生产环境稳定性利器
Takin
全链路压测技术交流

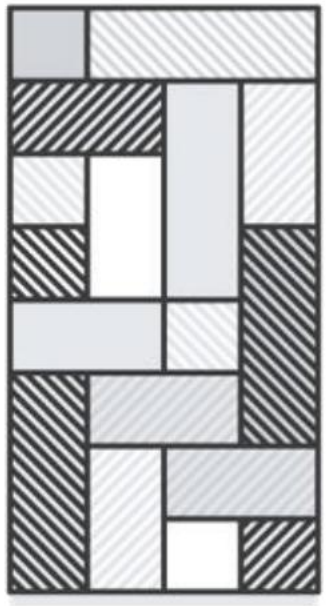
关于Takin

- https://github.com/shulieTech/Takin

- 核心代码做了开源社区版

从单体架构到微服务架构

一个紧密耦合的庞大
应用+竖井式团队



2001

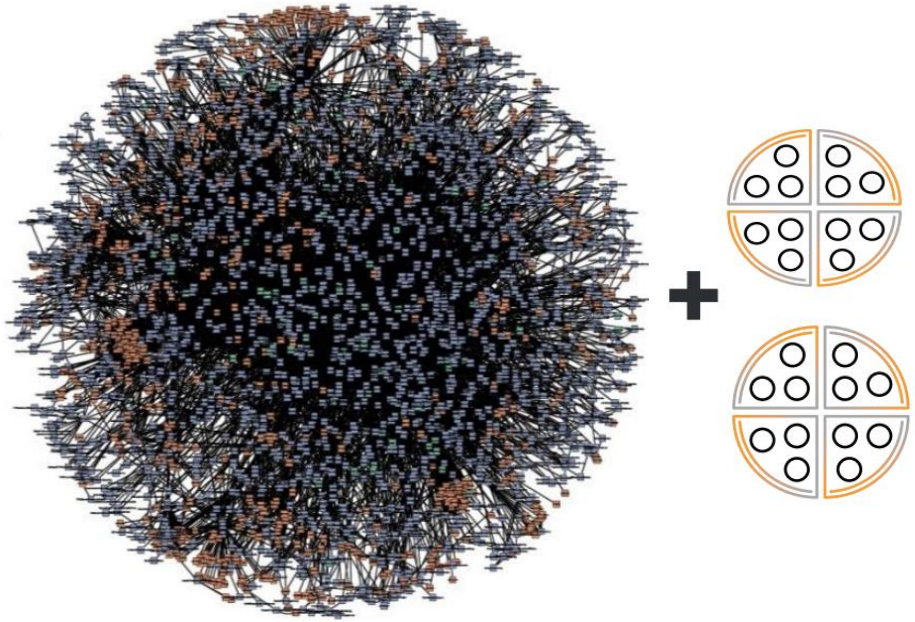
改不动

慢，贵

“创新乏力”

