

基于APM的智能运维体系在京东物流的落地和实践

付正全

京东物流 架构师

极客邦科技 会议推荐2019

5月

QCon 北京

全球软件开发大会

大会: 5月6-8日
培训: 5月9-10日

QCon 广州

全球软件开发大会

培训: 5月25-26日
大会: 5月27-28日

6月

GTLC
GLOBAL
TECH LEADERSHIP
CONFERENCE

上海

技术领导力峰会

时间: 6月14-15日

GMTC 北京

全球大前端技术大会

大会: 6月20-21日
培训: 6月22-23日

7月

ArchSummit 深圳

全球架构师峰会

大会: 7月12-13日
培训: 7月14-15日

10月

QCon 上海

全球软件开发大会

大会: 10月17-19日
培训: 10月20-21日

11月

GMTC 深圳

全球大前端技术大会

大会: 11月8-9日
培训: 11月10-11日

AiCon 北京

全球人工智能与机器学习大会

大会: 11月21-22日
培训: 11月23-24日

12月

ArchSummit 北京

全球架构师峰会

大会: 12月6-7日
培训: 12月8-9日

自我介绍



付正全，京东物流架构师，国家认证信息系统项目经理师，曾任浪潮集团系统架构师，专注监控平台研发工作 8 年，研究过市场上数十家厂商的监控平台产品，对 DevOps 和监控平台有比较深入的了解。目前负责京东物流火眼监控平台的架构设计和开发工作。

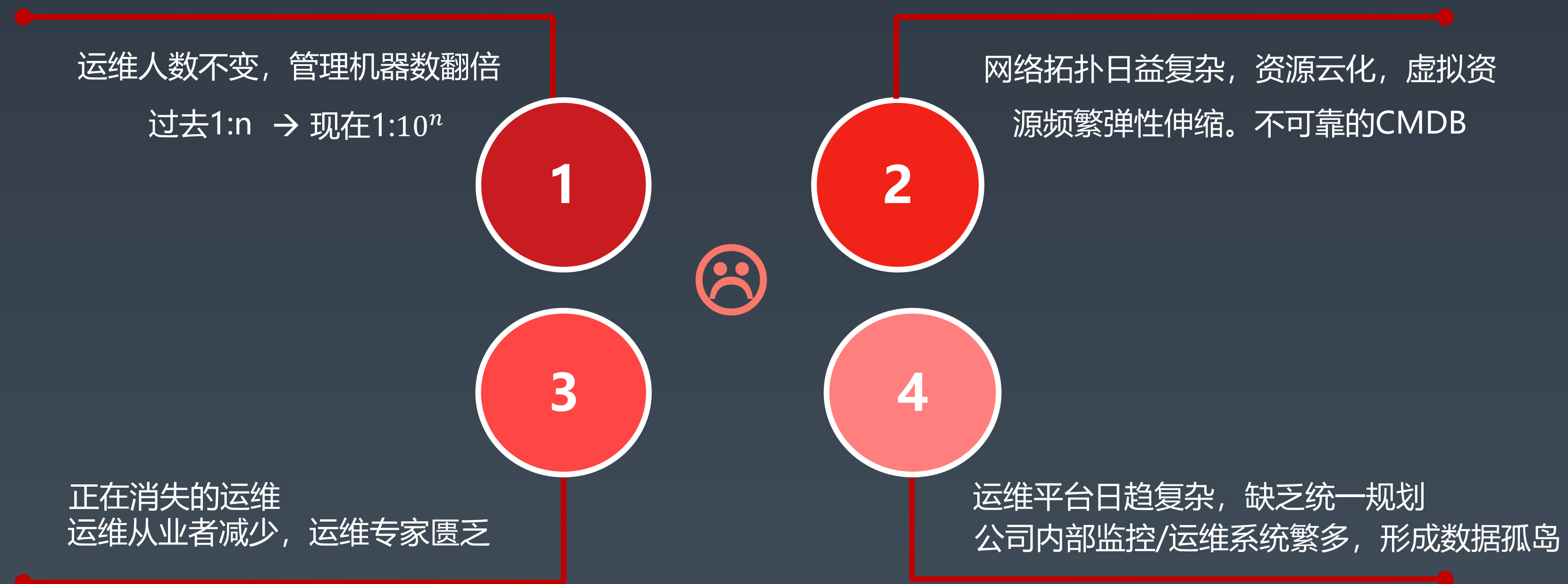
目录

- 业界智能运维发展现状及趋势
- 智能运维体系建设方法论
- 大规模实时监控平台的实践方案
- 智能故障定位与处理实践
- APM 在京东物流的落地实践
- 智能运维(AIOps)落地规划

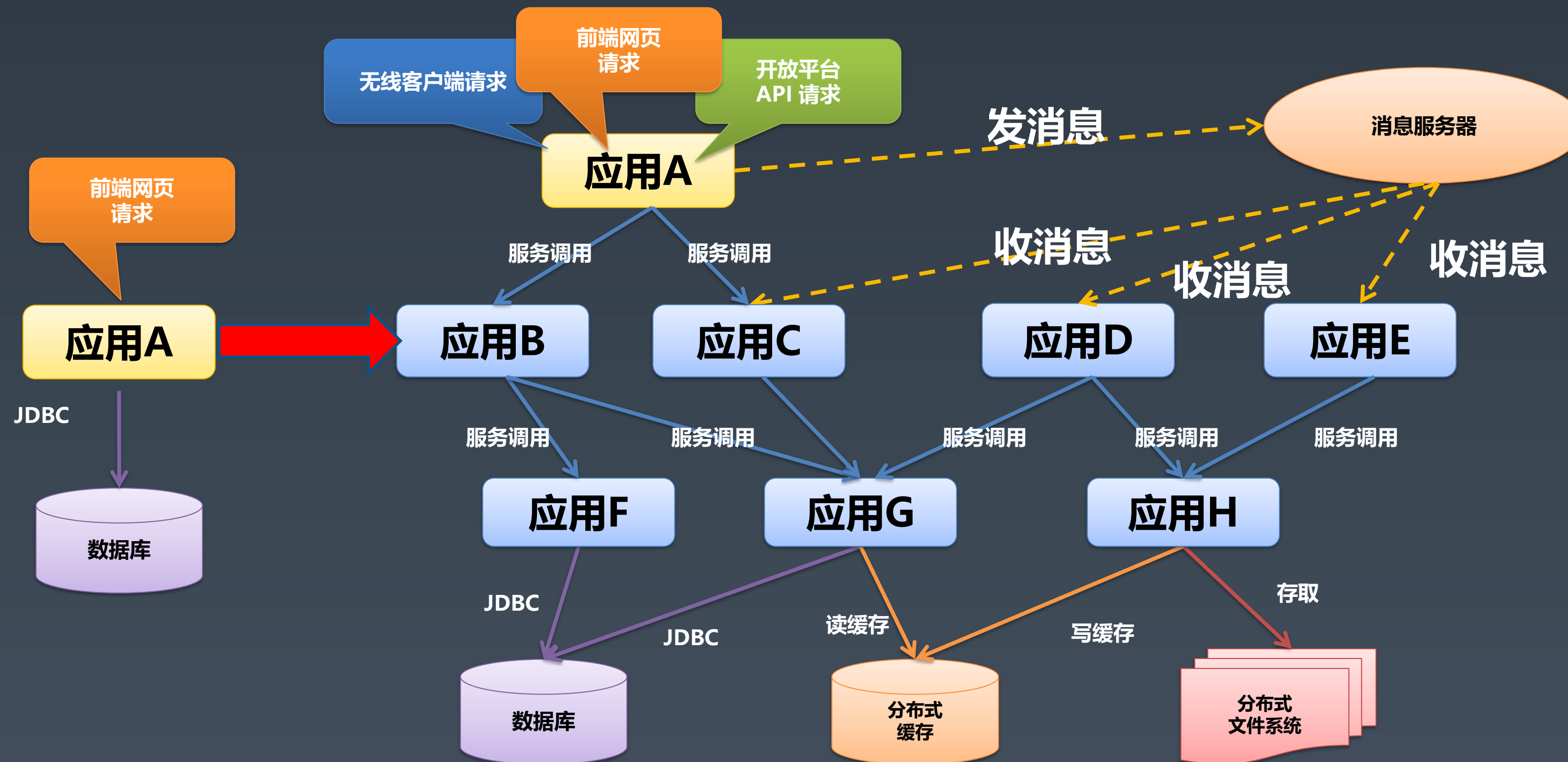
业界智能运维发展趋势



新的问题



越来越复杂的应用拓扑



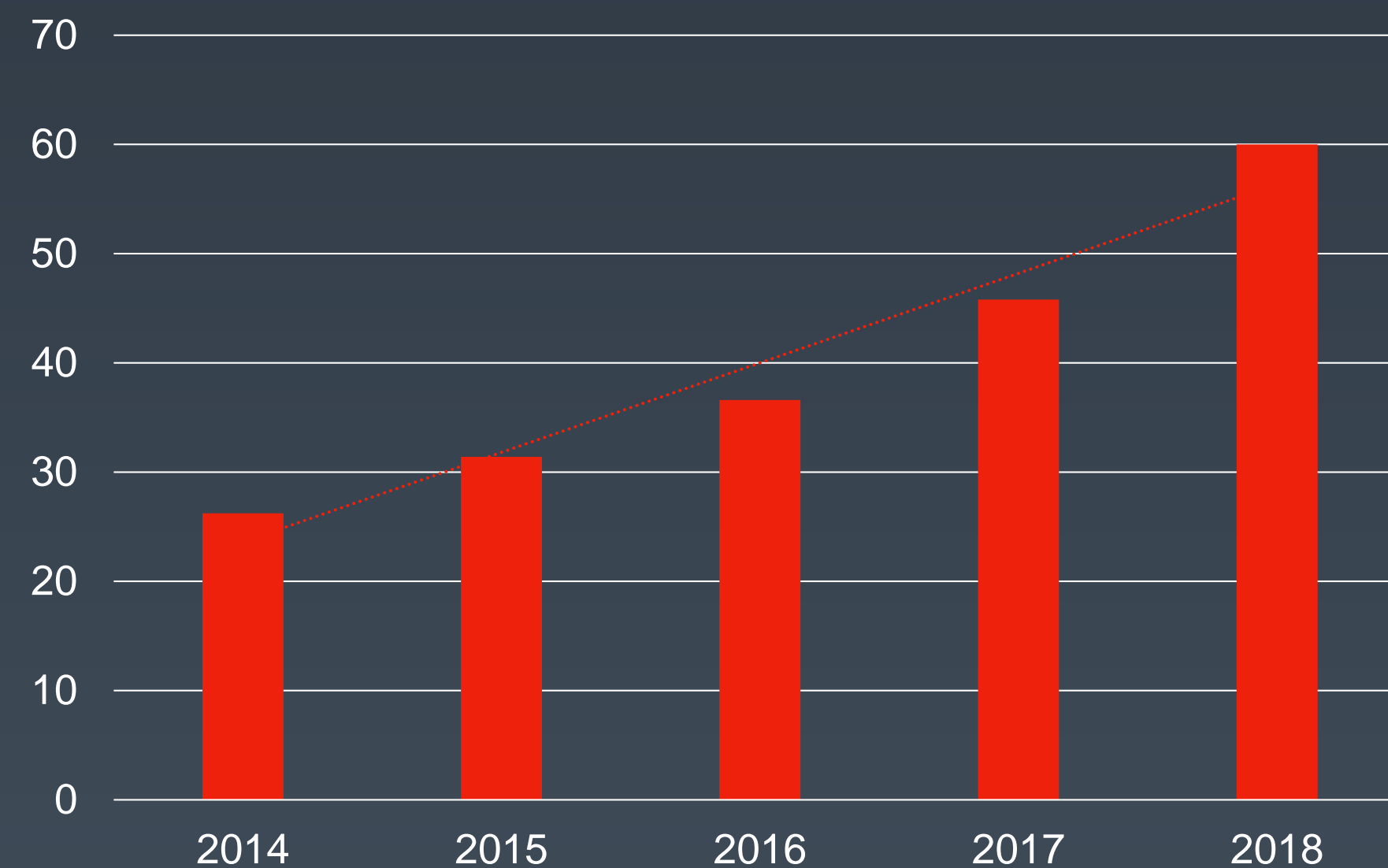
系统问题定位**难**

快速发展的APM

APM (应用性能管理)市场规模逐年递增

- 目前，全球APM市场规模大约在60亿美元左右，预计在五年内达到90亿美元
- APM成为ITOM成长最快的领域
- APM能够对企业的业务应用进行监测、诊断分析、优化，最终能够提高应用的可靠性和质量，保证良好的用户体验，降低IT成本

