Flashcat

Observability system construction from method to practice

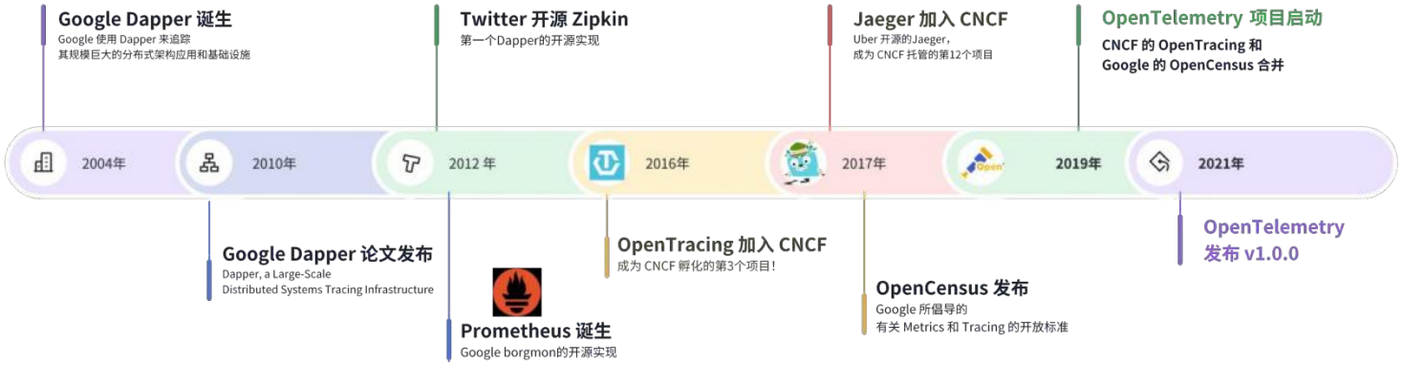
可观测性体系建设从方法到实践

北京快猫星云科技有限公司

目录

前言

2019 年，CNCF 托管的 OpenTracing 项目和 Google 发起的 OpenCensus 项目正式合并，以全新的品牌 OpenTelemetry 亮相，可观测性的概念迅速发展、普及并被广泛接受。截止今日，可观测性体系已经成为数字化服务的必备工具，OpenTelemetry 项目成为 CNCF 所有项目中发展最快的之一。但如何建设一个一体化、智能化、高价值的可观测性体系，仍然是很多企业追求和探索的目标。



本文将对可观测性从方法到实践进行全面解析。解读可观测性的概念、应用的主要场景、建设的范式、智能化路径，并详细介绍一个具体的可观测性产品 Flashcat 的实现。

什么是可观测性

可观测性（Observability）是一种软件开发和系统构建的理念，是对系统内部状态及行为的度量和推断能力，通常包括日志、指标、链路追踪等多个度量维度。也就是说，在软件开发和运维领域中，可观测性是指对于一个复杂的系统，能够通过日志、指标、链路追踪等手段，快速地发现、诊断、解决问题的能力。

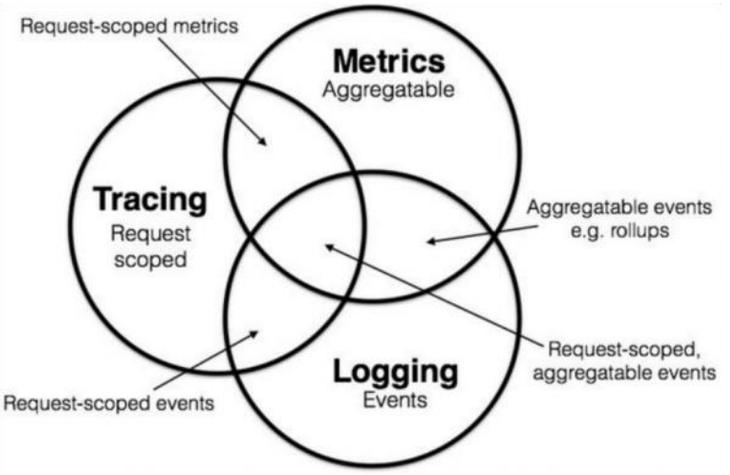
可观测性（Observability）最初是来自控制论的一个概念：

In 1960, Kálmán introduced a characterization he called observability to describe mathematical control systems in his paper. In control theory, observability is defined as a measure of how well internal states of a system can be inferred from knowledge of its external outputs.

简单来看，如果仅使用来自系统输出的信息可以估计和推断系统的当前状态，则系统被认为是“可观测的”。准确理解可观测性，除了观测能力外，还在于一个“性”字，“性”字代表了这个能力的可持续性。因此可观测性是：让服务可持续被观测的规范、标准、方法、工具和平台的集合。

可观测性的三大维度

OpenTelemetry 明确提出了可观测性的三大维度：Metrics（指标）、Logging（日志，包含事件 Events）、Tracing（链路），有些提法会把 Events 从 Logging 中独立出来，单独来看待。



传统监控和可观测性

可观测性是对传统监控在理论、技术和现实问题下的发展。

- ✧ 理论上：将原来各自独立发展的三条监控线统一了起来（Metrics、Logging、Tracing），提供标准化定义。
- ✧ 技术上：完备和发展了三条监控线的技术，并强调将三大支柱的数据打通关联起来，提供标准化工具。
- ✧ 现实问题：在云、容器、微服务的当前，传统监控已经很难适应服务的观测需要，可观测体系应运而生。
- ✧ 应用领域：监控仍然是可观测性应用的主要领域之一，但可观测性的应用场景已不局限于此，可观测性能力可以应用于如性能优化、安全、成本分析，甚至业务分析等等领域。

虽然可观测性的概念显得新颖而有吸引力，监控的概念比较陈旧，但要完全抛开监控来理解可观测性也是不符合实际的。

可观测性的演变

从监控开始，可观测性先后经历了三个阶段：传统基础设施阶段、互联网服务阶段，以及当前的云原生/微服务阶段。

- ✧ **传统基础设施阶段**：基础设施、架构、服务都简单，监控数据少，监控系统甚至可以做到开箱即用。
- ✧ **互联网服务阶段**：基础设施规模变大、架构和服务都开始变得复杂，监控数据开始逐步增长，服务监控的难度逐步增加，服务稳定性保障需要观测的数据维度也开始增加，Metrics（指标）、Logging（日志）、Tracing（链路）三个维度开始快速独立发展。
- ✧ **云原生/微服务阶段**：即当前阶段，基础设施不但规模变大而且呈现出多样性，云和容器出现，混合云基础设施成为普遍常态。同时，微服务的发展让服务的复杂性进一步增加。之前的监控已经难以适应这个变化，开始出现新的监控工具，如 Prometheus、Jaeger、云监控等，而且三大维度的观测能力对保障服务稳定来说都变得非常必要。各类基础设施的监控以及三大维度数据的融合成为趋势。

从核心出发点来讲，传统的监控和可观测性，背后解决的是同样的问题，就是及时、准确的掌握系统的运行状况，提升对系统运行的控制能力。那么从传统监控到可观测性，Gap 到底有多大？构建和完善可观测性体系，有哪些最佳实践，应该从哪些维度入手和进阶，又如何判断企业的可观测性体系建设处于什么样的水平？本章节将给出企业服务可观测性的成熟度模型，以作为可观测性体系建设的参考。

传统监控的局限

- ✧ **侧重于依赖「经验主义」，应对「已知问题」。**

  要预先知晓采集哪些指标，添加什么样的告警策略，定制什么样的仪表盘，以便发现某种类型的故障后，采用什么样的 Runbook 来应对。比如技术团队根据过往经验，知道一台服务器上打开的文件句柄数量不能太多，超过某个上限就会影响到网络通信以及文件读写，因此我们会采集一个 的指标，然后配置一个告警策略：当 则触发告警，同时我们会提前制作一个 Linux 主机 Dashboard，其中包含有 的趋势图。

  准备好这些工作之后，接下来就是守株待兔，等待告警的触发，值班的技术团队就可以按照 Runbook 中载明的排查步骤，检查是否有进程泄露文件句柄，或者是否有大量的网络链接建立等等。经验主义，总是有限的，无法预知可能发生的各种未知的故障。因此在实际情况中，告警策略的完善往往靠“故障复盘”来驱动，每次故障复盘后，必定会有的一个改进项：继续完善监控、加更多的告警。

  技术团队总会处于一种对未知故障缺乏掌控的不安全的状态中，产生焦虑感，反过来又会促使技术团队添加更多的监控，久而久之，告警会越加越多，却又永远不够，告警风暴就这样产生了。

- ✧ **告警驱动的传统监控，缺乏对故障的全局感知。**

  当使用传统监控方式，发出某个告警之后，值班的技术团队看到的只是一个孤立的”技术问题“，这个技术问题的影响面有多大，重要程度如何，是否需要立即处理，是否需要上升和协同，很难快速的做出判断。某个“技术问题”是否重要，是否紧急，不取决于该技术问题本身的难易程度，也不取决于所涉及的服务器规模多寡，唯一的衡量标准是“对用户体验产生的影响有多大”。使用传统监控无法快速的评估某个告警事件和用户体验之间的必然联系，导致无法投入准确的应急处置资源，无法确定合理的应急响应时效，也无法和其他资源产生有效的联动协同，最终使得稳定性保障工作效率低下。

- ✧ **传统监控认为，系统的开发者和系统的维护者，职责是相对分割的。**

  系统在设计之初，开发者的重心在于完成必备的业务逻辑，对于自身运行状态的暴露，并没有考虑的很完善甚至有时候都没有考虑。大家可能会经常遇到，做的好的开发者可能还会打印较为详细的日志，做的不好的，连日志也打的不全，更不必说提供主动暴露系统状态的 Metrics 接口或者为实现 Tracing 进行埋点了。一旦系统到了上线运行阶段，维护人员接手后，往往只能开启“**外挂**”模式，通过写各种各样的脚本，去探测进程是否存在、去分析匹配日志中是否有关键的错误字段。如果要进一步统计系统的访问量、访问延迟、资源消耗等等，就会更加被动。“外挂”往往是传统监控数据采集的特征。

- ✧ **传统监控面向的通常是基础设施，Metrics 是传统监控的基础。**

  基础设施的变化较慢，且变化带来的结果相对可预测。Metrics 类型的监控指标，具有采集存储成本低、简单直观、易于聚合计算的特点，因此在过去的二三十年里，基于 Metrics 为基础，出现了各种各样的采集器、时序数据库、可视化工具、告警工具等，基于前面提到的“经验主义”，尚能应付面向基础设施的稳定性保障工作。

可观测性技术理念

可观测性认为，你的应用是如何运行的以及是否在正确的运行，应该主动地、默认地通过 Metrics、Logging、Tracing、Events 等多种数据维度实时的暴露出来，然后通过工具进行可视化、告警、分析和数据洞察。对应用内部状态和行为的暴露，是系统设计之初就要考虑的重要组成，是系统功能不可分割的一部分。**在可观测体系下，“埋点”是一种文化，应用的开发者承担着主体责任**，系统的维护者反而作为数据的使用方存在。

相较于传统监控关注基础设施，以 Metrics 为基础，可观测性强调面向“Application”，Tracing 是关键抓手。随着云原生架构和微服务模型的普及，现代化的应用出现了一些新的特点：

- ✧ 相比单体应用，技术团队面临着更多的服务需要管理；
- ✧ 很多服务之间都是松耦合，而且像云数据库、云存储、第三方 API 等服务，都不处于你的掌控之下；
- ✧ 代码的发布和变更，频率越来越高，持续集成、持续发布成为主流；
- ✧ 基础设施动态化，容量也在动态的弹性伸缩；
- ✧ 相比单体应用，现代化的系统架构下，可能出现故障的点位越来越多，“长尾问题”出现的频率也越来越高，难以定位和分析；
- ✧ 研发工程师更多的参与到系统的运行维护工作中来；

传统监控所依赖的“经验主义”越发显的经验不够用了，我们迫切需要构建一个分析范式，他能够科学的分析和调查未知的问题。在这个分析范式下，不再重度依赖于“经验主义”，任何工程师都可以借助该范式，自助化或高效的完成故障发现、故障定位、troubleshooting、性能优化等工作。同时，随着大语言模型和智能体技术的快速发展，AI 成为可观测性数据的重要消费者，如何面向 AI 设计可观测性系统，以及将 AI 能力融入到可观测性体系中，以实现智能化的数据分析、问题诊断和故障修复。这将是现代化的可观测性体系建设需要达成的效果。

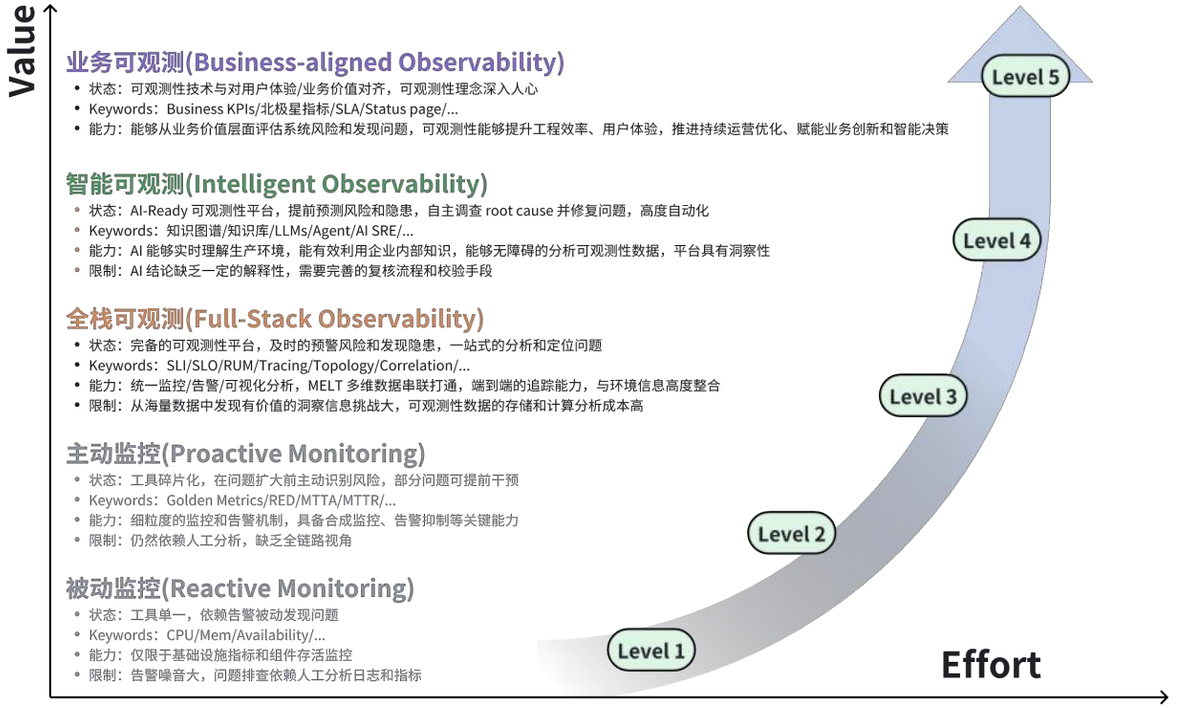
内容纲要

本书将从方法到实践，分别介绍当前可观测性产品服务的主要场景，可观测性产品建设的成熟度模型，以及结合方法论实现的一体化可观测性产品 Flashcat。

- ✧ 路线篇：阐述可观测性技术的发展趋势和演进方向，介绍一个评估可观测性体系成熟度的模型，用以评价一个企业的可观测性体系建设所达到的阶段，并以此作为企业建设或进一步完善可观测性体系的路径和方向参考；
- ✧ 场景篇：分析可观测性面向的主要场景——服务稳定性保障，介绍服务稳定性保障体系的建设思路、方法。其中也包含部分相关系统建设的思路；
- ✧ 实践篇：对稳定性保障的场景展开分析，结合可观测性的成熟度范式，介绍一款可观测性产品 Flashcat 的设计和实现思路；
- ✧ AI 篇：阐述构建 AI 驱动的可观测性体系所需要的技术要素以及技术方法，介绍 Flashcat 在 AI 智能化分析方面的创新工作；

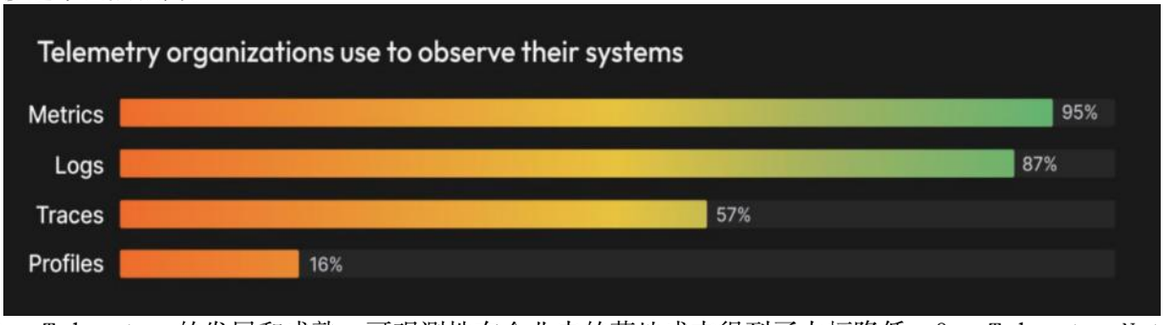
路线篇

准确的评估企业在可观测性技术当前所处的阶段，并提供改进路线图，很有必要性，成熟度模型是一种常见的表达方式，可以有很多版本，无对错之分。下面是 Flashcat 视角下的可观测性五级进化论，从 Level 1（被动监控）到 Level 5（业务可观测），从工具、数据、文化、智能、业务等多个方面综合评估，众多组织处于 Level2 往 Level 3 转型的过程，并且在快速的导入智能化（注意，一个企业不一定非得逐层升级，可以随时跳级，也可以同时实施不同 Level 的内容）。

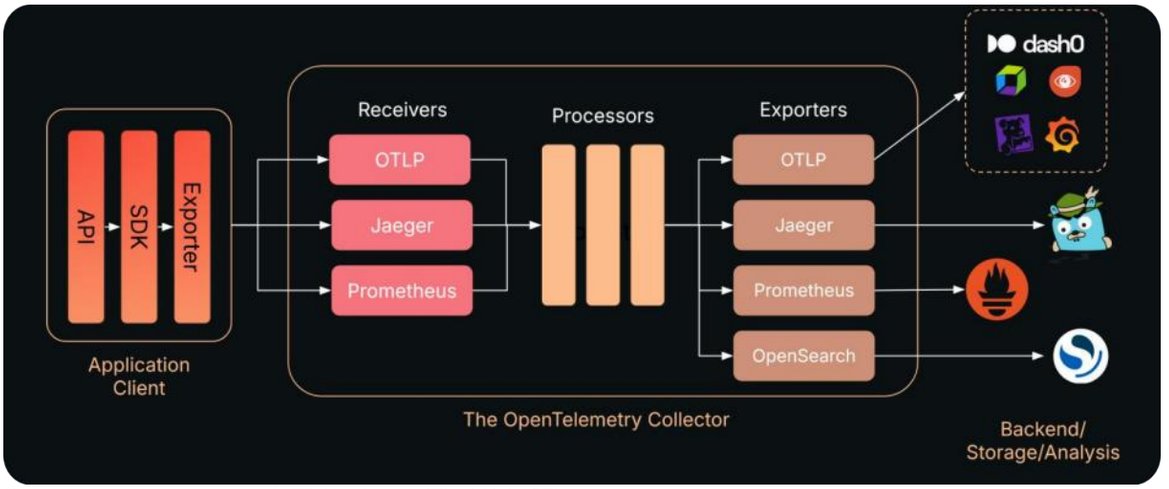


针对可观测性技术在企业的导入和使用情况，New Relic 2025 年对全球范围内的 1700 位工程师和技术管理者进行了一次调查，结果显示：一、AI 监控的使用率在快速提升，已从 2024 年的 42% 提升至 2025 年的 54%；二、企业在加速整合可观测性工具，从两年前平均使用 6 个可观测性工具，下降到现在的 4.4 个，其中 52% 的组织计划在未来的 12~24 个月内整合至统一的可观测性平台。智能化和全栈，是可观测性技术领域正在快速发生的两个变化趋势。

在可观测性领域“三大支柱”的概念中，Metrics 和 Logs 采用率极高，一定程度上是因为传统监控时代已有广泛落地，而 Traces 作为第三大支柱，其采用率一定程度上可以反映可观测性技术的渗透程度，在 Grafana Labs 的 2025 年度调查报告中显示已经有过半企业落地了链路追踪，表明可观测工具已经是企业内部 must-have 的基础设施，可观测性技术和理念的渗透率已然很高。



随着 OpenTelemetry 的发展和成熟，可观测性在企业内的落地成本得到了大幅降低，OpenTelemetry Native 正在成为众多企业的首选方案。



总结来说，OpenTelemetry 解决了以下三个层面的问题：

1. OpenTelemetry 定义了 Metrics、Logs、Traces 的标准规范和相应的 API，实现了针对各种语言的 SDK，解决了可观测性数据生成难、覆盖不全的问题；
2. 围绕 OpenTelemetry 建立了完善的可观测性的软件工具生态，解决了可观测性数据在不同软件工具之间的互通性问题；
3. 解决了 Metrics、Logs、Traces 各个维度数据的关联性问题；

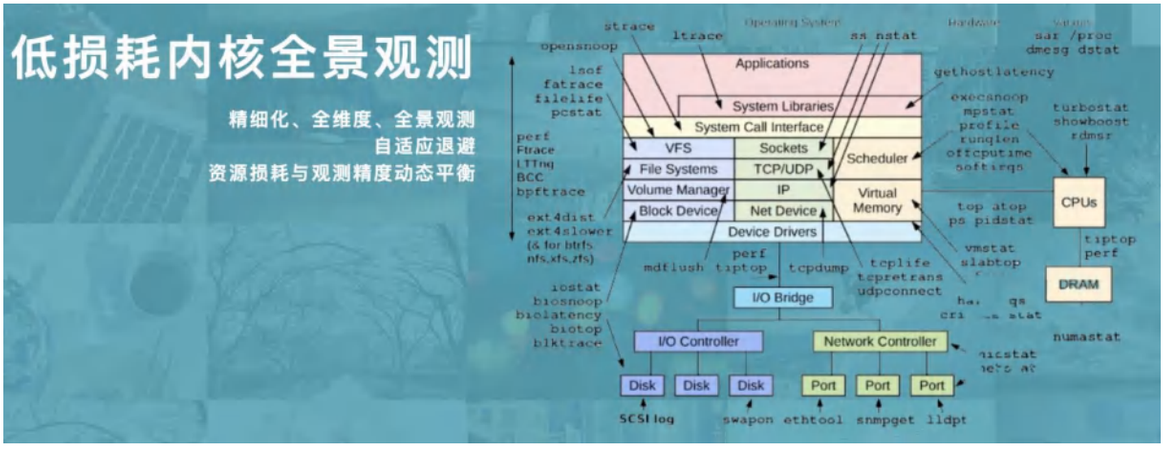
OpenTelemetry eBPF Instrumentation（OBI）于 2025 年 11 月 3 日发布了首个 alpha 版本，标志着 OpenTelemetry 在自动埋点工具方面迈出重要一步。OBI 具备多项技术优势：

- ✧ 支持多语言：Java、.NET、Go、Python、Ruby、Node.js、C、C++、Rust
- ✧ 轻量化：无需修改代码、无需安装依赖库、无需重启应用
- ✧ 高效自动埋点：通过 eBPF 探针采集 Trace 与指标，性能开销极低
- ✧ 分布式追踪：自动采集并上报分布式 Trace Span 至采集器
- ✧ Kubernetes 原生：为 Kubernetes 应用提供免配置的自动埋点能力
- ✧ 加密通信可见性：在不解密的情况下采集 TLS/SSL 加密流量中的信息
- ✧ 上下文传播：自动在服务之间传播 Trace 上下文
- ✧ 协议支持：HTTP/HTTPS、gRPC、gRPC-Web
- ✧ 低基数指标：提供 Prometheus 兼容的低基数指标，降低存储与计算成本
- ✧ 网络可观测性：采集服务之间的网络流量与调用关系
- ✧ 数据库追踪：采集数据库查询与连接信息

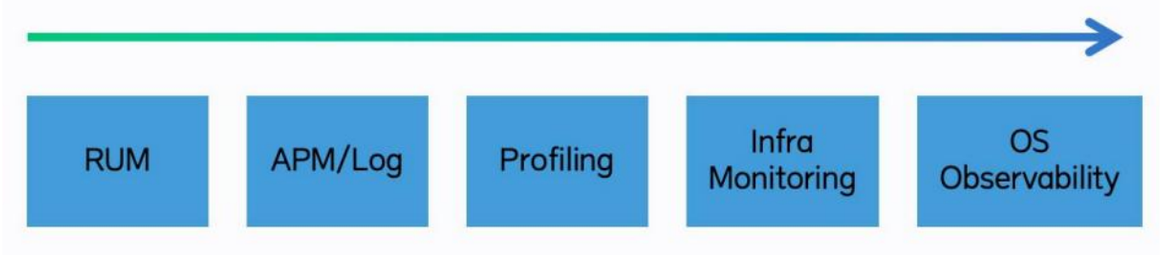
然而，在当前阶段，基于 eBPF 技术的 OBI 方案在大规模应用中仍存在一定局限性，包括对内核版本的要求较高、支持的语言与框架尚不够全面、与手工埋点相比灵活性与粒度有限、权限与环境依赖较强、对异步、协程及复杂上下文传递的支持尚不完善、生态仍在发展、配置与安全管理复杂度较高以及出现问题时排查难度大等。

