

The SACC logo is rendered in a bold, white, sans-serif font with a blue glow effect. It is positioned in the upper right quadrant of the image, above the main conference title. The background features a blue wireframe architectural design with a perspective view of a city skyline and a large gear-like structure at the bottom left.

# 2021 中国系统架构师大会

SYSTEM ARCHITECT CONFERENCE CHINA 2021

## 数字转型 架构重塑

IT168.com

ChinaUnix

ITPUB

云上会议 网络直播 | 2021.5.20-2021.5.22

# 携程酒店技术微服务实践

SACC 2021

傅向义

# 个人简介

携程大住宿事业部研发经理，酒店搜筛排前端服务负责人和技术专家

2015年加入携程，曾先后负责酒店前端服务转Java、搜筛排技术架构设计与开发、推荐与广告工程、微服务及中台架构设计与开发等工作。在酒店搜索、筛选、排序、微服务实践等方向有丰富经验。



# 大纲

1. 为什么要做微服务?
2. 怎么做?
3. 微服务为我们带来了什么?
4. 技术实践
5. 经验教训分享
6. 总结和展望

# 1. 为什么要做微服务?

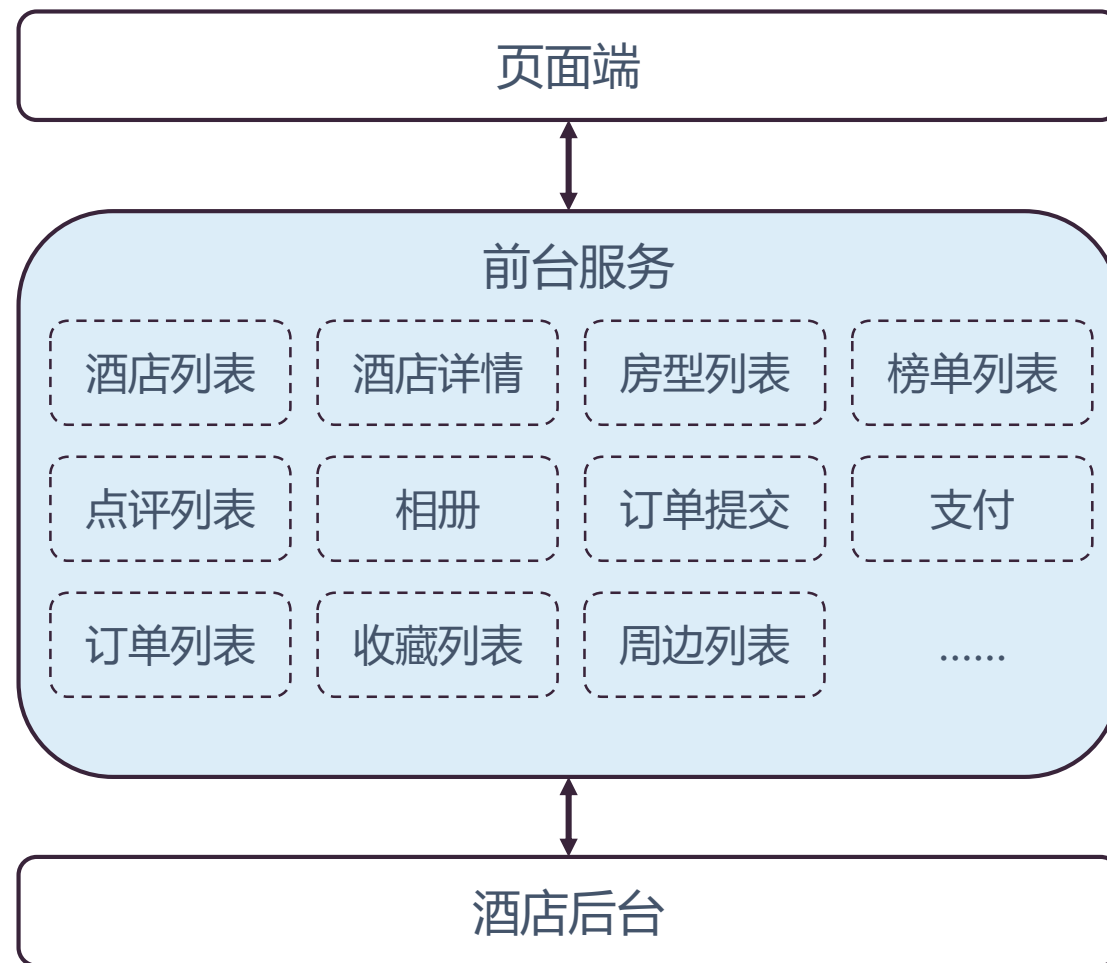
# 原因1：庞大的单体前台应用

## 特点

- 单体应用提供了500+服务
- 五百万行代码仓库

## 问题

- 代码项目越来越大，复杂度指数上升