重复建设

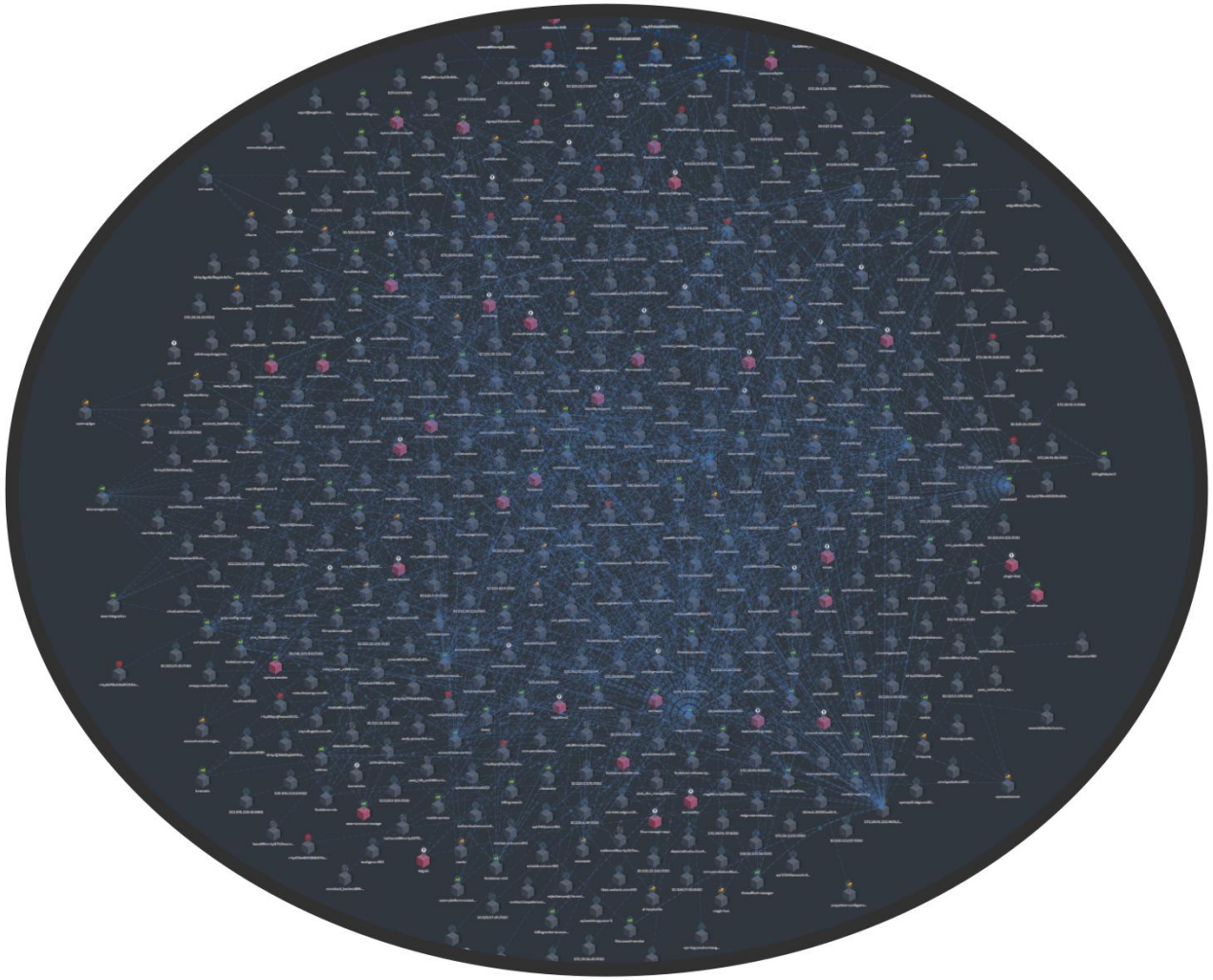
众多的微服务 + 小团队（**2 pizza teams**）



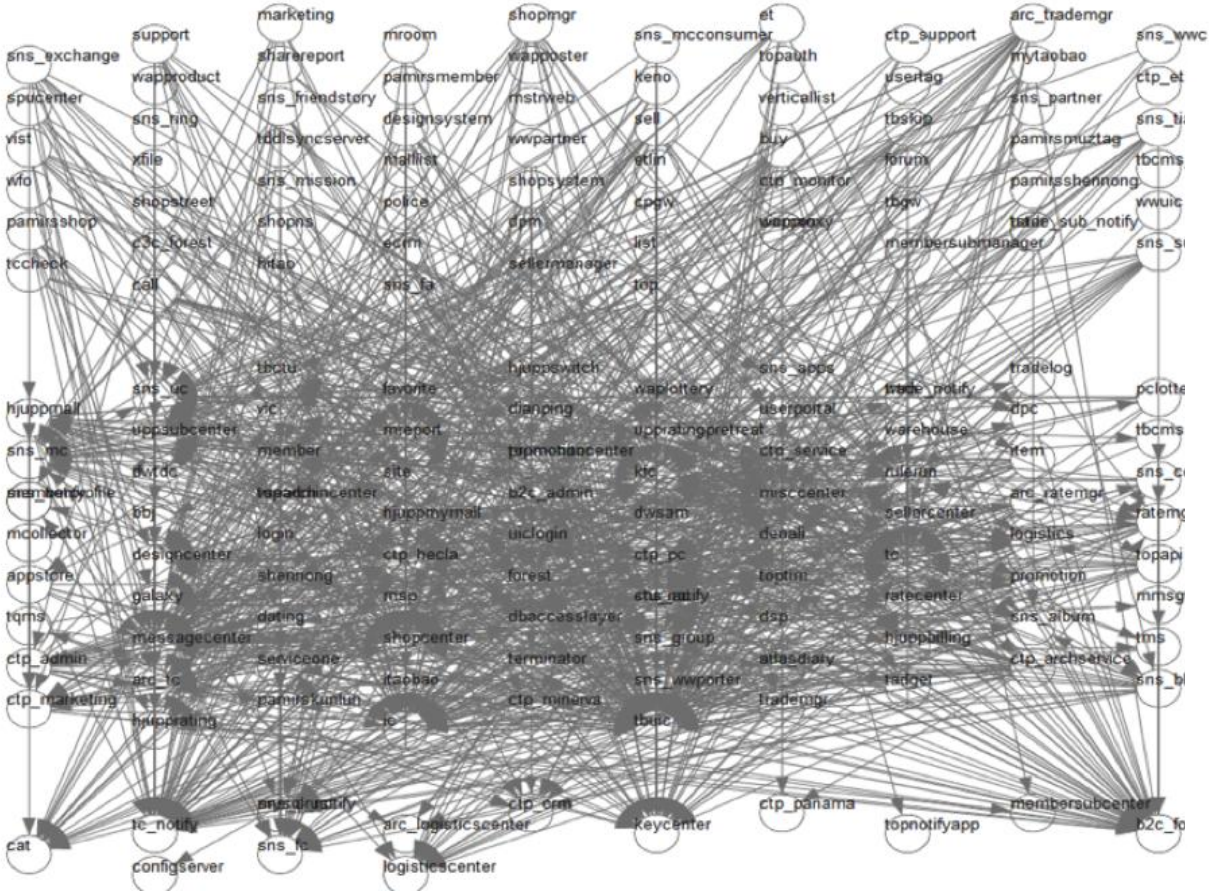
2009

2009年的亚马逊微服务调用集合

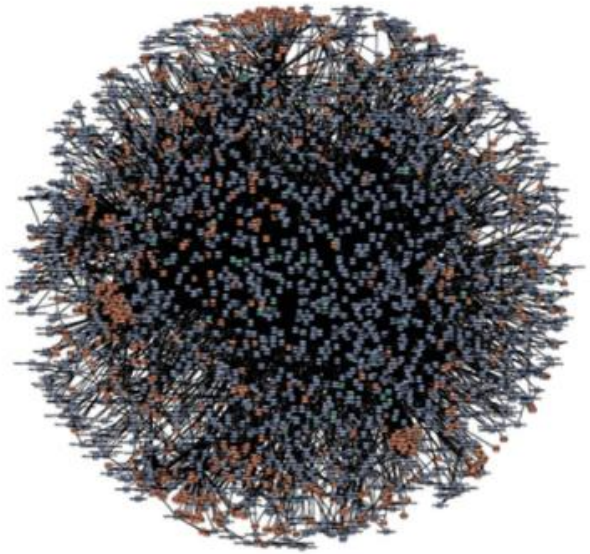
生产环境应用依赖



复杂的服务集合



2012 淘宝网微服务调用关系集合



amazon.com

NETFLIX

复杂的服务集合、复杂的基础设施、复杂的协同体系、超大流量、经常发版本

CDN

网关

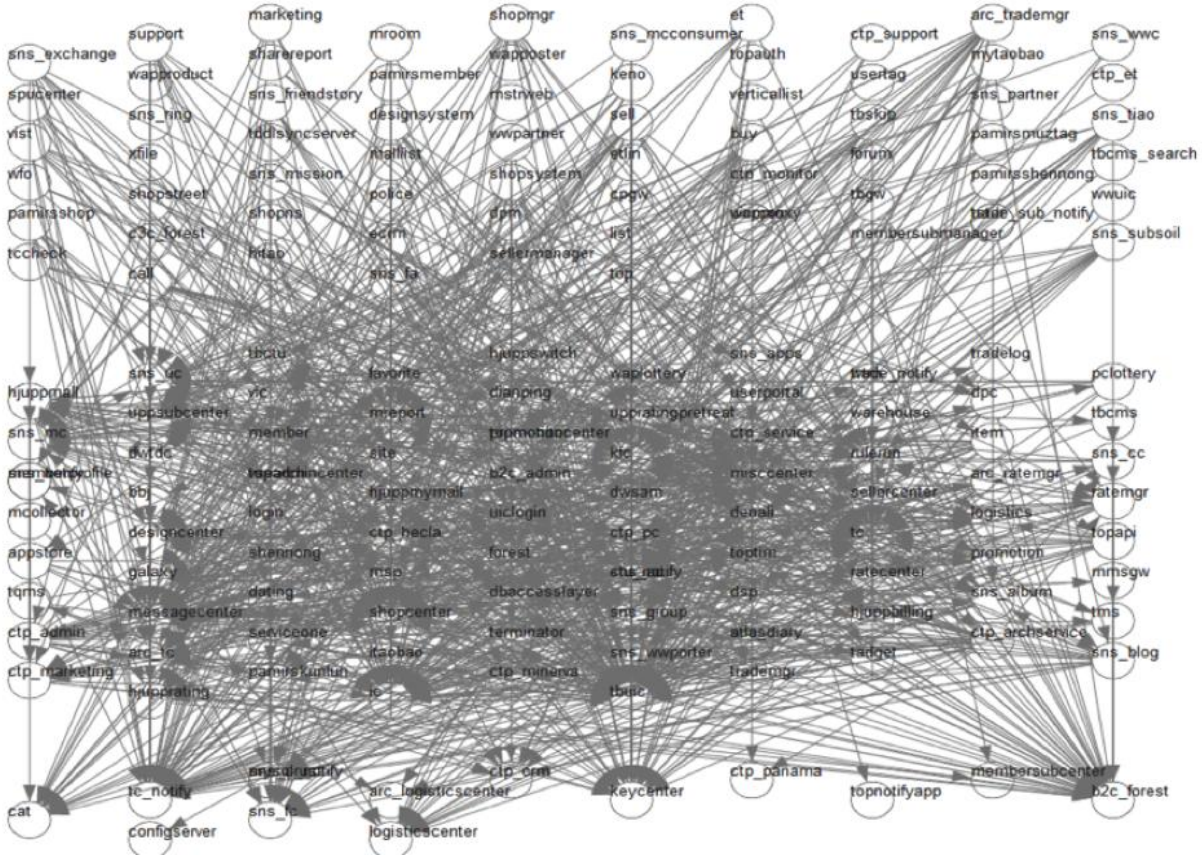
负载均衡

分布式页面系统

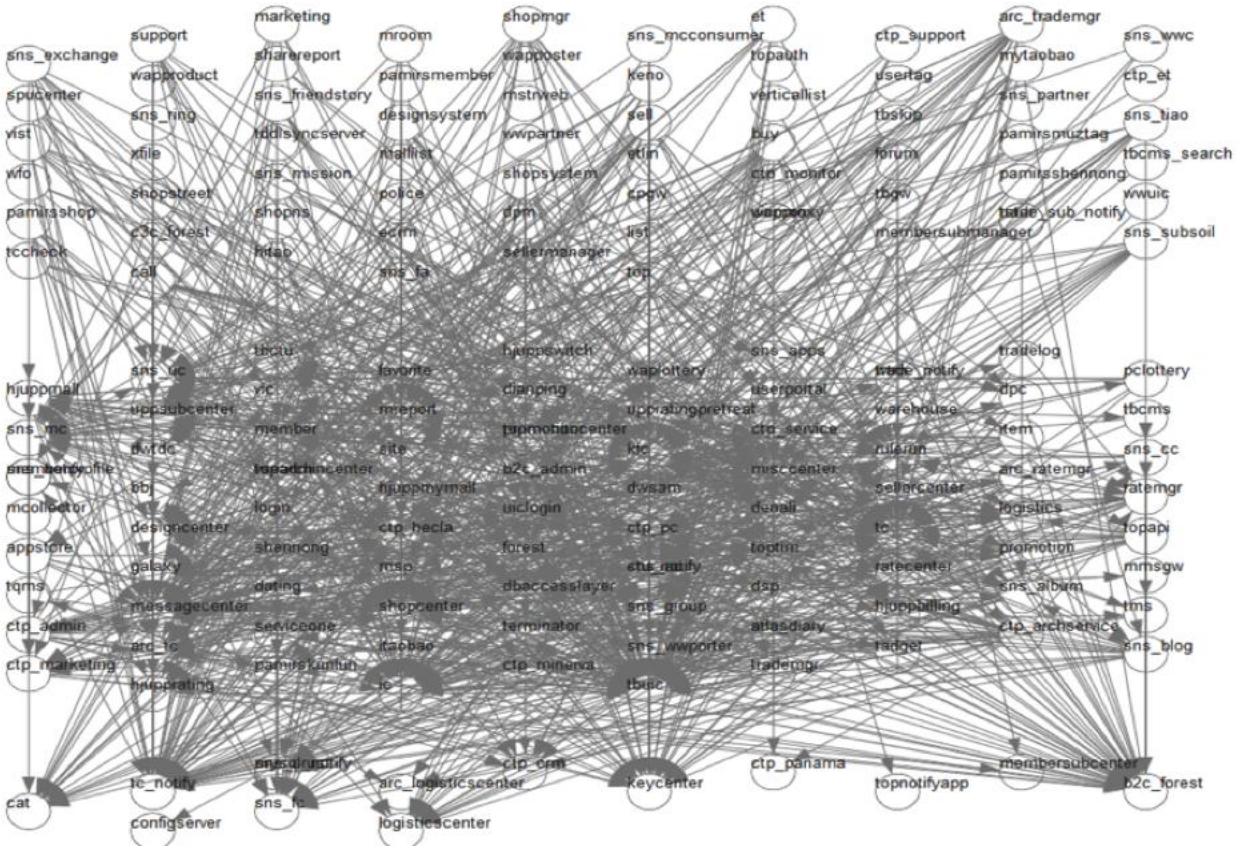
配置中心　分布式服务系统　异步消息

搜索引擎　缓存　存储　计算引擎

网络&基础硬件　第三方服务云



流量冲击超复杂系统，但依赖单点评估



预估业务量级 1

/

单机能力 2

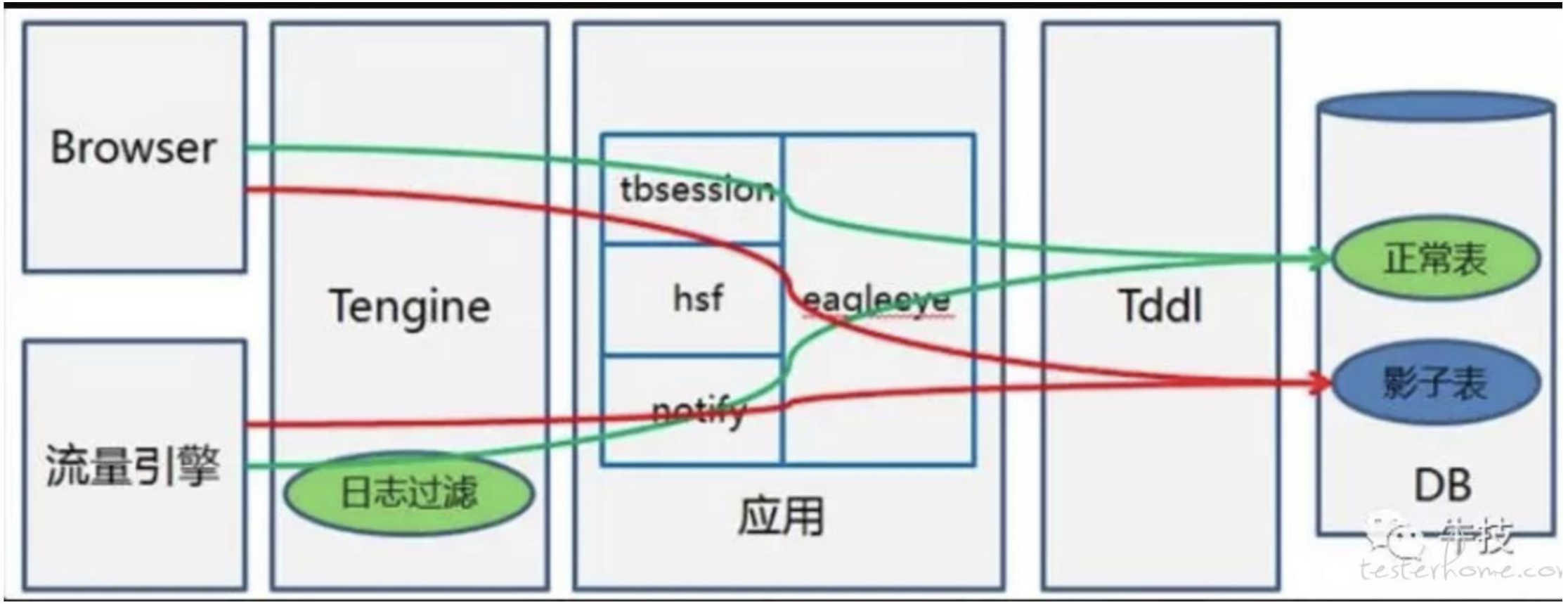
=

最少机器数 buffer

扩容

缩容

淘宝网生产环境全链路压测的实现原理



改造中间件，实现压测数据的识别和转发到影子区域

分布式服务调用框架HSF　　tbsession

分布式数据库中间件TDDL　　Notify

分布式链路追踪框架 eagleeye

各阶段遇到的问题

有没有可能在线上环境进行全仿真的测试?

不可能！！！这对于数据库是更加不可能的，最大的担心是压测流量产生的数据该如何处理，从来没听说过哪家公司敢在线上系统做压测，万一数据出现问题，这个后果将会非常严重

提议阶段，受阻

核心问题：压测数据应该放在哪里?

1.往真实表写入
2.影子库
3.影子表

论证阶段
2013年 7月底-8月中旬

改造鹰眼、HSF、TDDL、Notify、商品中心、订单中心等一系列的中间件和业务系统

从PD、研发、架构师、测试 、运维 加起来有近百人的团队。一起改了2个月

建设阶段
2013年 8月中到10月中

2013年10月15日凌晨2点-4点，第一次演练，将会达到6w/s创建的压力。第一次，演练失败。。流量发不起来。

反复了几次，终于成功。
最后一次生产压测：2013年11月2日 凌晨1点

艰难上线

大部分企业面临的挑战

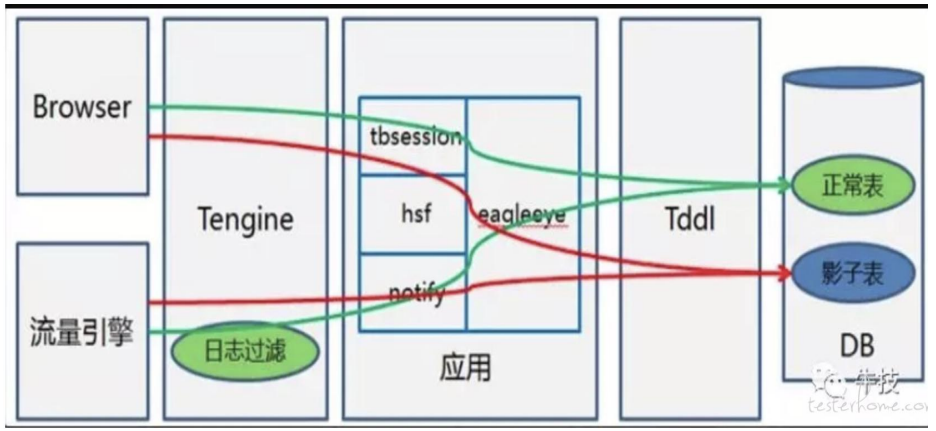
淘宝 Taobao

别人家

改造中间件，实现压测数据的识别和转发到影子区域

分布式服务调用框架HSF
分布式数据库中间件TDDL
分布式链路追踪框架 eagleeye

tbsession
Notify



大部分企业

中间件不统一

系统都是Java

微服务框架有SpringCloud、Apache Dubbo、Alibaba Dubbo等

人才储备

难以短期内吸引到技术专家来做中间件修改

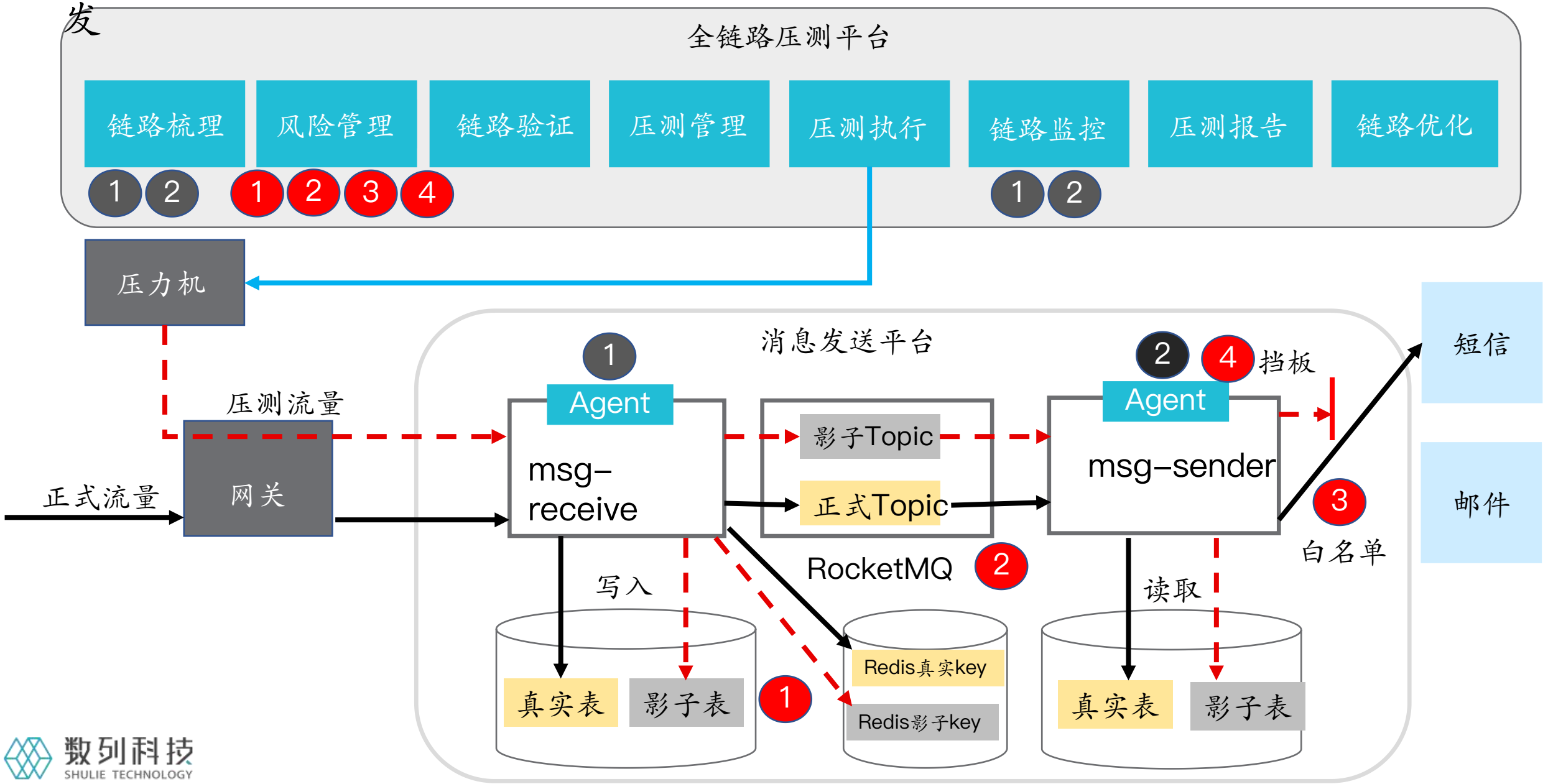
时间成本

对于管理者来说，自己招人做，ROI不高

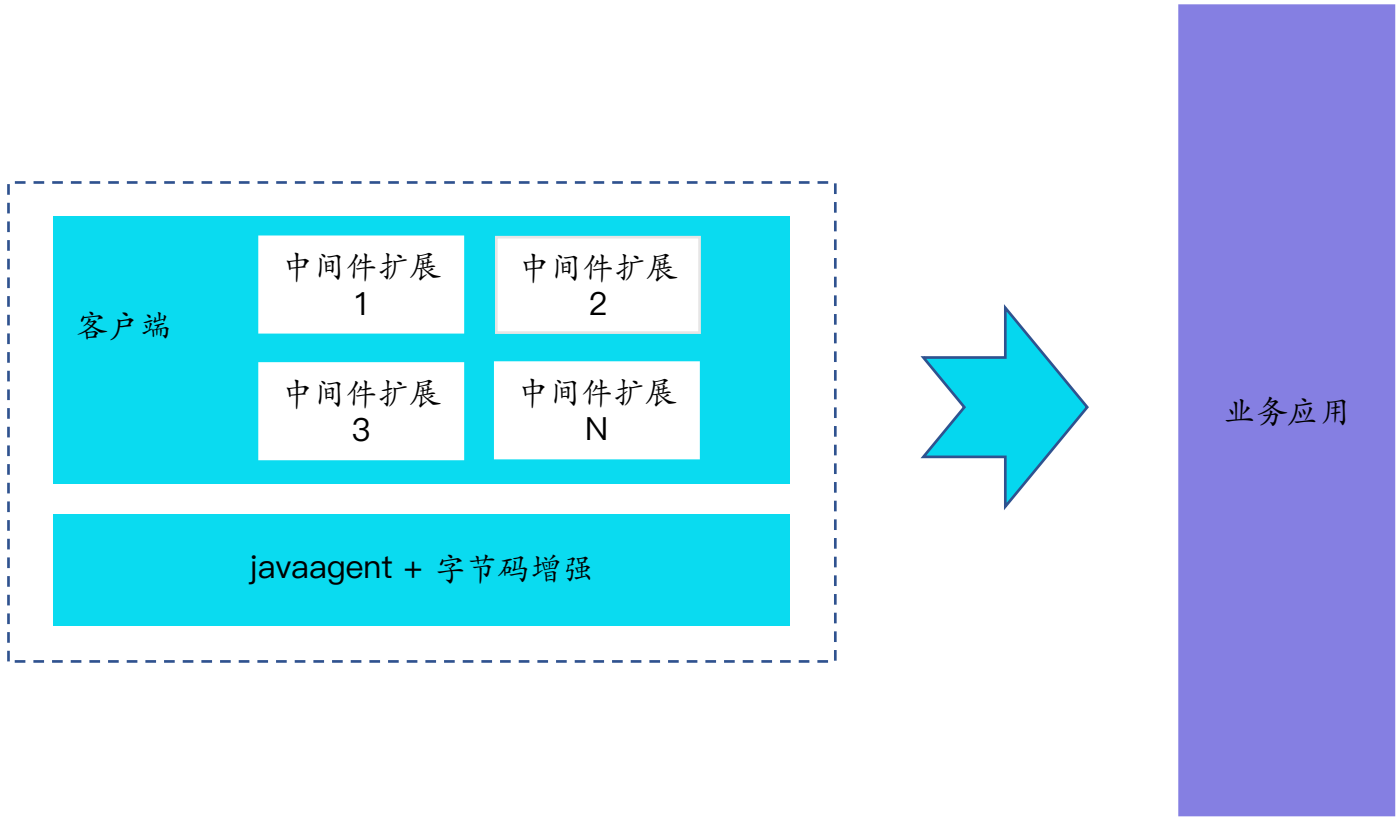
100多个应用，改起来时间周期太长

如何破局？

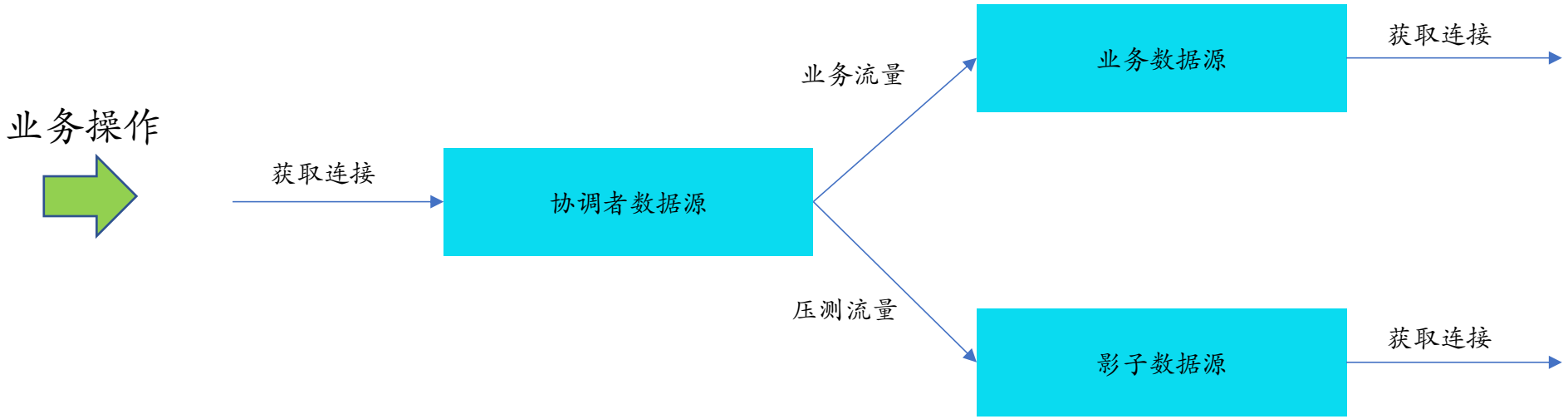
Takin :基于JavaAgent ， 业务系统无需改造，JVM层实现压测数据的识别和转发



