QCon[上海站]
全球软件开发大会2016

同程旅游

同程旅游微服务架构设计实践

SPEAKER

王晓波

International Software Development Conference

主办方 Geekbang 极客邦科技 InfoQ

Contents

目　录

同程服务架构的演进

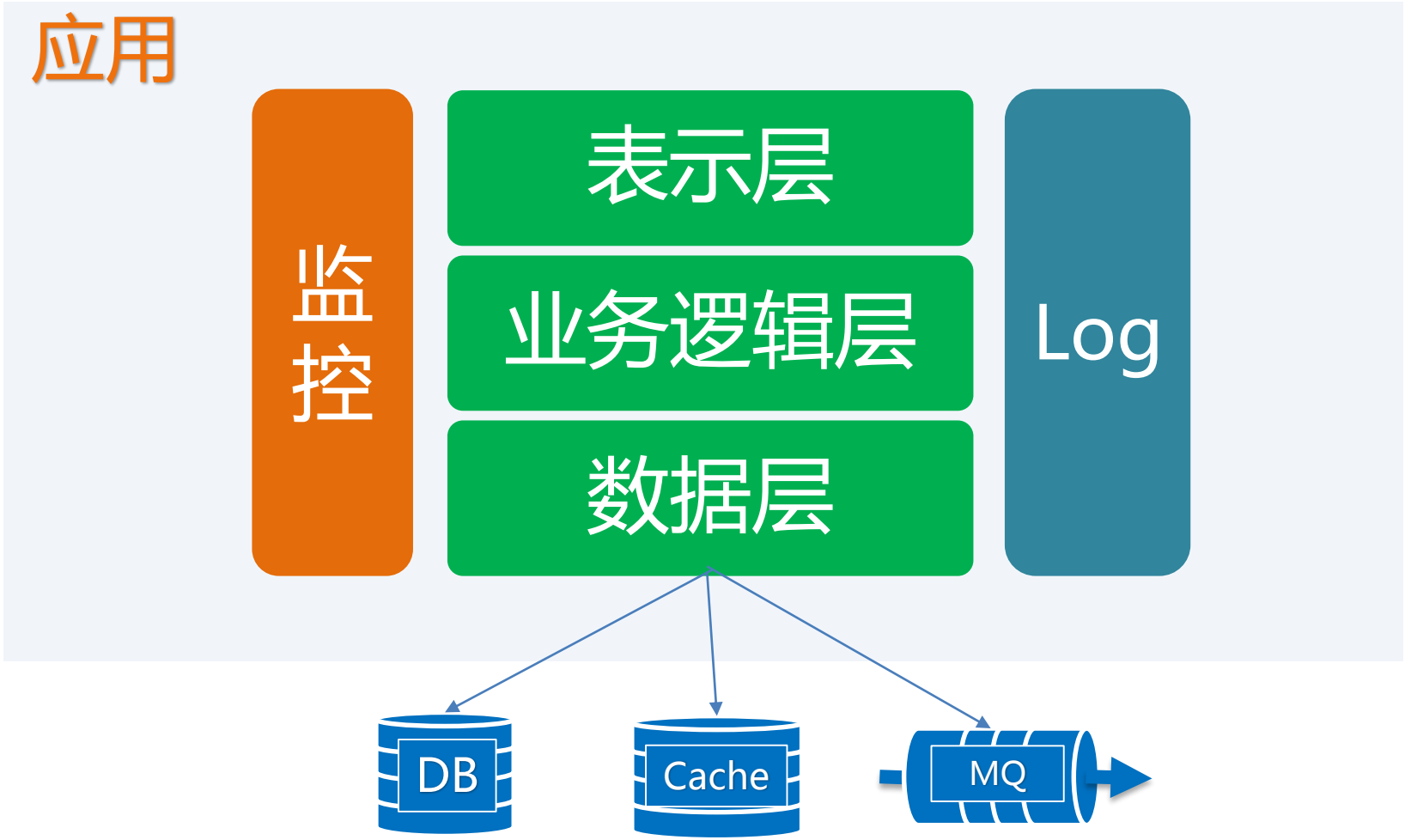
01 2002年 单体架构

服务化架构（SOA）2010年 02

03 2014年 多元化架构（多语言）

微服务架构 2015 04

单体架构



单体架构也有它的味道

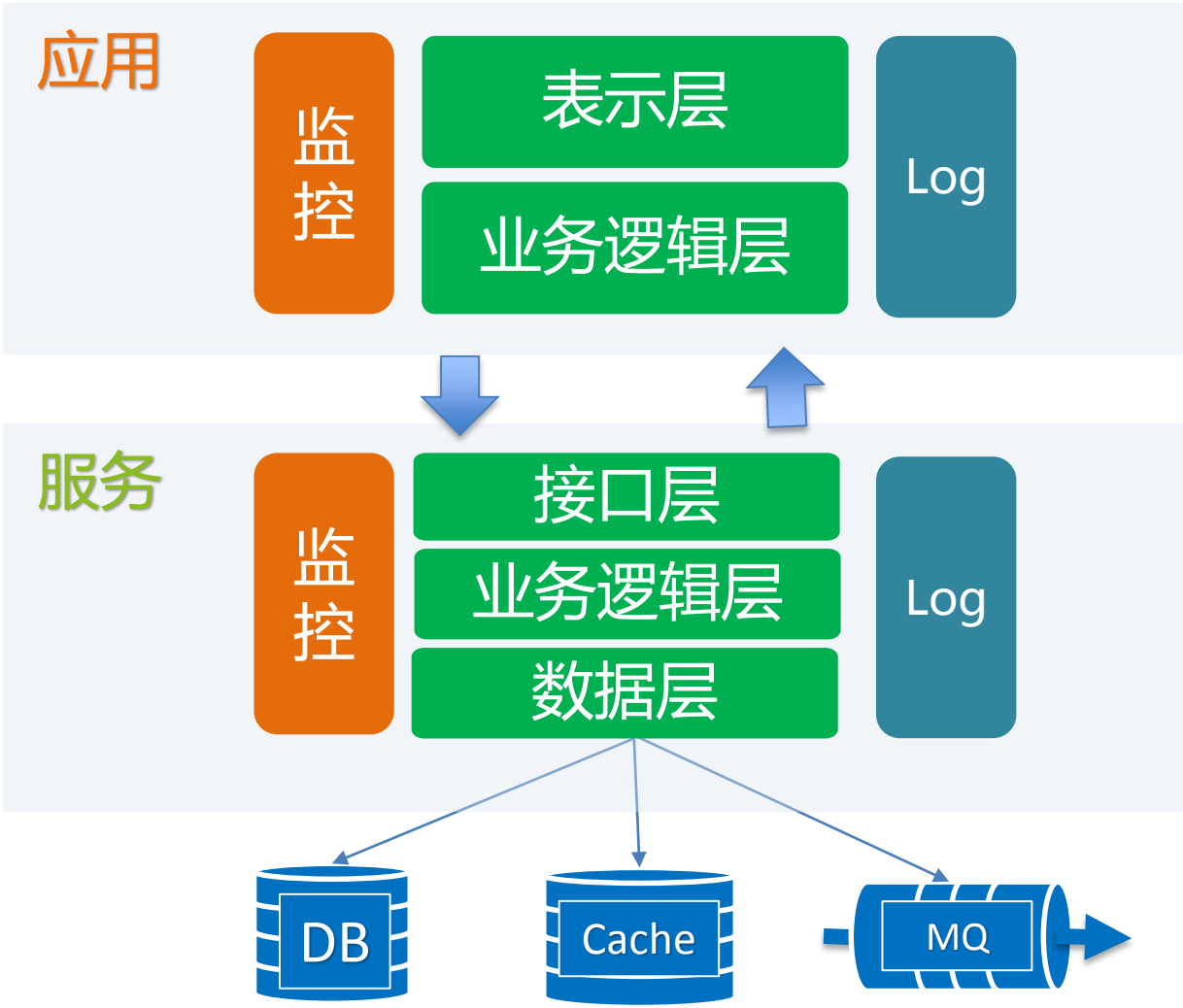
◆优点

- 开发方便（一两个人也能做）
- 部署简单（开发自己运维）
- 开发快（无需要基础也能做）
- 管理简单（一人个系统从头到脚）

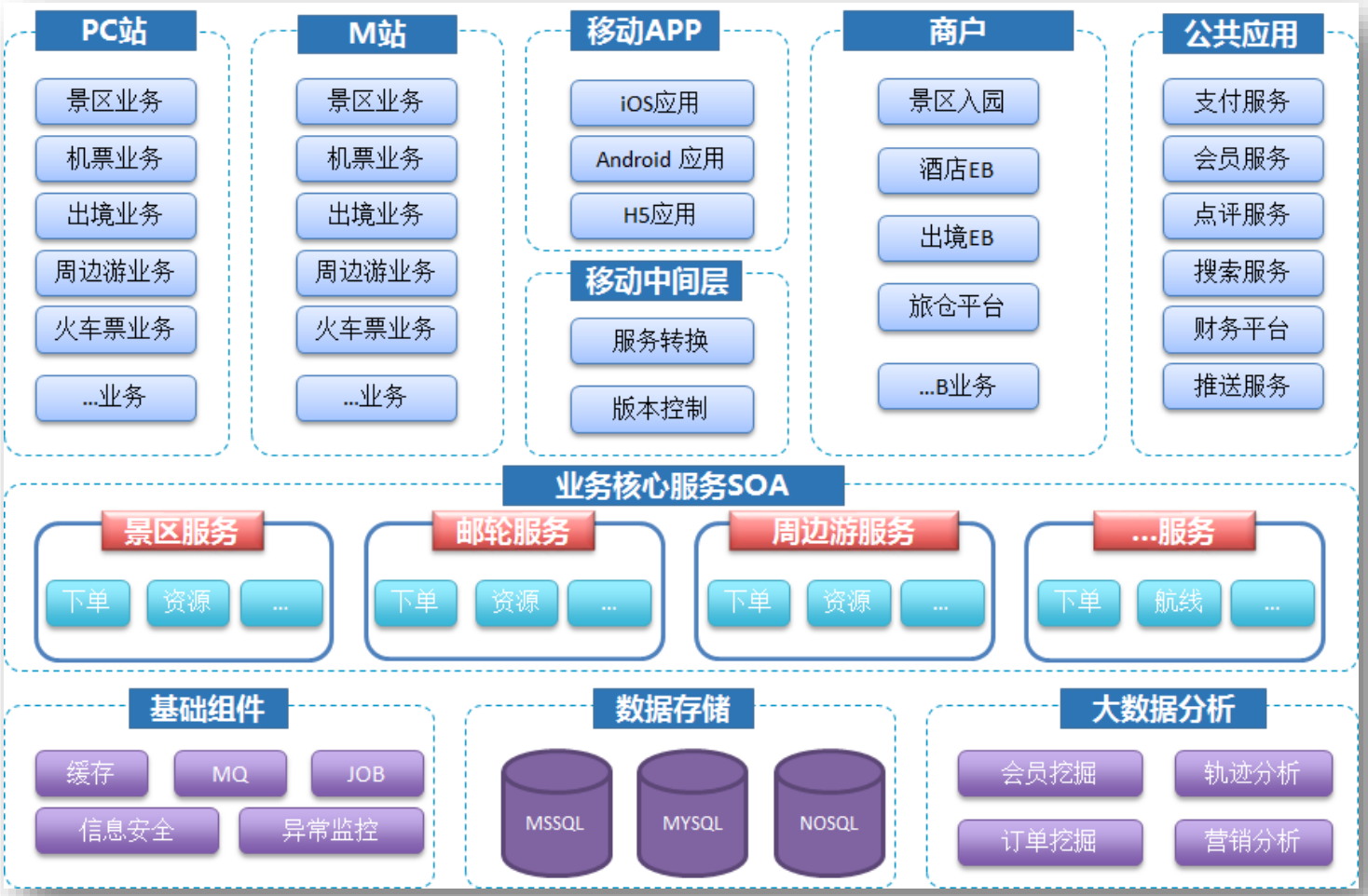
