SRE 实践白皮书

v1.0.6





2025年7月

SRE-Elite.com

修 订 记 录

1.0.6 修订记录：

本次新增约六万字

- **新增了第四章 “SRE 进阶”，**供 SRE 管理者参考，相关模块包括了“SRE 团队的生存与持续发展”“SRE 团队的综合算力调度”“SRE 团队的货币化改革”“SRE 货币化后的持续发展”，其中本次发布“SRE 团队的生存与持续发展”，其他模块后续奉上
- **章节架构调整：**将原第二章 SRE 组织架构 并入第一章；新增第二章 SRE 的基础（编写重）。”
- **第三章第 5 节《故障应急》：**结构进行了全面优化，新增 “xx 银行”，“腾讯 IEG”，“小米米家”，“广发证券”等多个案例，并对 原“美图”案例展望部分进行了更新

1.0.5 修订记录：

本次新增约三万字

- 第三章第 5 节《故障应急》：结构进行了全面优化，新增 B 站和蚂蚁的两个案例
- 调整“应用服务 SLI/SLO/SLA” “重大技术保障”章节至本章，并在后者新增《OPPO 春节业务保障》案例；
- 大幅完善故障应急体系章节内容

1.0.4 修订记录：

- 第三章第 2 节《研发保障》结构进行了优化，并增加了《某大型游戏全球研发保障实践》等合共 2 个案例，新增 2.7 万字。
- 第三章第 5 节《故障应急》结构进行了优化，依据 2024 年 6 月 22 日北京小米站沙龙更新并增加了《小米故障应急响应经验分享》等合共 5 个案例，新增 4.5 万字。

1.0.3 修订记录：

- 第三章第 4 节《变更管理》依据 2024 年 4 月 13 日上海 B 站沙龙更新约 4 万字，包括 6 篇不同类型的企业案例

1.0.2 修订记录：

- 增加了版权声明 为 CC BY-ND 4.0
- 修正了目录没有 3.1.1 的问题
- 修改了页眉的时间点
- 修正了部分错别字

目 录

第一章 SRE 整体介绍 1
1.1 前言 1
1.2 SRE 发展历程 2
1.3 SRE 的目标 4
1.4 SRE 的组织架构 5
第二章 SRE 的基础(编写中) 9
第三章 SRE 的职能 10
1 可靠性架构设计 10
1.1 应用韧性架构 11
1.1.1 分布式设计 11
1.1.2 解耦设计 11
1.1.3 冗余设计 11
1.1.4 熔断设计 12
1.1.5 限流设计 12
1.1.6 降级设计 13
1.1.7 可观测设计 13
1.2 基础设施保障 14
1.2.1 机房多活 14
1.2.2 网络容灾 14
1.3 数据灾备 14

1.3.1 数据备份 14
1.3.2 数据回滚 14
2 研发保障 15
2.1 研发保障体系设计 16
2.1.1 代码可靠性 16
2.1.2 代码仓库可靠性 28
2.1.3 构建可靠性 34
2.1.4 制品可靠性 38
2.2 研发保障工程体系设计 42
2.2.1 面向研发保障的持续集成流水线 42
2.2.2 面向研发保障的可观测设计 46
2.2.3 面向研发保障的操作调度操作平台 48
2.2.4 面向研发保障的 ITSM 平台 51
2.2.5 面向研发保障的容器平台 51
2.2.6 面向研发保障的编译加速平台 53
2.3 研发保障案例 55
2.3.1 腾讯游戏全球研发保障实践 55
2.3.2 某语音直播公司研发过程保障实践 128
3 入网控制 151
3.1 运行环境适配 151
3.1.1 运营环境设计 151
3.1.2 容器云适配 153

3.1.3 数据库存储适配 157
3.1.4 信创适配 158
3.2 运行环境交付 163
3.2.1 基础资源服务 163
3.2.2 可观测策略 164
3.2.3 自动化策略 167
3.3 测试策略 169
3.3.1 连通性验证 169
3.3.2 功能测试 171
3.3.3 性能压测 174
3.3.4 数据迁移 179
3.4 变更评审 180
3.4.1 稳定性架构设计评估 181
3.4.2 非功能性技术评估 183
3.4.3 变更保障准备工作评估 186
3.4.4 新系统或新业务上线保障评估 187
4 变更管理 189
4.1 发布管理与变更管理关系阐述 190
4.2 变更体系设计 192
4.2.1 变更体系设计原则 192
4.2.2 变更及发布流程设计 193
4.2.3 变更的工程体系设计 217

4.3 变更管理案例 247
4.3.1 B 站变更防控的设计与实践 247
4.3.2 携程云平台基础设施变更管理实践 270
4.3.3 某银行变更管理设计与实践 292
4.4 发布管理案例 312
4.4.1 中移互联网敏捷发布平台建设实践 312
4.4.2 某证券变更一体化平台建设实践 332
4.4.3 游戏 GitOps 发布管理实践 350
5 故障应急 358
5.1 故障体系 358
5.1.1 故障定义 358
5.1.2 故障等级 358
5.1.3 故障序列 359
5.1.4 关键业务场景 360
5.1.5 应用服务 SLI/SLO/SLA 362
5.2 故障应急体系 370
5.2.1 故障应急体系设计原则 370
5.2.2 故障应急流程设计 370
5.2.3 故障应急工程体系设计 398
5.3 故障应急案例 411
5.3.1 小米故障应急响应经验分享 411
5.3.2 中国联通数字化监控平台稳定性保障实践 440

5.3.3 腾讯全球化游戏故障管理实践 474
5.3.4 XX 银行应急管理一体化平台建设实践 513
5.3.5 美图故障管理体系搭建实践 527
5.3.6 B 站轻量级容灾演练体系构建 587
5.3.7 蚂蚁故障应急全流程体系构建及应用实践 625
5.3.8 xx 银行应急提升实践：深耕 1-5-10 工程 662
5.3.9 腾讯 IEG SRE 应急响应实践 700
5.3.10 小米米家故障应急保障体系实战：稳中求胜的构建之道 762
5.3.11 广发证券数智化重构故障管理：构建主动防御新体系 800
6 上线后持续优化工作 868
6.1 用户体验优化 868
6.1.1 基于用户端的直接用户体验优化 868
6.1.2 基于系统端的间接用户体验优化 869
6.2 运维琐事的日常管理及优化 872
6.2.1 运维琐事的介绍 872
6.2.2 运维琐事的质量管理 874
6.3.3 运维琐事的效率管理 875
6.3 业务全生命周期工具建设 877
6.3.1 研发期工具建设 878
6.3.2 上线期工具建设 879

6.4.3 运营期工具建设 880
6.4.4 下线期工具建设 881
6.4 运营成本分析及优化 882
6.4.1 运营成本分析及优化的必要性 882
6.4.2 运营成本实时监控 882
6.4.3 运营成本分析及优化的指标 883
6.4.4 运营成本的统计及分析方法 885
6.4.5 运营成本的优化方法 888
6.5.5 运营成本优化持续运营 891
6.5 持续改进 893
6.5.1 效率持续改进 893
6.5.2 质量持续改进 894
6.5.3 安全持续改进 896
6.5.4 人员能力持续提升 897
6.5.5 流程持续改进 899
7 平台工程 901
7.1 标准应用平台工程建设 901
7.1.1 应用元信息平台 902
7.1.2 统一资源供给 905
7.1.3 持续集成 907
7.1.4 持续部署 910
7.1.5 部署编排 913

7.1.6 可观测 917
7.1.7 成本（定价、用量、出账） 918
7.2 异构应用平台工程建设 921
7.2.1 总体设计 922
7.2.2 aPaaS 结构设计 922
7.2.3 iPaaS 结构设计 929
7.2.4 通用原子设计 931
7.2.5 SaaS 分级 937
7.2.6 服务管理 940
7.2.7 安全与审计 942
第四章 SRE 的进阶 947
1 SRE 团队的生存与持续发展 947
1.1 SRE 团队的生存法则 948
1.1.1 没有 SRE 岗位行不行 949
1.1.2 SRE 团队的预算从哪里来 950
1.1.3 SRE 上一级领导的视角 951
1.1.4 SRE 团队的生存法则总结 952
1.2 SRE 团队的阵地与产能 958
1.2.1 SRE 团队的阵地 959
1.2.2 SRE 团队的产能提升 960
1.2.3 SRE 团队的工程经济学思维 961
1.3 SRE 团队的货币化改革 962

1.3.1 SRE 团队的货币化方案 963
1.3.2 SRE 团队货币化的管理 965
1.3.3 SRE 团队的货币化工具 967
1.4 SRE 货币化后的持续发展 968
1.4.1 优化服务项的“商业”模式 969
1.4.2 SRE 内部的精细化管理 972
1.4.3 持续扩展货币化收入 974
2 SRE 团队的综合算力调度 (校对中) 976
3 SRE 团队的数据与 AI 场景(编写中) 976
4 SRE 团队的基础监控与应用观测 (编写中) 976
附录 976
1 参考文献 976
2 术语 976

版权声明

这项作品采用 CC BY-ND 4.0 许可进行授权。要查看此许可的副本，请访问 http://creativecommons.org/licenses/by-nd/4.0/

CC BY-ND 4.0 DEED

署名-禁止演绎 4.0 国际

您可以自由地：

共享 — 在任何媒介以任何形式复制、发行本作品 在任何用途下，甚至商业目的。

只要你遵守许可协议条款，许可人就无法收回你的这些权利。

惟须遵守下列条件：

署名 — 您必须给出 适当的署名，提供指向本许可协议的链接，同时 标明是否（对原始作品）作了修改。您可以用任何合理的方式来署名，但是不得以任何方式暗示许可人为您或您的使用背书。

禁止演绎 — 如果您 再混合、转换，或者基于该作品创作，您不可以分发修改作品。

没有附加限制 — 您不得适用法律术语或者 技术措施 从而限制其他人做许可协议允许的事情。

声明：

您不必因为公共领域的作品要素而遵守许可协议，或者您的使用被可适用的 例外或限制 所允许。

不提供担保。许可协议可能不会给予您意图使用的所必需的所有许可。例如，其他权利比如 形象权、隐私权或人格权 可能限制您如何使用作品。

第一章 SRE 整体介绍

1.1 前言

Google 在 2003 年启动了一个全新的团队——“SRE 团队”，该团队旨在通过软件工程的方法提高应用系统的可靠性；随着 SRE 相关理论和实践在 Google 的日臻成熟，SRE 实践也从 Google 慢慢地扩散到了整个行业。自从 SRE 的理念进入中国以来，就已经引起了很多企业的关注和效仿，但各企业实施 SRE 的方法各异，SRE 的实现效果也各不相同。与此同时，中国的互联网行业中涌现出了一批对 SRE 充满热情的倡导者，他们为社区做出了各种贡献；包括：孙宇聪翻译出版了《SRE：Google 运维解密》、赵成在极客时间开设了课程《SRE 实战手册》，以及赵舜东在社区里积极地布道分享等等，不胜枚举。

2022 年，由赵成等人牵头，首批来自互联网、运营商、金融等行业领军企业的 SRE 团队负责人齐聚一堂，组织了 SRE 研讨社区，定期开展社区分享活动，共同探讨 SRE 在各企业里的发展路径，分享各自的实战经验，并总结出了这份来自一线实战的、详实而持续更新的《SRE 实践白皮书》。社区每年都吸纳新的成员，逐年更新本白皮书内容，力求真实客观地描述国内企业 SRE 团队的工作方式。在《实践白皮书》初稿长达两年的整理过程中，我们看到了不同企业对 SRE 的理解，并尽可能统一大家对相似场景的定义；

网址：SRE-Elite.com 微信：SRE 精英联盟

我们看到了不同企业对SRE 职能领地的扩展，并将成功团队的经验提炼成案例供大家参考；我们也看到了在这两年的编写过程中，不同企业SRE 团队的真实变化，并及时将其更新到实践白皮书中。总之，在未来的每个季度，我们都会将各SRE 团队的最新职能、组织形式、技术迭代等现状，补充到《实践白皮书》中。

2023 年，中国信息通信研究院（下简称信通院）云计算与大数据研究所（下简称云大所）稳定性保障实验室的专家加入了SRE 研讨社区，深入地参与到社区交流当中，为《SRE 实践白皮书》的编写工作提供了专业指导。

1.2 SRE 发展历程

SRE 运动在全球的发展经历了20 年，下面是部分重要事件：

2003 年，Google 成立了第一个SRE 团队；

2010 年，Facebook 拥有了一个SRE 团队；

2014 年，USENIX 协会主办的首届SREcon（网站可靠性工程会议）在美国举行，大会成为SRE 专业人士交流经验和最佳实践的重要平台，标志着SRE 作为一个独立且重要的专业领域在全球范围内的正式认可。

2016 年，前Google SRE 孙宇聪翻译出版了首部中文专业书籍《SRE：Google 运维揭秘》，在国内引起了很大的反响，很多企业开始学习并成立自己的SRE 团队；

2016 年，Netflix 成立了“核心 SRE 团队”。Uber 开始撰写有关其如何使用 SRE 的文章；

2016 年，蚂蚁集团在国内成立了第一支 SRE 团队，主要攻坚容灾架构，后续拓展到高可用、资金安全等多个业务风险领域；

2017 年，LinkedIn 开始宣传其“ SRE 文化”；

2017 年，浙江移动正式组建应用 SRE 团队，开始收口 IT 系统的集成部署、应急保障、架构治理等工作职责，加速了传统企业的运维数字化的转型进程；

2018 年，赵成在某次 SRE 的聚会上，拉起了 “聊聊 SRE”微信群，国内 SRE 人才开始聚拢，SRE 社区初步成型，并逐步成为最具影响力 SRE 中文社区；

2021，阿里 CTO 线第一支横向 SRE 团队成立，隶属于技术风险与效能部，负责集团全局稳定性保障、资源成本等方面工作；

2022 年，腾讯在内部技术岗位设置中，新增了 SRE，标志着腾讯内部 SRE 体系的正式成立；

2023 年，信通院云大所稳定性保障实验室牵头编制《服务韧性工程（SRE）成熟度模型》标准，推动该领域深入研究与实践应用，并在稳定性保障实验室成立了专门的“SRE 工作组”。

1.3 SRE 的目标

Site Reliability Engineering（SRE）的主要目标是通过结合软件工程和系统运维的最佳实践，提高大规模分布式系统的可靠性、可用性、性能和效率。以下是部分 SRE 追求的核心目标：

可靠性：SRE 的首要目标是确保服务和系统的可靠性。这包括减少故障、提高系统的稳定性，以确保用户在任何时候都能够获得一致的高质量服务。

可扩展性：SRE 致力于设计和实施能够随着用户需求增长而扩展的系统。这涉及对系统的架构和资源进行优化，以便在不降低用户体验的情况下，适应实际工作负载持续不断的峰谷状态变化。

性能：SRE 关注系统的性能，旨在确保系统能够在合理时间内快速响应用户请求。这包括对系统瓶颈的持续监控和优化，以提高整体性能。

自动化：SRE 倡导自动化运维工作，以减少人为错误和提高效率。通过自动化，可以更快速地部署新功能、检测并响应故障，并合理地开展系统的升级和维护工作。

监控和告警：SRE 强调对系统的全面监控，以便及时发现并解决问题。通过设置有效的告警系统，可以在重大问题发生前迅速做出反应，从而减少对用户的影响。

故障恢复：SRE 强调迅速而有效地恢复服务，以最小化用户体验的中断。这包括制定完善的应急计划并持续演练。

企业实现 SRE 核心目标的过程并不相同，落地路径各异。不论 SRE 部门（团队）在企业中的存在形式和所处位置，SRE 相关实践工作存在于大量流程中。这些工作流程与研发、测试、运维、产品运营等团队紧密地融合在一起，所有参与团队都在上述共享的 SRE 目标上做着各自的贡献。

1.4 SRE 的组织架构

SRE 团队在组织中的存在意义主要是确保系统的可靠性和高效运行。通过引入 SRE 角色，组织可以更好地实现开发软件开发速率和系统稳定性之间的动态平衡，从而实现更高水平的可用性、性能和自动化。通常 SRE 团队在组织中使命如下：

可靠性优先：SRE 团队致力于确保服务的高可用性和可靠性。他们关注系统的稳定性，采取工程化方法来减少故障和提高系统的稳定性。

自动化运维：SRE 团队推动自动化运维工作，以减少手动操作的错误和提高效率。通过自动化，可以更快速、可靠地进行部署、监控、故障检测和修复等操作。

质量保证：SRE 团队参与服务的全生命周期，包括设计、开发、部署和维护阶段，以确保系统在不同阶段都能保持高质量。

快速创新：SRE 通过标准化的可靠性基础设施（如自动化容灾、内建可观测能力）以及预防性工程实践，协同开发团队将运维负担转化为创新资源：一方面以可量化的风险护栏显著降低试错成本，另一方面借助自动化能力释放开发精力，从而系统性地加速高质量创新。

