微服务架构设计与实践

聚合层[关键技术]篇

孙玄@58集团

关于我

- 58集团技术主席
- 58集团高级架构师
- 转转架构负责人
- 百度高级工程师
- 毕业于浙江大学
- 代表公司多次对外分享
- 企业内训&公开课

关于我

企业内训

- ✓ 华为
- ✓ 中航信
- ✓ 平安
- ✓ 银联
- ✓ 华泰证券
- ✓ 思科
- ✓ 云南电力
- ✓ 深信服
- ✓ 新华社
- ✓ 民生银行
- ✓ 招商银行
- ✓ ……

公开课

- ✓ 北京
- ✓ 上海
- ✓ 深圳
- ✓ 广州
- ✓ 成都
- ✓ ……

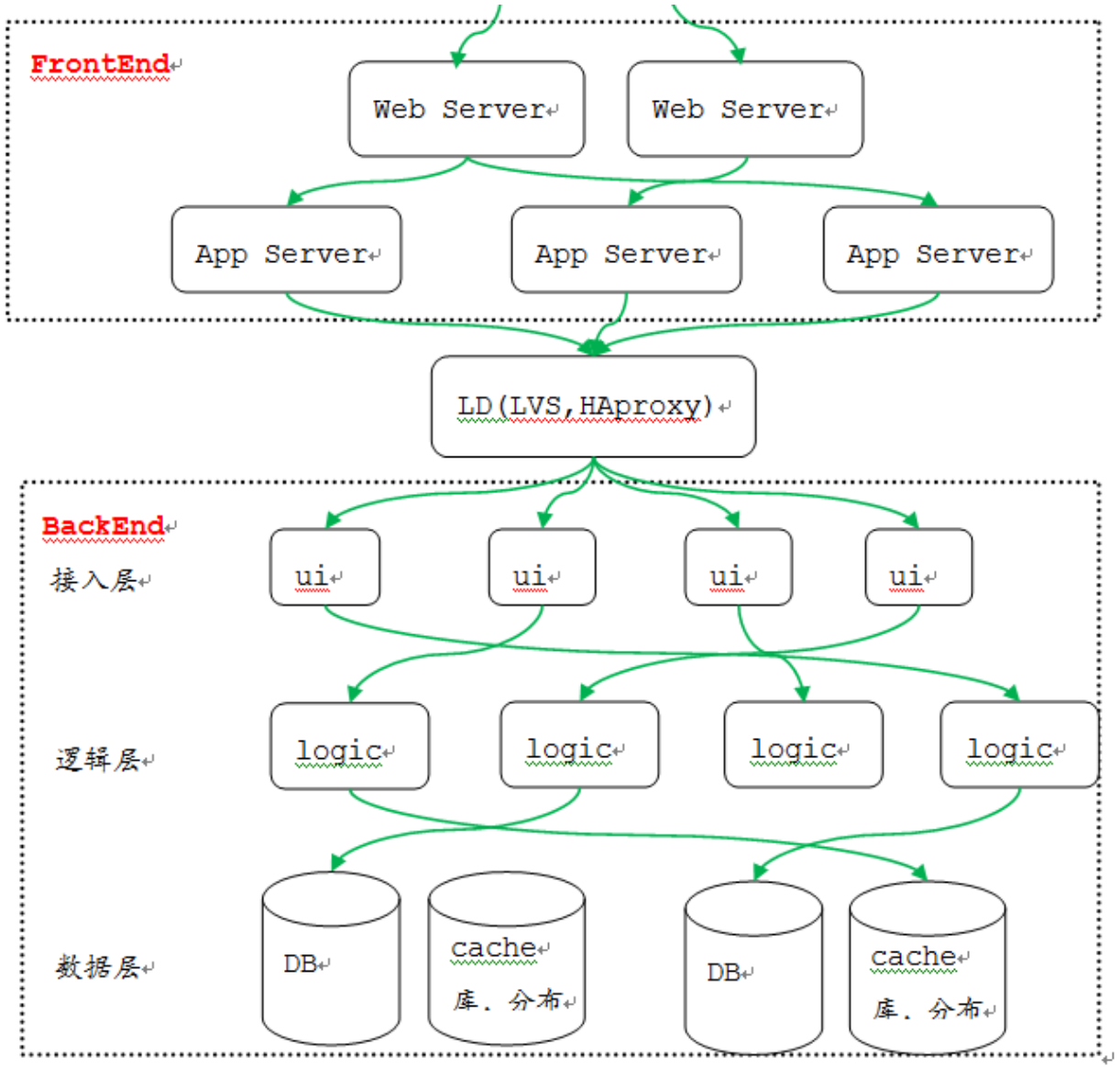
分享要点

微服务架构

1. 微服务聚合层设计
2. 无状态化&纯异步设计
3. 分级&超时设计
4. 服务降级
5. 幂等设计
6. 案例实践

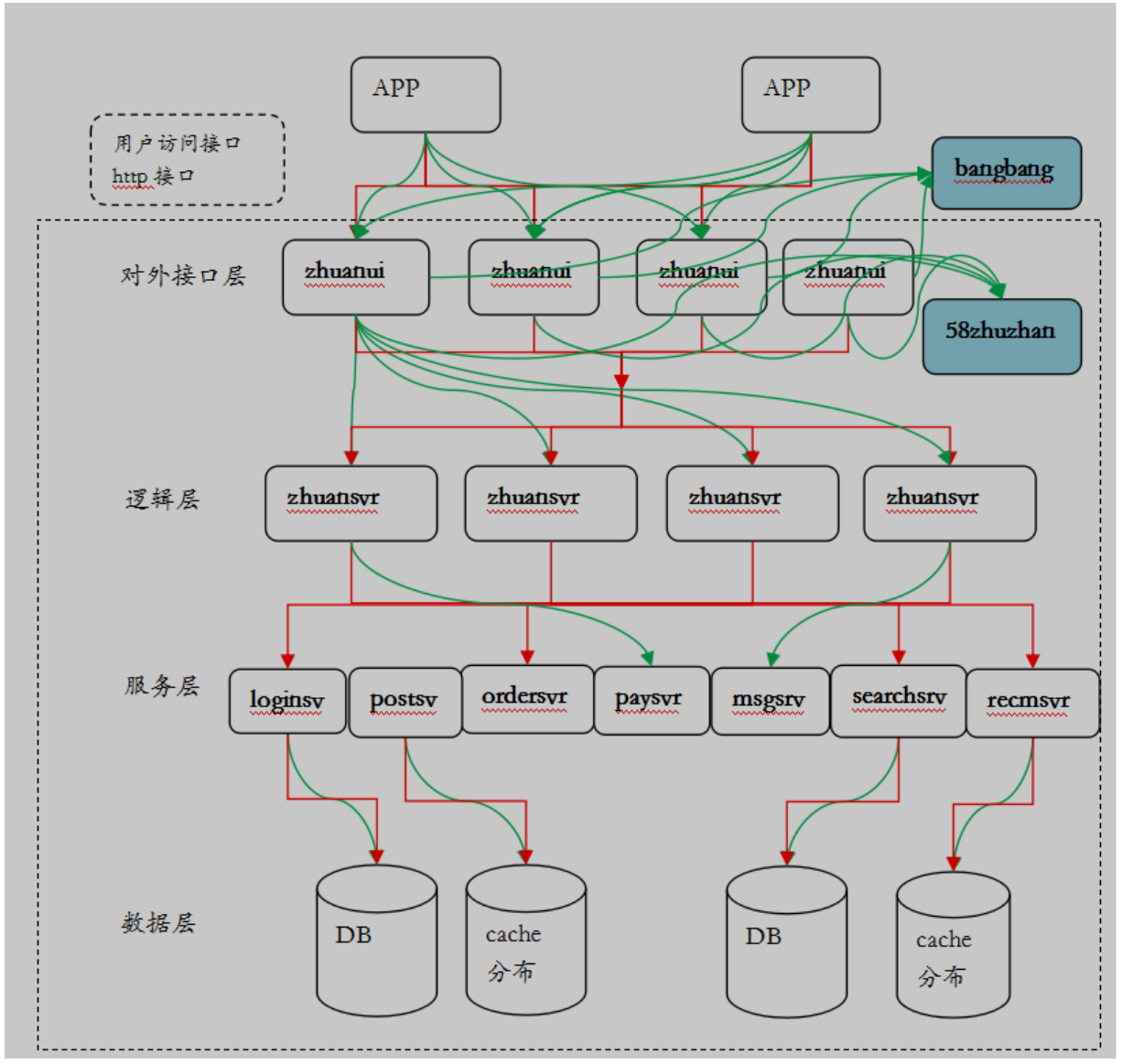
互联网产品通用技术架构

- 网关层
- **聚合层**
- 数据层