APM市场规模(亿美元)



运维角色转变



背锅侠

被动响应



主动求变



救火员

产品意识

需求提炼

产品化开发

产品化落地

技术运营

推广落地

业务数据分析

过程改进

业务增值

事件处理

业务分析

业务预测

架构运维

架构标准化

架构实施



架构优化

运维价值凸显 新运维时代来临

目录

- 业界智能运维发展现状及趋势分析
- 智能运维体系建设方法论
- 大规模实时监控平台的实践方案
- 智能故障定位与处理实践
- APM 在京东物流的落地实践
- 智能运维(AIOps)落地规划

智能运维体系建设方法论

相对成熟  加强支持 

■ 统一规划、避免重复建设

■ 标准化是前提

■ 产品化设计、产品化开发

■ 服务驱动

■ 运维中台

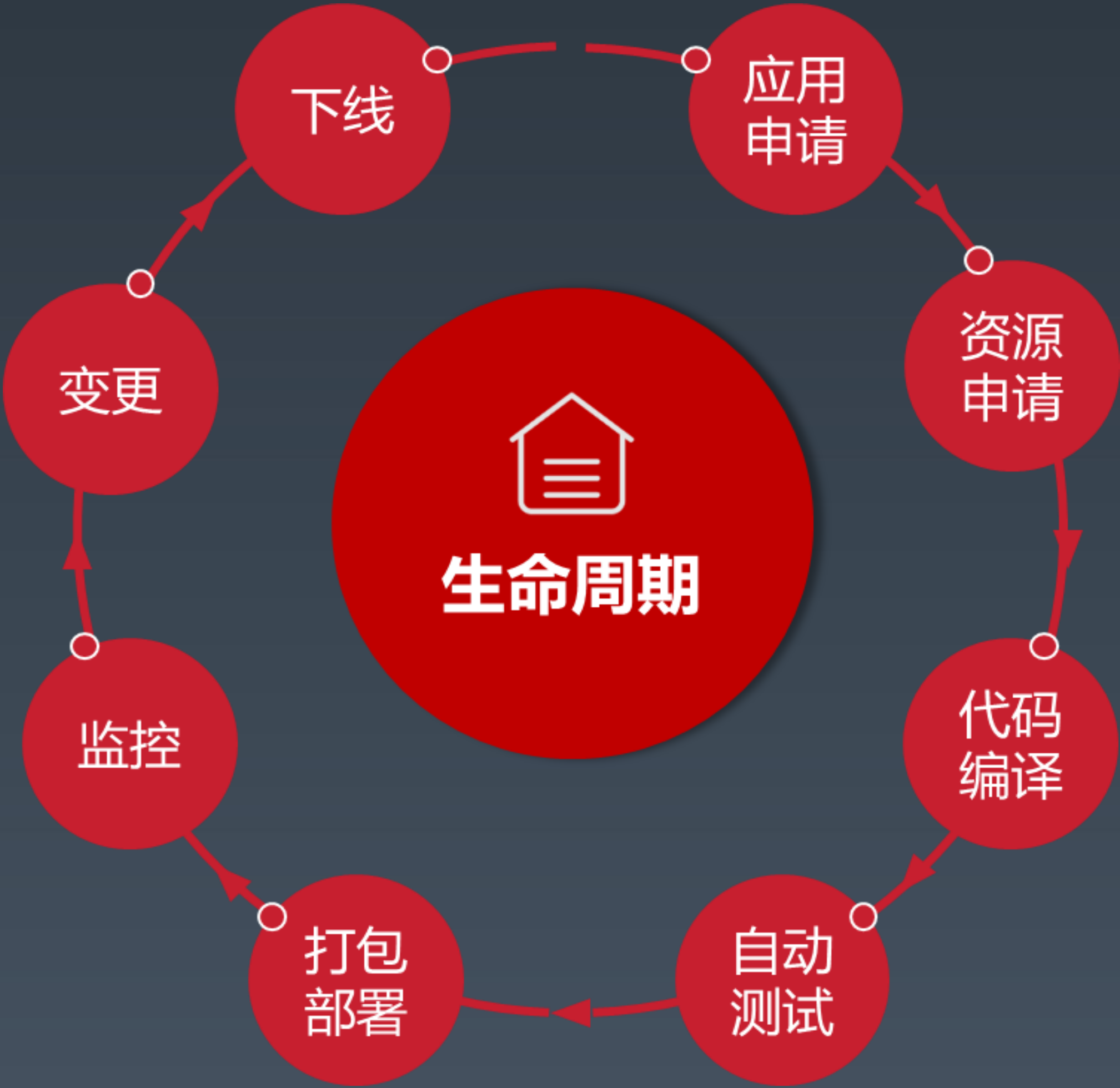
■ 业务增值

■ 过程改进



智能运维体系建设方法论

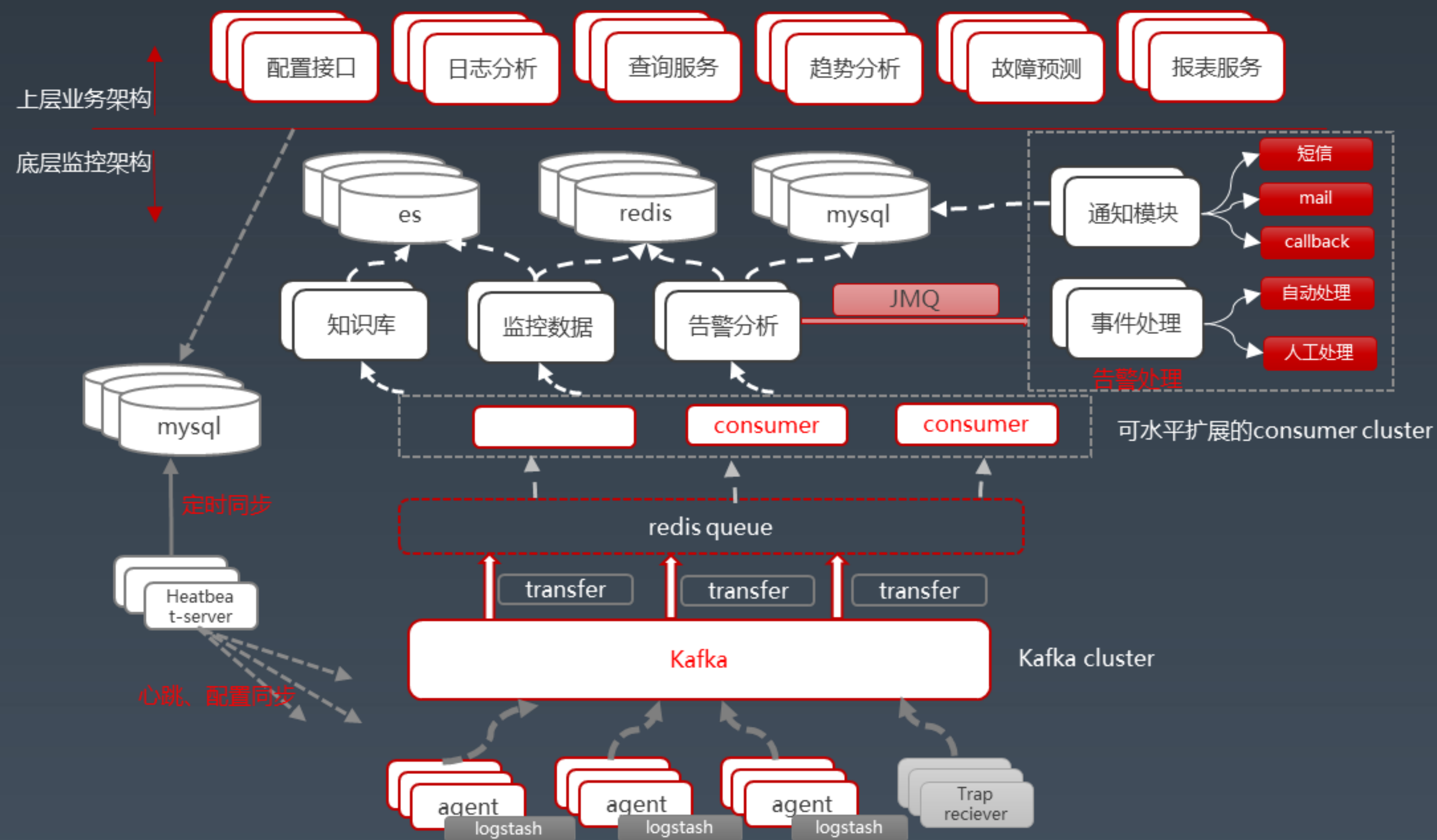
- 闭环
- 生命周期管理
- 流程管理
- 审计归档



目录

- 业界智能运维发展现状及趋势分析
- 智能运维体系建设方法论
- 大规模实时监控平台的实践方案
- 智能故障定位与处理实践
- APM 在京东物流的落地实践
- 智能运维(AIOps)落地规划