在组织架构中，SRE 团队的存在形式可以各不相同，这主要取决于组织的规模、业务需求和文化。以下是一些常见的 SRE 团队的存在形式：

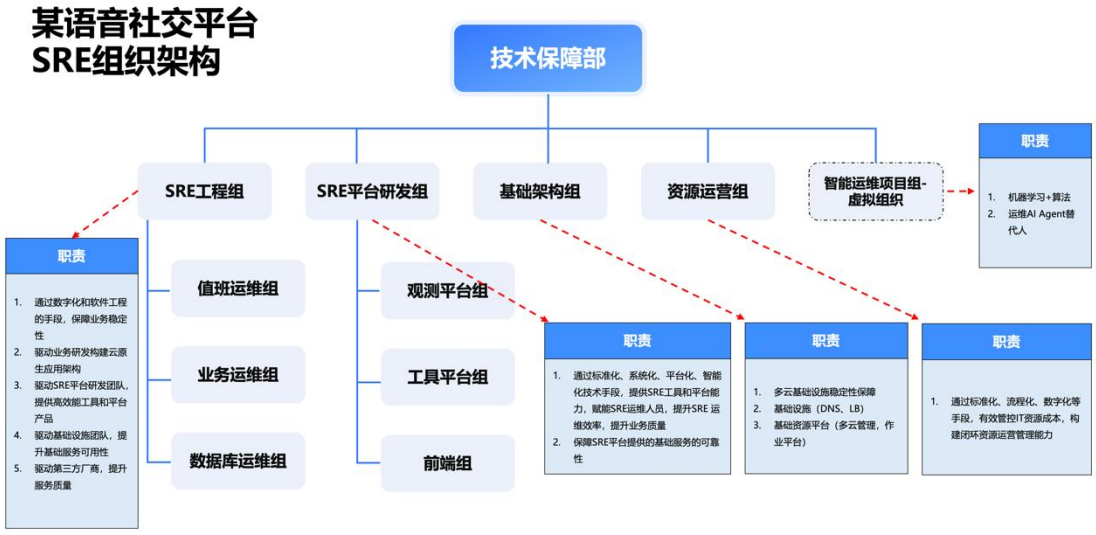
中心化 SRE 团队：由一个专门的 SRE 团队负责支持整个组织的可靠性工作。这种模式有助于集中专业知识，确保在整个组织中实施一致的最佳实践。

嵌入式 SRE 团队：SRE 团队成员被嵌入到各个产品或服务团队中，与开发团队紧密合作。这种模式有助于更好地集成可靠性工作到产品开发的全过程中。

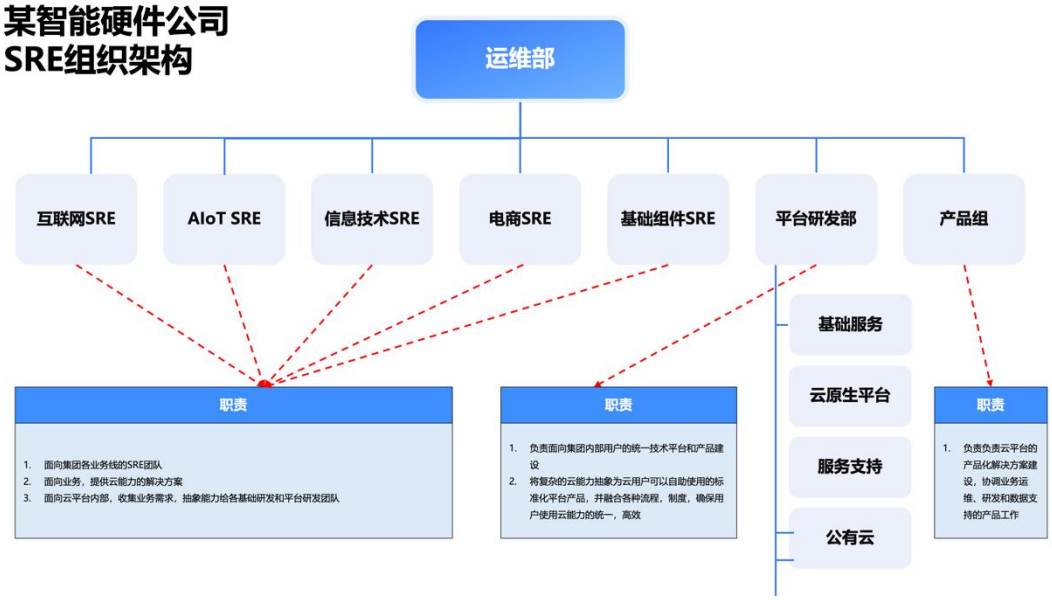
混合模式：一些组织采取混合模式，既有中心化的 SRE 团队，又在一些关键项目中嵌入 SRE 角色。这种方式能够兼顾专业化和贴近业务的优势。

每种存在形式都有其优势和适用场景，关键在于根据组织的需求选择最合适的模式。不论哪种方式，SRE 的目标都是通过自动化和工程方法提高系统的可靠性和效率。

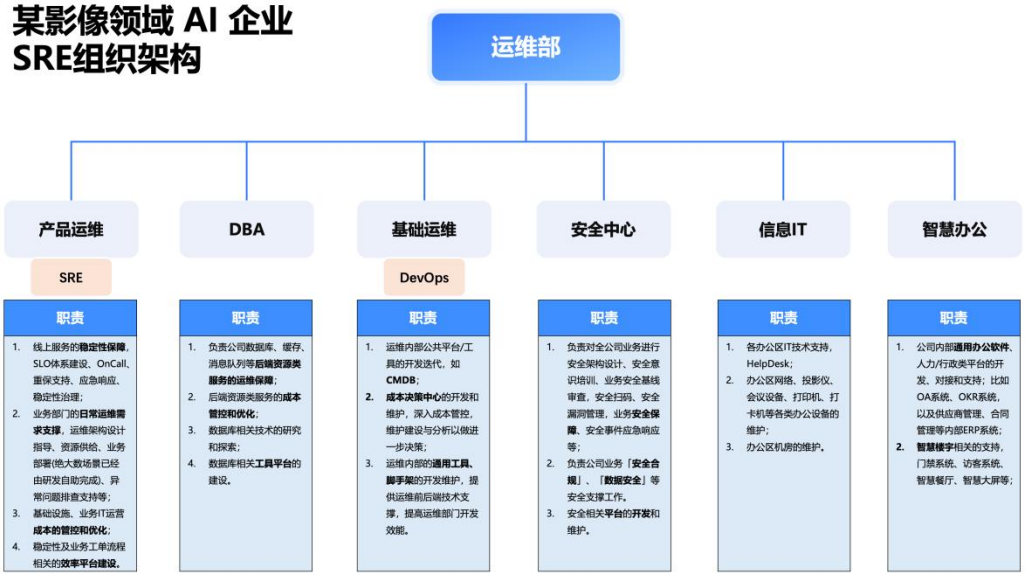
下面是国内某几家一线互联网 SRE 团队在组织架构中的设置模式。

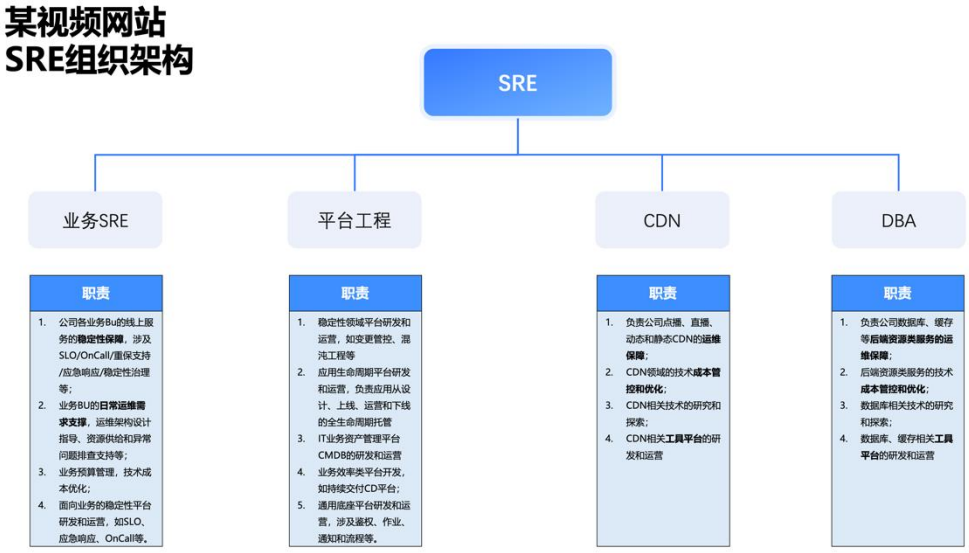


某智能硬件公司 SRE组织架构



某影像领域 AI 企业 SRE组织架构





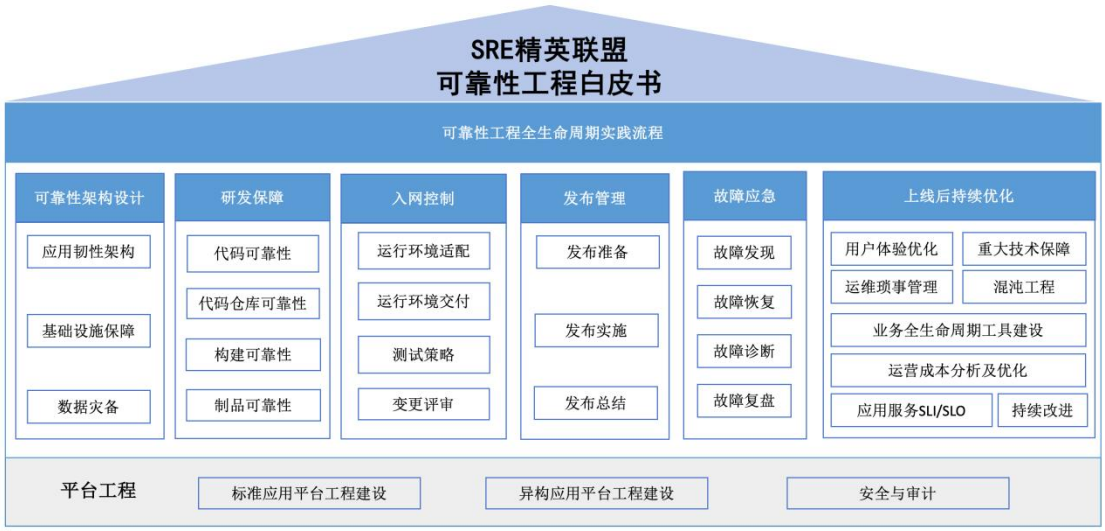
参考以上各个公司 SRE 团队在组织架构中的位置，通常 SRE 团队需要承担以下几项职责：监控、事故响应、事后回顾、测试与发布、容量规划、工具开发和可用性改进等。

由于各个公司的业务形态的不同，SRE 团队在组织架构中也有不同的定位和名称，包括：SRE 产品运维、互联网 SRE 组、AIoT SRE 组、信息技术 SRE 组、业务 SRE 组等。

第二章 SRE 的基础（编写中）

第三章 SRE 的职能

1 可靠性架构设计



可靠性架构设计是指在进行系统架构设计的过程中，根据系统的可靠性需求，采用分布式设计、解耦设计、冗余设计等高可靠性的架构设计方案，以提升系统的可靠性。

在进行可靠性架构设计的过程中，SRE 团队需要将应用架构设计流程完全融入其中，并与研发团队共同参与架构设计和评审工作。在系统设计阶段，应尽量消除可能出现的单点、依赖、容量等潜在风险，并提前为可能出现的系统架构风险做好应急准备。

1.1 应用韧性架构

1.1.1 分布式设计

在系统中，存在被划分为职责明确、粒度合适且易于管理的组件，这些组件（如计算资源、业务部分、数据等）都可以进行分布式的部署和运行。组件之间相互独立、互不干扰，通过分布式设计可以提高开发效率和可靠性。组件的拆分和分布可以通过复制、根据功能进行垂直拆分或根据用户访问模式水平拆分等形式。

在设计时应该充分考虑到组件间可能存在的相互干扰以及如何平衡不同组件之间的负载，并将系统所承受的压力进行均匀分配，以减轻压力对系统整体性能的不良影响。

1.1.2 解耦设计

在架构设计中，可以将各种逻辑功能划分为不同的服务模块，确保不同模块的故障对其他模块的影响是最小，从而最大限度地降低模块之间的耦合度。通过这种方式，可以将系统划分为多个相互独立的功能模块来实现。尤其值得注意的是，业务的主要逻辑与其他非核心模块是独立的，因此业务非核心模块的故障并不会对业务的核心功能产生负面影响。

1.1.3 冗余设计

为了确保资源有足够的安全余量，每个组件都需要有足够和合理的冗余实例，以确保单一组件实例的失效不会对业务的正常运行造成影响。对于不同类型的组件，我们需要明确地定义冗余量和冗

网址：SRE-Elite.com 微信：SRE 精英联盟

余类型。在实际应用中，由于设备故障或者操作不当等原因导致服务器出现性能下降或崩溃现象时，系统会出现异常状态并产生大量信息。应用程序可能部署多个机房，当这些机房中有数据冗余时，一个位置的错误可以通过另一个位置的数据进行修正，确保整个系统的连续性和可靠性。为了提高系统可靠性，通常采用读-写分离的技术进行数据的冗余管理。读写分离是一种冗余的设计方式，缓存和数据库之间存在数据冗余，当缓存服务宕机时，可以从数据库回源到缓存。

1.1.4 熔断设计

熔断机制是应对雪崩效应的一种微服务链路保护机制，如果目标服务的调用速度较慢或超时次数较多，则此时会熔断该服务的调用。对于后续的调用请求，不再继续对目标服务进行调用，直接返回预期设置好的结果，可以快速释放资源。一般来说，熔断需要设置不同的恢复策略，如果目标服务条件改善，则恢复。

1.1.5 限流设计

限流是一种系统设计技术，用于控制访问应用程序或服务的流量，防止资源过载。常见的限流策略包括固定窗口、滑动日志、漏桶和令牌桶算法。这些方法可以帮助系统应对高流量，保持稳定性和可靠性。在实施时，通常需要结合其他系统保护措施，如队列、缓存、服务降级和熔断，以实现全面的流量控制和系统保护。

当流量被限制后，系统通常会采取以下措施之一：拒绝多余的请求、将请求排队等待处理、返回错误码（如 HTTP 429 Too Many

Requests），或者提供一个降级的服务响应。这些措施可以缓解服务器压力。

1.1.6 降级设计

降级机制是当服务器压力剧增的情况下，根据当前业务情况及流量对一些服务和页面有策略地降级，以此缓解服务器资源的压力，释放服务器资源以保证核心任务的正常运行。从降级配置方式上，降级一般可以分为主动降级和自动降级。主动降级是提前配置，自动降级则是系统发生故障时，如超时或者频繁失败，自动降级。其中，自动降级可分为超时降级、失败次数降级、故障降级。

1.1.7 可观测设计

为了保证系统的透明性并迅速定位问题，采用可观测的设计方法变得尤为关键。可观测设计涵盖了日志记录、实时监控、追踪以及度量等多个方面，从而实现了系统状态和行为的可量化以及可分析性。在可观测的设计中，日志应当详细地记录所有的关键事件，监控系统需要能够实时捕获关键的性能指标，跟踪机制应具备跨服务请求的追踪能力，度量指标则应全方位地反映系统的健康状态。另外，健康检查机制需要自动地对系统组件状态进行评估，当出现异常指标时，告警机制会立即告知相关的工作人员。通过这些措施，我们可以清晰地观察到系统的运行状态，从而为后续的维护和优化工作奠定了稳固的基础。

1.2 基础设施保障

1.2.1 机房多活

系统所部署的机器或所在地须具备一定的冗余性。包括同机房多活、同城多活和异地多活等不同级别。将要建设的机房要求具有独立性，尤其是网络环境，机房之间通过专线来进行连接。

1.2.2 网络容灾

数据中心之间的互联网络是 DC 之间业务连接的重要载体，对存储灾备网络的时延要求较低、带宽较大、可靠性较高；业务灾备网络需要实现链路备份和快速的路由收敛。

1.3 数据灾备

1.3.1 数据备份

即对核心数据进行备份和恢复的能力。需对核心数据进行实时备份，并具备快速容灾切换的能力。需对备份恢复的能力进行周期性验证。

1.3.2 数据回滚

在系统出现异常情况下，迅速有效地恢复故障前数据状态，减少了故障给业务系统带来的冲击。回滚是否有效取决于回滚执行过程和回滚决策是否及时。

2 研发保障

通常说法是，SRE 处置的线上可靠性问题中，有 70%左右源自 CD 阶段（详见第三章第 4 节“变更管理”），15%左右源自研发阶段（例如代码的低质量、高维护成本等）。对于初创的 SRE 团队，应该将工作重心放在 CD 与 CO 阶段，做好救火工作，同时夯实 CD-CO 阶段的工程化，保障 SLA。在有阶段性成果之后，应该逐步向研发服务扩展进入 CI 领域，一方面尝试缩减那 15%的可靠性根因，更重要的是打通 CI-CD，以乙方服务心态保障业务团队的研发连续性，追求平台工程的研运一体化理念。

有些企业是由质量团队或者独立的效能团队承担这一部分职能，也有业务研发兼职的情况，SRE 团队应该本着以往运维团队的服务传统，将业务研发环境视为一个独立的线上业务，以服务心态尝试“接管”研发环境中代码仓库、CI 流水线、各类测试环境、代码分析扫描平台、制品仓库等研发基础设施的运维工作，并保障其可靠性，进而提高业务团队的生产效率，降低其出错概率。

有能力的 SRE 团队接下来可以依托自身的工程化能力将这些工具升级改进，与 SRE 前期已经建设的 CD-CO 平台整合，形成覆盖代码全生命周期的“研发运维运营”一体化的平台工程基础设施。这种发展路径借助运维 SRE 与业务开发团队的紧密关系减少了内耗，CI-CD-CO 全路径由 SRE 团队建设避免了轮子，还减少了例如 PE、DevOps 等小众的公司级岗位设置。

研发过程可靠性，指的是以 SRE 理论驱动研发连续性建设，提升研发管线的工业化水平，保障版本能够高质量迭代，从而持续保障业务的线上可靠性，同时兼顾了版本迭代效率；

SRE 主导的研发连续性保障，可以拆解为代码可靠性、代码仓库可靠性、构建可靠性和制品可靠性四个方面，每个阶段分别对其可靠性进行定义并提出相应的改善措施；

2.1 研发保障体系设计

2.1.1 代码可靠性

代码是基于一定需求实现，用于构建对应软件的文件集合。代码质量从基础上决定了软件的成败，是软件开发过程中不可忽视的一环。在软件版本快速迭代的今天，如何构建高质量的代码更显得至关重要。

代码可靠性的落地仅靠宣导或者文档还远远不够，需要建设完善的检查工具并量化效果，一般由平台 SRE 建设相关的工具，因此需要平台 SRE 需要深入了解影响代码可靠性的常见问题及提升措施，不断完善代码检查工具的能力；

