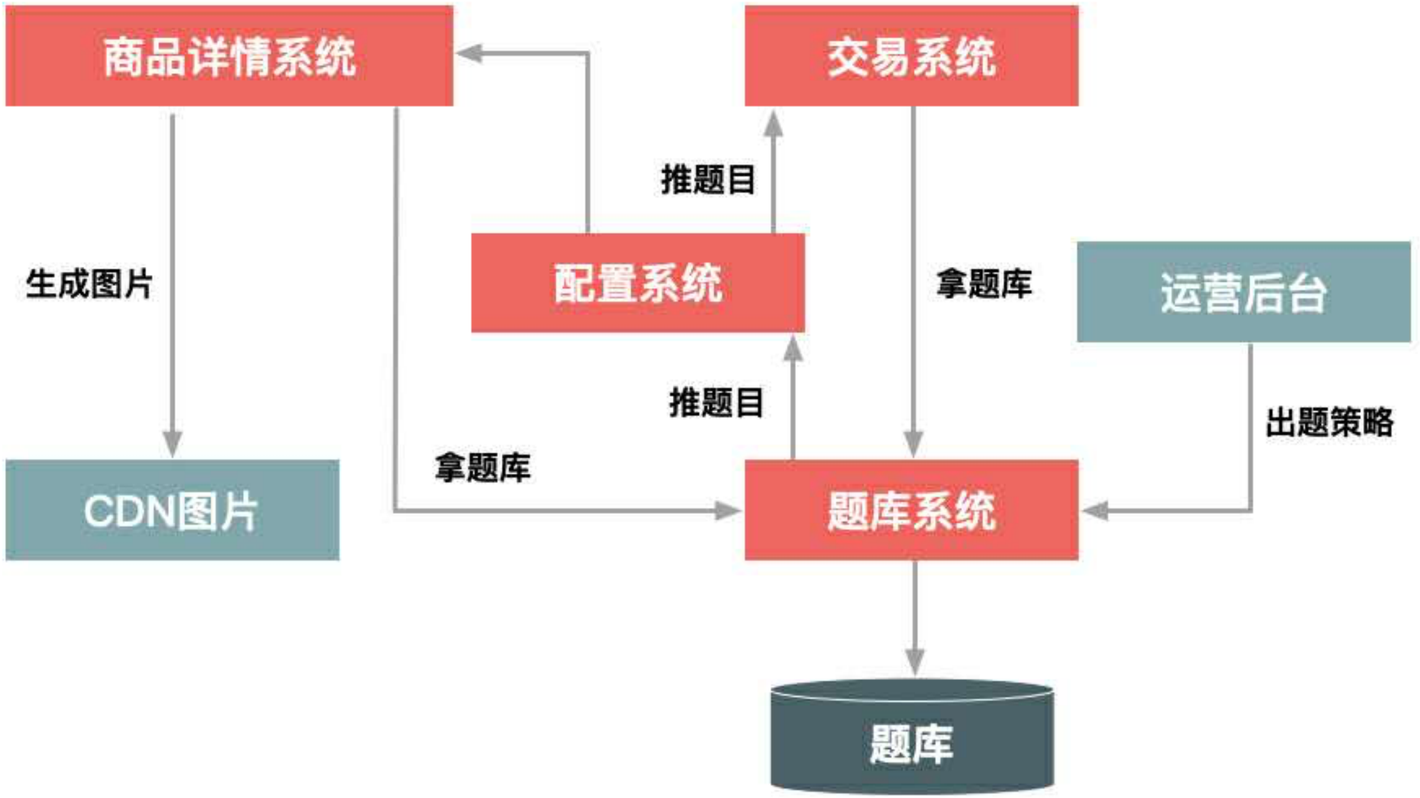
处理热点数据的思路

- 优化
  - 缓存热点数据，队列短暂缓存
- 限制
  - 对商品ID做一致性Hash，然后根据Hash做分桶，每个分桶设置一个处理队列，热点商品限制在一个请求队列里面，避免其他请求得不到处理
- 隔离
  - 隔离热点数据（业务隔离、系统隔离、数据隔离）