大规模实时监控平台V1.0

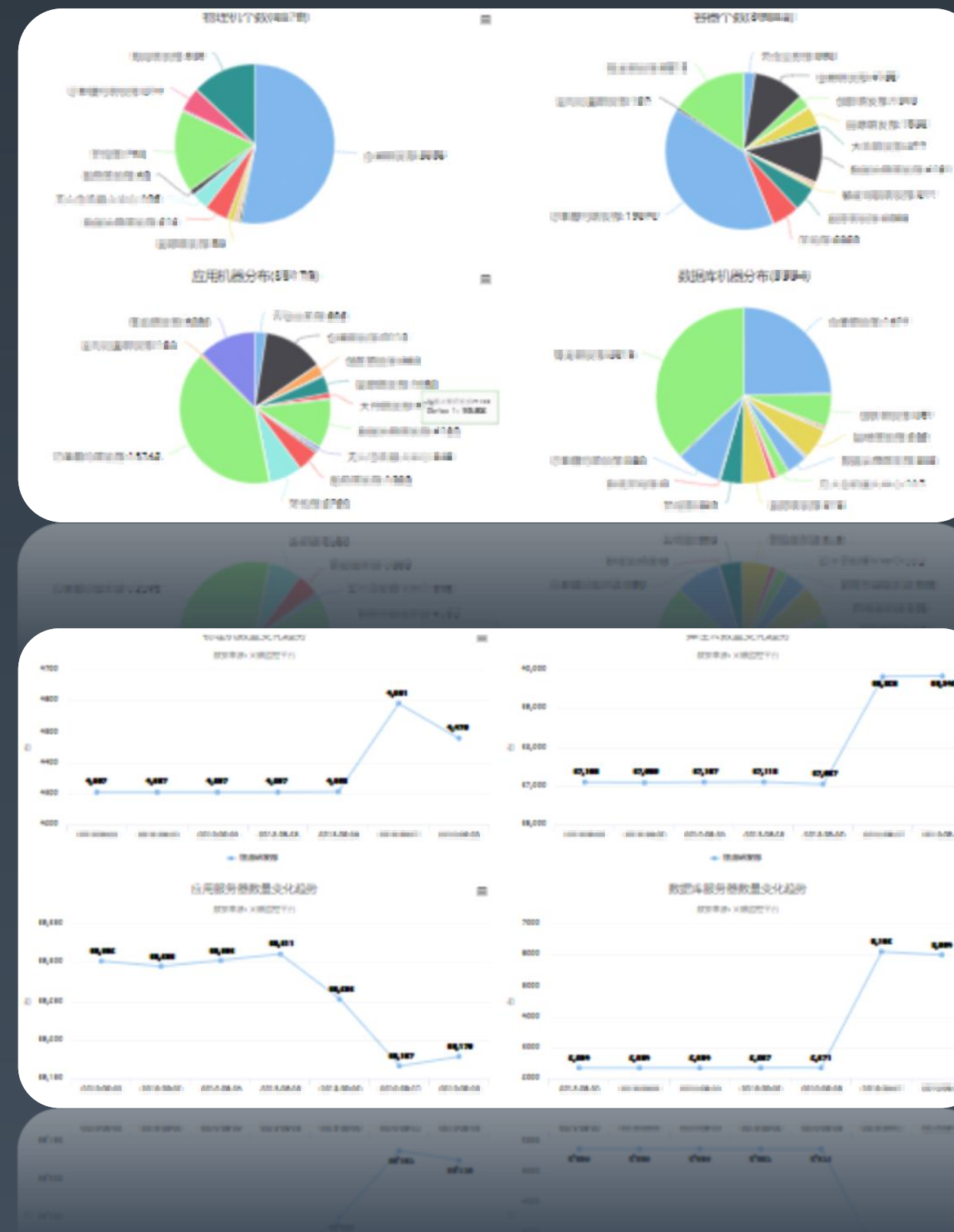


大规模监控平台架构

大规模实时监控平台V1.0

多维度使用率分析助力企业降本增效

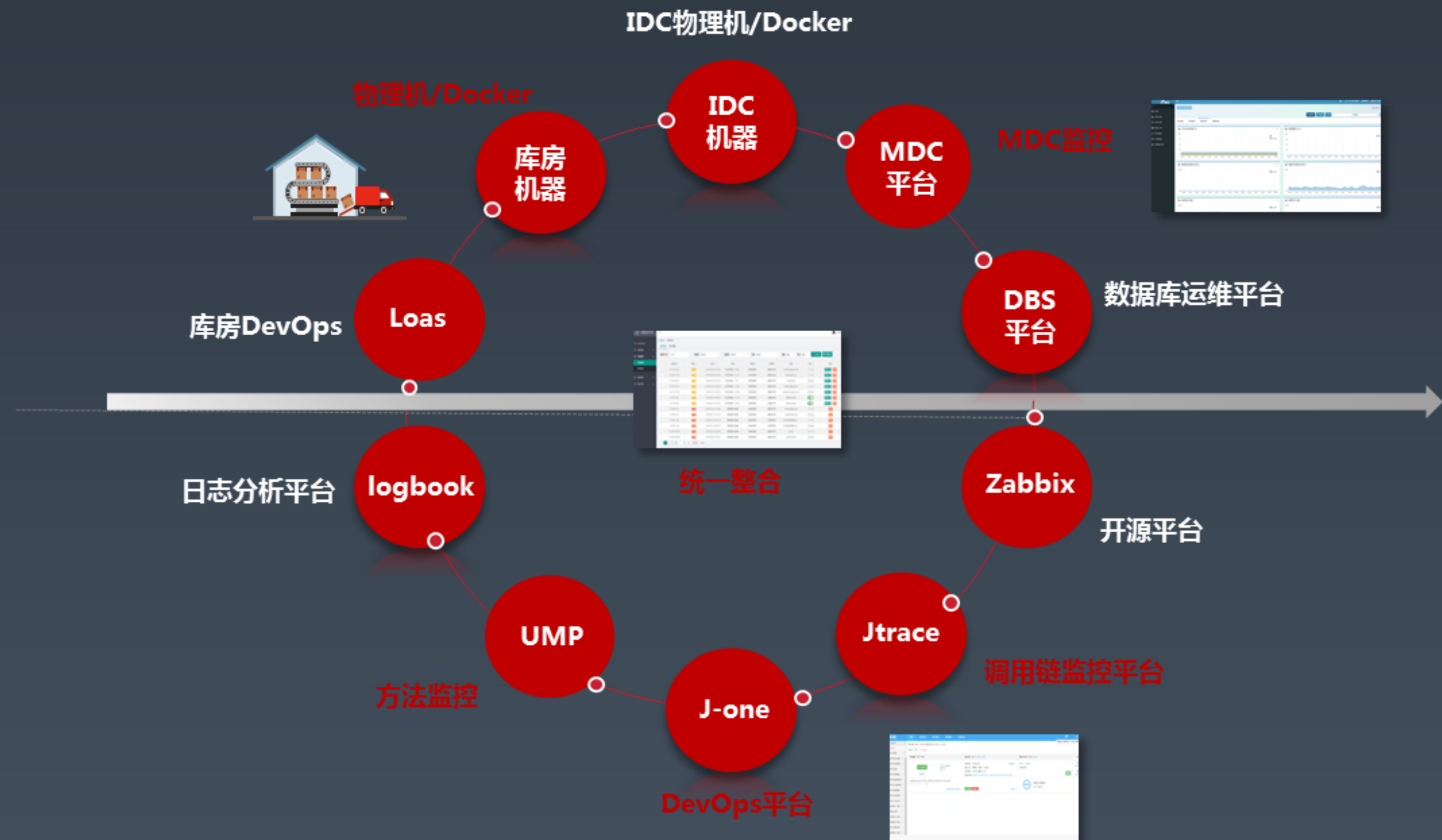
- 多级部门、应用多维度统计
- 日报、周报、同比、环比统计
- 低资源使用率TOP统计
- 低负载应用榜单
- 低资源使用率应用优化建议



