唯品会微服务架构演进之路

杨钦民

唯品会企业/应用架构部架构师

唯品会
全球精选 正品特卖

自我介绍

- 8年架构及技术团队管理经验，现任唯品会企业/应用架构部架构师，负责运营平台、电商平台核心交易架构规划、“419品牌特卖节”大促、“616年中特卖”大促等架构；
- 历任贝聊技术总监，网易高级工程师；
- 国内顶级技术大会ArchSummit全球架构师峰会讲师、网易云易经布道讲师。

唯品会微服务架构体系总体介绍

- 唯品会是业界在服务化走得比较彻底的一家公司，主要体现在两方面，一是在彻底落地，二是纯自研。从2015年开始建设至今，经过4个年头的打磨，微服务基础中台已经成为唯品会最为重要的技术基础设施之一，在唯品会所有关键业务场景全面深入落地，并承载着几乎全部用户流量的请求处理和后端服务调用，在体系内注册的独立服务数超过3000，独立方法数万级别，高峰时每小时承载着千亿次服务调用，服务化网关作为流量入口承载着百万级别 QPS。

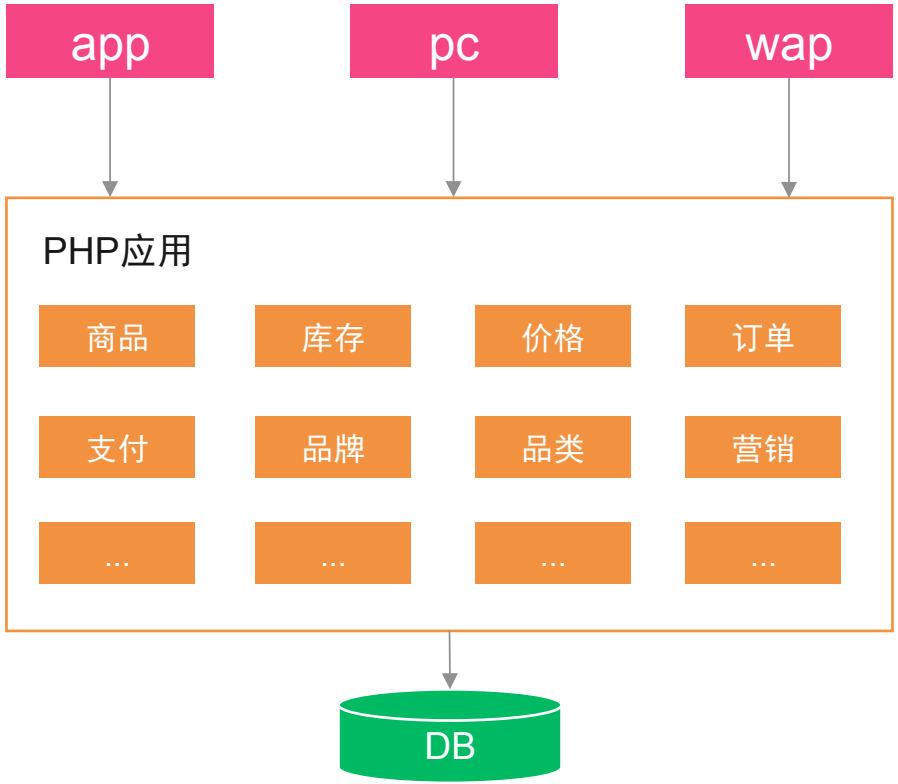
- 围绕微服务框架体系，自研了一系列的微服务基础中台，包括OSP微服务框架、服务治理中心、配置中心、全链路监控、API 网关、服务安全管理、分布式协调等，同时基于 Kubernetes+Docker 技术框架研发了云平台，涵盖应用的开发、交付、运维和运营全生命周期，OSP独创性的 proxy 代理架构，将治理能力完全从客户端剥离，并结合容器云平台迅速演进为 Service Mesh 部署架构。

目录
CONTENTS

01 微服务架构演进

单体架构

- 架构简单，采用LAMP架构
- 单个应用，所有功能全部耦合在一个应用中
- 单个DB，所有表全部维护在一个DB中
- 可以快速响应业务研发需求，快速开发、快速上线

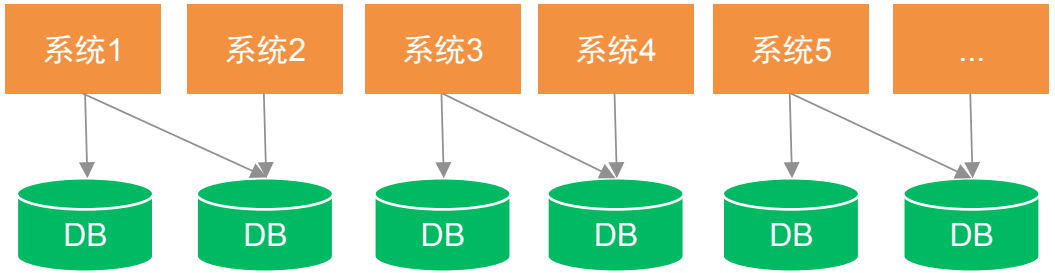


单体架构—面临的问题

- 所有业务代码耦合在一个应用，随着业务的发展，代码越来越臃肿
- 随着开发团队的壮大，团队成员技术水平参差不齐，缺乏统一的代码规范，整个应用代码越来越乱
- 新成员加入之后，由于业务代码耦合在一起，需要阅读过多不是自己负责的业务代码，花费过多时间才能融入
- 经常出现一个慢SQL，影响整个DB性能，甚至影响整个DB挂掉，影响全部业务都不能运转
- 各种业务代码耦合在一起，开发效率低，每次测试以及上线发版，均需耗费过多时间

垂直应用架构

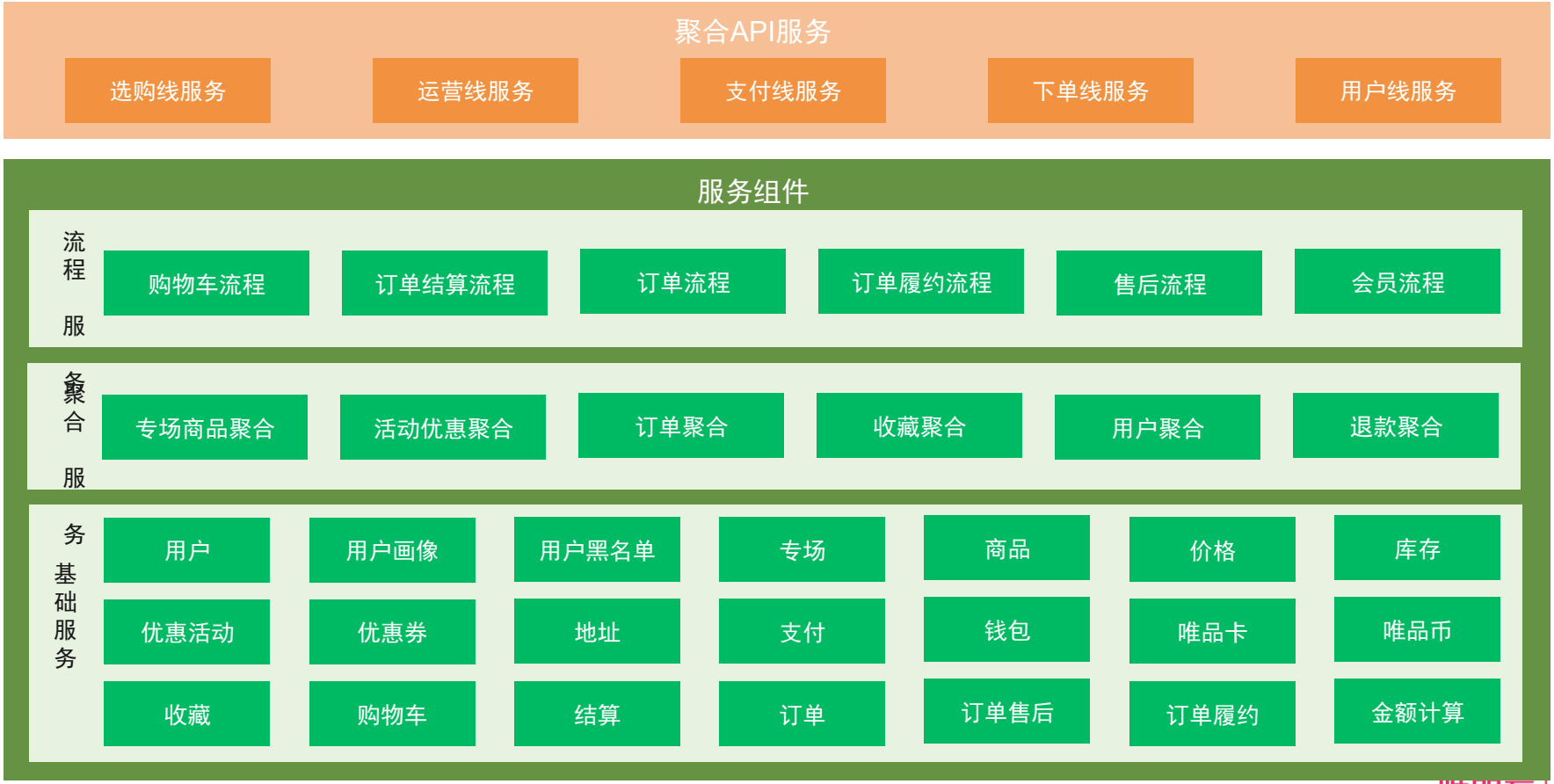
- 从单体应用按照业务垂直拆分出多个应用
- 各个应用独立部署
- 每个应用拥有独立的DB
- 每个系统拥有独立域名