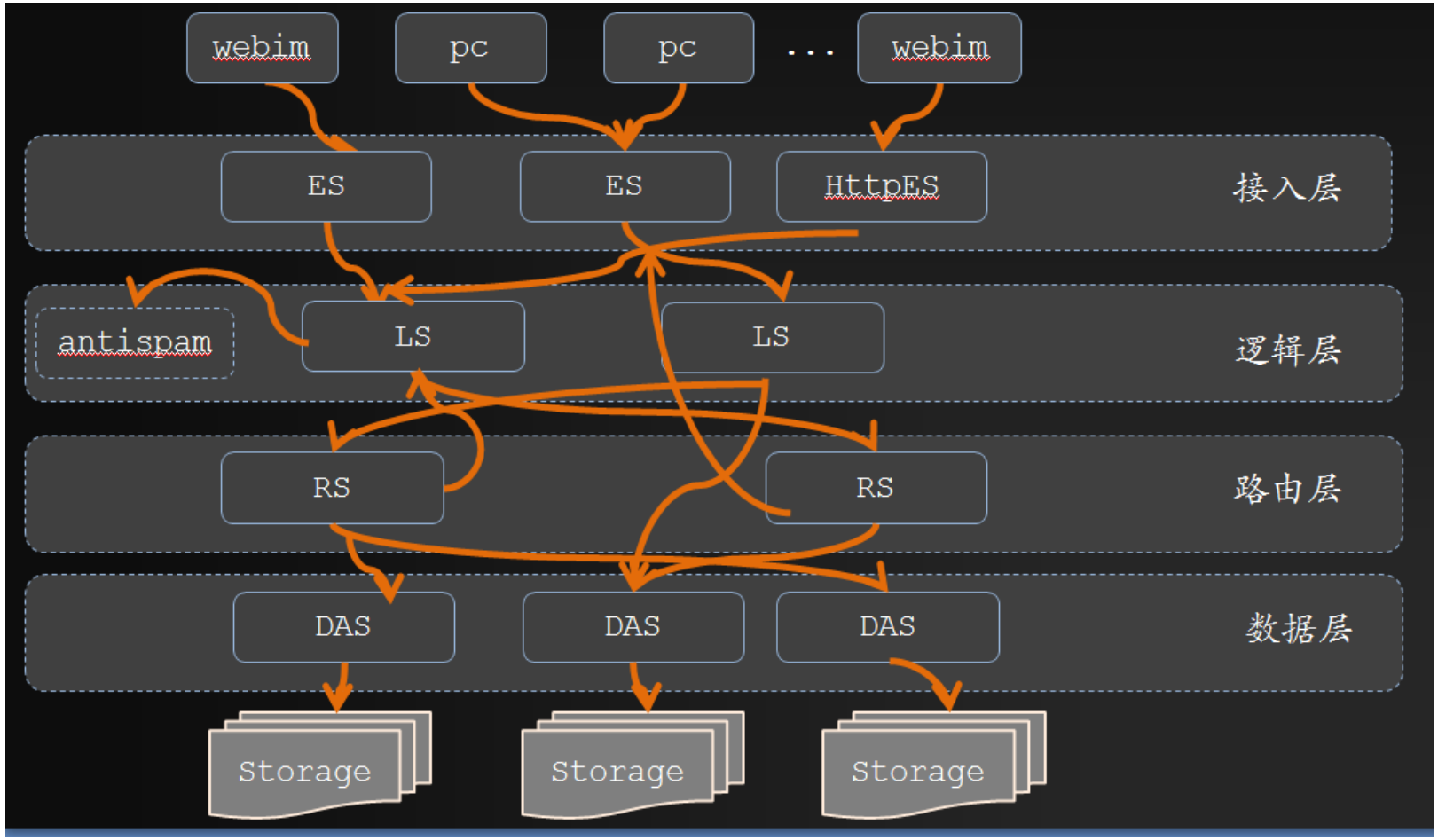
互联网产品通用技术架构

- **接入层**
- **逻辑层**
- **数据层**



互联网产品通用技术架构

- 网关层
- **聚合层**
- 数据层



微服务聚合层做什么

- **聚合层职责**
  - 整个系统中业务逻辑处理
  - 58帮帮为例
    - 用户
      - 用户登录登出、用户信息设置查询等
    - 好友
      - 添加好友、获取好友、删除好友、修改好友信息等
    - 消息
      - 收发好友消息、收发陌生人消息、消息确认、通用消息处理、离线消息等

聚合层架构

- **ALL IN ONE**
  - Monoliths
  - 所有业务一个整体
  - 一个文件
  - 一个类
  - ……

聚合层架构

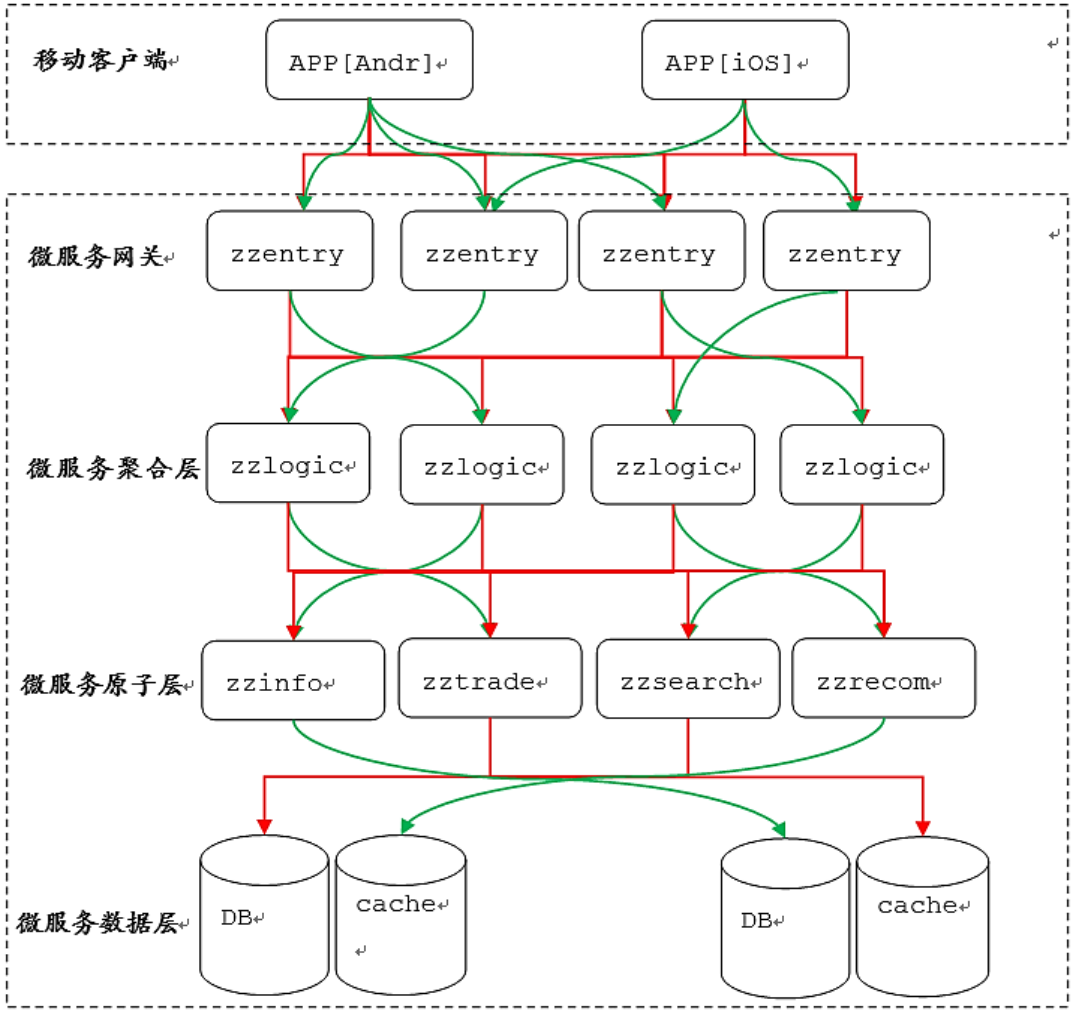
- **Monoliths**
  - **问题**
    - 耦合性严重
    - 开发代价高
    - 维护代价高
    - 牵一发动全身