总体来看，OBI 代表了 OpenTelemetry 在自动埋点与可观测性方向的未来发展趋势，其在多语言支持、无侵入部署、低性能开销与加密流量可见性等方面的优势显著，但若要实现更广泛的生产环境落地，仍需在生态完善、场景适配与易用性等方面持续演进，未来可期。

针对操作系统的深度可观测与实时监控，滴滴研发并开源了 HUATUO 工具。HUATUO 利用 eBPF 技术，常态化运行并全面收集内核各个子系统的统计指标和异常事件（性能损耗小于 1%），并根据生产环境经验值，当统计指标达到一定阈值或者出现某种异常事件时，HUATUO 从内核自动抓取异常上下文现场并及时保存，如抓取用户调用栈、系统态调用栈生成火焰图、异常中断上下文等重要信息。HUATUO 目前已捐赠给中国计算机学会，HUATUO 的开源，打通了操作系统可观测性最后一公里，填补了该领域的空白，详情可以阅读：https://mp.weixin.qq.com/s/3NdVB752aTDKHAB2aAUJfw



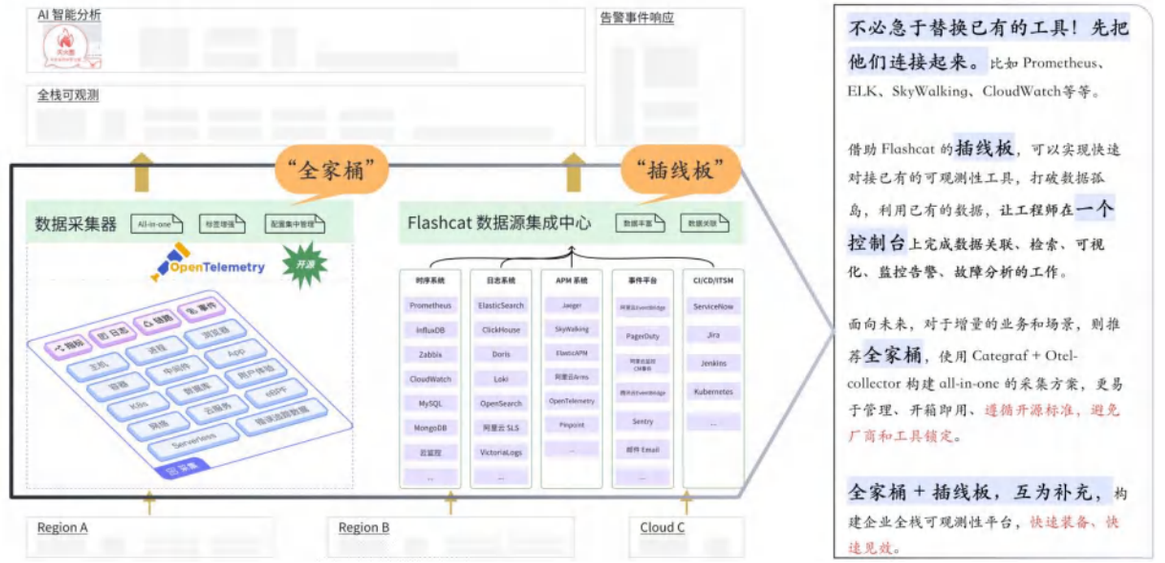
可观测性工具数量多、数据孤岛问题突出仍是众多企业的普遍现象。尺有所长、寸有所短，目前没有一款产品可以在可观测性的各个维度上做到尽善尽美，不同的产品定位不同，擅长的方向不同，企业选择多个产品组合解决问题，是合理选择。企业所使用的监控工具种类多数量多、整合收敛速度慢，在很长一段时间内仍然是用户不得不面对的现实问题。



Grafana 提出了 “Big Tent（大帐篷）” 理念，强调 Grafana 不试图取代已有的监控或可观测性系统，而是欢迎所有数据源加入一个共享的可视化和分析平台。它不会强迫用户只用自家产品，而是允许接入其他厂商的系统，如 Datadog、New Relic、Splunk、CloudWatch、Azure Monitor 等，具体表现为：

- ✓ 任何数据源都能接入
- ✓ 任何存储后端都能展示
- ✓ 用户自由组合最佳组件

在快猫星云 Flashcat 的产品设计过程中，也提出了类似的理念，称之为「插线板」。在建设可观测性平台的过程中，先不急于替换已有的工具，而是先把他们连接起来，「插线板」可以快速对接已有的可观测性工具，把原先分散的数据孤岛连接到 Flashcat 的全栈可观测平台上来，把已有的数据利用好。当把某个已有的数据源插到「插线板」上之后，用户就能在 Flashcat 这一个控制台上完成数据关联、检索、可视化、监控告警、故障分析的工作。“黑猫白猫，捉住老鼠就是好猫”，只要工程师和用户用的高效、分析的快，「插线板」的目的就达成了。此外，AI 进一步放大了“连接数据”的需求和迫切性，在「插线板」连接的数据基础上，基于大语言模型技术和智能体架构，驱动故障发现、根因分析与事件响应，能够显著释放可观测性平台的价值。



与「插线板」相对应的，「全家桶」是面向未来的设计方案。对于增量的业务和场景，则推荐全家桶，使用 Categraf + Otel-collector 来收集各种维度的可观测性数据，在数据采集的阶段完成数据标准化、数据治理、关联（correlation），数据格式遵循 OpenTelemetry 标准和协议，避免厂商和工具锁定。

2026 年，AI 的竞争来到了 Agent 层面，参与的玩家来自各个领域，数量很多，将会更激烈。模型层面的玩家，进行的是排位赛，新晋玩家已经失去了入围的机会。和 AI 的交互，不再局限于“聊天框”，虽然过去一年，这个聊天框已经被扩展到了“多模态”，能听说读写看，26 年将会看到更多赋予模型“执行力”的场景出现，对各个领域产生实实在在的影响。

过去的各种 App、网站的设计，更多着眼点于 UI 和交互上，是给人类看和使用的，这给 AI 带来了一些困难，因为大模型在处理文本信息，效率是更高的，要像人一样去“看” App 和网站界面，不是最高效的，更别说去“用”了。接下来，面向 Agent 设计系统，API-First 将变得更重要，App 和网站应用，所有数据都可以通过 API 访问，所有功能都可以通过 API 操作。API 返回的数据，结构化、自描述、高信息密度，以便于 AI 准确高效理解。

在可观测性领域，可以预期，SRE 工程师与 AI SRE 协同工作，将成为众多企业运维团队标准构成。在专业知识层面，AI 已经远超过了人类工程师，就拿大家诟病的「大模型幻觉」问题，实际上人类工程师的幻觉问题也不遑多让。 要让 AI SRE 成为值得信赖的团队同事，第一个关键点在于让大模型充分、准确、高时效性的掌握企业内部的 IT 环境信息，包括：部署架构、服务依赖、网络拓扑、服务所关联的监控数据位置、K8s 元信息、CICD 流水线、代码仓库、运维操作 SOP 等。第二个关键在于「受控的安全性」，即赋予 AI SRE Agent 明确的权限边界，哪些操作可以由 AI 执行，哪些操作必须取得用户 Approve，哪些操作则明确杜绝执行。

举个例子，在故障应急响应场景下，AI SRE 将主导整个工作，并显著加速分析处置过程：

- ✧ 智能触发：故障发生时 AI SRE 自动响应
- ✧ 知识准备：检索知识图谱获取环境信息等上下文
- ✧ 策略规划：AI 制定分析策略（Planning 阶段）
- ✧ 并行分析：同时查询指标、日志、链路、变更等数据
- ✧ 综合分析：AI 推理根本原因
- ✧ 报告生成：生成结构化报告
- ✧ 实时通知：推送至作战室，同步发布更新 Status Page
- ✧ 用户确认：获取用户许可后进行修复
- ✧ 自动修复：生成代码并提交 PR、自动扩缩容、切换失效组件等

场景篇

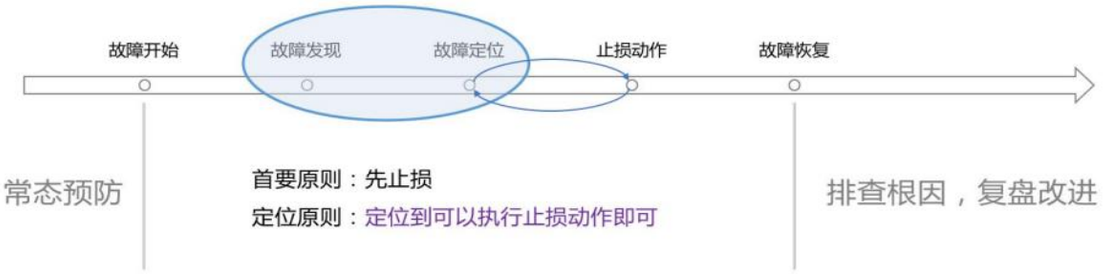
一款好用的产品，一定不只是产品，而是方法论结合场景的产品化落地实现。如果没有方法论作为后盾，即使拿到金玉良材，也很难锻造出匠心之作。

服务稳定性保障是可观测性产品面向的主要领域，而故障处理又是稳定性保障中的核心场景。在稳定性保障体系的建设中，有哪些方法论，如何落地这些方法论，是本章节探讨的核心内容。本章节将以故障处理为核心，介绍一种服务稳定

性保障体系建设的思路和方法论。

故障生命周期

保障稳定性，实际就是和故障战斗的过程。梳理故障的全生命周期，对我们做这个事情意义重大。在生命周期的各个阶段，分别应该做哪些事情让故障快速终止，甚至扼杀在摇篮之中？如果把这些事情做好了，故障的影响就可以降低，下面是典型的故障生命周期图：



稳定性保障是以故障处理为中心，分为**常态预防**、**发现处理**、**复盘改进**三大部分，故障出现的频率小（常态预防）、出现后处理快（发现处理），则稳定性保障能力就强，反之稳定性保障能力就弱。

常态预防，是稳定性保障的重点，让故障不出现是更值得追求的目标。相关的工作有很多，如压测、高可用架构、发布控制、服务巡检、风险量化等等。

故障是不可能被 100%预防住的，可预期的停机或者意外的情况总会出现，因此发现处理能力也需要重点建设。而发现处理的原则是先止损后排查，以恢复服务和用户的核心体验为首要目标。故障一旦发生，要尽快发现，并且快速定位故障的直接原因，注意，这里说的是直接原因，不是根因，知道直接原因就可以止损了，根因可以留待后续排查，有的时候定位到某个直接原因并且执行了止损动作，但是故障没有恢复，说明找的原因不对，需要继续寻找，继续止损，这个动作可能循环往复几次才能最终止损；

故障恢复之后就是复盘了，大家一块贡献线索，梳理时间线，找到具体是哪个代码导致的，亦或者哪个操作导致，亦或是流程缺失、架构设计不合理等等，然后形成改进项，做针对性的改进。复盘改进可以让我们从故障中吸取经验和教训，并输出增强预防和发现处理能力的工作。

故障预防

这个阶段需要投入足够的力量，防患于未然才是最好的稳定性建设方法。但是很多企业在没有故障之前看不到技术团队在稳定性保障工作中所付出的努力，必须要有个故障造成舆论危机了才敲响警钟，实属认知问题，憾事也。

重视程度不够将导致稳定性工作难以做到位，缺乏长期规划和持续投资，所以，预防阶段第一个工作是：**让企业认识到稳定性工作的价值。**

> 稳定性工作的价值，体现为：保障客户体验、避免资损以及社会负面舆论等。客户使用某个产品，可能是因为产品功能设计得好，也可能是因为响应速度快、稳定可靠。以滴滴和 Uber 的竞争为例，滴滴出现故障，打车的用户就会切换到使用 Uber，给 Uber 送去大量的用户，此时对滴滴而言是损失用户、影响品牌、造成资金浪费。反之亦然。

假设企业已经认识到稳定性的价值了，请继续往下阅读本文。接下来先看在故障预防阶段，应该从哪些方面来着手建设呢？首先，要明确和解决「权责利」的问题。

确立稳定性一号位

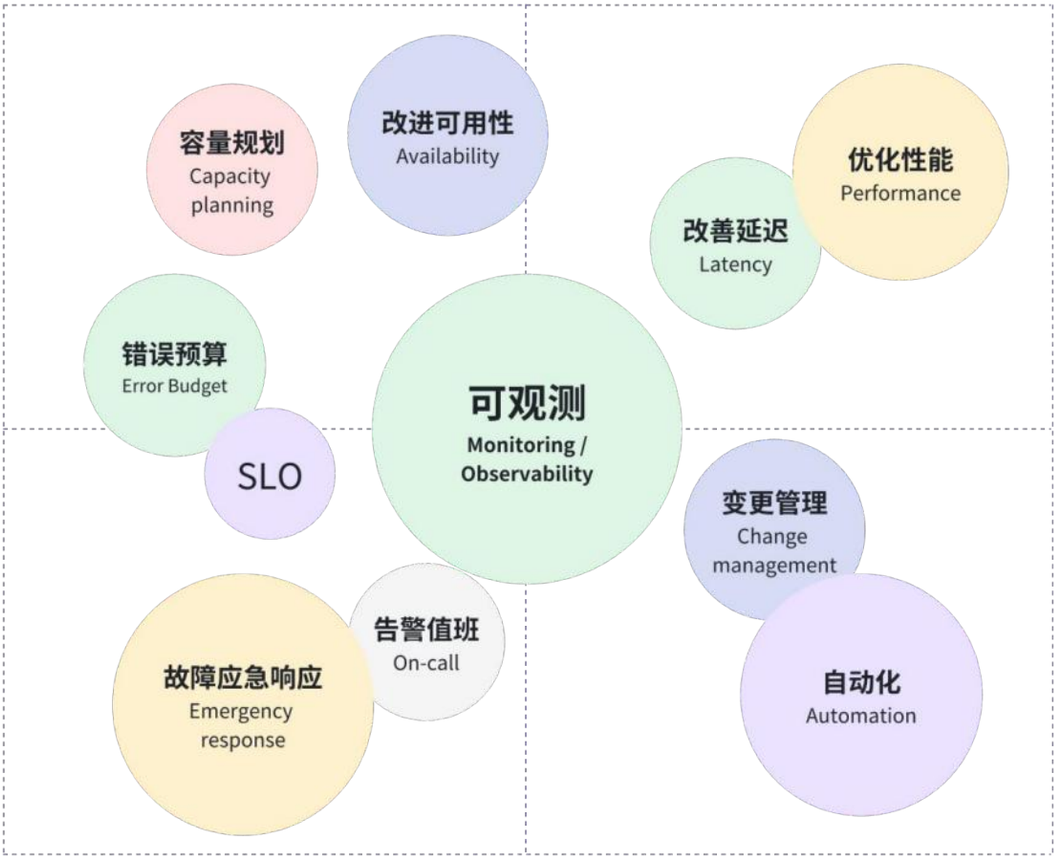
想象一个场景，老板收到了一个友人关于产品和服务的投诉，说下不了订单，老板会找谁来跟进？通常是两个选择，要么是 CTO/CIO，要么是产品线 GM（总经理）。这几个人通常也不会直接跟进稳定性工作，他们会继续找下面的人，最终跟进这个故障的，可能会有两个角色，一个是运维总监，一个是产品线的稳定性接口人。具体可以根据自己公司的情况来安排，但是责任到人，是必须的，要不然 CXO 和 GM 最后找谁要结果呢？所以结论来了，**稳定性工作需要有一号位。**

> 以 Google 为例，稳定性工作的牵头人，正是 SRE 的提出者和创建者，Google 工程 VP **Ben Sloss。**他有一句名言广为人知：“If Google ever stops working, it’s my fault.” 这句话正是体现了“稳定性保障一号位”的责任、权利和担当。

如果是公司有统一的运维团队，运维总监大概率要承担这个一号位的角色，如果没有统一的运维团队，每个业务线会有自己的稳定性牵头人负责自己的稳定性工作。但是造成故障的原因错综复杂，可能有基础网络的问题，有硬件的问题，有依赖的中间件和其他自研服务的问题。大故障的处理，通常要横跨业务线，所以，业务线的稳定性接口人通常不足以推动落地，通常来讲，**对于跨业务线的大故障，运维总监更适合牵头处理，拉通横向信息**。

运维总监来牵头，是否会有推动不了的事情呢？受限于企业的组织架构划分，显然没有那么容易。所以推荐的做法是：**建立稳定性专委会**，CTO/CIO 挂帅，运维总监日常实操，各个业务线出稳定性接口人予以配合，方能良好推进。而运维总监也不可能一个人搞定所有事情，所以，SRE 应运而生。

> SRE，由 Google 工程 VP **Ben Sloss** 在 2003 年提出并创建，20 年来，Google SRE 团队支撑着 Google 所有重要服务的可用性保障工作。SRE，既是一种职位，是一种思维方式，也是一组最佳实践，被全世界范围的众多企业采纳、应用和发展。



如上，有了稳定性一号位了，接下来要推进工作，在故障预防阶段可以做哪些事情呢？首先大家想到的，应该是架构优化吧。

架构优化

稳定性保障是对 IT 系统的稳定性保障，那如果这本来就是一架随时可能散架的破车，就算 SRE 有通天之能，也很难把这架破车开出布加迪威龙的效果。所以，IT 系统首先得有良好的鲁棒性设计，能否考虑各类异常情况，否则就只能通过外挂的手段来解决稳定性问题，付出的成本通常更为高昂。

首先，在哪个环节做架构优化？显然越前期越好，越前期成本越低，最好是研发设计系统的时候就考虑到。如果那个时候无法考虑到呢？那在系统上线之前，应该经过准入评审和准入测试。评审架构合理性，测试功能、性能、安全、稳定性。这里我们重点关注稳定性，要能良好支撑业务发展，这里通常可以引入几类软件产品：

- ✧ 软件运行环境：某个节点挂掉可以自动摘除，上面的应用可以自动迁移，流量可以跟随迁移，典型的方案是借助 Kubernetes，方便上层业务做高可用设计。
- ✧ 各类组件 PaaS：比如程序依赖的 MySQL、Redis、Kafka、ElasticSearch、分布式存储、分布式锁服务等等，这些基础组件就像撑起建筑物的关键支柱，只有这些基础组件是稳定的，上层业务程序才能起万丈高楼。
- ✧ 混沌工程系统：需要灌输给业务研发一个理念，就是他依赖的基础设施以及服务是不稳定的，这样才能倒推研发

设计鲁棒性高的系统，混沌工程，或者传统的故障演练，就是反向推动达成这个目标的。

- ✧ 全链路压测：这个产品通常是大厂必备，一个业务可能涉及很多系统、子系统、模块，通过全链路压测，可以更好的知道瓶颈在哪里，更好的应对大促和流量高峰。

延展一下这个话题，如果架构短时间做得不好，应该怎么办？有没有什么机制让研发团队更有动力做这个改进？显然最简单的办法就是在上线这个环节实施准入制度，设计层面有严重稳定性缺陷的系统不建议发布。那如果因为某些不可抗拒力的要求必须上线呢？可以请业务线负责人审批，承担这个稳定性风险。相关服务的 On-call 由研发团队主责，运维团队辅助。等到架构改好了，再由运维人员主责，研发辅助。

如果基础设施都做得挺好了，软件架构也设计得挺好了，是否就万事大吉了呢？因为线上问题五花八门，防不胜防，建设一套完备的可观测性体系显然是必需的。在预防阶段，需要做好两个事情，一个是埋点采集数据，另一个是组织数据，便于后续排障。在预防这一小节，会重点讲解一下数据采集，数据如何组织也非常关键，是要做在事前的，但是如何组织数据是需要契合故障发现、定位时的视角，会在 Flashcat 的实现章节再做介绍。

可观测性数据采集

可观测性的理念和监控有差别，偏重分析，相关数据需要应采尽采，因为我们无法预知线上会出现哪些问题，所以要尽量完备的可观测性数据，出了问题才能排查，当然，应采尽采又会造成资源浪费，需要靠经验做折中。可观测性数据，通常分四类：指标、链路、日志、事件。

指标

指标数据存储成本最小，应用最为广泛，Open-Falcon、Prometheus、Victoriametrics、夜莺都是专门处理指标的监控系统，OpenTSDB、VictoriaMetrics、M3DB、Thanos 都是专门存储指标的时序库。操作系统、网络设备、各类中间件、数据库、应用、业务，所有监控目标，都需要采集指标，常见的采集器有 Telegraf、Categraf、以及 Prometheus 生态的各类 Exporter。

很多指标是可以在运维侧搞定的，但是有些指标必须要研发侧埋点，比如应用层面的指标和业务层面的指标。Google 有一个规范，值得大家借鉴，他们要求所有的服务模块都要监听一个 HTTP 端口，暴露一个 的接口，请求这个接口，就可以返回这个服务的各种 metrics 数据。Prometheus 的数据拉群接口规范，也是类似的做法。

Prometheus 非常成功的地方，就是推广了这个监控指标的数据格式和采集方式，可以在公司内要求各个业务都遵从这个格式导出监控数据，如果是第三方采购的软件，也要求供应商的各个服务模块暴露 Prometheus 协议的监控数据，这样指标层面就规范了，而统一规范化是自动化的基础。

另外，说起指标监控，很多人肯定会想到 Zabbix，随着 Kubernetes 和微服务的普及，基础设施越来越动态，指标数量越来越多，如果现阶段做选型，建议选择 Prometheus 生态的产品，Zabbix 是资产管理式的，更适合监控传统的物理机、虚拟机。在 Kubernetes 和微服务的时代，首先考虑的应该是 Prometheus、VictoriaMetrics、Nightingale 夜莺之类的产品。

从传统的指标监控到可观测，发生了哪些变化？

1. 数据量十倍提升，基础设施层面的监控数据比重变小，应用层面的监控数据比重快速增加。
2. 监控数据的采集原则发生了变化，数据应收尽收，治理前置成为指导原则。
3. 监控数据模型的维度变得更丰富。
4. 监控数据的生命周期变得更短、更不确定。
5. 针对监控数据的 Ad Hoc 查询需求变大。
6. Metrics 与 Logs、Traces 的融会贯通重要性凸显。
7. 使用对象和使用习惯发生了重大变化。
8. 监控系统本身也要云原生。

链路

链路数据对于微服务数量比较多的公司有很大用处。服务数量多，关系错综复杂，用户在最前端操作失败，可能是某个隐藏在角落里的服务故障导致的，很难排查，此时需要引入分布式链路追踪的系统。

常见的开源解决方案有 Zipkin、Jaeger、Skywalking，当下，业界也推出了 OpenTelemetry 的规范和工具集，采集侧基本已经没有太大差异了，而且可以相互复用，使用开源的链路追踪系统基本可以解决链路监控的需求。但是，要推动链路追踪落地，需要所有的模块都接入才有价值，更多的是推动上的阻碍而非技术上的阻碍（当然 OpenTelemetry 项目的

一个重要目标就是 auto instrumentation，降低埋点的成本）。

日志

日志是最常用、最自然的监控数据类型之一，具有以下的优点：

- ✧ 日志的内容比指标更加丰富，可以提供更多的细节信息，帮助开发人员和运维人员更好地理解应用程序的运行状况，通过日志几乎可以重放、还原系统的完整工作过程；
- ✧ 日志的格式灵活，可以方便的记录多样化的信息，包括错误、异常和警告等，而指标通常只能提供统计数据，无法直接反映系统中的具体事件；
- ✧ 日志为文本格式，便于技术人员理解，同时可以被各种文本处理工具、文本搜索工具高效的处理；

日志数据是排查问题的重要手段，但是日志量大，存储成本高，我们需要管理得更为精细化一些。比如，比较久远的数据进对象存储。比较廉价，几乎没有查询需求，近期数据，比如最近 3 天的数据，进 Elasticsearch，用于搜索，排查问题。日志里一些数值类数据提取为指标，存入时序库，可以保存一年半载。

> 针对指标数据，推荐保存时长为 13 个月，以便技术团队可以检索去年同期的历史数据。

排查问题的时候，通常是指标有异常先被发现，然后查看相关时间段的日志，日志里可能有 traceid，再根据 traceid 查询链路数据，以此就可以把三类可观测性数据串联起来，帮助我们更快地找到故障直接原因。

事件

事件数据通常包含两类，一个是告警事件，一个是变更事件。统计数据显示，线上故障 70%都是变更导致的，出现故障之后找到临近时间的相关变更事件就是重要的排查手段之一。所以，从故障定位角度，告警事件、变更事件都应该统一收集到中心，从时间维度、相关性维度做关联分析，有助于快速定位问题。这样说比较抽象，举个例子：