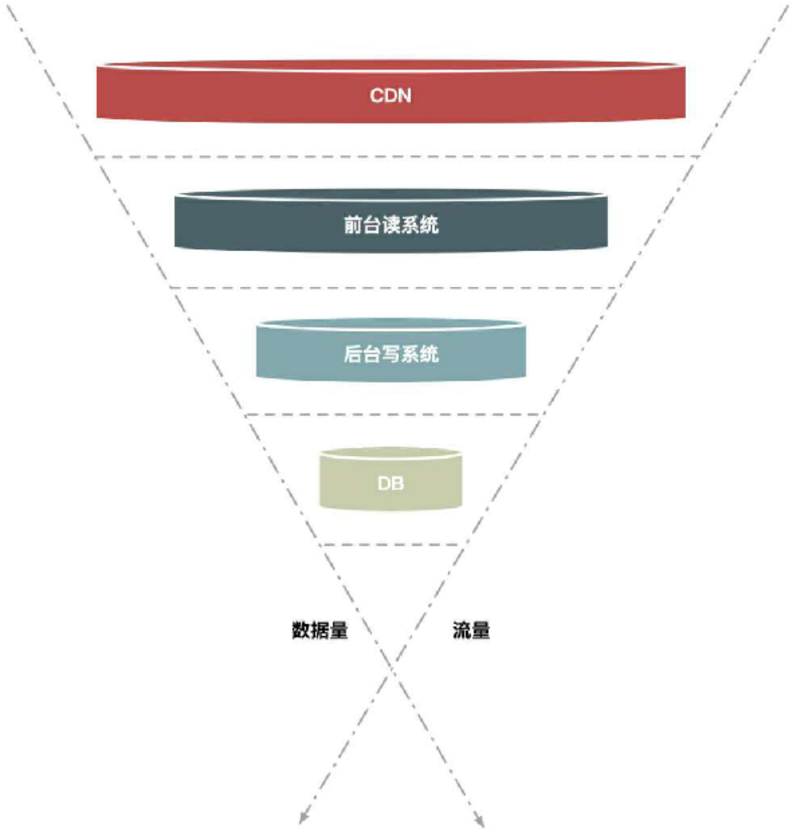
流量削峰

流量削峰

- 排队
- 答题
- 分层过滤



秒杀答题



分层过滤

分层校验的基本原则

- 将动态请求的读数据缓存（Cache）在 Web 端，过滤掉无效的数据读；
- 对读数据不做强一致性校验，减少因为一致性校验产生瓶颈的问题；
- 对写数据进行基于时间的合理分片，过滤掉过期的失效请求；
- 对写请求做限流保护，将超出系统承载能力的请求过滤掉；
- 对写数据进行强一致性校验，只保留最后有效的数据。

影响性能的因素

性能指标

- CPU
  - 主频
- 磁盘
  - IOPS（Input/Output Operations Per Second）
- 服务器
  - QPS (Query Per Secoond)
  - RT（Response Time）
  - 线程数= 2*CPU核数+1

如何发现瓶颈

- CPU诊断工具
  - JProfiler， Yourkit，jstach

如何优化

- 减少编码
    - 把静态的字符串提前编码成字节并缓存
- 减少序列化
- Java极致优化
    - 只需处理极少数动态请求
        - 直接使用servlet
        - 直接输出流数据
- 并发读数据
    - 单台缓存机器（30w/s）
    - 应用层的LocalCache

减库存

减库存

- 下单减库存
  - 问题：恶意下单
- 付款减库存
  - 问题：超卖
- 预扣库存

并发锁

- 应用层做排队
- 数据层做排队

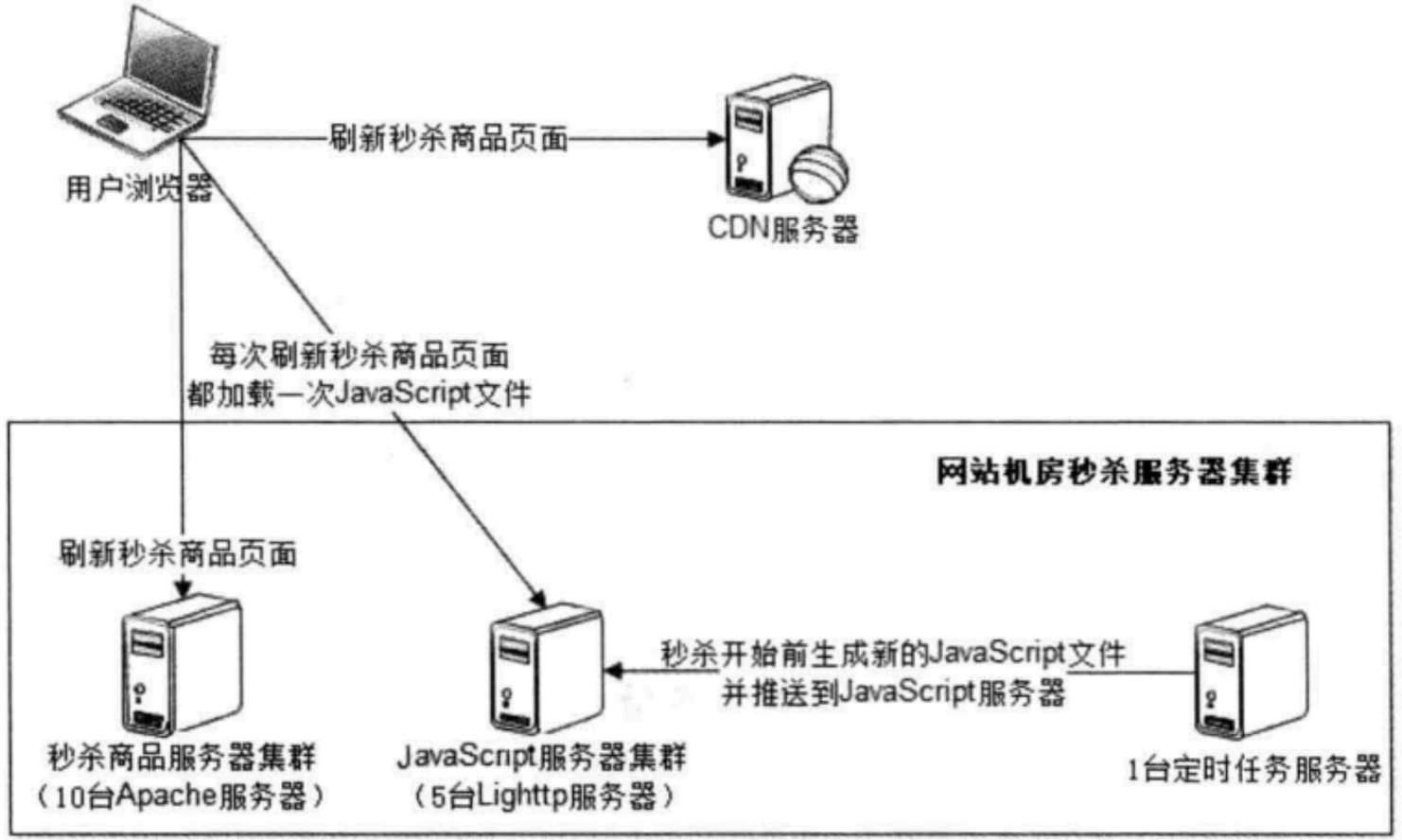

异地容灾
异步化
分组隔离
避免单点
架构阶段
限流保护
超时处理
异步线程
错误捕获
编码阶段
beta测试
自动化对比测试
测试阶段
分批发布
多版本发布
发布阶段
数据对账
自动降级
过载保护
实时监控报警
运行阶段
快速恢复
故障定位
故障发生