- 业务之间相互耦合严重
- 无效业务代码清理困难
- 发布部署成本高
- 对新人不友好

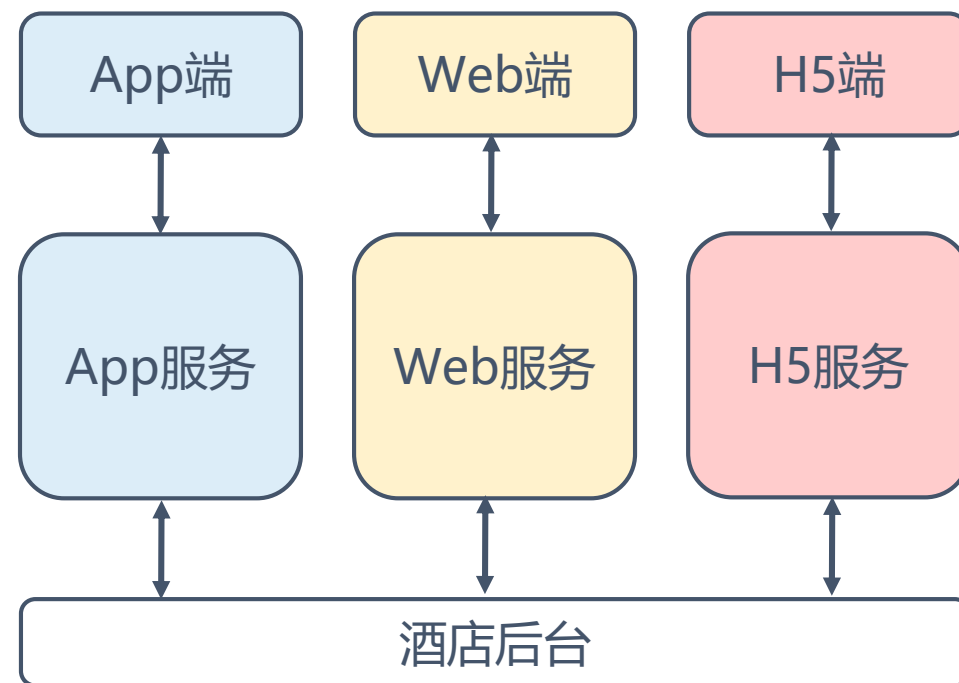


## 原因2：多平台一致性问题

每一个平台都对接各自的单个应用

信息不一致根因

- 开发团队不同
- 架构设计差异
- 数据源和数据模型不一致
- 不同团队代码风格和实现不一致
- 代码冗余严重
- 需求迭代速率不同

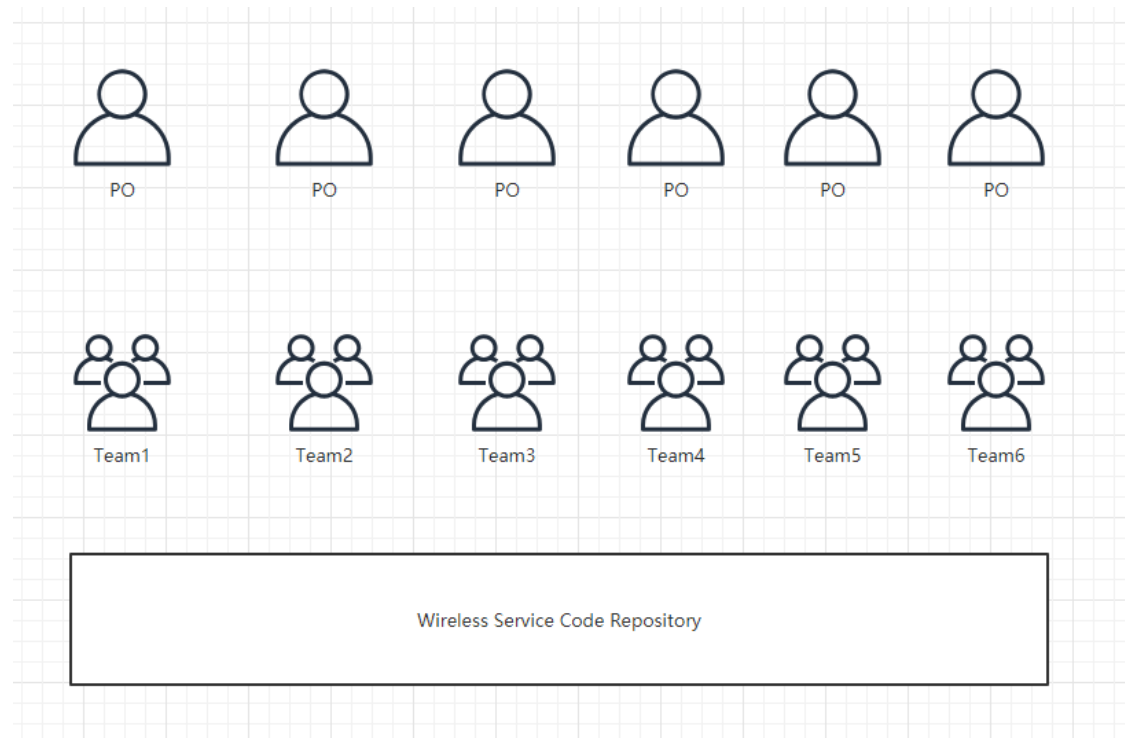


## 原因3：SCRUM模式成本加剧

多个团队同时迭代同一个代码仓库

### 问题

- 代码冲突严重
- 代码碎片多
- 集成测试需求冲突严重
- 发布迭代相互依赖
- 需求管理混乱





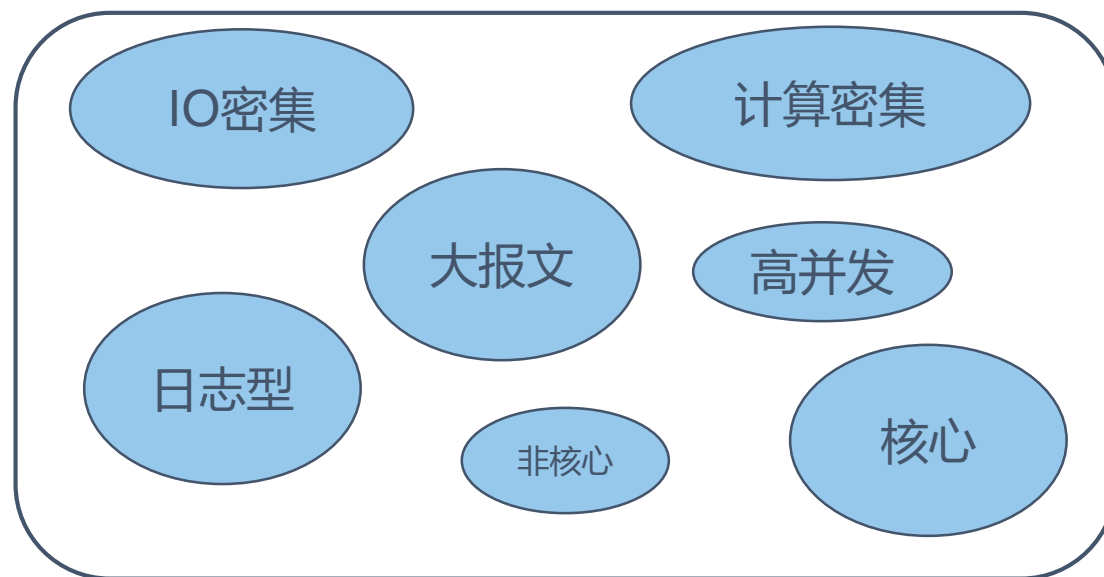
## 原因4：治理困难

不同服务对性能和资源的要求不一样

### 问题

- 非核心服务影响核心服务
- 伸缩性差
- 流量管理困难
- 资源利用率差
- 性能调优困难

单体应用



## 解决方案：微服务化

解耦

可复用

独立部署

弹性伸缩

独立团队

.....

让一个小规模的应用专注做好一件事

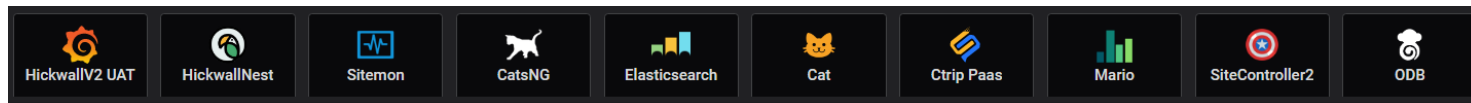
# 可行性分析

服务架构：SOA

技术选型：Java, Springboot, Tomcat, .....

携程云生态：

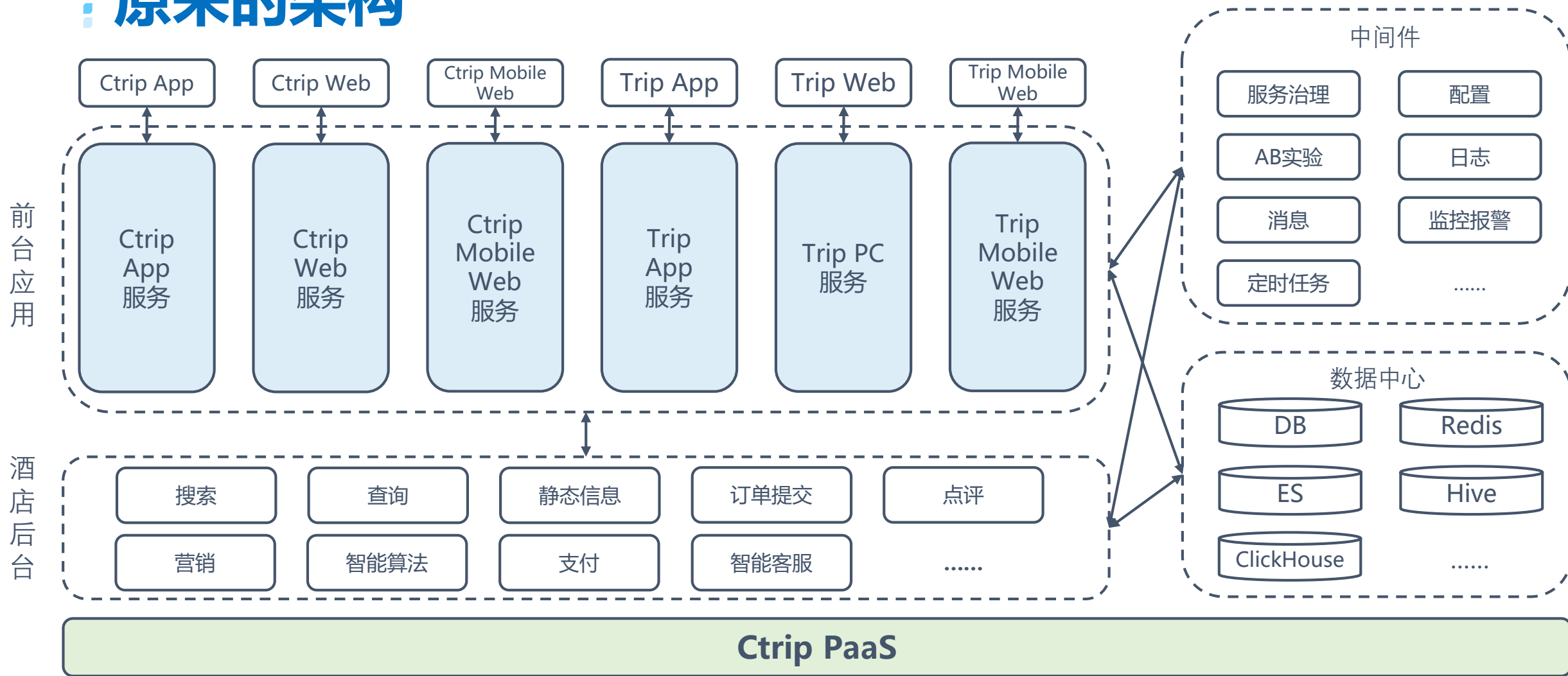
- CtripPaaS（容器化、集成部署、.....）
- SOA Portal（注册发现、熔断限流、负载均衡、配置中心、.....）
- CAT/Sitemon/Hickwall 日志与监控
- ES/Kibana 数据可视化
- .....