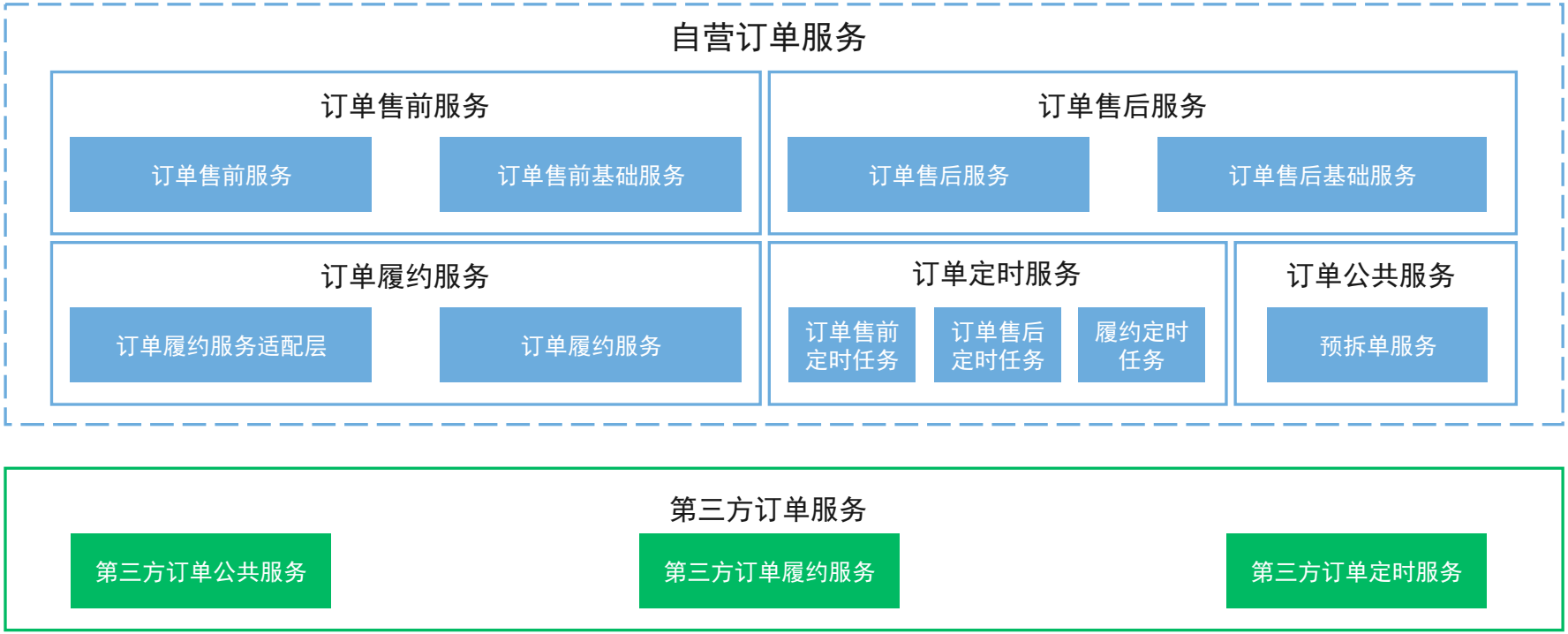
垂直应用架构—面临的问题

- 虽然按照业务线拆分系统，但拆分粒度过大，业务耦合依然严重
- 不同应用依然存在访问一个DB的情况，数据表拆分不彻底
- 业务团队各自为战，经常出现抢地盘，重复造轮子，浪费开发资源，团队协作效率差
- 多个业务系统边界不清晰

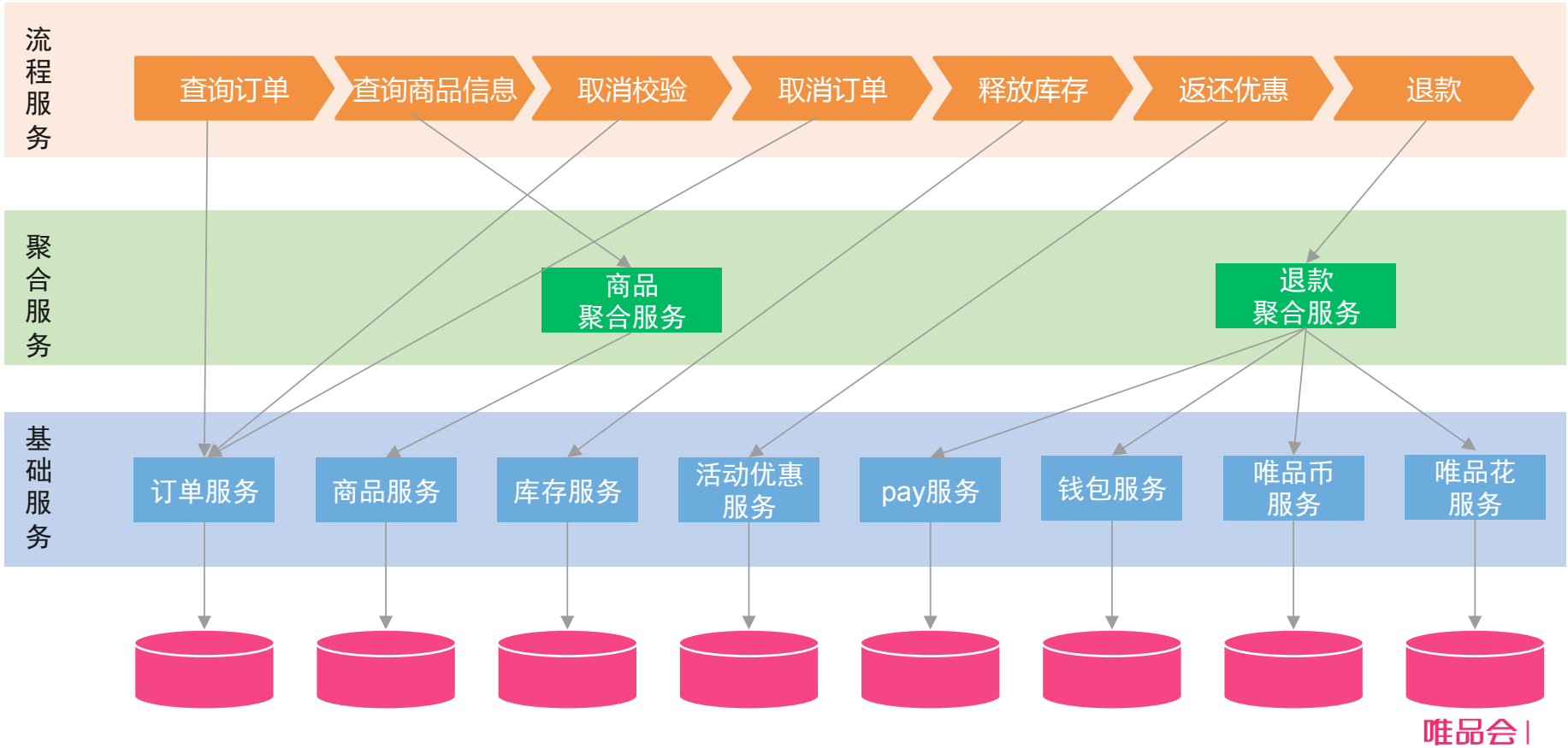
微服务架构—电商服务化架构



订单服务化


自营订单服务
订单售前服务
订单售前服务
订单售前基础服务
订单售后服务
订单售后服务
订单售后基础服务
订单履约服务
订单履约服务适配层
订单履约服务
订单定时服务
订单售前定时任务
订单售后定时任务
履约定时任务
订单公共服务
预拆单服务
第三方订单服务
第三方订单公共服务
第三方订单履约服务
第三方订单定时服务