高可用系统建设

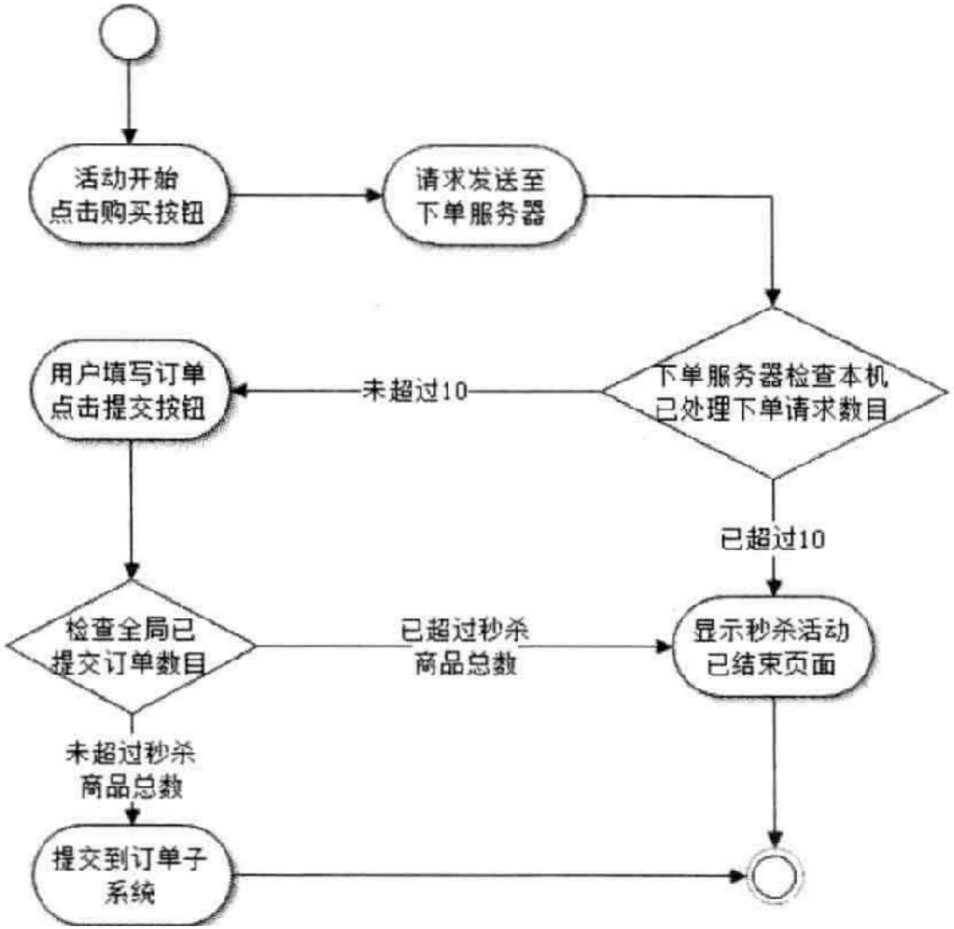
其它业务需求的实现

如何避免直接下单

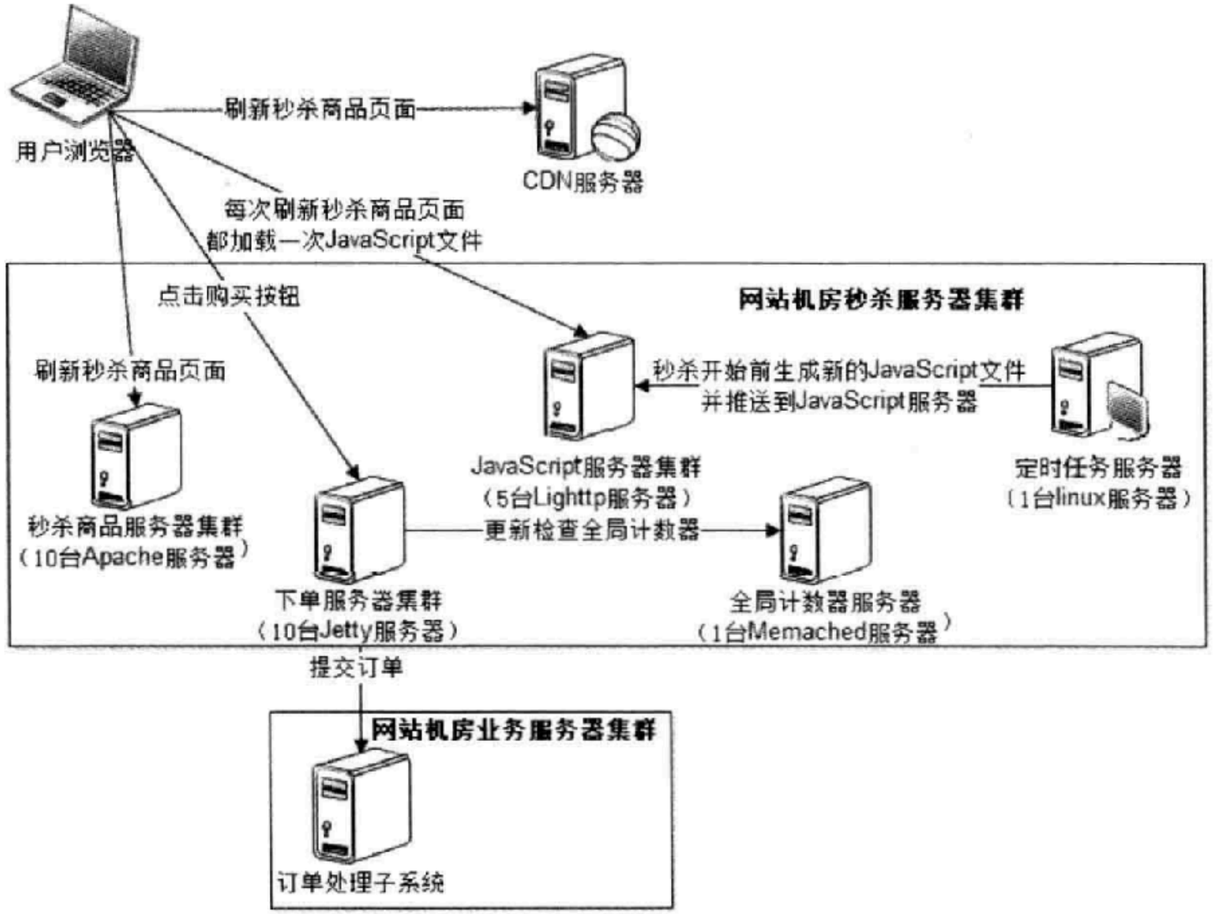
- 直接下单
  - 秒杀的游戏规则是到了秒杀才能开始对商品下单购买，在此时间点之前，只能浏览商品信息，不能下单。而下单页面也是一个普通的URL，如果得到这个URL，不用等到秒杀开始就可以下单了。
- 解决方案：
  - 为了避免用户直接访问下单页面URL，需要将改URL动态化，即使秒杀系统的开发者也无法在秒杀开始前访问下单页面的URL。办法是在下单页面URL加入由服务器端生成的随机数作为参数，在秒杀开始的时候才能得到。