聚合层架构

- **Monoliths**
  - **优点**
    - 架构简单
    - 性能高
      - Cell
      - 单元化
  - **适用场合**
    - 创业公司
    - 高性能

聚合层架构

- **架构演进**
  - **业务功能单元逻辑划分**
    - 一个业务功能单元一个组件
      - 目录或者文件
    - 逻辑拆分
    - Monoliths



聚合层架构

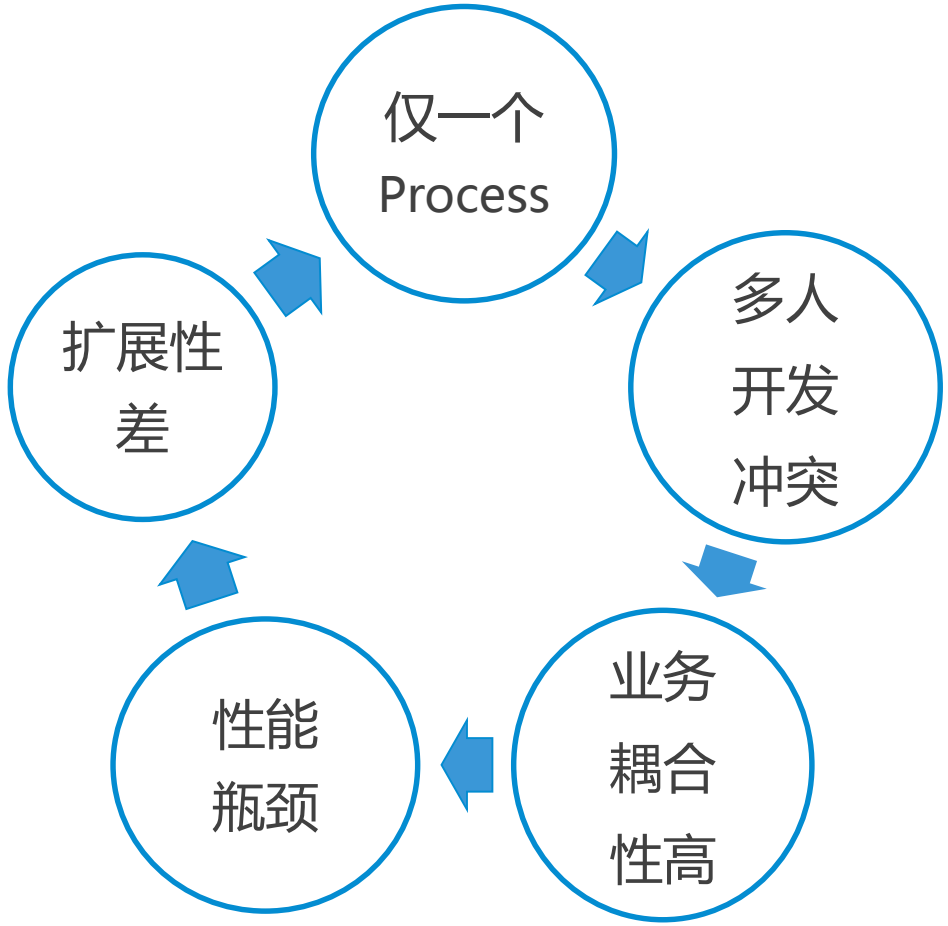
- **业务功能单元逻辑划分**
  - **优点**
    - 耦合性降低
    - 业务功能单元间开发互不影响
    - 开发效率高

聚合层架构

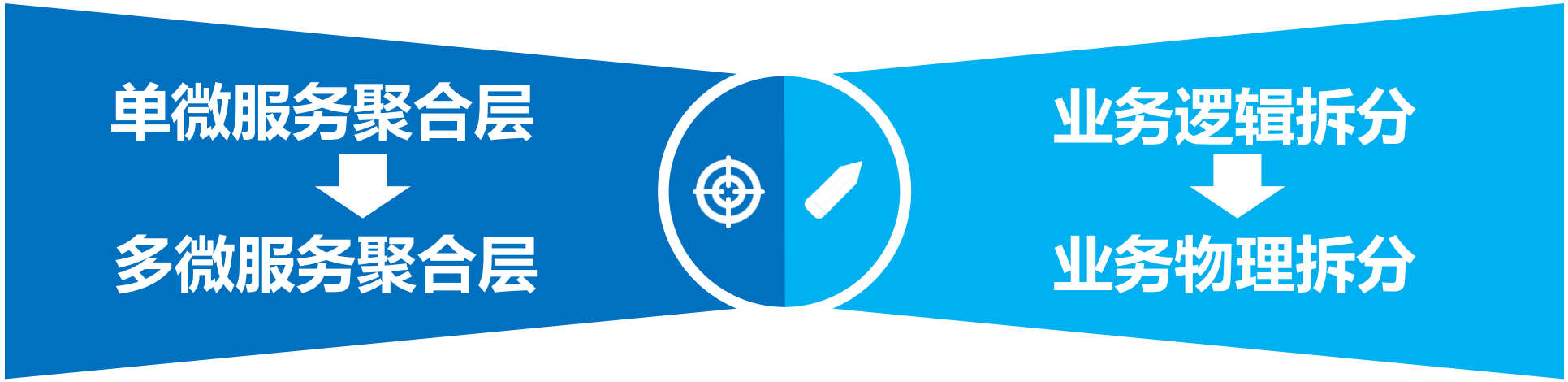
- **业务功能单元逻辑划分**
  - **缺点**
    - 物理上一个模块
    - 编译成本高
    - 一个业务上线，重启影响所有业务
  - **适用场景**
    - 互联网公司使用较多
      - 58、百度