他正在快速的长大的烦恼

- 一个简单的网站变的不简单了
- **问题:**
  - 不只是个网站
  - 各种应用增加3000多个，想一下小型模式玩不动了
  - 流量动不动就是海量（每天数亿级请求量）
  - 原来很简单的一句SQL，现在没法用了
  - 最痛苦的是连缓存也跑不动了
  - 服务器加到不知道放到哪里好
  - 运维最忙的事怎么是各种背不完的锅

初代的服务架构



他已变成这样了


PC站
景区业务
机票业务
出境业务
周边游业务
火车票业务
...业务
M站
景区业务
机票业务
出境业务
周边游业务
火车票业务
...业务
移动APP
iOS应用
Android 应用
H5应用
移动中间层
服务转换
版本控制
商户
景区入园
酒店EB
出境EB
旅仓平台
...B业务
公共应用
支付服务
会员服务
点评服务
搜索服务
财务平台
推送服务
业务核心服务SOA
景区服务
下单
资源
...
邮轮服务
下单
资源
...
周边游服务
下单
资源
...
...服务
下单
航线
...
基础组件
缓存
MQ
JOB
信息安全
异常监控
数据存储
MSSQL
MYSQL
NOSQL
大数据分析
会员挖掘
轨迹分析
订单挖掘
营销分析

问题来了

- 系统越来越复杂，基础设施难度越来越大
  - ✓ 我们来看一个例子，一段对话

要改变这一切

开发:我们的服务为啥不能访问了?

运维:刚刚服务器内存坏了,服务器自动重启了啊

开发:为什么我们的访问延迟这么大?

运维:大哥,不要在redis的Zset里放几万条数据,插入排序会死人啊

开发:我的服务有个应用每秒调用1万多次，我快被他玩了?

运维:你的服务那么多的调用方，我也找不到是谁
开发:刚刚为啥读取全部失败了啊

运维:网络临时中断了一下

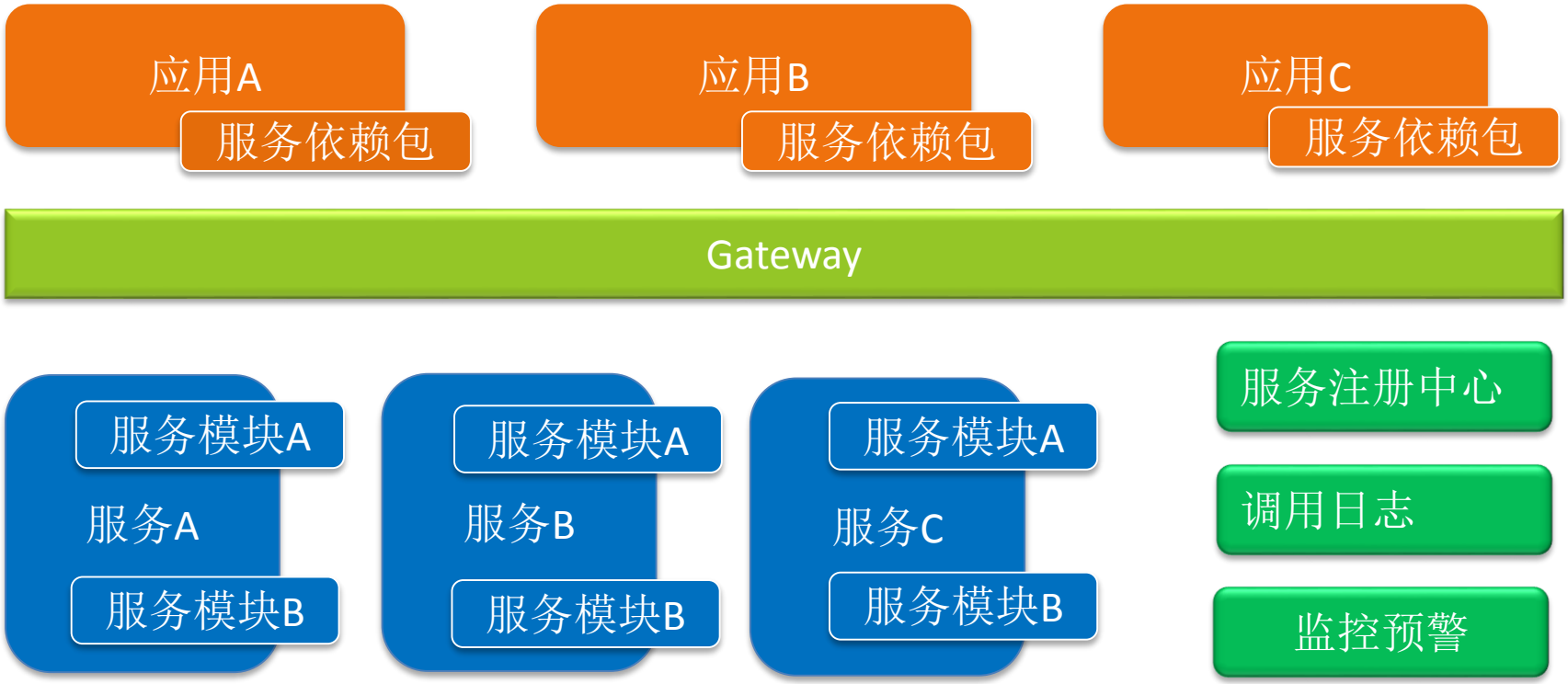
开发:我的系统,我需要800G的redis,什么时候能准备好?