如何控制秒杀商品页面购买按钮的点亮

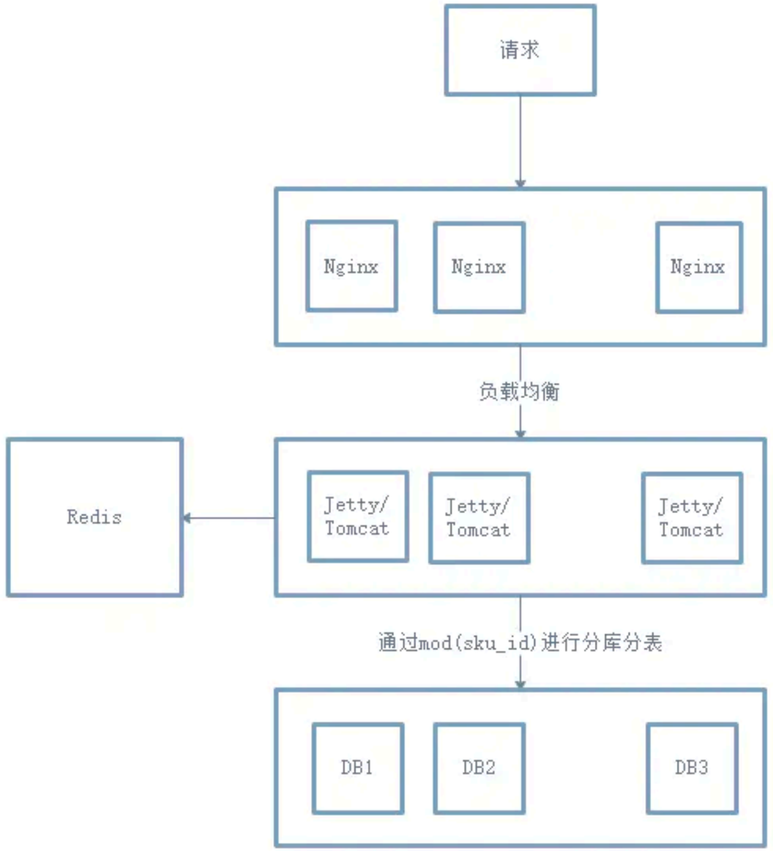


如何只允许第一个提交的订单被发送到订单子系统



如何只允许第一个提交的订单被发送到订单子系统

方案对比



Java方案

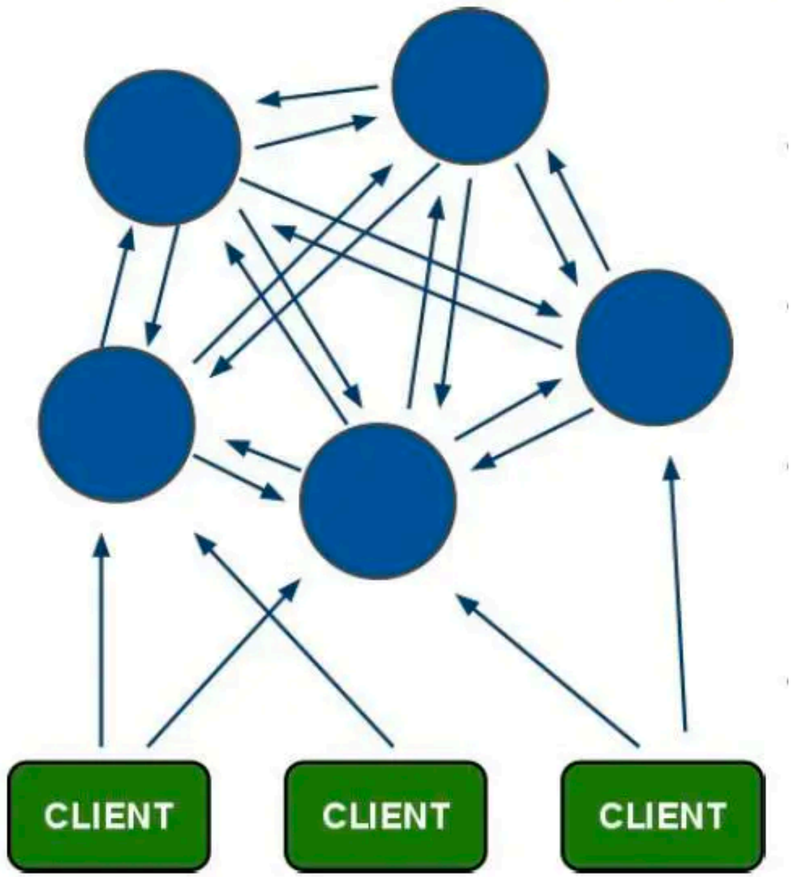
