百度智能小程序流量分发架构设计与实践

百度 • 史南胜

2020.10.22-10.24

目录
Contents

百度智能小程序

双引擎 → ← **开源联盟**

50万+ 入驻小程序

5亿+ 开放生态MAU

240+ MAU过百万

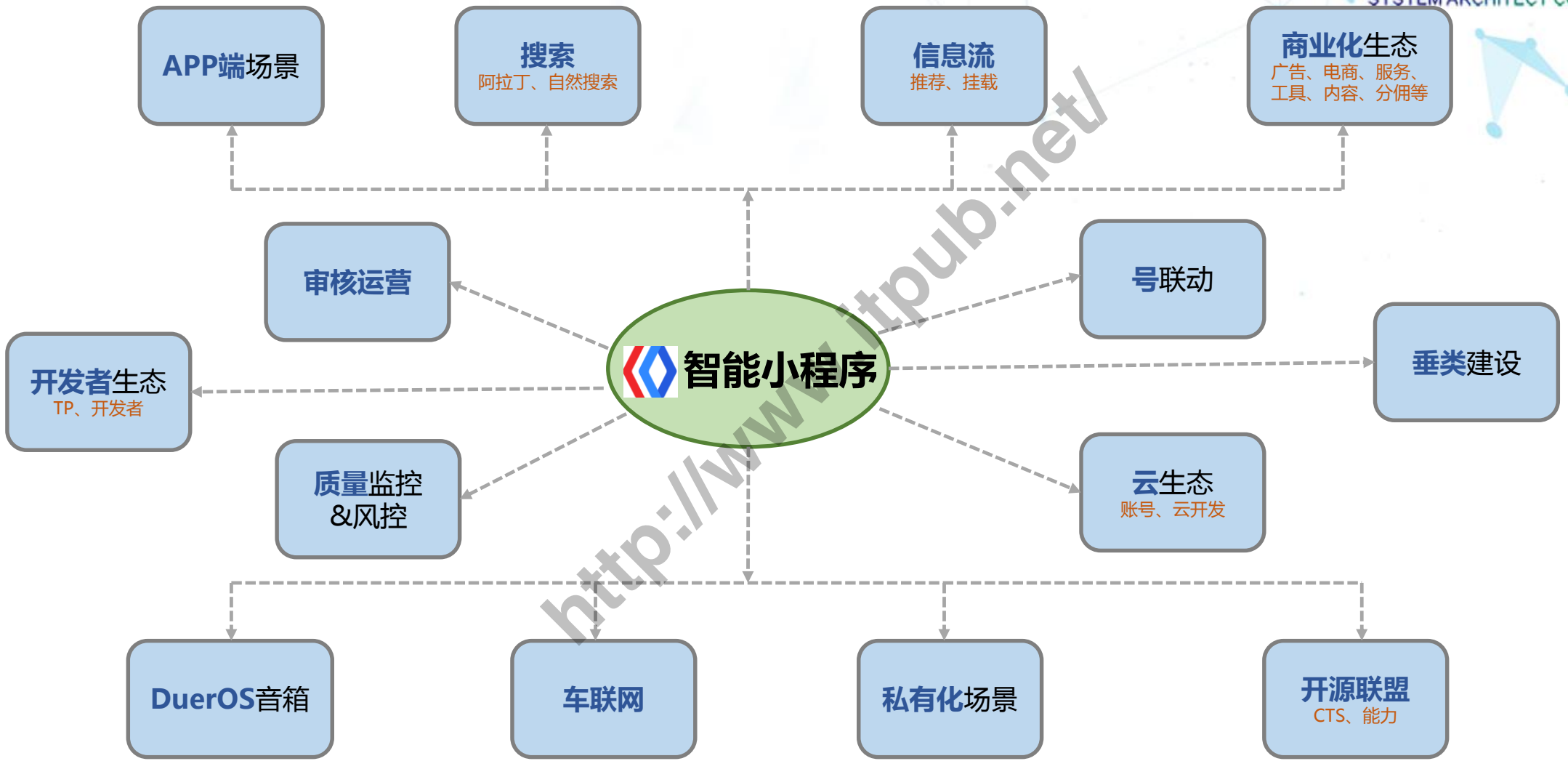
55+ 开源联盟伙伴

占据搜索流量分发**1/3**　　信息流分发**1/4**

百度APP**2亿**日活　　**80+**小程序入口

小程序，大连接

不仅仅是产品解决方案



水电媒

双引擎分发形态

搜索

自然结果

智能小程序被百度收录后，用户在百度App搜索关键词可展现相关结果

阿拉丁结果

定制阿拉丁样式更加丰富，用户可在卡片上与智能小程序进行简单交互

智能小程序单卡

若该智能小程序被搜索收录，用户就有几率在搜索结果页中看到小程序。

语音直达

百度App语音输入“小程序名称+小程序”，可直接打开智能小程序

Feed

信息流直接推荐-三图

信息流直接推荐-左文右图

信息流直接推荐-大图

信息流直接推荐-视频

信息流落地页-自动挂载

初级卡

中级卡

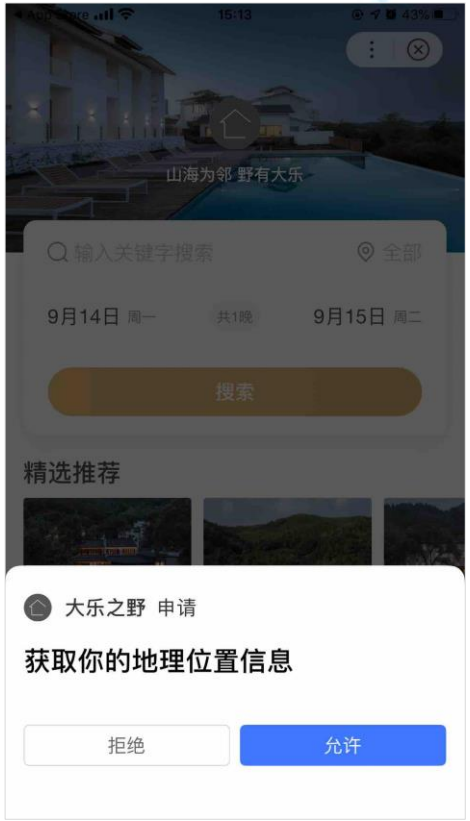
高级卡

品牌服务卡

开源联盟分发形态

爱奇艺 • 视频小程序

小红书 • 民宿小程序



引进来，走出去

模块多，链路长，生效周期要求高

- 资源web化、发现、抓取、渲染、索引、排序、展现、分发...
- 适配、规则

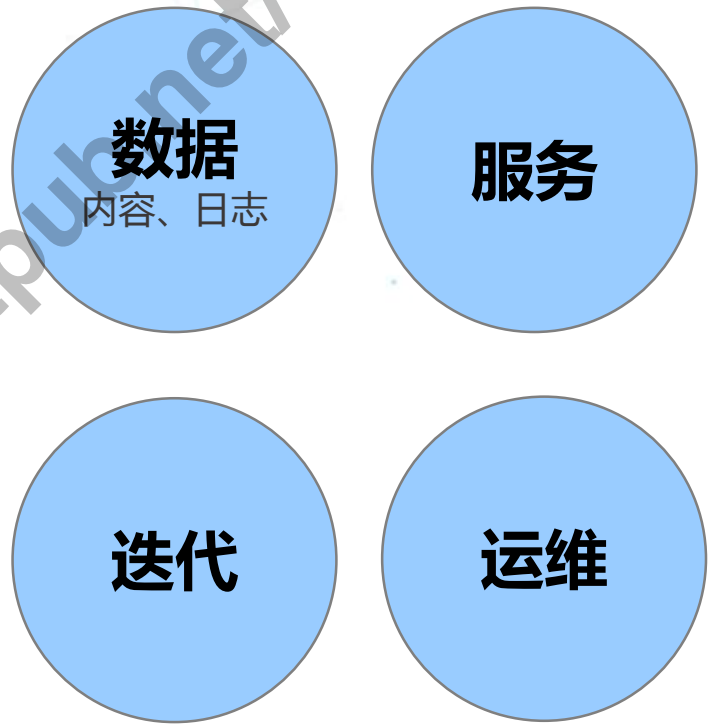
形态多样，数据量增长快速

- 行业、场景、图谱卡
- 几百个、几万、几十亿、几百亿....