聚合层演进-存在问题

微服务聚合层

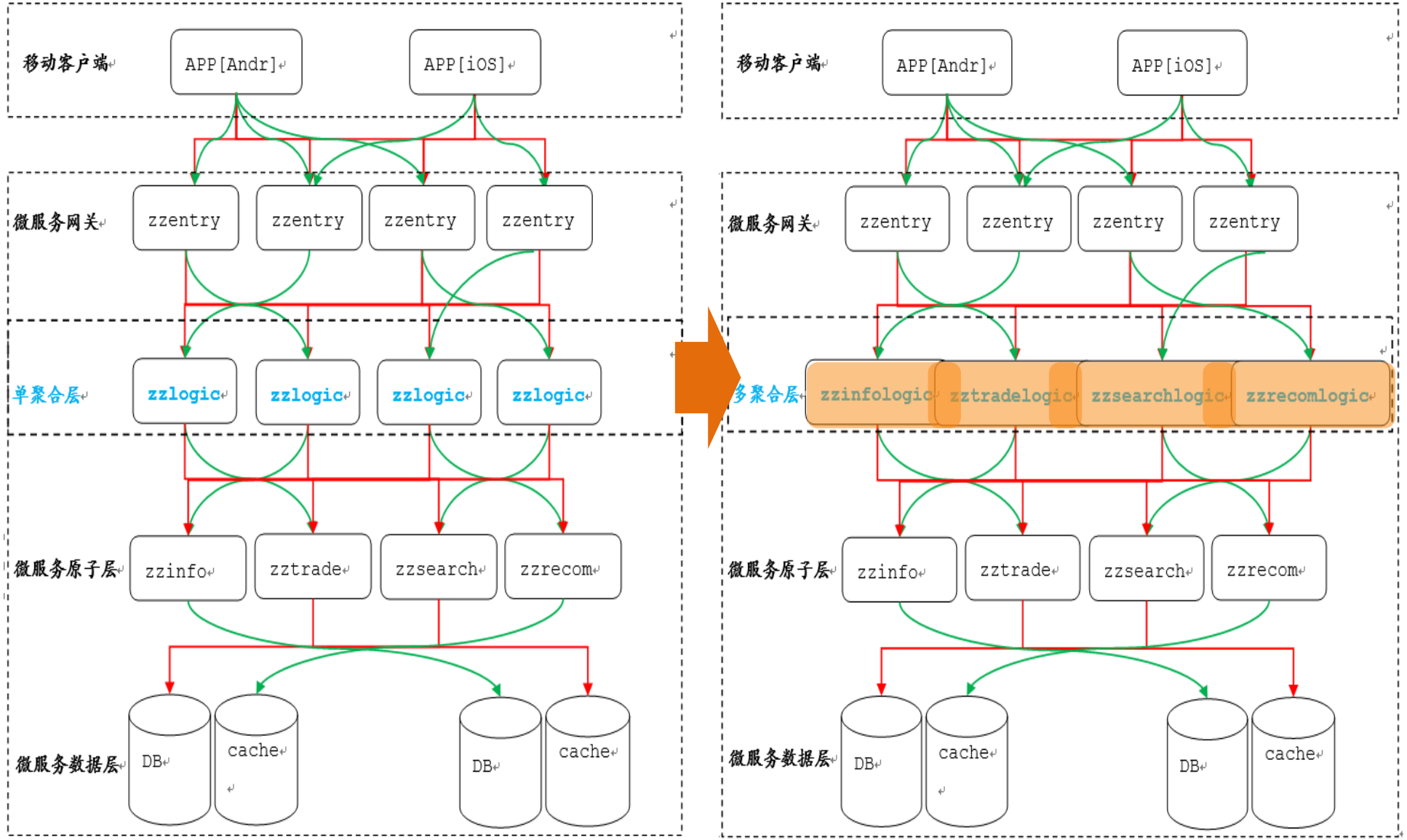


微服务聚合层演进



业务领域模型拆分

微服务聚合层演进


移动客户端
APP[Andr]
APP[iOS]
微服务网关
zzentry
zzentry
zzentry
zzentry
单聚合层
zzlogic
zzlogic
zzlogic
zzlogic
微服务原子层
zzinfo
zztrade
zzsearch
zzrecom
微服务数据层
DB
cache
DB
cache
移动客户端
APP[Andr]
APP[iOS]
微服务网关
zzentry
zzentry
zzentry
zzentry
多聚合层
zzinfologic
zztradelogic
zzsearchlogic
zzrecomlogic
微服务原子层
zzinfo
zztrade
zzsearch
zzrecom
微服务数据层
DB
cache
DB
cache

多微服务聚合层优点

独立

- ✓ 进程
- ✓ 开发
- ✓ 部署
- ✓ 运维

高效

- ✓ 快速迭代
- ✓ 持续交付

无状态设计

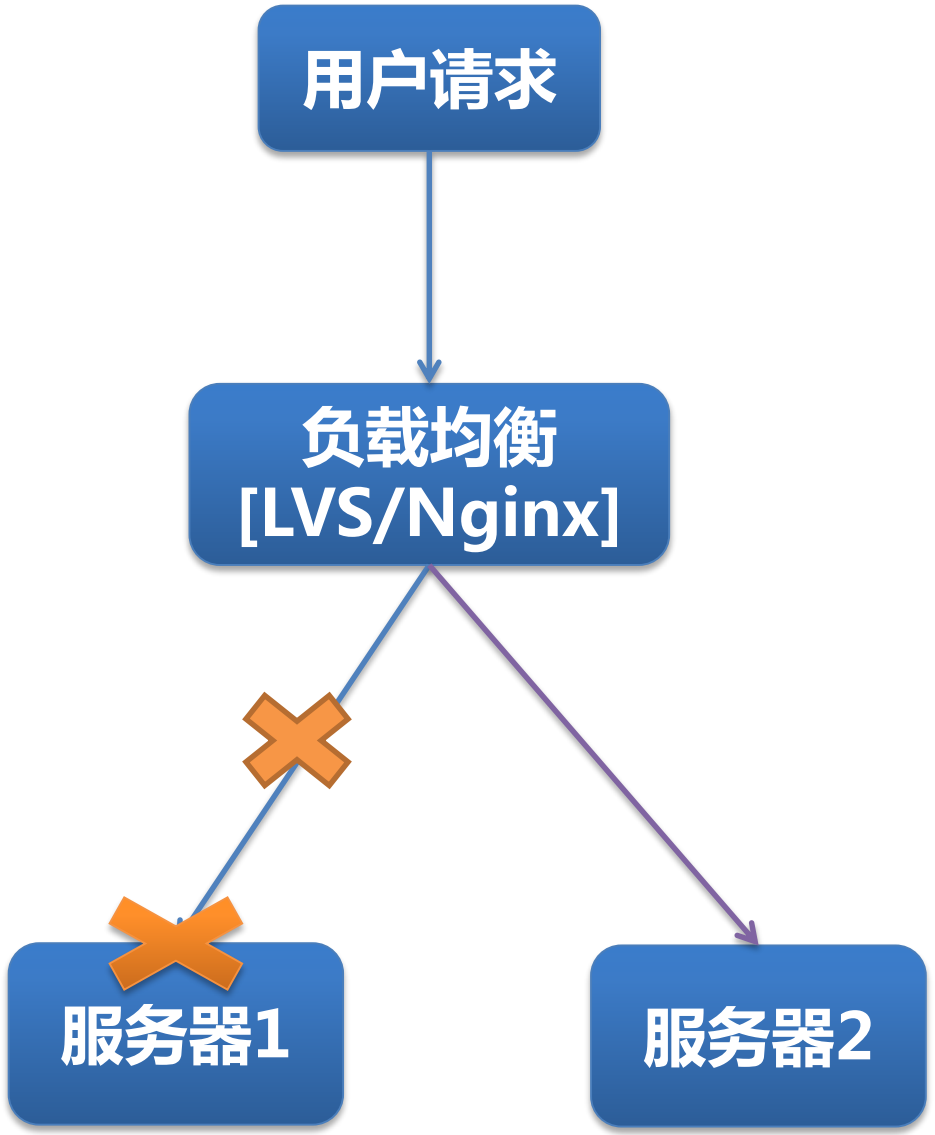
- **什么是无状态**
  - 系统不存储任何请求上下文信息
  - 仅根据每次请求携带数据进行相应处理
    - 请求返回后，所有中间数据清空
  - 多个模块（子系统）之间完全对称
  - 请求提交到任何服务器，处理结果相同

无状态设计

- **无状态设计**
  - **关键因素**
    - 聚合层不保存任何数据
    - 所有聚合层服务器完全对称
    - 当一台或者多台宕机
    - 请求可提交到集群中的任意可用服务器
    - 聚合层高可用
      - 负载均衡

无状态设计

- **负载均衡**
  - Keepalive
  - Retry



纯异步调用设计

- **同步**
  - 发出一个请求调用，在没有得到结果之前，该调用不返回
  - 调用者(线程)阻塞模式
- **异步**
  - 异步调用发出后，调用者立即返回。结果完成后，通过状态、通知和回调来通知调用者
  - 调用者(线程)非阻塞模式

纯异步调用设计

- **优点**
  - 线程(调用者)非阻塞，一直Running，CPU利用率高，系统性能高
  - 系统吞吐量高
- **缺点**
  - 实现成本高

纯异步调用设计

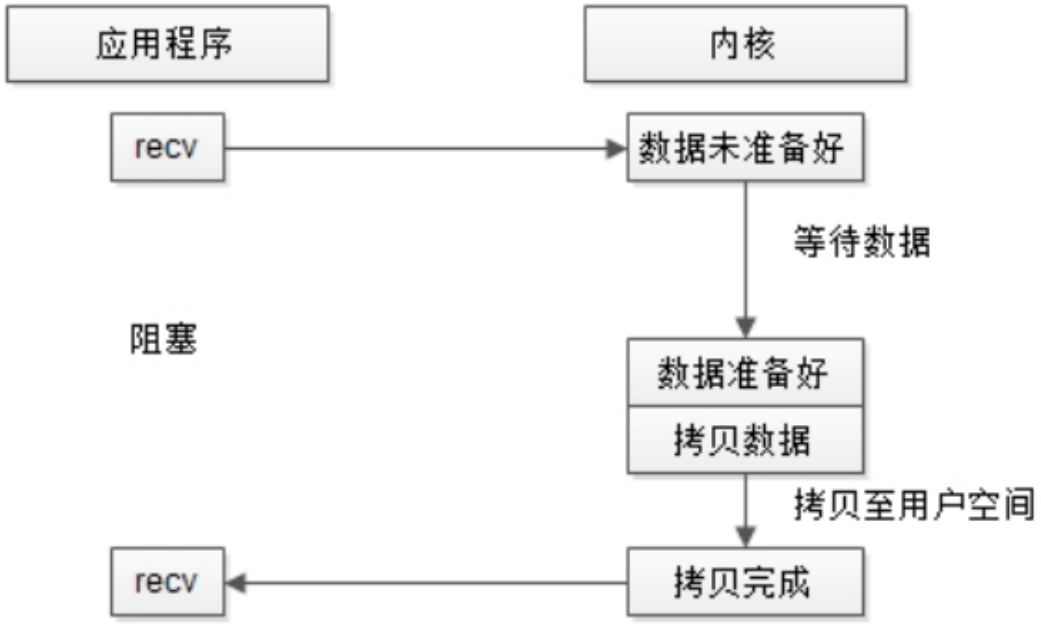
- **消息队列方案**
  - 通过消息队列
    - 缓存、持久化
    - 例子
      - 新用户注册请求
        - 第一步：用户名和密码写入数据库
        - 第二步：发送注册成功邮件