Java Nginx Tomcat Redis ShardingSphere

Redis基础数据类型

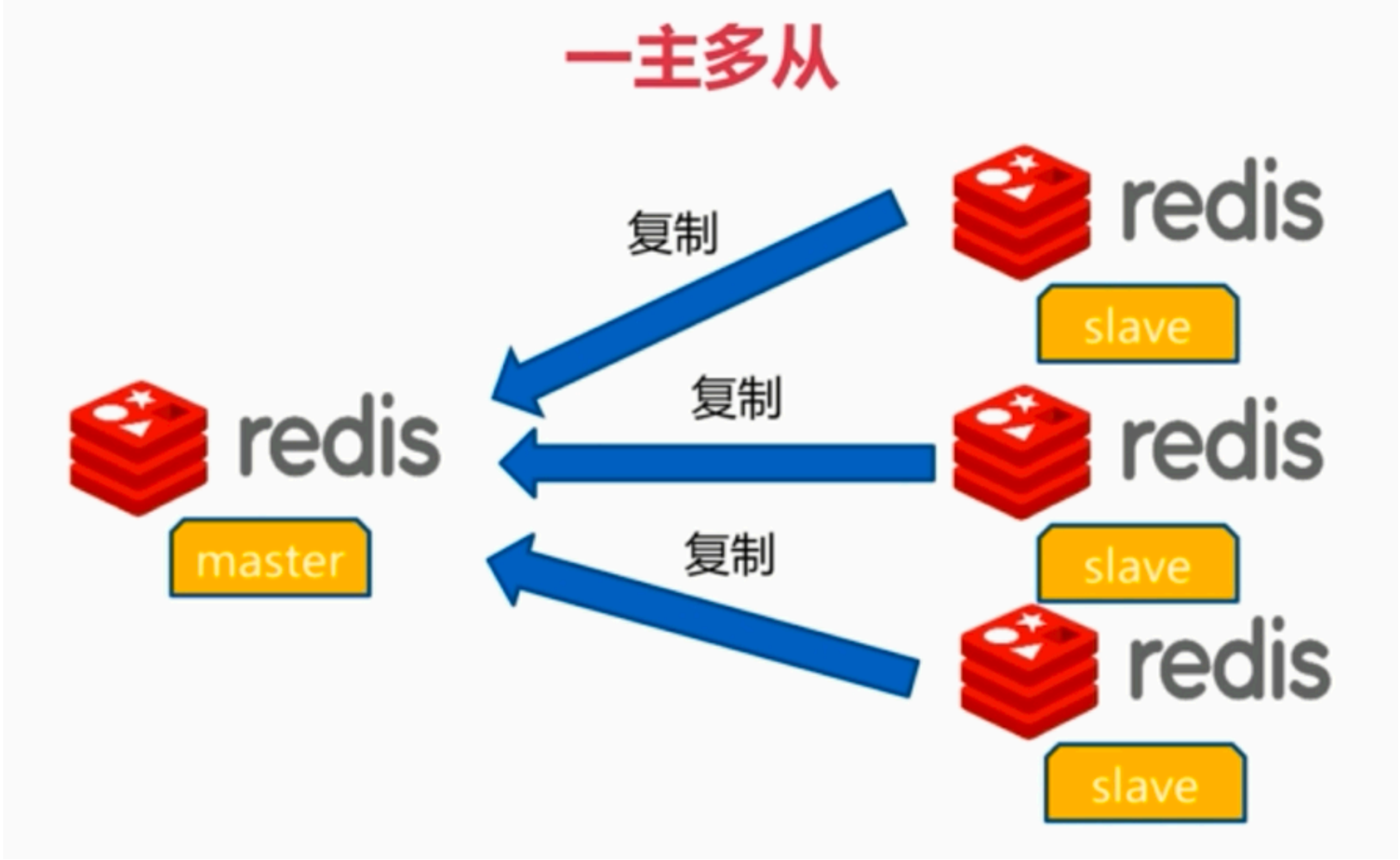
- String
  - 缓存功能、计数器、Session
- Hash
- List
  - 消息队列、文章列表或者数据分页
- Set
  - 共同好友
- SortedSet
  - 排行榜、带权重的工作