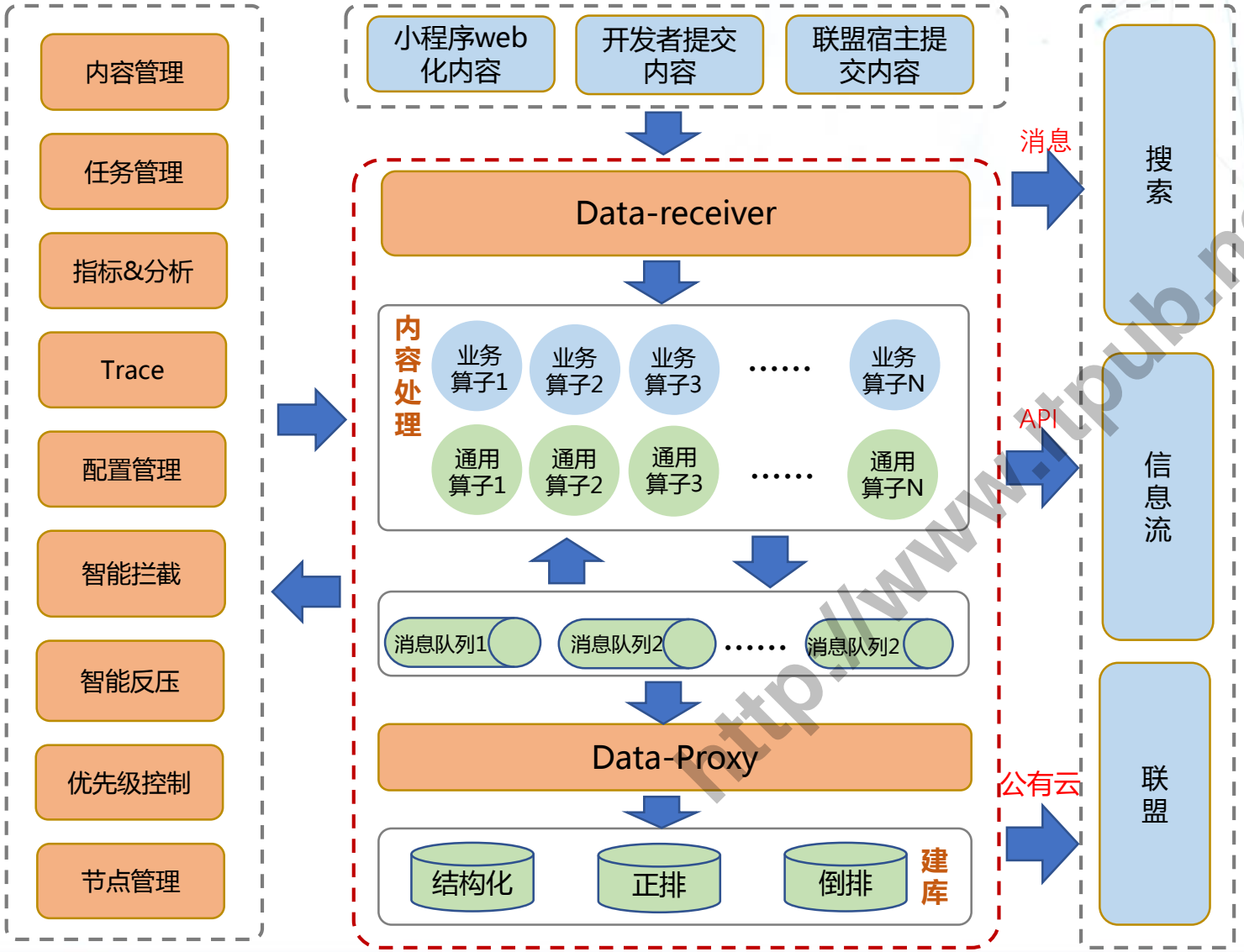
检索流量大，稳定性要高

- 每秒数万次请求，满足极端高可用性，极短时间的故障都可能引发大量的拒绝
- 服务质量

技术上关注什么？

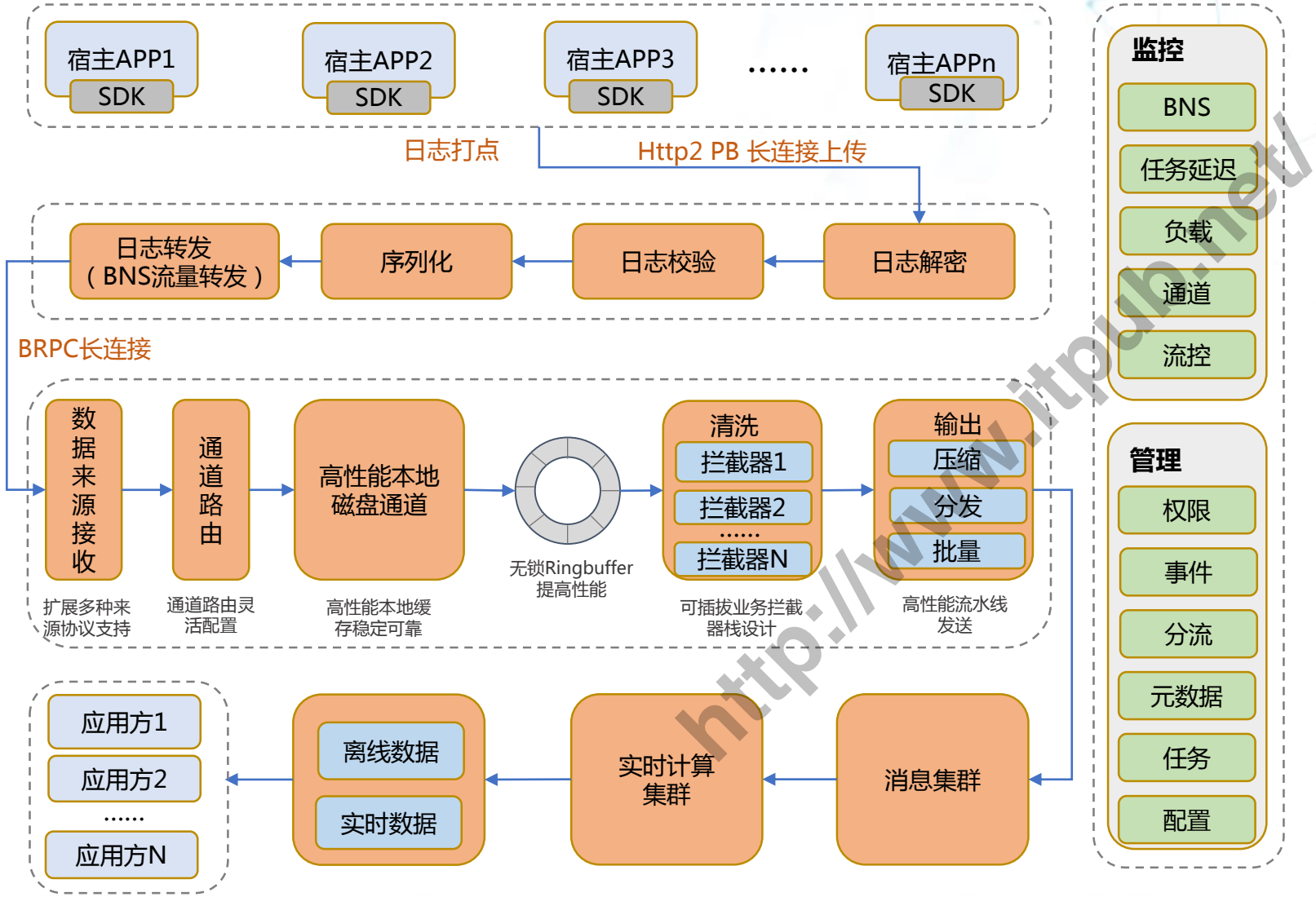


小程序内容分发架构



- **越来越多的场景**
  - ✓ 实时生效配置
  - ✓ Control_Workflow
  - ✓ 不搞平均主义，避免大锅饭！
  - ✓ 计算优先
  - ✓ Schema化，无代码上下线！
- **全面微服务化**
  - ✓ 以ID为控制单元 & 消息
- **多层级监控和自适应策略**
  - ✓ 配额、权限、云控
  - ✓ 自适应：自拉黑、自恢复
  - ✓ 备用通路，成本！
  - ✓ 柔性处理不雪崩