## 2. 怎么做?

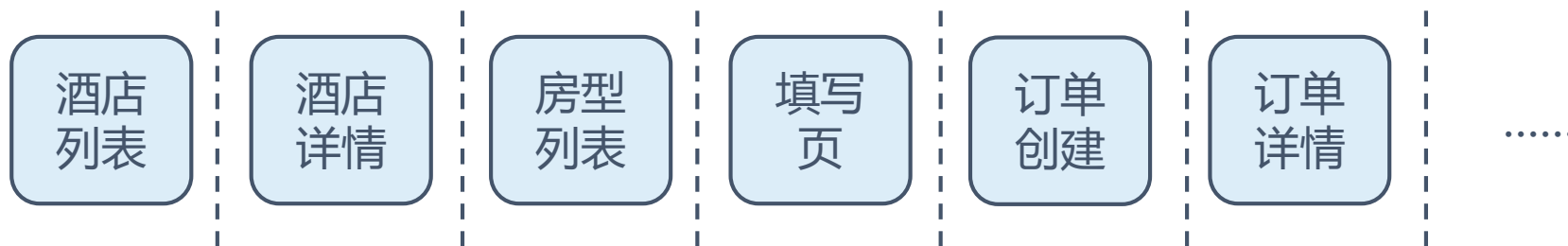


# 原来的架构

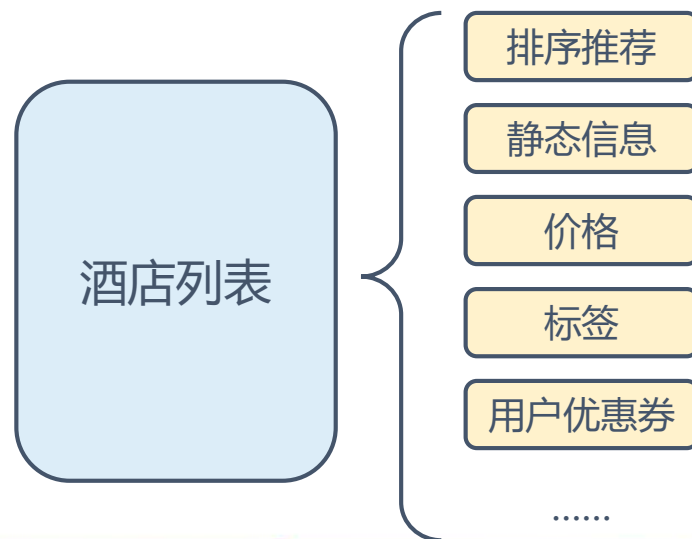


# 领域模型拆分

横向的核心业务拆分

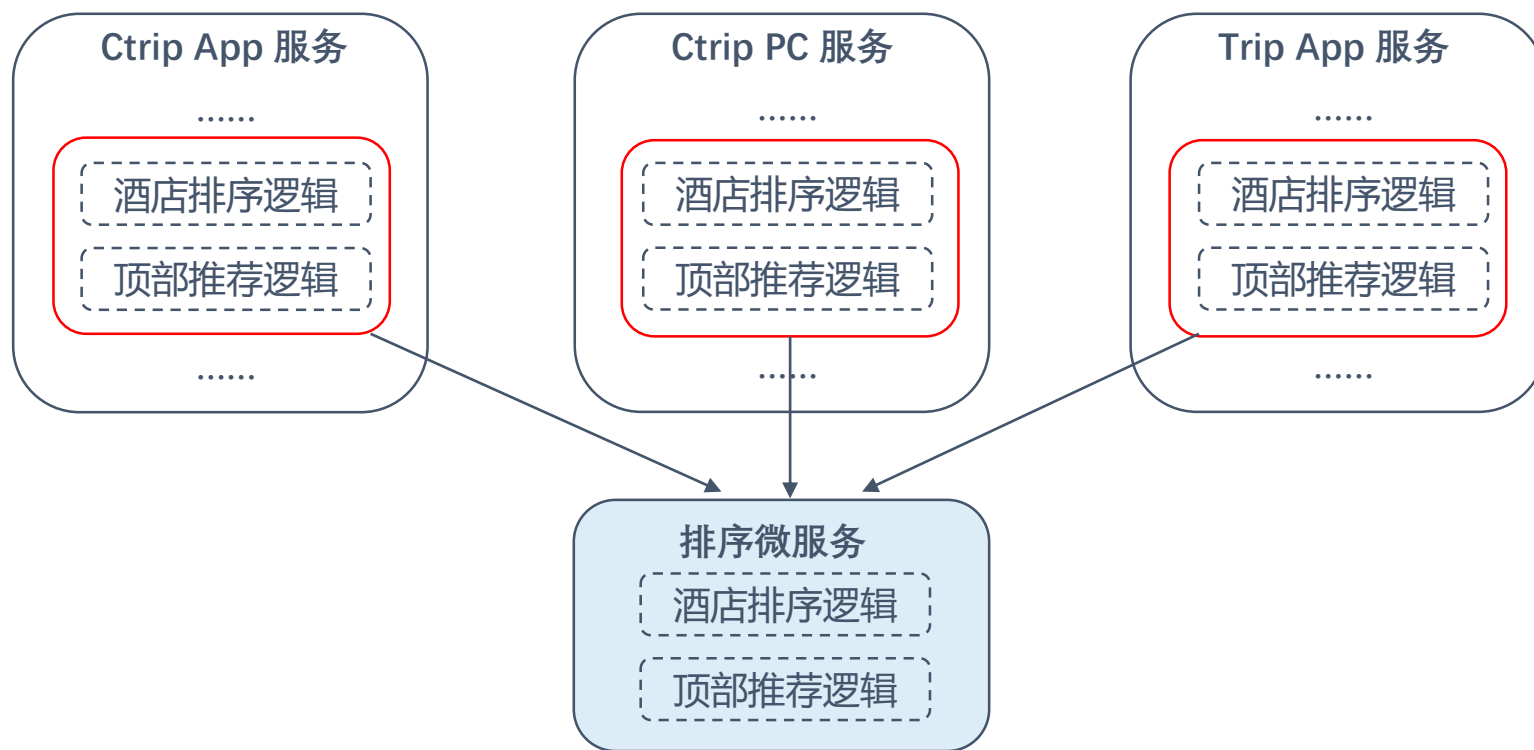


纵向的特有业务拆分