运维:大哥,我们线上的服务器最大就256G,不能玩这么大啊

想想还行，实际一团乱

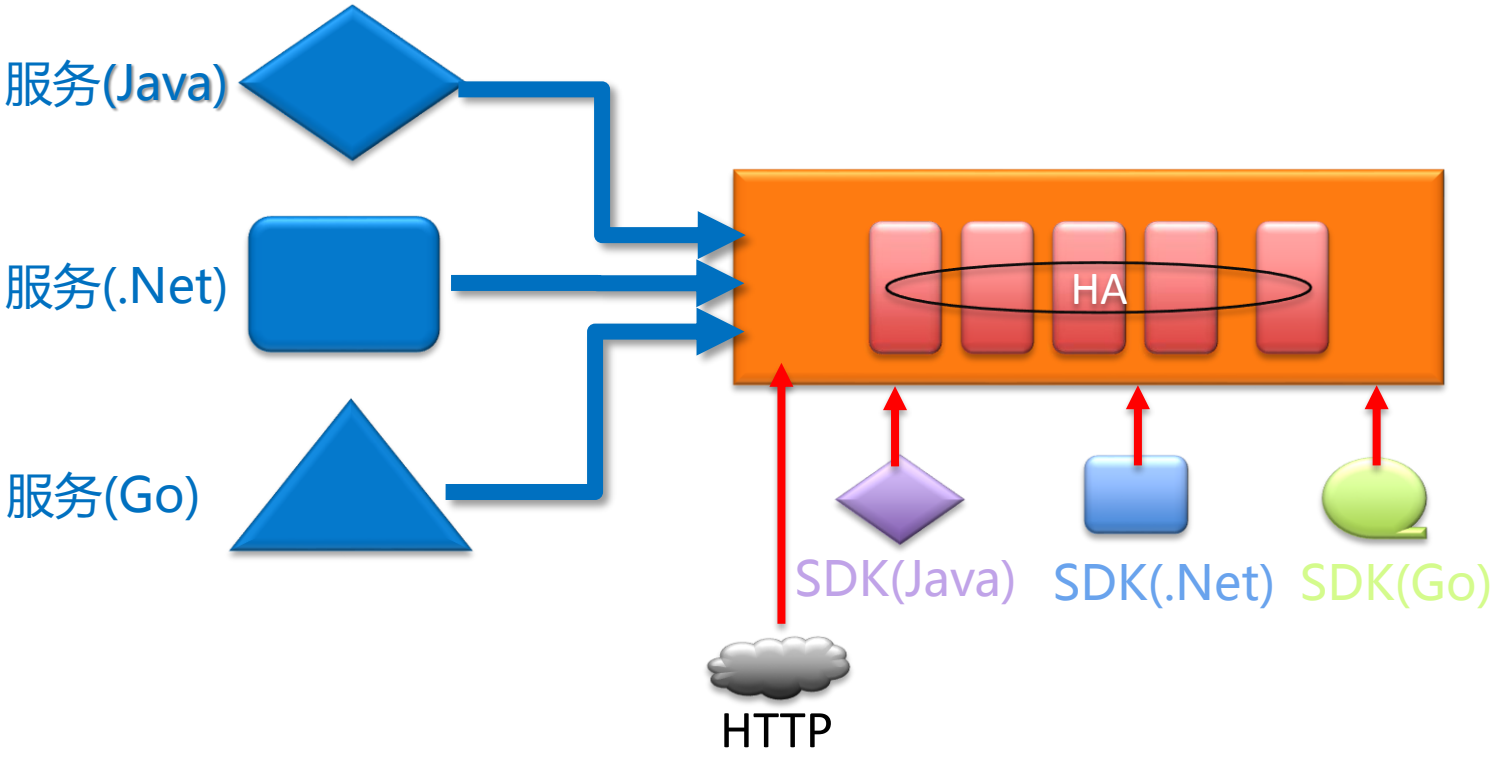
开始服务的治理

- 接口的统一
- 容错处理（限流、回退、隔离、熔断）
- 监控



多元时代的开始

- 从最初的单一.net向java,nodeJS,GO多语言开发转换
- **问题:**
  - 原来的服务有严重的客户端包依赖
  - 为各个语言开发依赖包不太可能
  - 原来语言的一些特性在多语言互联时无法支持