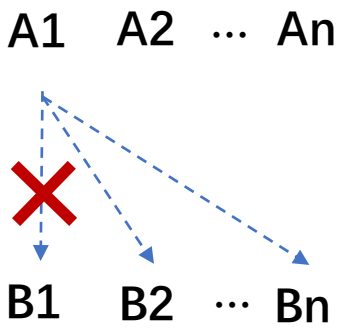
高并发下的实时数据计算

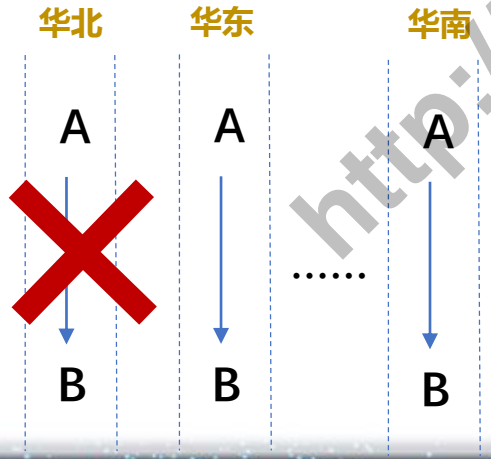


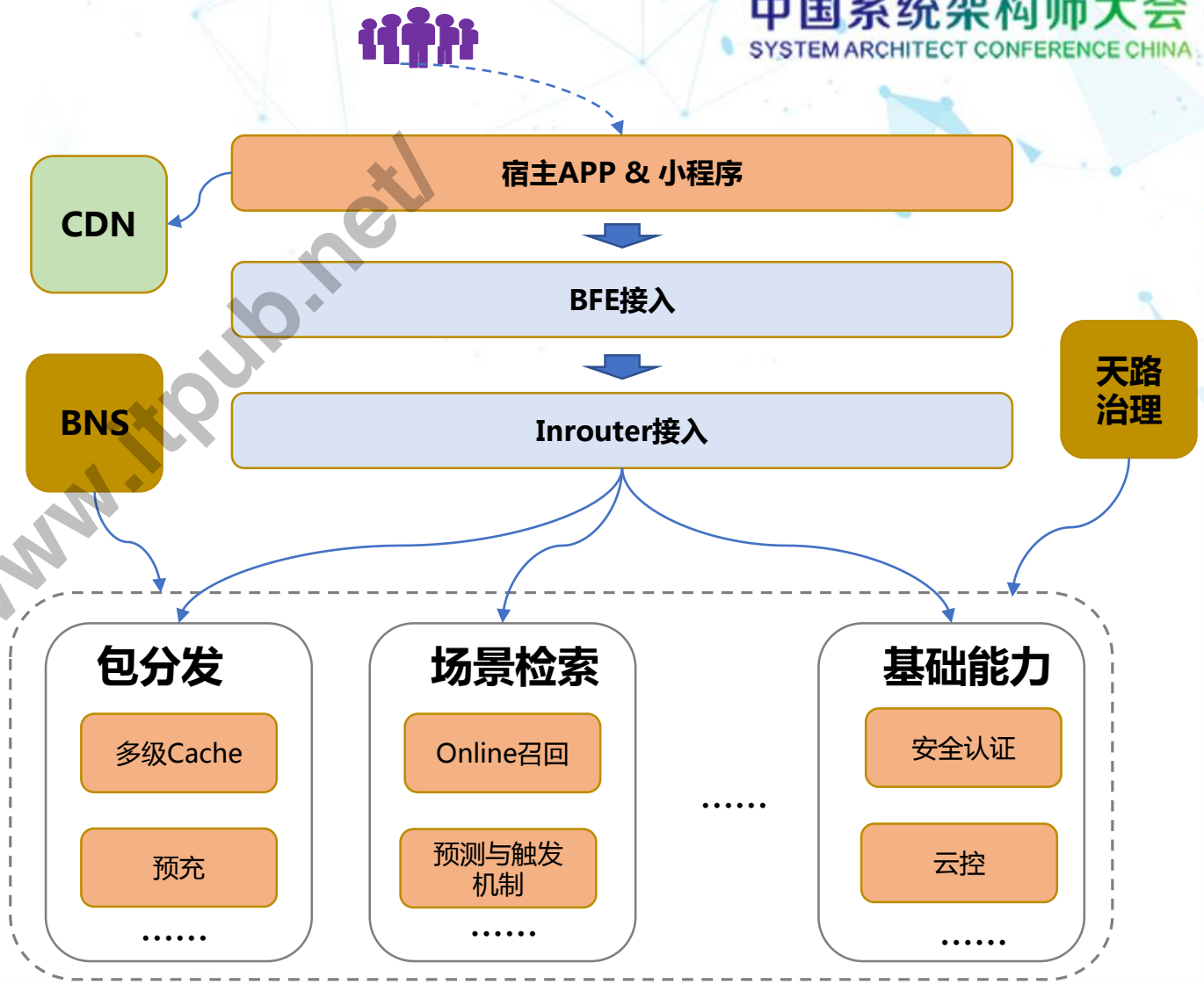
- **流式计算 + 函数式编程**
  - ✓ 组件式开发
  - ✓ 原生语言，Scala函数式编程
  - ✓ 更快、更适合
- **精细化**
  - ✓ 效率优先
  - ✓ 框架化、平台化、自助化
  - ✓ 全链路监控
  - ✓ Mem + Channel
  - ✓ 多数据副本
- **灵活可伸缩**
- **智能调度**

在线服务：稳字当先，预防为主

- 再好的服务，没有规范是不行的
- 分级发布、持续发布、幂等原则
- 逻辑服务单元拆分
  - ✓ 隔离
  - ✓ N+1
- 服务分级：核心链路、非核心，保护机制
- 容灾（有的放矢）：服务、数据
- 调度（准、快）：流量、资源、备份机制