纯异步调用设计

- **如何异步调用**
  - 场景
    - I/O
      - 本地I/O、网络I/O
    - I/O模型
      - 阻塞I/O模型
      - 轮询非阻塞I/O模型
      - I/O复用模型

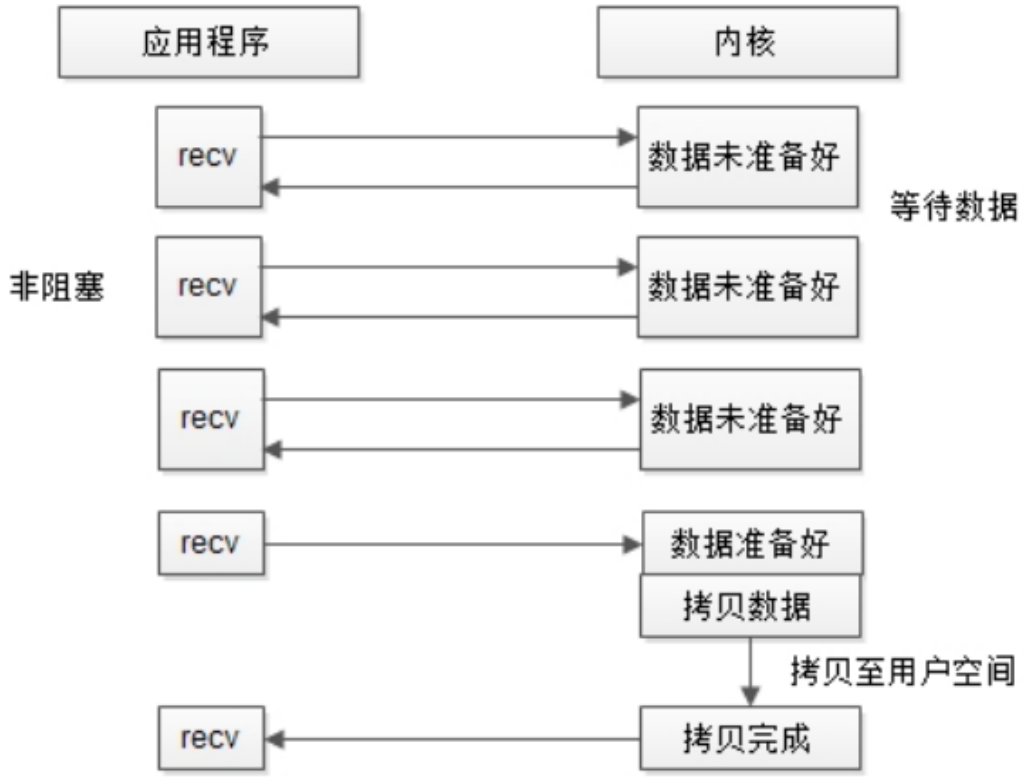
纯异步调用设计

- **阻塞I/O模型**



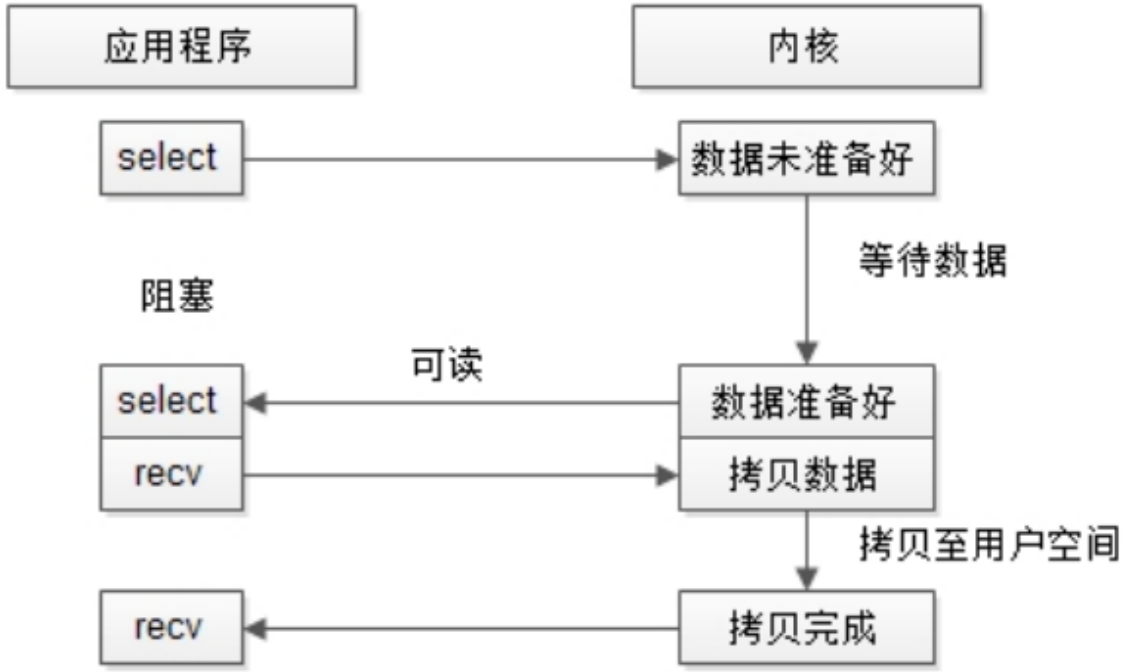
纯异步调用设计

- **轮询非阻塞I/O模型**



纯异步调用设计

- **I/O复用模型**
  - select
    - 1024
  - poll
  - epoll



纯异步调用设计

- **高性能纯异步网络调用设计**
  - server端连接池+server端收发队列
  - client端连接池+client端收发队列
  - 超时队列与超时管理器
  - 上下文管理器+状态机
- 案例里详细介绍

分级管理

- **硬件层面**
  - 核心系统使用好机器
  - 边缘系统使用差机器
- **部署层面**
  - 服务部署隔离
  - 核心系统部署在物理机
  - 核心系统部署不同机房
  - 边缘系统部署虚拟机
  - 边缘系统公用机器

分级管理

- **监控分级层面**
  - 核心服务更多类型的监控
    - 进程、语义、错误日志等
  - 监控粒度更细致
  - 邮件和短信发送通知
- **响应分级层面**
  - 核心服务开发、上线、运维响应、问题处理迅速

设置合理超时

- 下游服务宕机
- 线程死锁
- 下游服务忙
- .....