2.1.1.1 代码缺陷

代码缺陷是指影响代码稳定运行的问题，或者未达到设计时的预期功能。缺陷产生的原因有多种，比如：软件的复杂性、编写的错误、需求歧义等。代码缺陷的及时发现与修复，对项目进度与工程质量至关重要。

代码缺陷检测，一般可采用静态分析法或动态分析法。

静态代码分析是指无需运行被测代码，通过词法分析、语法分析、控制流、数据流分析等技术对程序代码进行扫描，找出代码隐藏的错误和缺陷，如参数不匹配，有歧义的嵌套语句，错误的递归，非法计算，可能出现的空指针引用等等。

动态分析方法则一般应用于软件的测试运行阶段，在软件程序运行过程中，通过分析动态调试器中程序的状态、执行路径等信息来发现缺陷。

1. 代码缺陷规避措施

代码缺陷种类较多，无法完全罗列，这里选取部分最为常见的缺陷并介绍对应的规避办法：

1）指针错误使用

指针的错误使用，一般要避免空指针、野指针，避免指针类型不匹配，不返回局部变量的指针。

2）内存非法访问

非法内存访问是指程序试图读取或写入未分配/受保护的内存，如数组越界等，这将会导致程序的不可控行为。因此，必须确保程序正确地分配和释放内存，避免缓冲区溢出等现象。也可使用静态分析工具检测代码中的内存非法访问问题。

3）变量未初始化

使用未初始化的变量，可能导致未知错误。一般来讲，变量需要在声明时赋予初始值。

网址：SRE-Elite.com 微信：SRE 精英联盟

4）资源泄漏

常见的资源泄漏包含 socket 泄漏，文件句柄泄漏，内存泄漏等。产生原因是未能正确释放已经分配的内存或其他资源，导致这些资源被长期占用。资源泄漏不仅会造成资源的浪费，系统性能下降，严重时超出系统限制会导致程序崩溃。

避免资源泄漏，在编程过程中，需要在资源使用完毕后进行资源的释放，如 socket/文件句柄的关闭，动态分配内存的释放。

5）竞争死锁

竞争死锁是指多个线程或进程持有资源，又互相竞争等待对方资源而导致死锁的情况。解决死锁问题，一般可采用超时使用机制、统一获取资源顺序和死锁检测机制来打破死锁产生的必要条件。

6）不当的 API 使用

不当的 API 使用，会导致程序异常。可通过仔细阅读 API 文档，了解 API 的使用方式，在使用 API 前进行充分测试的方法来规避。

2. 效果评估

代码缺陷数作为代码质量指标之一，可以从数量、严重程度来归类。在度量层面，一般使用百行告警数来衡量代码缺陷指标。通过配置代码缺陷规则集，采用缺陷检测工具扫描，生成检测报告。

根据缺陷的严重程度分为严重告警（空指针、数组越界等），一般告警（变量未初始化等）和提示告警（如代码风格等）。设定

网址：SRE-Elite.com 微信：SRE 精英联盟

各个告警等级的权重，统计代码行数，最终计算出百行代码缺陷告警数。

百行缺陷告警数=（严重告警 W1+一般告警 W2+
提示告警 W3）/代码行数 100

（W1/W2/W3 为权重系数）

2.1.1.2 代码规范

代码规范主要是指是否遵守了团队或者业界的编码规范。代码规范主要涵盖：代码风格（如注释、空代码块、命名、格式化等），与异常处理等部分。

代码规范有助于提高代码可读性与可维护性，从而提升团队内开发效率。代码可读性帮助相关技术人员能够轻松阅读并理解代码意图与实现方式。代码从分支发起到主干的合并请求前，必须进行代码检查，这也是提前发现问题的方法之一。

1. 代码规范提升措施

1）代码风格

良好的代码风格会帮助开发人员阅读和理解符合该风格的源代码，并且避免错误。此处所讲述的代码风格包括但不限于：命名规范，表达式与语句，缩进，对齐，注释，代码布局等。对于不同的编程语言，适用于不同的代码风格，对于同一项目或开发团队，应当使用统一的代码风格。

（1）命名规范，命名要能够直观地表达本身的意图，同时具备可读与可搜索性，尽量遵循一些通用的写法。在此基础上命名长度应当尽量精短，避免触发代码行字符数限制规范；

（2）缩进，一般采用 4 个空格缩进，而不使用 tab 键（特殊语言除外）

（3）统一字符编码格式，通常采用 UTF-8 编码

（4）单行字符数限制，长度一般不超过 120

（5）行尾换行符，一般使用换行符 LF，禁止使用回车键 CR

2）异常处理

异常处理是为了防止一些未知错误产生而采取的措施。适当地使用异常处理能够提高程序的容错性。在处理异常方面，需要遵循：

（1）只在可能出异常的块进行精准捕获处理；

（2）捕获的异常必须处理或抛出给上层调用方；

（3）异常处理效率较低，应避免使用异常做条件控制

2. 效果评估

对于代码规范的效果评估，可以采用百行告警数来衡量。

百行告警数=（严重告警 W1+一般告警 W2+
提示告警 W3）/代码行数 100

注：（W1/W2/W3 为权重系数）

网址：SRE-Elite.com 微信：SRE 精英联盟

百行告警数可以用来评估代码的质量和稳定性，较高的百行告警数可能意味着代码存在较多的缺陷和潜在问题，需要更多的测试和修复工作；

2.1.1.3 代码安全

安全性是指在为正常访问提供服务的同时，也能拒绝非法访问。同时不因为代码设计或实现的原因，导致信息泄露/非法侵入/系统崩溃等问题。

1. 代码安全性提升措施

1）防止敏感信息泄露

敏感信息可分为系统敏感信息与应用敏感信息。系统敏感信息包含业务系统的基础环境信息，如系统版本、组件版本等；应用敏感信息包含用户信息和应用信息等，如用户 TOKEN、密码、IP 等。系统敏感信息泄露会为攻击者提供更多的攻击方法，应用敏感信息泄露危害则因泄漏信息内容而决定。

解决方法：

（1）避免硬编码，禁止将密码等敏感信息写入代码，应该以配置或后台下发形式读取

（2）处理异常时，避免将系统信息、DEBUG 信息，或者敏感文件的路径输出到用户可见处

2）预防安全漏洞

代码安全漏洞，指编码过程中因不当的处理逻辑引发的安全风险；

常见的代码安全漏洞有：

（1）脚本（SQL）注入，可通过减少拼接命令，对命令参数值进行过滤/校验避免

（2）XSS 攻击，可以使用安全的 JavaScript 框架和组件，同时主动检测发现，转义输入等减少

（3）越权访问，是由于权限设计错误，未授权用户获取甚至修改其他用户的信息。需要通过最小化原则的权限设计与审计来规避

（4）通信安全，一般是由于未使用加密信道进行通信导致，可以通过使用加密/私有协议通信来避免

3）第三方组件安全

软件开发中不可避免地会引入依赖库，或者第三方 SDK。这些第三方组件作为系统的一部分，与原生代码并无本质区别，它们的安全性也同样影响整个系统。因此，需要在减少对第三方组件引入的基础上，加入相应的安全评估机制。

对于第三方组件的评估，我们主要从以下几个方面：

（1）组件安全风险应在引入前/上线前/定期进行安全扫描

（2）组件合规风险包括使用协议合规和监管数据合规

（3）组件稳定性应当使用经过实际验证的 LTS 版本

2. 效果评估

在衡量代码安全性方面，可以从敏感信息泄露、系统漏洞、第三方高危组件等几个方面来考量。可以通过代码扫描工具，扫描出已知系统漏洞与敏感信息，以及第三方高危组件的引入情况。

在得到敏感信息，安全漏洞个数，以及第三方高危组件个数后，可以制定代码安全性红线。一般来讲，敏感信息、安全漏洞是绝对不允许的。对于第三方高危组件，也要经过安全评估测试，其标准与本身代码相同。

（1）对于敏感信息与安全漏洞，必须彻底清除

（2）第三方高危组件，还可以采用 LTS 覆盖率可以用来衡量系统中使用的第三方高危组件稳定性。LTS 覆盖率=第三方高危组件 LTS 数/第三方高危组件数

2.1.1.4 代码圈复杂度

圈复杂度是一种代码复杂度的衡量标准，可以用来衡量一个模块流程判定结构的复杂程度。圈复杂度大说明程序代码的判断逻辑复杂，可用来表示对给定代码进行测试、维护或故障排除的难度，以及代码生成错误的可能性。同时，也可用来帮助开发人员确定是否需要对程序进行重构，以降低程序的复杂度，提高代码质量。

1. 圈复杂度改善措施

降低函数圈复杂度主要通过对代码重构来进行，一般有以下几种方法：

（1）将大函数拆分成多个小函数，每个小函数只负责单一功能

（2）将条件判定提炼出来，成为独立函数

（3）简化、合并条件表达式

（4）优化循环结构，减少循环嵌套、使用更简单的循环结构等

2. 效果评估

圈复杂度反映了代码的耦合度，圈复杂度越高的代码会有越多潜在的 BUG。

对于圈复杂度，可以从以下指标衡量：

（1）单函数圈复杂度最大值小于等于 20

（2）项目平均圈复杂度，一般不大于 4

项目平均圈复杂度=所有函数圈复杂度之和/所有函数个数

2.1.1.5 代码重复

代码重复指的是程序中存在相同或类似的代码段。在不同的位置或程序中出现相同的代码，会造成代码冗余和浪费。代码重复，不仅导致项目代码量的增加，影响程序的可读性和可维护性，增加代码的错误率和修改难度，也是设计不佳的一个标志。

代码重复的表现形式多种多样，常见形式有：

（1）完全一样的代码

（2）仅重命名标识符的代码

（3）仅变量赋值不一样的代码

（4）插入或删除语句的代码

（5）重新排列语句的代码

1. 代码重复改善措施

降低重复代码，是代码优化的重要方面之一，一般需要对相关功能进行重构。

常见的改善措施，主要是抽取公共代码、封装函数、使用继承和多态等

（1）抽象出公共方法或函数，将重复的代码封装在一个函数或方法中

（2）使用继承或接口，将共同代码放在父类或接口中，子类只实现自己的特定部分

（3）使用设计模式，如工厂模式、模板方法模式等

（4）利用现有的框架或库，避免自己重写

2. 效果评估

代码重复的度量，可以使用代码重复率来表示，可以通过静态扫描工具得出。

代码重复率，指的是在一段代码中重复出现的代码段的比例。代码重复率越高，代码的可维护性和可读性就越差。

代码重复率=重复行数/代码总行数

在实际的工程中，一般建议：

（1）单文件代码重复率最大值小于等于 10%

（2）项目平均代码重复率小于等于 10%

2.1.1.6 代码注释与 API 文档

代码的注释与 API 文档编写是软件开发过程中非常重要的一部分，可以提高代码的可读性和可维护性。通过代码注释可以帮助阅读者快速理解代码的功能和实现方式。API 文档则是其他开发者了解使用软件的重要途径。开发者应该养成写注释和文档的好习惯，为自己和其他开发者节省开发时间。

1. 代码注释与 API 文档提升措施

网址：SRE-Elite.com 微信：SRE 精英联盟

（1）注释目的在于使阅读者能够快速掌握注释对象的使用方式与原理，良好的注释应包

含注释对象的产生意图，设计考量与如何使用；注释一般应当包含文件注释，类/结构体注释，函数方法注释，变量注释以及适当的代码段注释；对于规范化命名的变量与简单函数方法，可以不进行注释。

（2）API 文档，应当描述各个类和方法的功能和使用方法，同时遵循行业和国际标准，具备兼容性和实时性。

2. 效果评估

对于注释与 API 文档的考量，可以从 API 文档覆盖率，代码注释行密度来衡量。

1）注释行密度=注释行数/总行数* 100

此标准用于衡量百行代码中，所包含注释行数，一般认为低于 5 表示几乎没注释。

2）API 文档覆盖率=已覆盖 API 接口数量/总 API 接口数量

此标准用于衡量对外 API 文档的完善程度；一般来讲，至少需要达到 80%覆盖率，即覆盖大部分的 API 功能，忽略了一些不太重要或不常用的 API；同时也需要定期更新，使文档保持最新、全面、准确的状态。

2.1.1.7 代码质量红线

代码质量红线是指在开发过程中，开发团队所设定的一些规则和标准，用于确保代码质量达到一定的水平，也是衡量代码质量的

一个综合考量。这些规则和标准通常是基于行业最佳实践和经验总结制定的，是团队开发的一种约束和保障。当代码质量超越红线时，就需要开发团队及时进行修正和优化，以确保代码质量和运行稳定性。

1. 质量红线改善措施

质量红线的触发标准是基于每个质量指标的。提升每项指标质量有助于避免触发红线，也可以帮助开发团队提高软件开发的效率和质量，减少错误。

2. 效果评估

质量红线一般包含以下几个指标：

（1）代码缺陷

（2）代码风格

（3）代码安全性

（4）圈复杂度

（5）代码重复率

（6）代码注释和文档

（7）单元测试覆盖率

通过在代码合并、转测等场景下，对以上每个指标单独设定阈值，可以划定出不同场景下的质量红线，当某项指标触发质量红线时终止后续的 CI/CD 流程，并要求开发团队进行修复。

2.1.2 代码仓库可靠性

代码仓库就是存放源代码和资源的地方，亦称版本库、代码库，其核心功能是版本控制，记录一个或若干文件的变化，以便后续查看特定版本修订的情况;

代码仓库出现问题，对代码拉取、项目开发、编译构建等都会造成影响，所以代码仓库的可靠性是整体研发流程可用性的关键一环;

代码仓库的可靠性包括：仓库性能、仓库容灾、仓库安全和仓库可扩展性四个方面;

代码仓库的可靠性主要侧重网络优化、部署优化、配置优化、安全提升等工作，可由服务 SRE 和安全人员来承担相关能力的建设;

2.1.2.1 仓库性能

代码仓库的性能通常是指代码仓库对代码的存储、管理和处理时的速度和效率，包括代码提交和拉取的速度、分支合并的速度等，高性能的代码仓库，可以减少开发人员的等待时间，缩短产品交付周期;

1. 代码仓库性能提升措施

（1）控制代码仓库的大小：代码库的大小会直接影响仓库的性能，因为大型代码仓库需要更多的时间来处理和查找文件。因此，

网址：SRE-Elite.com 微信：SRE 精英联盟

需要合理控制仓库大小，及时删除不需要使用的文件，并合理设置文件的保存周期；

（2）合理设置代码仓库结构：如果代码仓库的结构合理，可以更快地查找和访问文件，从而提高性能，例如，某些版本管理软件，支持大文件单独存储在仓库之外，仓库中实际只存储一个很小的文本指针，可以将存储大文件的目录设置使用更适合的存储形式；

（3）版本工具选型：不同的版本控制工具可能会对性能产生不同的影响。例如，Git 和 SVN、P4 的架构和设计理念的差异，在处理大文件的性能存在差异，需要根据业务资源文件的数量和大小、团队的多地分布特性等综合选择；

（4）网络优化：如果多个人同时访问代码仓库，网络连接的速度也会影响性能，在评估代码仓库性能时，需要考虑网络连接的速度和质量，可在离用户就近的网络区域，部署边缘节点，缓存最近的版本，减少网络距离传输损耗，提高访问速率；

（5）硬件升级：硬件也会影响代码仓库的性能，例如，使用较高 IO 性能的磁盘，可以提高代码仓库的读取速度；

（6）集群化：通过集群化来部署代码仓库，可以让代码仓库支持更大规模团队的使用；

2. 效果评估

一般采用下面 3 个指标来评估仓库的性能

网址：SRE-Elite.com 微信：SRE 精英联盟

（1）文件下载速度：通常是指每秒传输的数据量，常见的单位有比特/秒（bps）、千比特/秒（Kbps）、兆比特/秒（Mbps）和千兆比特/秒（Gbps）等；

（2）下载卡顿率：是指从仓库中下载时出现卡顿的频率或时间占比，通常使用百分比（%）来表示；

（3）并发请求量：一般团队多人同时拉取或者提交代码可能会影响速度，通常使用 QPS 来表示系统每秒钟的请求量；

2.1.2.1 仓库容灾

代码仓库容灾是指代码仓库在经受自然灾害、设备故障、网络故障、人为错误等不可预测的问题后，通过备份、容错机制和恢复策略在最短时间内恢复到正常可用的状态。完备的代码仓库容灾机制，可以避免团队或公司核心代码资产遭受损失。

1. 代码仓库容灾提升措施

（1）数据备份：定期对代码仓库的数据进行备份，确保在数据丢失或损坏时能够及时恢复。

（2）多地备份：将备份数据存储在多个地方，以防止单点故障。

（3）容错机制：使用容错技术，如 RAID 等，以防止硬件故障导致数据丢失。或者将本地普通硬盘替换为云硬盘，云硬盘中的数据以多副本冗余方式存储，会避免数据的单点故障风险。

（4）灾备恢复策略：制定灾备恢复策略，以便在发生灾难时能够及时恢复。

（5）人员培训：对相关人员进行培训，提高应对灾难的能力和应变能力。

2. 效果评估

一般使用以下几个指标来评估代码仓库容灾效果

（1）恢复时间目标（RTO）：Recovery Time Objective，他是指故障发生时间到故障恢复时间，两个时间点之间的时间段称为RTO；

（2）恢复点目标（RPO）：Recovery Point Objective，是指系统恢复到怎样的程度。这种程度可以是上一周的备份数据，也可以是上一次的实时数据；

（3）投入产出比（ROI）：Return of Investment，容灾系统的投入产出比，可以使用最高的性价比方案来达到容灾效果，为团队节省成本；

2.1.2.3 仓库安全

代码仓库的安全性是指代码仓库中存储的代码等数据受到保护的程度，以防止未经授权的访问、篡改、泄露和破坏。保护代码仓库的安全性包括但不限于访问控制、数据加密、代码审查、安全漏洞、操作审计、私有网络部署等。高安全性的代码仓库可以保护代码的机密性，完整性，避免因安全漏洞造成团队或者公司的损失和风险。