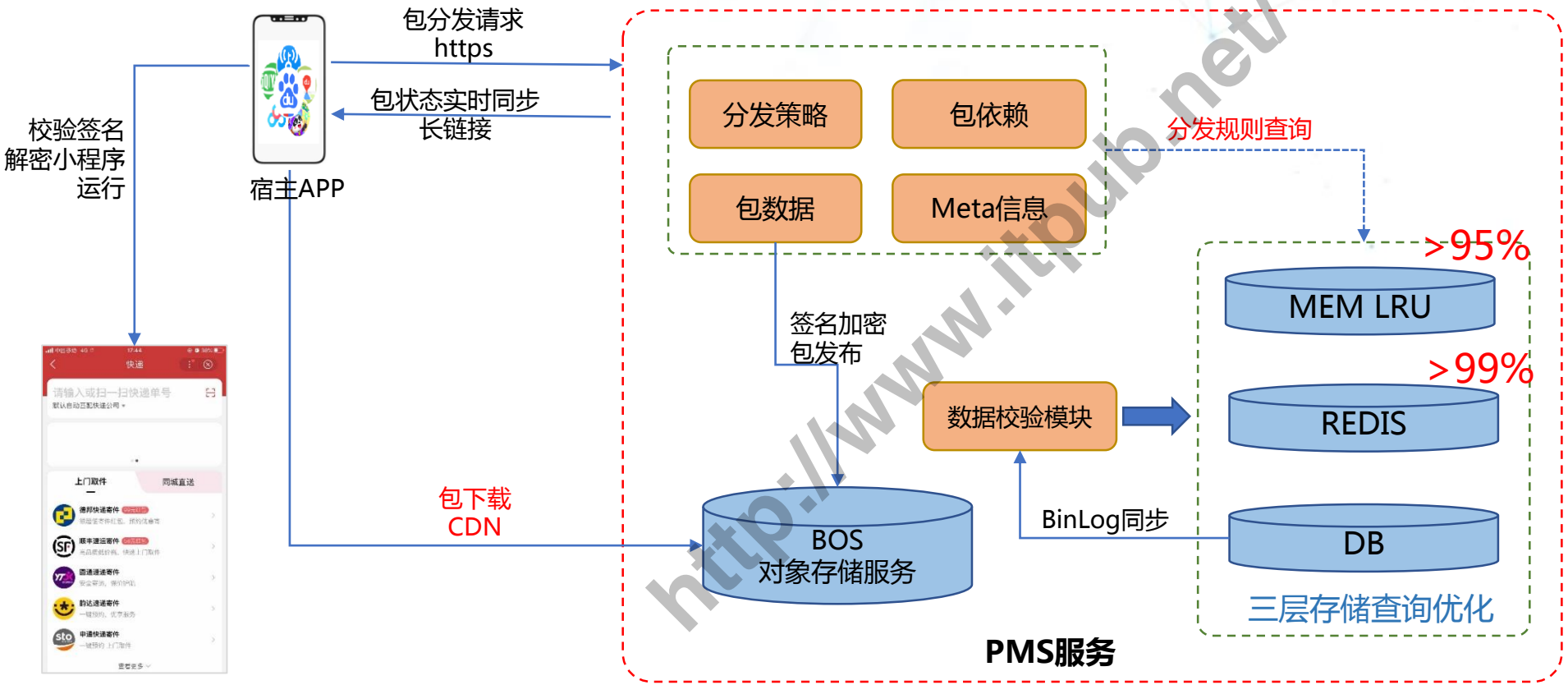






重视Cache的价值

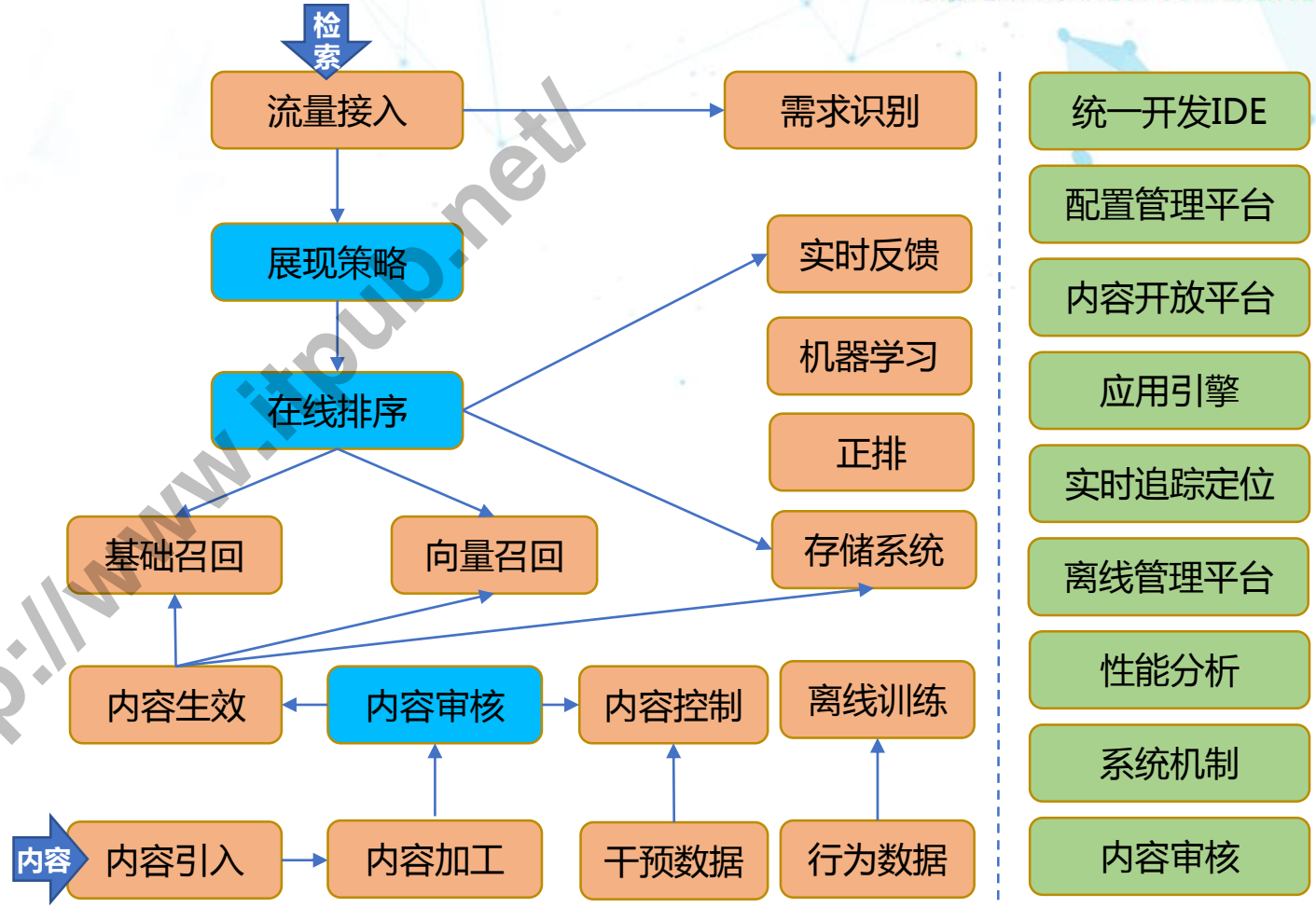
- 挑战：承接全网每日几十亿的包分发请求



- **策略复杂**
  - ✓ 多维度多分发策略支持
  - ✓ 实时分发状态校验
  - ✓ 多版本适配机制
  - ✓ 干预机制
  - ✓ .....
- **成本 、容灾？**
- **远程 vs 本地**
- **重视本地cache**