使用率报表

大规模实时监控平台V2.0

- 整合多端数据，解决数据孤岛问题
- 性能分析、告警分析更加准确
- 更全面评估应用健康状况



大规模实时监控平台V2.0

整合各种应用维度的指标分析，提供更全面的应用数据分析和故障诊断

■ 系统指标

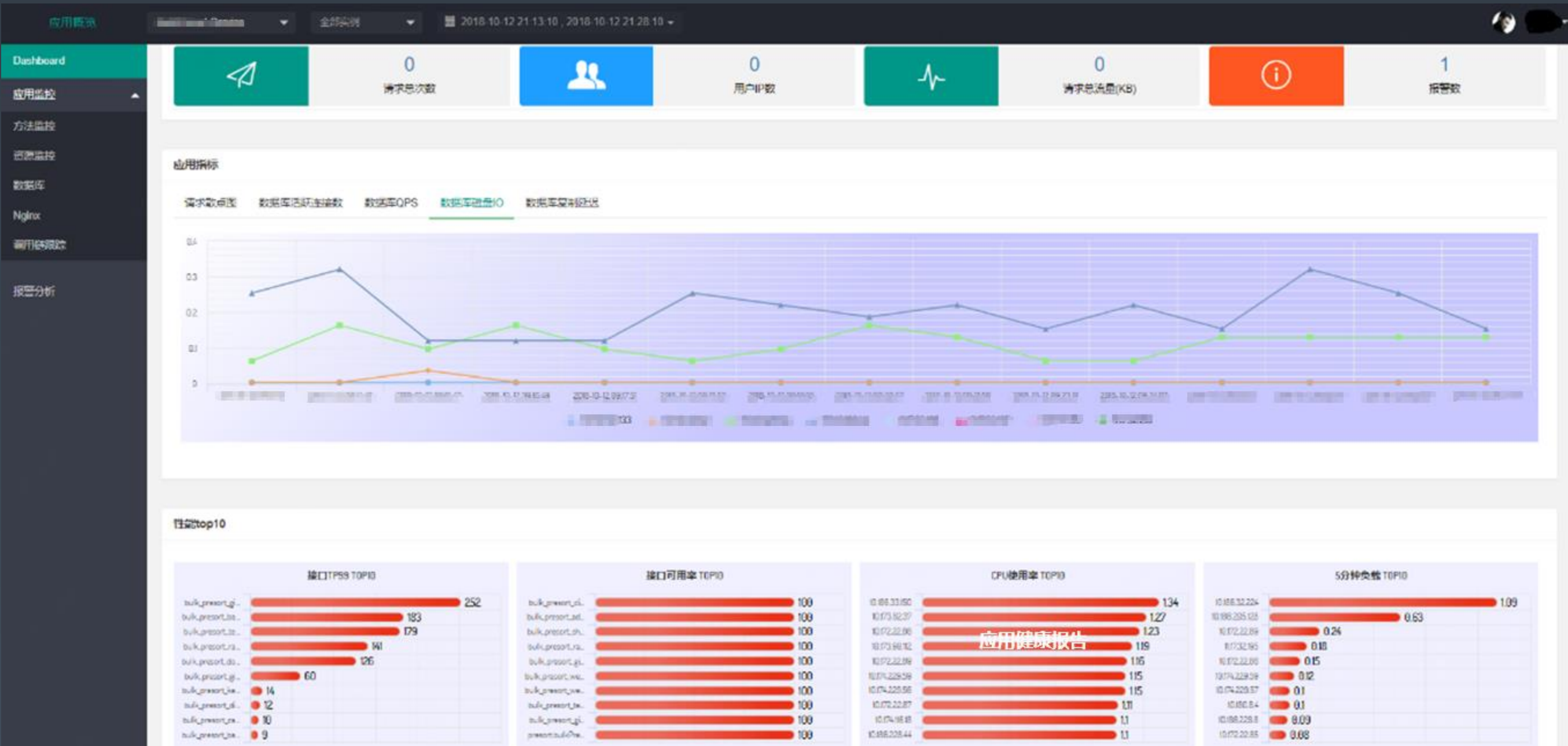
■ 调用链指标

■ 日志分析

■ 数据库指标

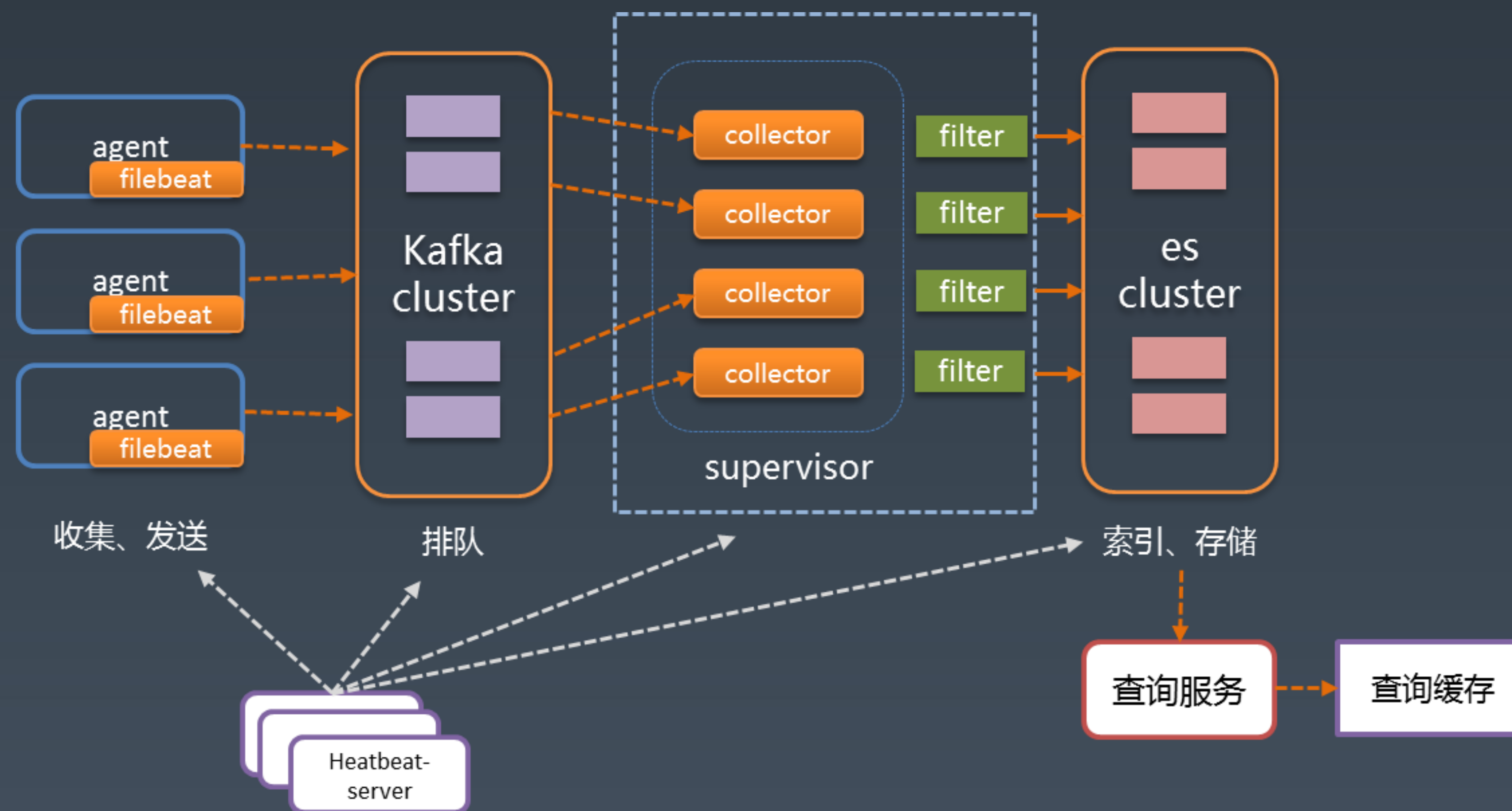
■ JVM指标

■ 应用拓扑自动探测



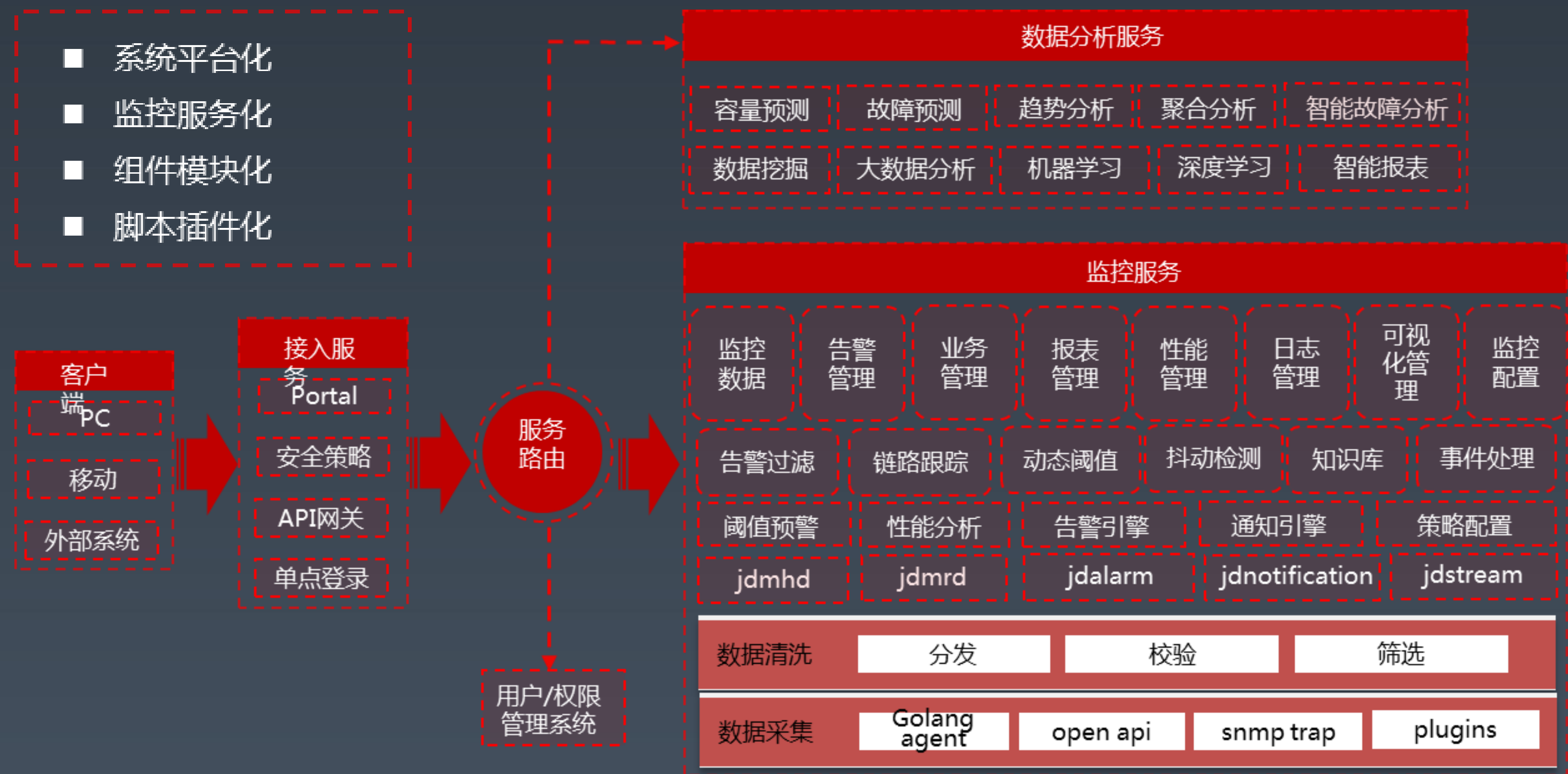
应用健康报告

大规模实时监控平台V2.0



日志处理架构

大规模实时监控平台V3.0



产品规划

大规模实时监控平台V3.0

预测分类： 故障预测、容量预测、性能预测

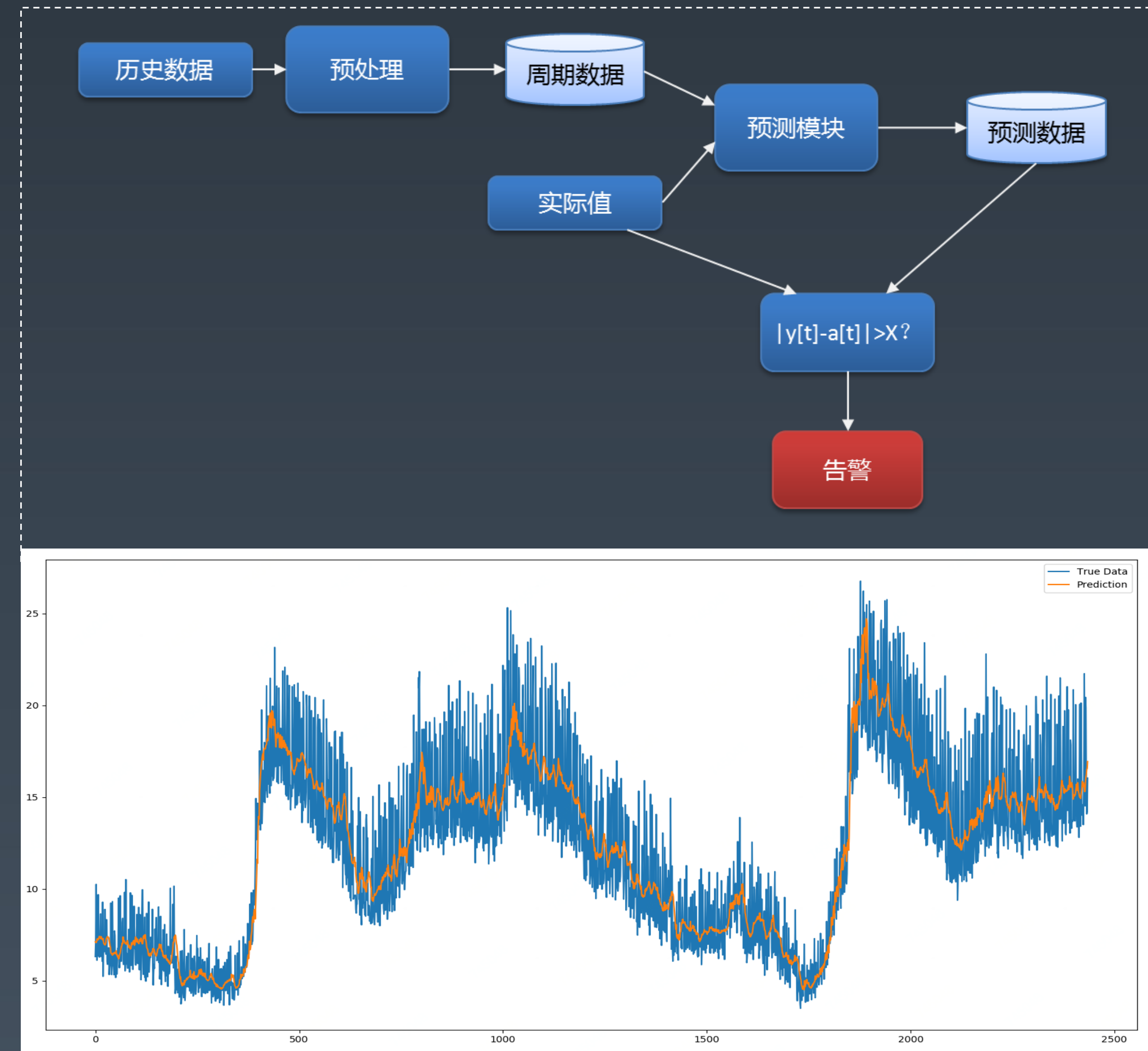
预测算法： LSTM、多元线性回归、决策树、随机森林、神经网络、朴素贝叶斯分类、最小二乘法、支持向量机 ...