1. 代码仓库安全性提升措施

（1）访问控制：评估代码仓库中代码的访问控制机制，包括用户认证、授权、权限管理等，确保只有授权的用户能够访问仓库中的代码。

（2）数据加密：评估仓库中存储的关键元数据或者敏感代码是否采用了合适的加密技术进行保护，以防止敏感信息泄露。

（3）代码审查：评估代码审查机制，确保代码质量和安全性。详细可参考：4.2.1.3 代码安全。

（4）安全漏洞：评估代码仓库中可能存在的安全漏洞，包括代码中的漏洞、第三方库中的漏洞等。

（5）操作审计：评估代码仓库的操作审计，确保所有操作可追踪，以便及时发现和应对安全事件。

（6）私有化部署：将代码仓库部署在私有网络中，例如，企业、学校的内部网络中，相比公网中的外部代码托管服务理论上会更安全；

2. 效果评估

安全往往只有 0 和 1 的概念，要么安全，要么不安全，不存在中间状态，因此代码安全可靠性的效果评估可通过代码泄露等安全事件的发生与否来评估，同时通过审计能力来保障安全事件发生时具备可回溯能力；

（1）代码泄漏次数：代码仓库代码泄露事件的次数；

（2）漏洞数量：代码仓库中发现的漏洞数量；

（3）访问控制：是否具备精细化权限控制的能力；

（4）审计能力：是否具备良好的审计能力；

2.1.2.4 仓库可扩展性

代码仓库的可扩展性是指代码仓库在面对不断增长的代码量、用户数量、访问频率等变化时，能够保持高效的性能，同时可以方便地通过软硬件来扩展功能和架构，以满足未来的需求，代码仓库的可扩展性对一个快速发展中的团队尤其重要。

1. 代码仓库可扩展性提升措施

（1）分布式版本控制系统：使用支持分布式版本控制系统，比如 Git，可以支持代码仓库的架构可扩展性，可以将代码库分散在多个服务器上，从而实现横向可扩展；

（2）集群化：使用集群化技术，如 Kubernetes 等容器编排系统，可以实现代码仓库的自动化部署和管理；

（3）功能扩展：使用插件等能力，结合业务的个性化诉求，扩展版本控制系统的功能，满足团队的研发需求；

2. 效果评估

代码仓库可扩展的能力可使用以下两个指标评估

（1）架构可扩展性：代码仓库架构是否支持水平横向扩展。

（2）功能可扩展性：是否可以通过插件扩展版本控制系统的功能。

网址：SRE-Elite.com 微信：SRE 精英联盟

2.1.3 构建可靠性

构建是指在构建机上把代码、资源文件等源文件编译打包成可执行的程序文件的过程；在当前的持续集成/持续交付的软件开发模式下，若构建出现问题，则新的软件版本无法快速发布验证，软件质量就会受到影响，所以构建的可靠性对于软件服务的可靠性和迭代效率起到了重要作用；构建可靠性主要由构建效率和构建成功率两个方面组成；

构建可靠性提升涉及构建工具建设与规划、业务层改造优化、软硬协同等多个方面，一般需要由平台 SRE、业务开发、服务 SRE 多角色共同参与；

2.1.3.1 构建效率

构建效率即构建速度，取决于从构建启动到构建结束的耗时情况，构建耗时过长的话软件版本无法按时交付，对业务版本迭代效率产生了影响，构建可靠性就无从谈起。

1. 构建效率提升措施

（1）流程自动化

利用自动化构建工具或脚本把构建各个环节串联起来，减少各环节之间的等待时间；

（2）并行化

通过把一些构建流程从串行改成并行，优化构建流程，提升构建速度。

网址：SRE-Elite.com 微信：SRE 精英联盟

（3）增量构建

在构建机上执行一次构建时把构建过程中的一些临时文件、中间产物存起来作为构建缓存，等下一次构建时，由于一般情况下只有部分代码被修改了，那么没有被修改的代码就能免去构建环节，直接使用上次的构建缓存，这样就通过增量构建的方式减少了需要构建的内容，降低了构建耗时。

针对构建机首次构建时没有缓存的问题，可以搭建构建缓存共享服务器（例如 UE 引擎的 DDC 服务器），一台构建机的构建缓存会生成到缓存共享服务器上，供其他构建机使用。

（4）分布式编译

相对单机有限的资源来讲，集群的力量无疑是强大的：一个人计算 100 道数学题，相比 100 个同样能力的人一人计算 1 道题，孰优孰劣不言而喻。分布式编译加速就是利用集群的资源，将单个节点的工作分配给一大批节点，然后再汇总结果。根据需要，资源数量可以近乎无限地扩充，不再受制于单机的物理架构；时间上，集群工作所需要的时间往往是原来的几分之一。

（5）软硬协同

部分构建任务比如代码预处理、资源文件处理无法通过分布式编译加速分发到远端，只能在构建机本地处理，此时本地构建机性能就成为瓶颈，可以针对性地提升本地构建机的 CPU/内存/磁盘 IO 性能，再结合分布式编译系统，软硬协同提升构建速度。

（6）多进程编译

网址：SRE-Elite.com 微信：SRE 精英联盟

有些编译软件默认只开启单进程编译，导致构建机硬件性能未得到充分利用，此时可以通过开启多进程编译来提升构建速度。

2. 效果评估

（1）基于基线：使用构建耗时超出基线比例来评估单次构建的效果，每次正常构建完成后，把本次构建耗时上报上去，持续若干天后，我们就能得到一段时期内的多次稳定构建耗时数据，把这些数据求一个平均值之后，我们就得到了这一段时期的构建耗时基线。当一次构建耗时超出基线很多时（比如超出基线 20%），这次构建就可能出现了性能问题，在得到构建耗时基线以后，我们可以将当前构建耗时与基线进行对比来作为 SLO，比如不超过基线 10% 即为健康；

（2）基于阈值：按不同的业务实际情况设置不同的阈值，例如设置构建耗时不大于 2H 为可靠性的衡量标准；

2.1.3.2 构建成功率

构建成功率指指定时间内，构建成功次数占构建总次数的比例。构建成功率低的话，需要多次构建才能产生可交付的版本，也是影响构建可靠性的一个关键因素；

1. 构建成功率提升措施

（1）保障构建环境可靠性

构建成功率受到构建环境影响，当构建机异常时（比如缺少某个依赖包，连不上代码仓库，磁盘故障等），构建就会失败。我们需要保证构建环境的可靠性，比如通过自动化方式批量部署构建

机，避免手动部署时遗漏某些依赖包；尽量利用云上的高可靠性网络、计算和存储。

（2）PreBuild 预编译检查

通常的构建是拉取代码后再执行后面的编译流程，这就要求开发编写完代码后必须提交到代码仓库再启动构建。其实可以在提交代码到仓库之前就把问题暴露出来，问题越早发现，修复解决的成本越低；提交到代码库的代码质量越高，问题越少，团队协作起来就越顺畅，越高效。PreBuild 就是在提交代码到仓库之前，利用本地 PreBuild 工具进行本地代码检查或传输代码到远端进行预编译代码检查，从而尽早发现问题，实现质量左移。

2. 效果评估

（1）基于基线评估：每次构建完后将成功/失败状态进行上报，统计一天的构建成功率，持续若干天后，我们就能得到一段时期内的多天稳定构建成功率，把这些数据求一个平均值之后，我们就得到了这一段时期的构建成功率基线。当某天的构建成功率超出基线很多时（比如超出基线 20%），这次构建就可能出现了性能问题。在得到构建成功率基线以后，我们可以将当前构建成功率与基线进行对比来作为 SLO，比如不低于基线 10%即为健康。

（2）基于阈值评估：按业务需求设置固定的成功率阈值，例如 80%，可以取一个固定周期的所有构建进行统计分析并进行对比，例如以天/周/月等单位。

网址：SRE-Elite.com 微信：SRE 精英联盟

2.1.4 制品可靠性

制品为构建过程的产物，包括软件包、测试报告、应用配置文件等，最终提供给研发、测试等多个角色，用于下载并部署到 PC、console、移动端、服务器等多种不同的设备，从而完成发布和交付；

制品的可靠性分为制品下载可靠性、制品部署可靠性、制品安全可靠性三个方面；

制品可靠性提升涉及制品库工具建设与规划、安全、部署分发等多个方面，一般需要由平台 SRE、服务 SRE、安全人员共同参与；

2.1.4.1 制品下载可靠性

随着企业的快速发展，研发团队规模的扩大，为拓展全球市场在国内海外多地建立研发团队，制品的高效交付和管理也成为影响研发效率的关键，制品分发缓慢、下载困难等为代表的制品库不可用挑战，急需快速解决；

1. 制品下载可用性提升措施

（1）多地分发：根据研发的不同地域分布，按需设置制品的分发策略，实现构建完成后多地制品分发，从而达到提升制品速率的目的；

（2）P2P：制品的下载往往具备一定的峰值规律，例如早高峰会出现大量的集中下载，为提升并发下载的速率，可以使用 P2P 的下载策略，下载用户越多，速率越快；

（3）专属工具：制品的下载使用专属下载工具，能够支持分片下载、断点续传、多线程、热点缓存等特性；

（4）镜像源加速：建设距离用户更近用户的镜像源，来提升制品依赖的拉取速度，减少下载中断；

2. 效果评估

（1）下载成功率，下载成功率=1-（失败请求数/用户请求数），失败请求是指制品库返回的错误码为服务器内部错误码的请求（错误码>500）；但不包括触发频控导致的限流请求或者制品库升级、变更、停机而导致的失败请求。用户请求指的是制品库服务器端接收到的用户发送的请求，但不包括未经身份验证、鉴权失败或者欠费停服状态下的请求。用户端由于黑客攻击而对制品库的请求，或者由于配置了跨区域复制、生命周期规则而在后端异步执行的请求，均不视为有效请求或失败请求

（2）下载速度，下载速度=包大小/下载耗时，下载速度和制品的存储区域分布及用户所在的区域有较大的关系，可针对不同地域、国家可设置差异化的可用性衡量指标；

2.1.4.2 制品部署可靠性

在制品下载完成后，还有一项重要的下游能力，即制品的自动化部署，使用户实现对制品产物即时体验的能力；

网址：SRE-Elite.com 微信：SRE 精英联盟

在大型业务研发过程中，多种研发角色手动安装制品产物非常耗时，在没有自动化部署的情况下执行诸如软件安装和升级会消耗大量研发人员的时间和精力，且对迭代的效率有较大的影响；

制品部署是指通过技术手段，打通构建流水线，将构建完成的制品主动推送到用户多种类型终端的过程，用自动化的方式对制品进行分发、安装、更新，让用户以较低的时间成本获取到构建产物，减少临时下载制品对迭代效率的影响，尤其对于具备超大制品的业务类型，制品部署的可靠性尤为重要；

1. 制品部署可用性提升措施

（1）多用途支持：为满足不同的测试验证目的，一个项目往往会设置多条构建流水线，例如用于逻辑、性能、开发调试等多种不同类型，制品部署能力需要支持到多种类型；

（2）多平台支持：制品的分类按平台分有 iOS、安卓、PC、console 等，每种平台都有多种不同类型的 OS 版本和设备型号，碎片化程度高，预部署能力需要良好的机型兼容适配能力，能够支持多种不同类型的设备和平台；

2. 效果评估

（1）部署成功率，部署成功率=成功部署次数/总分发次数

（2）部署人工耗时，一个完善的预分发方案应该尽量减少用户等待的时长，通过部署人工耗时指标来驱动减少人工参与；

网址：SRE-Elite.com 微信：SRE 精英联盟

2.1.4.3 制品安全可靠性

制品在持续构建过程中的包依赖，以及构建完成的产物（含 Docker 镜像、npm、helm、maven 等多种不同类型的格式），可能存在一些安全风险，导致制品不可靠；

1. 制品安全性提升措施

（1）漏洞扫描，制品需要经过漏洞、license 信息的扫描与分析，并对漏洞和不合规的 license 告警并输出安全合规报告，漏洞库需要及时更新；

（2）设置质量红线，禁止下载未经安全扫描或者未通过安全扫描的制品；

（3）访问控制，针对不同角色设置差异化的权限，防止资源泄露，减少恶意盗取制品风险，权限以最小化为原则；

（4）操作审计，提供制品库的操作审计功能，保证制品的下载使用等操作可追溯；

2. 效果评估

（1）漏洞扫描覆盖度：扫描能力能够覆盖足够多的制品库类型；

（2）漏洞扫描速度：漏洞扫描的速度和及时性，能够在漏洞产生后，以最快的速度发现异常；

（3）漏洞扫描准确性：漏洞库及时更新和维护，确保制品安全扫描结果的准确性；

（4）访问控制：是否具备精细化权限控制的能力；

（5）审计能力：是否具备良好的审计能力；

2.2 研发保障工程体系设计

2.2.1 面向研发保障的持续集成流水线

持续集成（CI）是现代软件开发中的关键实践，通过自动化的方式将代码的构建、测试、部署等环节有机结合，确保代码在每次提交后都能快速集成并验证，从而提高软件开发的效率和质量。面向研发保障的持续集成流水线设计，旨在通过一系列自动化工具和流程，保障工程编译，静态代码检查，测试用例运行和部署发布。

另外，通常情况下，平台需要提供以下关键组件能力：

流水线：通过可视化手段展示团队现行的研发流程，实现编译、测试、部署等环节的一体化管理，确保流程的顺畅与高效。

代码检查：提供精细化的代码审查解决方案，从多个维度检测代码中的缺陷、安全漏洞以及规范性问题，确保产品质量得到充分保障。

代码库：企业内部代码托管服务

凭证管理：为代码库、流水线等关键服务提供多样化的凭证与证书管理功能，增强系统的安全性。

环境管理：可以将企业内部的开发编译机托管至流水线，统一管理

研发商店：由流水线插件和流水线模板构成，插件用于整合企业内部的多种第三方服务，模板则助力企业研发流程的标准化。

编译加速：通过并行编译，编译缓存，硬件匹配优化的手段，显著提升构建任务的执行效率。

制品库：通常基于分布式存储架构，具备扩展能力，其功能涵盖制品扫描、分发、晋级、代理、包管理等，同时提供多种依赖源仓库，如 generic（二进制文件）、maven、npm、pypi、oci、docker、helm、composer、nuget 等，以满足不同开发需求。

持续集成流水线的设计需要遵循以下几个关键原则：

- 自动化：尽可能将所有的构建、测试、部署步骤自动化，减少人为干预和错误。
- 快速反馈：确保每次代码提交后，开发者能够快速得到构建和测试结果，及时发现并修复问题。
- 可重复性：流水线的每一步都应该是可重复的，确保在不同环境下执行结果一致。
- 可扩展性：流水线设计应具备良好的扩展性，提供接口及相关的扩展机制，对接更多的插件，满足多样化的流水线能力需求。
- 可观测性：提供全面的监控和日志记录，便于问题定位和分析。

在流水线可视化设计上，尽量采用 Stage/Job/Task 三层结构展示，方便快速定位问题。



1. Stage（阶段）

- 由多个 Jobs（作业）组成；
- 同一个 Stage 下的 Job 执行方式为并行，由于 Job 之间是相互独立的，某个 Job 失败后，其他的 Job 会被运行到完成；
- 一个 Job 失败，则该 Stage 失败。



2. Job



可以运行在一个构建环境里，比如运行在 macOS；也可以作为不需要构建环境的普通任务调度编排。它有如下特性：

由多个 Tasks(插件)组成；

一个 Task 失败，则该 Job 失败，其余 Task 将不会运行；

Task

也被称为流水线插件，通常是一个单独的任务，如拉取 Git 仓库代码等。

Task 必须包含在 Job 内，同一个 Job 内的 Tasks 都是从上往下顺序执行（启用了高级流程控制的 Task 除外）。

2.2.2 面向研发保障的可观测设计

面向研发保障的可观测设计旨在通过全面的监控和分析，确保流水线的透明性和可控性。在流水线中，构建机承受着极大的工作负载压力，CPU 负载接近满负荷，同时占用大量内存和磁盘空间。此外，由于代码和配置问题，构建任务失败的可能性较高。通过可观测性工具，团队能够实时监控流水线状态，快速定位和解决问题。

要做好流水线的可观测性，要从以下几个维度进行考虑：

1. 构建机的监控

CPU 和内存使用率：监控构建机的 CPU 和内存使用情况，确保资源分配合理，防止资源耗尽导致构建失败。

GPU 使用率：对使用 GPU 加速的构建任务，监控 GPU 的使用情况，包括利用率、温度和显存占用，以确保其高效运行。

磁盘空间：定期检查磁盘使用情况，清理不必要的文件，确保有足够的空间进行构建。特别是游戏制品巨大

网络流量：监控网络流量，确保构建机与其他系统的通信顺畅，避免因网络问题导致的构建延迟。

2.流水线监控

构建状态：实时监控每个构建任务的状态，包括成功、失败、进行中等，帮助快速识别问题。

任务队列：监控流水线中的任务队列，优化任务调度，防止任务积压。

执行时间：分析各个阶段的执行时间，识别性能瓶颈并进行优化。

3.日志监控

错误和警告：重点关注日志中的错误和警告信息，及时采取措施解决问题。

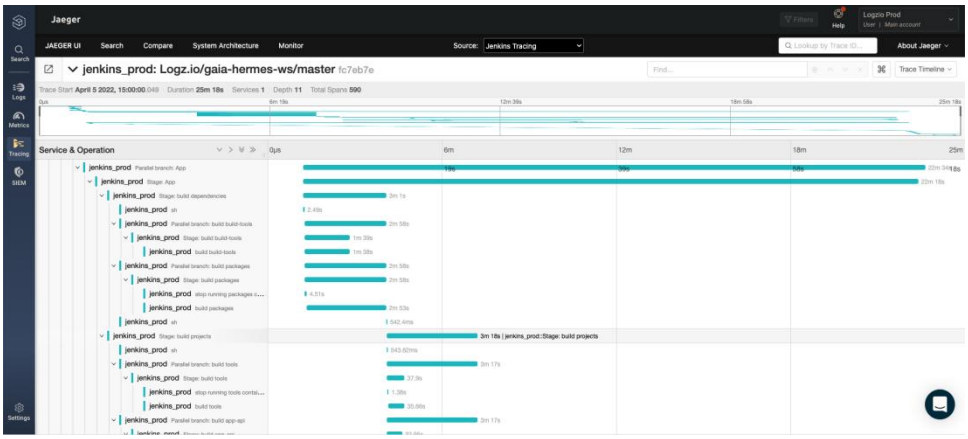
历史日志分析：通过分析历史日志，识别常见问题和趋势，改进构建流程。

4.Trace 跟踪

构建的过程基本上等同于一条 Trace ， 通过对构建过程进行 OpenTelemetry 等链路跟踪协议后， 进行上传， 可以实现更细粒度

网址：SRE-Elite.com 微信：SRE 精英联盟

的构建成功跟踪。如下图，通过上报到 Jaeger 后， 可以实现每个编译步骤的消耗时长的可视化。



5. 告警触达

该平台将指标、仪表盘以及告警信息推送至研发团队，使研发人员能够更全面地掌握持续集成环节中出现的问题，从而进一步释放系统可靠性工程师（SRE）的人力资源，并提高研发效率，降低沟通成本。