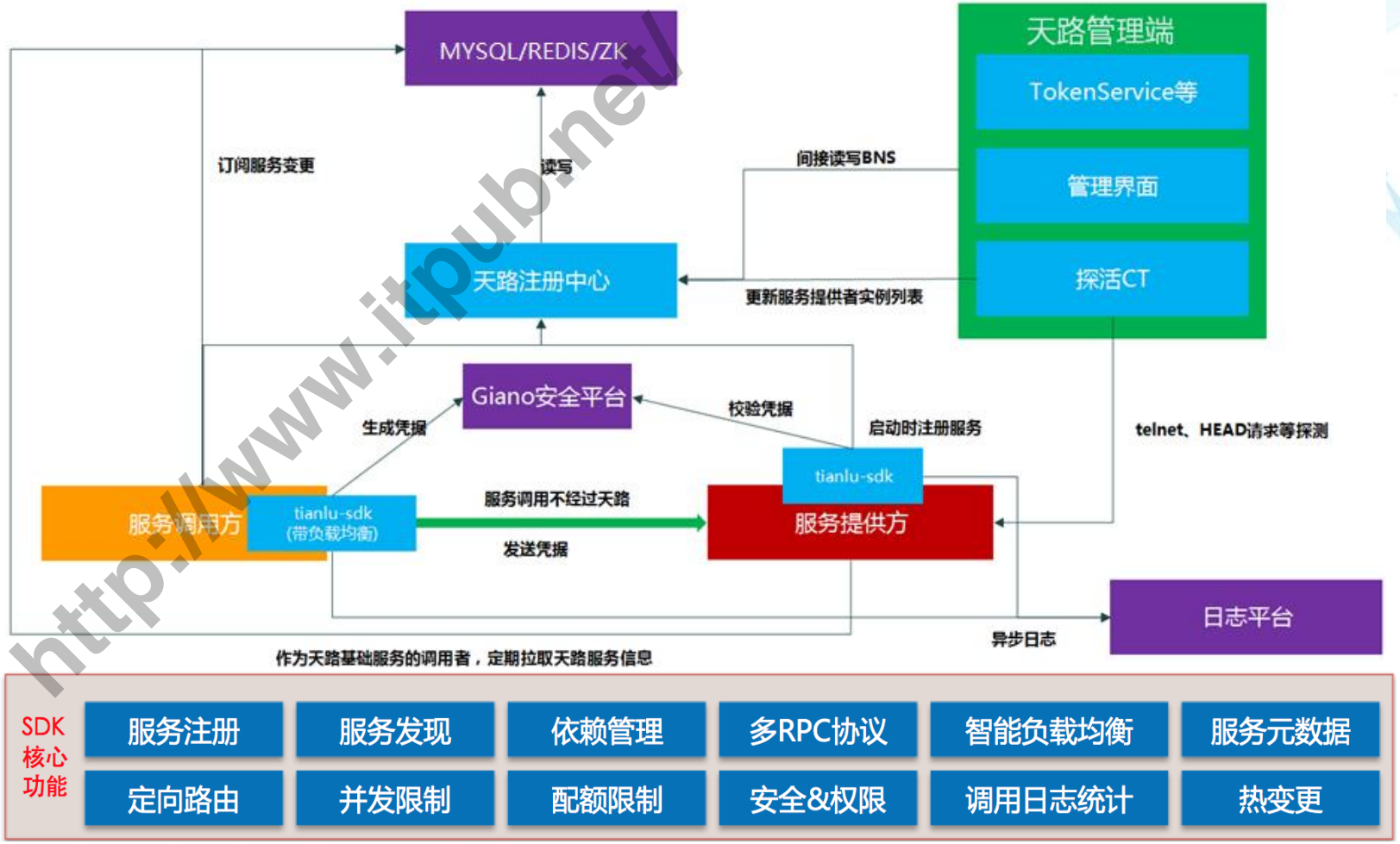
持续自我演化、技术驱动增长

- **高可持续迭代**
  - ✓ 业务分层：三层业务框架
  - ✓ 业务复用：打造沉淀和复用能力
  - ✓ 业务迭代：微服务化，自助化研发工具链
- **高性能**
  - ✓ Cache for Search
  - ✓ 性能、容灾、远程、本地
  - ✓ 成本？ Level? SSD?
  - ✓ 淘汰机制、智能预取
  - ✓ 串行 vs 并行（服务、策略）
- **小程序**
  - ✓ openCard、离线的高可用、高实时
  - ✓ 退场
  - ✓ 降级



服务高可用提供，依赖治理

- **集服务注册发现、服务管理等功能于一体的分布式服务治理平台**
  - ✓ 多协议支持(HTTP+JSON, Baidu-RPC)
  - ✓ 多语言支持(Java，Go）
  - ✓ 静态调用、动态调用、元数据
  - ✓ 权限及安全控制
  - ✓ 探活与报警
  - ✓ 软负载均衡
  - ✓ 路由功能
  - ✓ Mock功能
  - ✓ 日志的分析与统计
  - ✓ DTC分布式事务支持

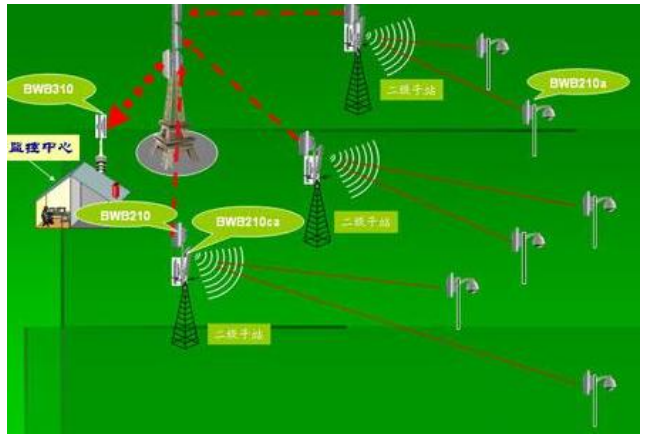


业务快速迭代，持续引入新功能、新依赖

用户量、数据不断增长

如何持续保障系统的高可用性？

两手抓，两手都要硬



防火

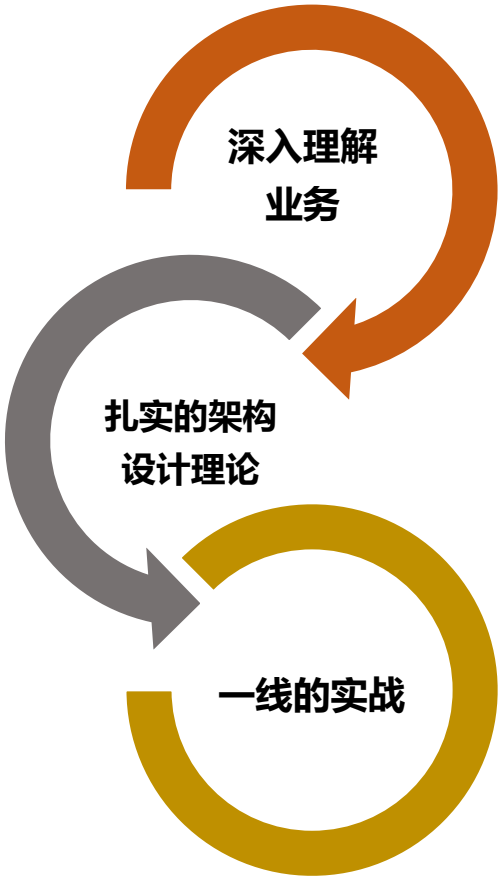
不出问题：减少故障出现的次数

救火

尽快解决：减少故障处理的时间

高可用 != 止损

预防：为异常而设计



- **消灭“未定义”的场景**
- **定规范，并执行**
  - ✓ 代码
  - ✓ 日志
  - ✓ 架构
  - ✓ ……
- **终极目标： “万无一失”的设计**

```
try {
    .......
    .......
    .......
} catch(Exception e) {
    .......
    logger.warning(“XXX Error”)
    ......
}
```