订单服务化—取消订单相关服务交互



服务能力开放共建生态

开放平台

供应商服务

- 地址库服务
- 品牌库服务
- 类目服务
- 商品服务
- 尺码服务
- 门店服务
- 库存服务
- 价格服务
- 订单服务

仓库和物流服务

- 物流服务
- 海淘订单服务
- 仓库服务
- 包裹交接服务
- 海外代运营服务
- HTS货代推送服务

多渠道服务

- 多渠道商品服务
- 多渠道库存服务
- 多渠道售后服务
- 多渠道运费服务
- 多渠道地址服务
- 多渠道订单服务

市场和渠道服务

- 唯品卡服务
- 用户群数据服务
- 渠道选品服务
- 渠道商品服务
- 渠道库存服务
- 渠道营销库存服务

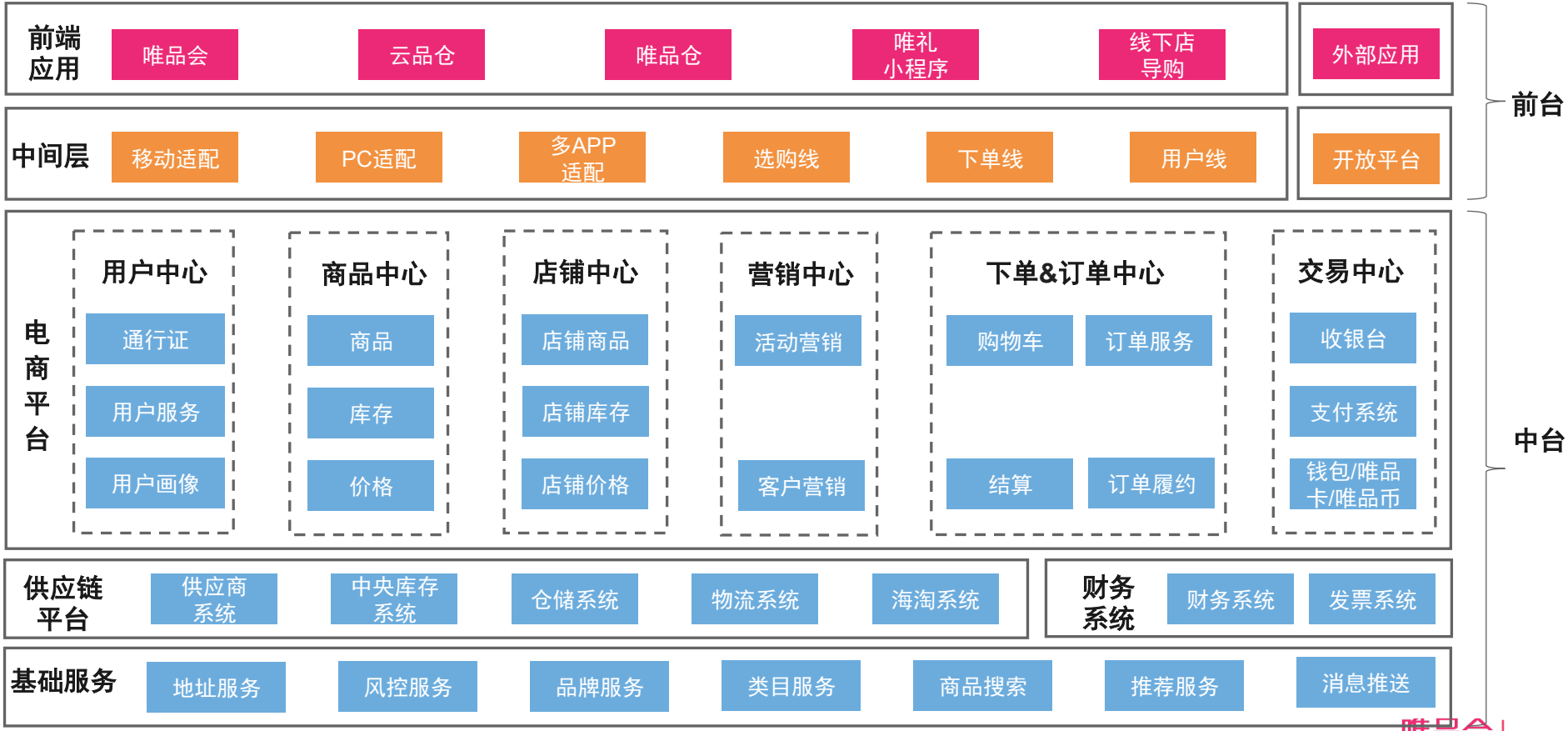
店铺服务

- 店铺商品服务
- 店铺商品运营服务
- 店铺服务
- 店铺品牌服务
- 店铺类目服务
- 店铺商品库存服务
- 店铺尺码服务
- 店铺商品价格服务
- 发票服务

其他服务

- 电子发票服务
- 授权服务
- 商品扩展服务

微服务整体架构


前端应用
唯品会
云品仓
唯品仓
唯礼小程序
线下店导购
外部应用
中间层
移动适配
PC适配
多APP适配
选购线
下单线
用户线
开放平台
前台
电商平台
用户中心
通行证
用户服务
用户画像
商品中心
商品
库存
价格
店铺中心
店铺商品
店铺库存
店铺价格
营销中心
活动营销
客户营销
下单&订单中心
购物车
订单服务
结算
订单履约
交易中心
收银台
支付系统
钱包/唯品卡/唯品币
中台
供应链平台
供应商系统
中央库存系统
仓储系统
物流系统
海淘系统
财务系统
财务系统
发票系统
基础服务
地址服务
风控服务
品牌服务
类目服务
商品搜索
推荐服务
消息推送

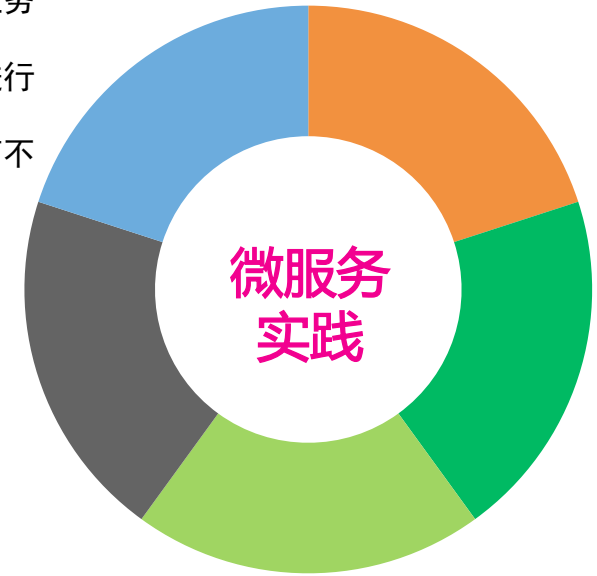
微服务架构—最佳实践

业务驱动原则

- 识别核心业务域，形成基础业务能力
- 根据业务定位、范围、边界进行服务的划分
- 首先关注服务的业务范围，而不是服务的数量、粒度

服务分层原则

- 划分基础、聚合、流程服务
- 基础服务贴近业务实体，提供业务的基础能力
- 聚合服务聚合基本业务场景，满足高一层业务场景并可复用
- 流程服务面向复杂业务流程实现，通过驱动多个聚合/基础服务实现一个完整的业务流程



服务松耦合原则

- 服务职责单一，一个服务聚焦在特定业务的有限范围内，有助于敏捷开发和独立发布
- 区分核心业务服务和非主核心业务服务
- 区分稳定服务和易变服务
- 每个服务只能访问自己的数据