2.2.3 面向研发保障的操作调度操作平台

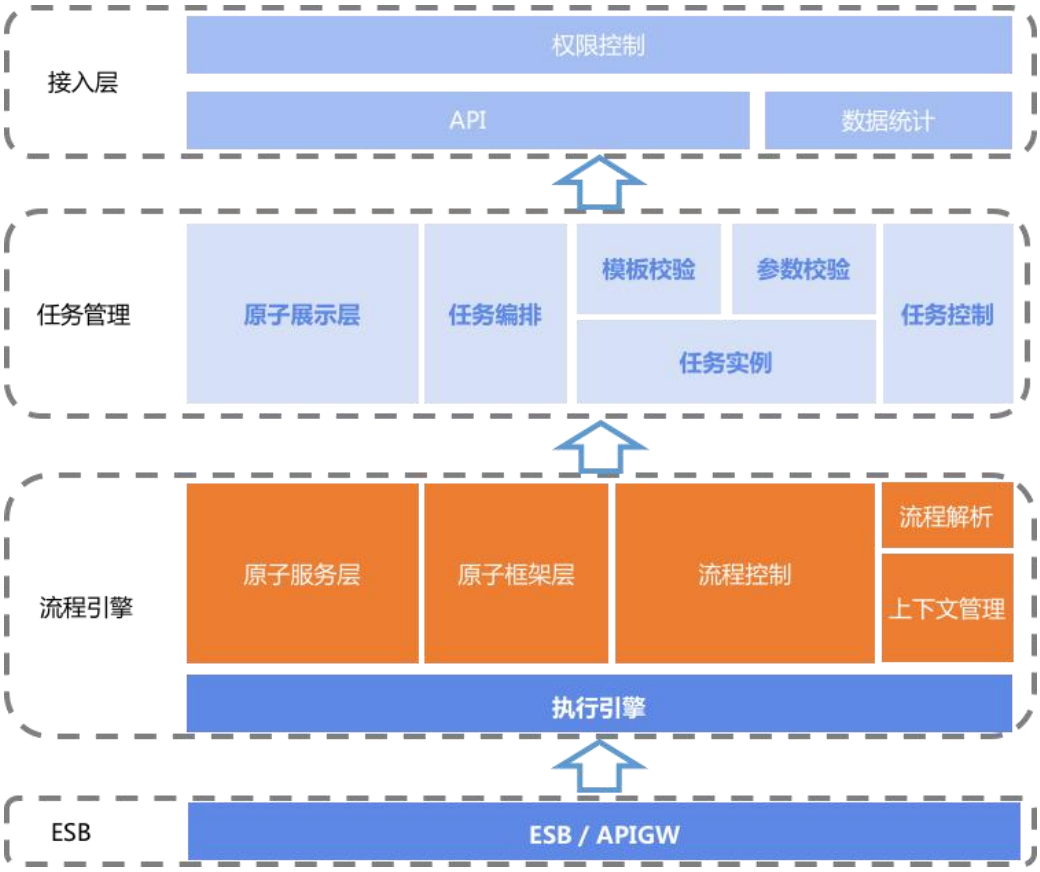
操作调度平台在研发保障过程中发挥了至关重要的作用，确保各项操作任务能够高效、准确地执行，并减少人为干预带来的风险。该平台主要负责以下几个关键工作领域：

1. CD 发布管理

在持续交付（Continuous Delivery, CD）过程中，操作调度平台承担着发布管理的核心职责。它不仅能确保从代码提交到生产环境

的全过程自动化和高效运作，还能够通过智能调度来优化资源使用和任务执行的顺序。

自动化流程编排：操作调度平台通过自动化流程编排，将代码提交后的各项任务（如编译、测试、部署等）整合到一个无缝的流程中，并自动调度这些任务的执行顺序，以避免资源冲突或执行瓶颈。通过与 CI/CD 工具链的深度集成，平台能够在任务之间进行动态调度，确保每个流程步骤都在最佳时间点执行。



智能调度算法：平台利用智能调度算法，根据任务的优先级、依赖关系和资源可用性，动态调整任务的执行顺序和分配的计算资源。这种调度策略不仅能够优化资源利用率，还能提高部署效率，减少因资源不足或冲突导致的延迟。

2. 批量化操作

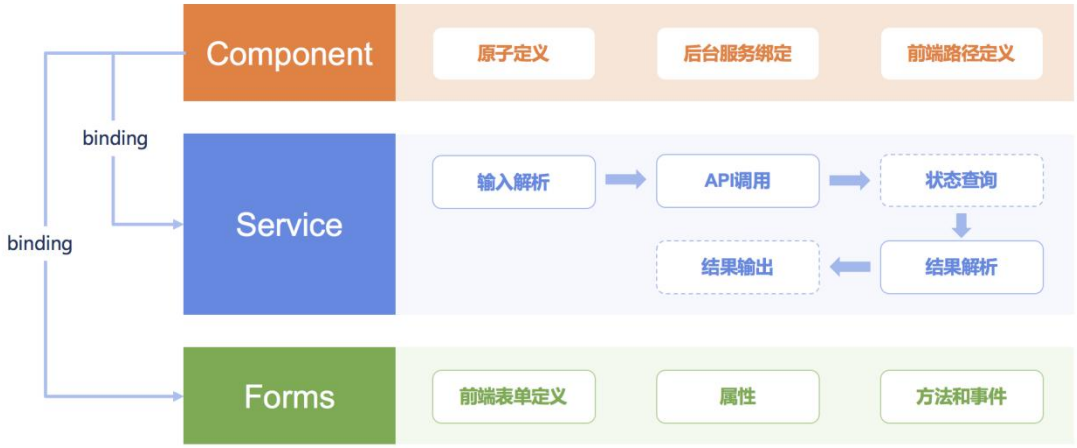
批量操作是操作调度平台的另一重要功能，它支持对大规模操作任务的统一管理和调度，极大地提高了研发团队处理日常运维任务的效率。

统一任务调度：操作调度平台能够将大量重复性、标准化的操作任务（如批量更新配置、批量重启服务等）整合到一套自动化流程中，并通过统一的调度机制进行管理。

并行执行与任务分片：为了提高批量操作的效率，平台支持并行执行和任务分片技术。它能够将大规模任务分解为多个小任务，并分配到不同的计算节点并行执行，从而显著缩短任务的整体执行时间。此外，平台还具备任务依赖管理功能，确保各子任务按顺序正确执行。

网址：SRE-Elite.com 微信：SRE 精英联盟

动态资源分配：批量操作过程中，平台能够动态分配计算资源，根据任务的实时需求调整资源分配策略，以确保在高峰期也能保持高效执行。



2.2.4 面向研发保障的 ITSM 平台

IT 服务管理平台（IT Service Management, ITSM）在研发保障中起到了至关重要的作用，为研发团队提供了统一、标准化的管理工具和流程，确保研发过程中的各类服务请求、事件管理、问题解决、变更控制等工作能够高效、有序地进行。

2.2.5 面向研发保障的容器平台

容器平台为研发团队提供了一个高效、灵活、可扩展的环境，以便快速部署、测试、迭代应用。

为了有效支持研发团队的工作流程，容器平台必须具备一系列专门设计的功能，确保研发过程的高效和稳定。

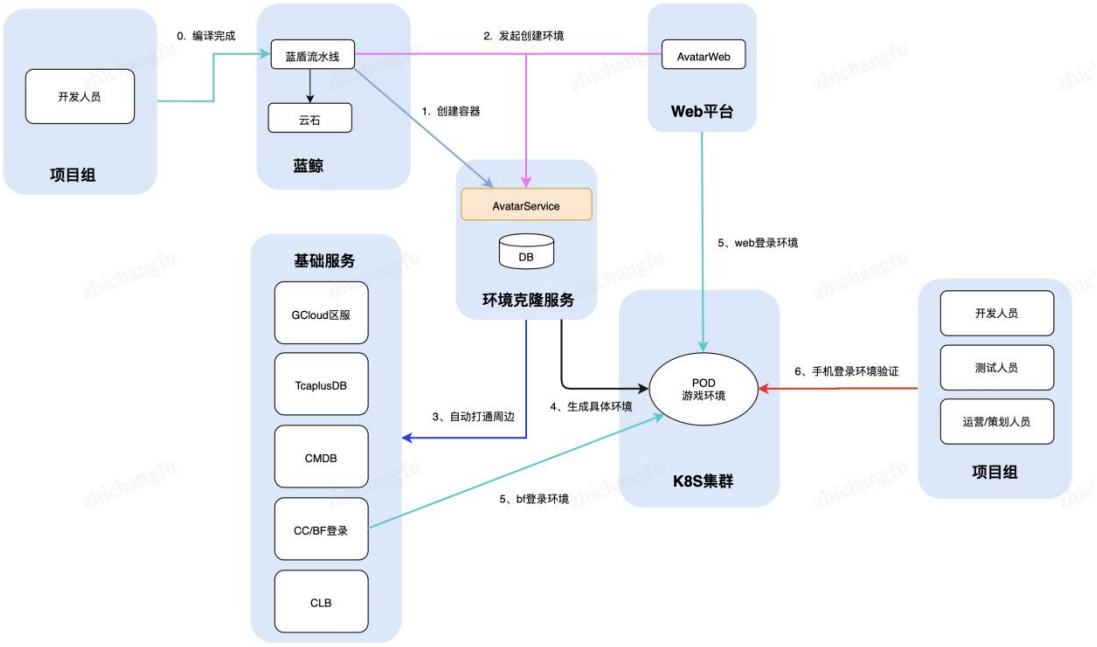
自动化 CI/CD 管道集成：容器平台与 CI/CD 工具链无缝集成，支持自动化构建、测试、部署。通过 Jenkins、GitLab CI 等工具，容器平台能够在代码提交后自动触发构建和部署任务，大大缩短了研发周期。

环境隔离与多租户支持：通过命名空间和 RBAC（Role-Based Access Control）等机制，容器平台可以实现环境隔离，支持多个团队和项目的并行开发。这种隔离不仅限于资源使用，还包括网络和安全配置，确保不同团队之间的独立性和安全性。

观测性平台对接：容器平台需要能与观测平台实现无缝的对接，将运行于容器的指标，日志，链路上报的观测平台中，并能实现监控告警。

敏捷发布与回滚机制：容器平台支持蓝绿部署、金丝雀发布等敏捷发布策略，能够在不影响整体服务的前提下逐步发布新版本并监控其表现。一旦发现问题，平台可以快速回滚到之前的稳定版本，减少对业务的影响。

测试环境治理：针对实际研发工作环节中，测试环境搭建流程过长，测试资源无法充分利用和测试设备日常维护人力成本高的问题，基于容器平台，以 k8s 工作负载的方式部署运行管理多个测试环境。具体架构如下

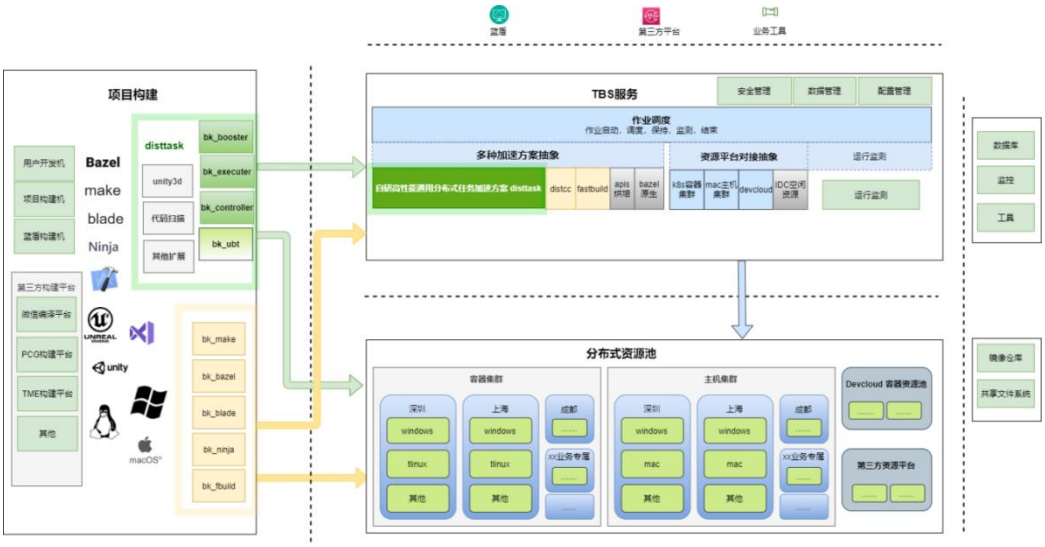


2.2.6 面向研发保障的编译加速平台

编译构建是项目开发和发布过程中的重要环节，同时也是非常耗时的环境，有些项目执行一次完整的构建需要几十分钟甚至几个小时，各种编译构建加速工具都能在一定程度上减小构建时长。

整体架构如下：

- 编译加速工具包运行于用户构建机，为构建机上的编译任务接入加速服务
- 分布式编译服务为集中式部署，负责接收编译加速工具包加速请求，调度加速资源，并负责加速任务过程全生命周期管理
- 分布式资源集群通过分布式编译服务引擎服务统一动态调度，直接为用户编译提供分布式加速



分布式加速底层通过 disttask 分布式任务引擎实现，disttask 各模块功能介绍如下：

- remoter worker 运行在分布式资源集群中，负责接收，执行和返回分布式任务
- local server 运行在构建机上，实现分布式任务底层基础功能，并可扩展不同应用场景的分布式任务实现
- disttask_executor 通用任务执行器，接管实际编译中的编译命令(如 gcc 命令，clang 命令)，是构建工具和分布式基础服务之间的桥梁
- 基于 disttask 提供的接口，根据实际场景需要，实现独立的构建工具

2.3 研发保障案例

2.3.1 腾讯游戏全球研发保障实践

SRE Elite 精选原因

这是一个完整的游戏行业研发保障案例。面对游戏研发中的复杂研发管线、大文件版本管理、冗长的构建过程和频繁的更新需求等挑战，SRE 团队通过稳定性保障、平台工具建设，以及与业务开发团队的有效分工，实现了高效的研发保障。

此案例覆盖了研发保障的多个关键模块，在代码可靠性，代码仓库可靠性、制品分发，以及构建加速等多个方面进行了优化，显著提升了代码提交和构建的成功率，并有效解决了代码库卡顿和文件分发效率低等问题。相关的优化内容非常详尽细节，具有很强的实践性，且大部分关键组件提供了开源的实现案例，非常值得参考。

（一）背景及设计原则

大型游戏研发对研发管线有独特的要求

伴随着腾讯游戏多元化的诉求日趋增长，越来越多的游戏开发团队开始尝试多地共研模式，这在工程版本控制、编译构建、制品共享等方面都提出了新的挑战。

游戏行业的研发具有以下独特特点和挑战：

1.复杂且研发管线：用户提交的不仅是源代码，还可能包含体积达数百兆的单个素材文件，这些文件需要进行版本管理，而且还需要对原始素材文件为不同终端进行渲染，耗时巨大，出包过程可能长达 5~6 小时。而如果中途构建失败，第二天项目进度就会受到影响。

2.版本库性能：版本库中包含大量大文件，整个代码库的体积可能超过上 PB，而且随着游戏生命周期的延续，代码库只会越来越庞大。以此未经优化前，用户在访问或修改版本库里头的资源时容易卡住，而游戏团队的规模可能达到上百人，人力成本极其高昂，如果卡住一个小时损失就会非常巨大。

3.项目团队协作及代码质量问题：目前游戏的制作已经进入工业化生产的阶段，每个游戏项目可能都有几十人甚至上百人的参加，如何解决由于参与人数众多而带来的代码冲突以及代码质量不统一问题。

4. 业务制品构建耗时长，严重影响协同效率和敏捷开发。对于普通互联网应用安装包普遍在 500M 以内，单次构建在 30 分钟以内，游戏安装包和资源文件加起来一般可达 5G-20G 以上，客户端包单次构建耗时需要 2~5H 左右；

5.激进的发布时间：游戏行业竞争极其激烈，游戏内部经常衍生出新的玩法和素材，用户期望极高，玩家口味变化极快，前期宣发费用极其高昂，项目必须如期上线才能减少前面投入的风险。

6.频繁的更新：根据用户反馈，游戏需要频繁出包进行调整，并引入新玩法、新皮肤和新角色，

7.制品分发效率及安全：现在的游戏动辄几十到上百 GB 的包， 如何进行快速，稳定地分发，送到大量的测试及渠道用户。如何保障制品的安全。

8.多方协作带来的安全问题：游戏行业已经从小作坊，演变成类似于好莱坞式工作室的分工协作模式，大量的资源需要通过外包的形式进行分包协作。由此带来安全问题非常突出，不少的竞争对手或者说相关的产业都会盯上我们的整体的研发流程，在游戏新版本的研发过程中，一旦出现了资源的泄露，玩法的泄露，数值的泄露，有可能就会导致竞争对手或者黑产的狙击，很有可能导致版本需要重做， 数百人月的工作量付诸东流。如牵涉到核心资产， 如核心代码， 核心引擎等，甚至会出现私服的情况，影响整个游戏的未来的生存。