重点关注： 算法匹配度评分

Kpi自动分类并匹配预测算法

日历适配、基于节假日的机器学习算法

基于业务关联关系的预测算法



预测

大规模实时监控平台V3.0

红绿灯



大屏



可视化

目录

- 业界智能运维发展现状及趋势分析
- 智能运维体系建设方法论
- 大规模实时监控平台的实践方案
- 智能故障定位与处理实践
- APM 在京东物流的落地实践
- 智能运维(AIOps)落地规划

智能故障处理

传统故障处理

被动故障处理：

1. 事后处理：出先故障后开始处理，易造成业务中断；
2. 人工处理：基于工作流的故障上报和处理，层层通知手工定位故障原因，故障修复时间长；
3. 无计划性：多为突发情况，进行临时处理，难免有疏漏之处；
4. 报警爆炸：随着业务增长，报警越来越多，运维人员不堪其扰



智能故障处理

主动故障处理：

1. 事前感知：通过故障预测算法，预测故障类型及发生时间，并提前通知项目负责人；
2. 自动处理：决策引擎根据预设的事件处理策略，自动执行处理指令以及基于机器学习的自动故障处理；
3. 定时巡检：平台化的定时巡检机制，给出应用健康报告，问题早发现早解决；
4. 报警收敛：对告警做告警筛选、过滤、合并操作，大大减少报警数量；



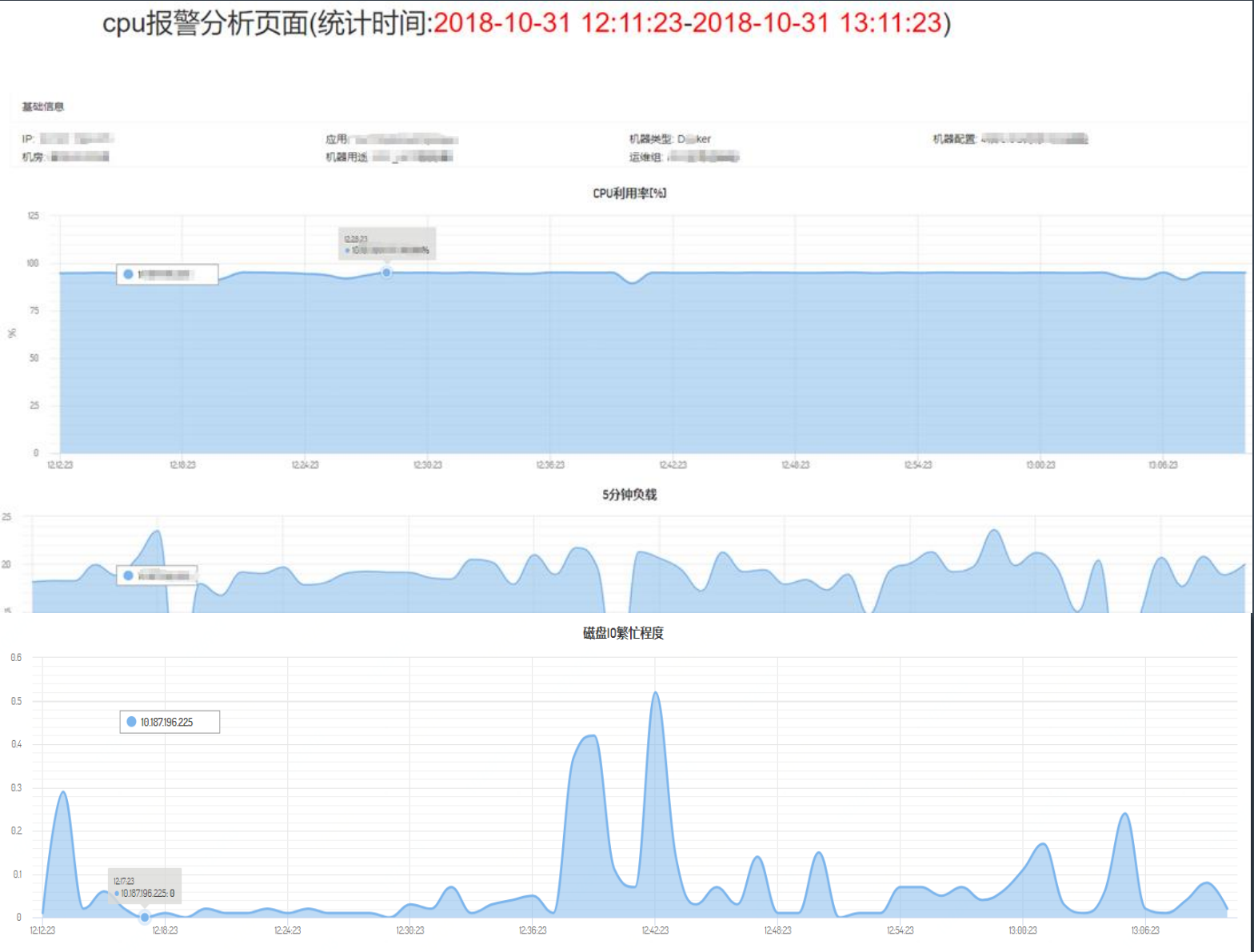
故障快照

■出现告警自动抓取现场快照信息

■快照信息持久化保存

■根据自学习的知识库提供异常原因分析

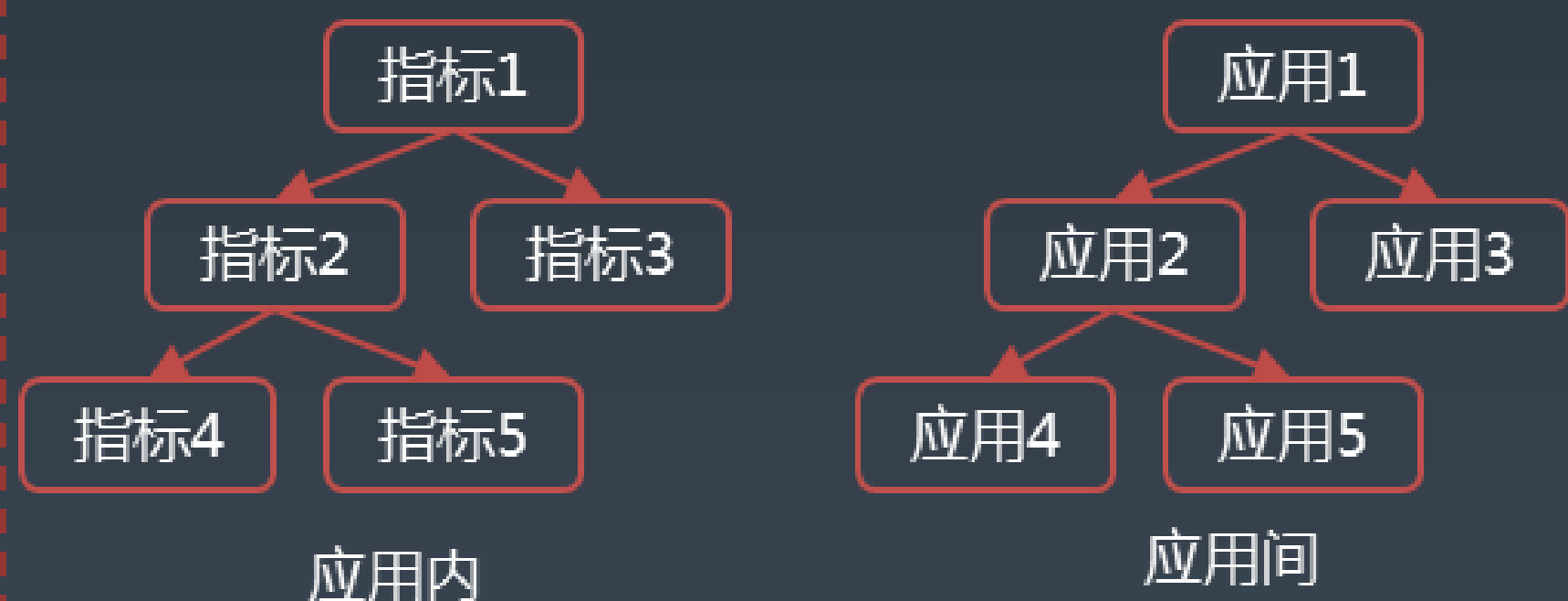
■集成Arthas诊断工具，快速诊断问题



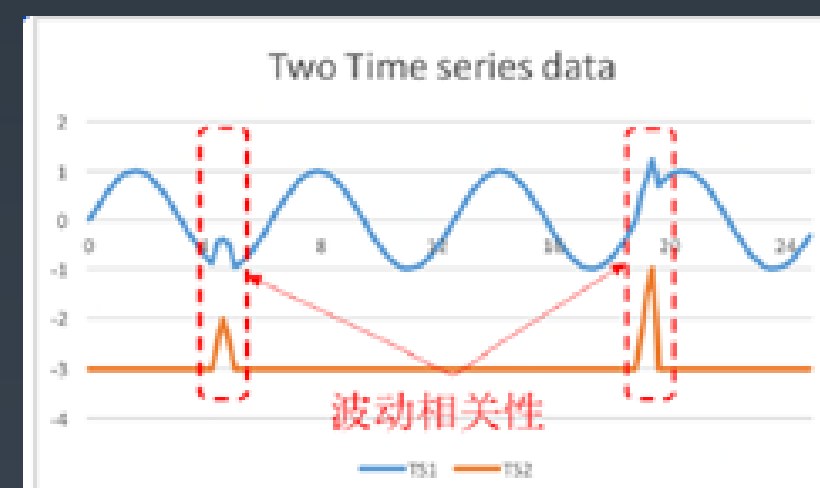
ID	NAME	GROUP	PRIORITY	STATE	%CPU	TIME	INTERRUPTED	DAEMON
26643	http-nio-1601-exec-478	main	5	TIMED_WAITING	39	12:42	false	true
69614	http-nio-1601-exec-523	main	5	TIMED_WAITING	16	7:58	false	true
18598	http-nio-1601-exec-458	main	5	TIMED_WAITING	10	16:44	false	true
282	J5F-Future-Checker-0-T-1	main	5	TIMED_WAITING	5	4:49	false	true
387	SystemClock	main	5	TIMED_WAITING	2	26:31	false	true
856	System_Clock	main	5	TIMED_WAITING	2	26:17	false	true
214	UMP-WriteLog2FileThread-bizLogger	main	5	TIMED_WAITING	2	17:53	false	true
213	UMP-WriteLog2FileThread-businessLogger	main	5	TIMED_WAITING	2	17:46	false	true
216	UMP-WriteLog2FileThread-commonLogger	main	5	TIMED_WAITING	2	17:35	false	true
215	UMP-WriteLog2FileThread-tpLogger	main	5	TIMED_WAITING	2	17:53	false	true
Memory								
	used	total	max	usage	GC			
heap	2723M	4062M	4062M	67.04%	gc-parnew.count		138093	
par_eden_space	4M	260M	260M	1.60%	gc-parnew.time(ms)		2098154	
par_survivor_space	33M	33M	33M	100.00%	gc-concurrentmarkswEEP.count		27	
cms_old_gen	2686M	3763M	3763M	71.38%	gc-concurrentmarkswEEP.time(ms)		45898	
nonheap	301M	309M	1792M	16.82%				
Runtime								
os.name	Linux							
os.version	2.6.32-504.16.2.el6.x86_64							
java.version	1.8.0_51							
java.home	/export/servers/jdk1.8.0_60/jre							
systemload.average	6.31							
"http-nio-1601-exec-534" Id=131649 cpuUsage=8% TIMED_WAITING on java.util.concurrent.locks.AbstractQueuedSynchronizer\$ConditionObject@4c6881e6								
at sun.misc.Unsafe.park(Native Method)								
- waiting on java.util.concurrent.locks.AbstractQueuedSynchronizer\$ConditionObject@4c6881e6								
at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:215)								
at java.util.concurrent.locks.AbstractQueuedSynchronizer\$ConditionObject.awaitNanos(AbstractQueuedSynchronizer.java:2078)								
at java.util.concurrent.LinkedBlockingQueue.poll(LinkedBlockingQueue.java:467)								
at org.apache.tomcat.util.threads.TaskQueue.poll(TaskQueue.java:85)								
at org.apache.tomcat.util.threads.TaskQueue.poll(TaskQueue.java:31)								
at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1066)								
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1127)								
at java.util.concurrent.ThreadPoolExecutor\$Worker.run(ThreadPoolExecutor.java:617)								
at org.apache.tomcat.util.threads.TaskThread\$WrappingRunnable.run(TaskThread.java:61)								
at java.lang.Thread.run(Thread.java:745)								
"System_Clock" Id=856 cpuUsage=3% TIMED_WAITING on java.util.concurrent.locks.AbstractQueuedSynchronizer\$ConditionObject@2a2ea835								
at sun.misc.Unsafe.park(Native Method)								
- waiting on java.util.concurrent.locks.AbstractQueuedSynchronizer\$ConditionObject@2a2ea835								
at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:215)								
at java.util.concurrent.locks.AbstractQueuedSynchronizer\$ConditionObject.awaitNanos(AbstractQueuedSynchronizer.java:2078)								
at java.util.concurrent.ScheduledThreadPoolExecutor\$DelayedWorkQueue.take(ScheduledThreadPoolExecutor.java:1093)								
at java.util.concurrent.ScheduledThreadPoolExecutor\$DelayedWorkQueue.take(ScheduledThreadPoolExecutor.java:809)								
at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1067)								
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1127)								
at java.util.concurrent.ThreadPoolExecutor\$Worker.run(ThreadPoolExecutor.java:617)								
at java.lang.Thread.run(Thread.java:745)								