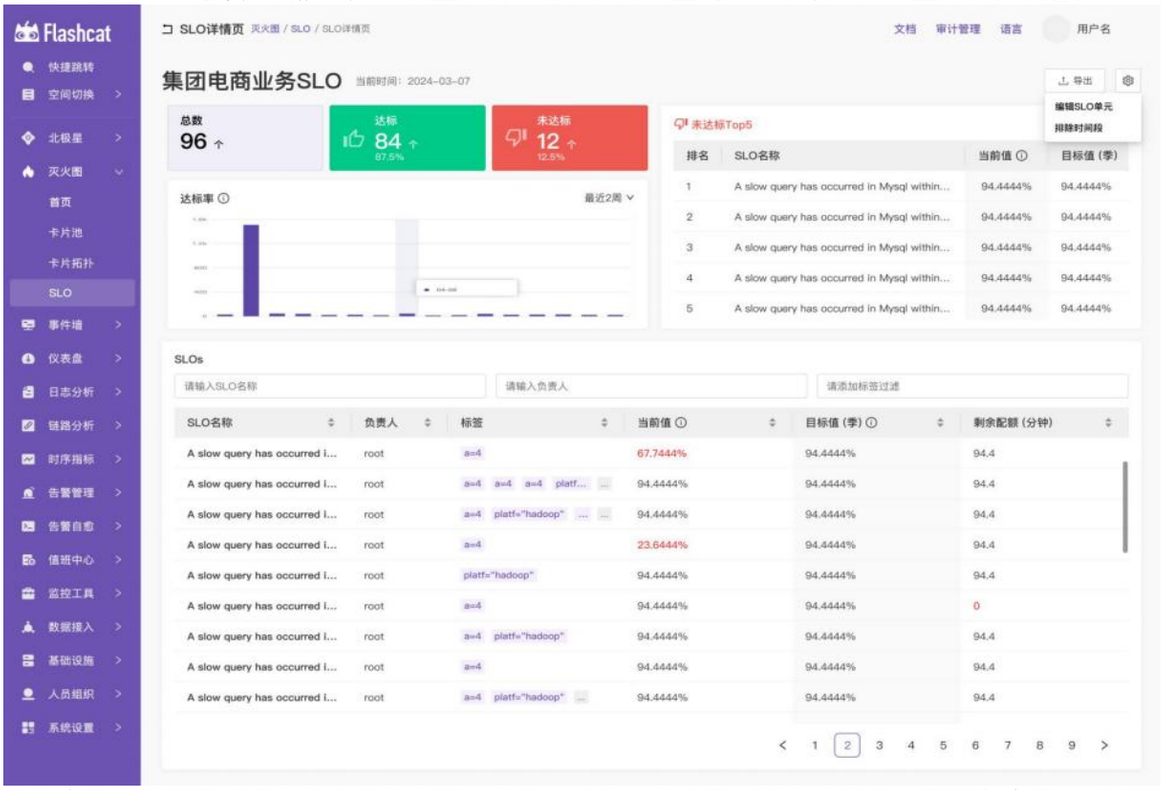
上图中的“北极星事件”指的是业务指标告警，Jenkins 则是变更系统的事件（当然，一个公司可能有很多变更系统，配置变更、服务变更、数据库变更、基础网络变更等等，相关的事件都要进事件墙，一些重要的运营事件也要进事件墙，数据越完备效果越好）。这个图只是个演示，便于我们理解其设计逻辑。中间那个纵向竖线，可以左右拖动，方便我们观察某个北极星指标告警的时刻，就近的变更有哪些，而这些变更很可能就是罪魁祸首，尽快回滚就可以止损。

观测体系建立之后，需要做好日常巡检，用程序和人工双重保证，如果程序巡检做得好，人工就省点力，如果程序巡检做得不太完备，人工就要多费劲。但是可观测性数据何止千万，人工如何巡检得过来？这就需要依托良好的数据组织方式和更智能的巡检机制了，在后文展开。

建立风险量化体系

可观测性体系建立了，紧接着就是风险量化体系，通过数据驱动的方式，来分析评价可观测性体系建设得如何，是否完备，比如应该采集的数据是否采集了，应该配置的告警规则是否配置了，进而提前发现潜在风险和隐患。

当然，除了量化观测体系，还可以量化变更体系，通过数据分析各个团队的变更操作是否值得信赖，是否经常不按规矩出牌。比如某个团队经常不做灰度直接全量上线，经常高峰期上线，经常回滚，那这个团队的变更质量就很令人担忧了。需要为各个团队量化出一个健康分或信用分，以业务线红黑榜的方式呈现，推动做得差的业务线去改进。



以上就是故障预防阶段需要重点投入的方向了，当然，还有一个也是要做在事前的，是预案建设，不过预案是在止损阶段发挥作用的，留待后面再做介绍。

故障发现

这个阶段核心要解决的问题就是及时发现故障，通常遇到两个问题：

- ✧ 告警很多，不知道哪些更重要，每天都被海量的告警淹没，狼来了太多，人疲惫不堪，对告警的关注度下降；
- ✧ 告警响应处理流程不清晰，时效性无法保证，协同混乱，经常等到客户投诉了，甚至老板都问询了，技术人员才知道现在有故障；

怎么解决上述问题？一方面我们要回归本质，**IT 系统是为业务服务的，如果有指标能衡量业务是否健康就好了**，这类指标通常不会很多，但是每一个都很关键，只要这类指标出问题，一定是代表有真故障。有没有这类指标？显然是有的，我们称之为**北极星指标**。另一方面要在告警 on-call 的流程上下功夫，用工具提效，用数据驱动的方式，不断优化告警发现和响应。

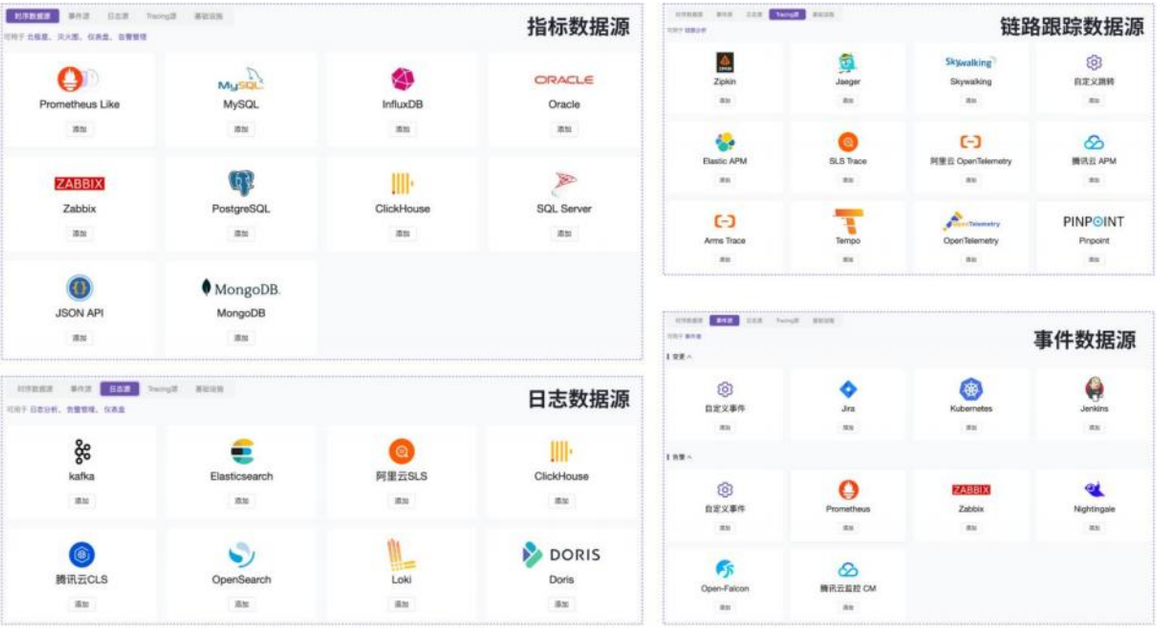
北极星

北极星指标通常都是业务关键指标，比如电商类型业务，北极星指标可能是订单量、支付量、加购物车的量；比如打车类型业务，北极星指标可能是呼叫量、应答量、在线司机数、支付量等；再比如内部 OA 系统，北极星指标则可能是流程发起量、工单数量、实时登录量等等。此类指标，通常老板和业务方都会很关注，老板和业务可不会关注什么机器 CPU 飙高了，磁盘用满了，数据库连接数打爆了之类的指标，业务关注的是**影响用户主流程的指标**，这类指标出问题，基本就意味着用户体验受损，或造成直接资损。

我们需要为北极星指标构建专门的系统，进行重点保障，如何重保？这个系统和普通的监控系统相比，有哪些不同？下面我们以快猫星云的北极星系统（为北极星指标专门构建的系统）来说明整个逻辑。

多数据源支持

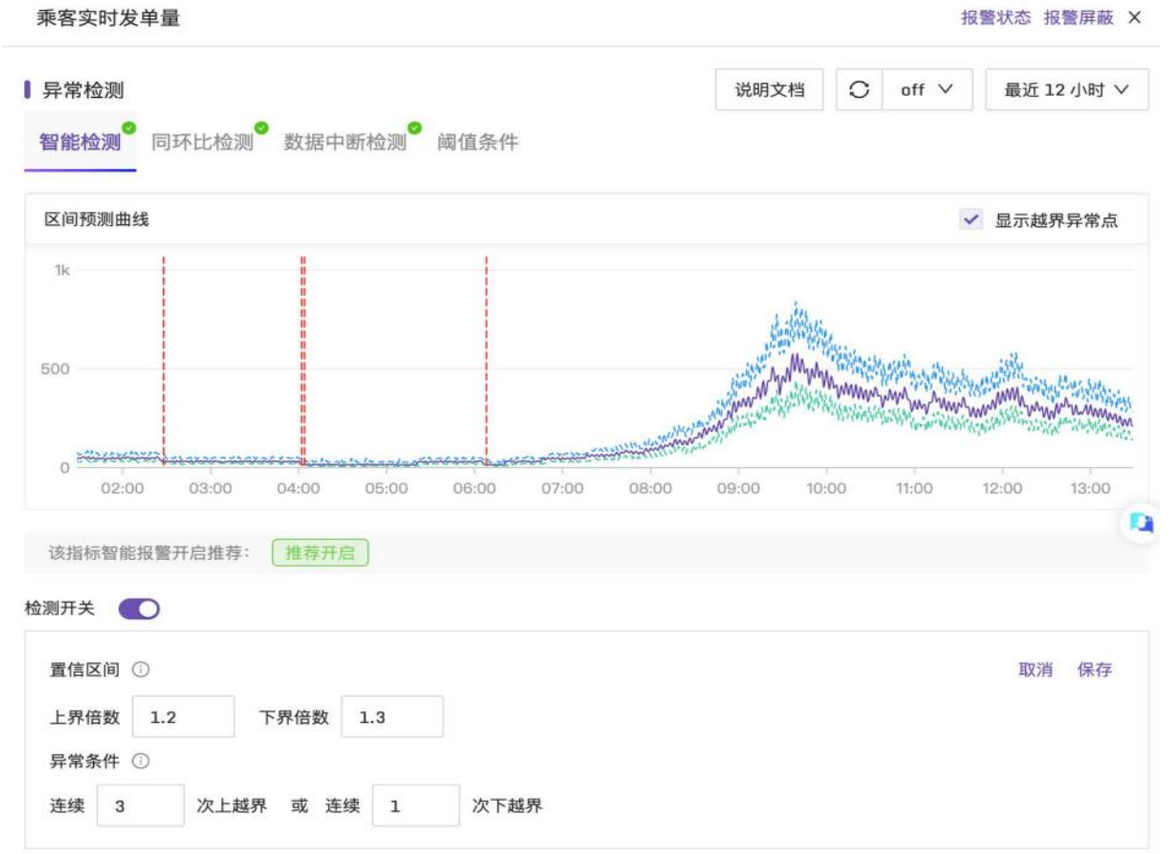
让北极星指标的创建更快捷易得。因为北极星指标大都是业务指标，不只是来自时序库，有可能来自日志、来自某些 OLAP 系统或者 OLTP 系统，所以北极星需要多种数据源的接入，方便用户以页面配置的方式，快捷的从不同的数据源生成业务指标数据。如果构建一个北极星指标，需要协调研发人员写代码和上线才能实现，那么门槛就太高了。



多种告警规则支持 准确实时

北极星指标很重要，出现问题需要及时知道，还不能误报，否则大家就不相信系统了。需要同时支持智能检测算法、同环比检测、数据中断检测、阈值条件告警等多种方式。其中智能检测算法对于有规律的业务指标尤为合适，不用配置阈

值，自动发现异常数据。



大屏支持

因为北极星大都是关键业务指标，所以老板和业务方很关注，研发、运维人员也很关注，以大屏形式投到电视上，就变成了一个强需求，快猫星云的产品 Flashcat 提供了几种默认模板的大屏，供参考：

实践值班 On-call

除过抽象北极星指标之外，如果你的技术团队面临以下一个或多个困扰，那么推荐在 On-call 维度上进行优化：

- ✧ 技术团队每天接收到大量的告警；
- ✧ 很多告警长时间无响应，长期无人问津；
- ✧ 告警与告警之间缺乏关联性，处理效率低下；
- ✧ 告警处理缺乏协同，处理过程不透明，信息难以共享，知识难以沉淀；
- ✧ 很多告警并未准确反应实际情况，无谓的消耗技术团队精力；
- ✧ 客户/用户往往先于技术团队发现故障，客户满意度持续走低；
- ✧ 无法量化的衡量应急响应的现状和效率，无法制定出改进和优化路线；

On-call 系统应该着重具备以下基础能力

- ✧ 告警聚合收敛：解决告警风暴问题；
- ✧ 告警认领、转派、升级：解决告警不能及时处理、告警漏处理、告警散落在各个监控系统的问题；
- ✧ 故障管理：解决跨团队的大故障协同不畅的问题；
- ✧ 多种触达方式：在电话、短信之外，提供各种 IM 触达渠道，提供手机 H5 操作方式，大幅提升值班人员的幸福感；

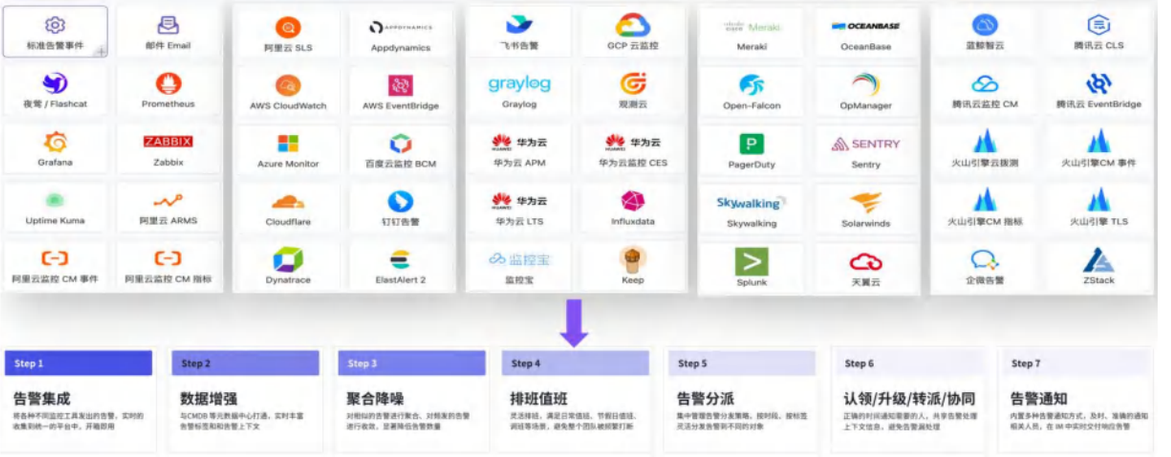
On-call 度量指标

没有度量，就无法改进。通过对告警数据全面分析，分析 MTTA、MTTR、告警数量变化趋势等数据，推动告警治理工作，优化告警响应流程和效果。以下是推荐关注的关键量化指标：

- ✧ **降噪比**：即告警的压缩比，通过算法、规则将众多相关的告警聚合后，再通知到值班人员。告警聚合能有效降低告警风暴，减少值班人员的工作量，提高信息处理的效率（该指标越高越好）。
- ✧ **响应比**：被认领的告警占所有告警的比例。在告警管理领域，需要响应或者认领的告警，才是有用的告警，因此通过统计和观察”响应比“，能整体的评估告警是否足够有效和有用，并持续的推动提升告警”响应比“（该指标越高越好）。
- ✧ **告警总量**：一段时间窗口内产生的告警数量。过高的告警总量，意味着值班的压力越大，对技术团队注意力的干扰越多，潜在的意味着告警的噪音可能也过大，因此过多的告警，会让整个系统处于不可运维的状态，应该该尽力的降低告警总量，譬如采用基于 SLO 的告警，就可以答复降低该指标（该指标越低越好）。
- ✧ **MTTA**（平均响应或认领用时）：从告警发生到值班人员响应或者认领的时间间隔。越快的 MTTA，标志着越高的告警处理效率，潜在的代表着越高的服务稳定性。通过 MTTA 我们可以有效的度量团队的工作压力，以便决策合适的资源投入，确保团队始终处于可持续发展的状态（该指标合适就好）。

- ✧ MTTR（平均恢复或解决用时）：从告警发生到问题解决的时间间隔。越快的 MTTR，往往意味着团队拥有更先进的观测技术、更强大的基础设施平台、更熟练的工作技能、以及对业务系统有更深入的理解（该指标越快越好）。

以快猫提供的 On-call 产品 Flashduty 举例，支持告警聚合、降噪、认领、升级、排班，让告警的触达既高效，又确保告警处理不遗漏、做到件件有回响。同时提供了降噪比、响应比、MTTA、MTTR 等各项关键指标在不同维度上的统计报表。



On-call 全流程

1. 告警集成

Flashduty 作为一站式的告警事件响应平台，其本身不产生告警数据，而是将第三方监控平台的告警事件接入到 Flashduty，以实现统一告警接收、降噪、分派、触达、解决、分析和自动化等，如夜莺/Flashcat、Zabbix、Promethues 等监控平台以及各大云厂商的告警，目前以支持近百种第三方监控平台。

2. 数据增强

Flashduty 故障详情中的故障标签以 key:value 的形式展示告警的各类源数据信息，这些标签信息来源于接入到 Flashduty 的各类告警事件，当系统以源数据自动生成的标签无法满足业务时，可以通过标签增强为告警丰富更多标签信息，提升故障处理效率。常见的数据增强方式如下：

- ✧ 提取：可以在告警标题、详细描述及现有标签字段中，运用正则表达式提取出所需要的信息，并自动生成附加的标签；
- ✧ 组合：组合规则可以通过 Go 语言模板语法构建新的标签，以{{.Labels.Field}}格式来提取标签值或采用固定值的方式生成新的标签；
- ✧ 映射：是将系统中的源键值通过映射关系生成新的键值对，需预先创建映射关系。映射关系的数据来源通常为 CMDB 或者其他元数据中心。
- ✧ 删除： 即删除指定名称的标签，如果删除的标签不存在则无效；

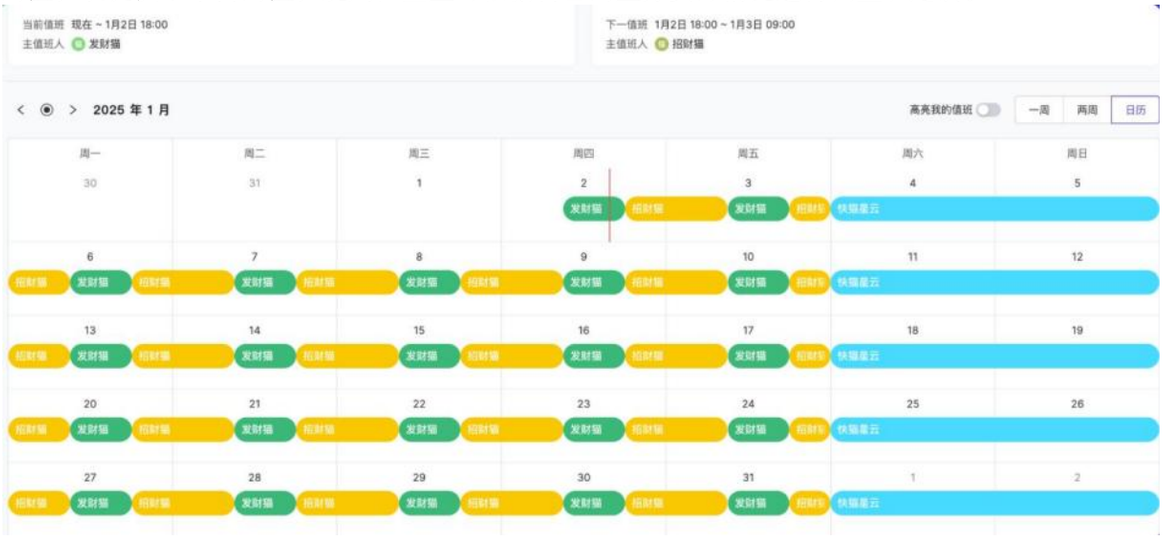
3. 聚合降噪

当 Flashduty 接收到告警事件（比如 Prometheus 的一条告警通知），系统会自动触发一条告警，而这条告警将会触发一条故障。多条相似的活跃告警，可能会被聚合到同一条故障中，一起分派、通知和处理，这可以显著降低通知频次并提高处置效率。降噪模型中的基本概念包括：

- ✧ 事件（Event）：来自于原始监控系统（比如 Zabbix），每一次触发和恢复通知都是一次事件；
- ✧ 告警（Alert）：由事件自动触发，原始监控系统中同一条告警在不同时刻发生的事件，将被 Flashduty 合入到同一条告警中；
- ✧ 故障（Incident）：Flashduty 平台处理的主要对象，一般由告警自动触发，也可以手动创建。故障可以理解为相似告警的组合，Flashduty 依据用户配置的聚合策略，将相似的告警自动聚合到一起进行分派和处理；

4. 排班值班

值班规则是企业内部管理方式的核心之一，可帮助企业合理安排员工的工作时间，确保业务的连续性和高效性。FlashDuty 的值班表提供了丰富的值班规则，企业可根据自己的业务实际情况进行调整和定制。

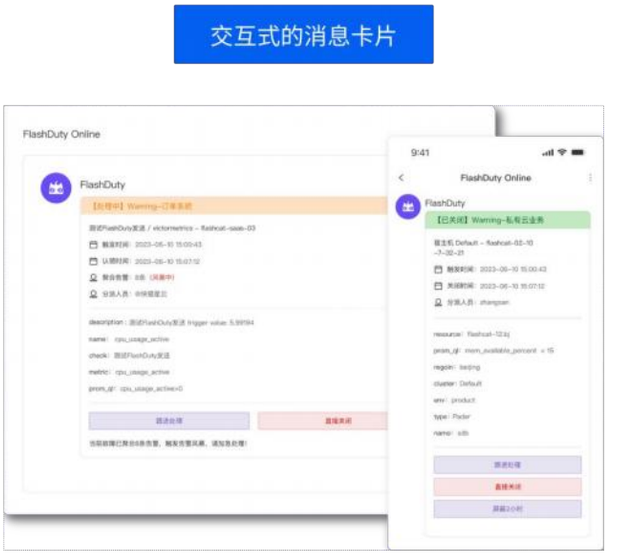


用户可以在值班管理页可以看到自己所有值班的汇总情况。值班会根据预先定义好的值班班次自动轮换，特别的当发生班次轮换的时候，可以以多种方式提前通知下一值班人：

- ✧ 提前通知：班次交接时，提前 N 分钟进行通知到下一值班人；
- ✧ 定时通知：固定每天某个时间点进行通知一次值班信息；
- ✧ 通知渠道：单聊（一对一进行通知，即短信、语音、邮件等方式）、群聊（推送到 IM 消息群，并对分派人员进行 @ 提醒）；

5. 告警分派/认领/升级/转派

6. 告警通知



7. 状态页

服务中断不可避免，快速应急处置固然重要，但是如果能与用户及管理层保持及时顺畅的故障信息同步，往往能大幅降低故障的「伤害值」。当告警响起，你的团队是否还在群聊、邮件和电话之间手忙脚乱，疲于应对来自内外的反复询问？让本已疯狂救火的工程师，停下来不断回复问题，可不是个划算的事情。

Flashduty 作为一站式告警事件响应平台，是故障应急响应的入口，推出的「状态页（Status Page）」 功能，正是为了解决故障状态发布麻烦、信息同步不及时的问题。它是一个面向用户、客户和合作方的统一服务状态展示页面，用于实时发布系统运行状态、故障通告和恢复进展。

Status Page 的核心价值在于：

- ✧ 一键同步信息，解放工程师：在争分夺秒的故障处理窗口，状态页支持“一次更新，多方同步”。无论是面向客户的公开页还是内部团队使用的内部页，信息都能实时触达。这让工程师能从重复的解释工作中抽身，专注于定位和修复问题本身。
- ✧ 透明沟通，化被动为主动赢得信任：主动、清晰地告知“发生了什么、我们正在做什么”，远胜于沉默或遮掩。状态页通过主动公布故障状态与维护计划，在不确定性中展现专业与掌控力，是维系客户与合作伙伴长期信任的关键。
- ✧ 沉淀资产，量化稳定性：每一次状态变更都会被记录，并生成可视化的服务可用性统计与历史事件归档。这让你对 SLA 的承诺，拥有了客观、可验证的数据支撑，将稳定性真正转化为可展示的运营资产。

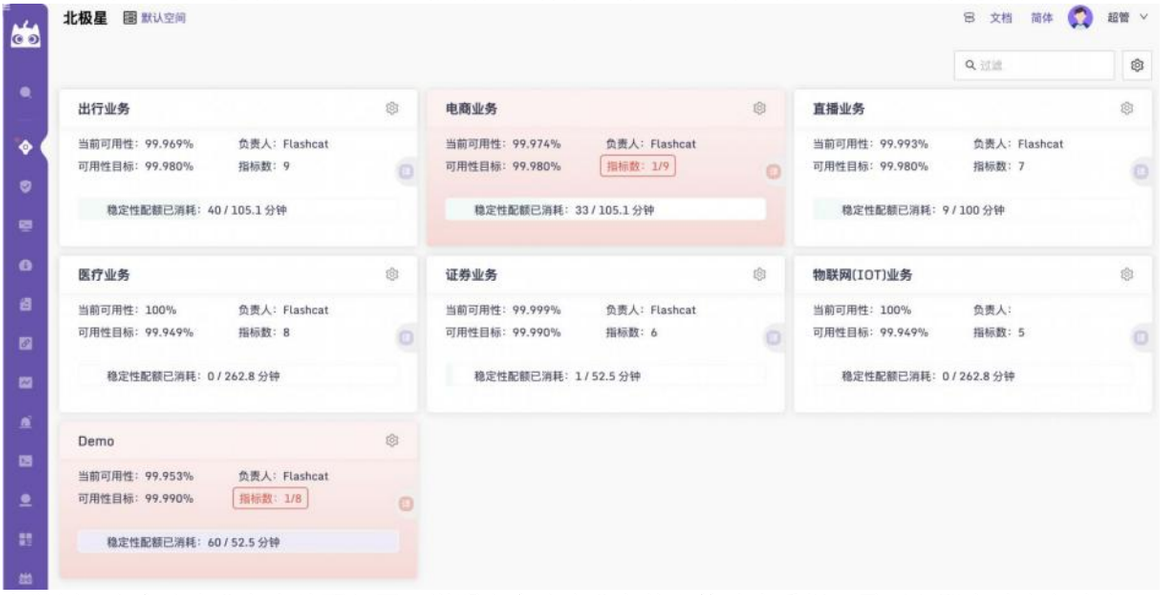
简单来说：采用 Flashduty Status Page 功能，在故障发生时，为你提供了一个统一、权威的信息发布源。对内，它让销售、客服、PR、管理层与工程师信息同频；对外，它主动管理客户预期，打造可靠专业的品牌形象。

8. 统计报表

基于统计报表，通过数据驱动，不断推动告警治理，推动 on-call 流程优化。

建立全局视角的业务稳定性视图

虽然已经把各类北极星指标创建好了，但是没有一个良好的层次化的组织方式，就会特别散乱。快猫星云 Flashcat 的做法是对北极星指标做了三个层级的划分：空间-业务-指标。空间通常用于划分生产、测试环境，或用于划分完全无关的 BU，业务在系统里体现为一张卡片，业务可能包含多个北极星指标，点击业务卡片进去，可以看到详细的指标，任何一个指标有异常，上层卡片就会飘红，这样就可以一目了然的知道各个业务是否正常，建立了全局业务稳定性视图。



以上图来举例，电商这个业务卡片飘红了，说明电商这个业务核心体验有受损，需要立即启动应急响应。

建立业务和技术的稳定性对话基础

上面的图中也有 SLO (Service Level Objective) 的体现，SLO 是 SRE 体系中的最重要的概念之一，国外不少公司效仿 Google 的最佳实践落地 SLO，很多服务商也支持了 SLO（如 Datadog），甚至有创业公司专门聚焦 SLO 的产品化方案。

北极星系统，本质上是对 SLO 概念的扩展，从 Service 层面延展到了业务层面。比如，北极星指标就是一种 SLI，稳定性配额对应着 Error budget，当北极星指标发生异常波动，突破了 SLO，就会消耗 Error budget。北极星系统把这些 SLI / SLO / Error budget 数据管理起来并实时度量，明确各个业务的 SLO 是多少，当下已经消耗了多少配额，这个有什么用呢?