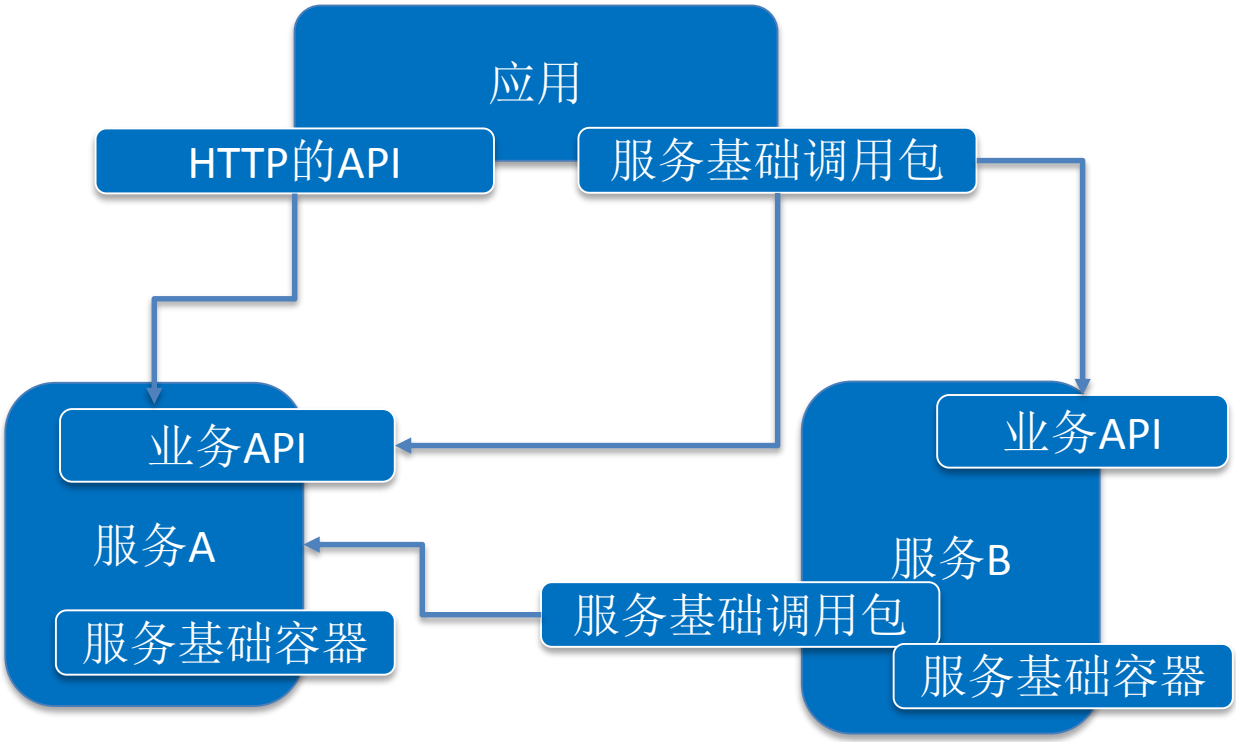
单个服务越来越大

- 服务以大业务为单位划分，系统的变化快，服务变成大胖子
- **问题:**
  - 核心服务与非核心服务在一起，无法高保故
  - 服务的代码不敢改（互相影响）
  - 单个服务的运维变的困难
  - 服务只加不减