服务独立部署原则

- 服务独立部署，能够独立发布或取消发布
- 服务可水平扩展，并支持单独扩展
- 实现持续集成和自动发布
- 实现服务的技术和业务监控

兼容性原则

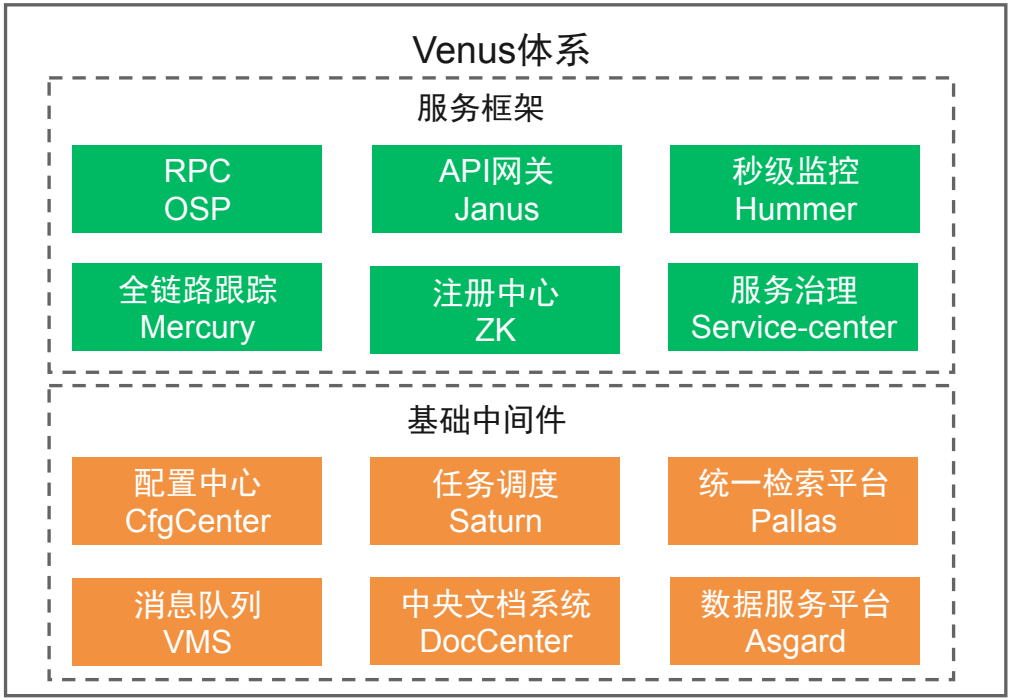
- 接口契约先行，提供最新在线服务文档
- 服务版本管理，保证向前兼容

02 微服务基础中台建设

微服务基础中台

Venus体系—服务框架：提供服务注册、发现、治理、全链路跟踪、监控、安全授权管理

Venus体系—基础中间件：提供分布式配置管理、分布式任务调度、统一检索平台、消息队列、OSP服务接口文档自动生成系统、统一数据服务管理平台



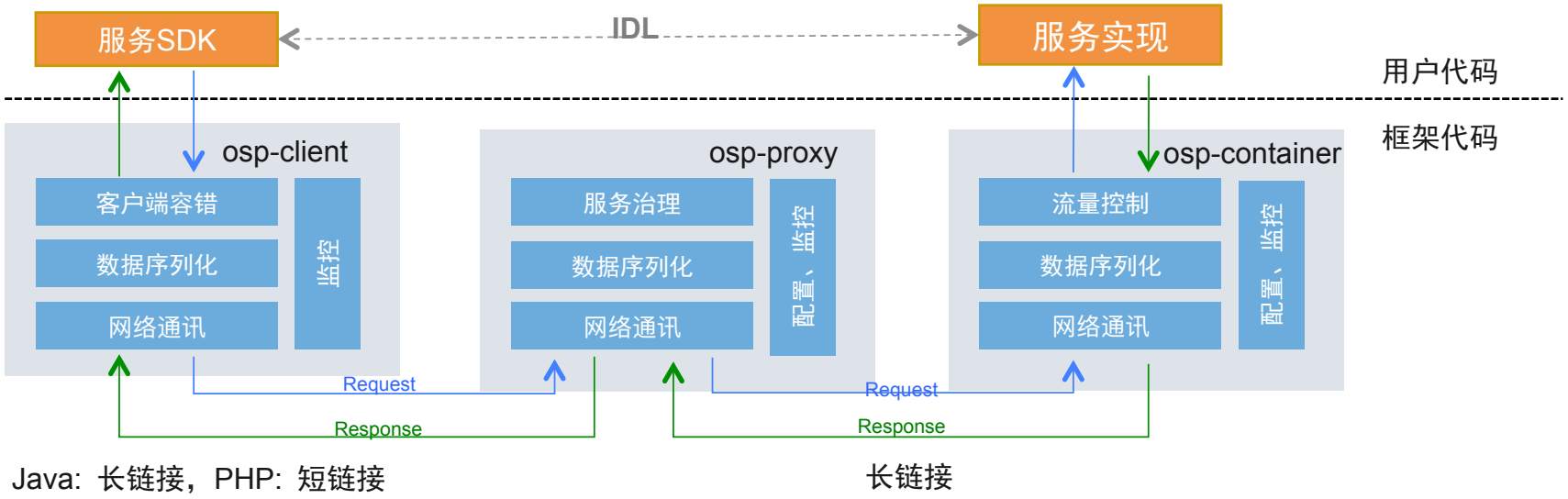
OSP服务化远程调用机制

高性能可扩展 RPC框架

契约化多语言 服务接口

高可靠性 基础设施

统一服务 治理平台



OSP高性能服务化框架优势

高性能

- 基于Thrift与Netty
- 单机数万QPS
- 高效二进制协议
- TCP长连接，异步传输，流式处理
- 海量并发连接

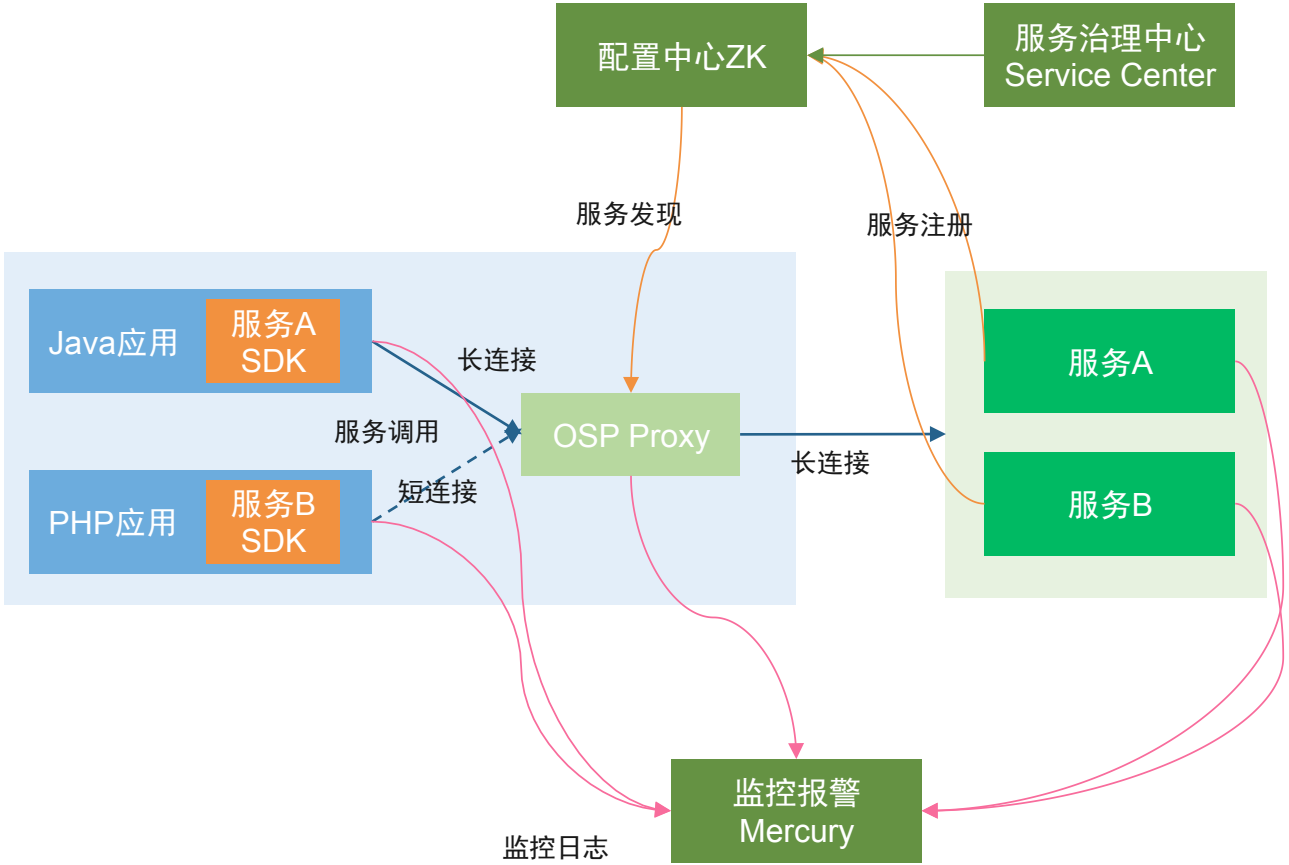
高可扩展

- 协议无状态
- 服务的自动注册与发现
- 不依赖中心化的Load Balancer

OSP服务化架构

客户端与Proxy部署在同一台机器的不同进程

通过服务治理中心管理安全、服务路由、服务熔断等



全链路监控Mercury功能介绍

监控告警

- ✓ 监控大盘快速展现系统问题
- ✓ 秒级实时告警第一时间报告critical issues
- ✓ 分钟级准实时告警周期性检测指标规则
- ✓ 从告警事件快速定位根源问题
- ✓ 灵活方便的多级告警策略定义
- ✓ Pigeon中央告警平台无缝集成

指标统计

- ✓ 域 / 主机 / 服务API级别指标展示
- ✓ 每秒请求数
- ✓ 响应时间
- ✓ 请求失败率(4xx/5xx/osp failed)
- ✓ 异常发生率
- ✓ SQL性能指标
- ✓ 拓扑依赖关系和性能指标