Takin - Java应用 无侵入探针零业务改动

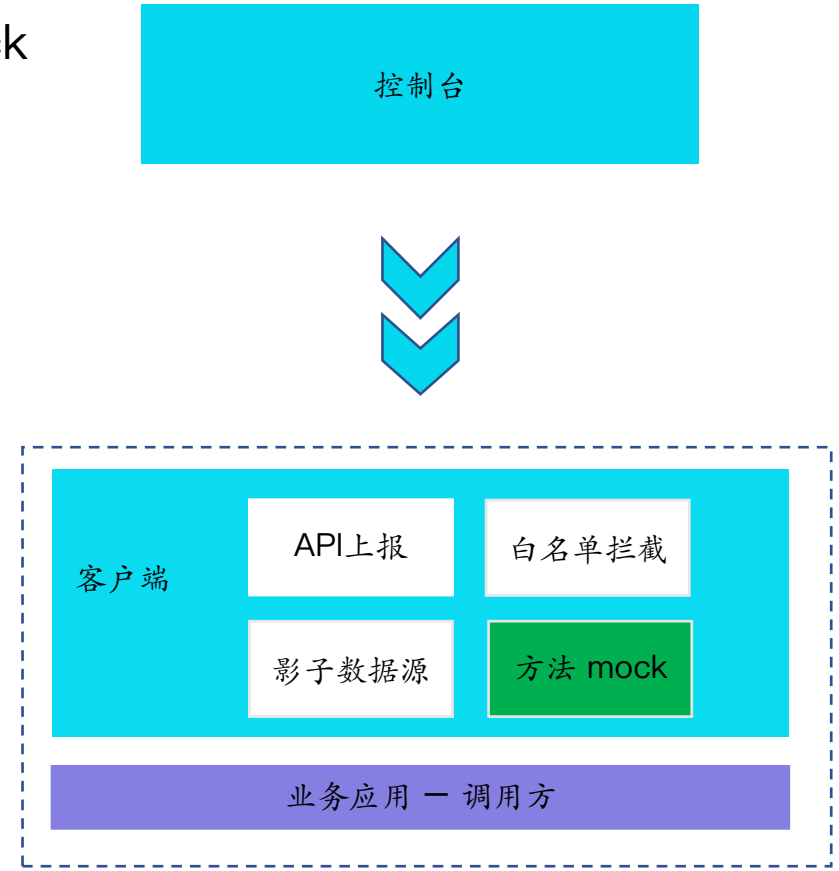


Takin产品特性 – 无侵入探针零业务改动



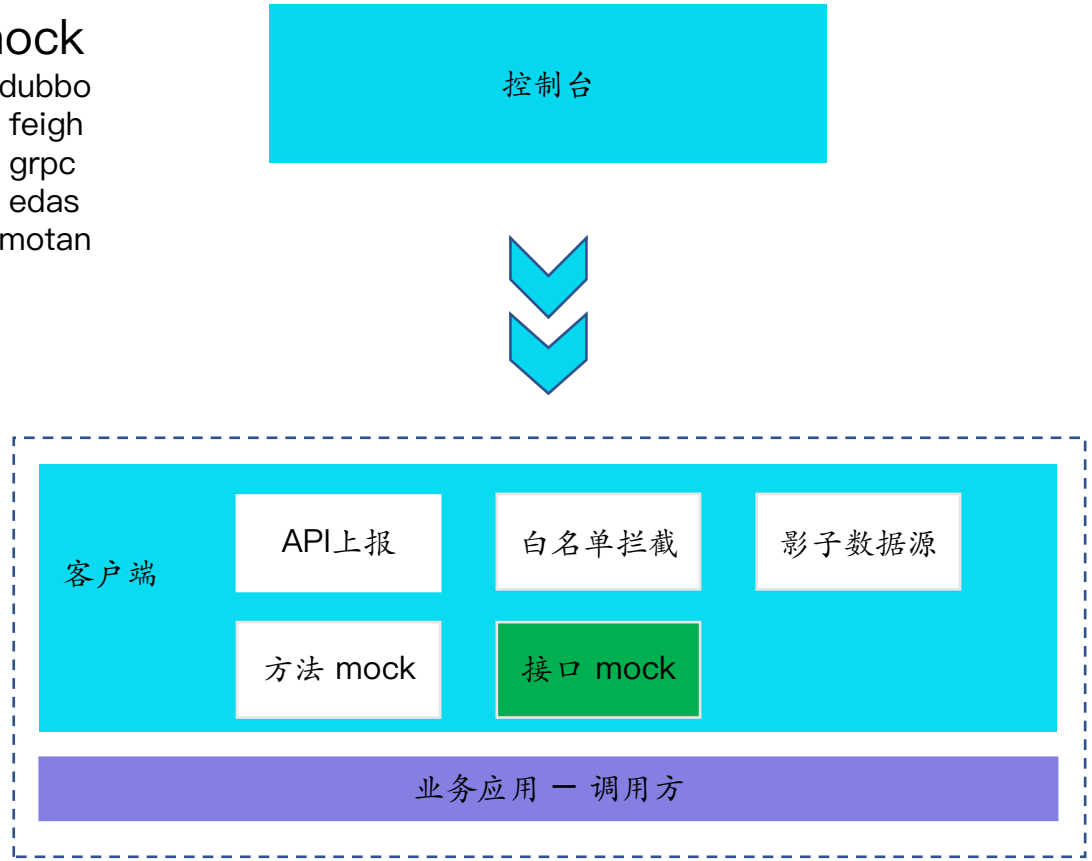
Takin特构演进 – 方法 mock、接口 mock、调用转发

方法 mock



接口 mock

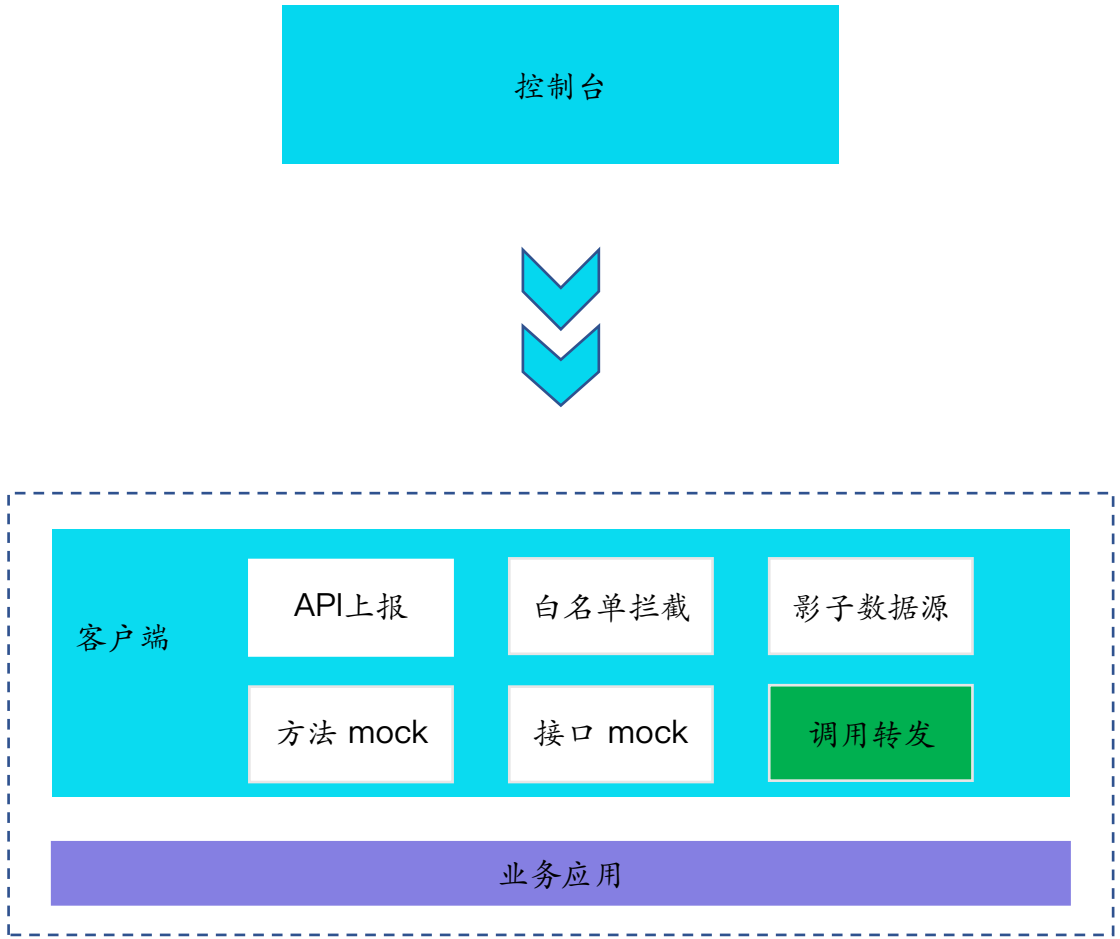
① 支持 dubbo
② 支持 feigh
③ 支持 grpc
④ 支持 edas
⑤ 支持motan