他需要减肥了

开始微服务化

- 服务注册发现，控制服务的API数量
- 统一代码框架，支持多种编程语言
- 服务依赖关系管理(服务的分级)
- 服务的CI/CD



注册发现

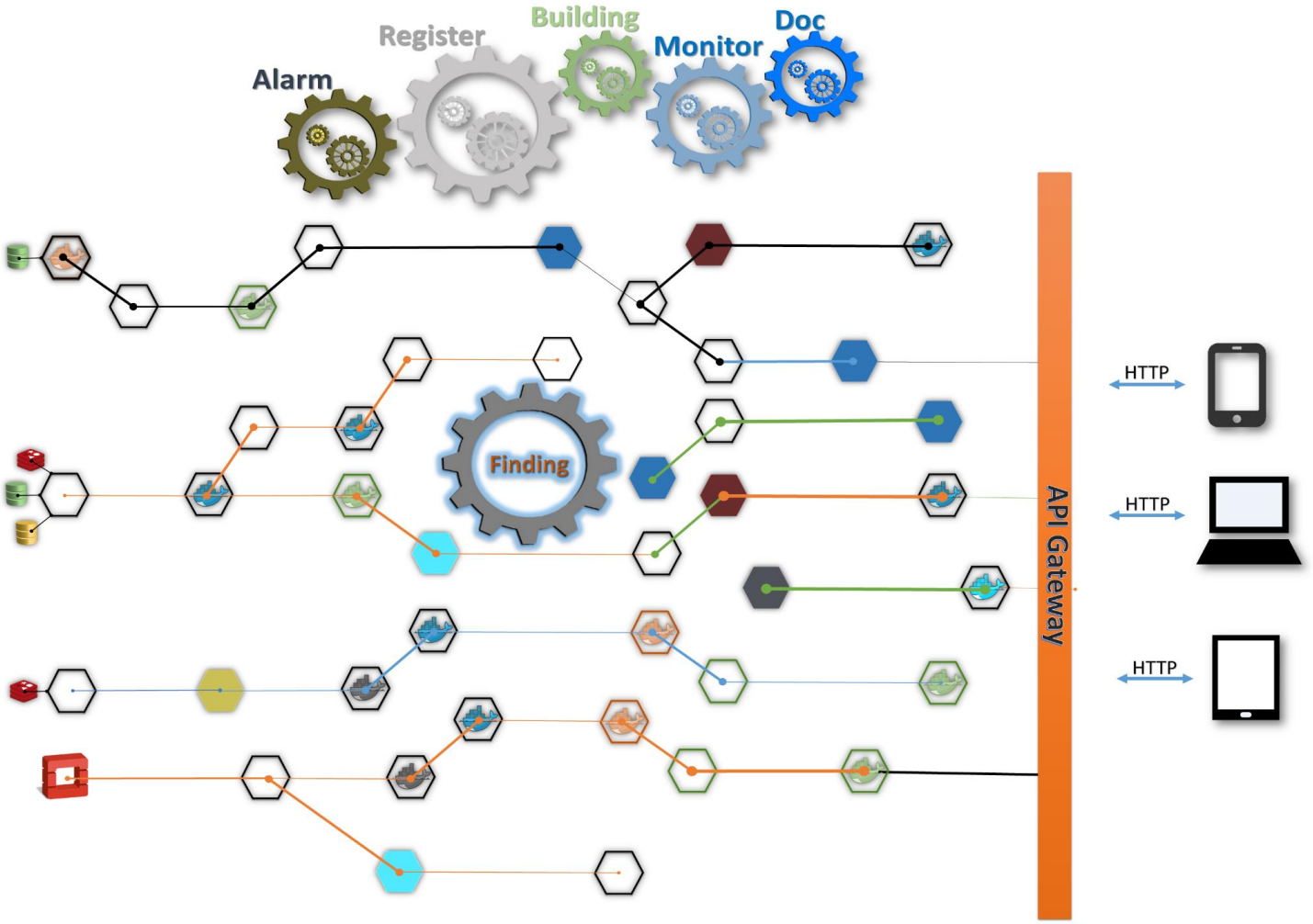
日志和审计

依赖关系管理

监控预警

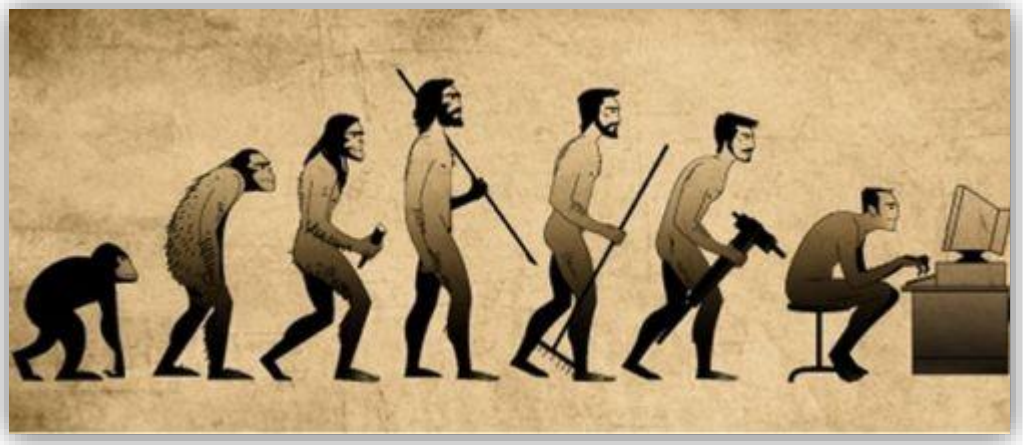
集成发布

一个理想的微服世界


Alarm
Register
Building
Monitor
Doc
Finding
API Gateway
HTTP
HTTP
HTTP

世界总是美好的，现实总是...

- 没有银弹
- **问题:**
  - 服务的数量成几何倍增涨
  - 伪微服务的出现（微服务也变肥了，当你发现总是在它出现后）
  - 旅游的新应用比较多（新的小单体应用，要速度长大）
  - 什么样的服务算微服务

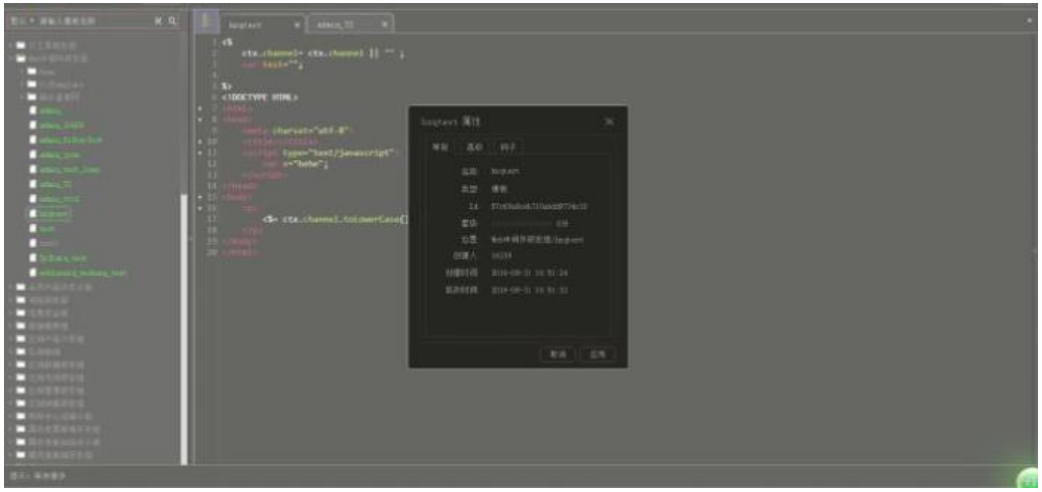


弃用过时的服务

- 微服务越做越多，但出现有增加没有减少的
- **问题:**
  - 因为调用方多等原因使一个服务的下线比较重，所以大部分选择不下线
  - 核心服务会经历多次的更新换代，而老的版本就会被抛弃。(但老版本还有有用)
  - 调用方不愿意更新（你的改变与我无法，别找我）
- 本质
  - 服务设计太过随意
  - 互相这间不知道已有业务存在，重复的轮子
- **解决:**
  - 引回版本管理中心
  - 加入一个应用市场
  - 服务注册监管