调用链跟踪

- ✓ 调用链检索（通过业务关键字）
- ✓ 慢调用查询
- ✓ 失败调用查询（4xx / 5xx）
- ✓ 调用链详情展示

全链路监控Mercury核心价值

IT运维 / 监控中心人员

- 快速故障告警和问题定位
- 把握应用性能和容量评估
- 提供可追溯的性能数据

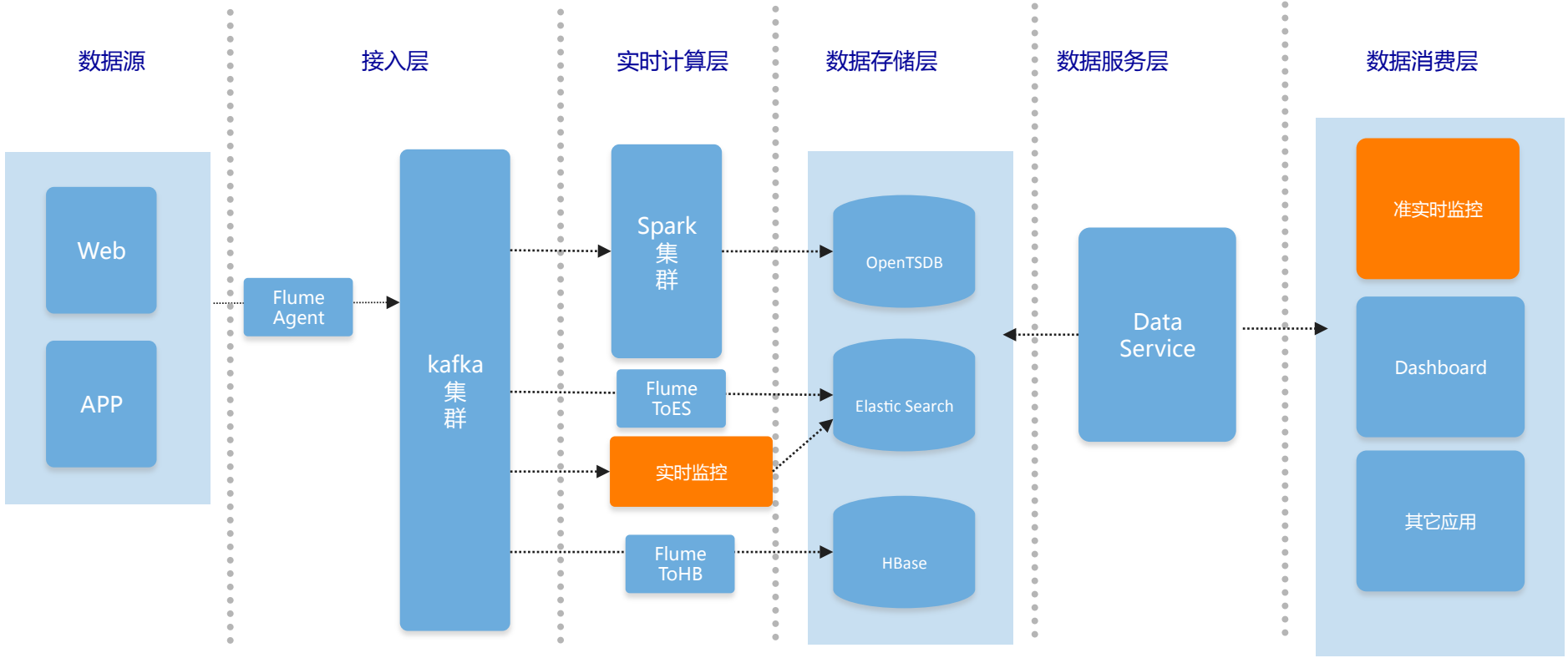
应用开发人员

- 定位线上服务性能瓶颈
- 持续优化代码和SQL
- 帮助快速解决线上问题

应用管理人员

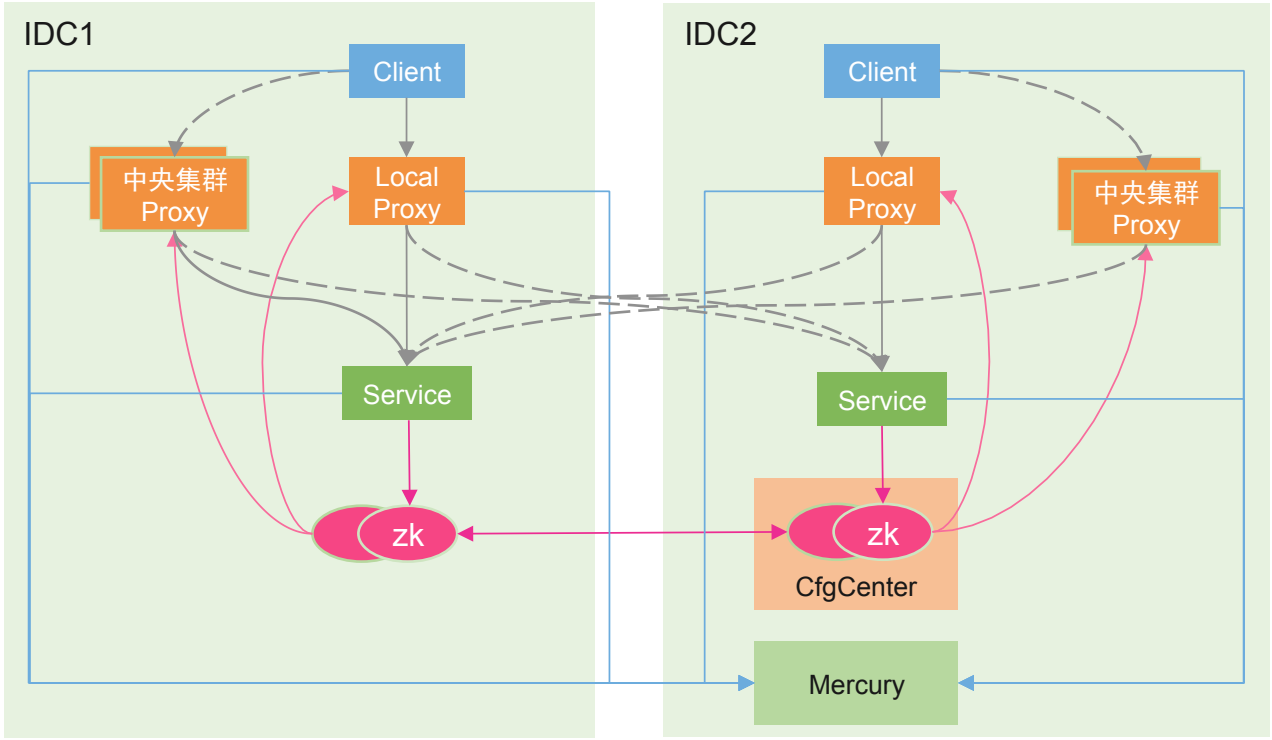
- 全方位把握应用整体拓扑结构
- 定位全网应用瓶颈
- 帮助优化关键业务

全链路监控Mercury架构



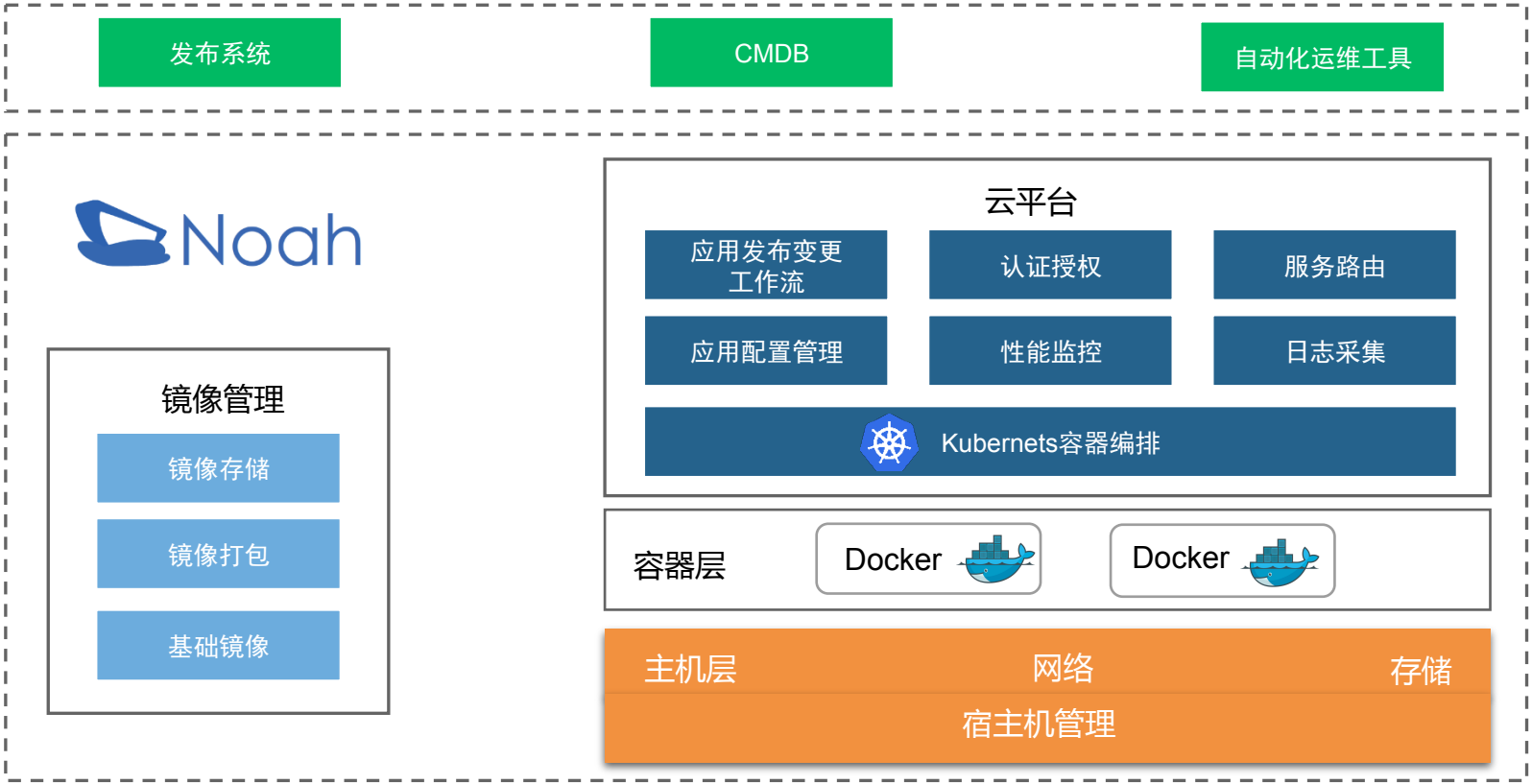
微服务多IDC部署架构

- 服务实例支持多机房部署，客户端通过proxy优先调用本机房服务实例，当本机房服务实例出现故障，可以跨机房调用其他机房服务实例；
- Proxy分为Local Proxy和中央集群Proxy，客户端默认使用本地Local Proxy调用服务实例，当本地Local Proxy出现问题，则通过中央集群Proxy调用服务实例，即通过Local Proxy和中央集群Proxy实现主备；
- 不同机房注册中心实现同步注册信息
- 配置中心CfgCenter以及全链路监控Mercury只部署在一个机房