Takin 产品特性 演进 – 方法 mock、接口 mock、调用转发

调用转发

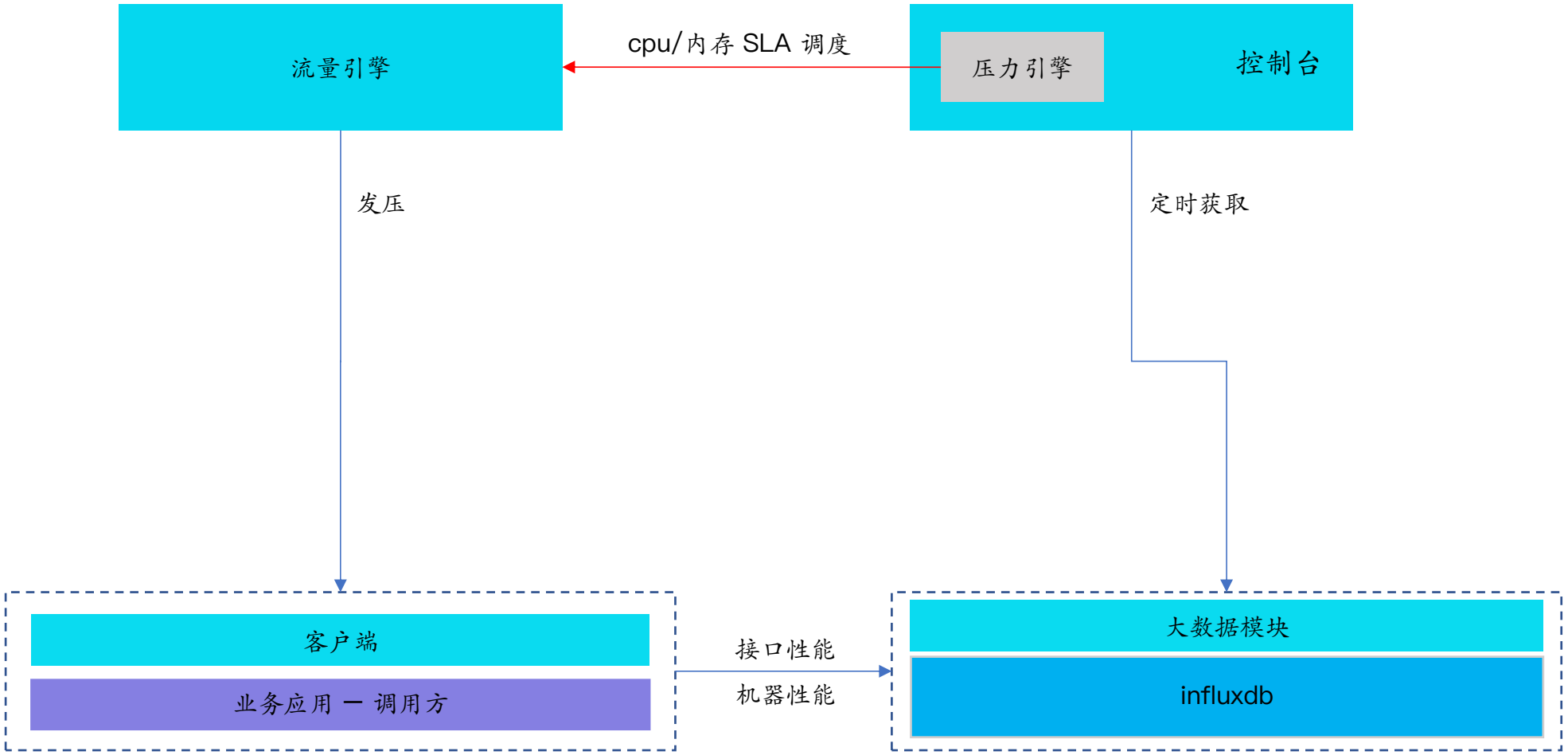
① 支持 http
② 支持 dubbo



Takin - SLA

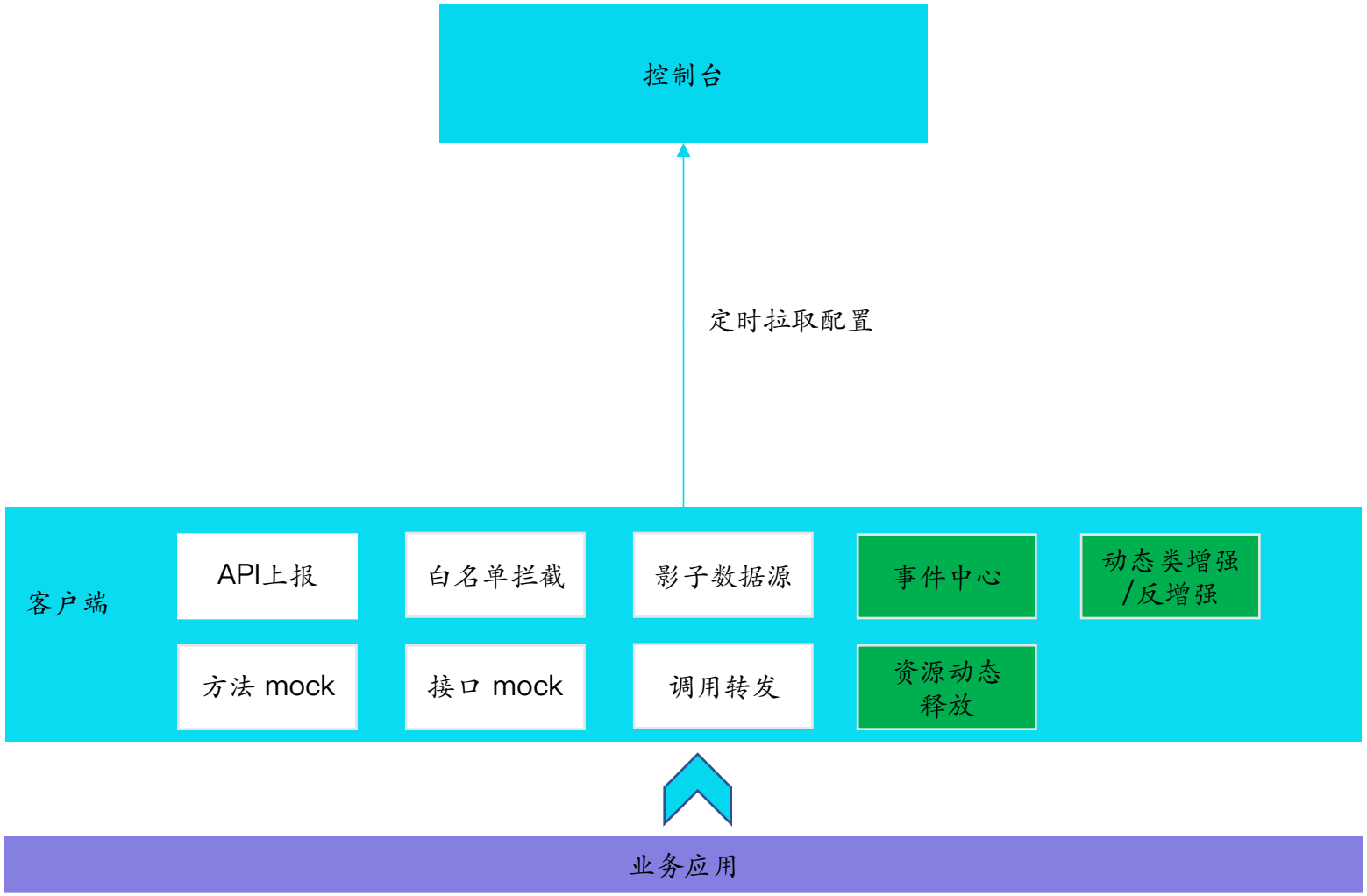
SLA 指标

① RT
② TPS
③ 成功率
④ SA
⑤ CPU 使用率
⑥ 内存使用率



① 每次修改配置需要重启应用才能生效，怎么能让配置实时生效
② 怎么能动态生效和禁用 agent 而不需要应用重启

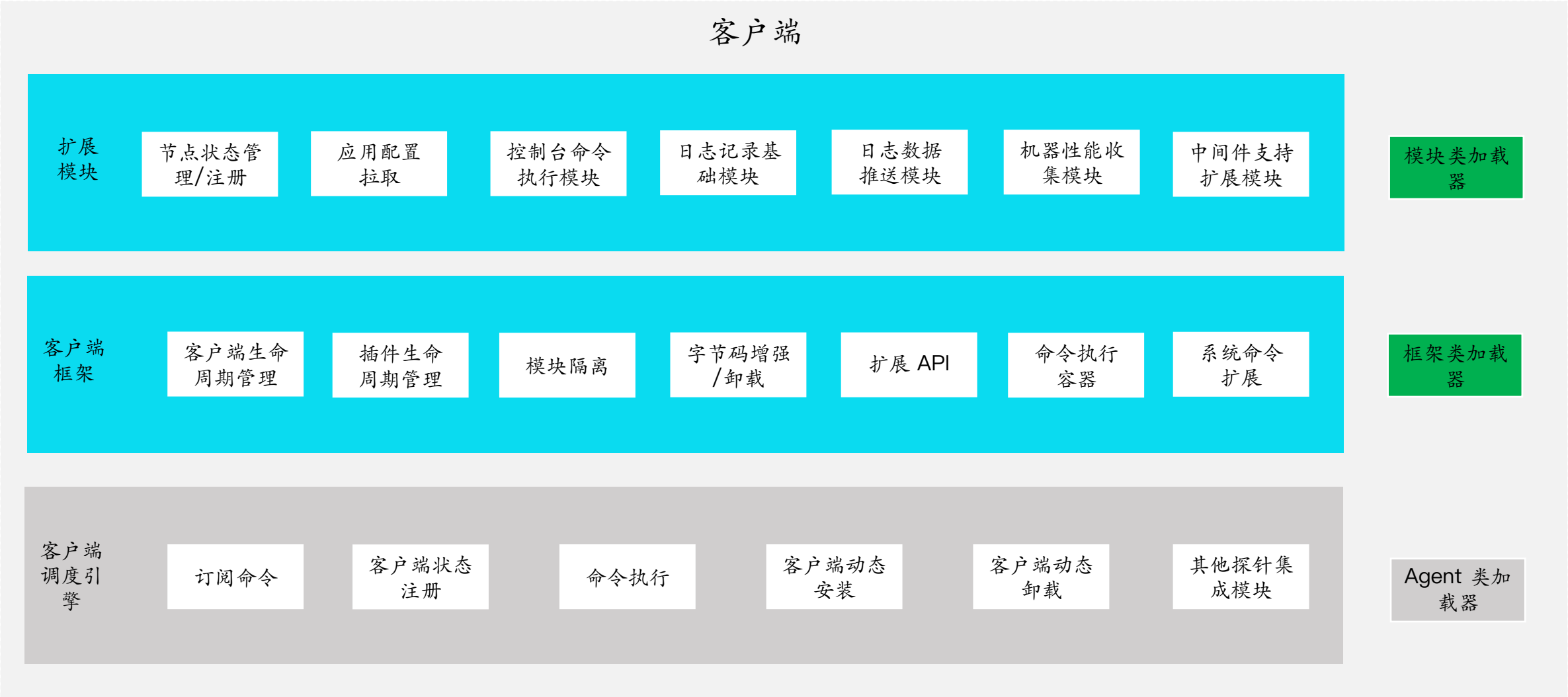
Takin - 配置动态生效



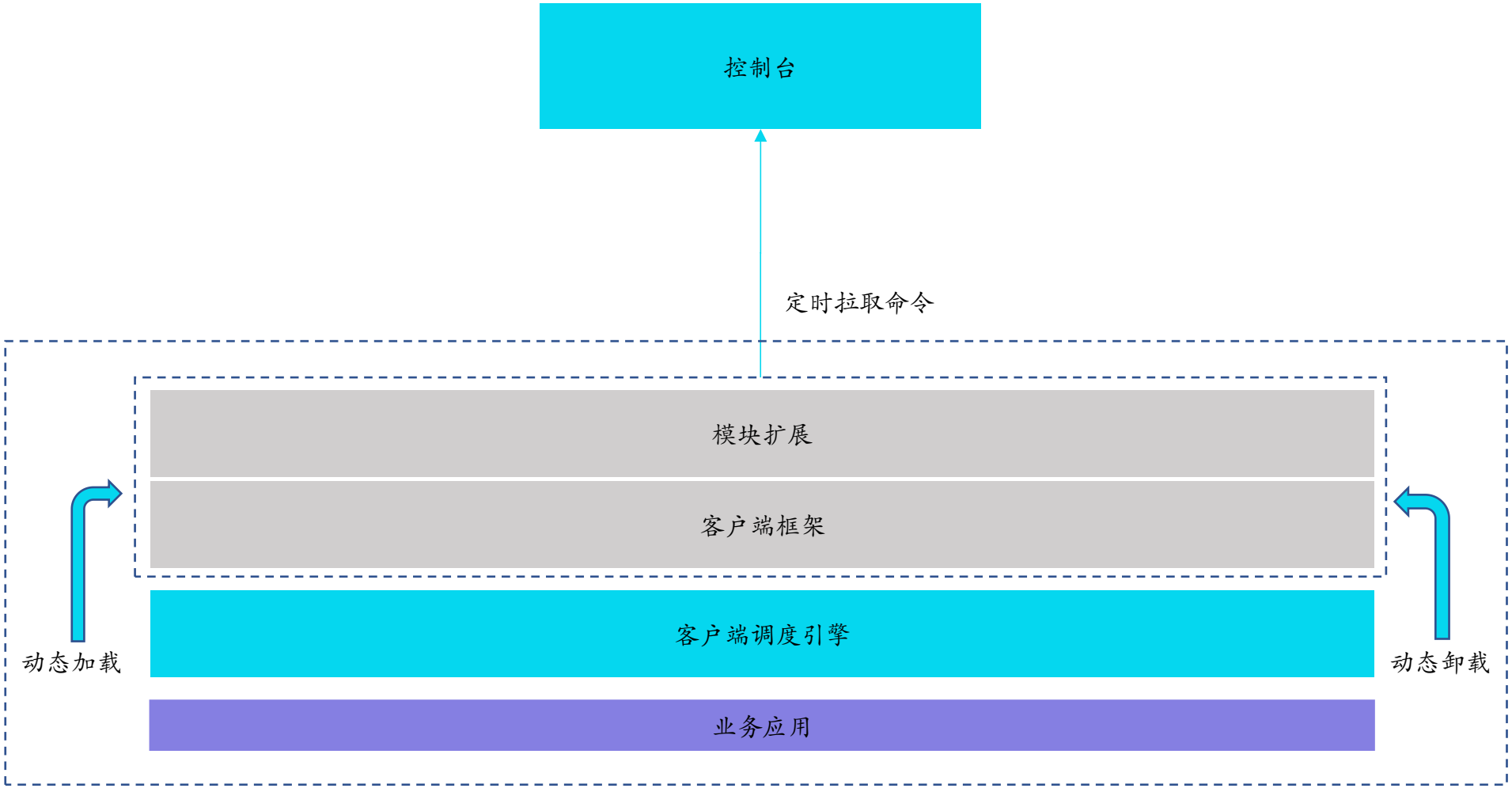
产品特性演进 – 客户端面临的问题

① 客户端不支持动态加载/卸载，安装、升级非常麻烦
② 框架与扩展模块之前的类隔离不彻底，经常会出现扩展模块与框架的类冲突
③ 扩展模块全部由业务应用类加载器加载，导致经常出现扩展模块与应用之间的类冲突
④ 经常出现因为获取配置延迟导致的加载延迟，这样会阻塞应用正常的启动
⑤ 模块不支持扩展式命令，不能方便的扩展一些辅助特性

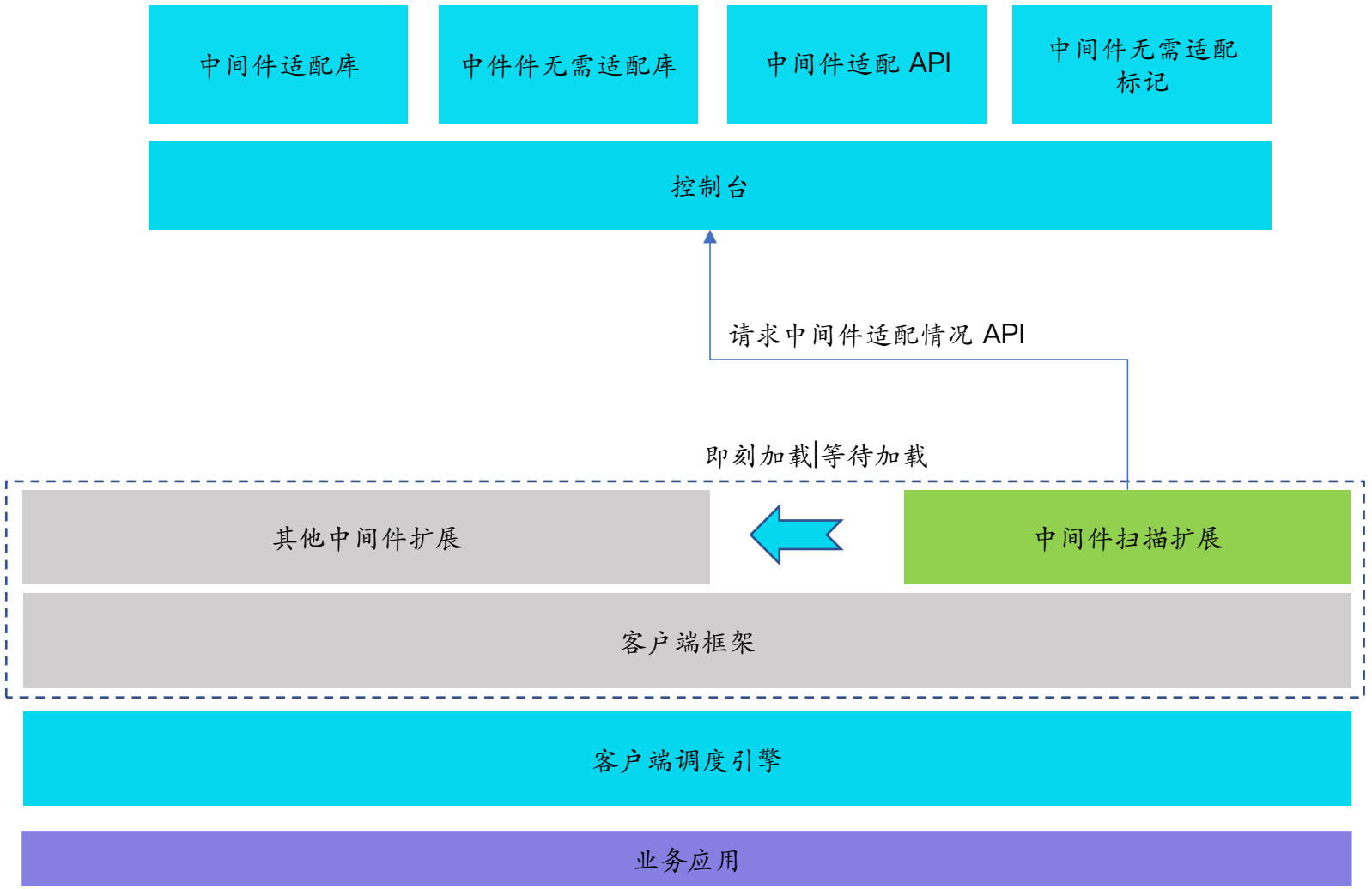
产品架构演进 - 客户端


客户端
扩展模块
节点状态管理/注册
应用配置拉取
控制台命令执行模块
日志记录基础模块
日志数据推送模块
机器性能收集模块
中间件支持扩展模块
模块类加载器
客户端框架
客户端生命周期管理
插件生命周期管理
模块隔离
字节码增强/卸载
扩展 API
命令执行容器
系统命令扩展
框架类加载器
客户端调度引擎
订阅命令
客户端状态注册
命令执行
客户端动态安装
客户端动态卸载
其他探针集成模块
Agent 类加载器