网址：SRE-Elite.com 微信：SRE 精英联盟

举个例子，某日凌晨 1 时许，由于机房故障，部分流水线的构建机下线，我们本以为这么晚应该不会有用户进行构建，本决定第二天再修复。结果，凌晨 2 点和 3 点就有用户在公司论坛进行投诉，据了解，因为代码改动直接到了当天凌晨 1 点多，因此他们的构建时间只能选择在凌晨 2 点，才能满足版本在当天进行提测发版的进度要求。此事让我们更深刻理解到**研发管线的可靠性对业务的重要价值。**

总之，解决以上问题，做好研发保障，保障研发管线的高效安全运行，价值巨大，挑战巨大。

SRE 团队负责研发保障是一个合适的选择

研发阶段的可靠性是指，以 SRE 理论驱动研发的可用性建设，提升研发管线的工业化水平，保障版本能够按正常周期迭代，从而实现高质量持续交付有效价值的目标

从软件工程的角度：

- SRE 的稳定性保障工作，本质上是软件工程的一部分，也就是运维左移，从源头保障软件的质量

从 SRE 专业能力的角度：

- 利用 SRE 团队既有平台能力又有服务能力的特点，从平台和服务两个地方切入，建立 SRE 团队主导的工具平台和服务体系，让研发团队在平台和服务上需求都得到最大化的满足。

- 服务上，SRE 团队做研发服务，可以复用现有的运维支持体系，与研发相比，SRE 有高度的业务可用性意识和能力，能很好地保障业务研发工具链的可用性，防止因为工具链的不可用或性能低下，影响业务团队研发效率。
- 工具上，SRE 团队可以复用，开发，并扩展在 CO /CD 领域的工具去进行研发保障，例如，利用可观测的平台，对研发工具链的健康状态以及使用体验进行观测，度量研发工具链上下游健康状态，故障时快速定位根因，同时提供成熟的可观测落地方案，标准化实施，降低建设成本

从组织分工上的角度：

- SRE 团队负责研发保障，业务开发负责产品功能开发，对于企业来说，可以发挥最大的投入产出比。SRE 可以更体系化和规模化进行全面保障。业务开发的本质逻辑是更加聚焦快速开发迭代版本，完成产品功能开发。理想状态下，业务开发无需关注研发平台上下游所有组件的稳定性。
- 从职业发展来说，SRE 工程师在 SRE 组织中，有明确的成长路径和岗位通道，且通过从 CO/CD 扩展到 CI 域，扩展了职业发展方向，并且与相关能力设置在业务开发团队相比，更具有稳定性和成长性。

简单来说，SRE 团队是既有服务意识，又有平台及产品的建设能力，从软件工程的角度整体保障角度，完成了业务研发不擅长但是有迫切需求的工作。因此，我们认为由 SRE 工程师去从事研发保障的工作，是一个高 ROI，合乎组织利益及各方干系人利益，能提升研发效率，可持续的解决方案。

本案例最后总结部分，会有我们业务研发同学对 SRE 从事专业度的认可和反馈。

在此基础上，SRE 切入研发保障，需要注意以下三点：

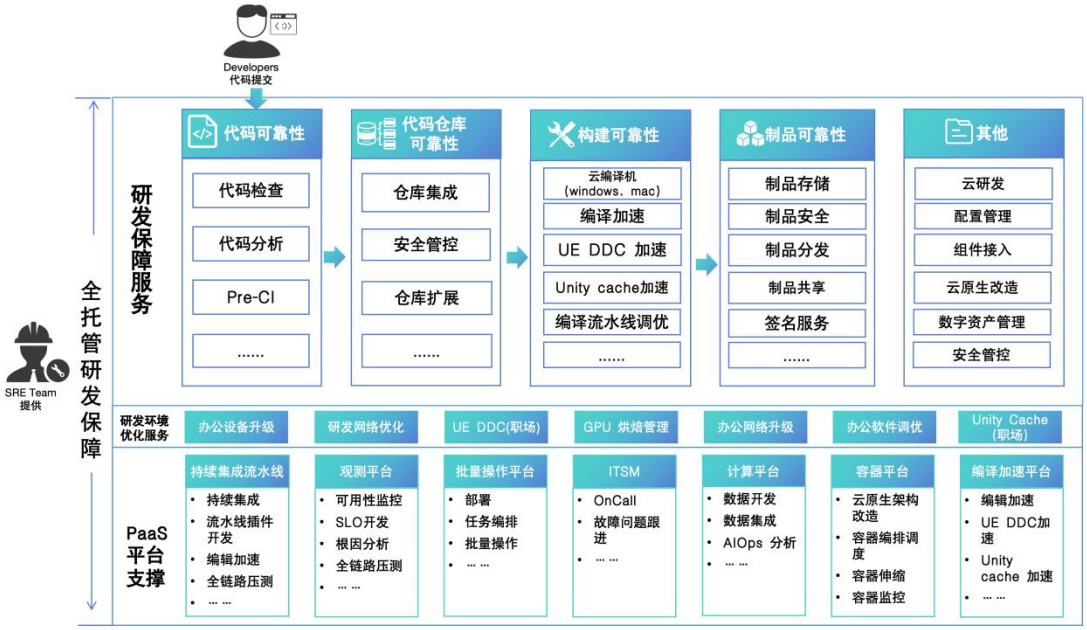
稳定高效：需要 SRE 团队在服务上有韧性，和研发团队建立信任。

低成本定制：SRE 团队能够低成本地为每一个业务定制研发流水线方案，定制个性化需求，例如兼容多种代码库等这就要求我们的方案必须是 PaaS 模式，也就是平台工程的效果。

安全可靠：一方面是研发团队对代码安全的要求，例如外包开发团队的代码安全。我们通过离岸云研发模式解决另一方是研发环境的安全性，避免代码泄露事件发生。

平台能力和服务能力方面，做到稳定高效，低成本定制，安全可靠，让用户对 SRE 团队提供的研发服务产生依赖，用户用得越深入，与 SRE 团队的绑定就越紧密，SRE 团队的研发服务就会可持续，从而建立长期服务关系。

（二）体系设计及关键流程



SRE 团队提供的全托管研发保障体系，涵盖从代码提交、代码仓库管理、构建流程到制品分发的全过程。通过代码分析、仓库安全管理、编译加速、制品共享等多项服务，确保了研发过程的高效性和安全性。大部分情况下，研发只需要往平台提交代码后，后继的环节将不需要进行干预，即可高效获取到相应的制品。

图中还包括由 SRE 团队研发并提供的支持的 PaaS 平台，SRE 团队利用相关的平台能力，提供持续集成、观测、批量操作、ITS 管理、计算和容器平台等功能，进一步优化研发环境。

整体体系强调 SRE 团队在提升研发效率、保障流程稳定性和安全性中的关键作用。

备注：案例中可能会提及各个平台的简称，此处进行统一进行备注说明：

●蓝盾：蓝鲸持续集成平台的简称，开源地址

https://github.com/TencentBlueKing/bk-ci

●标准运维：蓝鲸智云标准运维的简称，开源地址

https://github.com/TencentBlueKing/bk-sops

●ITSM：蓝鲸流程服务的简称，开源地址

https://github.com/TencentBlueKing/bk-itsm

●容器平台：蓝鲸容器管理平台简称，开源地址

https://github.com/TencentBlueKing/bk-bcs

●BKTurbo：蓝鲸编译加速平台简称，开源地址

https://github.com/TencentBlueKing/bk-turbo

●CodeCC 代码检查：CodeCC 插件(腾讯代码分析插件），开源地址 https://github.com/TencentBlueKing/ci-CodeCCCheckAtom

●制品库：蓝鲸制品库简称，开源地址

https://github.com/TencentBlueKing/bk-repo

（1）代码可靠性研发保障实践

某游戏客户端通过质量红线保障各阶段代码质量

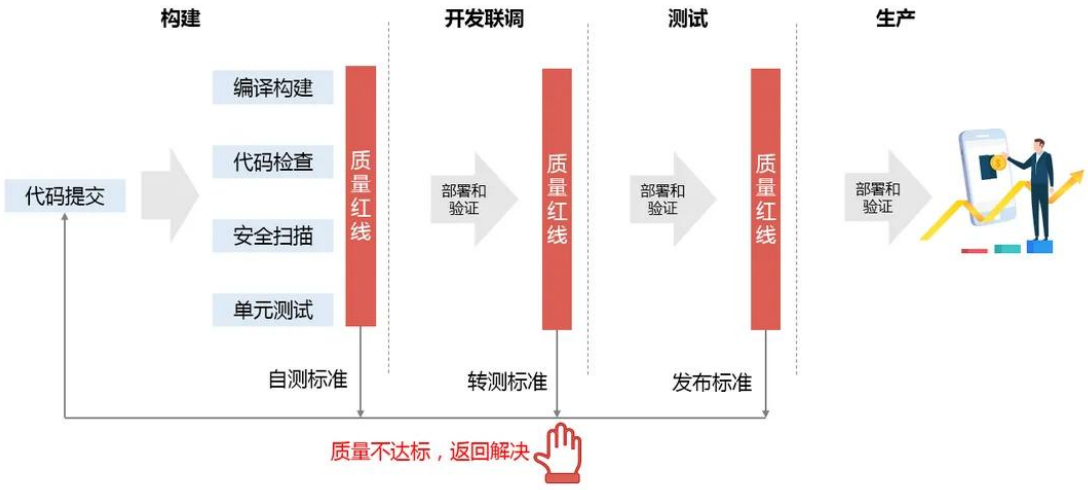
在某游戏客户端业务研发过程中，我们经常会有类似的困惑：

- 确立了团队的编码规范，但还是有不符合规范的代码被合入；
- 拥有自己的转测试标准，但是只能人工跟进，不能落实到自动化流程中；
- 在版本的后期，Bug 数居高不下。但很多问题是可以在前期通过代码检查和单元测试发现的。

为了解决上述问题，借鉴丰田精益生产的思想，腾讯打造了质量红线 Gate 服务。

质量红线的思想起源于丰田精益生产的立即暂停系统（stop-the-line Andon），又称为安灯（Andon）系统。当车间生产流水线上的员工遇到麻烦，立即拉一下信号灯，班组长就会立即跑过来帮助解决，其生产流水线也会停止直到问题解决。这样能够尽早暴露问题，解决问题，而不是把问题流到生产汽车的后续步骤。

解决方案



质量红线是指通过设置质量标准，控制流水线的行为，使得每一阶段的出口质量都必须符合质量标准的一种服务。

设计思想

设计思想如下

1、独立成质量红线服务，基于关键点控制质量

从用户使用场景出发，将开发、部署测试环境、部署生产环境等相关的插件（共 10 个）选出作为关键控制点。这些控制点插件在流水线中可以非常灵活地编排，但只要质量红线设置了标准，就必须质量达标控制点才能执行通过。

2、指标灵活可扩展

设计了统一的指标定义规范和指标灵活可扩展的机制。主要包括系统插件指标（主要是 CodeCC 代码检查）、脚本任务指标、 研发商店插件指标；

3、支持为代码合入设立红线

支持设置质量要求，并通过流水线 Git/Github 事件触发机制与工蜂/Github 串联起来。以工蜂为例，当用户在工蜂发起 MR 时，会通过 Merge Request hook 触发流水线执行。流水线执行拉取代码、代码检查、单元测试等操作。若不满足质量要求，质量红线就会让流水线失败，并将失败的结果回写到 Code，作为 MR 的辅助决策。

4、与流水线融合

质量红线被设计为一个独立服务后，虽然有集中管理的便利，但用户对其感知度不高。只有用户正确配置了红线之后，才能在流水线看到，较难被用户发现。且用户创建红线时各项信息只能一项项填写，有时不知道怎么填比较好。为了用户更好地使用它，我们在设计上将红线和流水线服务进行了融合。

关键流程包括：

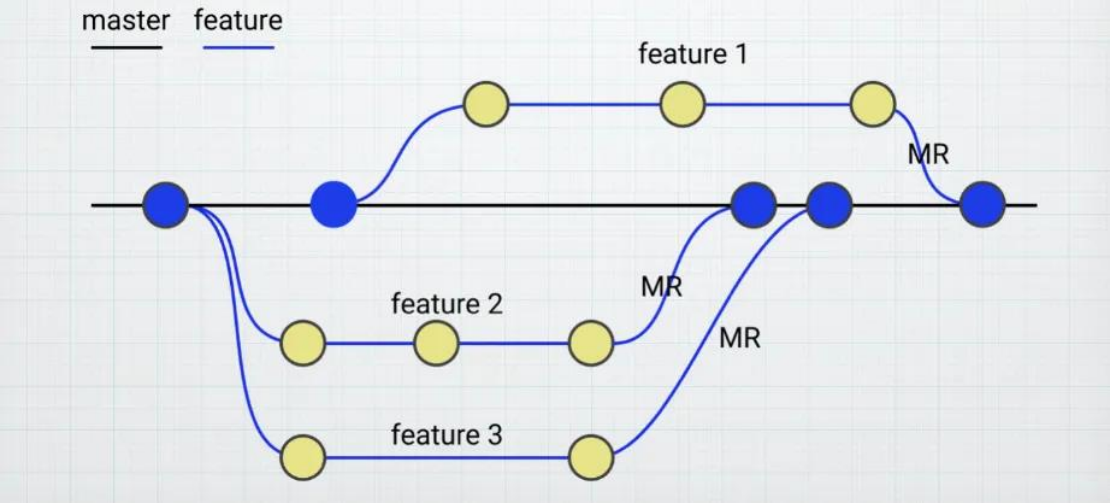
1 确立团队 Git 工作流

Git 相比 SVN，其优势不仅在于是一个分布式版本控制系统，而且在于其强大的分支管理能力。其中 Git 工作流，主要是需要定义好分支策略。

在项目初期，团队只有 2~3 人。此时往往不需要额外分支，一个 master 打天下。但是随着团队逐步壮大，大家发现代码冲突越来

越严重，同时由于缺少代码检视（Code Review），部分质量较差的代码和无关内容也时不时被提交上去。

为了解决这个问题，“分支开发、主干发布”模式呼之欲出。团队从主干上拉出分支，并在分支上开发软件新功能或修复缺陷。当某个分支（或多个分支）上的功能开发完成后要对外发布版本时，通过 MR（Merge Request）合入主干。在 MR 时，进行 CR 代码检视。



2 确定团队质量标准

为了保证合入 master 的代码质量，团队需要设立自己的代码质量标准，例如编码规范、单元测试等。但是单单依靠人肉代码检视 CR，很难保证新合入的代码都符合规范。而工具自动化执行，能够做到客观准确、持续检查。因此我们可以将质量标准，放到腾讯 DevOps 平台提供的一系列工具之中。

以腾讯代码检查平台为例，可以支持五个维度的代码质量标准。像缺陷、安全和代码规范，都能通过规则配置来进行具体调整，以便适应不同团队的要求。

缺陷：BKCheck 工具可检查出空指针、内存泄漏、数据越界、并发缺陷等问题；

安全：敏感信息工具可以检测密码泄露、内部 IP 泄露等问题；

代码规范：九款工具可以检测出逻辑、变量、代码风格、最佳实践等代码规范问题；

圈复杂度：可以检查出过于复杂的函数。函数复杂度越高，存在缺陷的风险越大；

重复率：可以检查项目中复制粘贴代码片段等问题，避免产生大量冗余代码。

以腾讯 DevOps 平台上的 Bash/Batch Script（脚本任务）为例，支持用户自由编写自己的脚本。例如可以支持单元测试，将单元测试用例执行失败数作为团队要求，也能支持获取 Tapd（腾讯内部需求管理/缺陷平台）遗留缺陷数作为质量要求。

3 配置流水线和质量红线

在确定了这些质量标准之后，如何将其与 Git 工作流无缝衔接起来呢？

3.1 创建一条 MR 触发执行的流水线

网址：SRE-Elite.com 微信：SRE 精英联盟

首先，我们需要配置一条 DevOps 流水线，它能在发起 MR 时自动执行，拉取相应代码进行分析。

添加 Git 事件触发的插件，并将事件类型设置为 Merge Request Hook。当工蜂项目中新增 MR 时会触发该流水线。

添加拉取 Git 代码的插件，并将分支名称设置为 ${hookSourceBranch}。${hookSourceBranch}是一个流水线变量，指代源分支。例如从 feature1 分支向 master 合并代码，那么 feature1 就是源分支，master 是目标分支。因为源分支在 MR 中可能会变化，有时是 feature1，有时是 feature2，有时是 feature3，因此使用流水线变量更加灵活方便。