03 基于Kubernetes、Docker打造云平台

Noah云平台总体架构

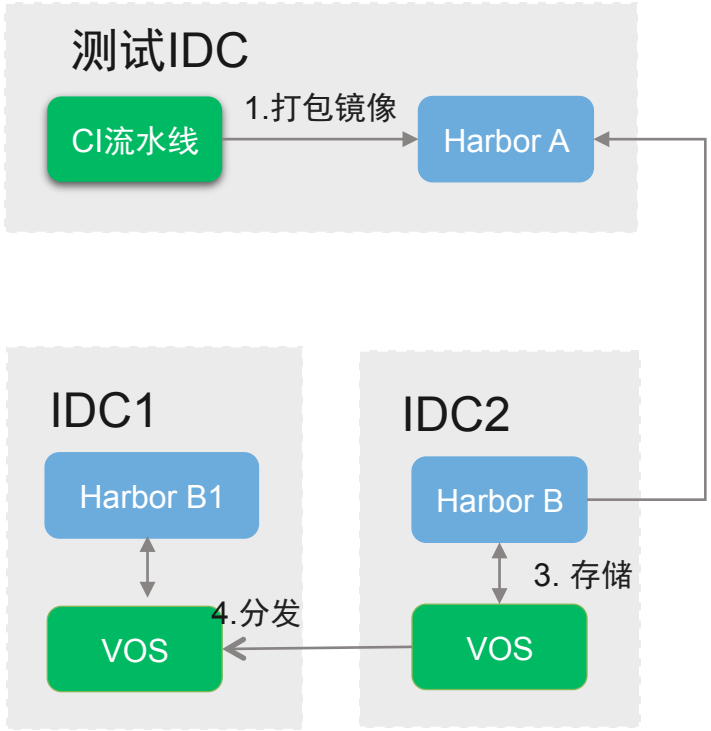


Noah云平台：提供Docker容器配置、镜像管理、kubernetes容器编排、管理后台等

Noah容器云镜像存储以及分发

定制版的Harbor（开源镜像仓库）
Docker 镜像仓库接口
Harbor UI

VOS（VIP Object Storage）
海量，高可用的镜像存储
多机房分发



Noah容器云Kubernetes容器编排

节点选择

- 剩余的CPU / 内存
- 同域 ”尽量” 跨机器，跨机架部署

高可用保证

- 端口 / Health Check URL 定时探测
- 探测失败，同节点重建实例
- 节点失效，新节点重建实例

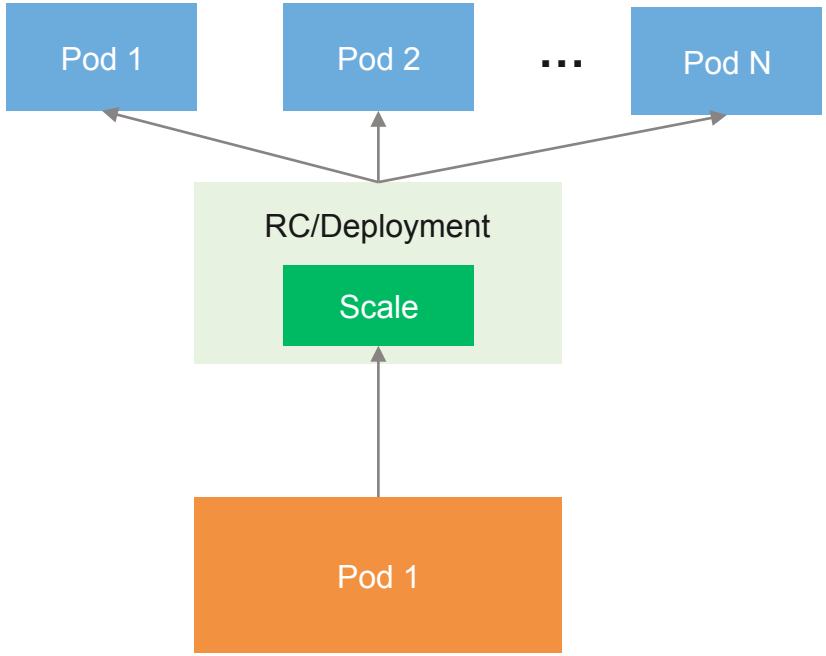
Noah容器云自动扩缩容（HPA算法）

TargetNumOfPods = ceil(sum(CurrentPodsCPUUtilization) / Target)

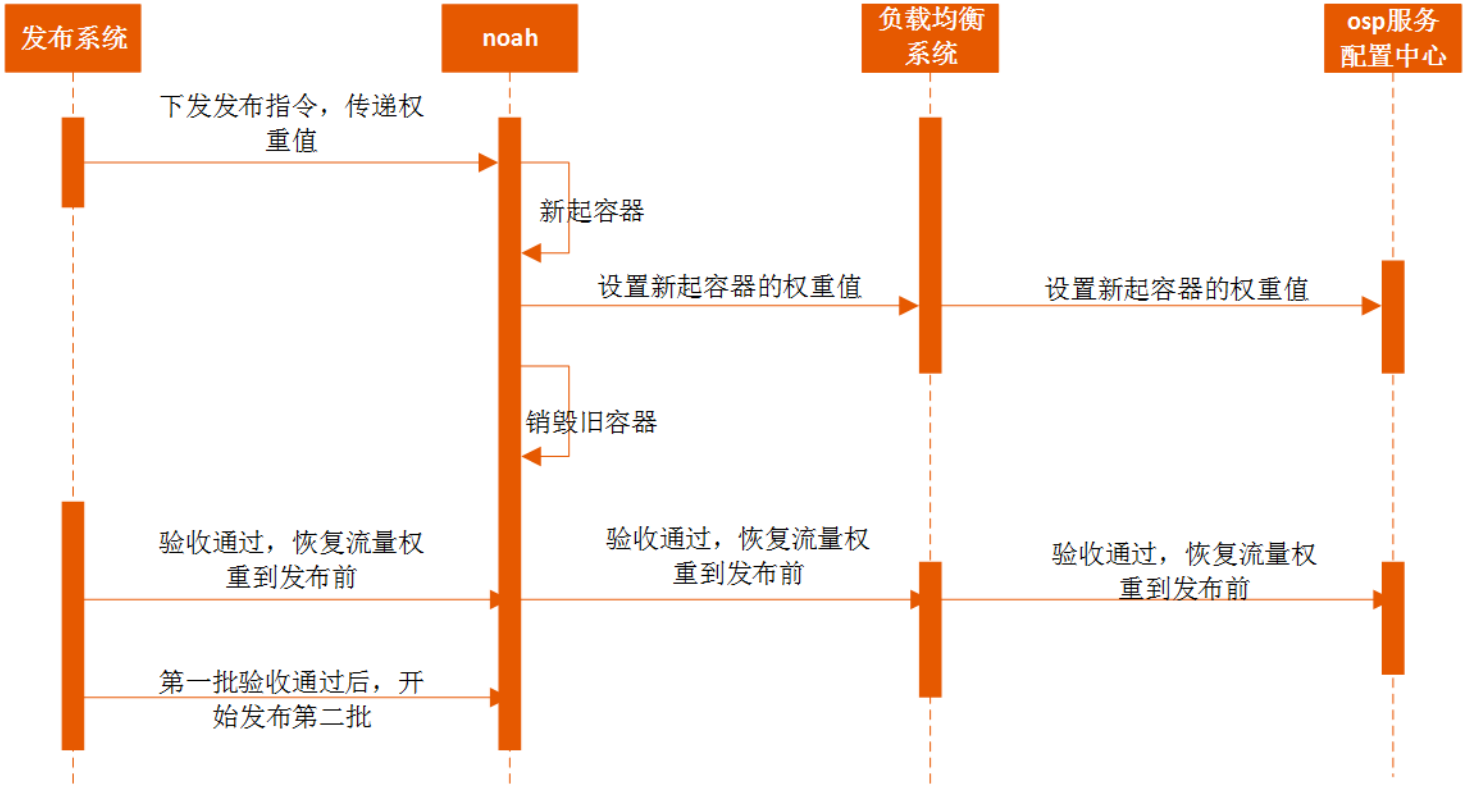
参照Kubernetes的HPA（Horizontal Pod Autoscaling）算法，自研发实现自动扩容缩容