根因分析

①业务关系构建



②告警关联分析



联动分析

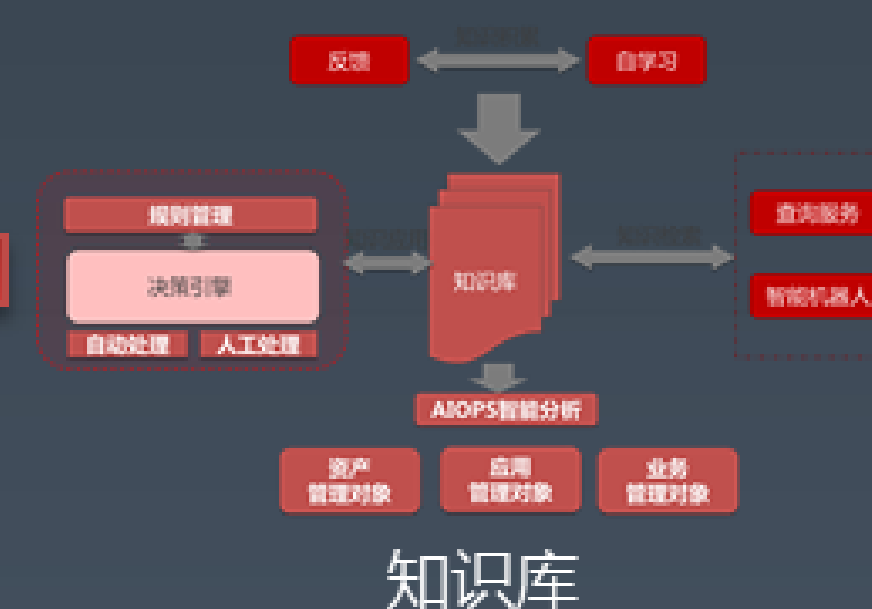
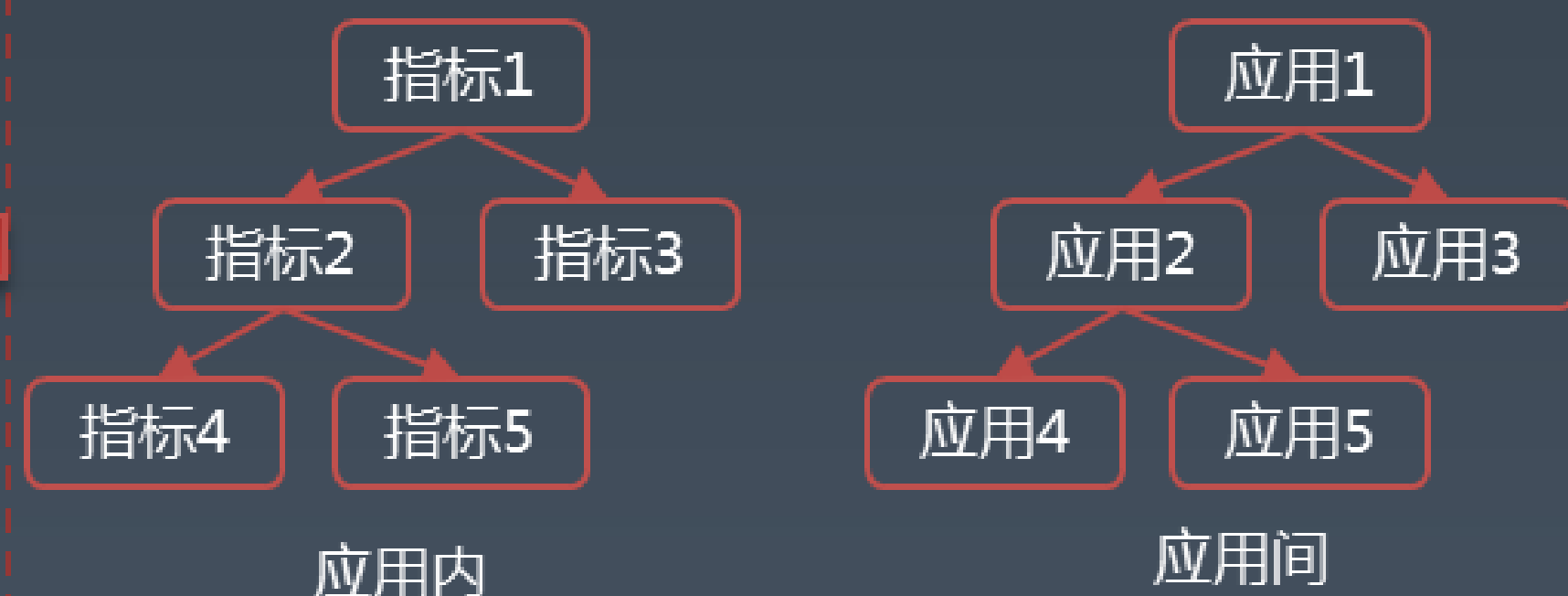
故障分类	告警	可能的原因
故障1	告警1,告警2,告警3,...	原因1,原因2,原因3...
故障2	告警1,告警2,...	原因1,原因2...
故障n	告警2,告警3,...	原因2,原因3...

基于故障知识库的告警关联分析

④最终的告警影响关系

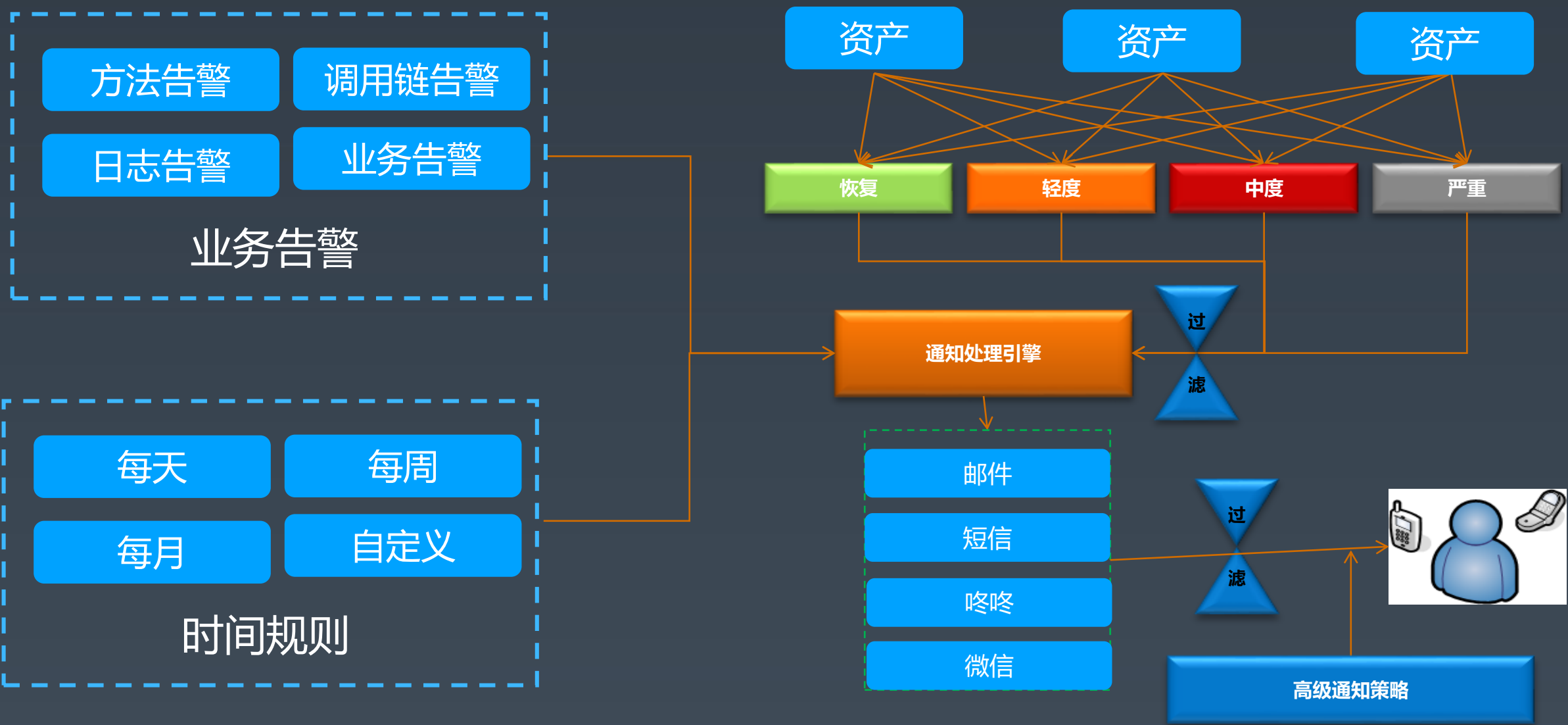


③可能的告警影响关系



基于双向过滤的告警通知

- 为保证告警信息能够及时准确的传达给系统管理员，监控模块需要实现灵活的告警通知策略
- 双重过滤的通知方式：资源和通知联系人分别应用通知策略，实现对通知的双重安全过滤



目录

- 业界智能运维发展现状及趋势分析
- 智能运维体系建设方法论
- 大规模实时监控平台的实践方案
- 智能故障定位与处理实践
- APM 在京东物流的落地实践
- 智能运维(AIOps)落地规划