# 逻辑统一

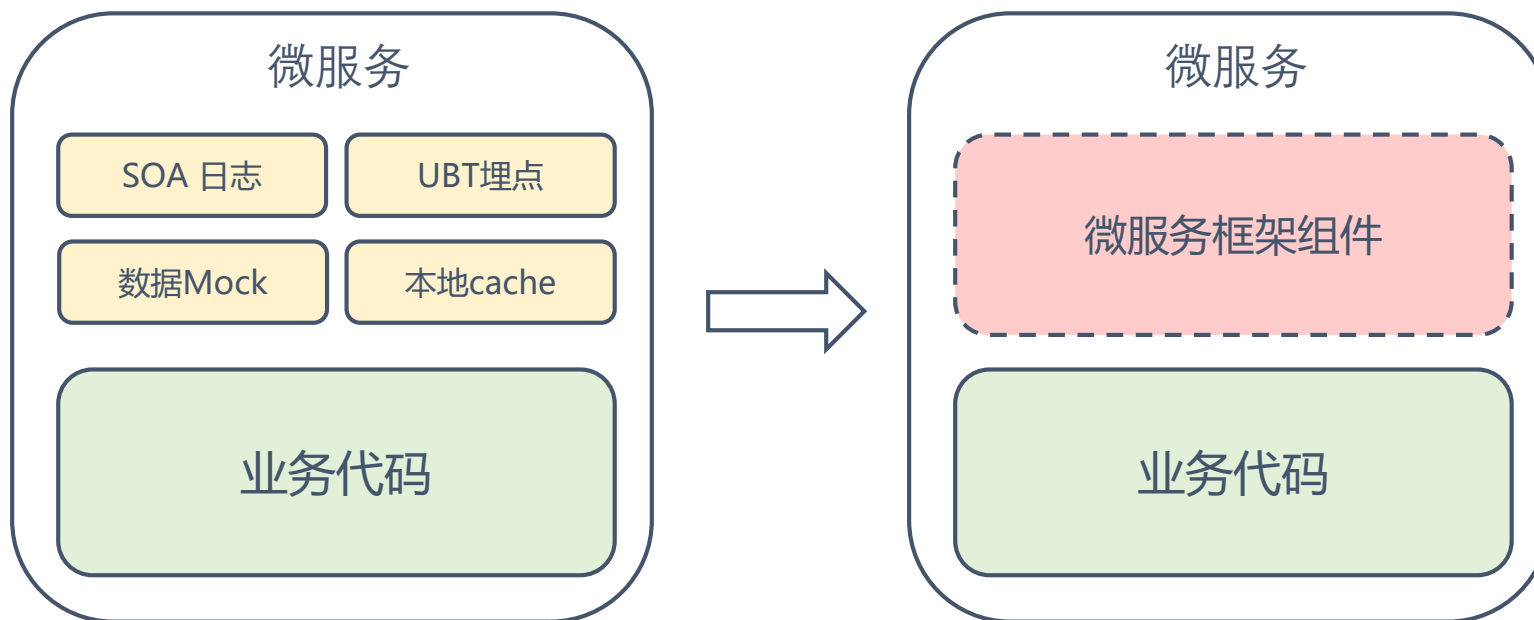
## 多平台逻辑统一收口



# 通用组件抽象

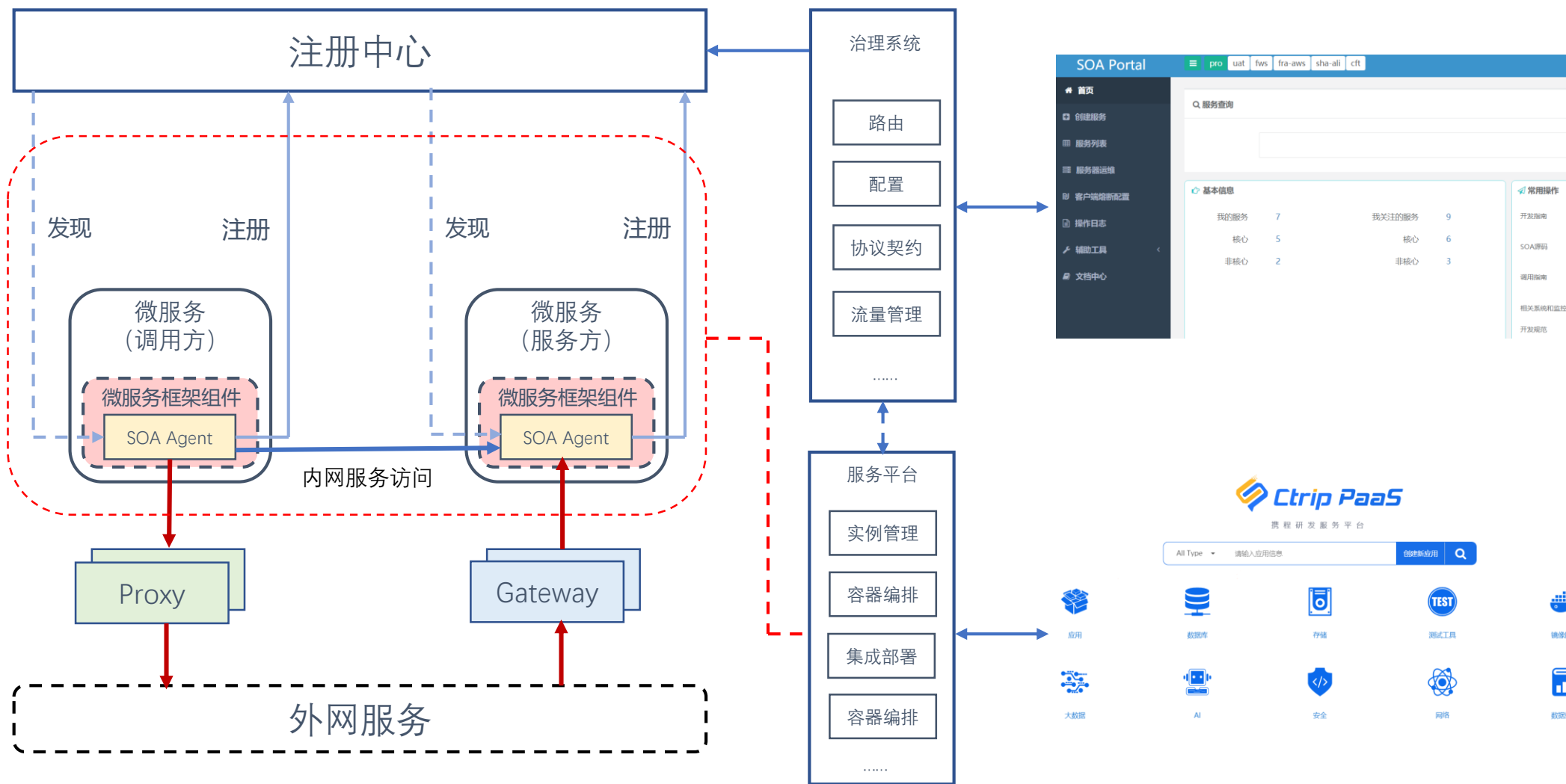
## 微服务框架组件

- 整理统一maven依赖，打包成parent-bom
- 微服务非业务代码解耦
- 功能扩展
- Sidecar (计划中)