显然，可以作为技术团队稳定性保障工作效果的考核依据。但是更关键的，是作为业务和技术团队的对话基础，业务方对很多技术术语不了解，但是他们了解这些业务层面的 SLI 和 SLO，而且参与制定并最终拍板。像上图的例子，Demo 这个业务的稳定性配额已消耗殆尽，这就说明：这个业务的用户经常受到稳定性方面的困扰，在稳定性方面的体验不佳，可能要投入更多人力来做稳定性，而不是投入更多人力来开发新功能，尽量减少上线次数来提升稳定性。

此外，基于 SLO（Service Level Objective）能够提升告警的效率，能够有效解决以下痛点问题：

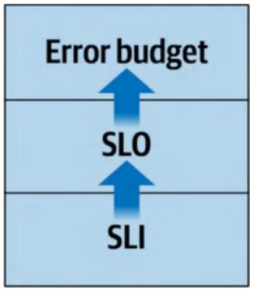
- ✧ 告警越加越多，却永远不够，告警风暴让团队疲于应付。
- ✧ 基于固定阈值的告警，在负载变化多端的现代架构下，很多都是防御性的提醒，效率不高，久而久之团队对告警免疫，进而麻木。
- ✧ 告警发生后，无法清晰的判断出对业务的影响面。
- ✧ 关于稳定性保障应该投入多少资源才合适，无法量化的决策。

一个告警是否是“**有用**”，一取决于是否反应了对用户体验的影响程度，二取决于是否需要有人去响应并采取行动。“无用”的告警，持续消耗着技术团队的精力和注意力，也会造成狼来了效应，让技术团队产生麻痹心理，渐渐地对告警失去了敏感度。同样一个系统是否“稳定”，取决于从用户的视角出发评判对于服务的满意度。SLO 是技术团队从用户视角出发，来衡量服务健康状态的一种手段，也是构建面向外部的用户协议或可用性承诺(SLA)的关键部分。SLO 的设定相比 SLA 更严格，通过增加这种严格性，技术团队拥有了一个缓冲带，在 SLA 不达标之前识别和修复技术问题。

以下是从用户视角衡量系统的稳定性的几个例子：

1. 对于用户的访问请求，我们能多快地向用户提供正确的响应?
2. 用户可以连接到我们的服务吗?
3. 我们的客户端应用程序能否快速、正确地呈现和渲染内容吗?
4. 对于关键的数据，我们能否快速、正确地处理吗?

关于 SLO，Google SRE 在其实践中，有三个核心概念：



选择合适的 SLI（服务水平指标)是基础。SLI 是指在一个时间窗口内，通过对一个 metric 背后所代表的行为“好与坏”的判断，从而计算出表示服务健康状态的一个百分比。比如我们要衡量过去 5 分钟内用户访问 web 页面的速度如何，就可以选择一个 的指标，如果 就是“好”， 反之则是“坏”，那么该 5 分钟内，“好”的次数 / 总的次数，就称之为 SLI。

一些典型的 SLI 举例如下：

1. 每 5 分钟窗口内，P99 延迟小于 300ms 的请求数量 / 总的请求数量。
2. 每 5 分钟窗口内的所有 http 请求中，响应码为 200 的请求数量 / 总的请求数量。
3. 每 1 分钟窗口内成功下单的订单量 / 总的下单数。
4. 每 1 分钟窗口内在线的用户数 / 昨天同期在线的用户数。

给定一个时间周期，比如 1 个月，这个时间周期内为 SLI 设置的要达到的目标，称之为 SLO。 SLI 和 SLO 之间的差距越大，Error budget 就越少。如果 Error budget 消耗速度过快，也就是 说 SLI 和 SLO 差距越大，触发告警的必要性就越高，告警的重要性和紧急程度也越高。

我们推荐使用 SLO 来衡量系统的稳定性，使用基于 SLO 的告警作为提升告警效率、解决告警风暴的核心抓手。SLO 告警，起到了提纲挈领的作用，当 SLO 告警发生后，一定代表着核心用户体验有受损，那么技术团队应该立即响应，也可以量化服务受损的程度。反过来，如果 SLO 告警静悄悄，那表明我们的系统和服务是能满足用户要求的，技术团队可以放心、专心的投入精力到产品功能开发工作中去。比如快猫星云开发的 Flashat 平台，就落地了针对业务关键指标、系统核心指标的采集和量化体系，当这些指标偏离目标时，会第一时间被 Flashcat 内置的智能算法检测到，并通知技术团队启动故障应急响应，把问题发现的确定性、时效性提升到了一个新的台阶。

故障定位

北极星是全局业务稳定性视图，我们可以知道某个业务的某个指标出问题，比如电商业务订单量下跌，但是无法知道具体是哪个基础设施、哪个系统、哪个子系统、哪个模块、哪个接口的问题，因为订单量这个指标，可能跟很多技术系统相关，比如：

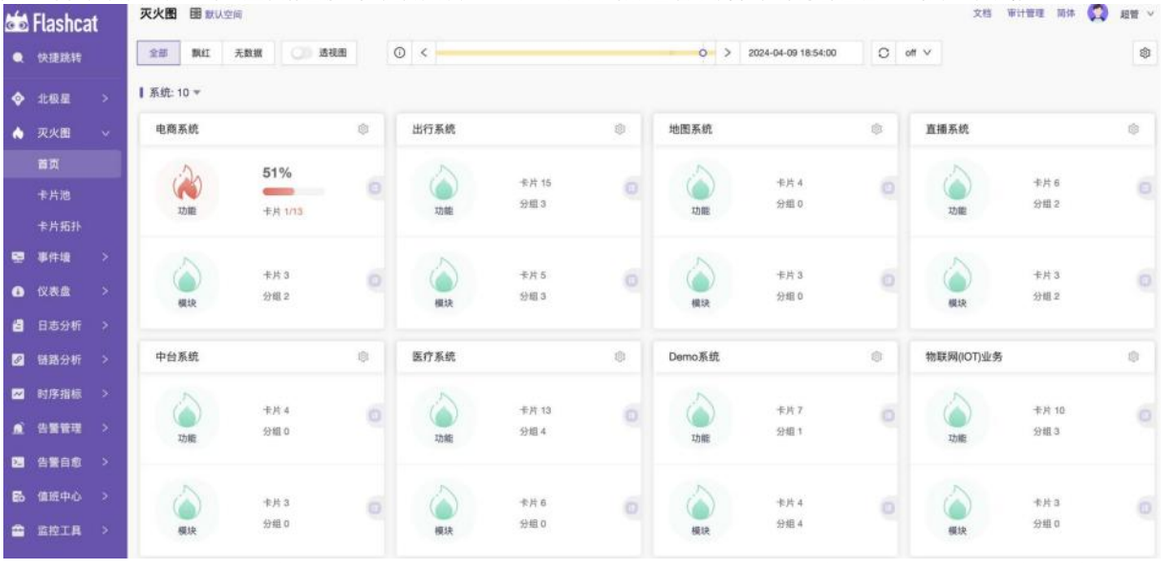
- ✧ 可能跟登录认证系统相关：如果登录不了，自然无法下单
- ✧ 可能跟 SKU 服务相关：无法获取 SKU，自然也无法下单
- ✧ 可能跟购物车系统相关：商品选好了，但是没法加入购物车，自然也无法下单
- ✧ 可能跟优惠券系统相关：明知道有个优惠券，想用却用不了了，自然也不想下单

但是，这么多基础设施、系统、子系统、模块、接口，海量的指标、海量的日志、海量的链路数据，而且重大的故障时刻，会产生一片一片的告警，怎么应对？经验数据表明，在故障处理过程中，绝大多数的时间耗费在确定具体的故障范围。如果能有一个系统，能够帮我快速圈定故障范围，甚至告诉我直接原因就好了，有没有这种系统？显然回答是“有”！

Flashcat 平台中抽象了“灭火图”概念，就是应用于故障定位的。灭火图，实时度量 IT 服务的核心功能和核心模块的健康状况，快速收敛故障范围，并且同时把可观测性数据良好的组织在一起，形成一个体系化的信息大厦。做个比喻：各种观测数据，更像是平铺的散的一块块砖，一个个柱子，而灭火图更像是一个基于这些砖、柱子组成的大厦。数据更有层次性，做好了数据串联，可以较为方便地定位直接原因。那灭火图产品的解题思路是什么呢？下面我们挨个拆解。

建立故障分析定位矩阵

北极星是业务视角的，灭火图则是技术视角的，可以一目了然地看到各个系统是否正常，举例如下：

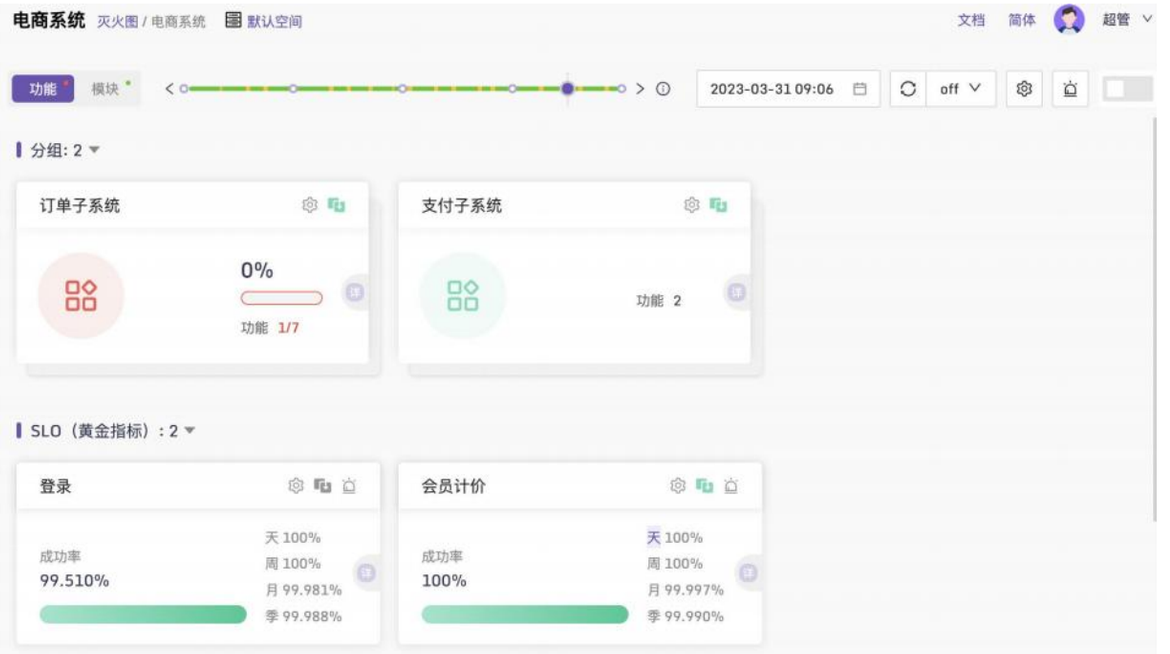


上图中可以很明显看出，“电商系统”有问题，该系统有 13 个 核心功能，其中 1 个功能异常，其他系统都是绿色，表示没有问题。快速地帮我们圈定了故障范围！

要实现这个效果，需要把这个业务系统的核心技术指标梳理出来，可以参考 Google SRE 实践中的四个黄金指标法则（流量、延迟、成功率、饱和度）以及 RED 方法论。也可以根据业务主流程的路径，梳理核心功能接口。某个接口或者某个模块有问题，会有一个冒泡上浮的过程，会把问题上浮到这个系统级卡片上，让卡片飘红，这样一来，即使是某个底层功能/模块有问题，我们也可以一目了然了解到了。

层层下钻引导式定位

飘红的卡片上，有详情按钮，点击即可查看详情，可以看到这个系统的各个子模块、功能的健康状况，还是拿刚才的电商系统举例，点击之后可以看到 2 个功能分组分别为“订单子系统”和“支付子系统”，其中“订单子系统”飘红，其他都是正常的，如下图所示：



继续点击“订单子系统”卡片，可以看到这个分组下面的子卡片，原来是“订单更新 DB”这个核心功能出了问题，当前成功率为 0（从卡片上的统计信息也可以看到天周月等时间跨度的统计数据）。

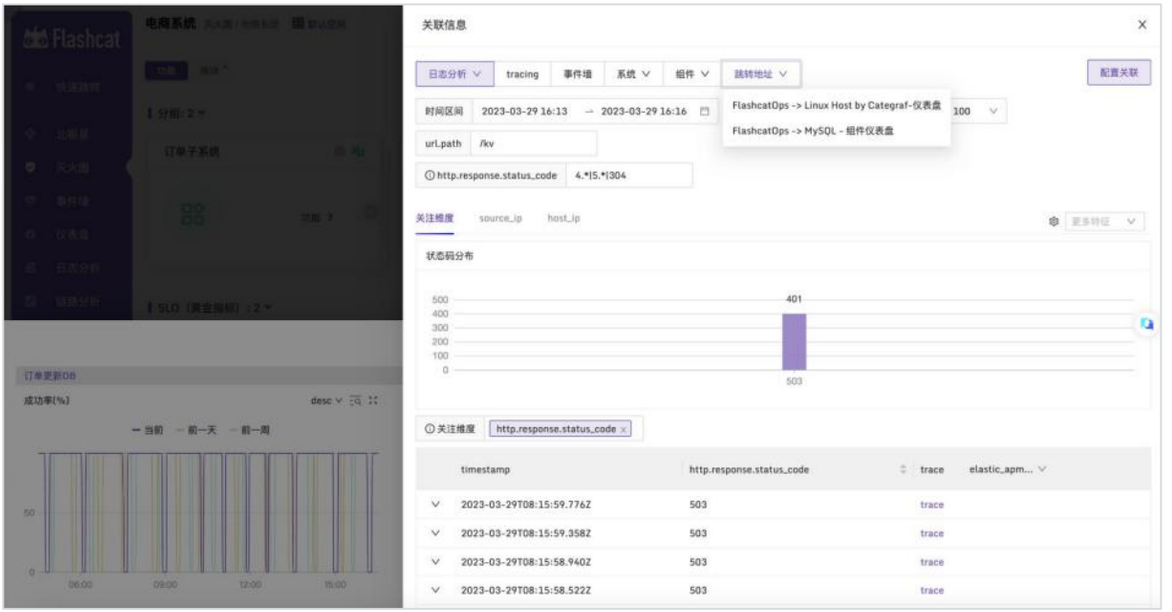
定位到功能模块，对于有些场景其实就够了，可能只需要把流量切走，故障就可以止损。但是很多公司其实没法做到这一步，就需要继续深究，到底是什么原因导致了这个模块的故障。通常，可以从以下方面着手：

- ✧ 变更：生产环境 70%的故障都是变更引起，这个故障时刻有哪些变更？应该通过“事件墙”尽快去确认，尽快去回滚
- ✧ 容量：是不是流量太大，超过了可以承载的流量水位，导致系统出故障，那就需要启用限流策略
- ✧ 日志：查看异常日志，如果成功率掉了或者延迟变大了，通过日志大概率可以发现一些端倪
- ✧ 链路追踪：针对错误请求或者异常日志，进行全链路 tracing，查看该请求流经上下游各节点时的状况，按图索骥
- ✧ 依赖的基础网络：查看基础网络的链路情况，包括连通性和质量
- ✧ 依赖的中间件数据库：查看依赖的中间件、数据库的核心指标以及埋点数据，可以断定是否是这些依赖的问题
- ✧ 依赖的其他服务：通过埋点数据或者日志，通常可以发现是否是依赖的服务的问题

涉及的内容挺多的，未必所有研发、运维都知道去哪里能快速找到这些数据，如果知识沉淀不够、传承得不好，很可能只有资深的研发、运维才知道去哪里查，怎么查。有没有办法让这些经验沉淀到系统里，有没有办法让人人都可以调查处理故障？在系统里做好这个数据串联呢？灭火图就可以做到！

沉淀经验的数据串联

灭火图中，可以为卡片为中心，配置或者自动生成串联逻辑，最终达到的效果是，点击折线图上的异常位置，就可以自动呼出关联信息的抽屉，展示那个时刻的异常日志、相关链路数据，当然，也可以配置外部系统的链接地址，并且带上当前卡片的上下文参数，比如自动链到相关的仪表盘，并且自动填充仪表盘的大盘变量。



上图样例中是展示了异常时刻的相关日志,有很多 status code 是 503 的日志,而且自动解析出了日志中的 traceid,出现了 trace 按钮，此时点击 trace 按钮就可以自动呼出链路的甘特图，或自动跳转到链路系统的页面，如丝般顺滑。

通过灭火图、事件墙之类的系统，我们快速地定位了故障的直接原因，接下来就是止损了，止损环节我们应该建立什么样的流程机制或者系统呢?

故障止损

首先，我们需要确立止损原则：**优先止损，后查根因**。很多新手研发人员通常习惯于遇到问题先找代码根因，甚至在线上 Debug，严重贻误止损战机，而且可能带来二次故障。运维人员的意识则会不同，通常会优先止损，不管根因，只要知道直接原因了，一般就可以确定对应的止损动作了，就要立即去执行，这样才能最快的止损故障。

要快速找到“直接原因”，就必须借助可观测性平台带来的“特征分析“能力，先通过”特征分析“，自动推荐可能引发该故障的异常特征作为线索，然后引导工程师进一步确认或者排除该线索是否是故障直接原因，是则执行相关联的止损方案，否则再次寻找其他异常特征，循环往复分析验证，直到找到引起故障的直接原因，从而止损并恢复服务。

我们推荐观测平台具备以下必要能力：

- ✧ 分析关键特征：
  - ○ 定位异常出现在什么范围，如果异常只局限在多活的某个单元内，切流即可快速止损；
  - ○ 定位异常是否局限于某个版本上，则应该进一步查看该版本对应的变更，将其回滚即可；
  - ○ 定位异常是否局限于某些实例上，则在容量允许的情况下摘除这些实例或可止损；
  - ○ 定位异常是否局限于某些来源，则封禁异常的来源 IP 或可止损；
- ✧ 分析关键事件：
  - ○ 定位到受影响的模块，查看相应模块的变更记录，将正在进行的变更回滚止损（统计数据显示，70%的故障都是因为变更引发的）；
  - ○ 查看是否有相关的配置变更、开关变更，有则回滚；
  - ○ 查看是否存在相关的营销推广活动，营销活动可能会造成流量的突增，进而引发容量的问题，则应该限流、扩容或者停止活动推广；
- ✧ 分析关键告警：
  - ○ 分析是否有关键的数据库服务宕机告警，则应该尽快切换数据库；
  - ○ 分析是否有关键的网络链路中断告警，则应该切换网络链路；

- 分析是否有第三方关键服务的不可用高兴，则应该切换供应商；

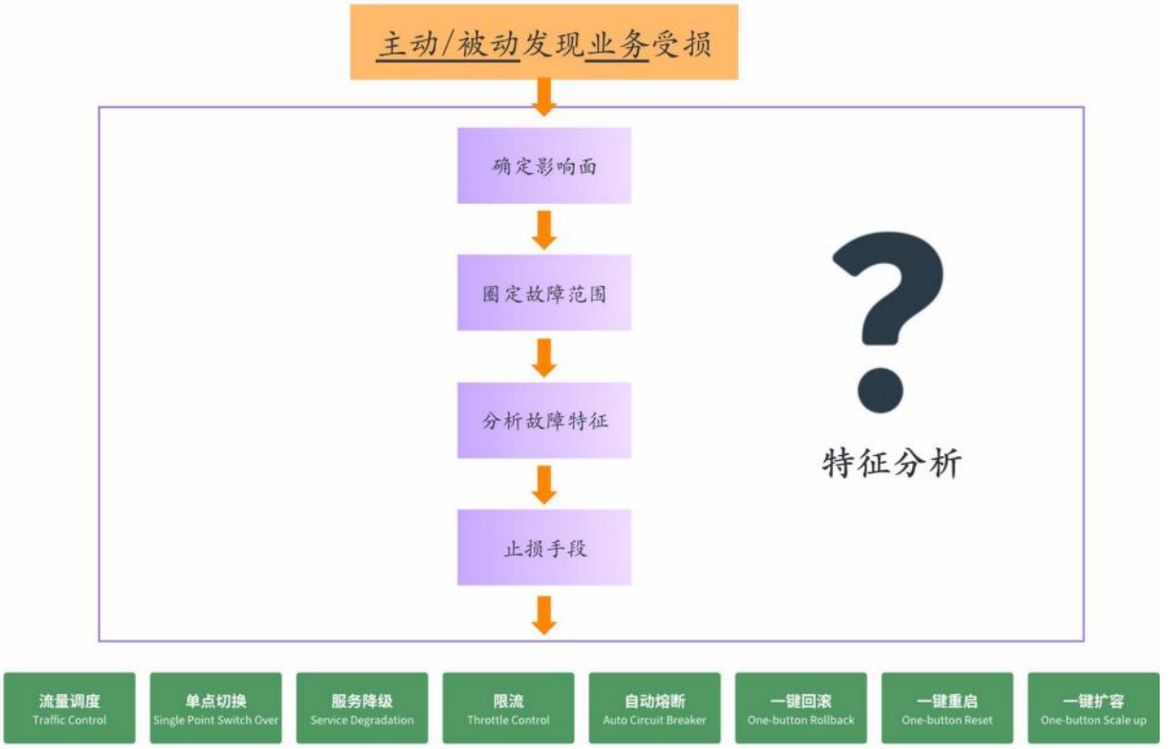
✧ 固化分析路径：

- 比如从异常的 metrics 点击下钻查看具体的日志信息，从日志中依据 trace_id 或接口信息向后 tracing，在 tracing 路径上的异常模块又能够跳转查看相对应的 metrics 或 logging 信息，加速分析效率；

✧ 工程师的经验平台化沉淀：

- 智能推荐不是银弹，把人的经验和机器的分析能力结合在一起，会让故障定位的流程效率更高；

止损需要提前建立预案，如果故障发生了，止损动作是现攒的，那对止损效果也会产生巨大影响，而止损这个动作，应该是分秒必争。预案可以是一个 SOP 文档，可以是一个脚本，可以是一个按钮，可以是 Chatbot 的一个命令，虽然故障五花八门，但是止损手段通常是可枚举的（**其实在代码设计之初就应该按照这些可枚举的预案来倒推设计架构**），有些公司梳理了自己的止损三板斧，或者八大止损手段，什么重启、回滚、切流量、限流、降级等等之类的，从侧面可以证明常见止损手段是可枚举的。



监控系统通常有个告警自愈的功能，快猫星云的产品 Flashcat 就具备这个能力，即在告警之后自动触发某个脚本，这在一些场景是非常有效的，只要止损逻辑可以固化为固定的脚本，比人手工去操作要靠谱得多。但是很多止损逻辑远非一个脚本可以搞定的，所以很多公司会做一个『预案管理系统』，专门用于管理预案、执行预案。这也是稳定性体系里的重要一环。

预案是否完备？是否有效？是需要有个衡量手段的。通常我们看两个指标，一个是告警规则的预案预置率，一个是故障的预案应用率。告警规则的预案预置率，是统计配置了关联预案链接的告警规则占总的告警规则的百分比，如果绝大部分告警都没有配置预案链接，要么是预案做的不完备，要么是告警规则不合理；预案应用率是统计历史告警事件和故障，看看有多少是用既有的预案止损的，有多少是现攒的，这个比例越高，显然预案做的越好。另外，预案需要定期演练测试，如果很久不用的预案，真用的时候很可能会发现已经跟现有的逻辑不匹配了，不好使了，延误了故障恢复的时间，这类问题屡见不鲜。