举例：内容全网分发平台

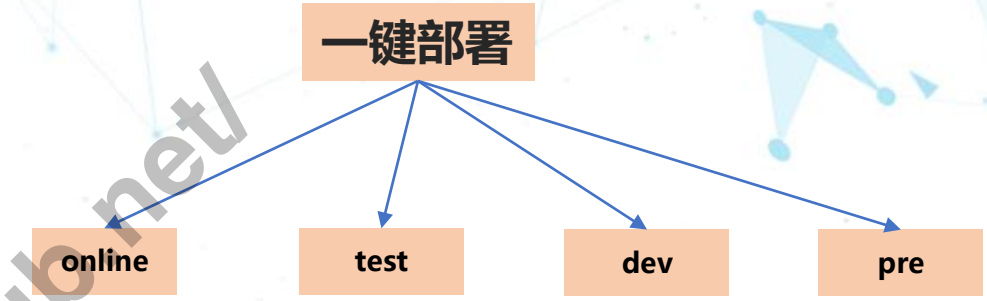
- ✓ 几十种状态的扭转，均有状态变更
- ✓ 上百种的场景适配
- ✓ 任何状态生效和环路都会出现问题

预防：修内功

- 持续集成、持续交付、持续部署
- 修bug ⇄ 写bug

工程能力总分	需求	技术架构	开发	代码准入	测试	运维
***等级	***等级	***等级	***等级	***等级	***等级	***等级
82.5/120	4.57/6	18.60/20	6.18/7	23.16/24	22.45/48	7.53/15

需求	技术架构	开发	代码准入	测试	运维
需求管理	部署时间	分支规范	CodeReview	功能回归	分级部署能力
Bug管理	服务可迁移	提交规范	源码安全	性能测试	自动检查
指标3	指标3	指标3	指标3	压力测试	指标3
指标4	指标4	指标4	指标4	指标4	指标4
.....	指标5	.....	指标5	指标5	指标5
	指标6		.....	指标6	指标6
	指标7			指标7	.....
	指标8			指标8	
	.....			指标9	
				.....	