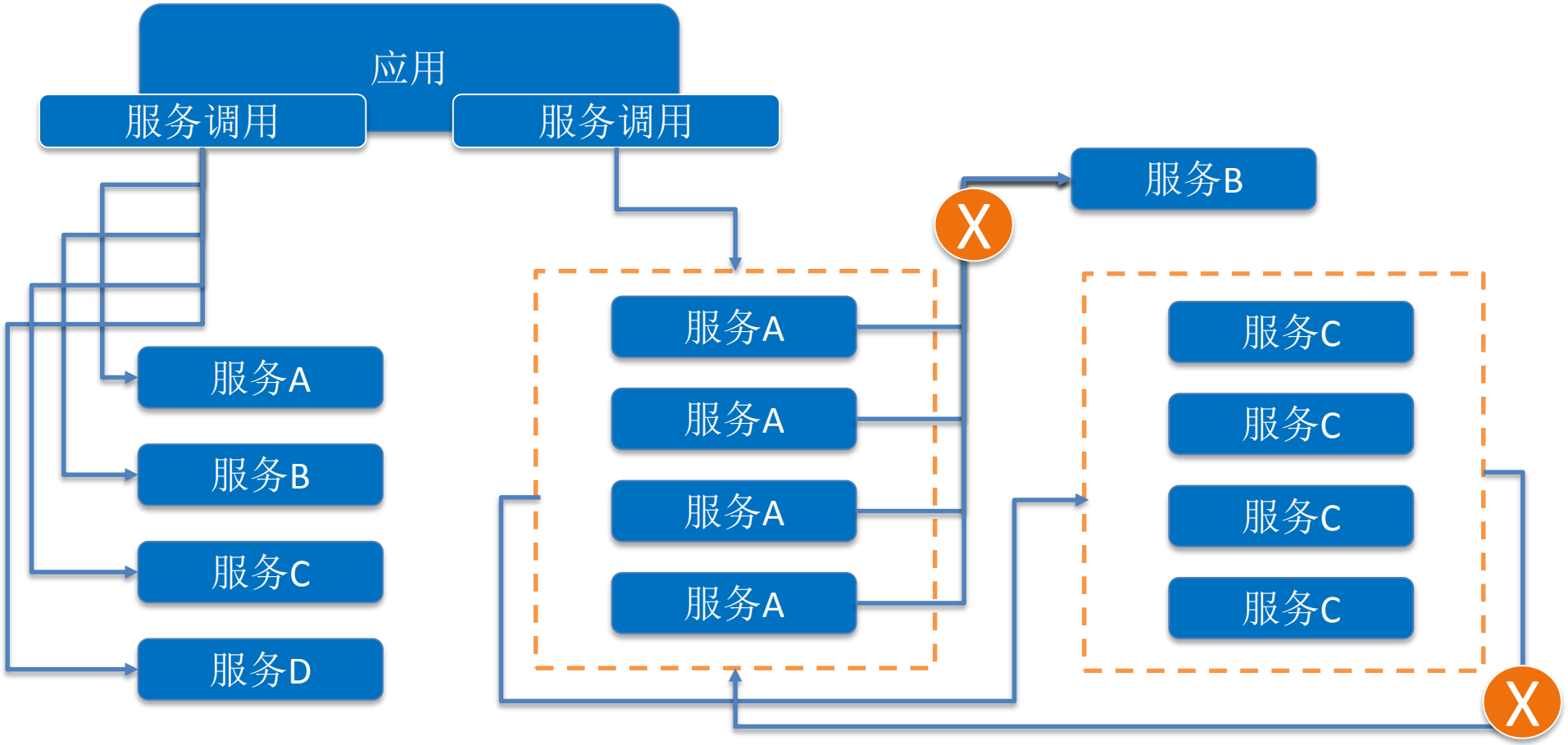
快速构建一个服务

- 微服务因为小，轻。所以出现随意编写。
- **问题:**
  - 如容错处理（限流、回退、隔离、熔断）是要对应异常处理逻辑的，但有忘记写
  - 基础包有很多比较好的编程包，但不是所以多会用
- 本质
  - 编程的人员是五花八门，有高有底的
- **问题:**
  - 服务开发脚手架



服务间的关系与分级

- 可预测的性能是基本要求
- 深度的可靠性



服务的版本问题

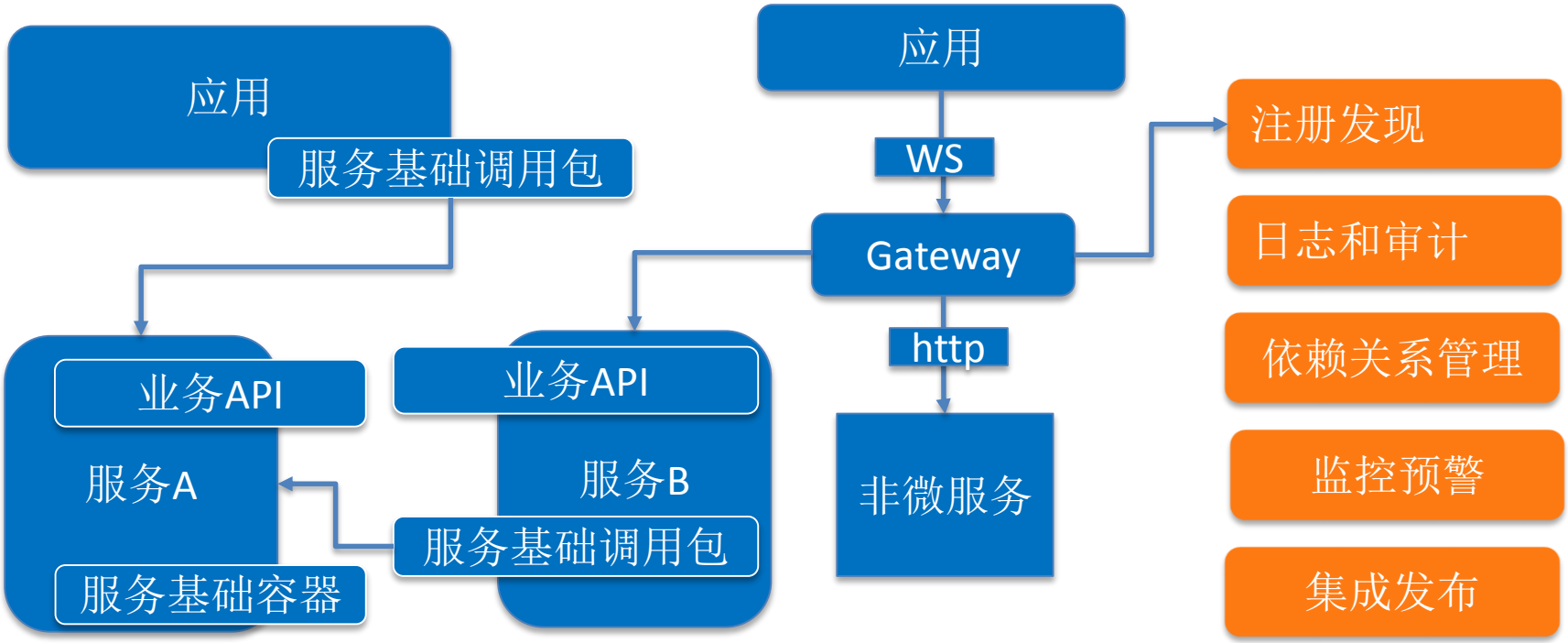
- 微服务要不要有版本，一直是一个要深思的问题
- **不要版本:**
  - 微服务的粒度是已经很小，接近单接口，版本无意义
- **我们选择要版本：**
  - 微服务的粒度其实并没有一个标准，多小是微服务？
  - 望望可独立运行应用块为一个微服务所以接口还是会有多个
  - 多个调用方需求的变化时间不同需要老版本支持
  - 灵活的版本控制，接近于无版本
- **处理:**
  - 不做强版本的依赖:强版本会导致调用方无谓的升级
  - 不做强版本的依赖:线上的版本无穷无尽的增加
  - 不做强版本的依赖：服务部署的资源浪费到哭