Redis

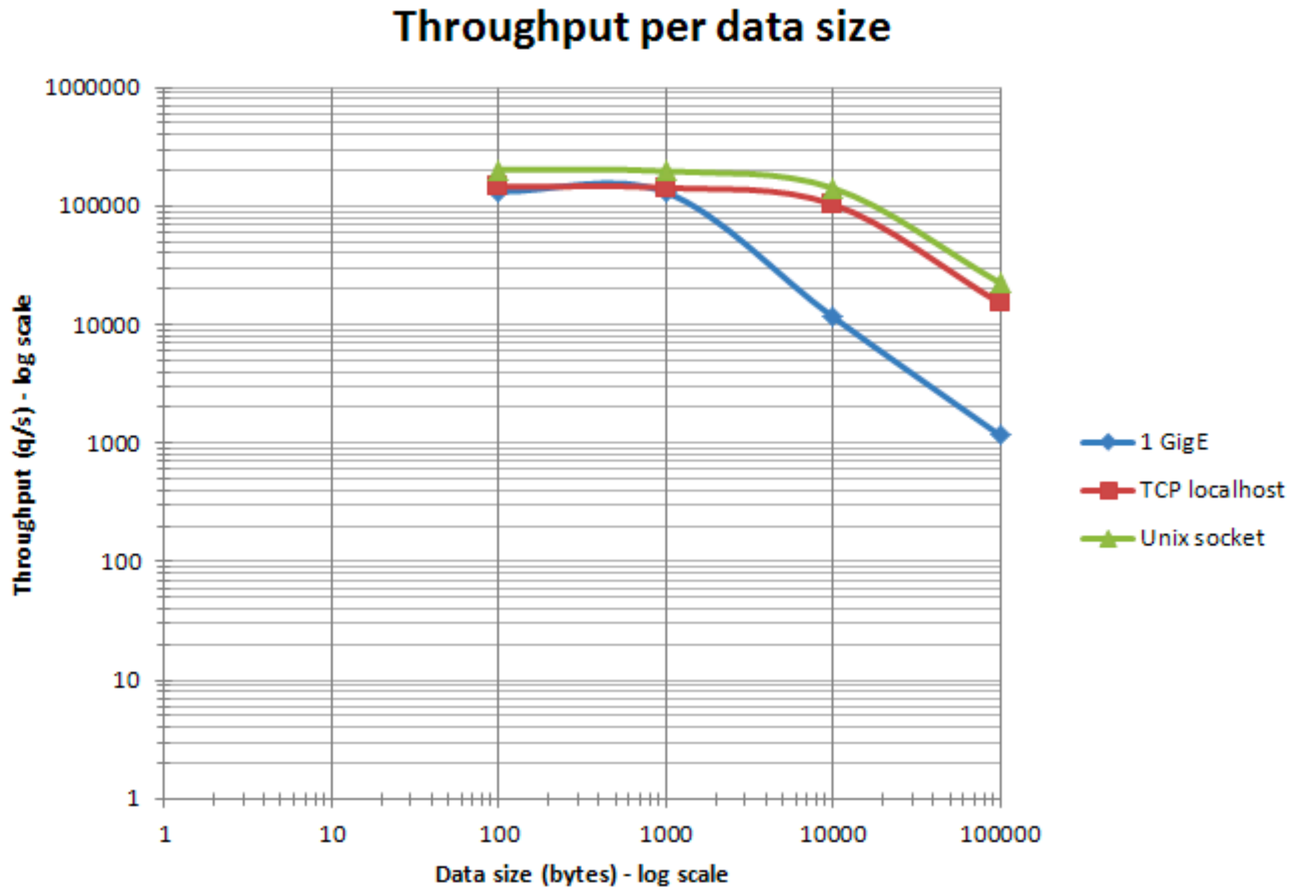
- Redis 集群提供了以下两个好处：
  - 将数据自动切分（split）到多个节点的能力。
  - 当集群中的一部分节点失效或者无法进行通讯时， 仍然可以继续处理命令请求的能力。