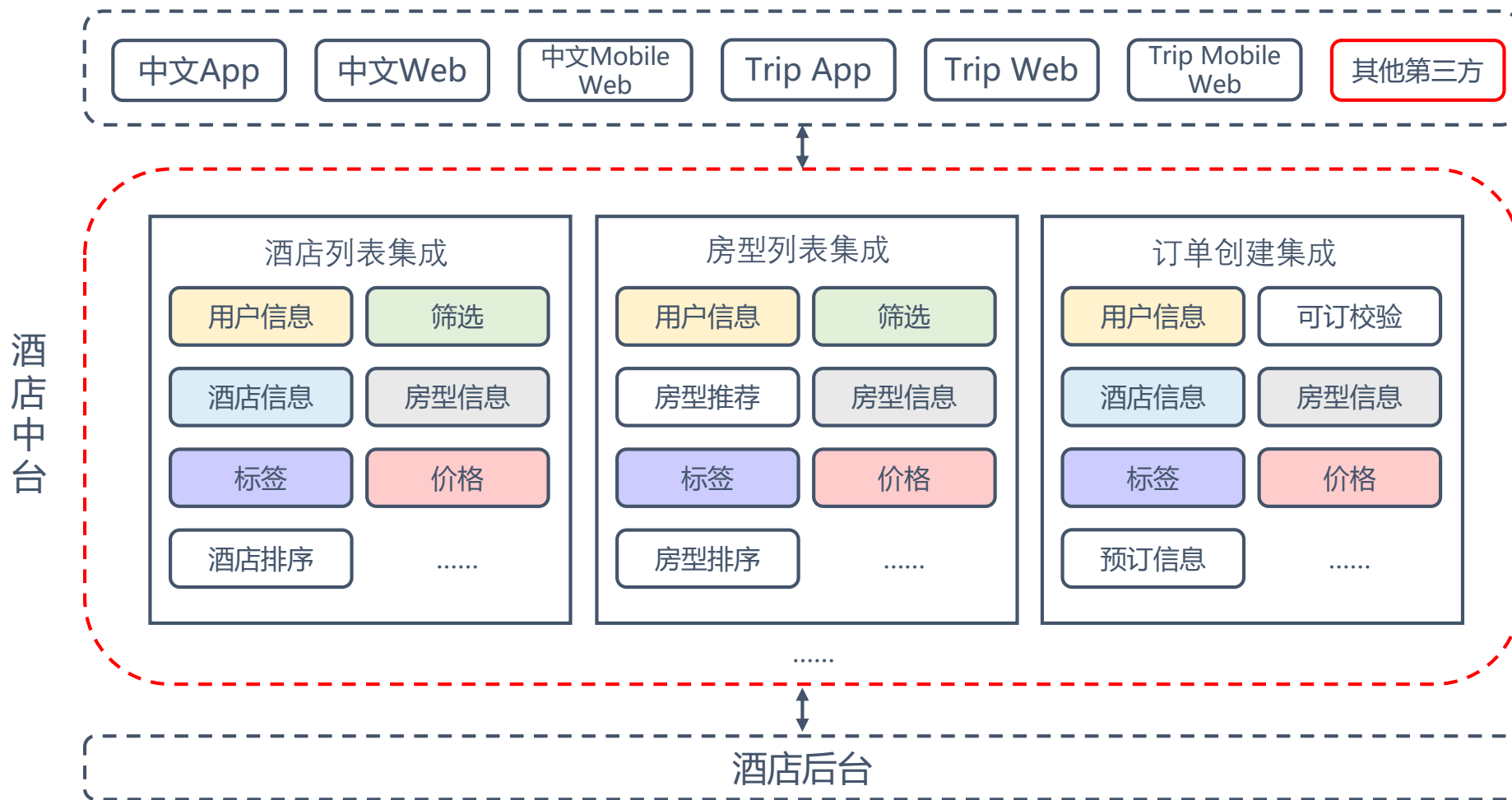




# 微服务平台架构



# 微服务业务架构



# 3. 微服务为我们带来了什么?

# 好处

## 数据一致性大大提升

- 酒店信息、房型信息各端一致率53% -> 100%

## 系统容错力提升

- 独立部署
- 弹性伸缩 (SmartHPA)

开启 SmartHPA 后, SmartHPA 将自动管理实例数量, 根据用户配置的目标 Metric, 系统将在低负载时缩容、高负载时扩容。动态的实例数量管理将有助于提高资源利用率、节省计算资源、更高效地应对突发的流量。

开启 SmartHPA

## 研发效率提升

- 同一份业务代码
- 微服务发布灵活成本降低
- 消化更多需求

## 研发人员的成长 (码农 -> 专家)

- 成为微服务owner
- 微服务持续性能优化



# 带来的新问题

## 问题排查环节变多

- 单应用内的处理 → 微服务间的通信
- 分布式缓存节点分散在
- 解决方案：分布式日志追踪（后面介绍）

## 发布管理复杂

- 一个团队管理的微服务数量变多
- 总发布频次增多
- 解决方案：发布计划管理



| ID: 231 8.35版本发布 |        |    |   |       |       |
|------------------|--------|----|---|-------|-------|
| 计划 发布            |        |    |   |       |       |
| 序号               | 计划发布时间 | 类型 | 发布项   | 开发负责人 | 测试负责人 |
| 1                | 未设定    | 应用 | APPID: 100022244<br>htl-basicproductinfo-service  | 教     |       |
| 2                | 未设定    | 应用 | APPID: 100015205<br>htl-user-profile-service      | 宇     |       |
| 3                | 未设定    | 应用 | APPID: 100025614<br>htl-hotelfrontpayment-service | 宇     | 清     |
| 4                | 未设定    | 应用 | APPID: 100017590<br>ibu-hotel-assist-service      |       |       |
| 5                | 未设定    | 应用 | APPID: 100018705<br>ibu-hotel-meta-job            |       |       |
| 6                | 未设定    | 应用 | APPID: 100005603<br>htl-wireless-order-service    | 良     |       |