微服务转换过程中的新老共存

- 微服务的整体推进是要时间的，不可能一次改完，中间过程需要过渡
- 还有很多为快度打样的做的单体应用，或一些简单应用
- 一个微服务经过长时间迭代更新后也需要新重构
- **问题:**
  - 整个体系中并是只有微服务
  - 管理，调用多会因为这个不得不变复杂起来了
- **处理:**
  - 我们做一个叫“监狱模式”的思想来解决这个问题

监狱模式

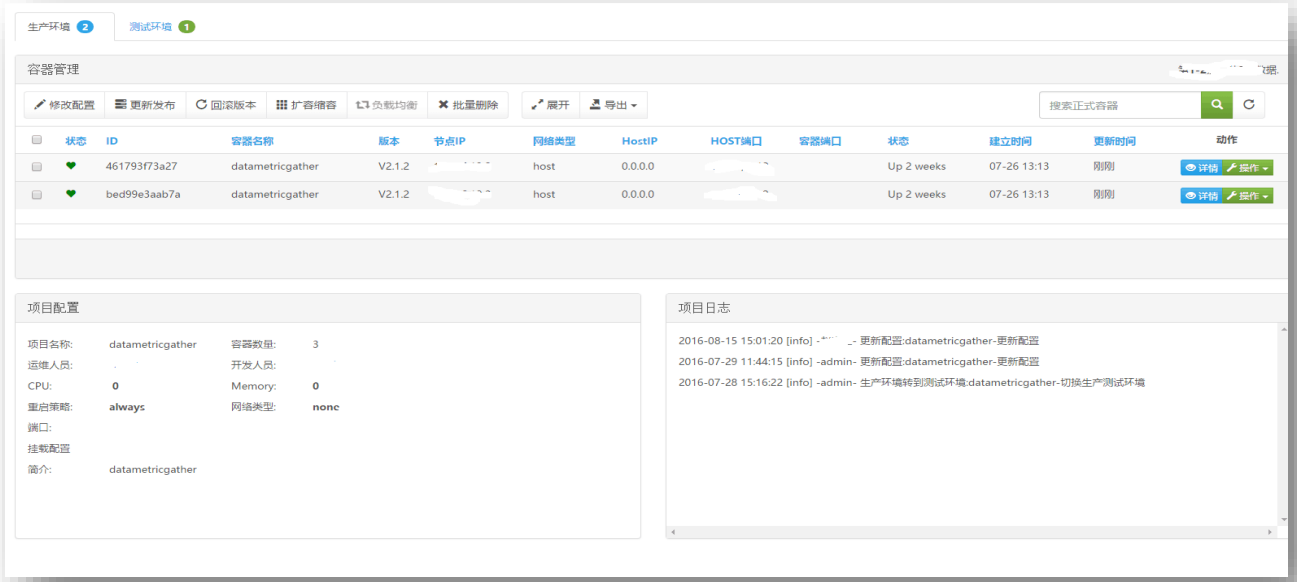
- 原理
  - 使用一个Gateway来屏蔽非微服务的调用
  - 也用这个Gateway来伪装非微服务的应用变为微服务



看起来是美好的

更快的运维和处理问题

- 部署的独立了，代码也更轻了。
- 但部署的单位数量上升了
- 运维也要钉到细节点(最好谁开发谁运维)但开发其实不是运维
- **我们以上问题并没有出现：**
  - 因为我们的Docker应用。
  - 我们几乎与微服务化同时开始推进的Docker
  - Docker的推进比微服务快，目前已经完成



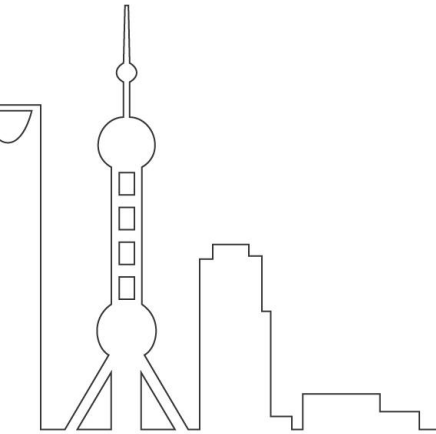
总结起来这一切就是挖坑和填坑

怎么填坑，这是个艺术活

- 直接全部重做吧，这看起是个好办法。
- **问题:**
  - 需要多长时间做完
  - 互联网是每天变的在开等你花半年做新的，业务已经变的不认识了

从小系统到大系统长大本身就是一种美也是一个必须经历的过程

- 小步快跑，找个压力不大的慢慢改
- **问题:**
  - 慢慢改就等于不改
  - 等技术来逼你改的时候，你会发现你已经改不了
  - 新的系统是在不停的增加的，每加个系统就是一个坑
  - 我要的是快速开发，随便写个就是高并发高性能的系统



Thanks!

International Software Development Conference

主办方 Geekbang. 极客邦科技 InfoQ

简介