设置合理超时

- **请求超时设置根据请求平均响应延迟**
  - 上游超时时间一般设置下游平均响应延迟2倍，避免过长时间等待
  - 响应延迟高/低，超时时间设置长/短
    - 3s/100ms
  - 下游请求超时后，业务层根据预设的调度策略，继续重试
    - 一般3次
    - 多次无好处
    - 请求转移到下游不同服务

微服务降级-为什么需要

流量高峰期

短时请求量大

服务能力有限

性能下降

服务宕机

系统雪崩

怎么办 ➡ 微服务柔性可用

微服务降级-如何做

目标：保证核心服务可用；非核心服务弱可用，甚至不可用

微服务降级-系统降级

拒绝部分请求

关闭部分服务（业务紧密）

微服务降级-系统降级

拒绝部分请求

1 拒绝部分老请求

- ✓ 减轻微服务请求处理数量
- ✓ 确保“新”请求正常响应
- ✓ RPC队列方式（请求入队、出队时间处理请求时，检查请求在队列请求时间超过一定时间[比如1s]，直接丢弃）

2 优先级请求方式

- ✓ 非核心请求直接丢弃

（1）业务紧密

（2）转转

3 随机拒绝方式

- ✓ 随机丢弃一定比例请求
- ✓ 网站一会可用，一会不可用

微服务降级-数据层降级

01

更新请求

- ✓ 持久到消息队列
- ✓ 只更新缓存

02

读请求

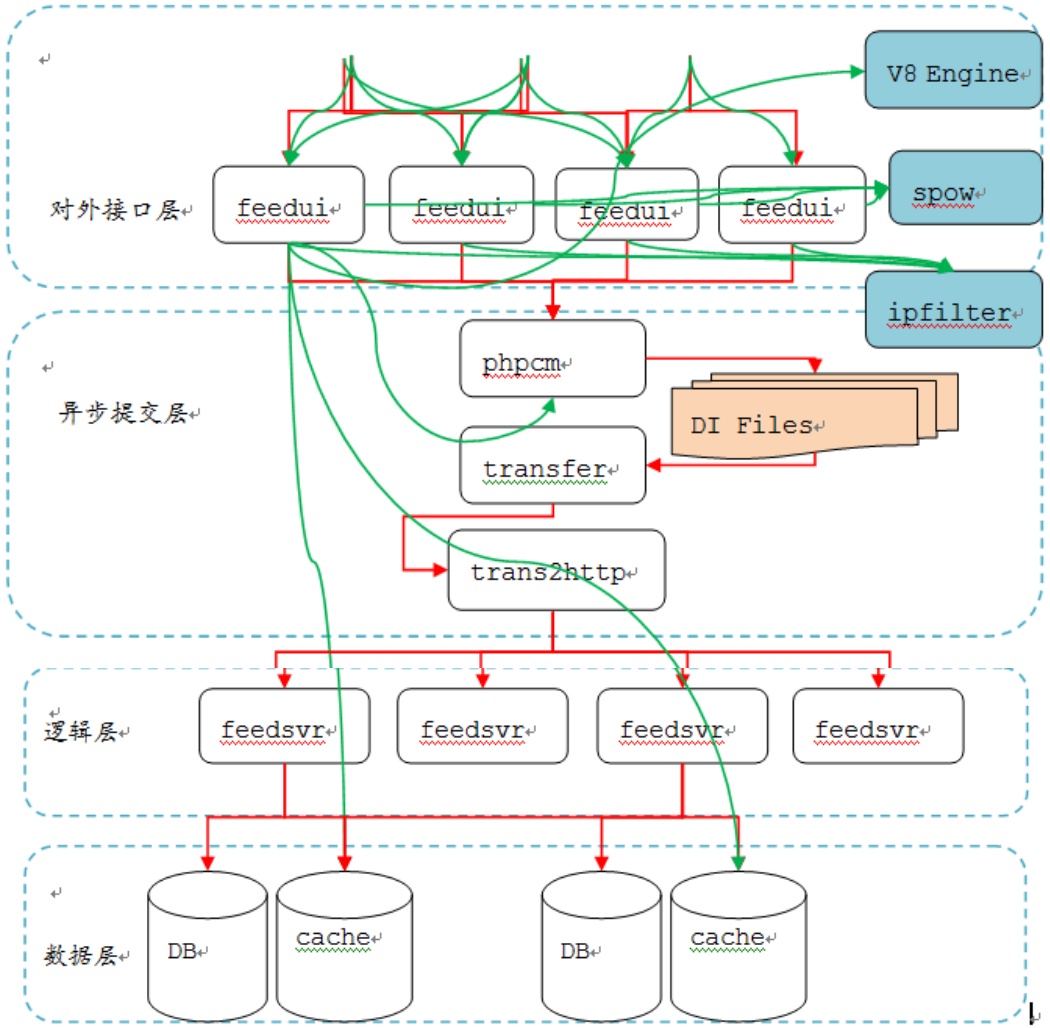
- ✓ 读缓存

03

数据补齐

- ✓ 消息队列->数据库

微服务降级-数据层降级


V8 Engine
对外接口层
feedui
feedui
feedui
feedui
spow
ipfilter
phpcm
异步提交层
DI Files
transfer
trans2http
逻辑层
feedsvr
feedsvr
feedsvr
feedsvr
数据层
DB
cache
DB
cache

服务降级-可用策略

自动打开

不依赖于人肉

演练

保证线上生效

幂等设计

- **幂等性**
  - 保证请求重复执行和执行一次结果相同
- **请求失败**
  - 重试
- **不保证幂等性**
  - 结果灾难性
    - 转账
    - 交易

幂等设计

- **数据四种操作**
  - Create
  - Update
  - Read
  - Delete

幂等设计

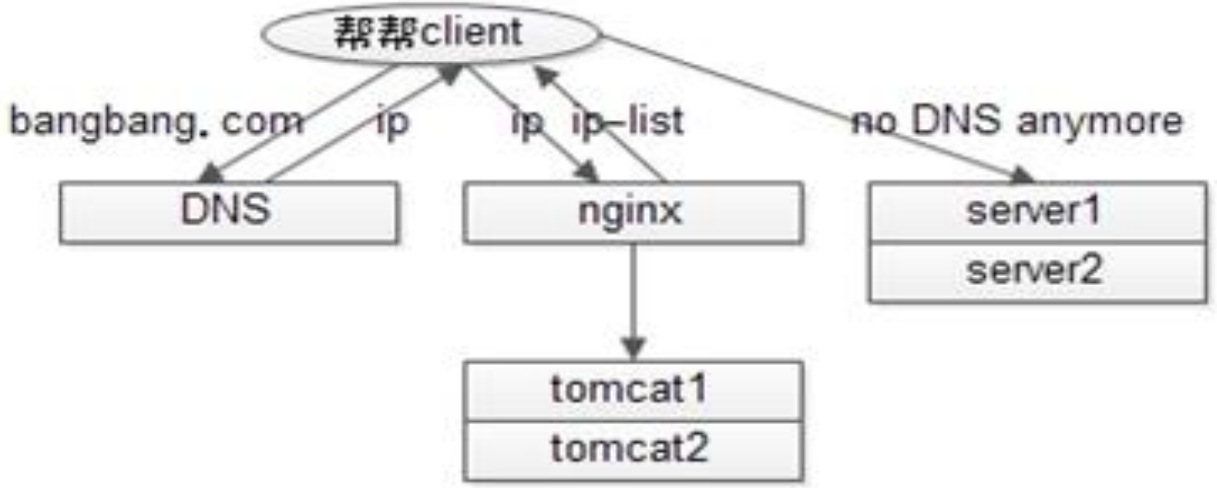
- **服务器幂等设计**
  - 天然幂等
    - QQ离线消息设置已读
  - 非幂等->幂等设计
    - 商品确认收货
      - 订单状态
      - 打款
      - 事务保证
        - 本地、分布式

移动时代微服务系统柔性可用

- **移动环境特点**
  - 网络
  - 流量
  - 电量
  - 安全
  - ……

移动时代微服务系统柔性可用

- **网络不稳定柔性可用**
  - **DNS优化**
    - App第一次访问，先拉取server iplist保存到App本地
    - 未来访问，App直接使用iplist中的IP来访问server，不再需要DNS
    - 如果DNS劫持或者不可用，直接使用iplist中的IP访问，达到柔性可用



移动时代微服务系统柔性可用

- **DNS优化**
  - APP负载均衡

APP随机访问iplist中IP

移动时代微服务系统柔性可用

- **DNS优化**
  - 后端水平扩展

直接在iplist中增加IP即可

移动时代微服务系统柔性可用

- **DNS优化**
  - 每次访问都要拉取iplist，废流量，优化方案

增加一个版本号，第一次拉取iplist，不但把iplist放到APP本地，并把版本号也拿到。未来每次先拿版本号，如果版本号不变，直接使用本地iplist。只有版本号变化时，才需要重新拉取iplist

移动时代微服务系统柔性可用

- **DNS优化**
  - 异构服务器负载均衡
    - 192.168.1.1服务能力为1，192.168.1.2服务能力为2，192.168.1.3服务能力为3
    - 方案一
      - 使用iplist，加上权重参数，实现异构服务器的负载均均衡，