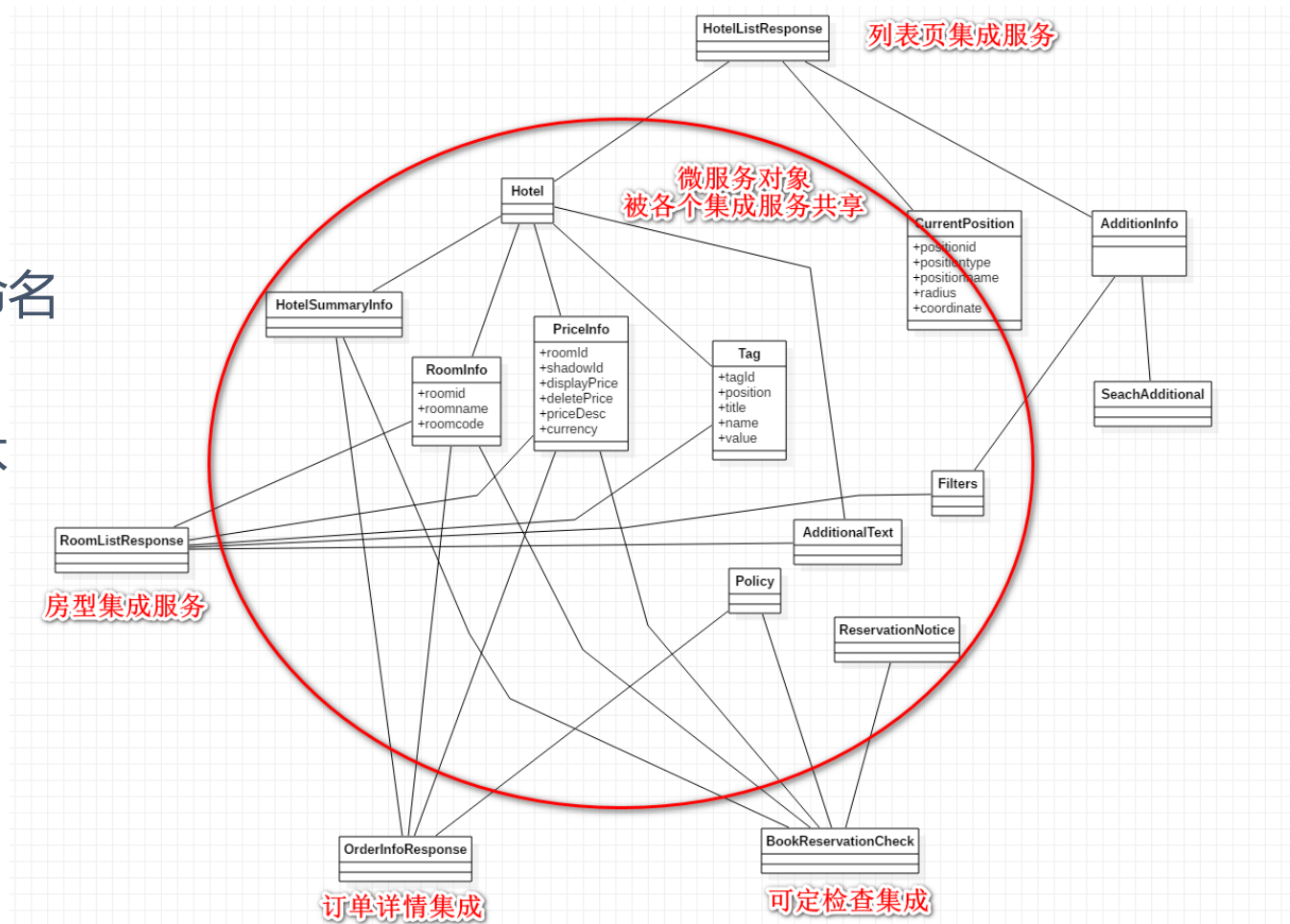
## 4. 技术实践

# 通用数据模型建设

减少数据模型差异

统一字段名定义，减少五花八门的命名

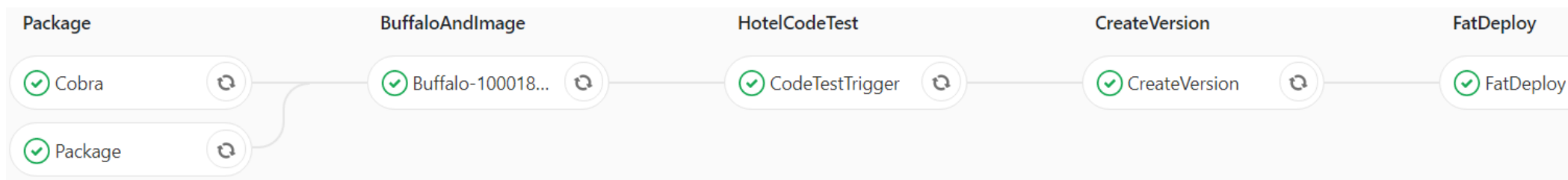
降低微服务之间协议契约的定制成本



# CI/CD实践

## CI&CD接入

- 由git merge触发，经过一系列代码检测，打包发布部署
- 支持发布生产环境





# CI/CD实践效果

## 微服务规范和要求:

- CodeReview -> 100%
- Sonar问题 -> 0
- UT新增覆盖率-> 80%

## 接入应用

所有微服务均接入  
CI&CD流程

## 覆盖率

UT平均覆盖率: 86%

## 发布效率

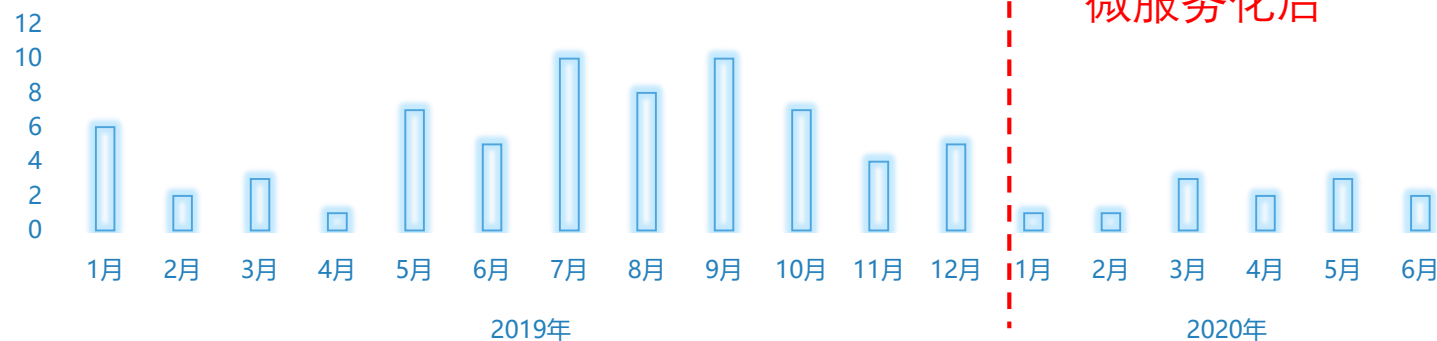
每次发布节省约1h  
静态扫描: 节省约30m  
自动化运行和分析: 节省30m

## 线上质量

质量数据:

- 1) 无影响订单的RCA
- 2) 生产事件: **10个**, 对比同期减少60%

## 生产事件数量



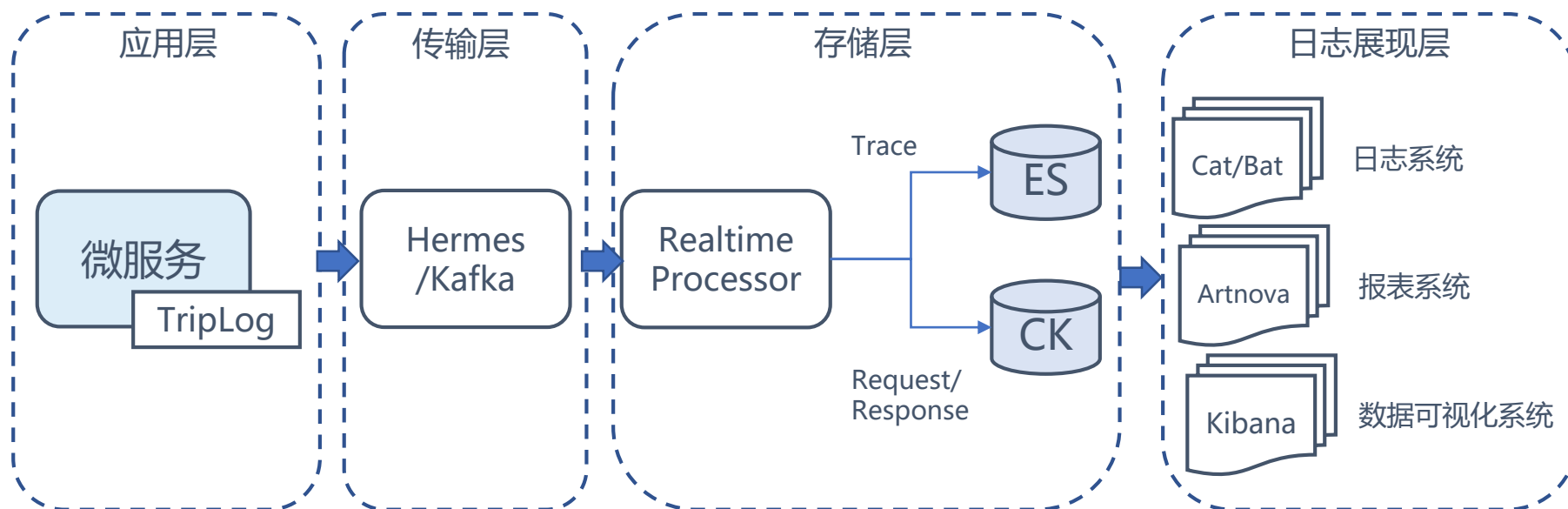
生产事件数量减少60%

# 日志架构

日志框架

日志标准化

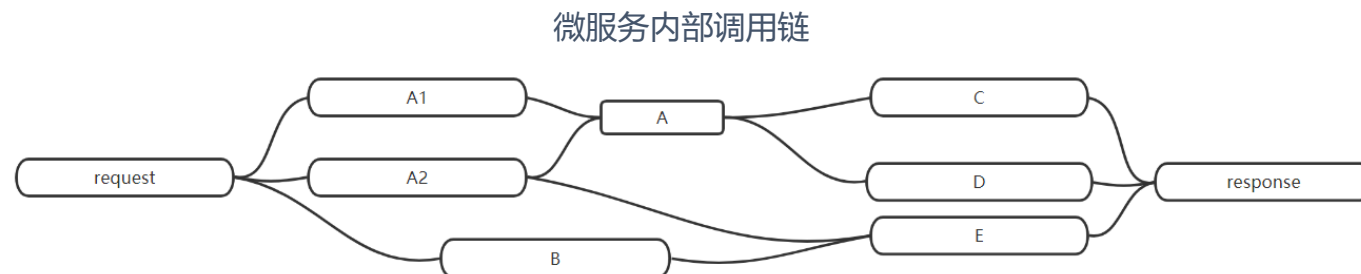
- 日志/埋点分布式框架
- 微服务日志实现



每个微服务内部可能依赖多个第三方或者后台服务

## 唯一标识前后串联单次请求中的完整链路

## 提升问题的排查效率



getHotelRoomList

getUserProperty

unpack

getUserInfo

getAccountInfoB...

getAvailablePoi...

hasUserReward

queryUserMember...

userCouponGet

hotelRatePlan

getRoomInfo

getRoomRecommen...

recommendRoom

roomIncentiveSe...

getRoomSort

roomPack

getIpInfo

GetThirdPartMem...

checkBlackList

getUserPrefer

getScene

hotelMemberInfo

getTimeZone

Show Whole

Json Mode

Scenarios:

hotel-middle-list-svc-ap

hotel-roomlistinfo

hotel-wireless-trace

> 100023790 - hotel-middle-list-svc-ap [CK]

> 100023790 - hotel-roomlistinfo [CK]

timestamp

cat\_client\_appidSubServiceAlias

Request

Response

2021-05-10 08:20:27

100023790

getRedirectRoomPriceList

{ "POS": { "ClientFromDetail": { "DeviceID": "...", "ClientID": "...", "RatePlans": [{ "HotelRef": { "Address": { "Text": ...

2021-05-10 08:20:27

100023790

{ "head": { "platform": "IPhone", "clientVersion": "835.002", "clientId": "...", "ResponseStatus": { "Timestamp": "/Date(162063482748...oom ...

2021-05-10 08:20:27

100023790

getIpInfo

{ "ip": "...", "result": { "status": 1, "countryCode": "CN", "countryName": "eCH": " ...

2021-05-10 08:20:27

100023790

checkBlackList

{ "head": { "platform": "IPhone", "clientVersion": "835.002", "clientId": "...", "ResponseStatus": { "Timestamp": "/Date(162063482733...n": ...

2021-05-10 08:20:27

100023790

getThirdPartMember

{ "uid": "...", "thirdPartType": "...channel": "...", "returnCode": 302, "message": { "...634827343+0800) ...

2021-05-10 08:20:27

100023790

getUserRewardInfo

{ "requestHead": { "requestType": "Hotel.Order.MiddleGroundService.HotelMemberInfo", "clien...tAppId": "...", "ResponseStatus": { "Timestamp": "/Date(162063482734...n": ...

2021-05-10 08:20:27

100023790

getUserPrefer

{ "head": { "platform": "IPhone", "clientVersion": "835.002", "clientId": "...", "ResponseStatus": { "Timestamp": "/Date(162063482735...n": ...

2021-05-10 08:20:27

100023790

getScene

{ "head": { "platform": "IPhone", "clientVersion": "835.002", "clientId": "...", "ResponseStatus": { "Timestamp": "/Date(162063482735...n": ...

2021-05-10 08:20:27

100023790

getTimeZone

{ "lat": "...ng": "...latAndLonType": "GaoDe", "ResponseStatus": { "Timestamp": "/Date(162063482734...n": v ...

2021-05-10 08:20:27

100023790

getUserProperty

{ "head": { "platform": "IPhone", "clientVersion": "835.002", "clientId": "...", "ResponseStatus": { "Timestamp": "/Date(162063482735...n": ...

## 5. 经验教训分享

# 微服务化=重构

微服务化的过程，其实就是对原有代码的重构

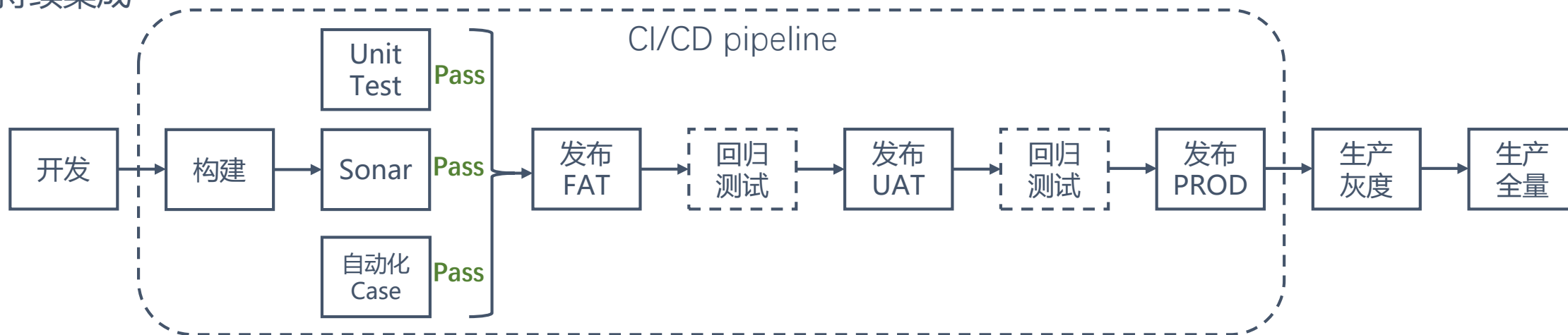
我们到底都有哪些业务逻辑？

- 文档维护缺失+人员流动，唯有代码，而且注释几乎没有，导致前期整理困难
- 大量无用代码、开关、ABTest实验等影响业务分析

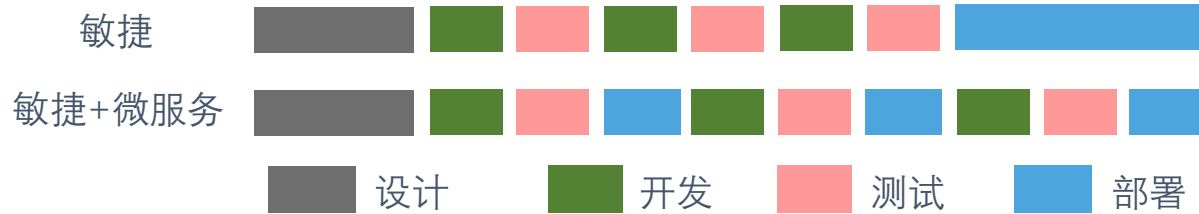
因此，微服务化也是一次文档的再整理，以及代码的清理过程。

# 生产流程标准化

持续集成



迭代开发流程变化





# 技术升级与业务需求的矛盾

微服务化本身会消耗研发资源

- 与产品沟通达成妥协，在一定周期内减少业务需求量
- 抽调人员单独组建微服务研发团队

不能 “Stop The World”