添加 CodeCC 代码检查插件，选择语言和规则集。例如可以选择 IEG 缺陷检查、腾讯代码规范，还有啄木鸟安全检查、重复率和圈复杂度等。



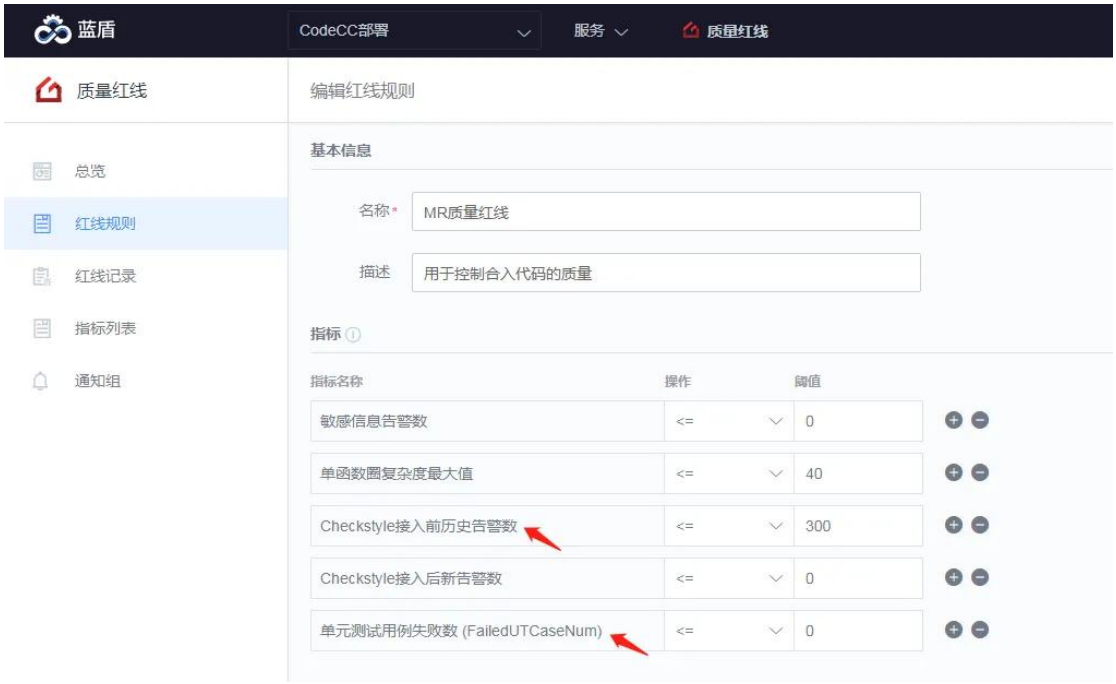
3.2 配置质量红线

接着我们需要给这条流水线配置一条质量红线。通过质量红线，我们可以将团队代码质量标准固化到流程之中。以某 Java 项目为例：

指标 设置如下图。

对于 CheckStyle，一般接入后首次扫描会产生不少告警，全部修复清零较为困难，因此这里设置的指标为“Checkstyle 接入前历史告警数<=300，Checkstyle 接入后新告警数<=0”，保证新增代码都是符合代码规范的。

单元测试用例失败数 (FailedUTCaseNum)是一个自定义的指标。用户也可以在指标列表中自定义其他指标，在 Bash 脚本任务中通过 setGateValue 函数将其上报给 DevOps 平台的后台，用于质量红线的判断。



4、Git 平台结果展示

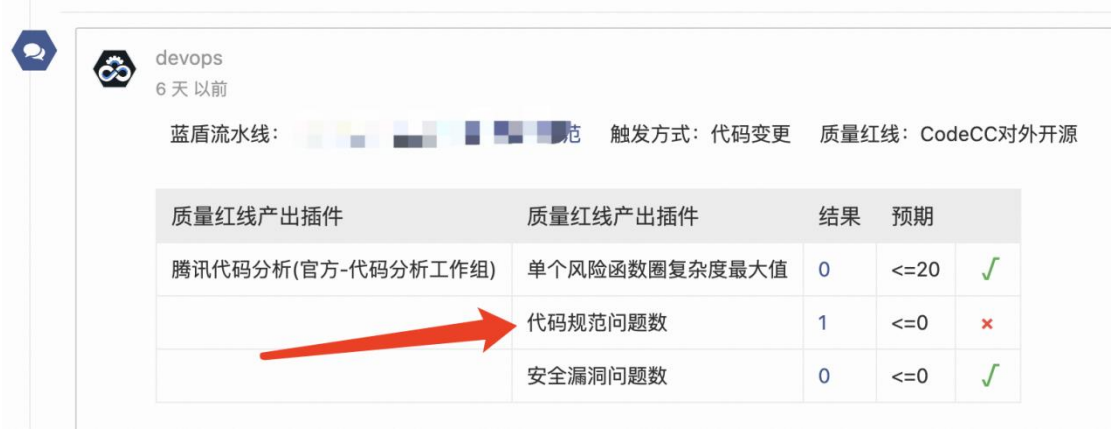
当我们在工蜂发起 MR 检查时，此时会触发流水线执行。流水线会自动拉取发起 MR 分支（源分支）的代码，进行单元测试和代码检查。



如上图所示，由于敏感信息告警数没有符合质量红线的要求，导致流水线执行失败。此时，DevOps 平台会与 Code 平台联动，将结果同步到 Code 平台。

执行结果第一部分是流水线整体执行结果。它会在工蜂 MR 页面的顶部显示，包括流水线名称、时间、状态和详情链接，点击“详情”，可以跳转到 DevOps 平台流水线详情页面。通过这个流水线整体执行结果，评审人能够快速判断出待合入代码（源分支）是否符合了团队质量要求。





5、处理告警

为了使得代码能够符合团队的质量要求，保证代码能够被合入master，我们需要修复和处理质量红线拦截指标的告警。



（2）代码仓库可靠性研发保障实践

研发工具链平台众多、监控建设程度参差不齐，无法度量整个研发工具链的健康状态（运行是否正常、体验好不好，故障时快速定位根因），需要一套统一的研发基础设施可观测体系和配套的保障方案，为业务提供稳定可靠的研发工具链服务。

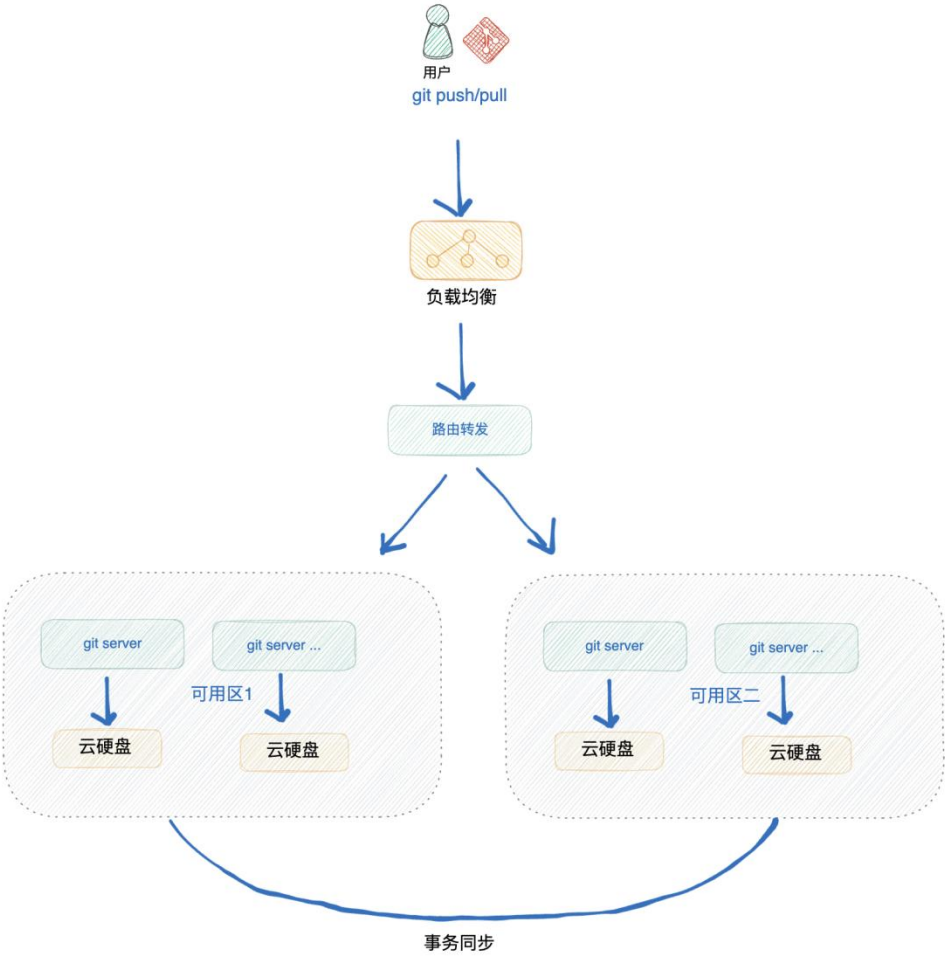
以下是一个缺少代码仓库可靠性保障的案例，用户频繁反馈代码仓库卡顿。



Git 代码仓库可靠性案例

部署架构设计

以下是某公司的代码仓库部署方案。



仓库容灾

如上图的部署架构图，在多个可用区部署 Git 服务器，根据路由转发将数据多写到多个可用区。

在多个可用区间保持事务同步，确保数据的一致性。当其中一个可用区出现网络故障时，另一个可用区仍然可以独立提供读写服务。

大文件加速技术方案

架构设计

Git 本身不善于处理大型二进制文件，因为这些大文件通常会导致性能问题、仓库体积的急剧膨胀。Git LFS（Large File Storage）是一个 Git 管理大文件的扩展，Git 版本仓库仅需维护大文件指针，大文件本身存储在文件服务器，通过这种方式可以在 Git 仓库中高效管理和存储大文件。

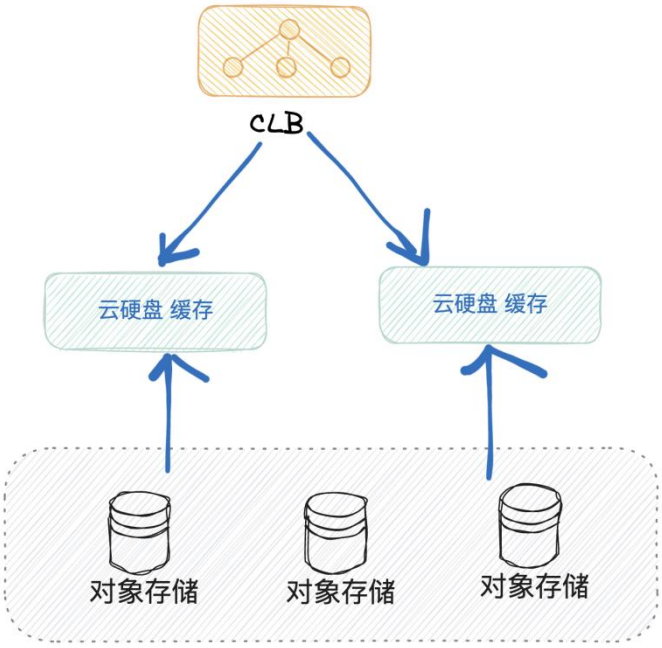
某项目 Git 代码仓库 LFS 加速案例：

改造前：

- 项目前期使用 云文件存储（Cloud File Storage，CFS) 作为 LFS 的文件服务器，随着业务发展遇到存储瓶颈和 CFS 带宽瓶颈，导致文件下载性能急剧下降。

改造后：

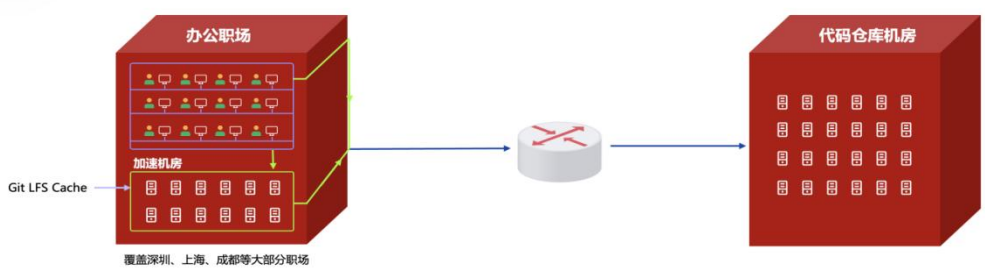
- 使用对象存储 COS 作为 LFS 文件服务器，可以实现单个存储桶容量在 PB 级别，通过创建更多的桶对存储容量进行平行扩展，同时可以复用 COS 的异地容灾能力，实现 LFS 文件服务器异地多活。
- 在 LFS 之上提供使用 高性能云硬盘（Cloud Block Storage，CBS）作为缓存节点，基于一致性 HASH 将相同文件请求到相同的缓存节点，进一步提升请求速度。



边缘加速

文件的下载速度和物理距离有强相关性，尤其 Git 仓库中的大文件，Cache 加速的原理是在办公职场的就近位置建设加速节点，从而提升 Git 大文件的下载速度。

在某公司的实际案例来看，相对于从直接从代码仓库拉取，平均速度提升一倍以上。



可观测工程

通过建设 Git 服务的可观测工程，我们可以度量和保障 Git 的服务质量，以下是某公司 Git 服务可观测工程的案例。

基于 6.7 应用服务 SLI/SLO 章节的实践方案，建设研发保障领域 Git 的可观测工程。

1）SLO 整体服务水平

计算公式：

服务可用性 =（1 - 不可用的总时长 / 总服务时长）* 100%

服务不可用：指标计算结果低于目标，则该类别提供的服务不可用



2）对应 SLI 指标

a) 用户拉取和推送代码服务可用率，预期>99%



b) 页面访问服务可用率，预期>99%

计入不可用需要满足的条件：可用率<99%且请求量>1000



SVN 代码仓库可靠性案例

SVN 仓库容灾——主从架构

SVN 选择成熟的主从架构来进行容灾。

1. 主服务器（Master）：主服务器负责存储和管理主要的版本库。所有的提交操作都首先发生在主服务器上。
2. 从服务器（Slave）：从服务器是主服务器的一个副本，它复制了主服务器上的所有数据。从服务器的主要作用是备份主服务器的数据，以及在主服务器出现故障时提供访问服务。
3. 同步过程：为了保持主服务器和从服务器之间的数据一致性，需要定期进行同步操作。同步过程可以通过 SVN 的 svnsync 工具

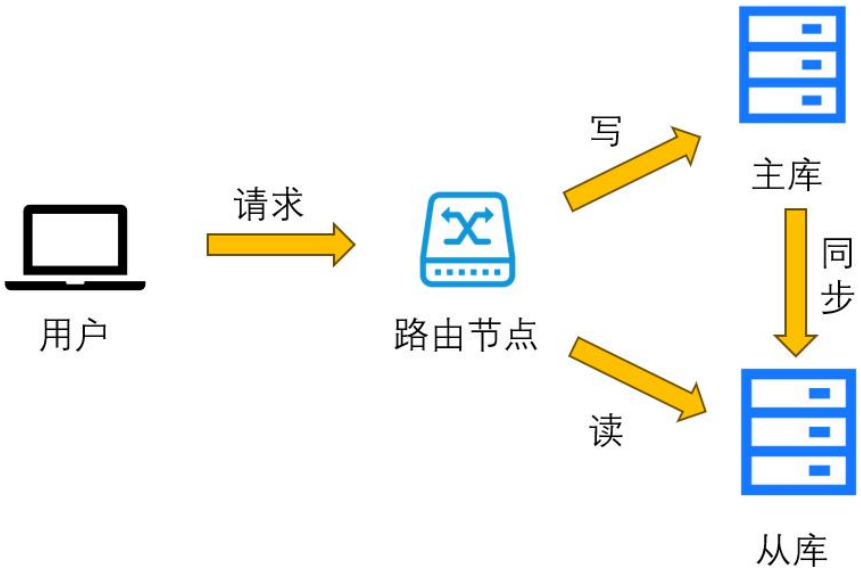
来实现。同步操作会将主服务器上的最新更改复制到从服务器上。

4. 故障切换：当主服务器出现故障时，可以将从服务器提升为主服务器，继续提供服务。

SVN 仓库性能

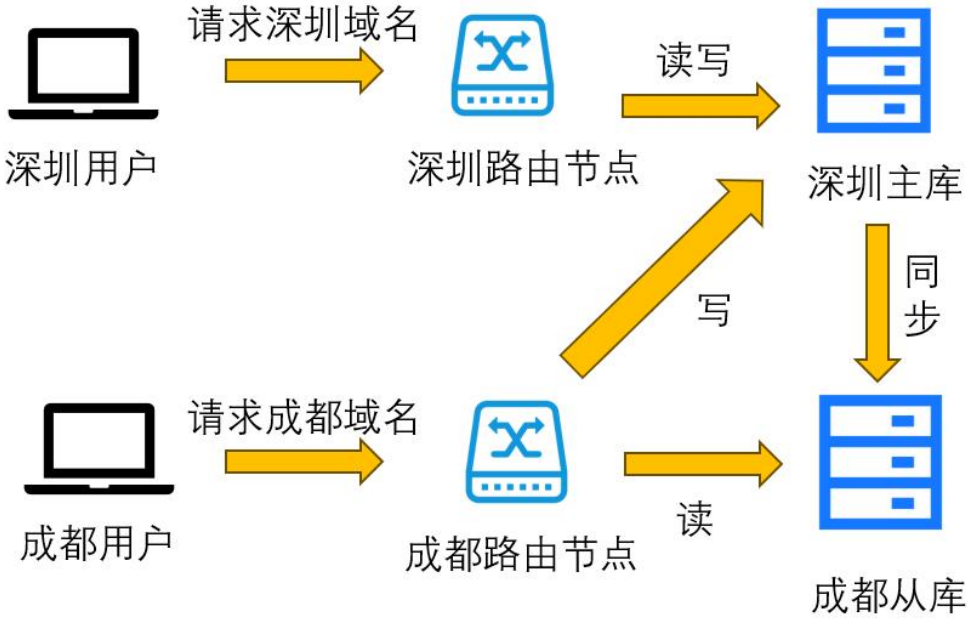
1. 读写分离

保证主从一致性后，可以在 Server 前面添加路由节点，将用户请求转发到从库，实现读写分离。



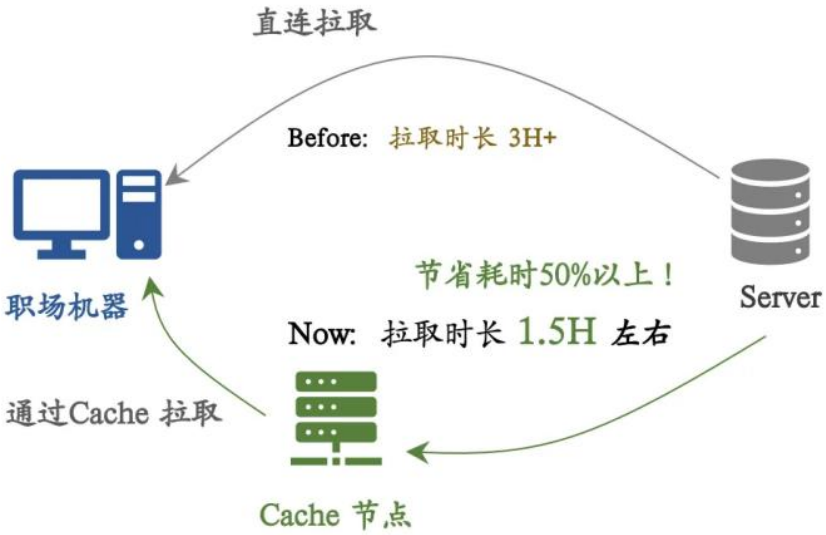
2. 就近读取

对于跨地域集群的仓库，现在支持一个集群部署多个跨地区的节点，集群内自动同步。用户通过访问当地的域名，实现就近访问和跨地区协作。



3. 边缘加速

边缘加速服务简单来说就是通过在职场办公区域楼层或者大厦，就近部署一个 Cache 节点，当有内容下载的时候，会自动地缓存到 Cache 节点。后续有其他同职场的同事再次进行拉取的时候，将会优先从 Cache 节点进行拉取，这样一来，不仅能够享受局域网的下载速度，也能分担多人下载时服务端的压力，让服务更稳定，同时下载速度也会更快。