CurrentPodsCPUUtilization为当前每台pod的CPU使用率，Target为定义的应用CPU使用率指标

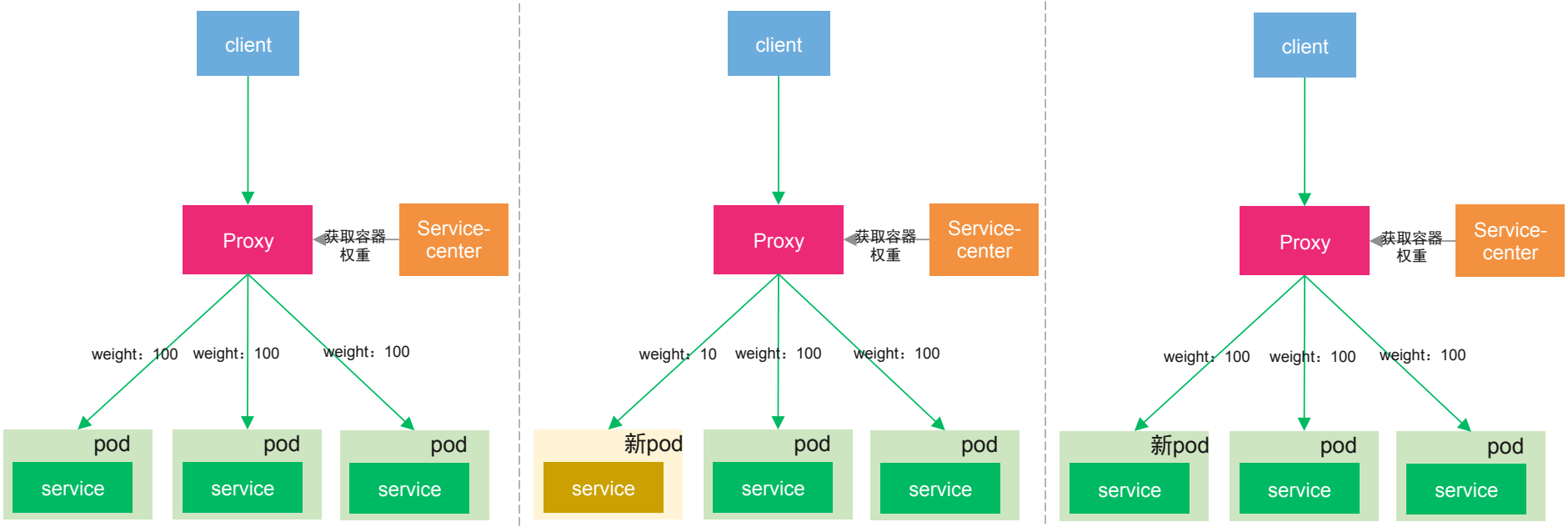
假设Target 定为为50%，当前容器CPU使用率的平均值为60%，有10个容器实例，则CurrentPodsCPUUtilization=60%*10=6，则TargetNumOfPods=ceil（6/0.5）=12，则需要扩容12-10=2台容器



Noah容器金丝雀发布—发布流程



Noah容器金丝雀发布—灰度发布



- 灰度发布前，三台容器权重各为100，流量占比分别为33.3%
- 灰度发布一台容器，新容器的权重设为10，则三台容器的流量占比分别为4.8%、47.6%、47.6%
- 灰度发布后，恢复到发布前的权重，即新容器权重重新设置为100，则三台容器的流量占比分别为33.3%
- 通过灰度发布设置容器权重，可以将流量的百分之一控制到一台机器

04 Service Mesh架构

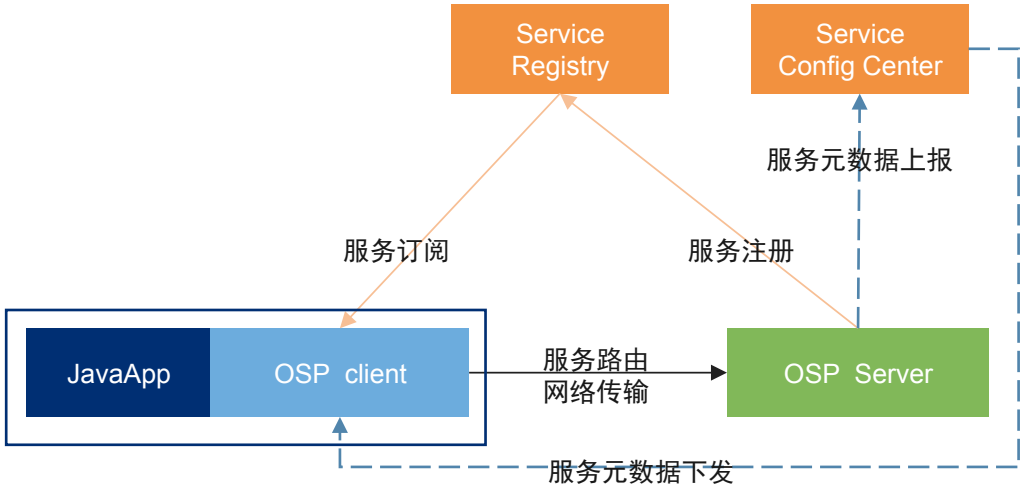
服务化体系诞生

特点

- OSP（Open Service Platform）
- Thrift over Netty
- 基于Java语法的DSL
- 注册中心Zookeeper
- 胖客户端（业务代码与OSP框架绑定在一起）
- 基本服务治理能力

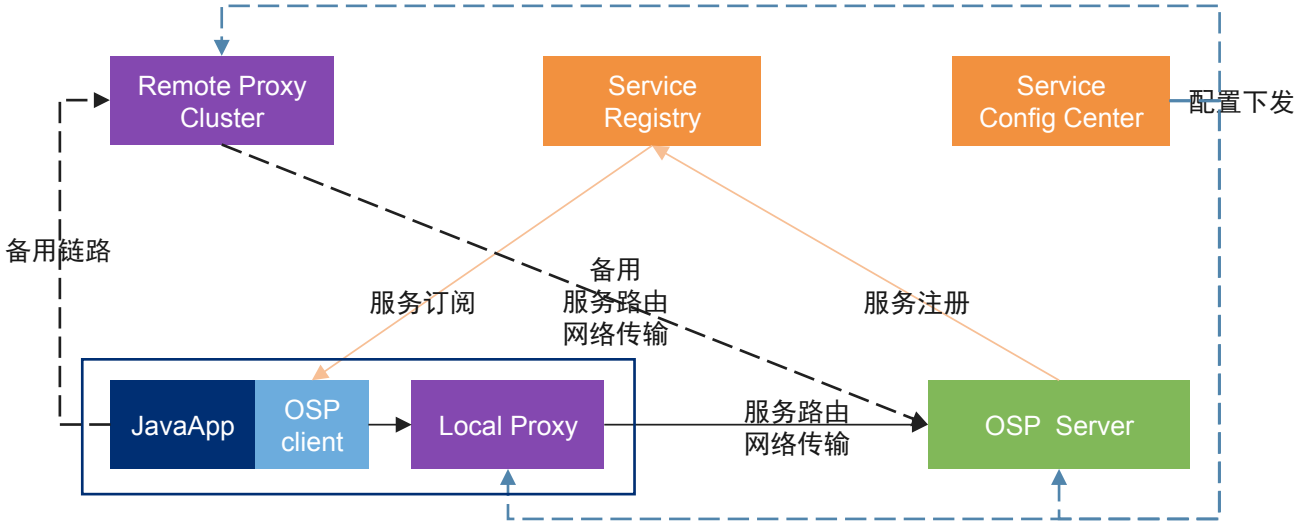
缺点

- 语言单一，只支持Java
- 业务代码与OSP框架绑定在一起，升级新版本困难，不利于框架演进
- 复杂框架代码嵌入对业务进程影响大



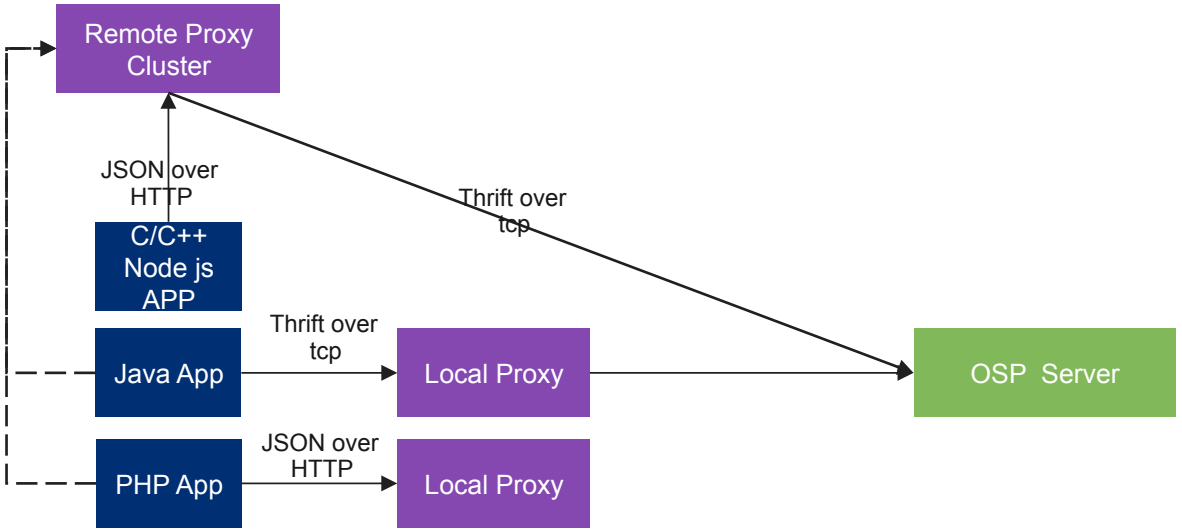
服务化体系进化—Service Mesh架构雏形

- 客户端Sidecar
- 本地代理以及备份中央代理集群，主与备
- 轻量级代理客户端，本地调用，服务治理从业务代码抽离到Proxy
- 由Local Proxy负责服务治理与远程通信
- Remote Proxy负责备份和非主流流量
- 配置信息统一在Service Config Center进行配置，并进行配置下发