业界分布式跟踪系统

Google: Dapper

Naver: Pinpoint

Twitter: Zipkin

点评: Cat

阿里: EagleEye

京东: JTrace、JD-Hydra (已废弃)、Callgraph、SGM

新浪: Watchman

美团: MTrace

又拍云: Tail

其他: OpenTracing、SkyWalking

服务厂商: Compuware、iMaster、博睿Bonree、听云、New Relic、云智慧、OneAPM、AppDyn、Amics

京东物流Jtrace分布式跟踪系统

定义了四个具体的设计目标



低消耗



应用级透明



延展性

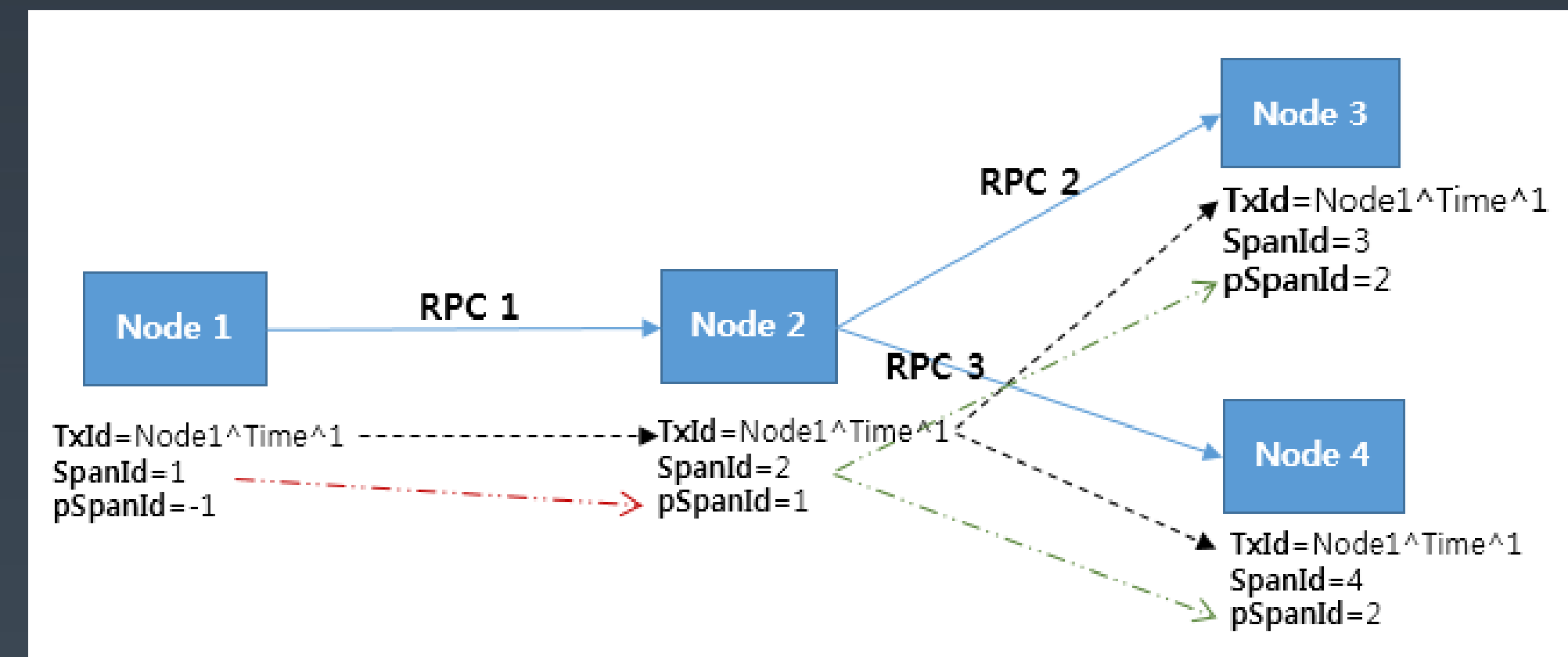


智能分析

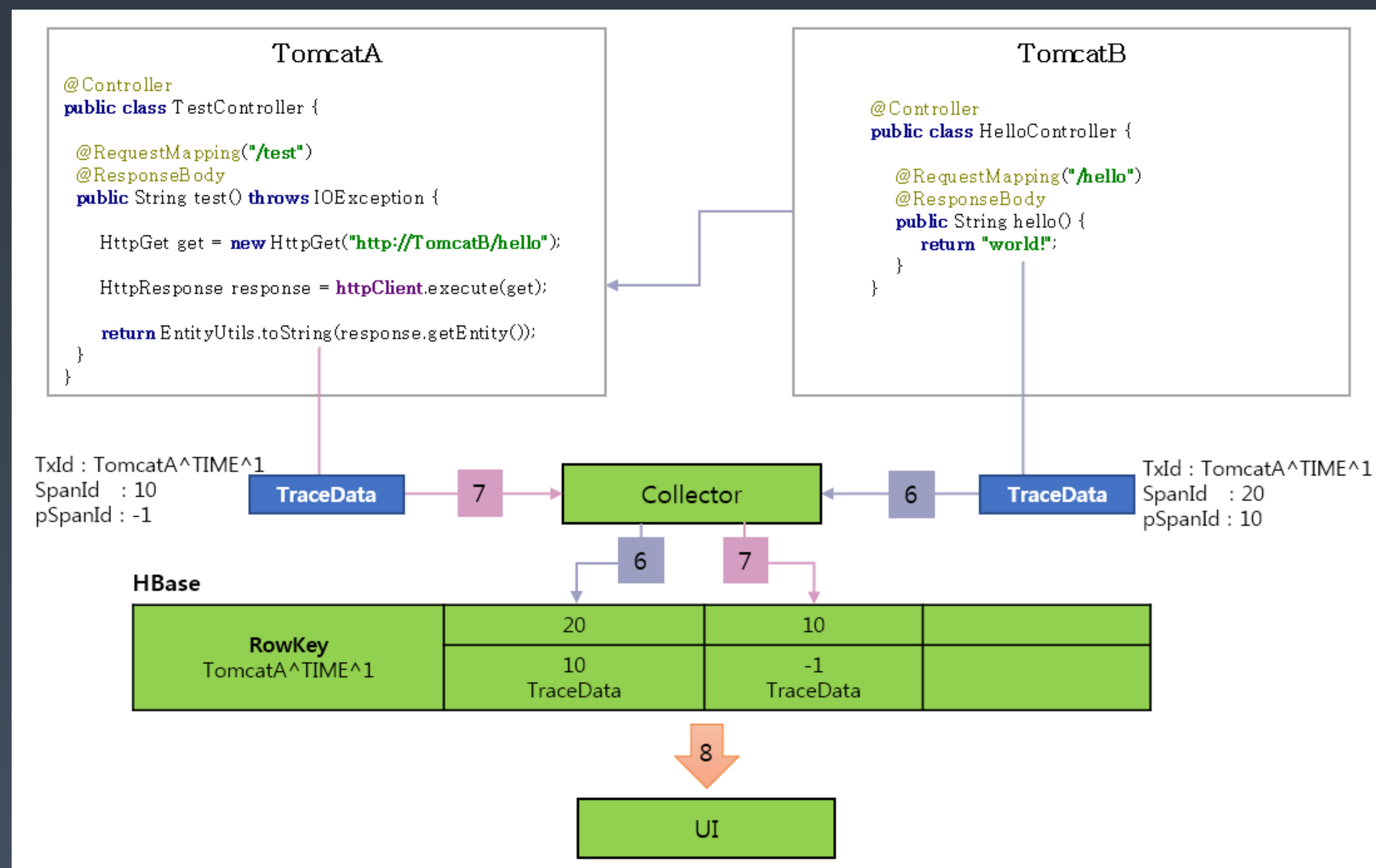
JTrace数据结构

核心数据结构由Span, Trace, 和 Traceld组成:

- Trace: 多个Span的集合;
- Span: RPC跟踪的基本单元;
- SpanEvent: 内部方法调用基本单元
- Traceld:
 - TransactionId (TxId) : 全局唯一消息的ID
 - SpanId
 - ParentSpanId (pSpanId)



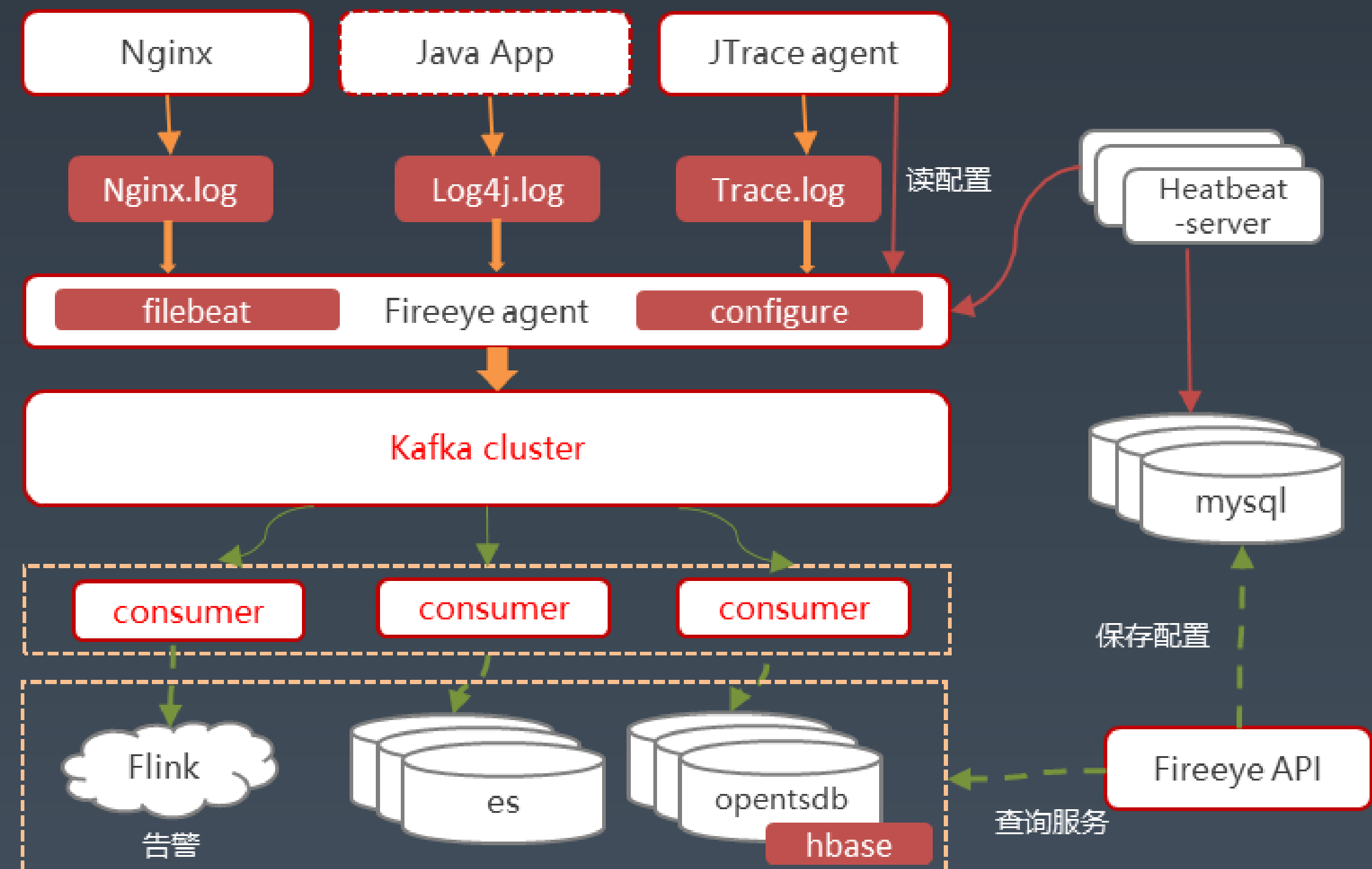
Jtrace应用示例



架构设计

七大能力：

- 分布式事务跟踪，跟踪分布式应用消息
- 自动检测应用拓扑，帮你搞清楚应用的架构
- 水平扩展支持大规模服务器集群
- 提供代码级别的可见性以便轻松定位失败点和瓶颈
- 使用字节码增强技术，添加新功能无需改动代码
- 集成SQLAdvisor
- 智能化采样率