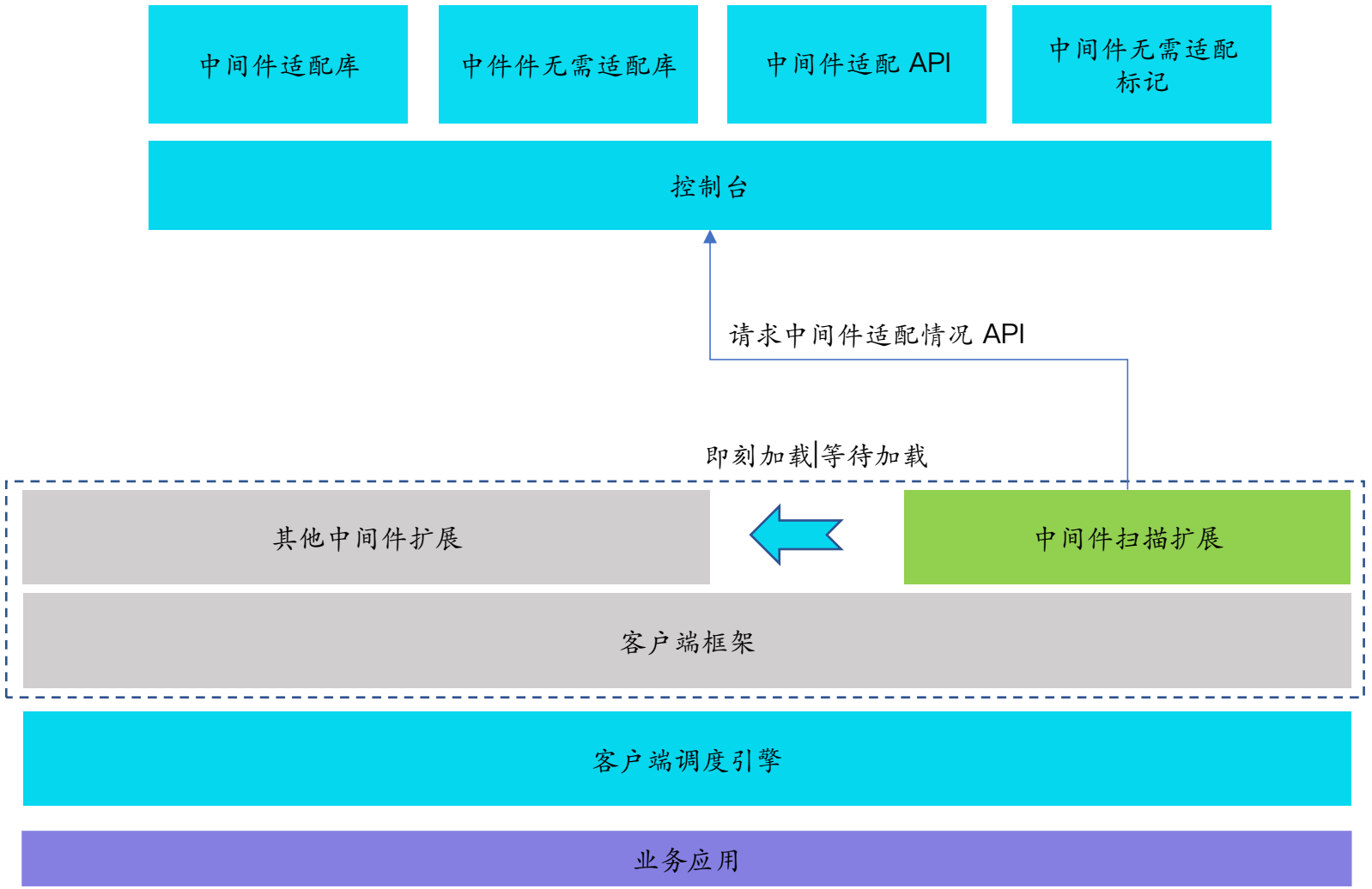
产品特性演进 – 动态安装、卸载



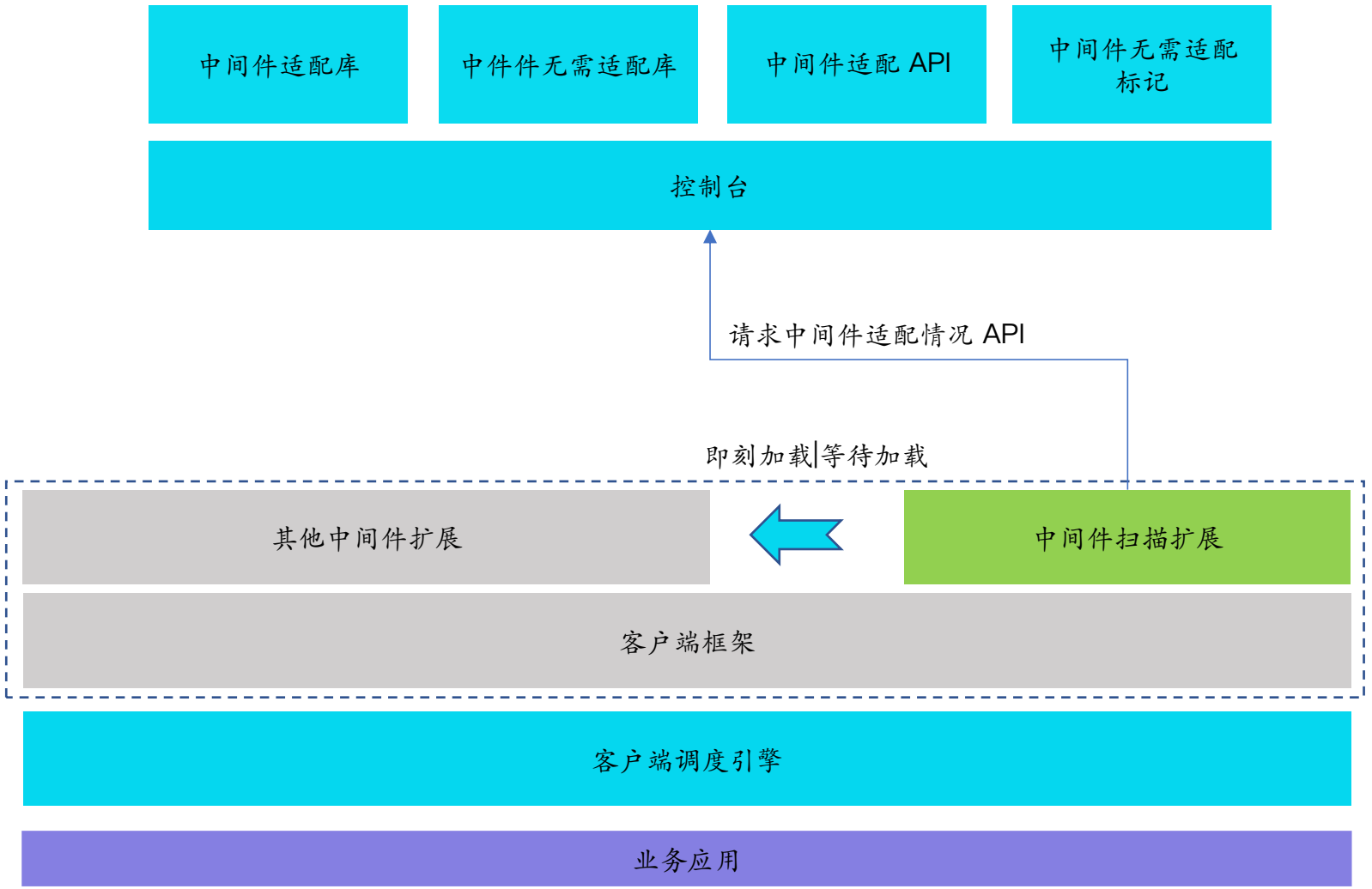
产品特性演进 – 中间件适配库



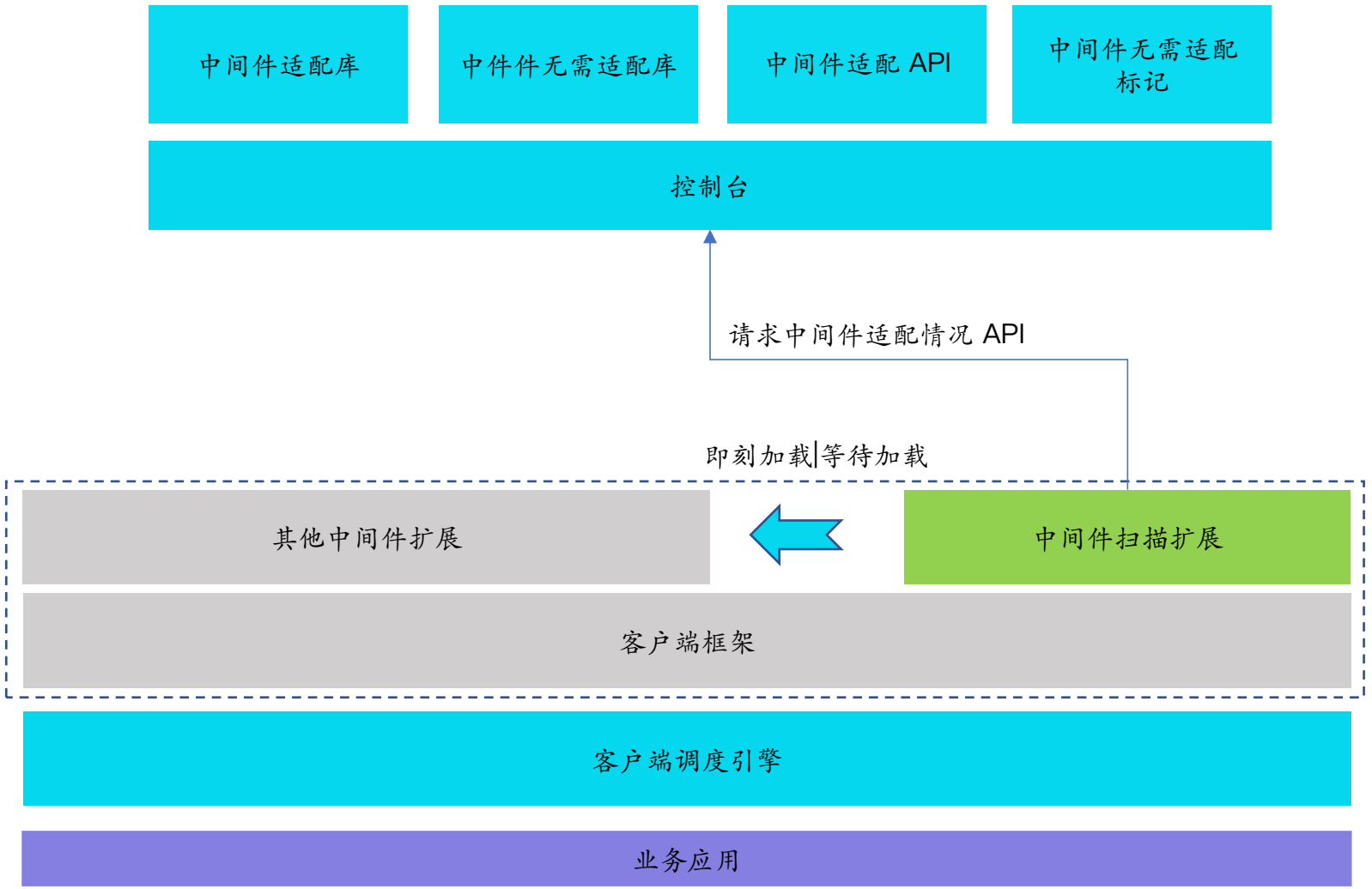
产品特性演进 – 中间件适配库



产品特性演进 – 中间件适配库



产品特性演进 – 中间件适配库



基于Takin做全链路压测的整体流程

确定需要压测的场景、系统

扫描依赖的中间件
Java应用重启接入探针

HTTP 接口标注
微服务依赖梳理

影子存储建立
铺数
白名单配置

链路调试通过

测试环境联调通过

生产环境联调通过

生产压测

系统优化

预案整理、演习验证

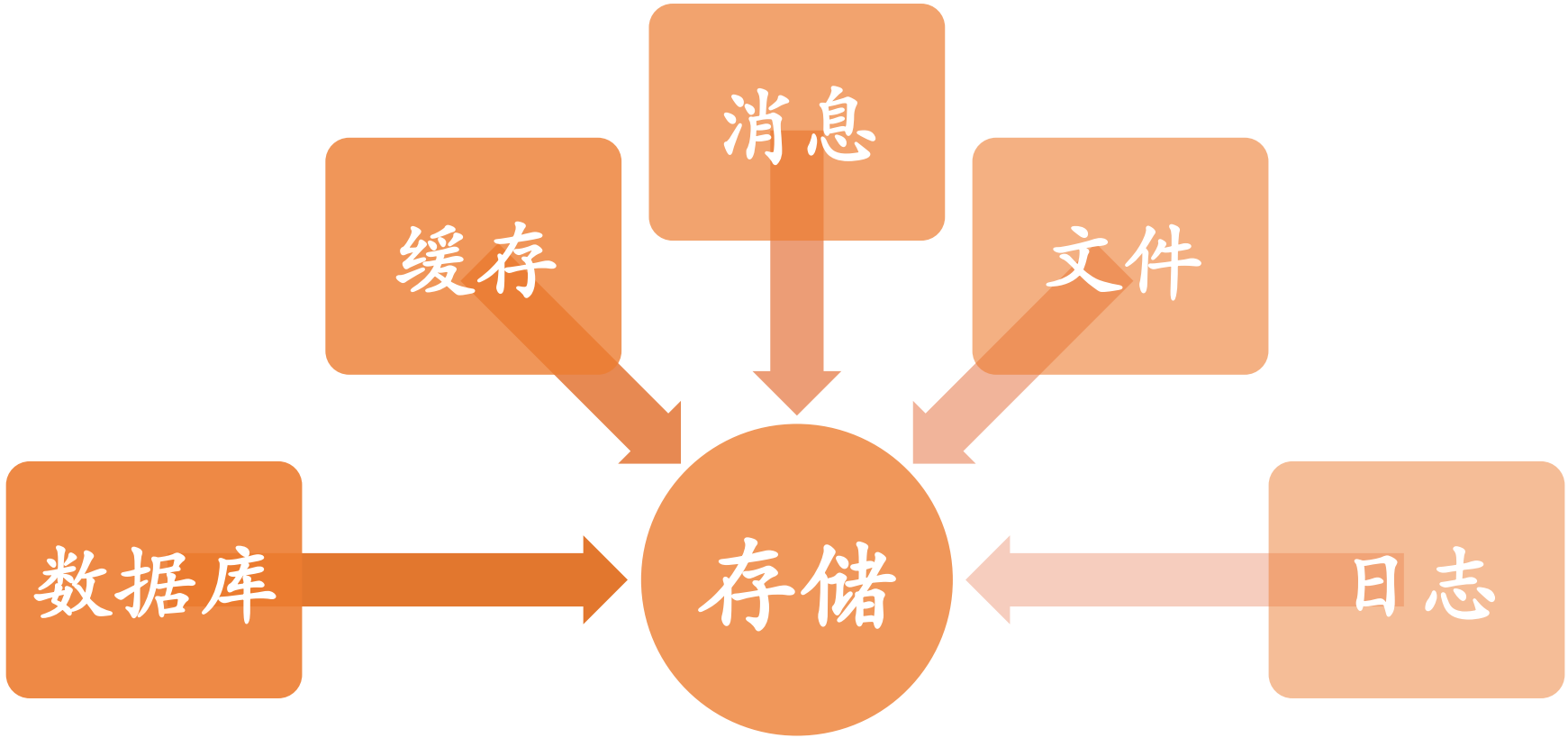
数据风险会有哪些？

数据隔离

隔离安全

数据隔离

我们的数据都会如何存储和使用?