故障复盘阶段

每发生一次故障，都是付出了代价的。从故障中学习，总结经验、落实改进，是增强故障预防能力的宝贵机会。有些公司对故障的处理有一个明确的 1-5-10 要求，即：1 分钟发现，5 分钟定位，10 分钟恢复，但是对故障的复盘，缺少明确的指导性要求。而实际上，故障复盘非常重要，Google SRE 倡导的 postmortem 5 whys 方法，就是一种很好的理念，核心就是不断追问原因背后的原因，找到最本质的问题，有点第一性原理的味道。

另外，对事不对人，一切都是为了改进，而不是为了指责，否则复盘最终会沦为相互推诿，而真正需要大家齐心协力

的改进点却无法落地。

小结

本章节围绕稳定性保障中的故障生命周期，梳理了各个环节的关键逻辑，介绍了相关系统的建设思路和样例。要构建完备的稳定性保障体系，还需要软件运行环境规范、各类组件 PaaS 建设、混沌工程系统、全链路压测、风险量化系统、预案管理系统等众多系统协同。稳定性保障是一项复杂的系统工程，而可观测性体系建设无疑是其中一项重要的基础工程

以被全世界范围众多企业采纳、应用和发展的 SRE 方法论为例，可观测性技术是 SRE 武器库中的关键兵器。

后续章节将围绕可观测性体系建设的范式和具体案例深入分析。

实践篇

Flashcat 是一个云原生/微服务时代的可观测性产品，旨在通过建设一个一体化的可观测平台，解决数字化服务稳定性保障的难题，致力于让可观测性技术更好的落地和发挥价值。本文将全面分析当前阶段企业落地可观测性技术可能遇到的问题和挑战，并详细介绍 Flashcat 的解决思路和产品方案。

本章节共分为三大部分：

1. 第一部分：当前阶段落地可观测性的问题和挑战
2. 第二部分：落地好一个可观测系统的三大要素
3. 第三部分：面向稳定性保障的可观测性产品 Flashcat

当前阶段落地可观测性的问题和挑战

当前阶段，虽然 OpenTelemetry 提出了可观测性的概念，极大促进了可观测体系的发展，但一个统一的可观测体系在企业落地仍然任重道远。

可观测性落地的痛点

从接触的大量国内企业来看，在可观测性领域目前普遍存在两个痛点：

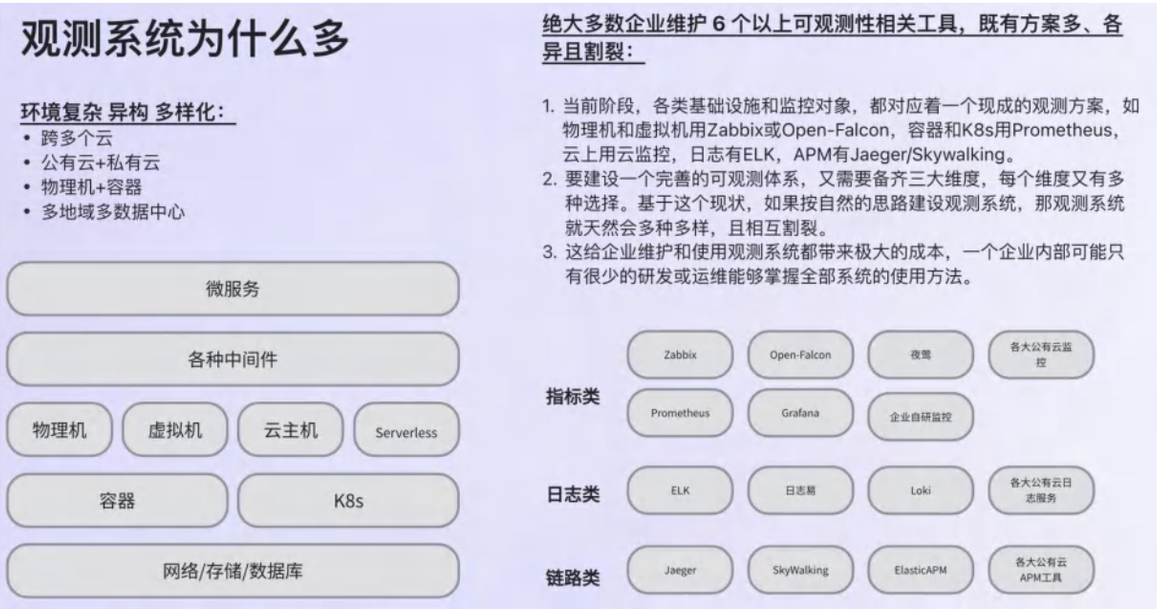
- ✧ 一是，观测系统多，一个统一的观测系统是普遍的需求，但缺少好的产品和可行的落地方案；
- ✧ 二是，稳定性保障难，虽然各种观测数据都有了，但在故障发现、故障定位上仍然存在发现慢，定位难，协同难等问题，在稳定性保障上技术团队经常处于被动；

观测系统为什么多

当前阶段，各类基础设施和监控对象，都对应着一个现成的观测方案，如物理机和虚拟机用 Zabbix 或 Open-Falcon，容器和 K8s 用 Prometheus，云上用云监控。

要建设一个完善的可观测体系，又需要备齐三大维度，每个维度又有多种选择。基于这个现状，如果按自然的思路建设观测系统，那观测系统就天然会多种多样，且相互割裂。绝大多数企业维护着 6 个以上的可观测性相关工具，既有方案多、各异且割裂。

这给企业维护和使用观测系统都带来极大的成本，一个企业内部可能只有很少的研发或运维能够掌握全部系统的使用方法。



服务稳定性保障为什么难

虽然有众多观测系统，但由于缺少对数据的融合，以及面向稳定性保障场景的专业建设，导致稳定性保障仍然很难，典型的现象是：

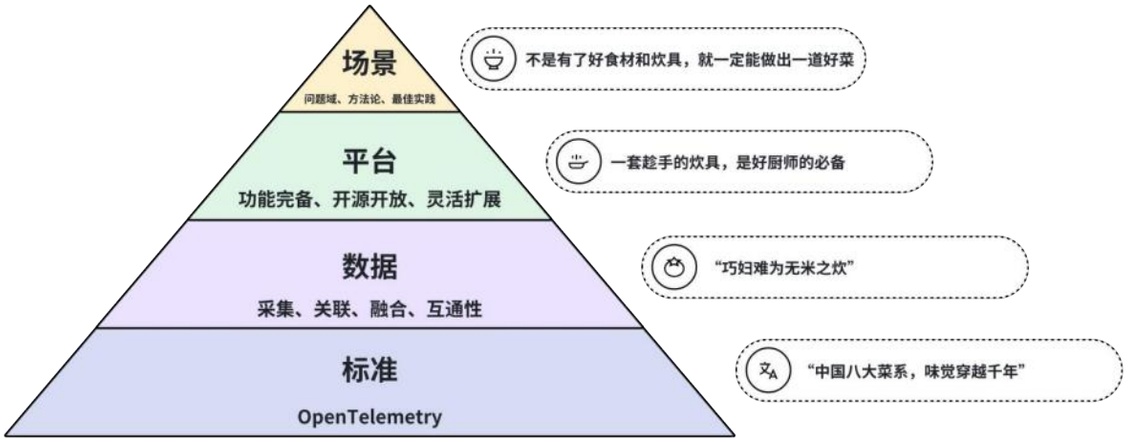
- ✧ 每天报警很多，多到处理不过来，但真正影响业务的故障出现时，却是由业务部门反馈或客户投诉来发现；
- ✧ 一旦确认故障，群里就炸了锅，相互确认、相互等待，团队协同难，故障定位慢；
- ✧ 技术团队努力工作一整年，到年底却难以讲清楚稳定性保障的效果，价值难以量化，导致无法获得长期资源投入；

很多企业可能已经不缺少可观测性数据，但缺少的是将数据价值在稳定性保障领域发挥出来的产品、方法和最佳实践。

落地好一个可观测系统的四个要素

经过和企业客户及大量开源用户的合作，结合快猫星云团队的大型服务稳定性保障经验，快猫星云团队总结了解决企

业可观测系统落地问题的四个要素：标准、数据、平台、场景。



假如把建设一套面向稳定性保障的可观测系统比喻为做一道好菜，那标准就是菜谱，数据就是食材，平台就是炊具，场景就是厨艺。

- ✧ 标准：缺乏标准的数据，就属于“garbage in, garbage out”，很难自动化，数据价值大打折扣。
- ✧ 数据：巧妇难为无米之炊，需要做好稳定性保障，备齐各维度的数据在所难免。
- ✧ 平台：一套趁手的炊具是好厨师的必备，监控和可观测所需的通用功能和接口需要友好而高效，便于支持上层场景的实现。
- ✧ 场景：不是有了食材和炊具就一定能烧好一道菜，稳定性保障的经验、方法、和最佳实践是整个系统最后输出效果的关键。

标准

2019 年 ，CNCF 托管的 OpenTracing 项目和 Google 发起的 OpenCensus 项目正式合并，以全新的品牌 OpenTelemetry 亮相，可观测性的概念迅速发展、普及并被广泛接受。截止今日，可观测性体系已经成为数字化服务的必备工具，OpenTelemetry 项目成为 CNCF 所有项目中发展最快的之一，成为可观测性技术权威标准：

- ✧ 定义了 metrics、logs、traces 的标准规范和相应的 API，实现了针对各种语言的 SDK，解决了可观测性数据生成难、覆盖不全的问题；
- ✧ 围绕 OpenTelemetry，建立了完善的可观测性的软件工具生态，解决了 telemetry 数据在不同软件工具之间的互通性问题；
- ✧ 解决了 metrics、logs、traces 各个维度数据的关联性问题；

数据

观测系统的数据采集和输入在当前都有哪些普遍的问题?

- ✧ **传统的监控采集方案不能适应云和容器环境的数据采集：**

容器资源随时产生，随时消亡的特点，是传统以物理机为中心的采集器难以适应的。云上的服务组件如 RDS、LBS 等有自己的监控采集通道，很多组件的关键指标或宿主的指标外部的采集器难以直接获取。

- ✧ **新的采集方案质量良莠不齐，管理成本高：**

适应 K8s 和容器环境的配套监控系统是 Prometheus，Prometheus 依赖各种 exporter 导出各类采集对象的数据，但 exporter 的研发和采集配置并无统一标准，导致 exporter 的配置和管理复杂，质量不一。

- ✧ **Metrics、Logging、Tracing 各类数据的采集方案差异大，有没有统一的采集方案?**

针对 Metrics、Logging、Tracing 三大维度数据的采集，如果也能由一个 agent 来负责，将更为理想，既能减少维护多个 agent 的成本，又可以在数据采集阶段对三大维度数据在源头进行关联。 目前实现了这个采集能力的是商业配套并开源的 datadog-agent，但由于历史原因，datadog-agent 体积庞大结构复杂，并且有独立的协议，主要服务于 datadog 自己的平台。

- ✧ **落地一个可观测系统要不要把已有的采集方案推倒重来?**

无论是部署指标 agent/exporter，还是安装日志采集器，特别是需要研发在代码中嵌入 SDK 的方案，整个过程落地一次非常不易。如果引进一个新的观测系统需要把这些工作重做一遍，可能很多人会望而却步。特别是在还没确认新产品的

效果前就去做这个事情，无异于一次冒险。

本章节将在后续部分介绍 Flashcat 产品的解决方案。

平台

一个好的观测平台，应该具备哪些特性?

✧ **功能上是完备的，能够覆盖可观测性的三大维度**

支持 metrics、logging、tracing 的数据存储、可视化、告警。支持云上云下和多种软件架构。

✧ **数据上是互通的，能够和整个可观测性生态系统对接**

既可以将其他可观测性系统产生的数据作为输入，加以利用，也可以把经过平台加工过后的数据，开放给周边的系统，加强协同作用。

✧ **分析上是智能的，能够大幅降低用户理解数据的门槛和成本**

可观测性数据量越来越大，如果缺乏有效的洞察手段和数据处理手段，那么数据多反而会变成一种干扰和负担。如何提供全局统一的数据视图，建立有效的信息系统，增强分析的智能化水平，降低用户使用数据、理解数据的成本，尤为关键。

✧ **架构上是可扩展的，能够高效、稳定的支撑业务的发展**

可观测性系统作为超一级服务，稳定性至关重要，如果出现故障，相当于整个 IT 系统处于“无人驾驶”状态。此外，在现代化的微服务架构下，可观测性数据量呈数量级增加，因此平台能否水平扩展是关键因素。

Flashcat 在平台的建设和选择上做了哪些思考，将在后文介绍。

场景

服务稳定性保障是以故障处理为中心，分为常态预防、发现处理、复盘改进三大部分，故障出现的频率小（常态预防）、出现后处理快（发现处理）则稳定性保障能力就强，反之稳定性保障能力就弱。

✧ 常态预防：是稳定性保障的重点，让故障不出现是更值得追求的目标。相关的工作有很多，如压测、高可用架构、发布控制、服务巡检等等。

✧ 发现处理：故障是不可能被 100%预防住的，可预期和意外的情况总会出现，因此发现处理能力也需要重点建设。而发现处理的原则是先止损后排查，以恢复服务和用户的核心体验为首要目标。

✧ 复盘改进：是为了从故障中吸取经验和教训，并输出增强预防和发现处理能力的改进工作。

要定位好可观测性在稳定性保障中的作用，还需要对以上三大部分做进一步的分解，如下：

可观测服务的环节包括日常巡检、日常排障、故障发现和故障定位。其中故障发现和故障定位，是整个过程中时间要求最为紧迫，也最有挑战的两个环节。

基于以上分析，结合稳定性保障的经验，一个面向稳定性保障的观测产品和通用观测产品至少应有以下不同：

✧ 故障发现要明确能量化故障的关键指标，并在产品上做 VIP 对待，而不只是简单的平铺或做个分级；

✧ 故障处理的首要原则是先止损后排查，而面向止损和面向根因定位的实现有重要的区别，**故障定位不是一味的排查“根因”，而是一个将故障的特征和关键事件连接到止损动作的过程**；

✧ 故障定位也不存在一种包打天下的方法，各种可用的手段都应该用上，可用的手段越多定位能力就越强；

✧ 做好稳定性保障不只是产品本身，相应的流程和机制也需要配套建设，能定义故障的告警，其发送渠道以及响应机制和一般的告警是完全不同的流程和优先级；

后面将重点介绍 Flashcat 产品在设计实现上都结合场景做了哪些考虑。

面向稳定性保障的可观测性产品 Flashcat

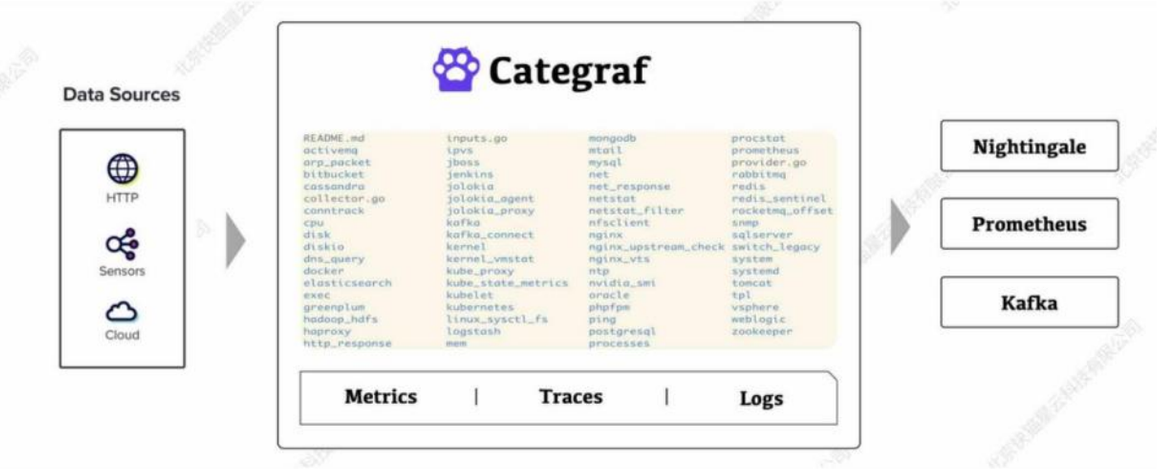
Flashcat 是帮助用户在云原生/微服务普及的当前阶段落地好可观测能力，做好服务稳定性保障的可观测性产品。Flashcat 在数据、平台、场景层面的实现和思考将在本部分重点分析和介绍。

数据输入

如上一章分析，目前观测平台的在数据输入上有四大问题，Flashcat 是如何解决这些问题的呢？Flashcat 采用了搭配采集器并支持数据集成（Integration）相结合的方案。

采集能力

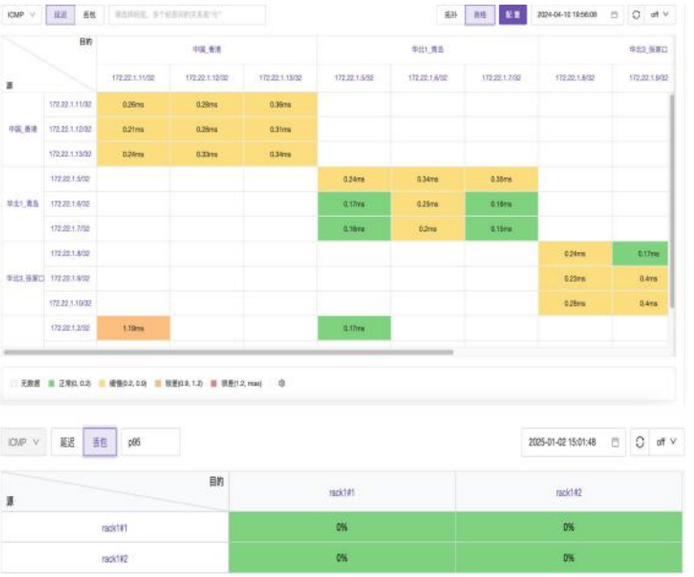
Flashcat 开源了一个 All-in-one 的采集器：Categraf，能够把数量众多、分散的各类 Exporter，作为 Categraf 的一个个插件内置进来，统一管理起来，用户不再需要满世界去找各种 Exporter 和各类组件的采集配置，并担心采集器的质量问题，并且每个采集器的每个插件的配置可以在 Flashcat 控制台上批量修改和下发，方便管理和维护。同时，Categraf 也能够采集 Logging 数据，实现了统一采集的方案。Categraf 采用插件机制，灵活扩展，现已支持 100 种插件，开箱即用，100 多万次下载，广受信赖。

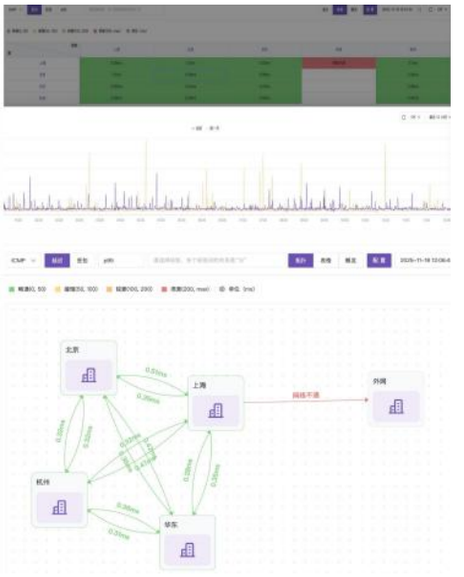


> Categraf 开源项目 GitHub 地址：https://github.com/flashcatcloud/categraf

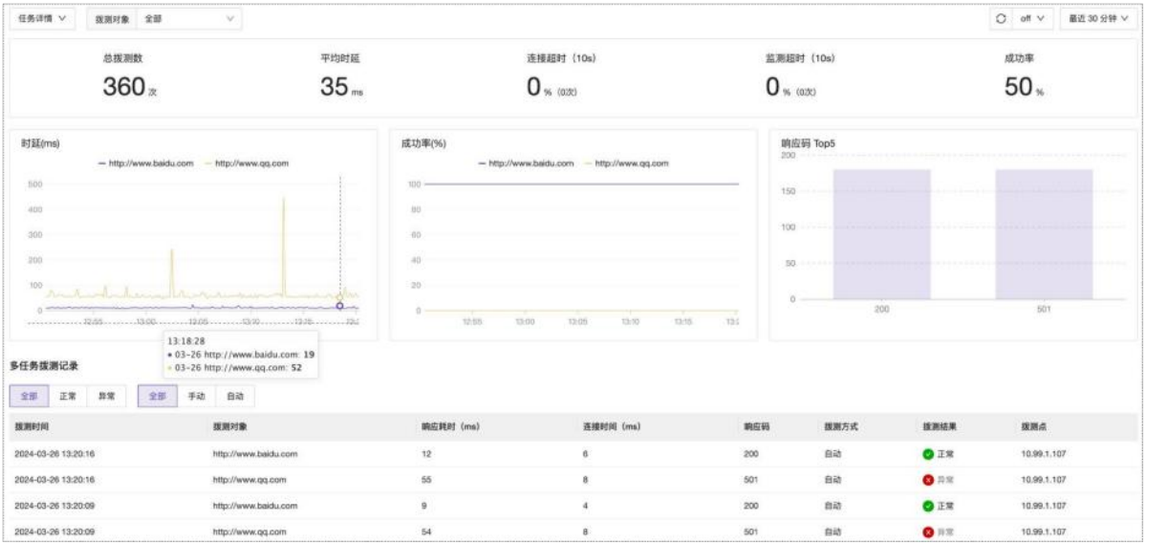
基础网络是影响稳定性的重要因素，如何快速、准确的发现和定位大规模基础网络的故障，是业界难题。Categraf 在

网络层面的监控数据采集功能上，特别的做了增强，比如 Pingmesh。Pingmesh 是一种用于测量和监控网络性能的技术，通过在一组通信对等体之间执行 Ping 测试来评估网络的可用性和延迟。Categraf 支持 Pingmesh 功能，支持了 TCP、UDP、ICMP 三种探测协议，在设备之间进行互相探测，并绘制出各个层面的连通性视图，从全局视角观测整个网络的连通性，这对于大规模、全球化布局的网络架构特别有价值，能够帮助工程师一目了然的看清楚不同的数据中心之间、不同的机柜之间、不同机器之间的网络连通质量。



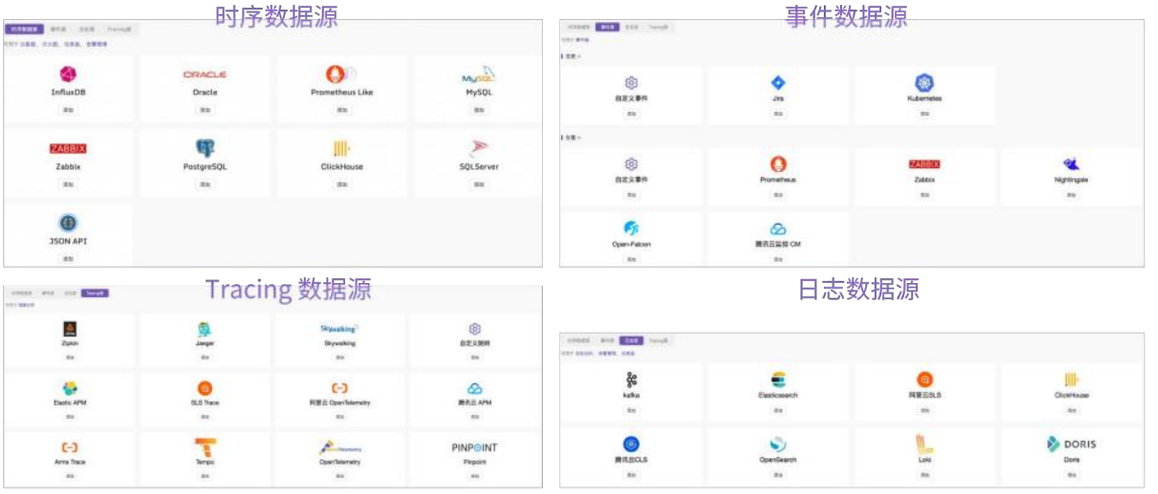


Categraf 也集成了网络拨测功能，提供了多协议的网络拨测能力，支持从多个 Categraf 节点，模拟用户发起请求，对探测目标进行多方位的探测，及时发现网络故障问题，包括网络丢包率、时延等，并输出探测报告。



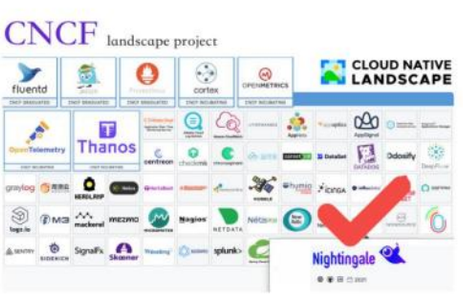
数据集成

对于企业内部已经采集不希望重新采集的数据，以及云上的数据，Flashcat 则采用了数据源集成的方案。Flashcat 能够集成如 Prometheus、Zabbix、ELK、Skywalking、Jaeger、Zipkin、以及各类云监控、阿里云 SLS、腾讯云 CLS 等数据源（Flashcat 现已支持 40 多种数据源）



平台建设

Flashcat 采用开源夜莺监控（Nightingale：https://github.com/ccfos/nightingale）作为产品的平台层。Nightingale 夜莺是一款开源云原生监控告警工具，是 CCF 接受捐赠并托管的第一个开源项目。夜莺的统一告警引擎，可以对接 Prometheus、Elasticsearch、ClickHouse、Loki、MySQL 等多种数据源，提供全面的告警判定、丰富的事件处理和灵活的告警分发及通知能力。