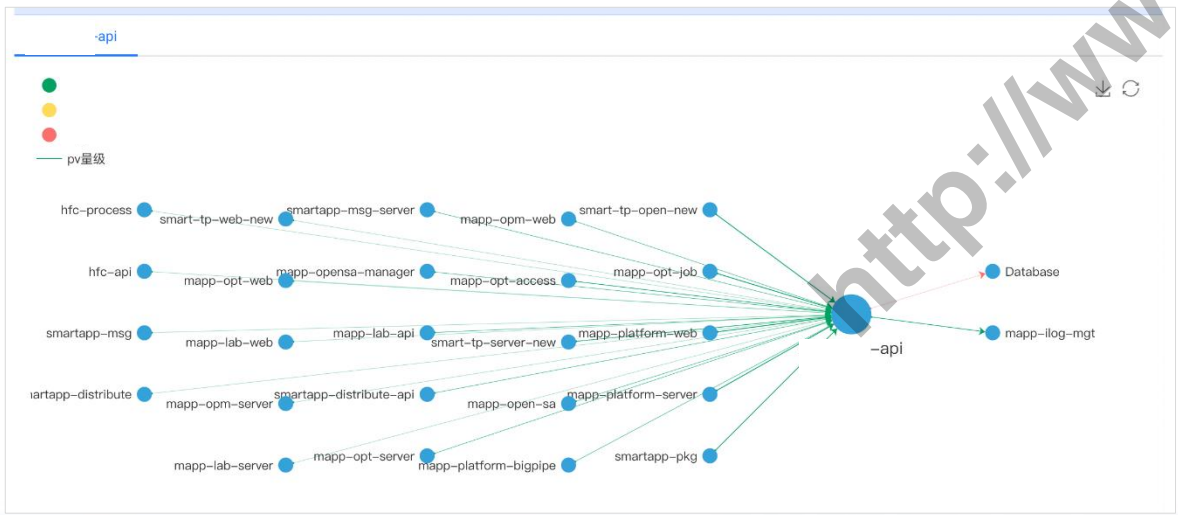


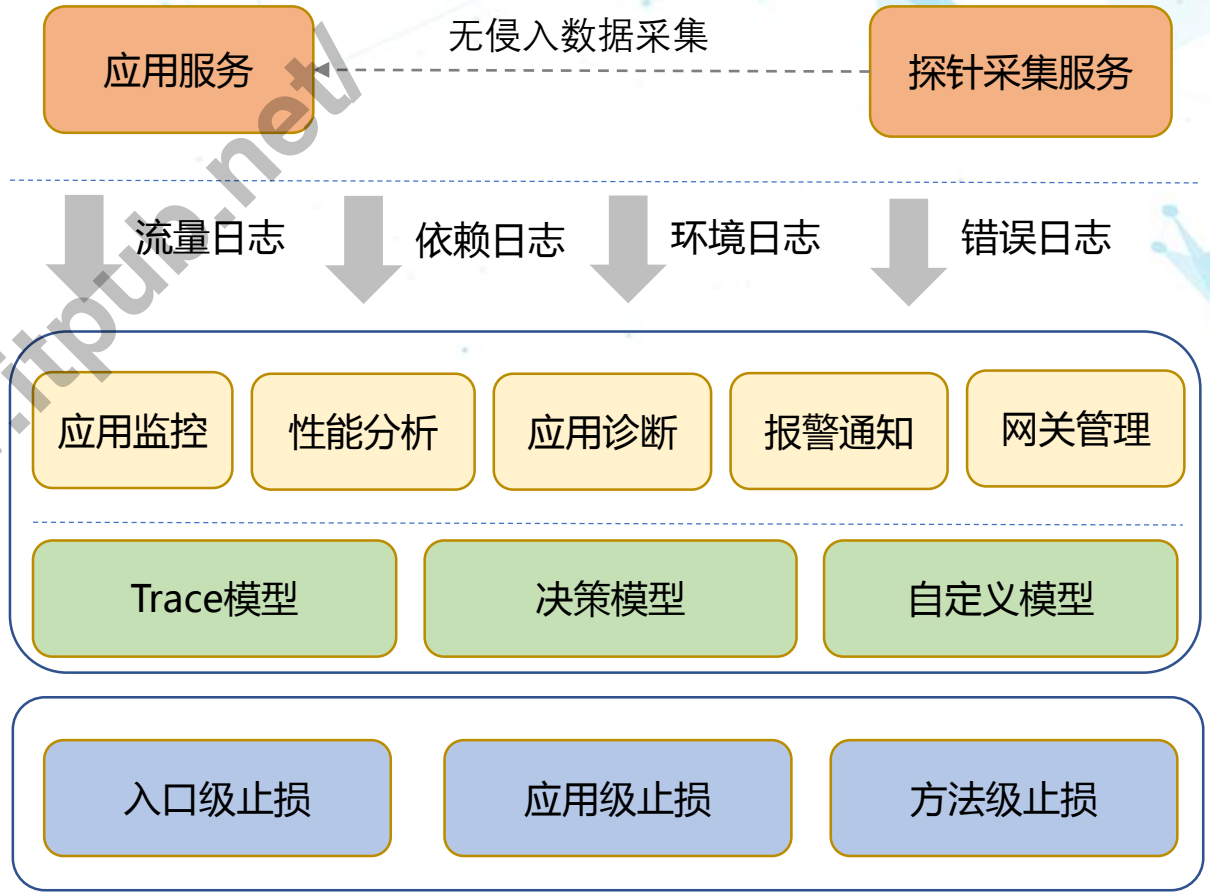
- 尽早发现问题
- 并行开发
- 快速迭代
- 快速发现问题

功能需求迭代，和delay说再见！

发现与止损：完备的监控

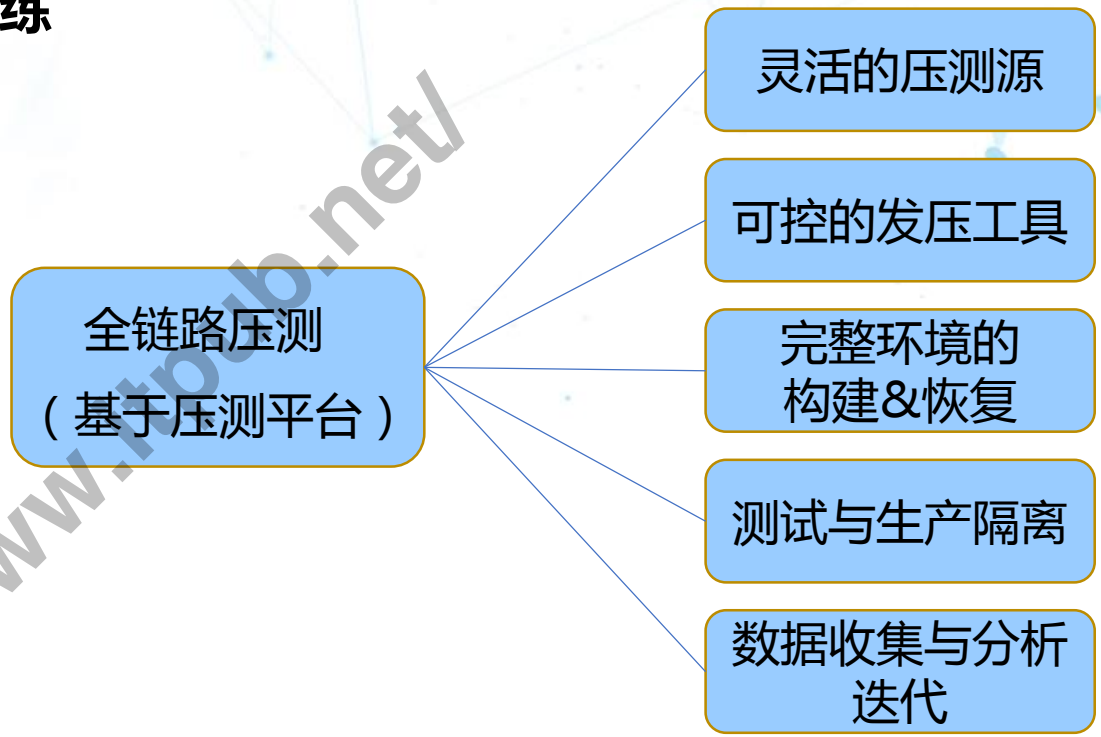
- **大规模、高负载系统问题排查困难在哪里？**
  - 信息量大、位置分散、噪声多
  - 问题难复现
  - 牵扯领域多
  - 很可能引入新的问题
- **根因定位 + 全局视角**
  - 云、管、端





能力验收：混沌工程

- **再好的架构设计，也需要上战场真实规模演练**
- **全链路压测、故障演练、线上流量管控等**
- **核心关注点：**