数据隔离 — 数据库

影子库

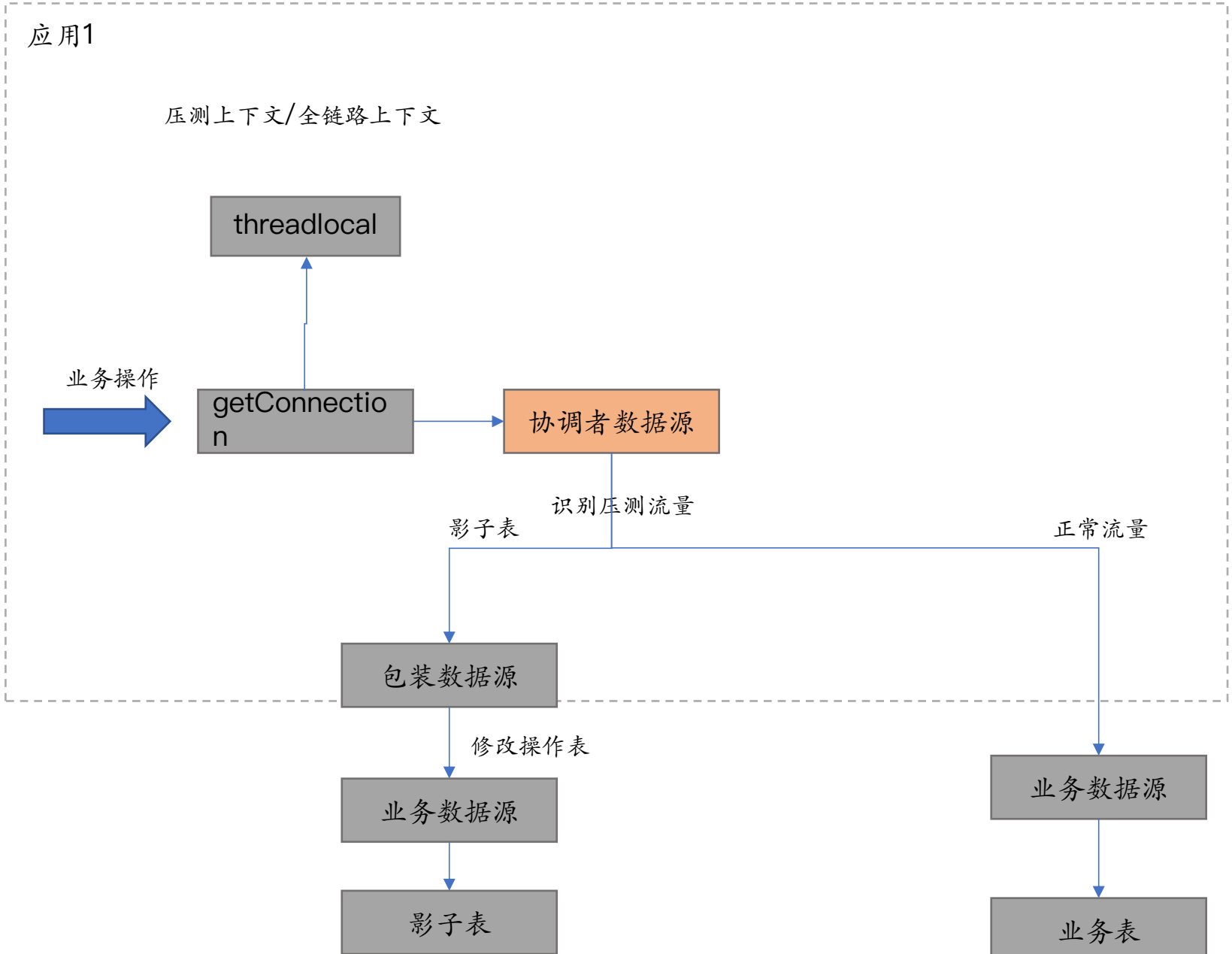
- 数据准备方便，使用方式灵活

影子表

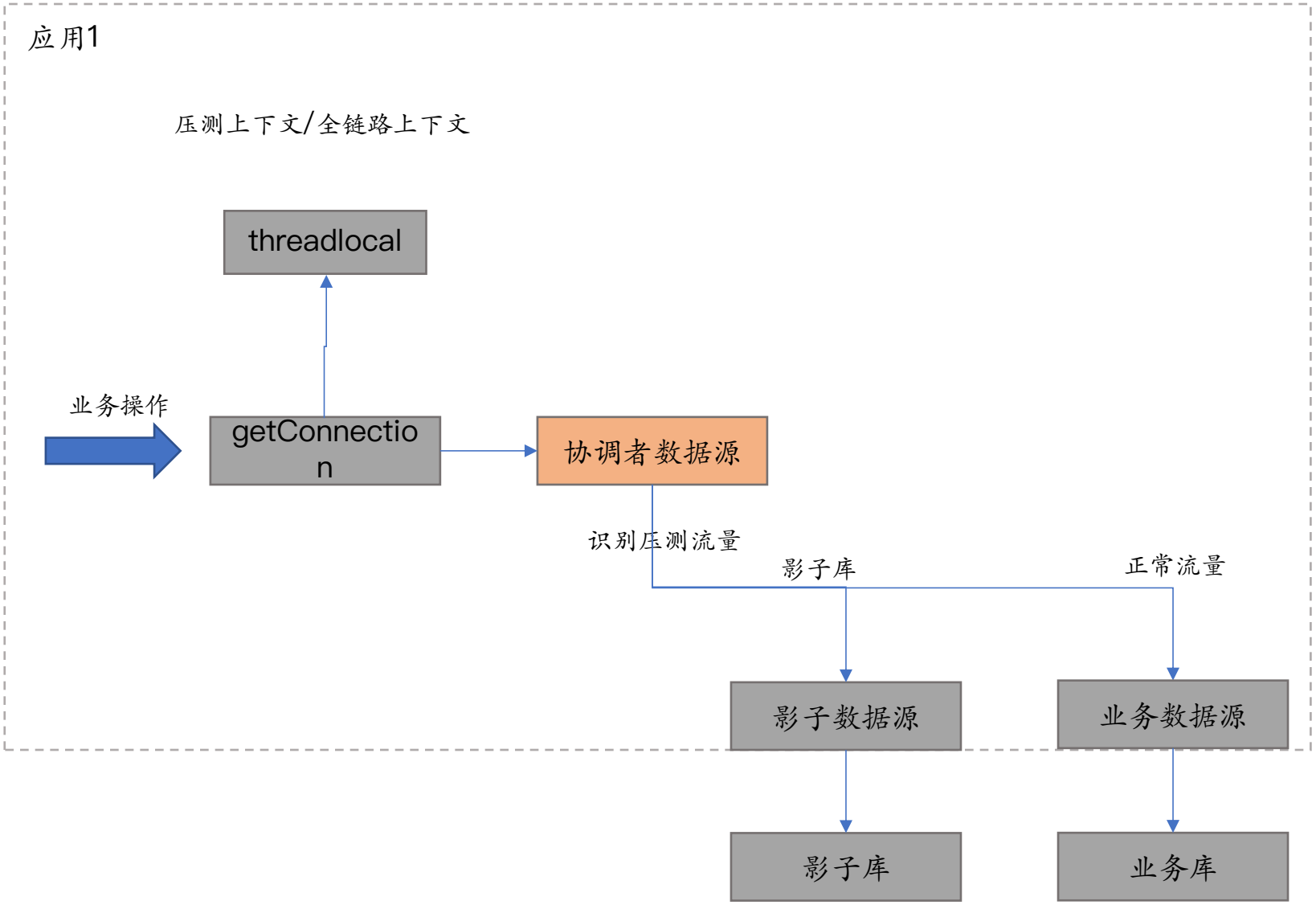
- 数据铺底范围小，占用空间少

数据隔离 – 影子表

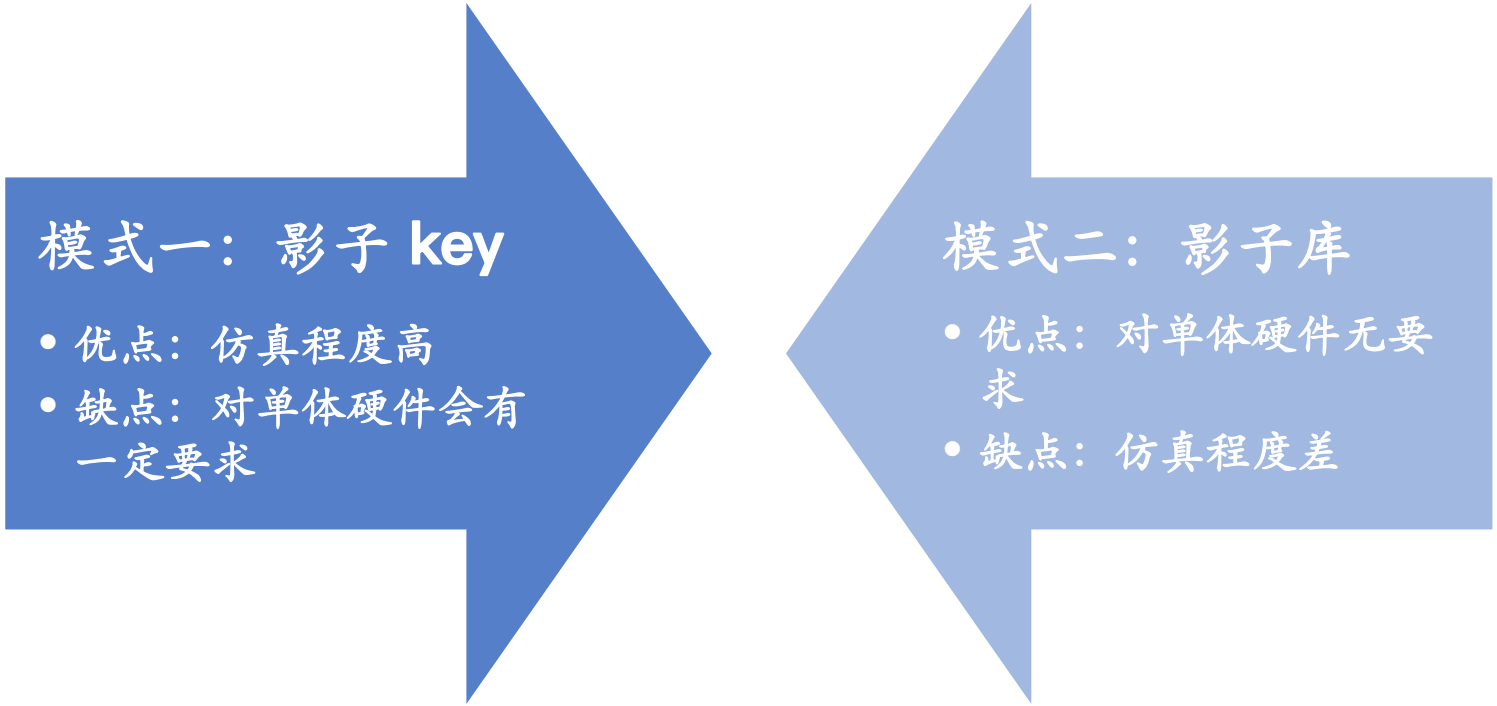
影子表方案



影子库方案


应用1
压测上下文/全链路上下文
threadlocal
业务操作
getConnection
协调者数据源
识别压测流量
影子库
正常流量
影子数据源
业务数据源
影子库
业务库