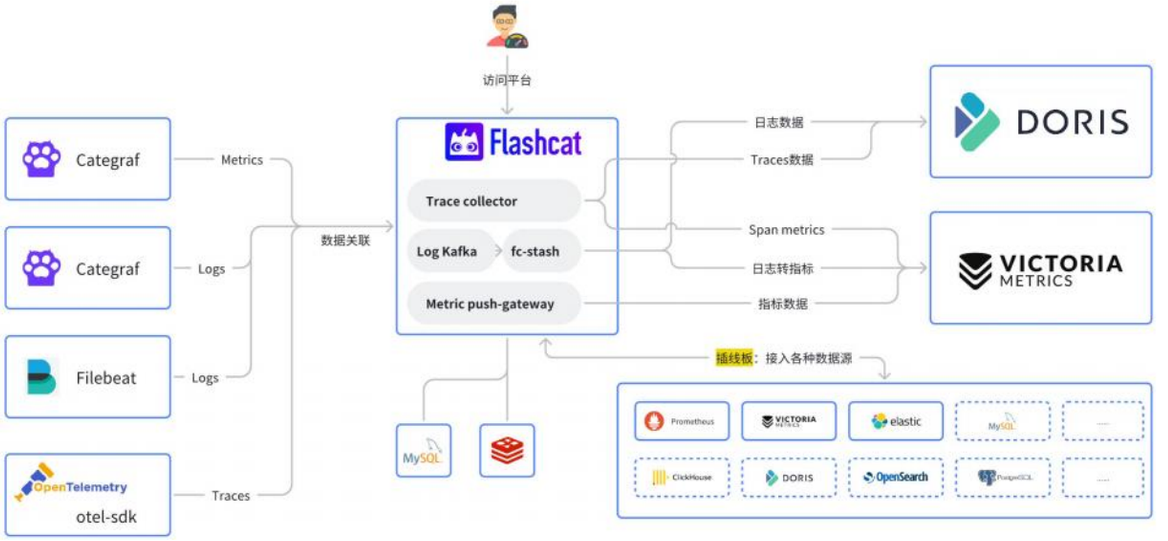


中国计算机学会接受捐赠并托管的第一个开源项目

Flashcat 选用开源夜莺监控作为可观测产品的平台层，有几个优势：

- ✧ 协议：遵循 OpenTelemetry 协议，在 Metrics 维度融入 Prometheus 生态，均是目前适用最广的标准；
- ✧ 采集：在采集上推荐使用 Categraf，并针对两者的结合做了优化；
- ✧ 存储：时序数据存储推荐使用 VictoriaMetrics，兼容 Prometheus 接口，日志和 APM 存储推荐使用 Apache Doris，快猫星云团队在 VictoriaMetrics/Doris 具有大规模使用的丰富经验；
- ✧ 架构：简单的架构设计，可以方便的实现单集群或分布式高可用模式；

Flashcat 平台总体架构如下：



✧ 功能：开源夜莺具备完备的通用监控及可观测功能，如 Prometheus 等常见的数据源集成、指标查询、日志查询、链路查询、统一的告警配置、大盘、告警自愈、用户管理、权限管理等；

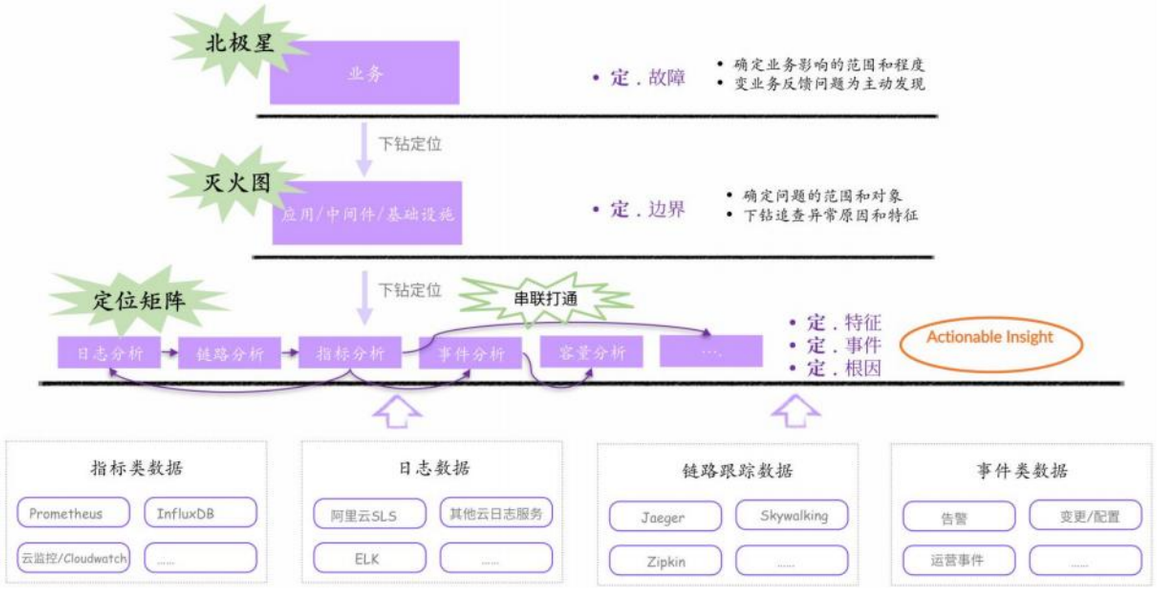
✧ 扩展：开源夜莺提供了 API，可以方便在此基础上增加更多的功能，支持多样化的场景功能实现；

Flashcat 在开源夜莺的基础上完善了更多的监控和观测能力，功能涵盖指标、日志、APM 、RUM、监控告警，同时增加了更多的集成数据源、更完善的数据采集方案等。

场景方案

完成了数据输入和观测平台建设后，Flashcat 在此基础上，面向稳定性保障场景进行了专业的产品设计，这也是 Flashcat 产品设计的主要目标和独特之处。

Flashcat 首先针对故障处理中最有挑战的两个环节做产品实现,即上一章中分析的“故障发现”和“故障定位”。Flashcat 的基本实现逻辑是针对不同环节的目标，结合最佳实践实现有针对性的产品功能。

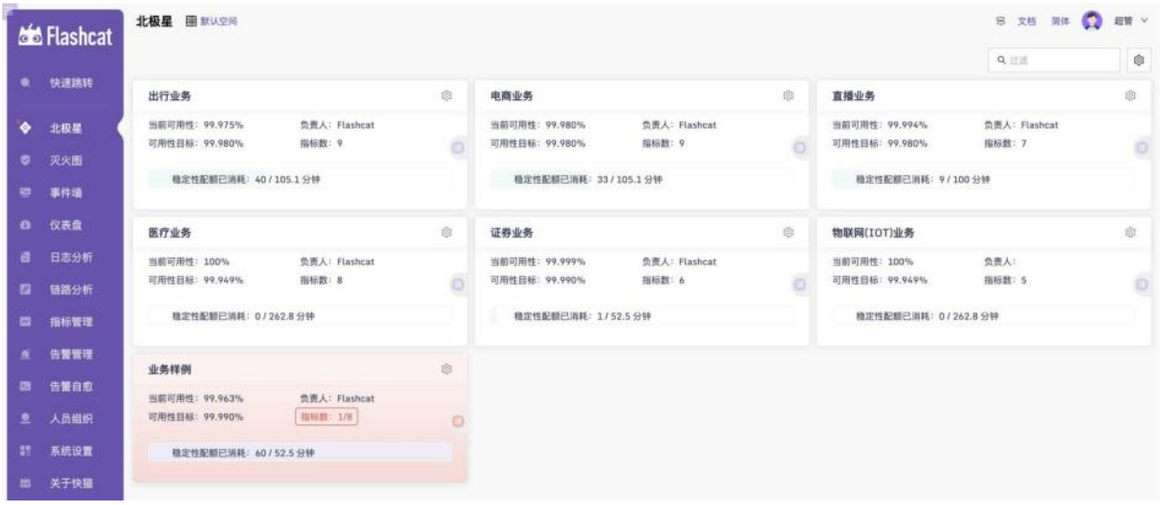


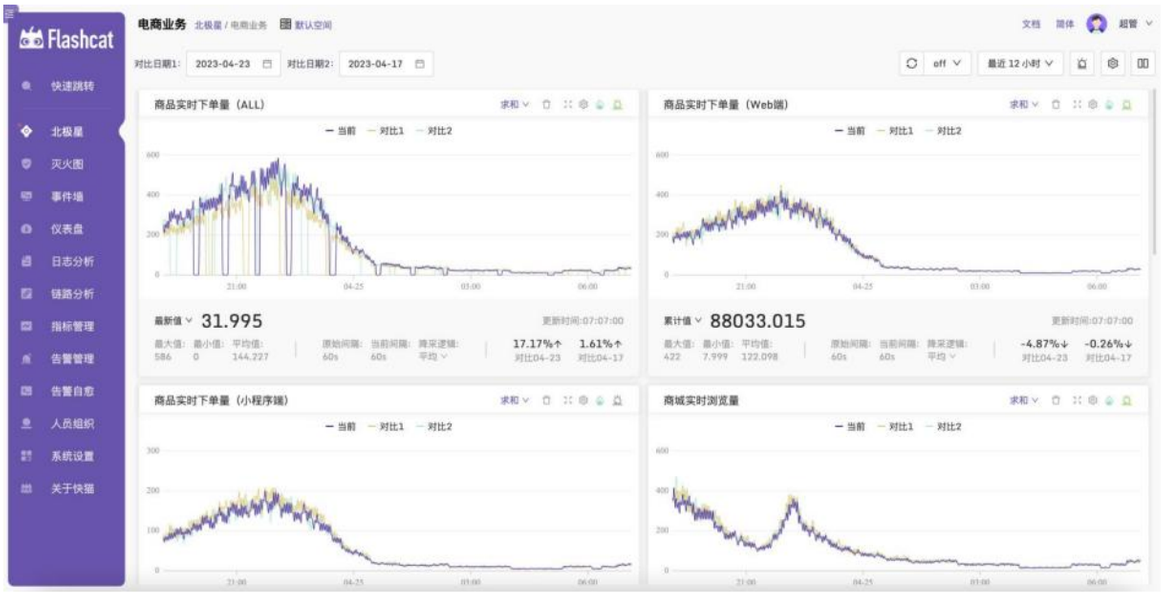
Flashcat 的产品分层为：北极星、灭火图、定位矩阵（日志分析、链路分析、事件分析、指标分析、容量分析等），以及这些系统间的引导下钻和相互串联功能。

北极星

针对“故障发现”实现，用于量化和定义“业务”健康度，推荐管理的指标如实时在线用户数、实时订单量、实时支付量等，通常是“量”相关的指标，是业务负责人和老板们能看懂并关心的指标。

除了业务指标，北极星指标也可以是用户体验的核心指标，核心是能够量化和定义故障，并且便于业务、技术以及上下游工程师共同理解。





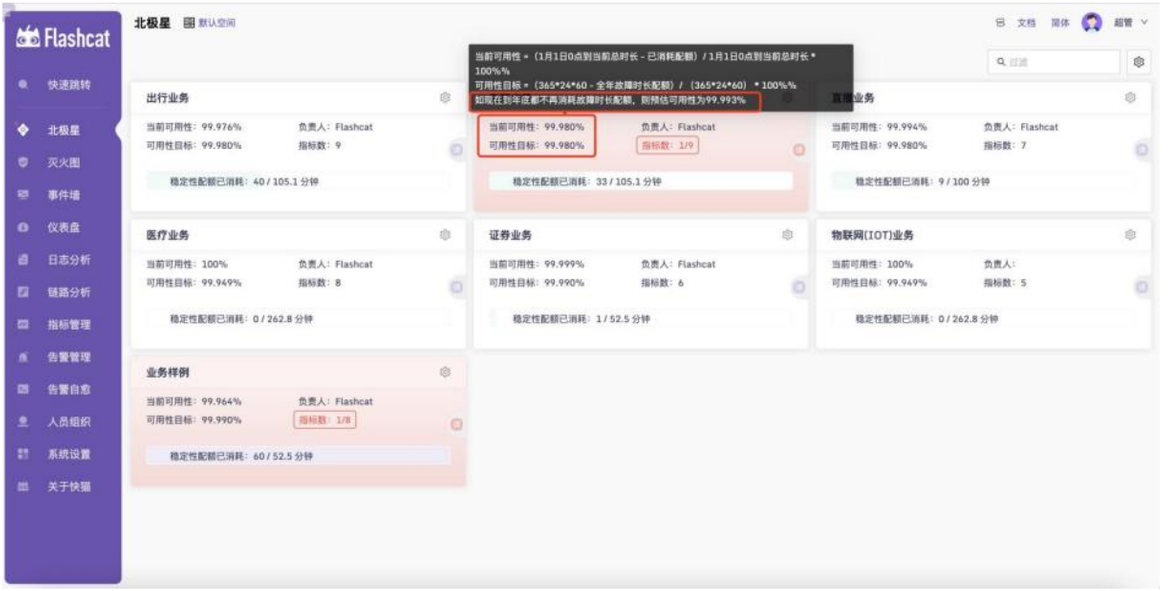
北极星的指标可以来自 Categraf 采集的数据，也可以来自日志分析和链路分析系统产生的数据，也可以来自各种在 Flashcat 系统中集成的数据源。推荐直接从业务的数据库表中查询相关数据生成指标，这样获得的北极星指标最为准确，Flashcat 提供了从库表中生成指标的多种方法，并保障相关链路的稳定和指标数据的正确性。

当然，实施过程中可以视业务和架构的情况来决定数据来源，遵循先落地后优化的原则。

北极星的核心功能包括：

✧ 智能报警：北极星多为“量”相关的指标，并且通常具备周期性，非常适合使用智能检测算法来发现异常；

✧ 稳定性 SLO 管理：北极星定义了业务的健康度，因此可以基于北极星管理稳定性的配额和可用性目标；



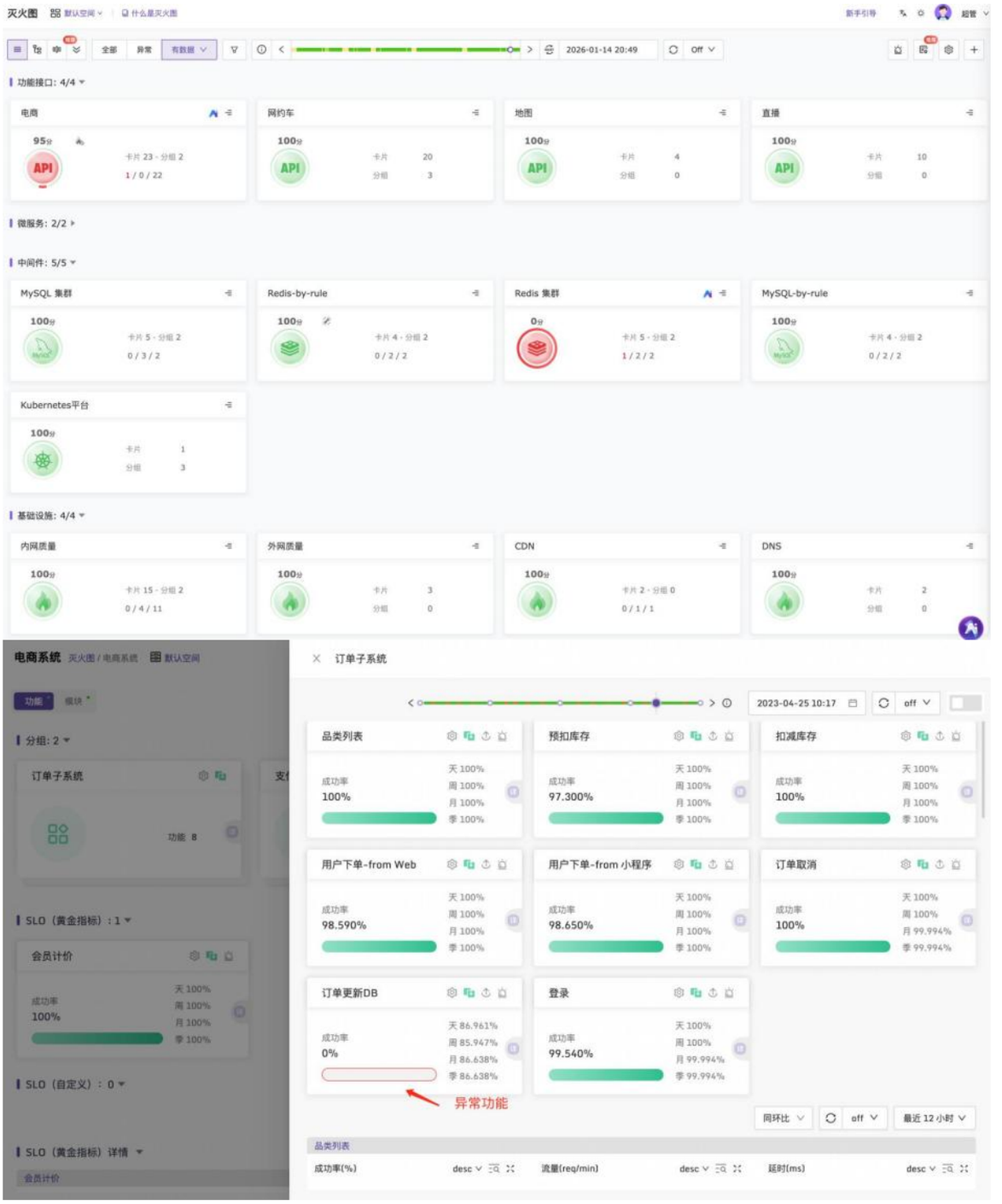
✧ 故障事件管理：北极星曲线上可以标识记录事件，方便用户了解历史曲线的异常原因；

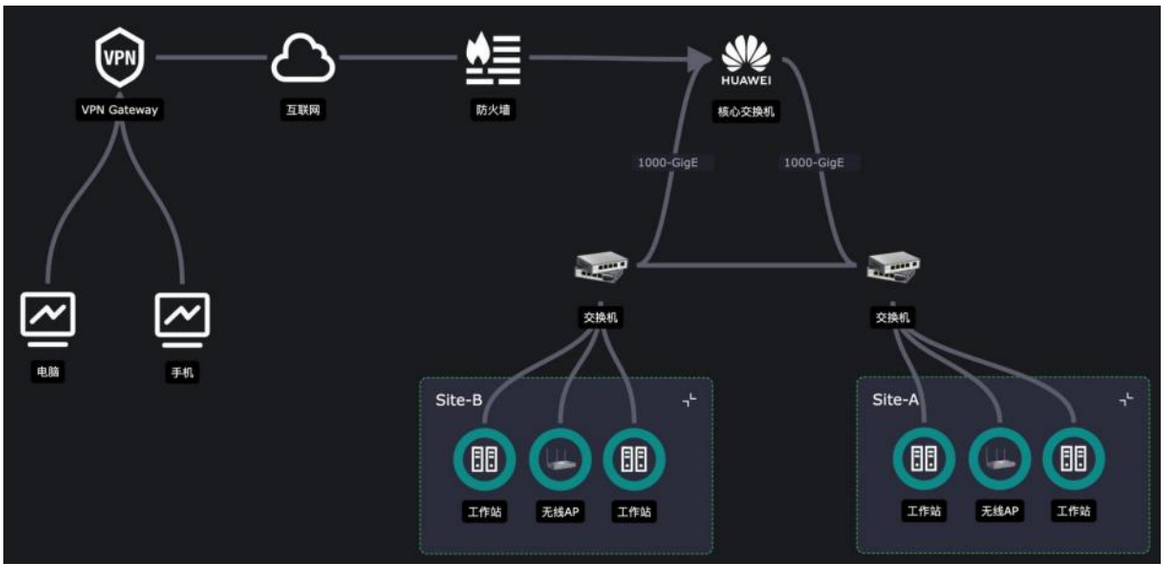
✧ 大屏生成：北极星指标就是活动大促、节假日等业务高峰期的最佳大屏数据来源，Flashcat 支持一键从北极星系统内生成大屏；

北极星的重要功能还包括：下钻追查功能、指标管理、告警管理、权限管理等。

灭火图

“故障定位”中的第一步是希望看到故障源是来自哪个 IT 团队所负责的服务，或服务层的影响面（哪里着火了），以便快速缩小追查范围，确定责任团队，有的放矢。


灭火图
默认空间
什么是灭火图
新手引导
超管
全部
异常
有数据
2026-01-14 20:49
Off
功能接口: 4/4
电商
95分
API
卡片 23 · 分组 2
1 / 0 / 22
网约车
100分
API
卡片 20
分组 3
地图
100分
API
卡片 4
分组 0
直播
100分
API
卡片 10
分组 0
微服务: 2/2
中间件: 5/5
MySQL 集群
100分
卡片 5 · 分组 2
0 / 3 / 2
Redis-by-rule
100分
卡片 4 · 分组 2
0 / 2 / 2
Redis 集群
0分
卡片 5 · 分组 2
1 / 2 / 2
MySQL-by-rule
100分
卡片 4 · 分组 2
0 / 2 / 2
Kubernetes平台
100分
卡片 1
分组 3
基础设施: 4/4
内网质量
100分
卡片 15 · 分组 2
0 / 4 / 11
外网质量
100分
卡片 3
分组 0
CDN
100分
卡片 2 · 分组 0
0 / 1 / 1
DNS
100分
卡片 2
分组 0
电商系统
灭火图 / 电商系统
默认空间
功能
模块
分组: 2
订单子系统
功能 8
SLO（黄金指标）: 1
会员计价
成功率
100%
天 100%
周 100%
月 100%
季 100%
SLO（自定义）: 0
SLO（黄金指标）详情
会员计价
订单子系统
2023-04-25 10:17
off
品类列表
成功率
100%
天 100%
周 100%
月 100%
季 100%
预扣库存
成功率
97.300%
天 100%
周 100%
月 100%
季 100%
扣减库存
成功率
100%
天 100%
周 100%
月 100%
季 100%
用户下单-from Web
成功率
98.590%
天 100%
周 100%
月 100%
季 100%
用户下单-from 小程序
成功率
98.650%
天 100%
周 100%
月 100%
季 100%
订单取消
成功率
100%
天 100%
周 100%
月 99.994%
季 99.994%
订单更新DB
成功率
0%
天 86.961%
周 85.947%
月 86.638%
季 86.638%
异常功能
登录
成功率
99.540%
天 100%
周 100%
月 99.994%
季 99.994%
同环比
off
最近 12 小时
品类列表
成功率(%)
desc
流量(req/min)
desc
延时(ms)
desc



Flashcat 根据最佳实践定义了一套量化 IT 服务健康度的指标和算法。量化对象包括功能（核心功能，如下单、查询等）、模块（核心微服务模块，通常为多个实例）、组件（如 MySQL、Kafka 等）、基础设施（内外网、CDN、DNS 等）。

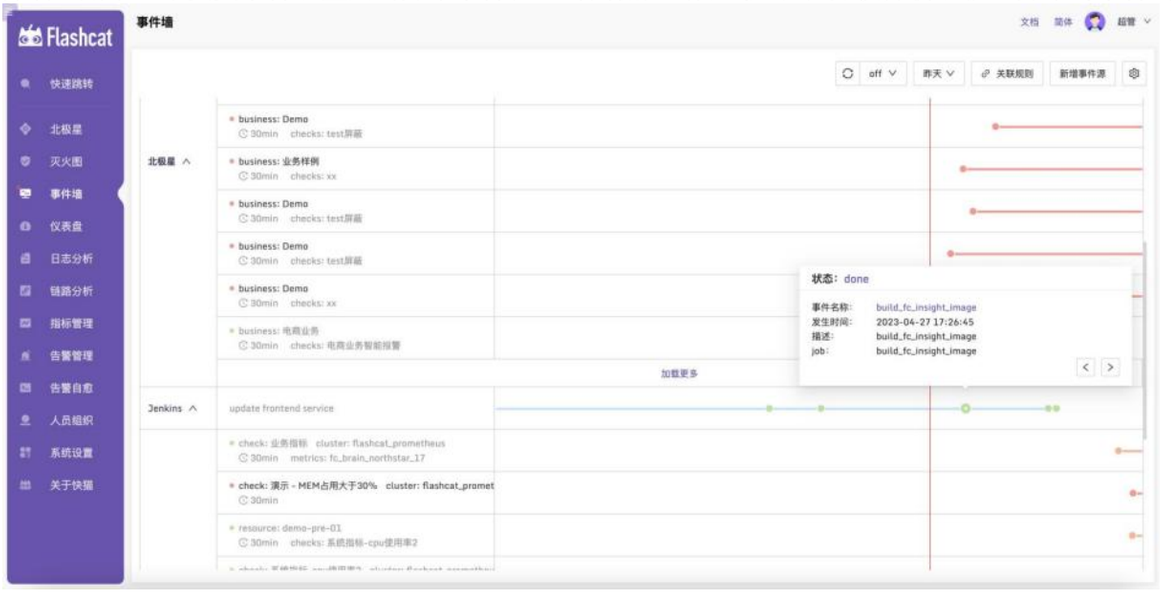
- ✧ 功能：黄金指标成功率、流量、响应延时，可自定义
- ✧ 模块：实例存活率、实例状态、CPU/内存/磁盘使用率，可自定义
- ✧ 组件：各类组件的量化指标不同，可自定义，和组件仪表盘联动
- ✧ 基础设施：各类基础设施的量化指标不同，可自定义，和仪表盘联动