        192.168.1.1， 10

        192.168.1.2， 20

        192.168.1.3， 30

移动时代微服务系统柔性可用

- **DNS优化**
  - 异构服务器负载均衡
    - 假设192.168.1.1服务能力为1，192.168.1.2服务能力为2,192.168.1.3服务能力为3。
    - **方案二**
      - IP重复书写多次

        192.168.1.1

        192.168.1.2

        192.168.1.2

        192.168.1.3

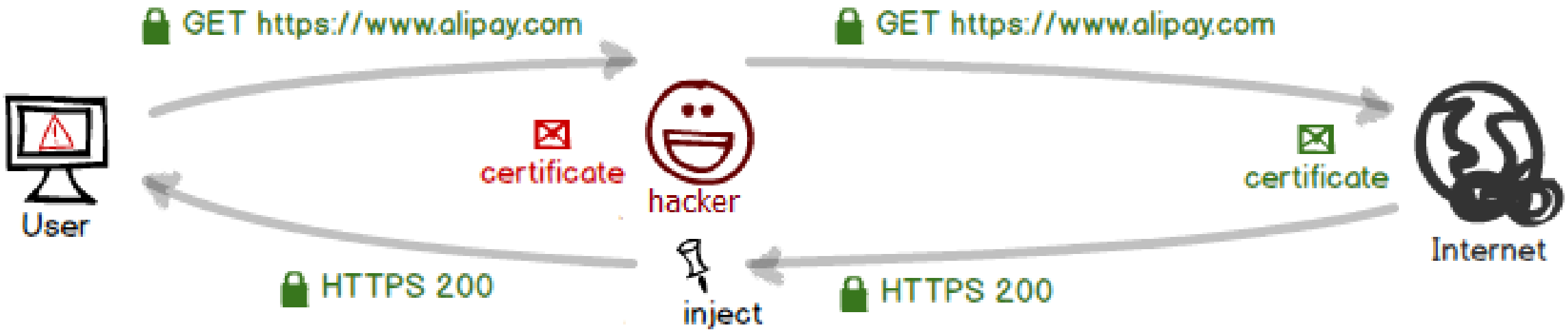
        192.168.1.3

        192.168.1.3

移动时代微服务系统柔性可用

移动时代微服务系统柔性可用

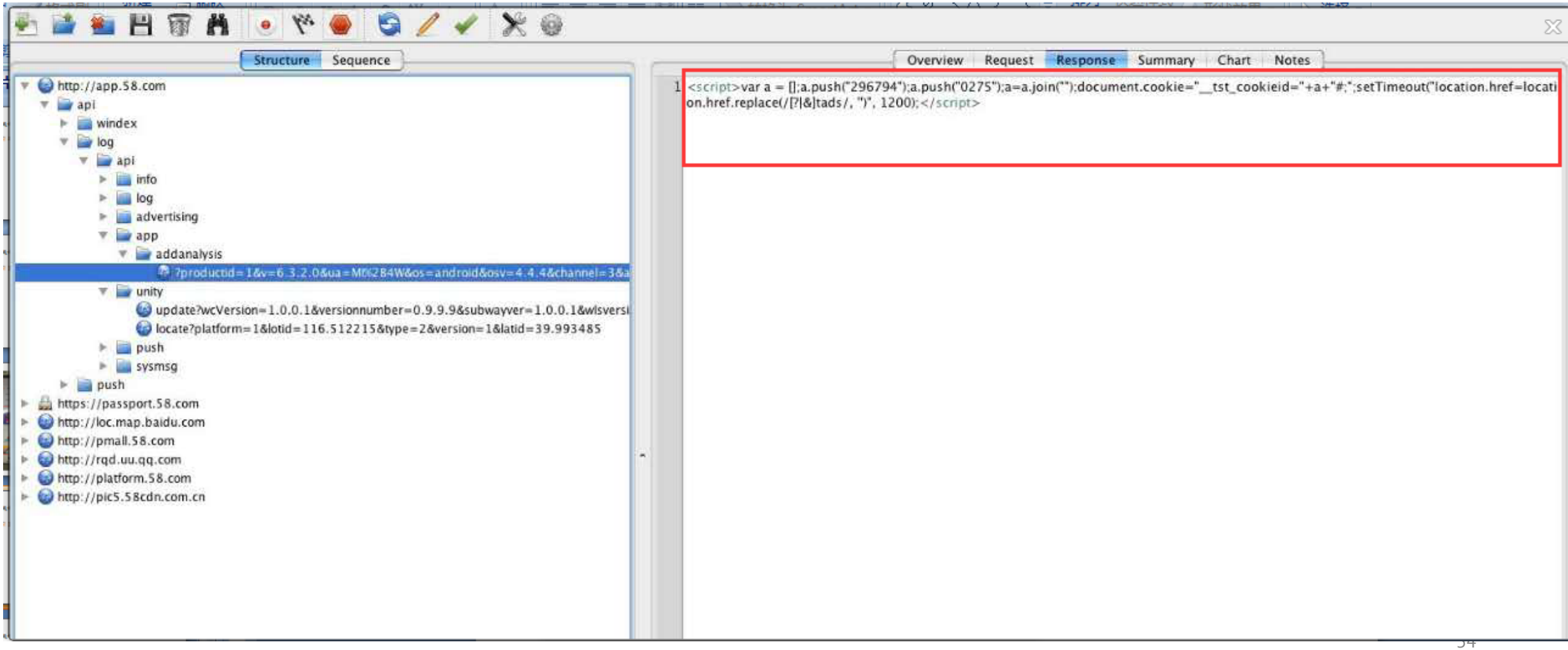
- **流量劫持**



移动时代微服务系统柔性可用

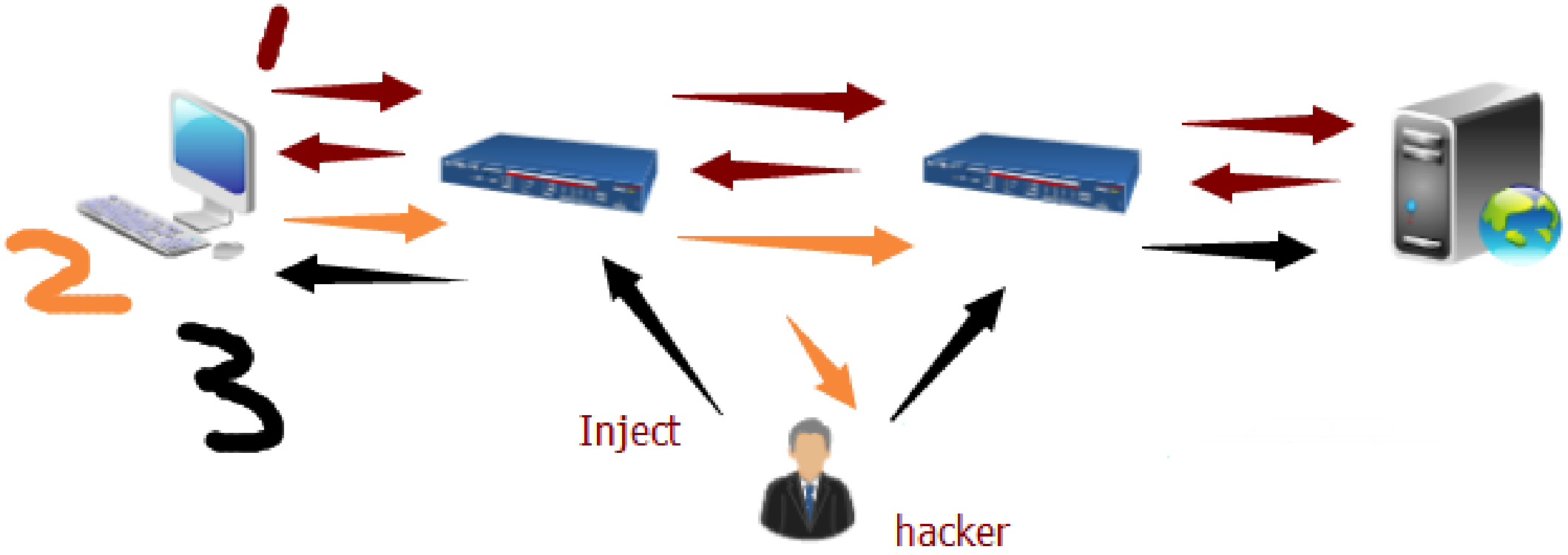
- **流量劫持**

移动时代微服务系统柔性可用



移动时代微服务系统柔性可用

- **链路劫持**



移动时代微服务系统柔性可用

- **链路劫持**

移动时代微服务系统柔性可用

- **流量劫持**
  - **解决方案**
    - **传输数据密文化**
      - HTTPS
        - 全站HTTPS
      - SSL/TLS
      - 自定义加密
        - RSA/AES