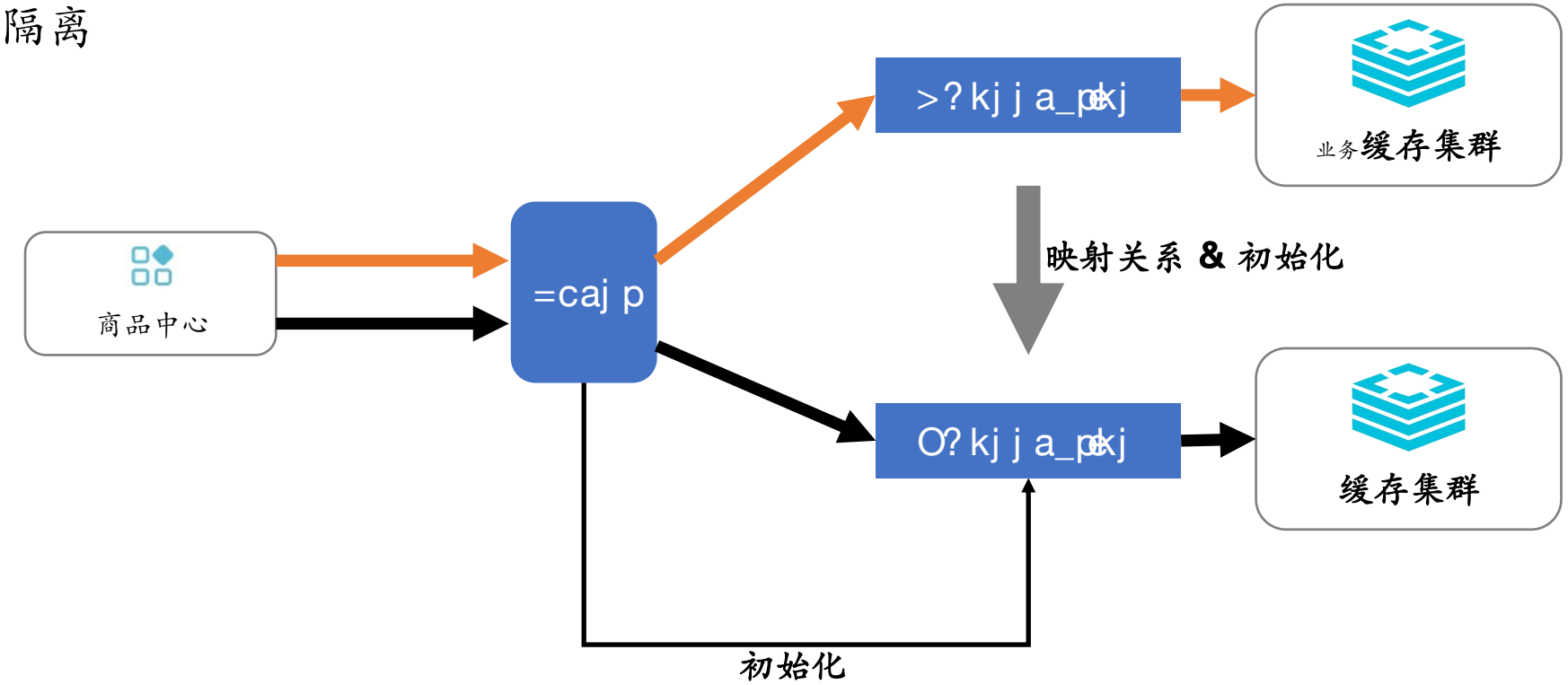
数据隔离 — 缓存


模式一：影子 key
• 优点：仿真程度高
• 缺点：对单体硬件会有一定要求
模式二：影子库
• 优点：对单体硬件无要求
• 缺点：仿真程度差

缓存– 影子库模式

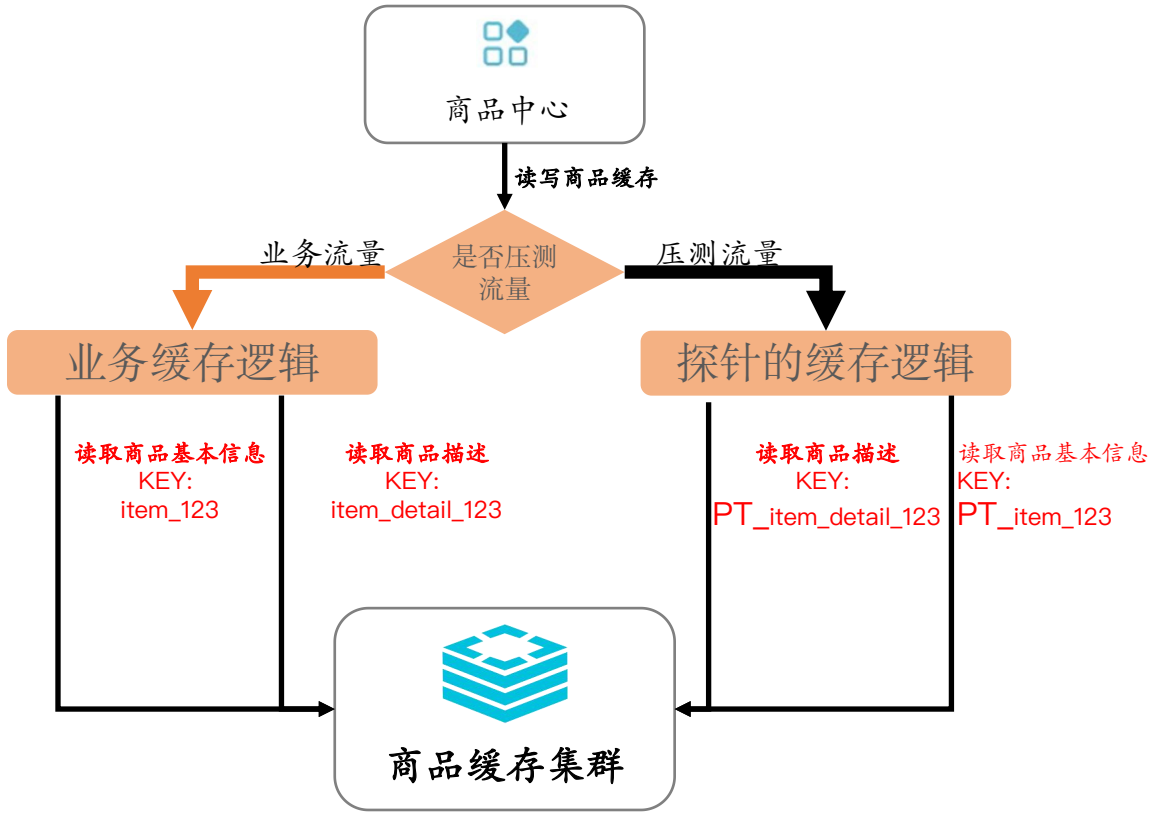
◆独立集群

◆物理层面隔离

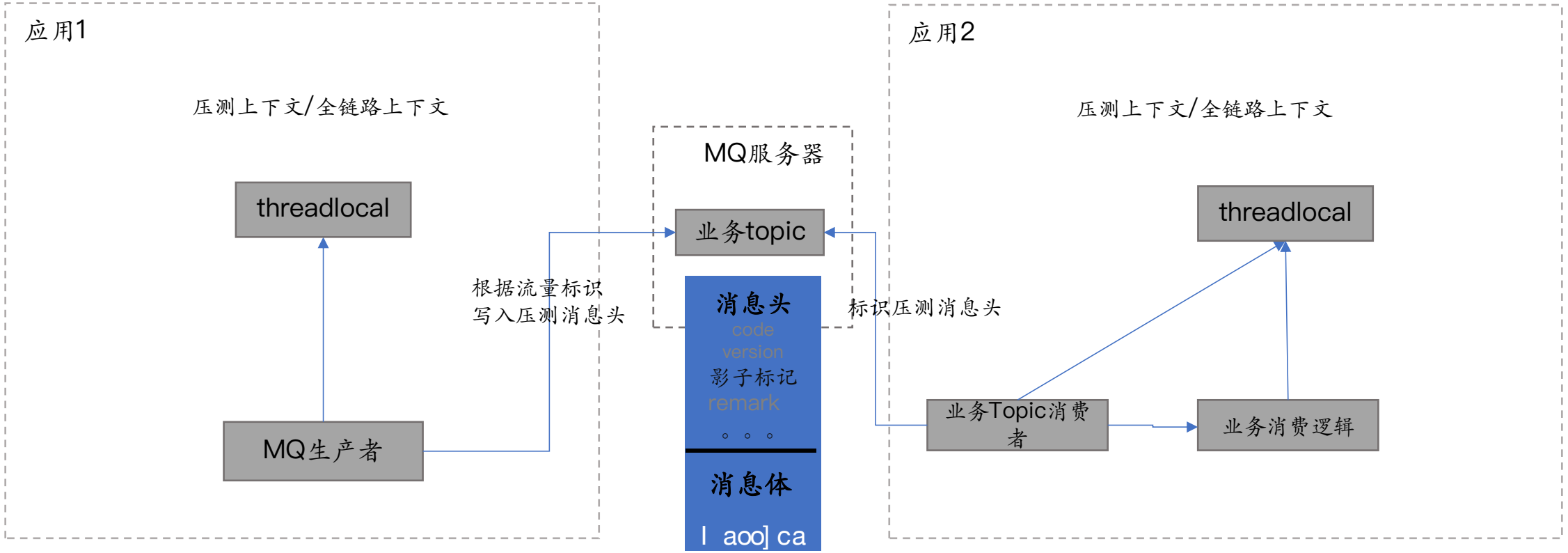


缓存 — 影子key

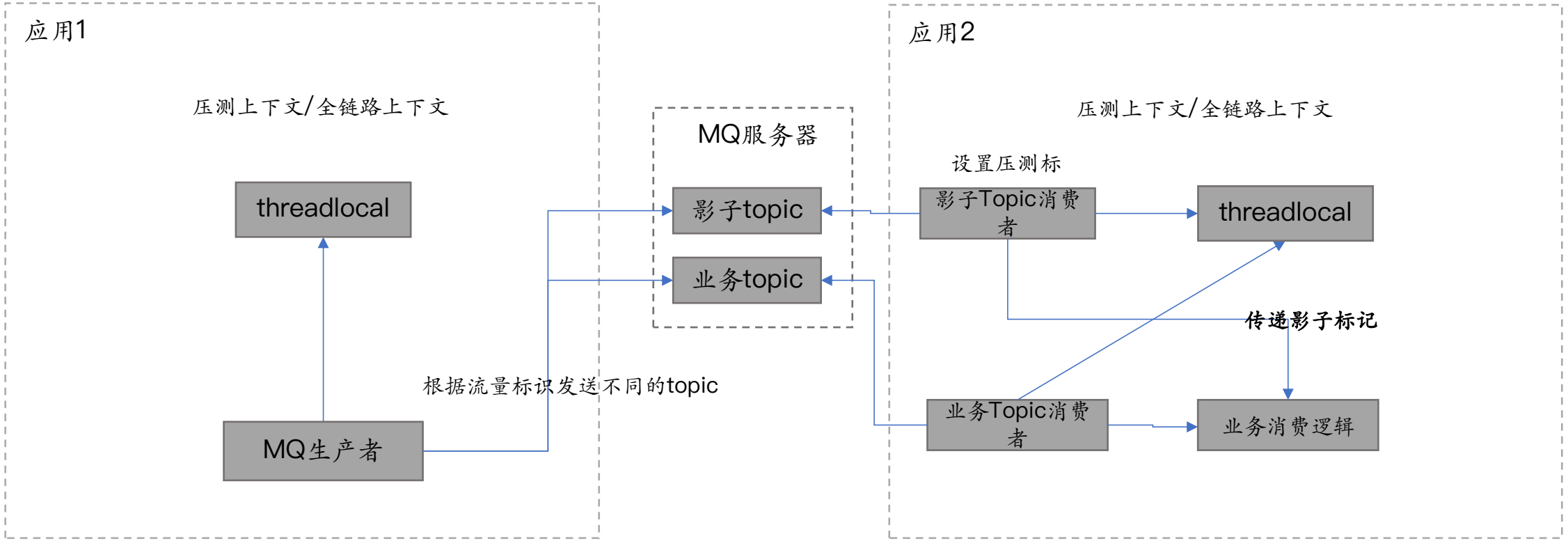
- 同一个实例
- 在缓存 key 上做区分



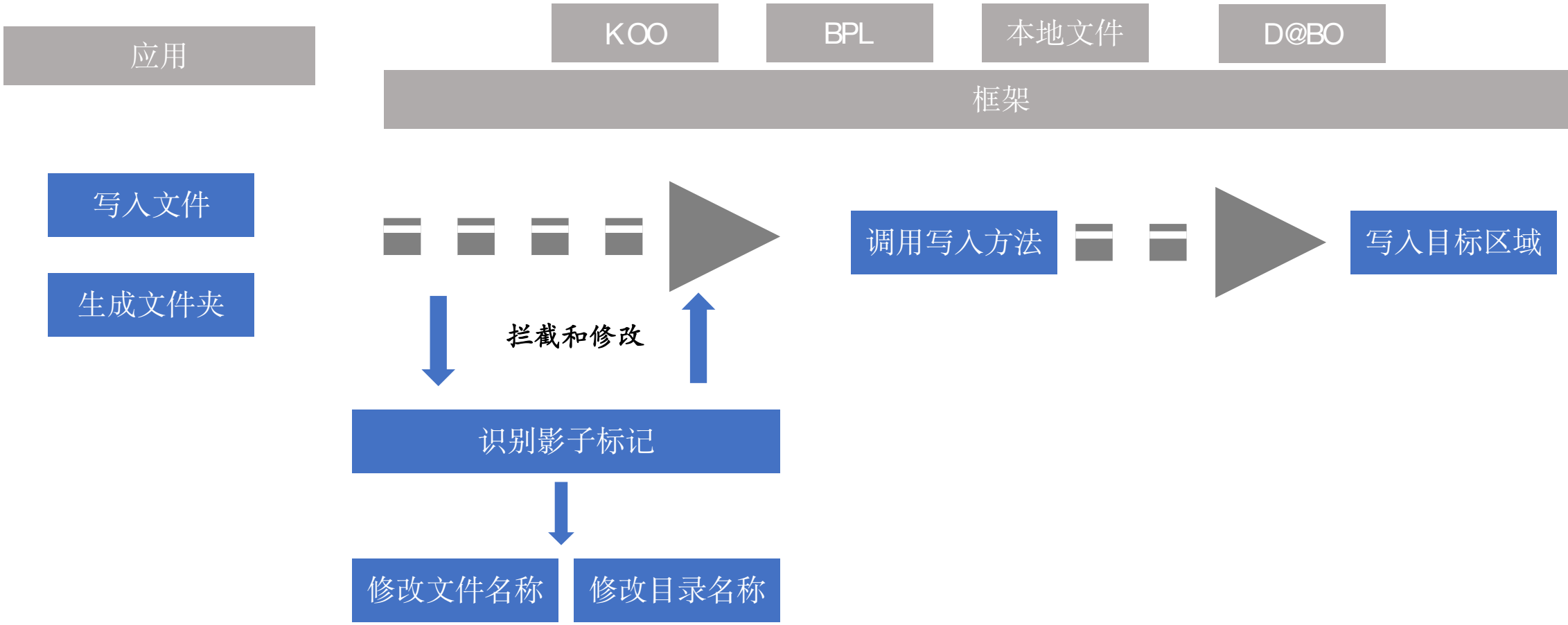
消息隔离方案——消息属性隔离



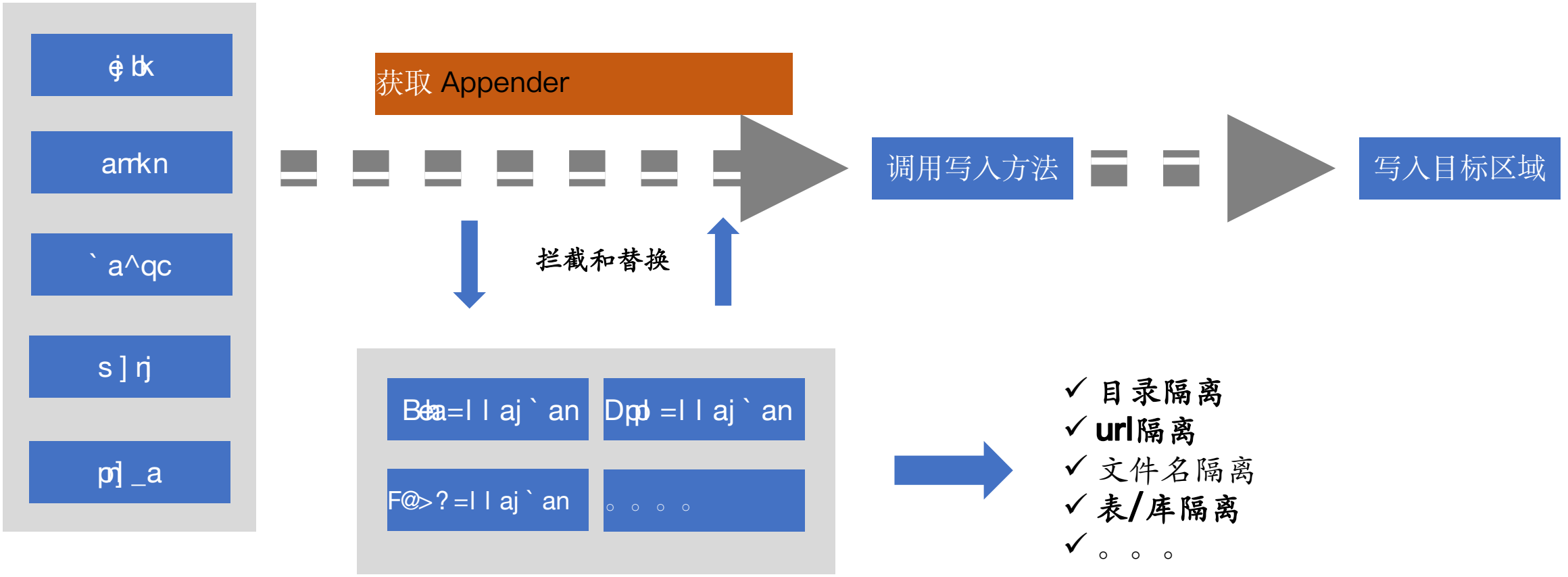
消息隔离方案——Topic隔离



数据隔离 — 文件


应用
KOO
BPL
本地文件
D@BO
框架
写入文件
生成文件夹
调用写入方法
写入目标区域
拦截和修改
识别影子标记
修改文件名称
修改目录名称

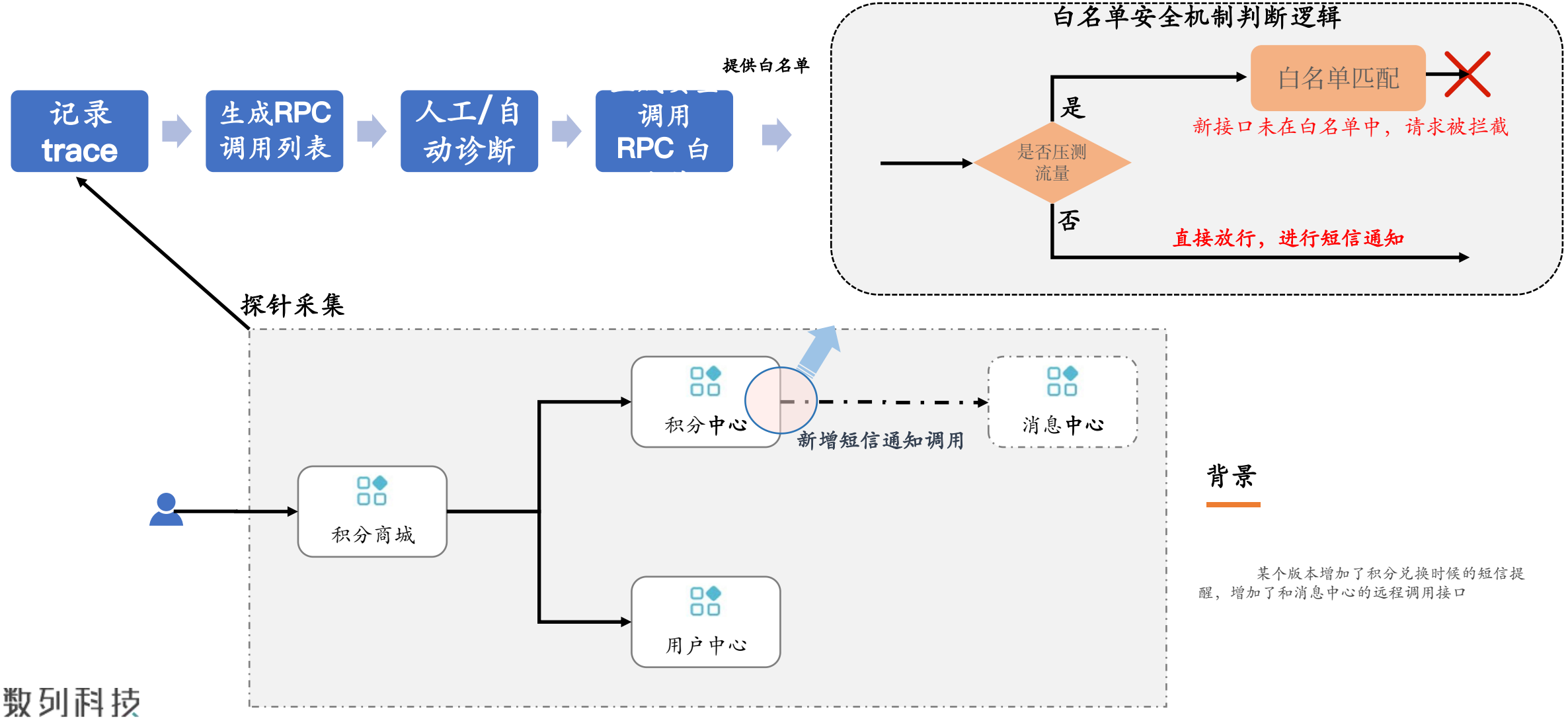
数据隔离 — 日志



隔离安全

上述的数据层面隔离都可以做到100%了，是否数据安全就没问题了？

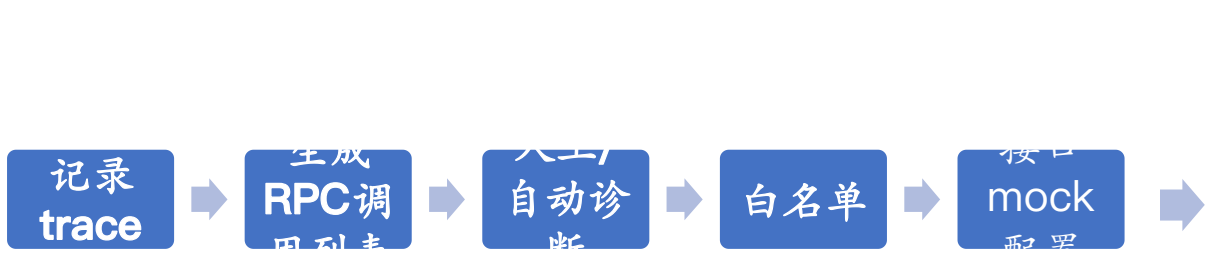
隔离安全 — 新增了调用关系