字节码增强技术

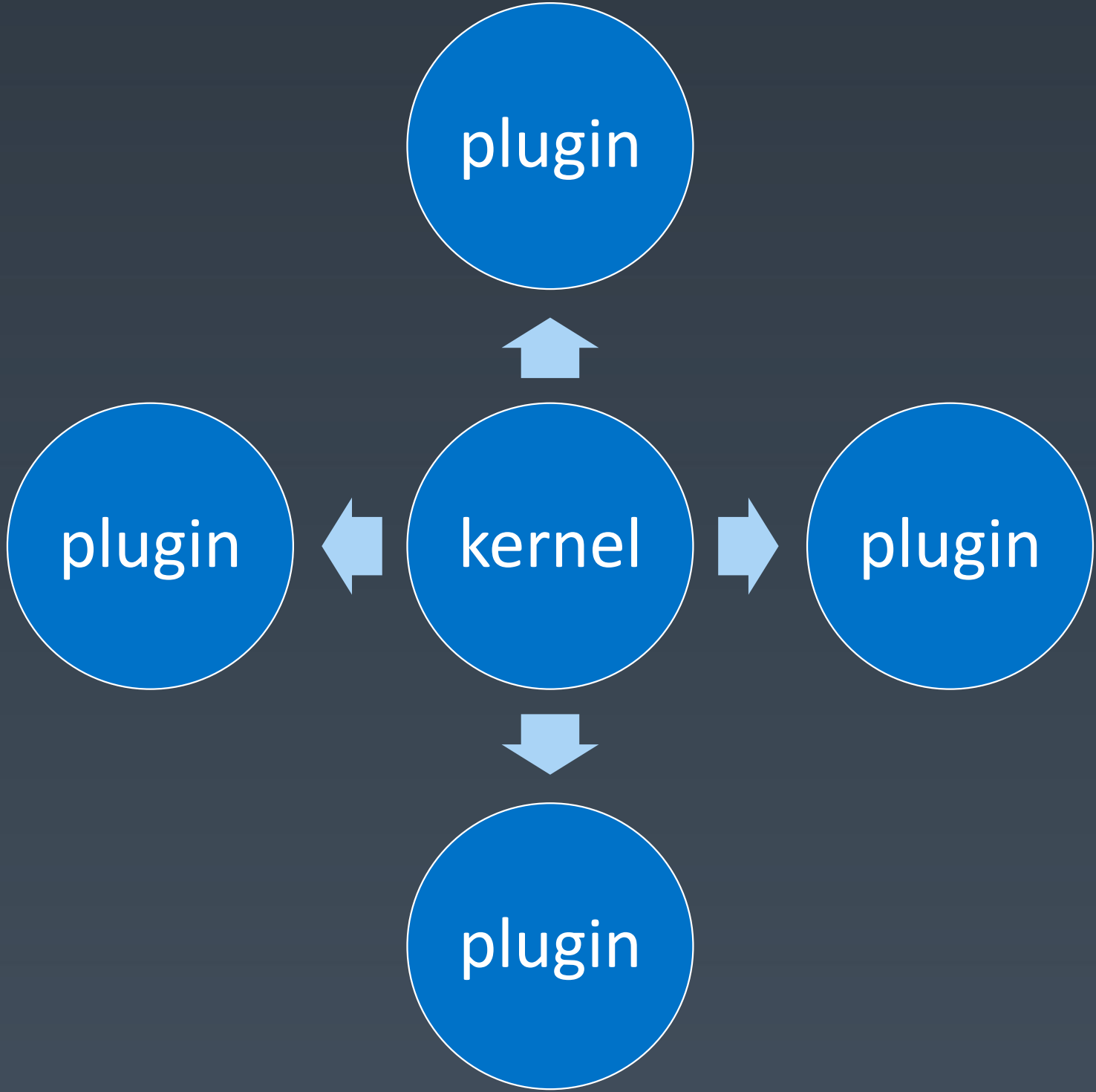
JavaAgent:

```
java -javaagent:myagent.jar=mode=test Test
```

功能:

- 可以在加载class文件之前做拦截，对字节码做修改
- 可以在运行期对已加载类的字节码做变更，但是这种情况下会有很多的限制。
- 还有其他一些小众的功能
 - 获取所有已经加载过的类
 - 获取所有已经初始化过的类（执行过clinit方法，是上面的一个子集）
 - 获取某个对象的大小
 - 将某个jar加入到bootstrap classpath里作为高优先级被bootstrapClassloader加载
 - 将某个jar加入到classpath里供AppClassloard去加载
 - 设置某些native方法的前缀，主要在查找native方法的时候做规则匹配

java字节码框架



Agent内部是采用微内核+插件的方式

微内核:

封装了通过ASM或Javassist字节码框架对类进行增强

插件:

插件中指定要增强的类和方法以及增强内容

	优点	缺点
手工埋点	1. 要求更少开发资源 2. API可以更简单并最终减少bug的数量	1. 开发人员必须修改代码 2. 跟踪级别低
自动埋点	1. 开发人员不需要修改代码 2. 可以收集到更多精确的数据因为有字节码中的更多信息	1. 开发难 2. 开发人员要求高 3. 增加bug发生的可能性

字节码增强的价值

隐藏API

一旦API被暴露给开发人员使用，我们作为API的提供者，就不能随意的修改API。这样的限制会给我们增加压力。

而使用字节码增强技术，我们就不必担心暴露跟踪API而可以持续改进设计，不用考虑依赖关系。

容易启用或者禁用

使用字节码增强的缺点是当JTrace自身类库的采样代码出现问题时可能影响应用。不过，可以通过启用或者禁用JTrace来解决问题，很简单，因为不需要修改代码。

```
-javaagent:$AGENT_PATH/pinpoint-bootstrap-$VERSION.jar  
-Dpinpoint.applicationName=<The name indicating a same service>
```

APM性能优化

- 使用二进制格式(thrift协议)
- 使用变长编码和格式优化数据记录 (thrift CompactProtocol)
- 用常量表替换重复的API信息, SQL语句和字符串
- 处理大量请求的采样
- 使用异步数据传输来最小化应用线程中止
- 使用UDP协议传输数据

目前接入应用**677**个, 接入机器近**9000**台

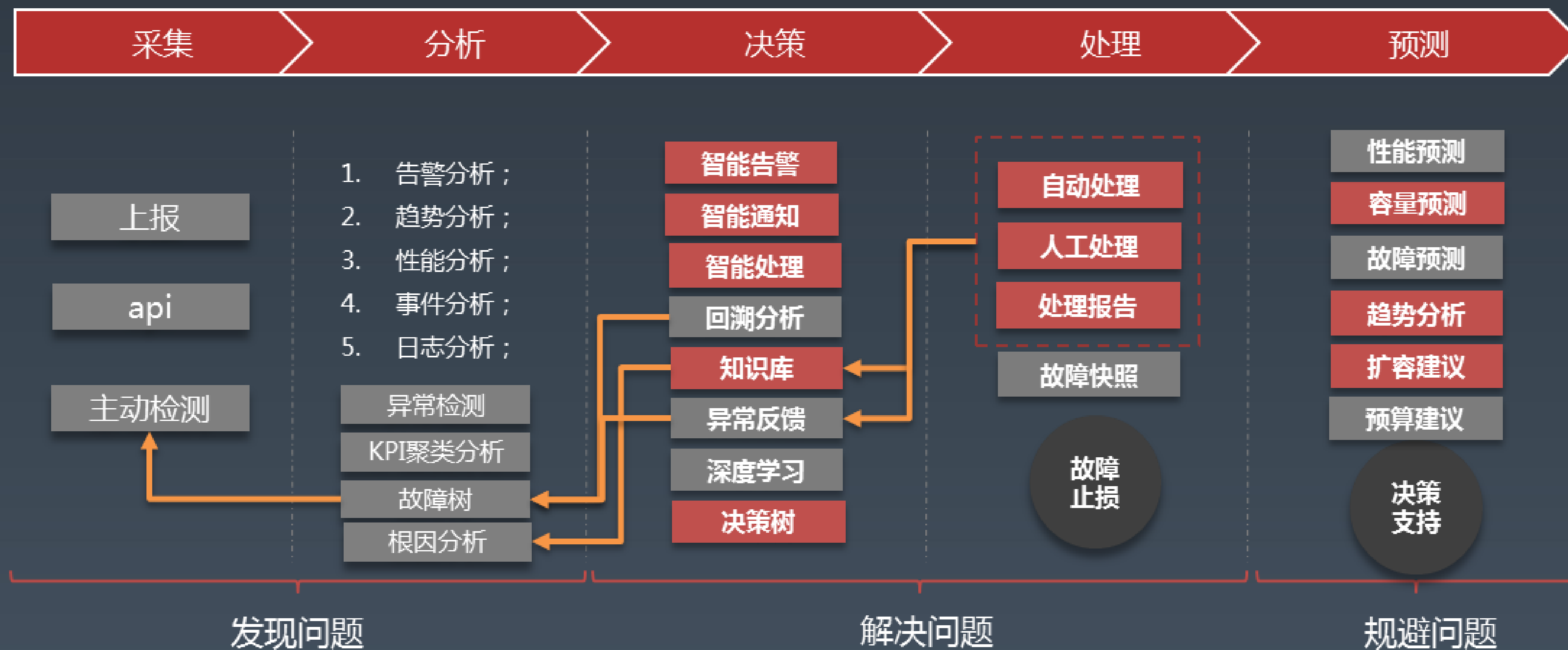
经过数论压测计算Agent端会有**3%**的性能损失

到目前为止还没有出现因为Agent出现性能问题。

目录

- 业界智能运维发展现状及趋势分析
- 智能运维体系建设方法论
- 大规模实时监控平台的实践方案
- 智能故障定位与处理实践
- APM 在京东物流的落地实践
- 智能运维(AIOps)落地规划

AIOP总体建设思路



AIOPS落地规划



想做团队的领跑者 需要迈过这些“槛”

成长型企业，易忽视人才体系化培养
企业转型加快，团队能力又跟不上

VS

从基础到进阶，超100+一线实战
技术专家带你系统化学习成长

团队成员技能水平不一，
难以一“敌”百人需求

VS

解决从小白到资深技术人所遇到
80%的问题

寻求外部培训，奈何价更高且
集中式学习

VS

多样、灵活的学习方式，包括
音频、图文 和视频

学习效果难以统计，产生不良循环

VS

获取员工学习报告，查看学习
进度，形成闭环



课程顾问「橘子」

回复「QCon」
免费获取
学习解决方案

极客时间企业账号 # 解决技术人成长路上的学习问题

THANKS!

QCon 