Redis集群


一主多从
redis
slave
复制
redis
master
复制
redis
slave
复制
redis
slave

主从复制模型



Redis性能

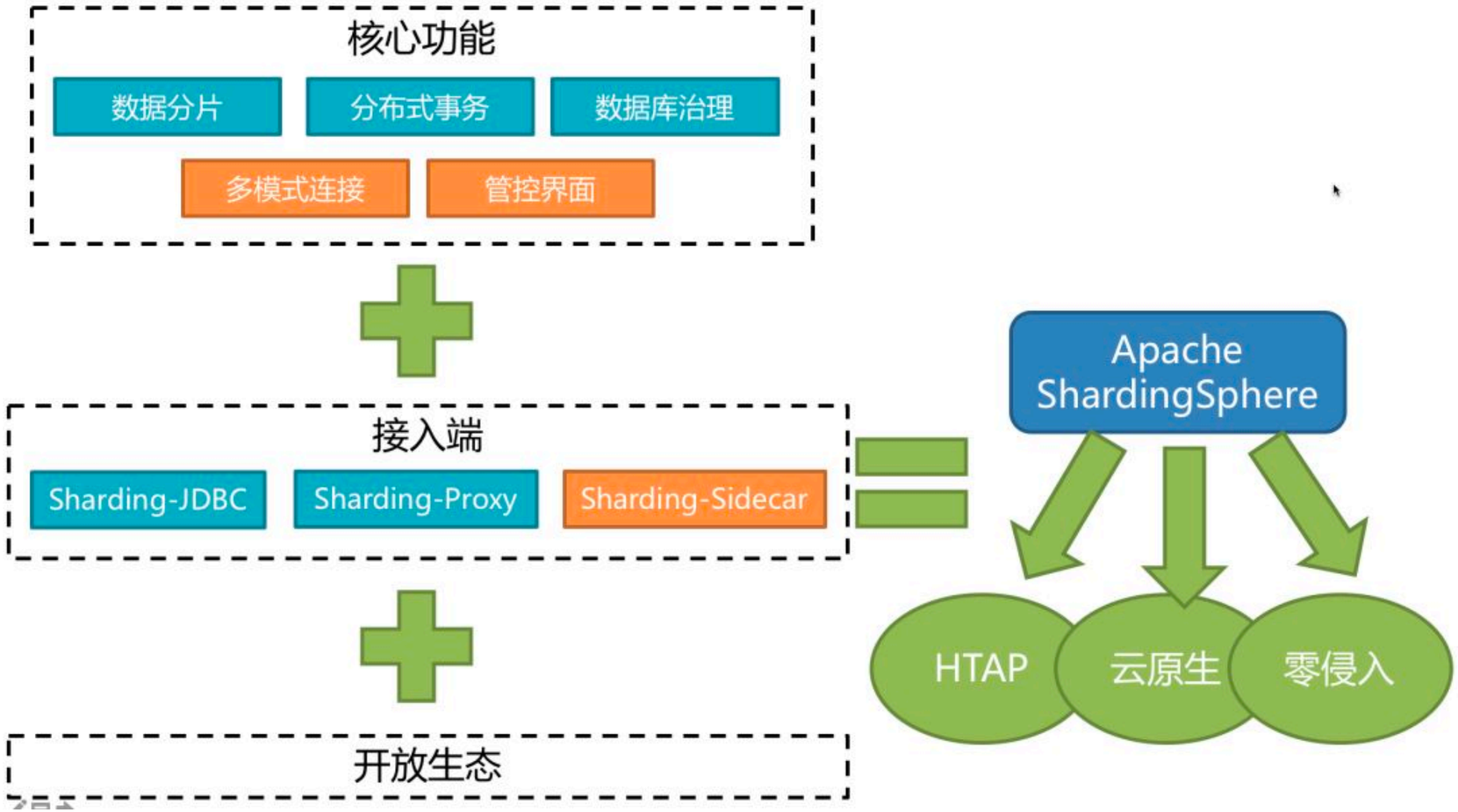
payload 1k 基本是 Redis 性能的一个拐点

性能

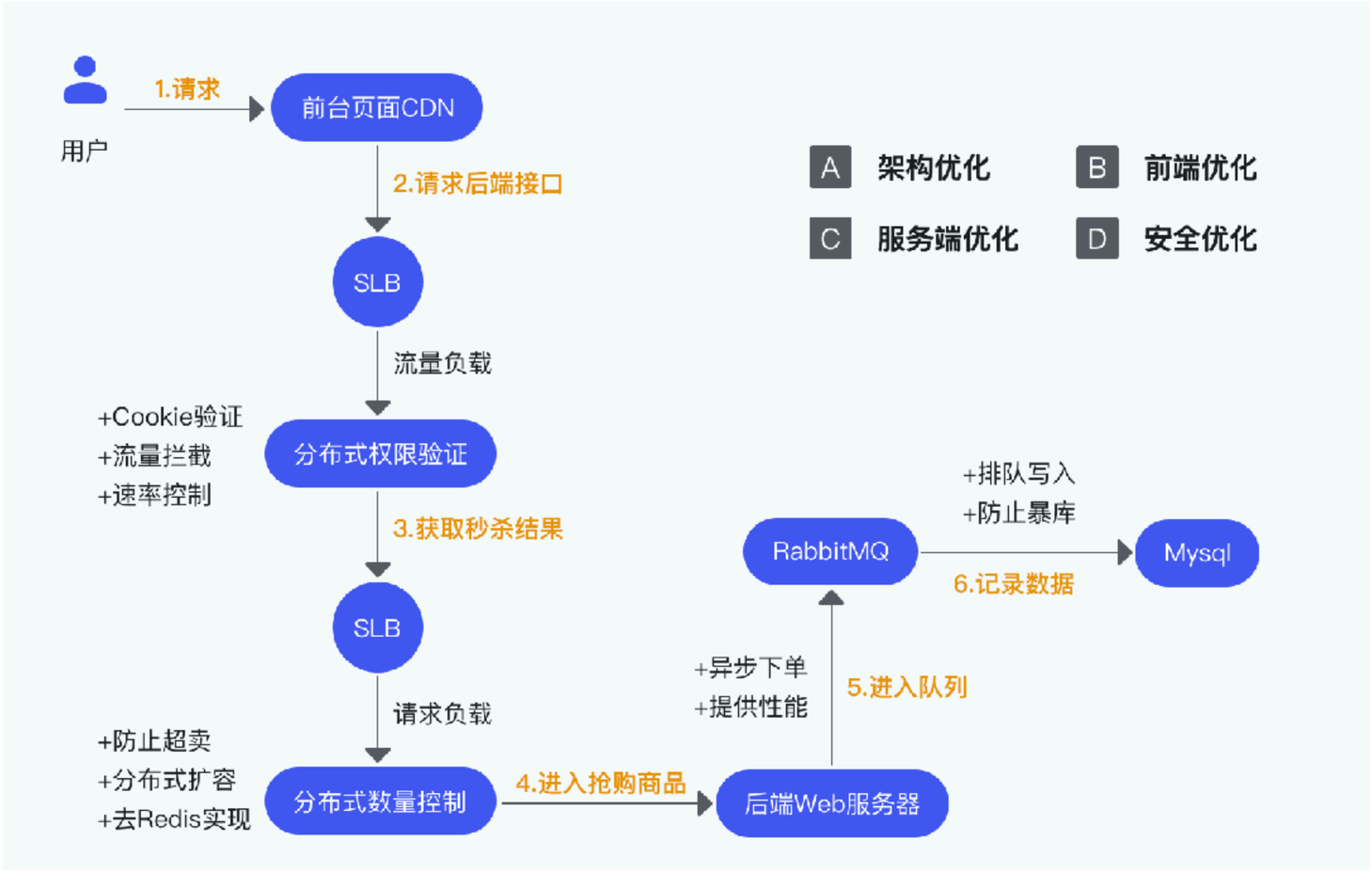
- 单个 Redis 的性能上限带来的瓶颈问题。
  - Redis 的单线程设计机制只能利用一个核，导致单核 CPU 的最大处理能力就是 Redis 单实例处理能力的天花板了。
- 如果业务量峰值继续增高，看起来单个 Redis 分片还有大约 20% 的余量就到单实例极限了。
- 缓存击穿、缓存雪崩