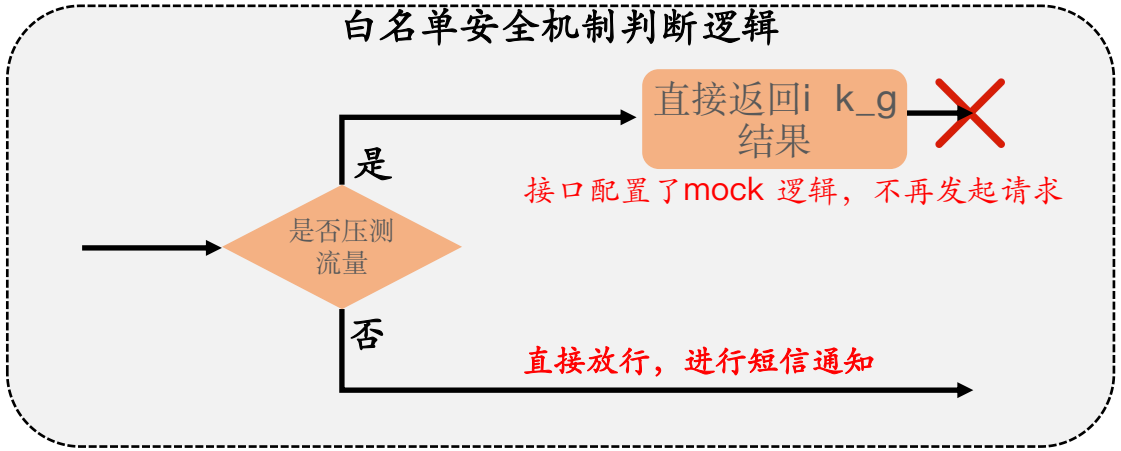


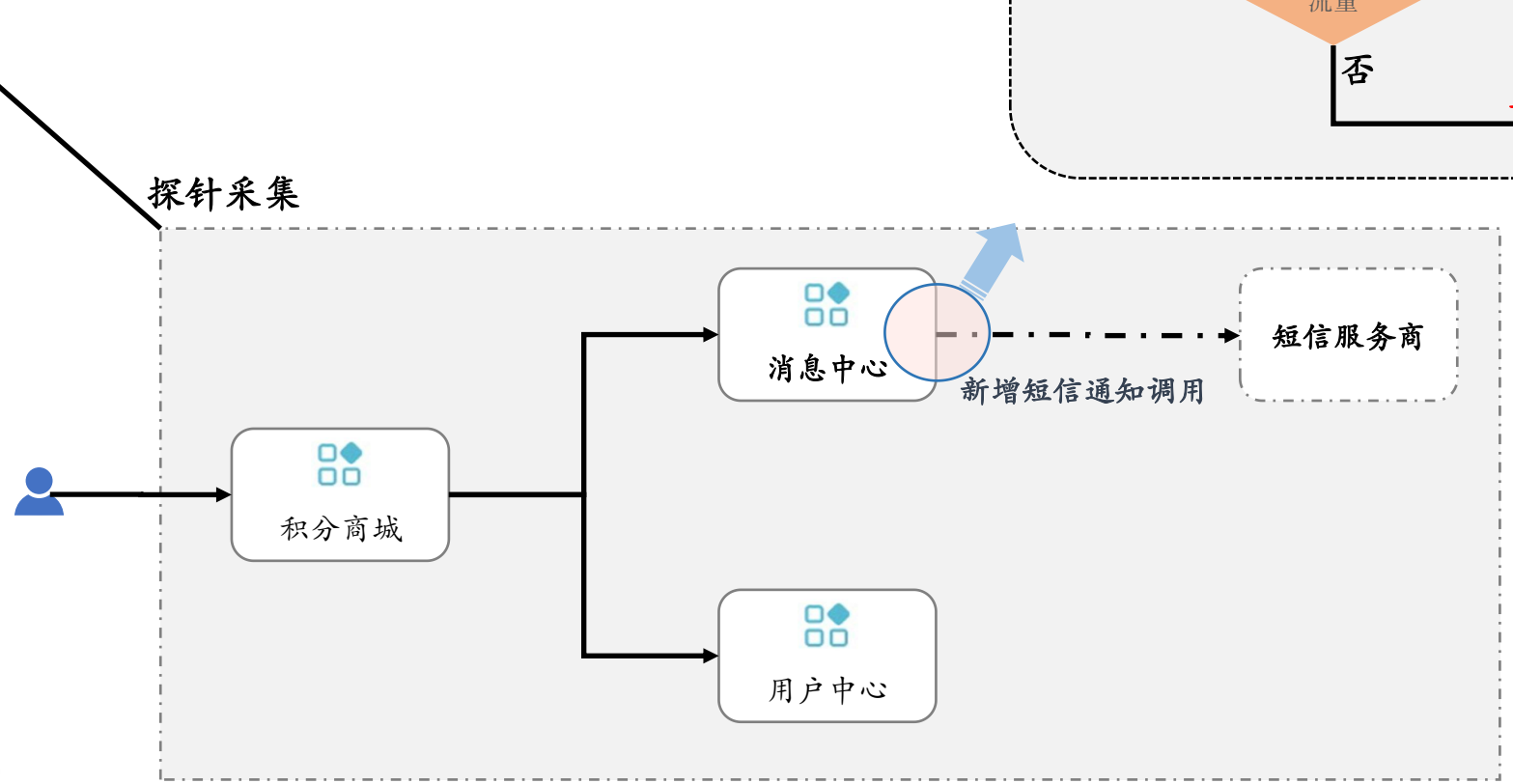
背景

某个版本增加了积分兑换时候的短信提醒，增加了和消息中心的远程调用接口

隔离安全 — 三方调用



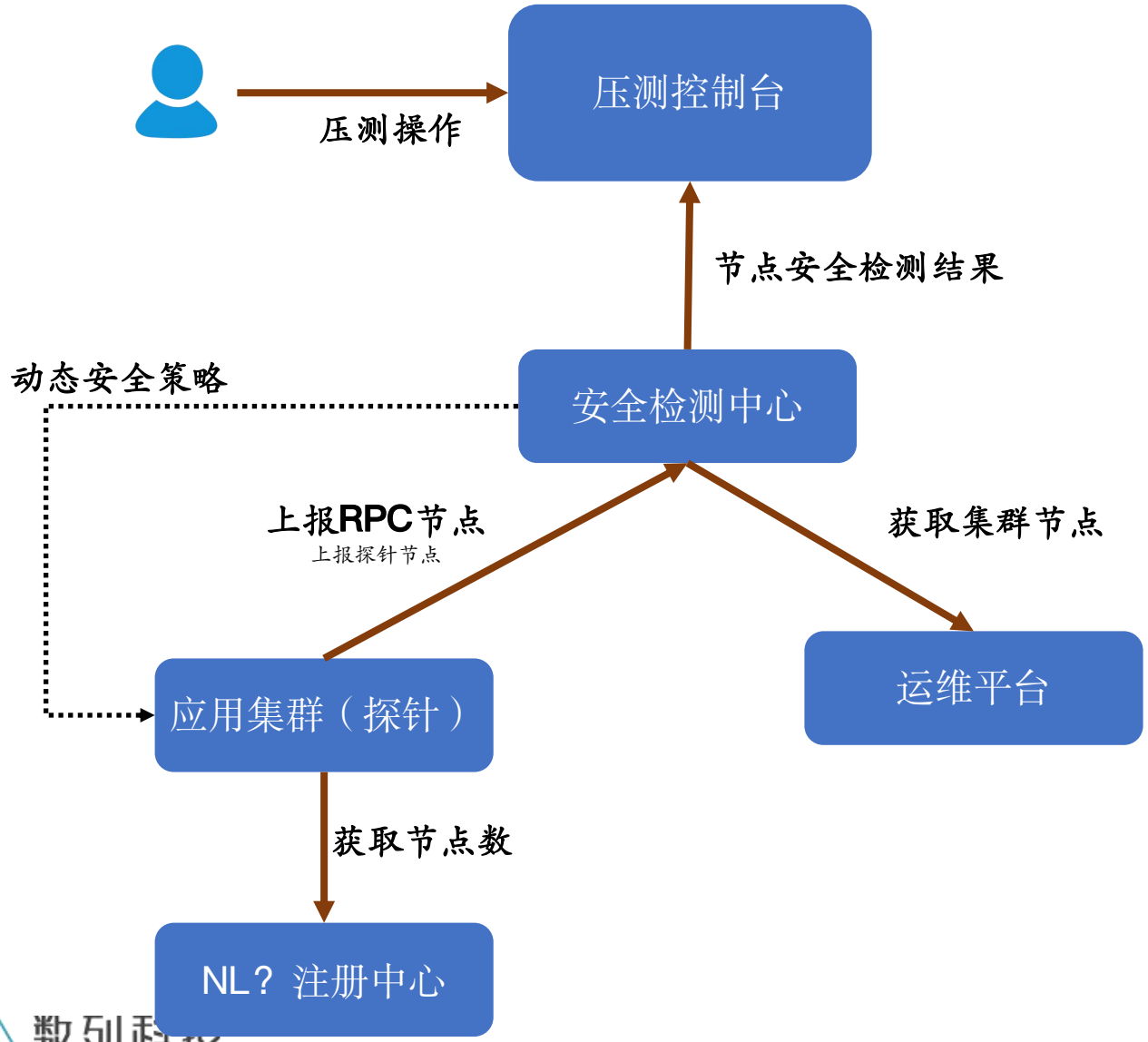


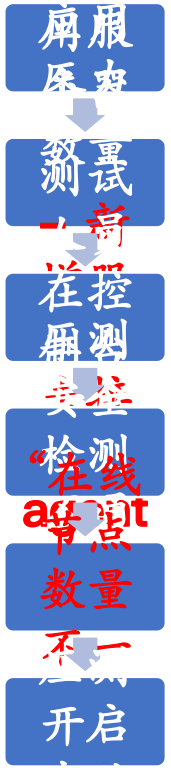


背景

某个版本增加了积分兑换时候的短信提醒

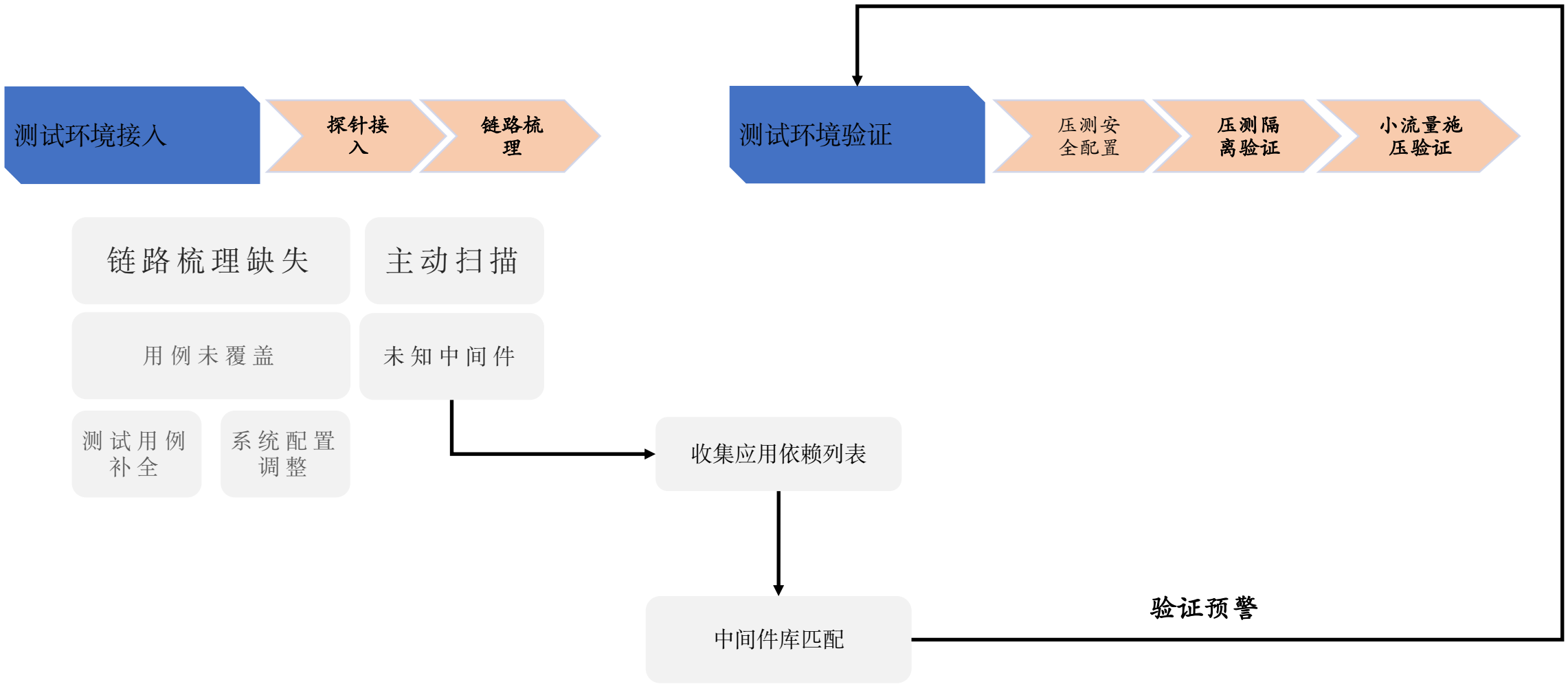
隔离安全 — 应用集群节点变化



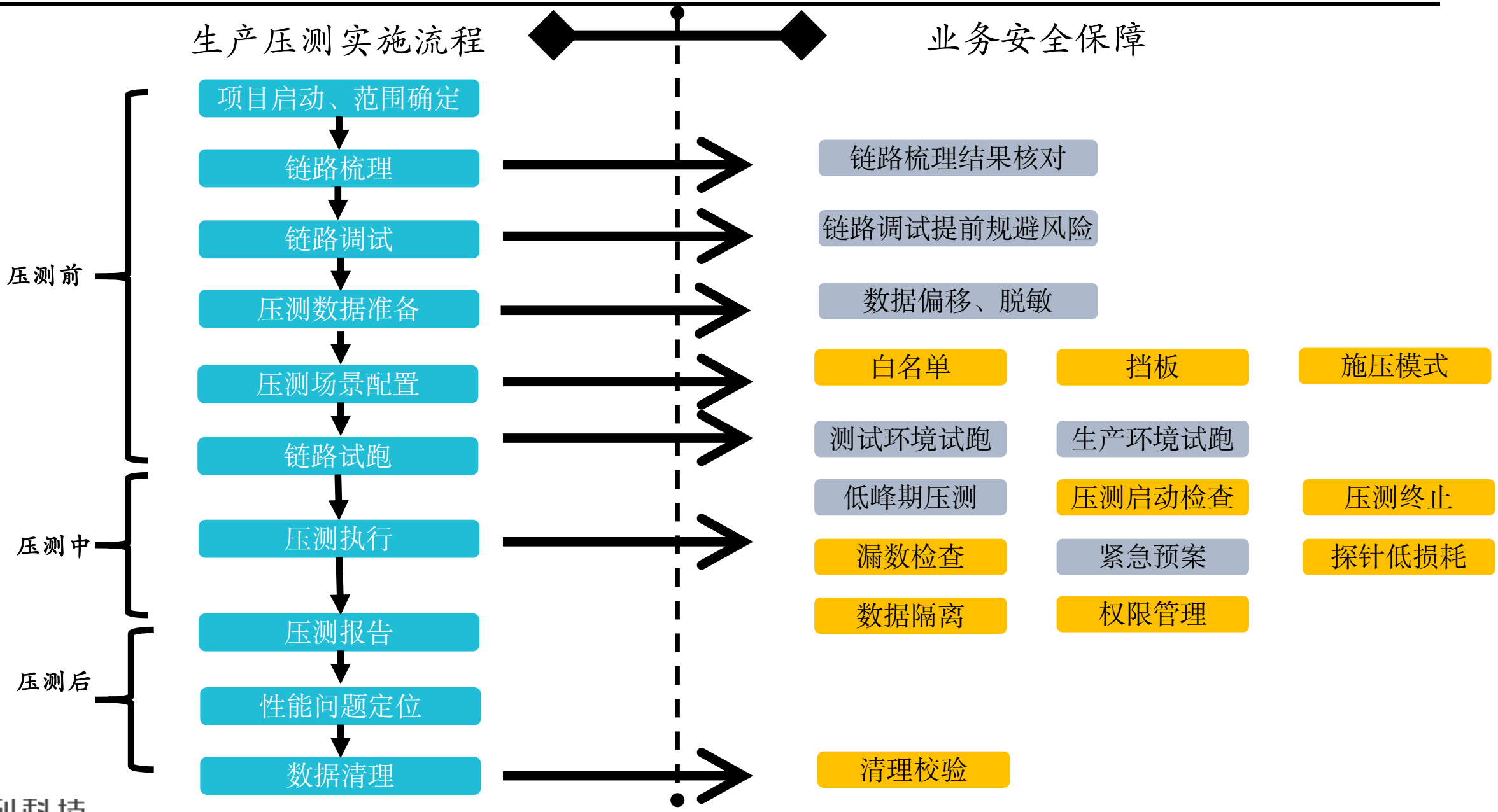


避免在有数据风险的情况下发起压测

隔离安全 — 引入了新的中间件



全链路压测安全保障



关于Takin FAQ

开源社区版和企业版的区别

开源社区版是从企业版的代码剥离出来的。目前也在不断的从企业版抽代码到社区版

探针的性能损耗

3%–5%；如一个应用峰值QPS在1000，则挂载探针后，当应用QPS到达1000，则有3%–5%的损耗。低峰期只会看到80M以内的内存增加。

可以在具体场景进行压测验证。（挂载探针与不挂载）

和SkyWalking是否有冲突

具体到场景来进行Takin agent的适配。此前有多个企业有Skywalking，均能解决。

开源社区地址

https://github.com/shulieTech/Takin，核心代码做了开源社区版