移动时代微服务系统柔性可用

- **链路劫持**

移动时代微服务系统柔性可用

- **网络不稳定柔性可用**
  - **APP登录优化**
    - **APP端快速重连**
      - 第一次走全部流程
        - 加密过程
      - 后续置后台
        - 快速重连

移动时代微服务系统柔性可用

- **网络不稳定柔性可用**
  - **push推送优化**
    - **移动特点**
      - TCP长连接+push推送
      - 优先TCP长连接
      - 离线push推送
    - **解决**
      - 弱网环境下消息到达的问题
      - 通道优先级解决消息实时性

移动时代微服务系统柔性可用

- **流量、电量优化柔性可用**
  - **数据拉取优化**
    - **典型业务场景**
      - 移动IM好友信息
      - 变动频率不高
      - 每次从移动后台拉取，流量、电量浪费
      - 有必要做到针对性拉取

移动时代微服务系统柔性可用

- **流量、电量优化柔性可用**
  - **数据拉取优化**
    - **解决方案**
      - 时间戳机制
        - 每次有数据更新移动后台更新时间戳
        - App拉取时间戳并存储本地
        - 每次上线从移动后台拉取时间戳，并比对
        - 如果没变化，不需要拉取实际数据
        - 有变化，针对性拉取一次数据

移动时代微服务系统柔性可用

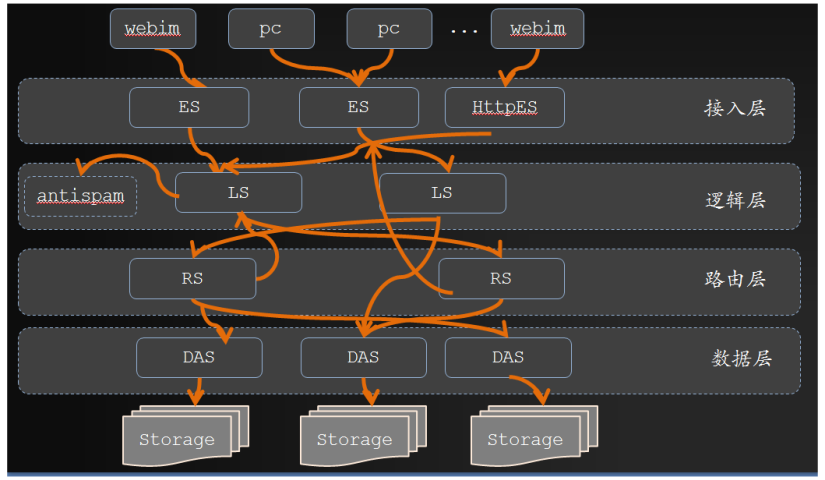
- **流量、电量优化柔性可用**
  - **实时&延迟拉取优化**
    - **数据优先级不同**
      - 高/中/低
    - **解决方案**
      - 实时拉取优先级高数据
        - 实时拉取
      - 延迟拉取中/低数据
        - APP判断网络环境/WIFI环境拉取

移动时代微服务系统柔性可用

- **流量、电量优化柔性可用**
  - **协议优化**
    - **典型业务场景**
      - 埋点、日志上报
    - **解决方案**
      - 协议字段尽可能短小
      - 批量合并请求
        - HTTP头占比大
      - 数据压缩
        - 请求数据->tomcat->gzip

实践案例

- **58帮帮业务场景（加好友）**
  - IM的加好友服务，用户A将好友添加到一个分组中
    - 拉取好友列表
    - 拉取分组
    - 检查是否符合antispam策略，
    - 拉B的加好友策略，看是否需要验证
    - ……
  - 列举的步骤需要帮帮聚合层访问下游服务



实践案例

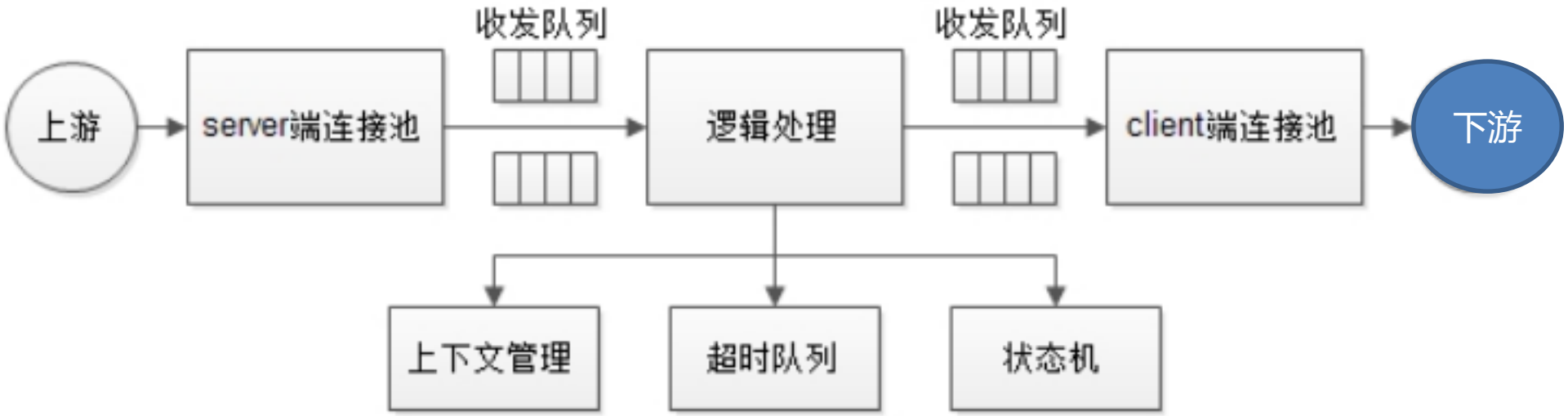
- **58帮帮业务场景（加好友）**
  - 业务复杂
  - 涉及多个下游模块
  - 需要和多个下游模块网络交换
  - 需要高性能、高可用
    - 网络异步模型
    - 聚合层无状态
    - 冗余部署
    - 动态扩展

实践案例

- **名词解释**
  - **客户端**
    - 网络通信中主动发起通信的一端
  - **服务端**
    - 网络通信中被动通信的一端，为客户端提供服务器
  - **异步通信**
    - 客户端在于服务器通信的过程中可以并发的发送多个请求，而不用没发送一个请求就停下来等待服务器的响应，收到服务端的响应后底层通过某种机制通知上层应用（比如函数回调）

实践案例

- **58帮帮业务场景（加好友）**
  - **框架结构**



实践案例

- **58帮帮业务场景（加好友）**
  - 进程启动下游连接加入主动连接池
  - 建立上游客户端连接，并加入被动连接池
  - 上游客户端数据放入接收队列
  - 工作线程开始处理
  - 发送到下游服务器，请求到客户端的发送队列，并加入超时队列
  - 收到下游服务端响应，请求接入接收队列，删除超时队列，通知上层回调
  - 下游超时，删除，并回调
  - 响应加入到服务端发送队列，回复给客户端

实践案例

- **58帮帮业务场景（加好友）**
  - 超时管理器
    - 发送下游包的超时管理
    - 避免无限等待
    - 单独线程
    - 定时扫描
    - 超时处理

实践案例

- **58帮帮业务场景（加好友）**
  - 上下文管理器
    - 请求上下文
    - package_key
    - 请求的唯一标示
    - 超时等删除上下文

实践案例

- **58帮帮业务场景（加好友）**
  - 状态机管理器
    - 异步调用的状态机
    - 标志请求的状态
    - 串行执行的状态机

要点回顾

微服务架构

1. 微服务聚合层设计
2. 无状态化&纯异步设计
3. 分级&超时设计
4. 服务降级
5. 幂等设计
6. 案例实践

欢迎关注本人公众号 “架构之美”

Thanks!

让 生 活 更 简 单