  - 全链路、高覆盖
  - 模拟真实
  - 灵活、可控、自动
- **混沌工程不是独立的事件：**
  - **标准**：能力等级定义（L0-L5）
  - **剧本**：业务理解力
  - **成本**：人、资源



我们还做了哪些关键的工程实践？

- 弹性资源供给和动态调度
- 容量预估和评估常态化
- 蓝军计划
- 重点基础框架的降级预案和演练
- 服务等级
- 故障复盘（Case Study）
- ……

思考与展望

服务注册与发现

服务调用

服务追踪

分布式事务

服务通信

负载均衡

数据一致性

自动化构建

服务熔断

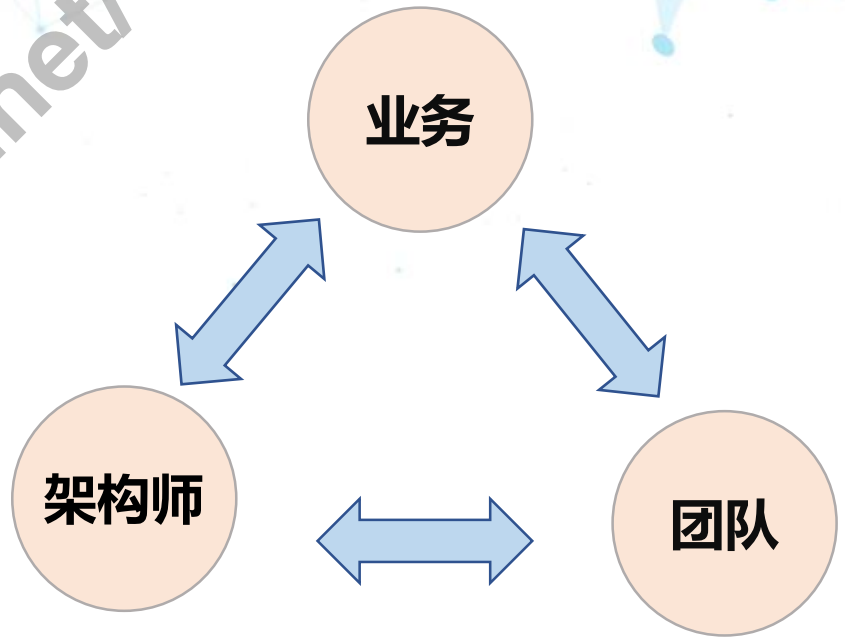
部署升级

服务追踪

服务安全

服务监控

服务降级



为业务而生，仗是打出来的！

Thanks