Service Mesh架构—多语言客户端接入

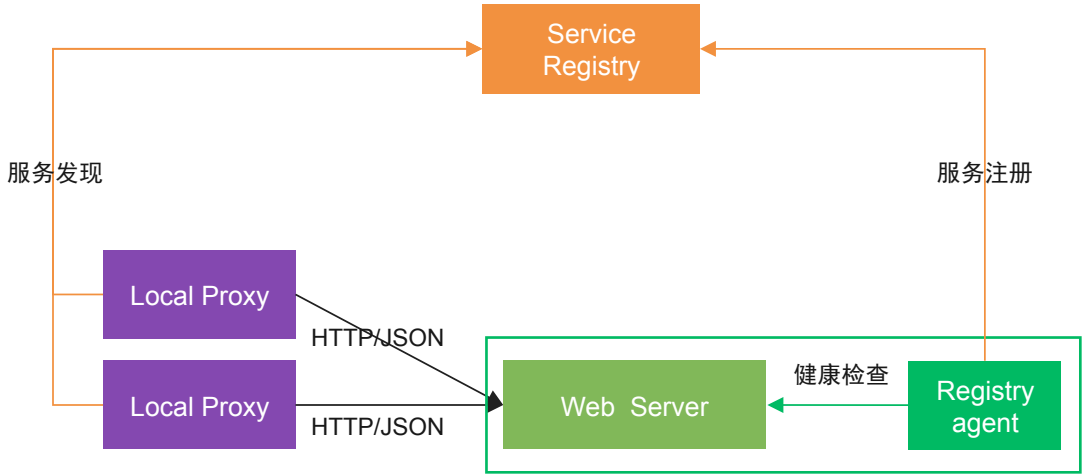
- Java采用Thrift over tcp，PHP/C/C++/Node js采用JSON over HTTP
- Java/PHP，Local Proxy主，Remote Proxy Cluster备
- C/C++/Node js，Remote Proxy Cluster
- 根据不同的接入语言制定不同的接入策略



Service Mesh架构—多语言服务端

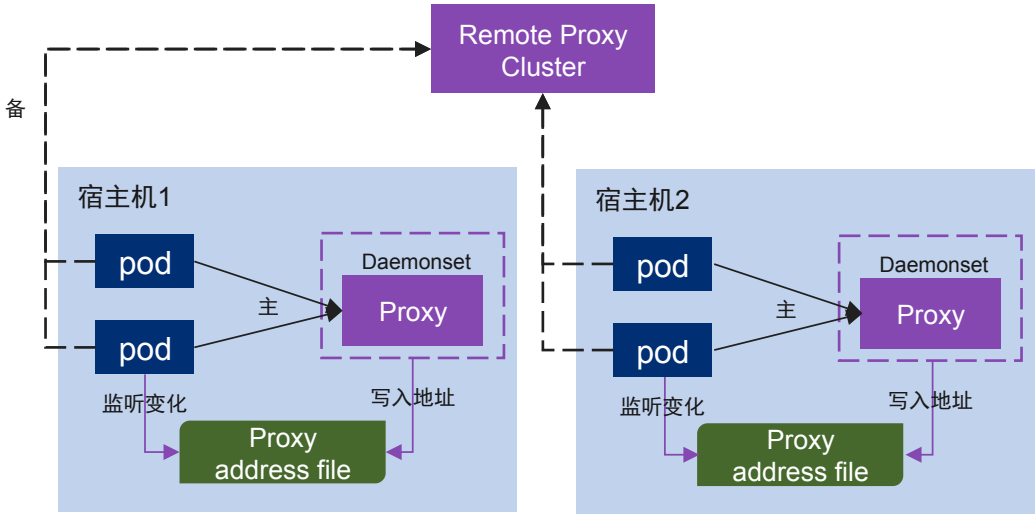
服务端采用Registry agent

- sidecar
- 注册代理
- 健康检查

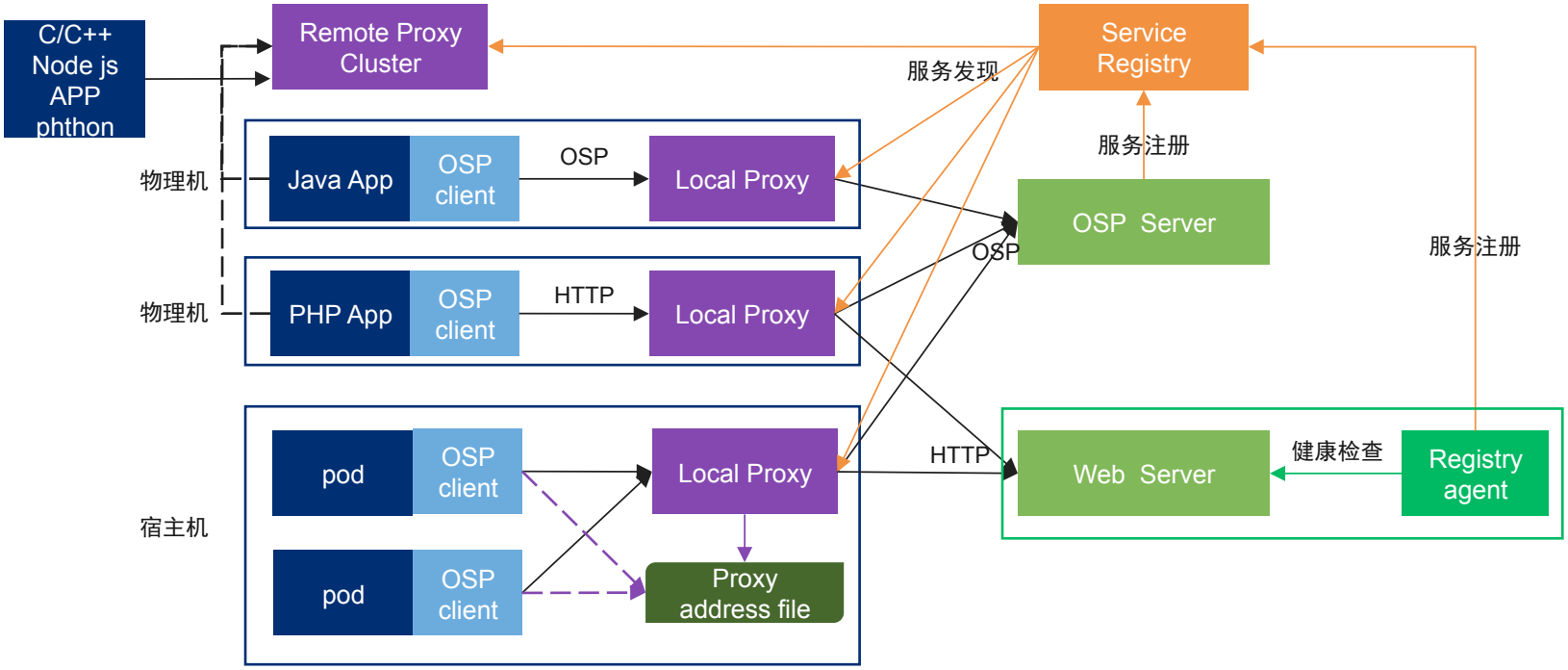


Service Mesh架构—容器化

- 每台宿主机部署多个pod
- 每台宿主机部署1个osp-proxy
- 每个pod部署1个业务Docker容器
- osp-proxy通过daemonset发布到宿主机地址文件，pod根据地址文件找到proxy，并可以实时监听地址文件变化



Service Mesh与Noah容器整合部署架构



- 多语言支持，Java、C/C++、Python、Node js
- Proxy与业务代码解耦，负责服务治理，独立部署，快速升级
- 容器、物理机混合部署

THANKS

Geekbang. 极客邦科技 InfoQ

唯品会
全球精选 正品特卖