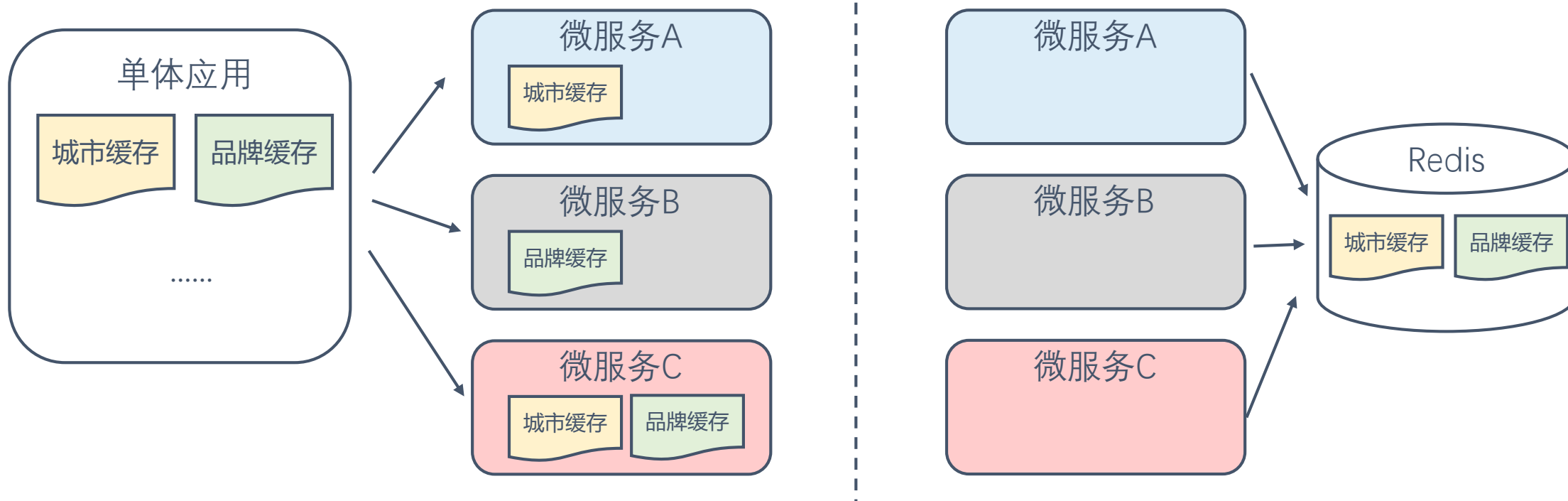
- 生产对比时发现缺了需求，再紧急补，影响原来上线计划
- 同一个需求，微服务里做一份，原来的单体应用里也要做一份
- 仅做在了微服务里，但是因为微服务有问题，切回导致功能丢失

改造期间，与各业务开发团队及时沟通，做好需求同步

# 分布式缓存变化

单体应用的local cache分散到各个微服务中，反而增加了资源的消耗

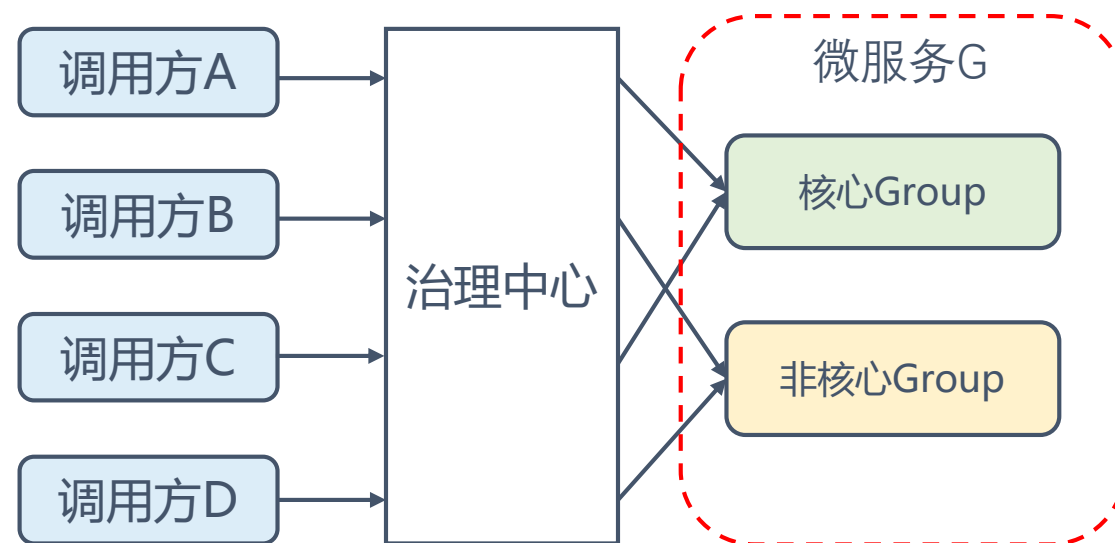
本地缓存往redis里迁移，牺牲一些性能来节省资源



# 服务流量治理

微服务化后，每个微服务可能会承接多个调用方

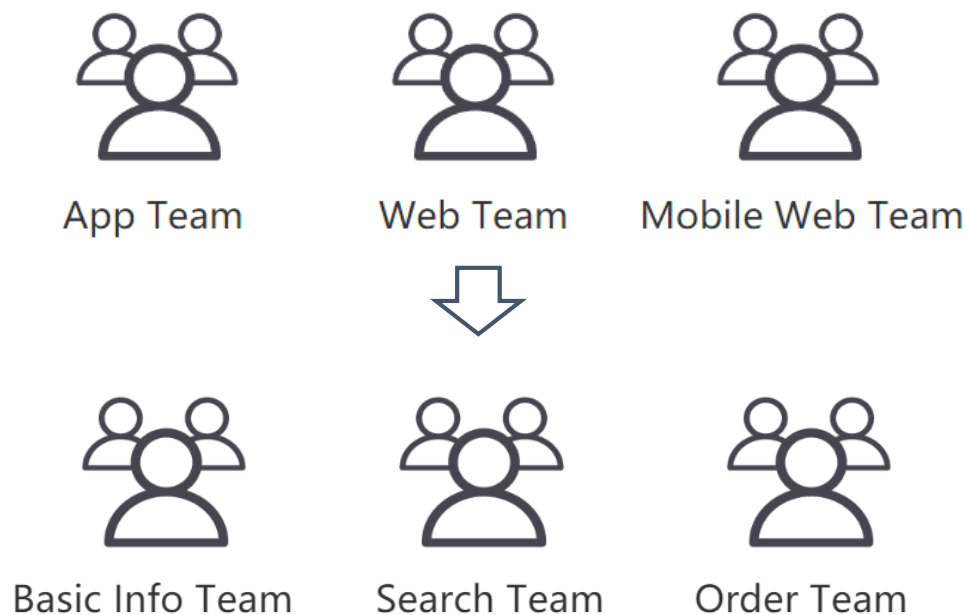
- 流量管理：黑白名单，服务级qps限制，应用及qps限制等
- 拆分核心和非核心Group，设置不同的降级策略，进一步提高系统稳定性



## 团队的变化

在逻辑收口到中台微服务后，多平台的团队将重组，并且将微服务分配到每个人/团队管理

- 按照领域划分团队可以最大程度上减少团队之间的应用重合度



## 6. 总结和展望

## 总结

- 微服务化会改变原有生产流程，是对老的生产全链路的再升级
- 微服务化带来了轻量化的小规模应用，高复用性和灵活性的同时，对例如稳定性、一致性、减少生产事件等等问题有一定的帮助
- 微服务化的同时，组织结构也会相应发生变化
- 微服务化会是一个长期持续的过程，分分合合，不断平衡



## 展望

- Service Mesh——Istio的实践
- Scrum团队的划分边界有待进一步探索
- 自动化测试高效化和智能化探索

## 一些建议

- 在开始微服务化之前一定要想明白
- 您理解的微服务是什么？
- 为什么要微服务化？
- 目前公司基础建设程度？
- 是否值得花成本来做？
- 是否符合公司战略？

The background is a deep blue with a complex wireframe pattern of rectangular shapes, resembling a stylized city skyline or a data visualization. A bright, glowing blue line runs diagonally from the top left towards the center. The word "THANKS" is written in large, white, sans-serif capital letters, centered horizontally and slightly offset vertically. A horizontal blue glow or lens flare effect passes through the middle of the text.

THANKS