一致性

- Redis 并不能保证数据的强一致性. 这意味这在实际中集群在特定的条件下可能会丢失写操作。因为集群是用了异步复制. 写操作过程:
  - 客户端向主节点B写入一条命令.
  - 主节点B向客户端回复命令状态.
  - 主节点将写操作复制给他得从节点 B1, B2 和 B3

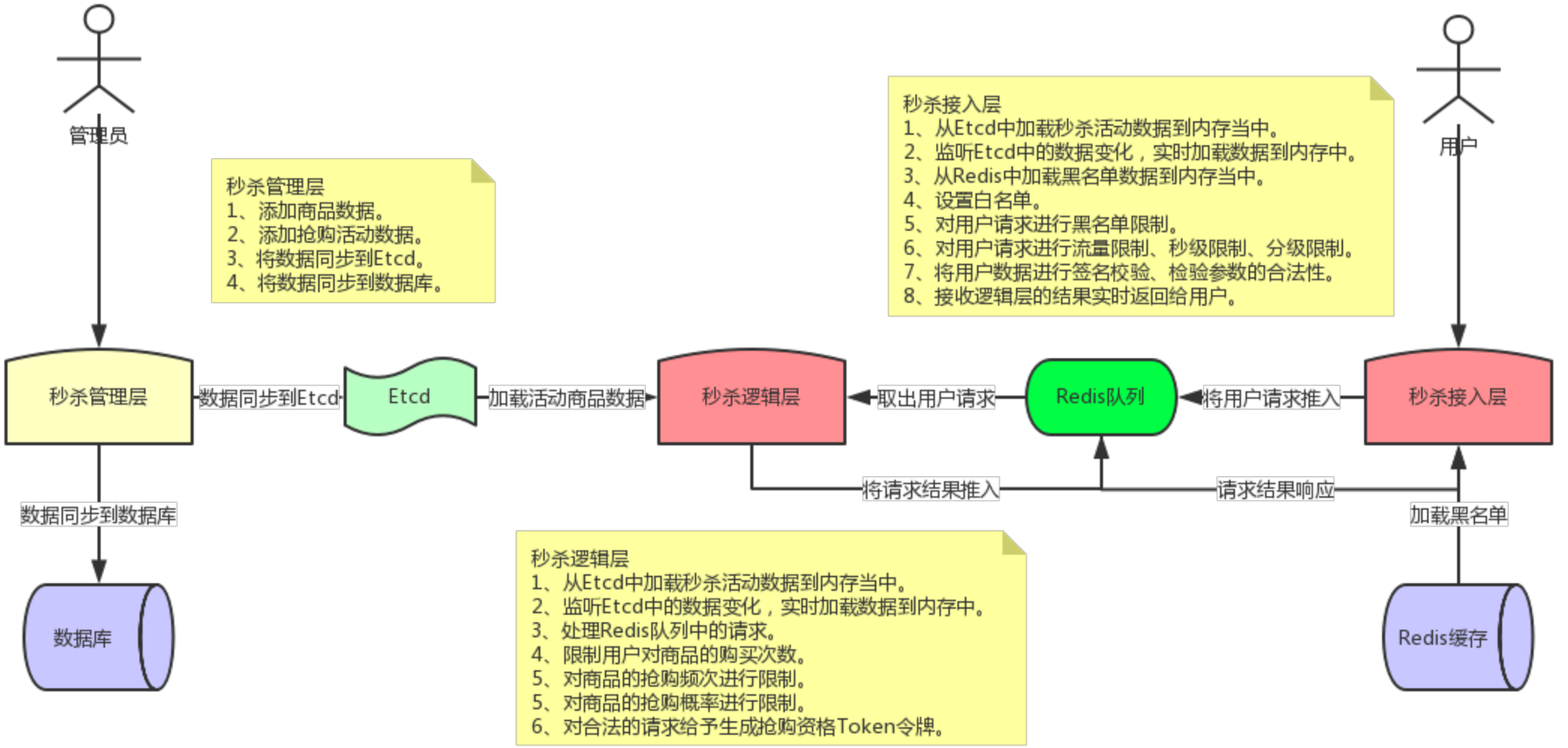


ShardingSphere



Go方案

Golang RabbitMQ Iris MySQL



另一个Go+redis方案