灭火图的重要功能包括：下钻追查、飘红设置、告警管理、历史回溯、指标管理、关联管理、巡检功能等。

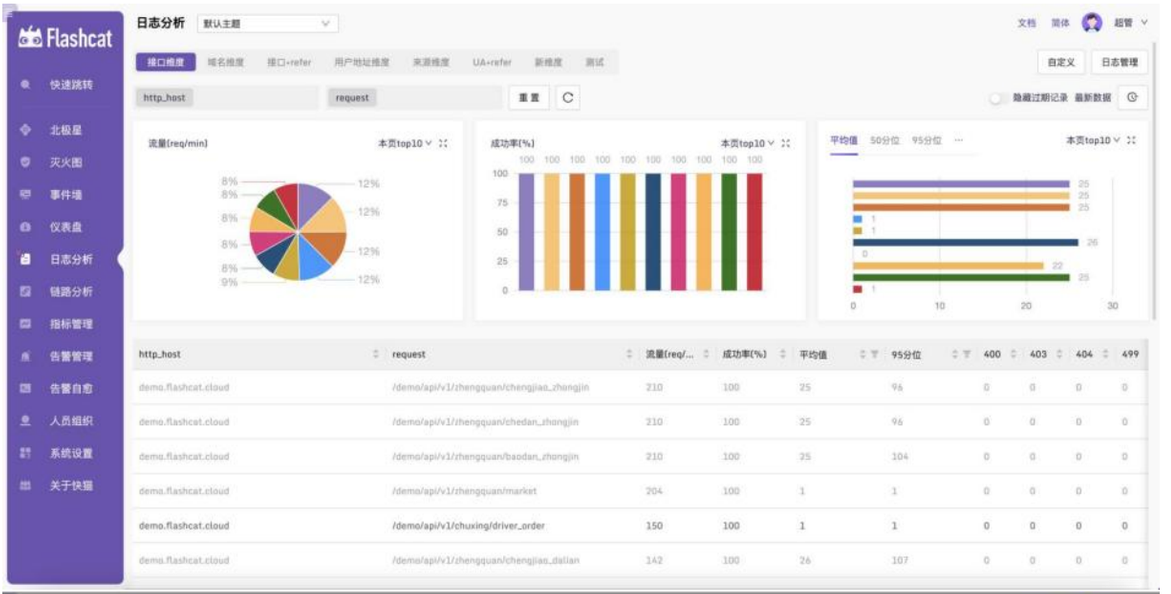
多维定位矩阵

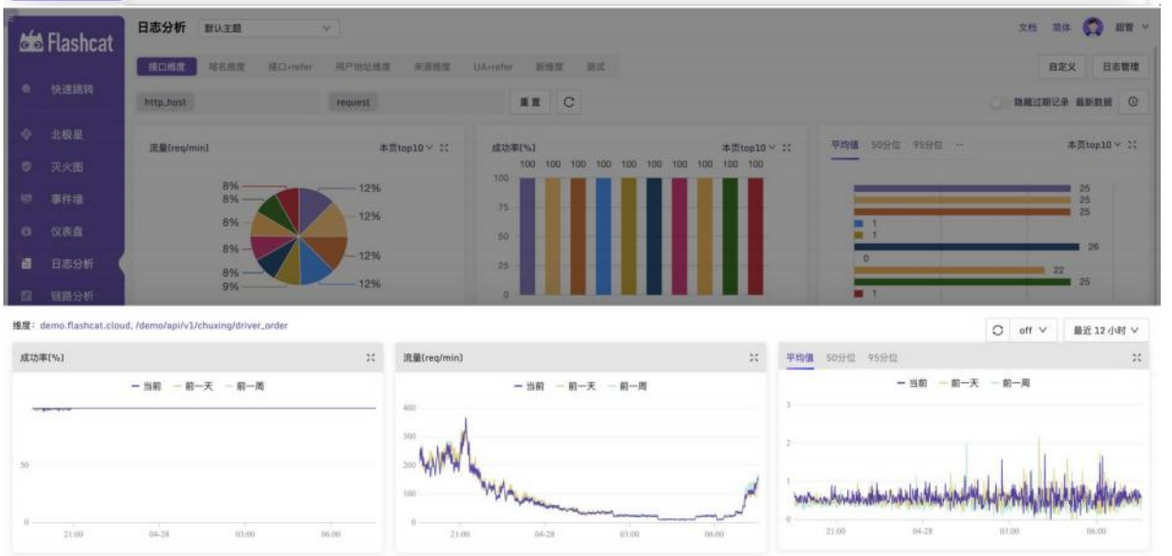
故障定位的方式有很多，没有哪种方式能包打天下，增加一种方式整体定位能力就能增强一分，有点像堆积木，堆积的积木越多，定位能力越强。Flashcat 的定位能力矩阵主要包括：

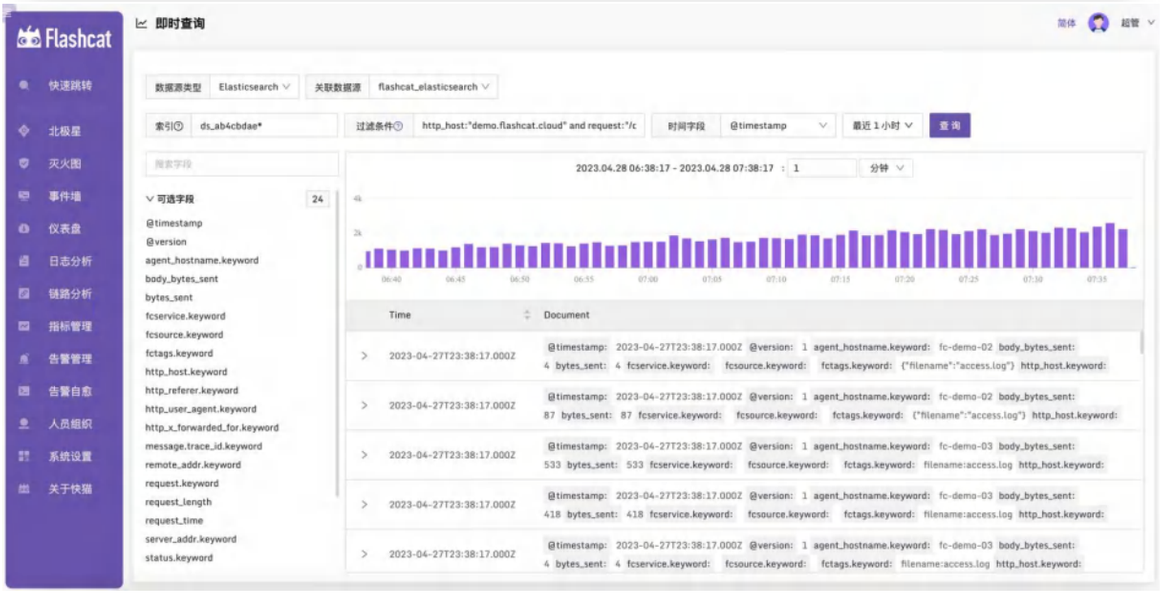
- ✧ 事件墙：用于汇聚变更事件、重要的异常事件，以时间流图的方式呈现，方便分析北极星异常与事件的关联



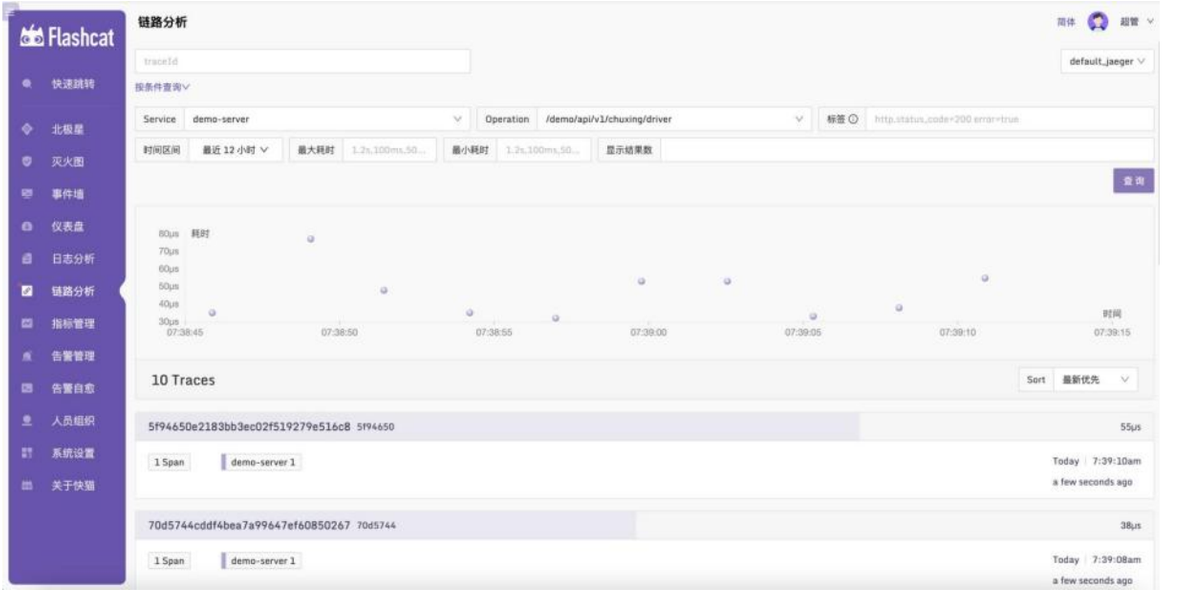
- ✧ 日志分析：采集、提取、格式化、存储日志，并基于格式化的日志自定义各类分析维度的报表和指标，提供分析故障和问题特征的快捷方法。


Flashcat
日志分析
默认主题
文档
简体
超管
接口维度
域名维度
接口+refer
用户地址维度
来源维度
UA+refer
新维度
测试
自定义
日志管理
http_host
request
重置
隐藏过期记录
最新数据
流量(req/min)
本页top10
成功率(%)
平均值
50分位
95分位
http_host
request
流量(req/...
成功率(%)
平均值
95分位
400
403
404
499
demo.flashcat.cloud
/demo/api/v1/zhengquan/chengjiao_zhongjin
/demo/api/v1/zhengquan/chedan_zhongjin
/demo/api/v1/zhengquan/baodan_zhongjin
/demo/api/v1/zhengquan/market
/demo/api/v1/chuxing/driver_order
/demo/api/v1/zhengquan/chengjiao_dalian
快速跳转
北极星
灭火图
事件墙
仪表盘
日志分析
链路分析
指标管理
告警管理
告警自愈
人员组织
系统设置
关于快猫


Flashcat
日志分析
默认主题
维度：demo.flashcat.cloud, /demo/api/v1/chuxing/driver_order
off
最近 12 小时
成功率(%)
流量(req/min)
平均值
50分位
95分位
当前
前一天
前一周


Flashcat
即时查询
简体
超管
数据源类型
Elasticsearch
关联数据源
flashcat_elasticsearch
索引
ds_ab4cbdae*
过滤条件
http_host:"demo.flashcat.cloud" and request:"/c
时间字段
@timestamp
最近 1 小时
查询
2023.04.28 06:38:17 - 2023.04.28 07:38:17
分钟
可选字段
24
@timestamp
@version
agent_hostname.keyword
body_bytes_sent
bytes_sent
fcservice.keyword
fcsource.keyword
fctags.keyword
http_host.keyword
http_referer.keyword
http_user_agent.keyword
http_x_forwarded_for.keyword
message.trace_id.keyword
remote_addr.keyword
request.keyword
request_length
request_time
server_addr.keyword
status.keyword
Time
Document
2023-04-27T23:38:17.000Z
@timestamp: 2023-04-27T23:38:17.000Z @version: 1 agent_hostname.keyword: fc-demo-02 body_bytes_sent: 4 bytes_sent: 4 fcservice.keyword: fcsource.keyword: fctags.keyword: {"filename":"access.log"} http_host.keyword:
@timestamp: 2023-04-27T23:38:17.000Z @version: 1 agent_hostname.keyword: fc-demo-02 body_bytes_sent: 87 bytes_sent: 87 fcservice.keyword: fcsource.keyword: fctags.keyword: {"filename":"access.log"} http_host.keyword:
@timestamp: 2023-04-27T23:38:17.000Z @version: 1 agent_hostname.keyword: fc-demo-03 body_bytes_sent: 533 bytes_sent: 533 fcservice.keyword: fcsource.keyword: fctags.keyword: filename:access.log http_host.keyword:
@timestamp: 2023-04-27T23:38:17.000Z @version: 1 agent_hostname.keyword: fc-demo-03 body_bytes_sent: 418 bytes_sent: 418 fcservice.keyword: fcsource.keyword: fctags.keyword: filename:access.log http_host.keyword:
@timestamp: 2023-04-27T23:38:17.000Z @version: 1 agent_hostname.keyword: fc-demo-03 body_bytes_sent: 4 bytes_sent: 4 fcservice.keyword: fcsource.keyword: fctags.keyword: filename:access.log http_host.keyword:

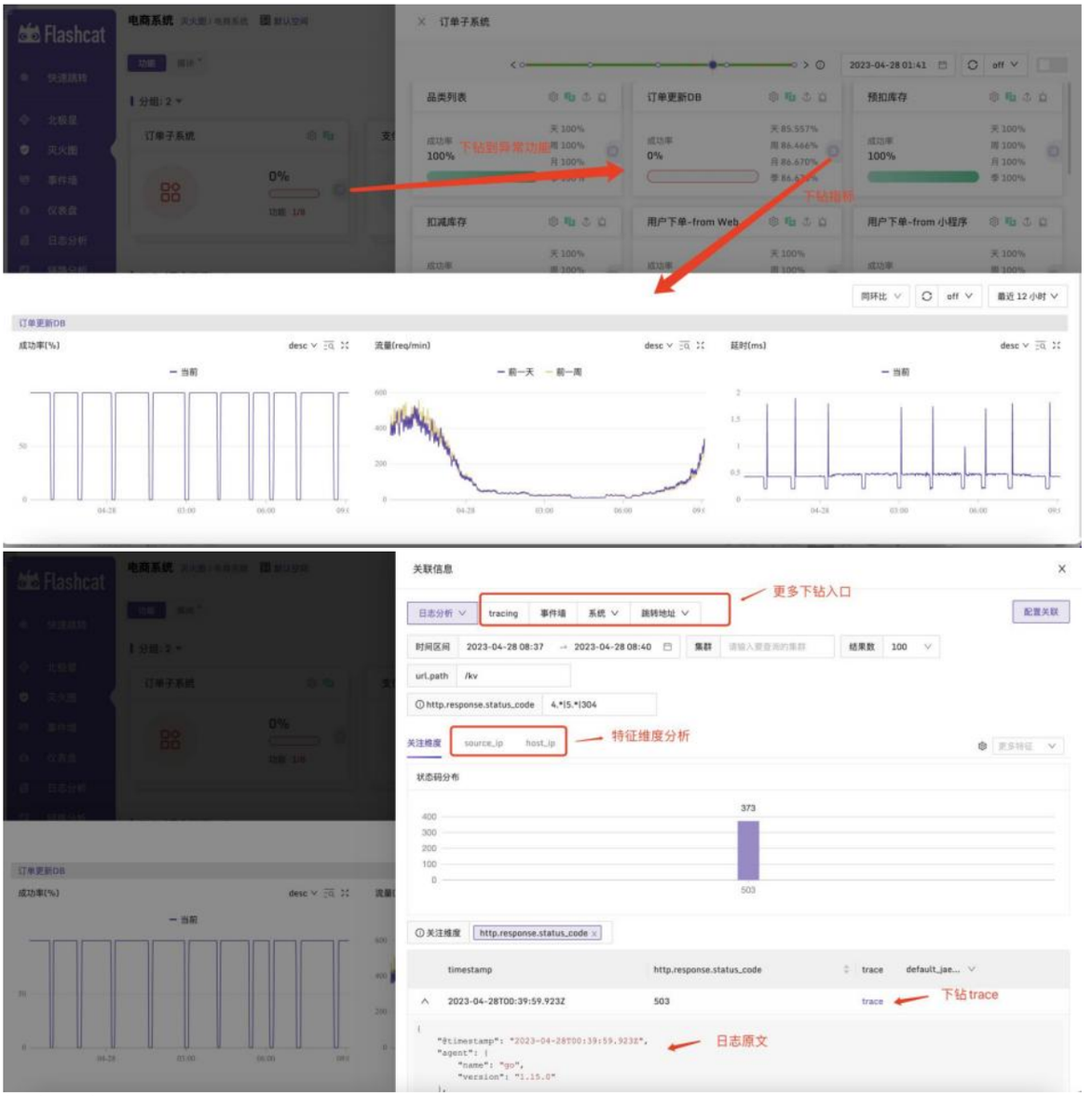
✧ 链路分析：可以方便的接入的 tracing 数据进行查询分析。



关联下钻能力

定位矩阵提供了多种故障分析定位的手段，有了这些能力矩阵后大家会期望这些矩阵之间是能够串联打通的，一方面进一步方便信息的查看分析，另一方面可以相互印证。Flashcat 产品的特点之一即是在真实的故障定位场景中将各类数据打通融合，按照故障定位路径的需要，做到下一步所需信息即时可得。

Flashcat 的关联下钻功能举例：



重要问题及解决思路

Flashcat 结合故障定位场景建设了一个分层的服务健康视图（业务、应用、组件、基础设施），并可以关联这些视图中的元素，以便快速发现和下钻追查问题。这里有两个重要问题需要解决：

1. 微服务的更新变化很快，健康视图如何适应服务的变化，是否能够做到自动生成、自动更新？
2. 下钻关联的元素之间是如何建立起关联的？

Flashcat 产品解决方案：

1. 根据指标 label、日志字段的信息，设置规则自动添加并更新灭火图视图。

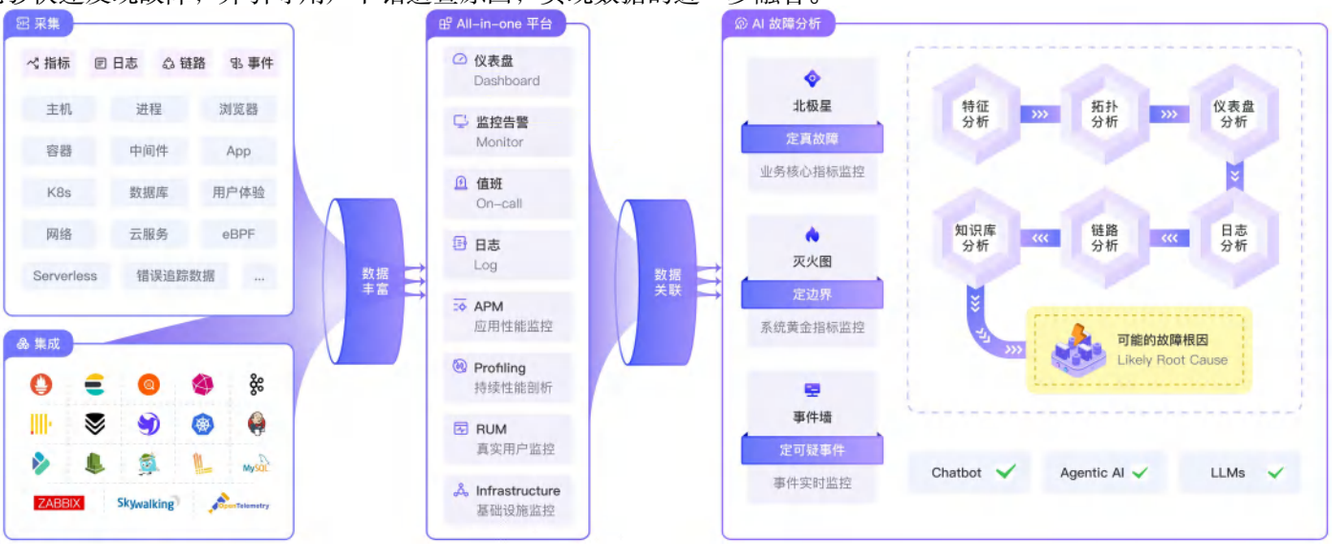
基于以上方案，一种可行的微服务健康视图建设规范：

- ✧ 在指标生成和采集的源头把握好标签和字段的规范
- ✧ 可观测视图根据标签和字段完成动态映射、动态变化

2. 同样，基于指标的 label 和日志字段等信息，辅助手动补偿，也能建立起元素之间的自动关联。

效果

基于以上产品建设，Flashcat 形成了一个从上向下的稳定性保障视角和能力，Flashcat 不仅能够聚合各类观测信息，还能够快速发现故障，并引导用户下钻追查原因，实现数据的进一步融合。



小结

本章节系统的分析了可观测性的历史和现状，并针对当前阶段企业落地可观测能力的问题和挑战进行了具体分析。重点介绍了 Flashcat 产品解决这些问题和挑战的思路，并从可观测性产品建设的三大要素出发介绍了 Flashcat 产品在这三大要素上的思考和具体实现，以及部分难点问题的解决思路。

AI 篇

AIOps 最早可以追溯到 2011 年，Netflix 的“Chaos Monkey 实践”开创了故障注入与自动化修复的先河，之后在 2016 年，Gartner 首次提出 AIOps 的概念：Algorithmic IT Operations，描绘了算法、机器学习、大数据分析在自动化运维领域的重要性和广阔前景。随后的十多年间，众多创新企业、互联网巨头、科研机构，都在孜孜不倦的追求智能运维领域的“AGI 时刻”，不过都未能取得突破性的进展。一些典型的方案，都局限于某个垂直领域和单个场景，比如事件关联分析、时间序列数据预测、日志聚类分析、特征分析等，这些尝试方案，在不同的企业、不同的场景、不同的业务系统之间难以有效迁移、不具备可复制性。

2022 年进入 ChatGPT 时代，大语言模型的智能层次和进化速度，刷新了所有人的认知，2025 年 DeepSeek 时刻，则

开启了大模型“提智降价”的螺旋竞争，足够智能的大模型和可负担的 API 调用成本，充分释放了智能的力量。大模型和智能体在重塑每个行业。

在可观测领域，New Relic 最新的一项调查报告中显示，AI 监控的使用率已经从 2024 年的 42% 提升至 2025 年的 54%。一方面可观测性数据的消费者发生了变化，从面向人转变为面向 AI 智能体，另一方面，可观测系统本身也在快速往智能进化。

针对可观测系统，智能进化可以划分为以下 4 个阶段，当前正在处于 L3 到 L4 之间，最终“智能体”将成为我们信赖的同事，进入“人”与“智能体”协同工作的新时代：**AI SRE** ！

要到达 AI 自主规划、自主分析的目标，一些基本的构成要素分析如下：

- ✧ 要素一：AI 要能够理解你的系统：用观测数据构建企业 IT 系统的知识图谱。
- ✧ 要素二：AI 能够检索你的数据：分析过程必然需要查询数据，数据通道的权限和查询理解、能力是否为具备。
- ✧ 要素三：AI 具备业务和行业的知识基础：观测数据可能是通用的，如何结合企业业务进行分析解读，需要具备业务和行业的基础知识。

要素一：AI 能够理解你的系统

可观测性数据往往是海量的，一个指标系统通常有百万、千万、上亿的指标。一个日志系统动则有百 T 上 P 级的日志文本数据。一个链路系统存储数万、十万、百万的链路数据是常态。

AI 不可能快速消化这些海量数据来识别你的系统，即使从这些海量数据中学习到系统的结构，也需要存储为一个知识图谱，以作为每次分析的基础。这样一个知识图谱是 AI 能够理解企业 IT 系统的关键，一个 IT 系统的知识图谱具体应包括：

- ✧ 观测对象的描述：首先，需要将 IT 系统抽象描述为观测对象。如网络设备、专线、存储、MySQL/Redis 等标准组件、容器服务、微服务、功能接口等。
- ✧ 观测对象的属性：如名称、标签、相关联的观测数据。如系统的下单接口是一个观测对象，相应的关联观测数据包括实时的流程、成功率、响应时间，以及对应的日志、链路数据等。
- ✧ 观测对象的状态：基于观测数据量化的资产健康状态，标识资产当前的运行状态是否正常。如下单接口的请求成功率低于 99.9%或响应时间高于 100ms 定义为异常，否则正常。
- ✧ 观测对象的依赖：层级依赖、调用依赖等。如一般情况下，网络的异常一定影响上层的组件、微服务、功能接口。组件的异常也一定影响上层调用它的微服务、功能接口。

要素二：AI 能够检索你的数据

AI 的分析过程中一定涉及到按需查询观测对象的观测数据，比如发现一个微服务异常，可能需要查询该微服务的请求量/成功率/响应时间或输出的具体日志信息等等。AI 是否能够便捷的获得或查询到相关的观测数据，决定了 AI 是否能够完成整个智能分析过程。

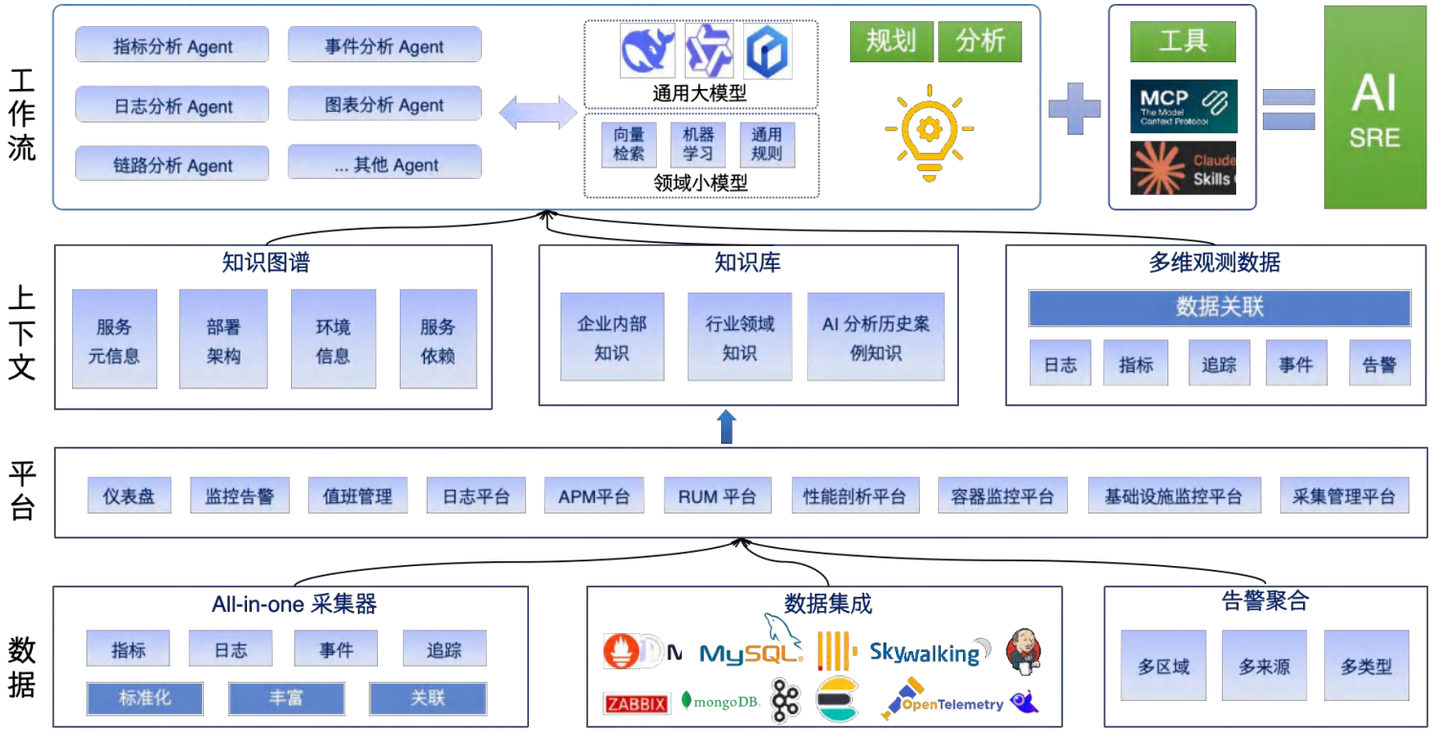
要满足这个条件，则需要打通并管理好企业内部所有观测数据的查询通道，现实数据的按需查询和权限管理。这里有三种常见的方法：

- ✧ 从零开始：从零开始或推倒重来，建设一套完整的观测系统。一般适合从零开始建设观测系统的企业。
- ✧ 同步转储：将不同观测系统的数据转储到一个系统，集中管理和使用。业界较为多见的思路。
- ✧ 集成打通：将不同观测系统的数据查询集成到一个系统，按需通过 API 调用查询数据。业界不常见但其实最为简

便有效的路径。

要素三：AI 具备业务和行业的知识基础

企业的业务往往有一定的个性化和行业的特色，一个 IT 系统上支撑着一个怎样的系统，AI 分析的方向是否需要有补充的知识和侧重？这些信息需要有途径提供给 AI，作为 AI 分析数据基础的一部分。当前这个方向有相对成熟的 RAG（Retrieval-augmented Generation）技术可以实现，但在具体的观测产品中，需要有相应的功能支持。



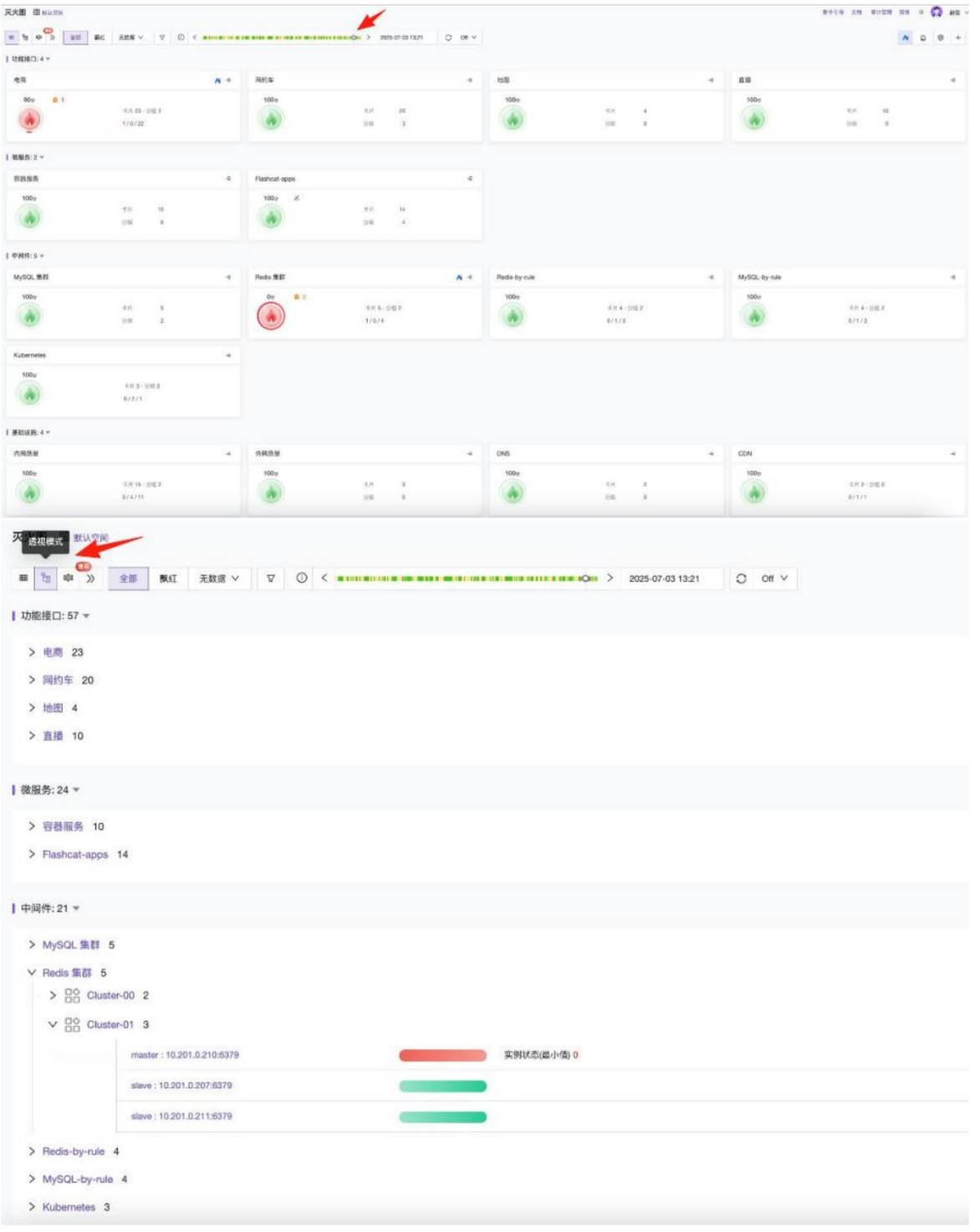
快猫 Flashcat 平台的 AI 实践

灭火图：让 AI 理解你的系统

灭火图就是 Flashcat 系统中的 IT 系统知识图谱。灭火图设计实现的初衷是为了加速人工理解和分析系统，但进入 AI 时代，灭火图自然的成为 AI 理解系统的基础。

灭火图的核心是通过规则，将 IT 系统中的观测对象映射并生成为具体的卡片/节点，通过指标和条件触发卡片的飘红/飘绿来标识观测对象的健康状态。同时 Flaschat 通过模板和映射关系，将观测数据和依赖精准的关联到观测对象上。

当出现问题时，用户只需要从异常对象（卡片/节点）下钻点击即可完成所有相关路径和数据的遍历分析，而不用在各个观测系统间来回切换，也不用关心数据来自哪个观测系统。



插线板：让 AI 便捷检索数据

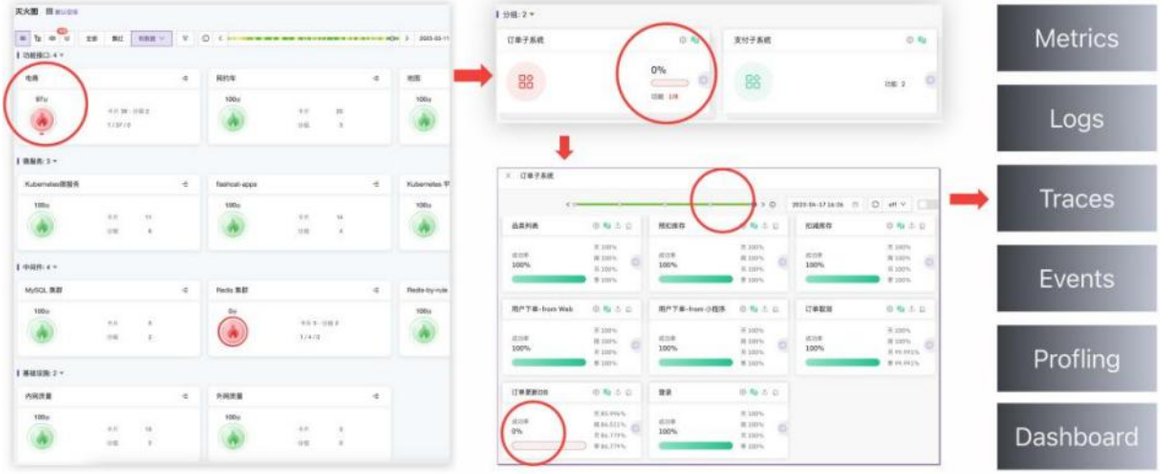
如何将企业内部的 metrics、logs、traces、events 等各个维度的数据封装成统一的接口，暴露给 AI，是核心要完成的工作。在 Flashcat 中，将这些工作分解为 5 个步骤。

1、 集成数据：使用 Flashcat 插线板机制，使用数据源集成的方式，将企业已经拥有的 Prometheus、ElasticSearch、ClickHouse、Doris、Zabbix、MySQL 等，以及公有云提供的各种可观测性工具如阿里云 SLS、阿里云 Arms、AWS Cloudwatch、Azure 云监控等，直接以数据源的方式，注册到 Flashcat 平台中，再经由 Flashcat 统一暴露给 AI agent。Flashcat 目前支持了 40 多种常用的数据源对接。

2、采集数据：如果已经拥有的数据不够完整，那么推荐使用采集器 Categraf 来查漏补缺，对缺失的数据进行采集之后，同样存储到 Flashcat 平台中，再暴露给 AI agent。

3、数据增强：各类可观测性数据，有是一回事，质量高不高是更关键的另一回事。在可观测性领域，“标签（Label）”是一个关键概念，我们可以通过对 metrics、logs、metrics、events 等各维度的数据，打上足够丰富的、一致的“标签”，来表达可观测性数据之间的联系，描述实体的属性、实体之间的交互关系，这个打标签的动作，我们称之为数据增强。数据增强可以发生在数据采集的阶段，我们就可以让采集器和 CMDB、K8s APIServer、公有云的控制器进行交互，为采集到的数据增加更多相关的标签。也可以发生在数据传输的 pipeline 中，对这些数据进行实时的标签提取、转换和映射，从而让可观测性数据的标签更丰富和一致，让 AI 更容易理解。

4、数据预处理：前面三步使得我们拥有了完备的、相对高质量的数据。接下来面临一个现实问题，可观测性数据量往往都很大，一股脑全给大模型分析，单纯靠大力出奇迹，是行不通的。在 Flashcat 里，利用“灭火图”，构建了 CI/CD、Infra、Kubernetes、Metrics、Logs、Traces、Events、Runbook 的完整环境地图，描述了 API、服务、Pods、组件和其他环境之间的交互关系，根据这些交互关系，可以精确的把相关性高的数据，发送给大模型分析，同时这些上下文信息本身也是大模型理解问题的关键输入。此外结合使用日志聚类分析、相似性去重等方法，减少对大模型的数据处理压力。



5、数据导出：Flashcat 采用类似 MCP Server 的方式，把以上各类可观测性数据与上下文环境信息，导出给大模型，未来也可以和企业内部的各种 MCP Client 共享 Flashcat 这些有价值的数据。

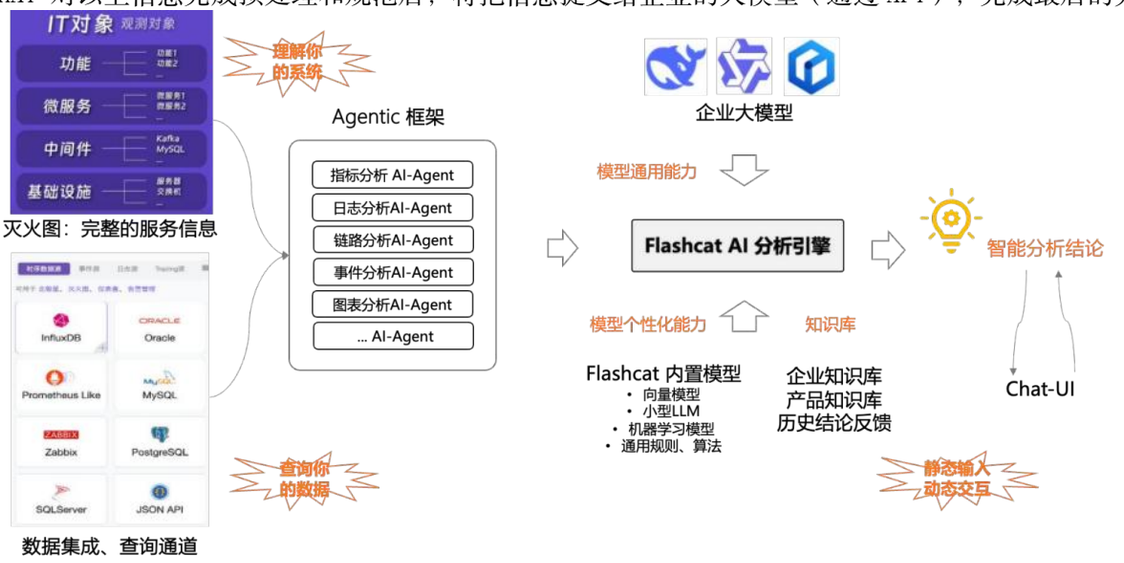
FlashAI 知识库：让 AI 获取业务/行业知识

Flashcat 的 AI 产品叫做 FlashAI，FlashAI 对业务/行业知识的输入有 3 个途径：

✧ RAG：FlashAI 能够通过 RAG 完成对 Flashcat 产品和可观测性及稳定性保障相关知识文档的处理，在企业部署

Flashcat 时，将这部分数据一并部署到环境中。

- ✧ 知识库按需输入：FlashAI 提供了自定义创建知识库的能力，作为 FlashAI 分析时的知识输入。后续还将实现和企业内部文档系统的直接集成。
- ✧ 动态交互输入：和常见的大模型产品一样，FlashAI 提供了和大模型交互的对话框，可以灵活的补充或修正 AI 的输出。
- ✧ FlashAI 对以上信息完成预处理和规范后，将把信息提交给企业的大模型（通过 API），完成最后的分析输出。



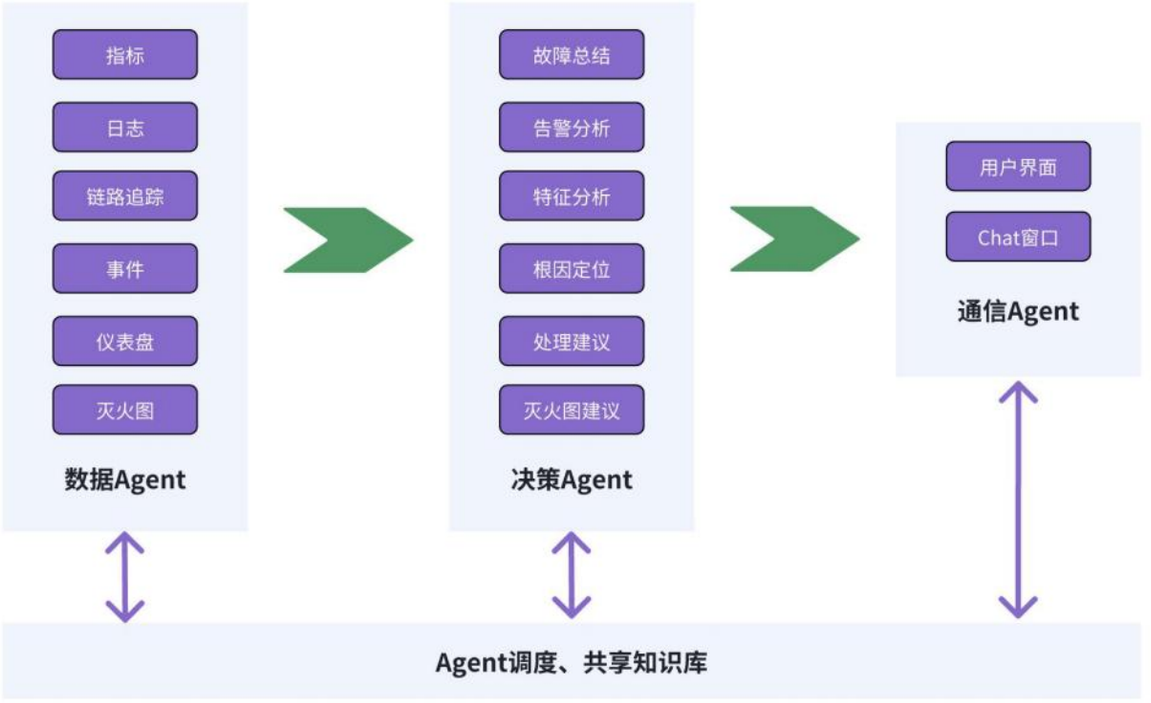
原理

FlashAI 已经学习了 Flashcat 相关的概念、产品文档，积累了在可观测、运维等领域的常见知识、特征库，并且已经和 Flashcat 对接的各类数据源进行了数据打通。通过相关的数据接口，结合大模型的能力，可以理解用户的指标、日志、事件等数据的含义，在需要时给出准确的分析建议。

FlashAI 依赖两个组件：

- ✧ Flashcat 私有化模型：Flashcat 针对自身场景的定制化模型。Flashcat 部署时自带；
- ✧ 基座大模型：用户环境的通用大模型，如 Deepseek、通义千问、OpenAI 等。可通过数据集成直接集成到 Flashcat；

其工作流程可以概括如下：



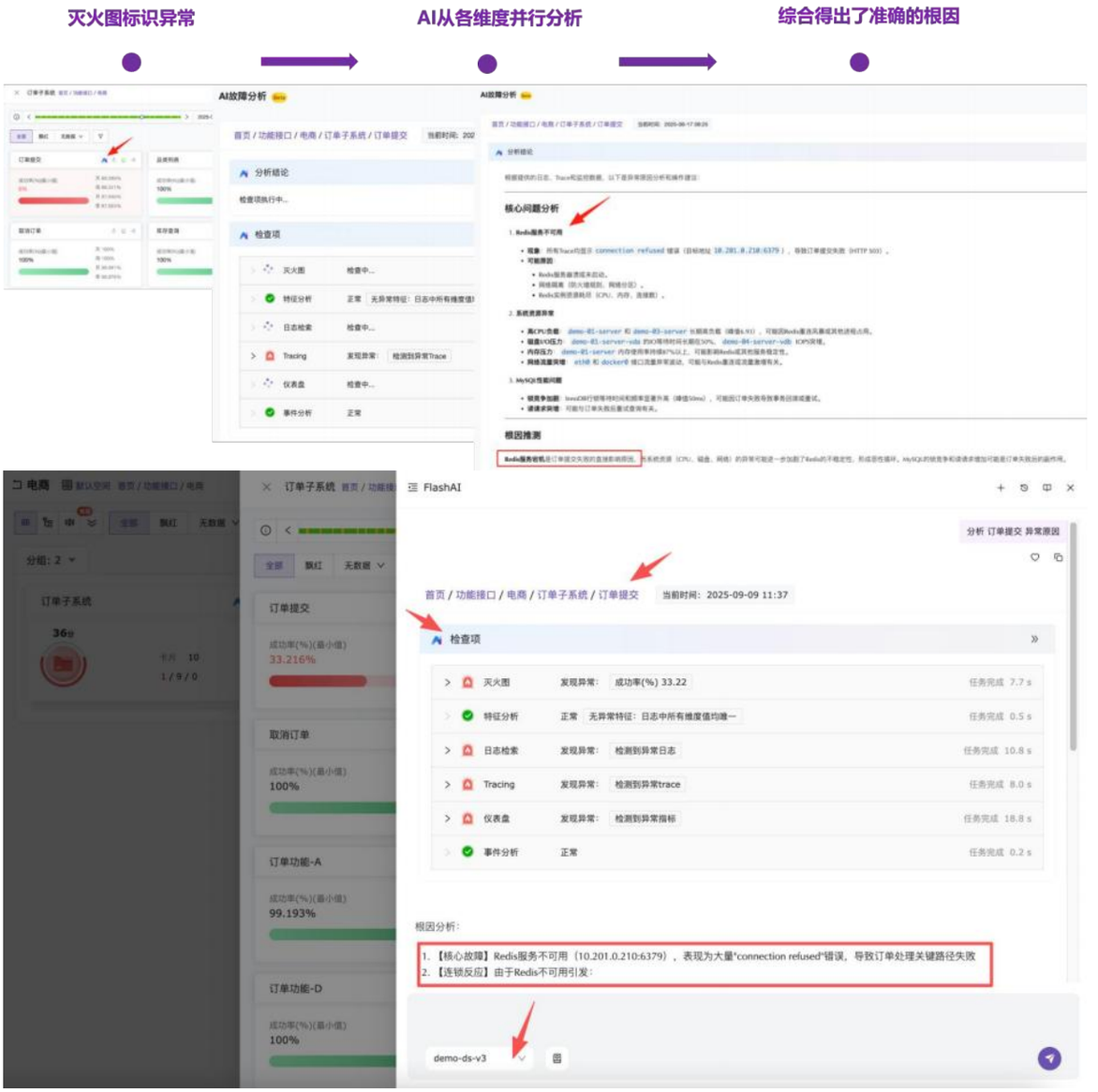
- ✧ 数据 Agent： 主要负责工具调用、数据拉取和分析，基于各个下钻项分析故障疑点；
- ✧ 决策 Agent ： 负责汇总数据、生成最终的故障定位结论、处理建议、巡检报告等功能；
- ✧ 通信 Agent ： 支持用户界面交互，允许用户通过页面反馈，动态调整或重新生成分析结论；
- ✧ 调度层：基于用户反馈和历史 case 维护共享知识库，支持 RAG 方式增强结论准确性；

可以将 FlashAI 看待成如下角色：

- ✧ 「SRE 专家」：发生故障时，FlashAI 可以基于灭火图，进行故障分析诊断，给出故障原因分析。日常运维工作中，FlashAI 可以提供日常巡检，给出 AI 巡检报告以及治理建议；
- ✧ 「架构师」：在规划可观测性建设方案时，可以和 FlashAI 交互，FlashAI 会提供专业的建设意见。比如相关最佳实践、灭火图建设方案等等；
- ✧ 『技术支持人员』：在日常工作中，FlashAI 可以做为 Flashcat 的技术支持人员，提供相关的答疑工作，比如数据如何采集，告警如何配置等等；

效果

通过以上三步的实现，Flashcat 具备了为 AI 提供理解 IT 系统的知识图谱、提供了观测数据的查询通道、提供了补充业务信息的能力。实践中，FlashAI 已具备达成 L4 级立体分析的能力，在内部的放火演练中，FlashAI 能够重复正确的指出系统异常的根因，并在部分用户的生产环境中得到了验证。



总结

数据建设是智能化的核心，大模型是智能化的催化剂。总结起来，在可观测性实现智能化的路上，我们首先要重视数据建设工作，寻找合适的产品和技术来支持可观测性数据建设，以达到 AI-Ready 状态。在此基础上加上大模型作为最后一公里的催化剂，以实现智能化的真正飞跃。最终“智能体”将成为我们信赖的同事，进入“人”与“智能体”协同工作的新时代：**AI SRE** ！

关于快猫星云

快猫星云是智能时代的全栈可观测性解决方案提供商，产品涵盖指标、日志、APM 、RUM、监控告警和 On-call，基于 AI 驱动故障发现、根因分析、事件响应和性能优化，帮助企业显著降低 MTTR，保障业务稳定运行，持续提升用户体验。

- ✧ 快猫星云 Flashcat 是以开源夜莺为内核打造的全栈观测平台，支持指标、日志、链路追踪数据的统一采集、存储、监控告警、可视化分析，只需一个 Flashcat 平台即可全面覆盖云上、云下、Kubernetes 的观测需求。Flashcat 预置了行业领先的故障发现定位最佳实践，并深度使用 AI 加速故障的分析过程，大幅缩短故障恢复时间。
- ✧ 快猫星云 Flashduty 是一站式告警事件响应平台，支持告警聚合、降噪、排班、协作，内置飞书/钉钉/企微/短信/电话等多种通知方式，在 IM 和 App 中响应和处理告警，全生命周期管理告警事件、分析告警数据，提升

On-call 体验，缩短 MTTA/MTTR。

✧ 快猫星云 RUM 能够追踪并分析真实用户在使用 Web 应用或者 App 时的实际体验。与传统模拟测试不同，RUM 直接从用户浏览器或者 App 采集数据，呈现应用在真实环境中的运行状况，如性能、错误、交互等。RUM 的会话重放，支持全程记录用户点击、滚动、输入等操作，结合控制台与网络请求，完整还原现场，助力问题精准复现。

联系我们:

- 官方网站: https://flashcat.cloud
- Flashcat 产品介绍 PPT: https://fcpub-1301667576.cos.ap-nanjing.myqclouddownload.flashcat.cloud/flashcat.pdf
- 电子邮件: contact-us@flashcat.cloud
- 微信: picobyte

版权说明