4. 存储冷热分级

随着整体业务不断稳步前进，SVN 仓库的数量不断增加，其所需存储计算资源也随之上涨。在降本增效的大环境下，成本飞涨、资源利用率低等问题显露无遗。对此，我们针对所有仓库进行冷热分级，确保在不影响用户正常使用的前提下，尽可能地压榨服务器性能，提升资源利用率，节约运营成本。

整体方案



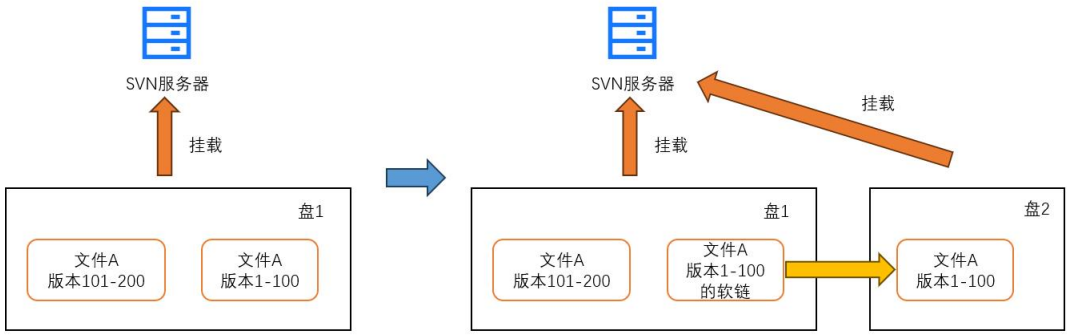
数据采集：对仓库数据使用情况进行多维度采集

热度分析：分析仓库数据热度情况，进行标记

冷数据迁移：对标记的冷数据迁移到低成本存储集群

SVN 仓库可扩展性——版本分片

一些项目经过多年发展，仓库容量已经快达到单块盘存储容量上限了，我们采取了称之为“版本分片存储”的方式，挂载多块盘+软链来解决这个问题。



如上图所示，我们通过在服务器上新挂载盘 2，将老的版本文件迁移到新挂载的盘 2 上，并在原来的盘 1 上创建软链，软链指向新挂载盘 2 的文件夹路径。

当用户访问 SVN 仓库时，还是访问盘 1 的文件，大部分时间都是访问较新的版本文件，如果访问到旧的版本文件时，还能通过软链访问到迁移到盘 2 的文件。

SVN 仓库安全

源代码对企业的重要性不言而喻，因此加强源代码保护至关重要。可以从源代码本身、内部企业人员权限和做好监控审计等三方面着手，进一步加强企业重要源代码的安全性。

一、源码分级

对源码进行分级，确保和明确重要源码的保护措施。

企业内部源码具有优先层级，明确哪些核心代码需要被保护。对于保密等级高的代码：

1、授权范围：仅仓库的项目成员有权限访问。

2、禁止通过任意方式突破代码仓库原有的授权范围，包含不仅限于 fork、push 到其他仓库等。

3、禁止在无权限访问的范围保存、发布、讨论、传播。

4、禁止下载或保存源代码到非开发环境机器，例如本地办公机、个人电脑、U 盘等存储媒介以及云平台（如各类网盘、网络笔记等）等。

二、精细化控制

精细化访问控制，对于员工的权限进行限制。

为不同角色分配访问权限应遵守最小授权原则，根据场景设置必要权限；

针对转岗或职责调整员工应及时收回相关项目代码仓库权限并清理本地保存代码；

对于非项目参与同学不应该给予项目访问权限，特殊情况应该邮件申请报备；

任何人如果需要调用其他工程的代码，在不必要的情况，不提供源代码，应采用 API 调用等方式；

三、监控和安全审计

配套建设代码仓库安全审计和监控系统

尝试越权访问时触发告警

大批量异常拉取代码时触发告警

代码拉取、提交记录长期可追溯

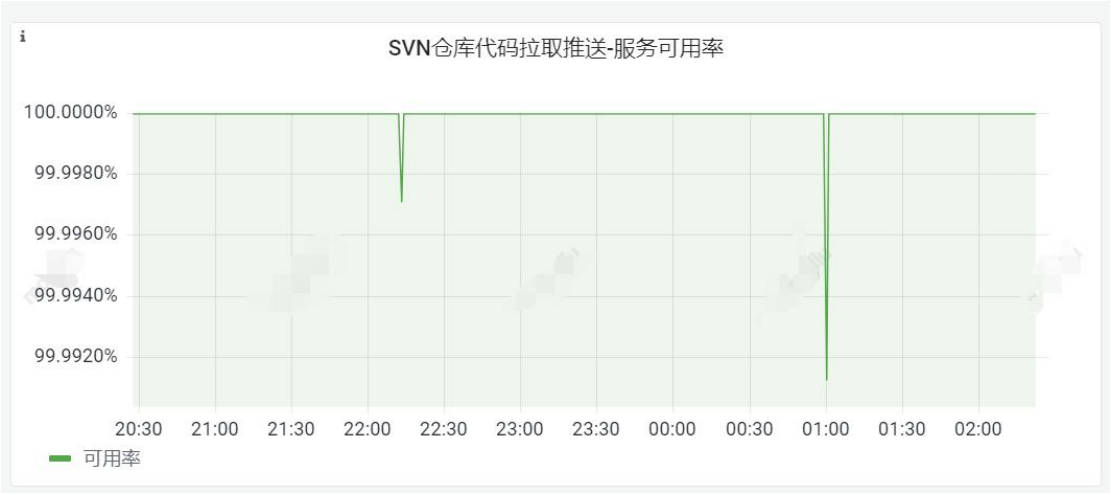
网址：SRE-Elite.com 微信：SRE 精英联盟

SVN 仓库可观测

1. SVN 仓库 SLI 定义

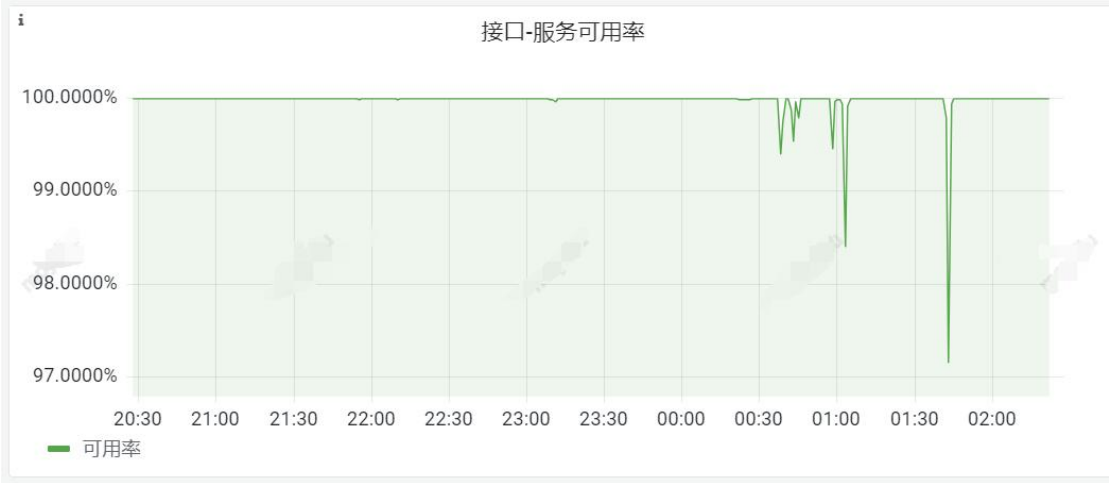
代码仓库提供代码拉取推送、页面访问、接口三种服务类别，按照下表定义考核目标

类别	指标	计算方式	目标
SVN 仓库代码拉取推送	服务可用率	（用户拉取和推送代码成功请求数 / 用户拉取推送代码总请求数）* 100%	>99%
页面访问	服务可用率	（用户请求成功数 / 用户访问总请求数）* 100%	>99%
接口	服务可用率	（接口请求成功总数 / 请求总数）* 100%	>95%



网址：SRE-Elite.com 微信：SRE 精英联盟





2. 可用性 SLO

● 服务不可用：指标计算结果低于目标，则该类别提供的服务不可用

● 服务可用性 =（1 - 不可用的总时长 / 总服务时长）* 100%



P4 仓库可靠性案例

P4（Perforce）仓库介绍

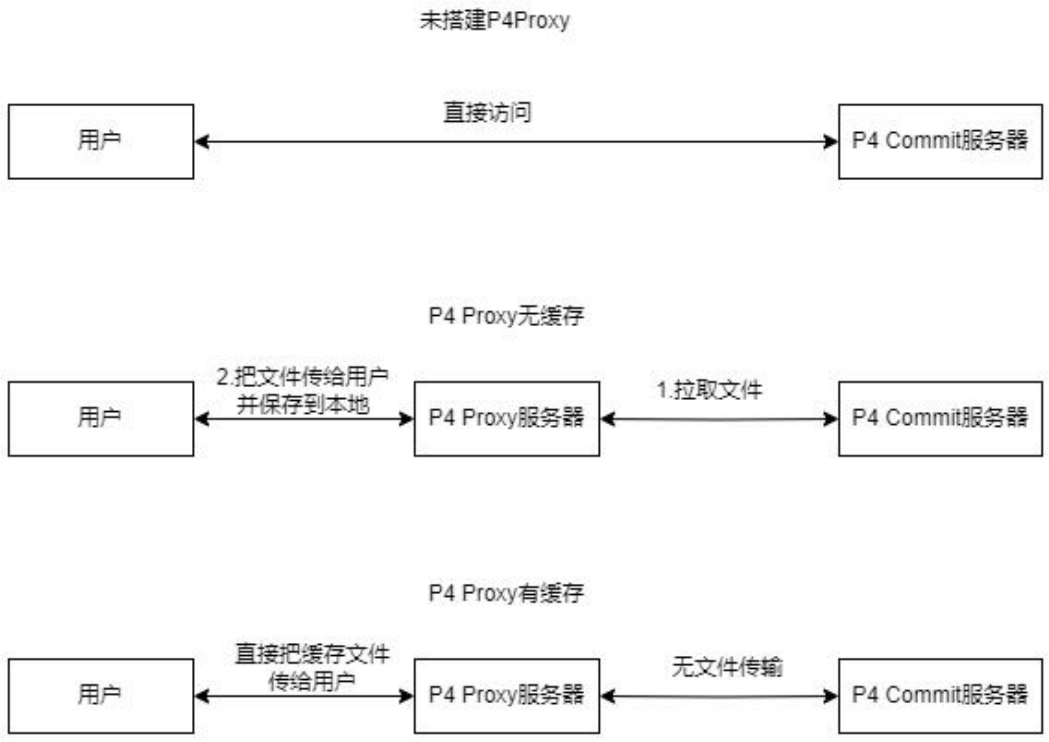
Perforce，简称 P4，是一款功能强大的集中式版本控制系统，广泛应用于软件开发、游戏开发、芯片设计和数字资产管理等领域。它提供了版本控制、工作空间管理、变更处理和分支模型等功能，支持跨平台操作，并且能够处理大型文件和二进制文件。国内使用 P4 的公司有阿里巴巴、腾讯、华为、字节跳动、长安汽车等，海外使用 P4 的公司有育碧、甲骨文、思科、英伟达、三星等。

注：Perforce 是 Perforce Software, Inc.的商标和产品，使用 Perforce 需要向 Perforce Software, Inc.购买产品使用授权。

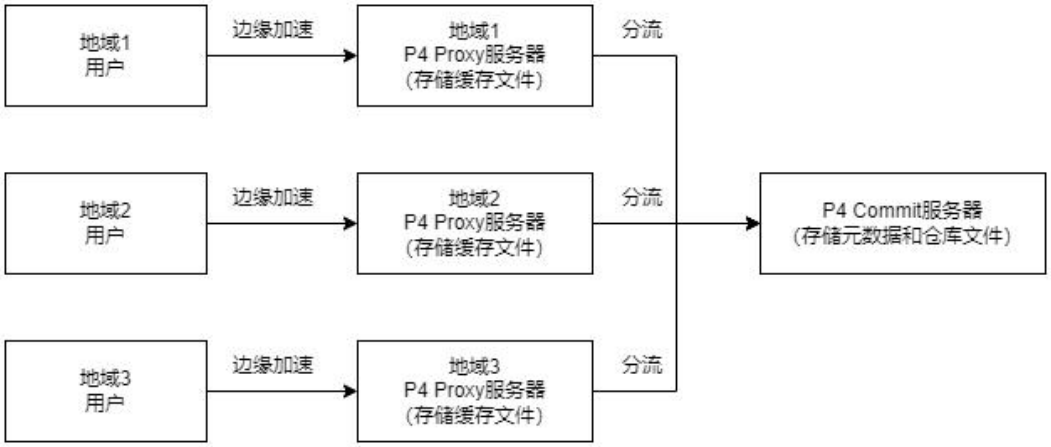
P4 仓库性能

1. 架构优化-Proxy

由于 P4 Commit 服务器（主服务器，存储所有仓库和提交记录）默认是单点服务的模式，当用户量很大的时候，就会遇到网卡和磁盘 IO 瓶颈，导致拉取很慢。此时可以搭建 Proxy 作为缓存服务，用户访问 Proxy，Proxy 发现本机上有缓存文件，就会直接返回给用户，而不会再去请求 commit 服务器，这样就减轻了 commit 服务器的压力。如果 Proxy 发现本地没有缓存文件，就会请求 commit 服务器，把文件拉下来存在本地作为缓存，并把文件传给用户。整个机制与 CDN 类似。



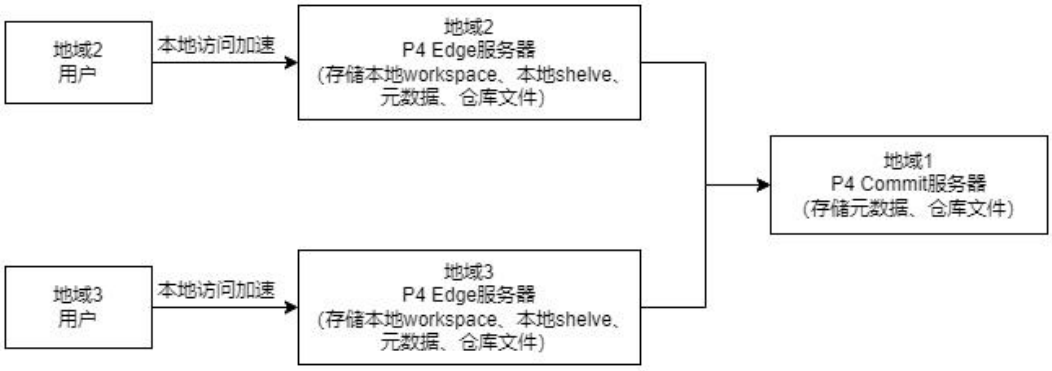
可以部署多台 Proxy，分流用户请求，提升并发承载能力，另外 Proxy 部署需要尽量靠近用户地域，起到边缘加速作用。



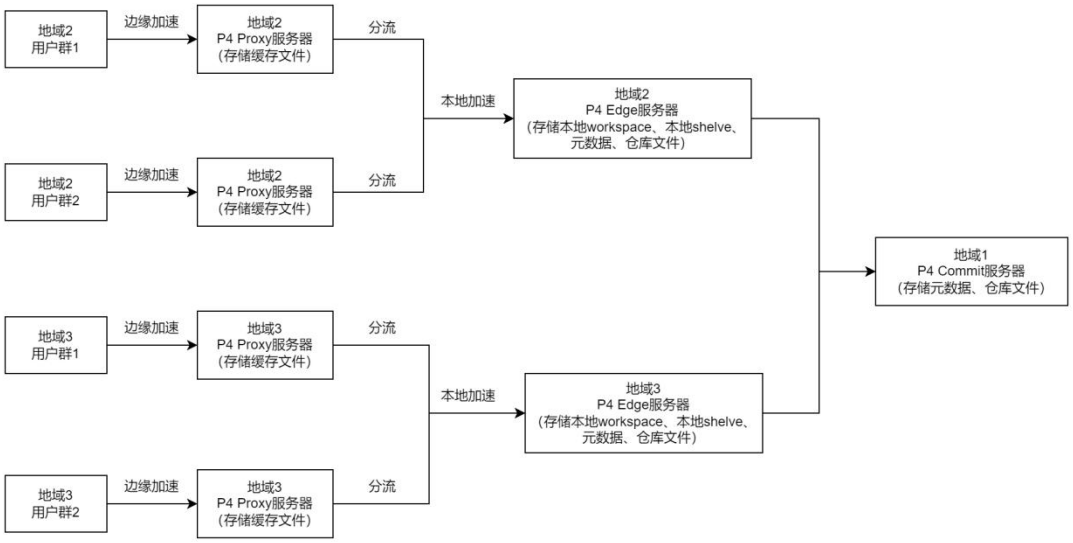
Proxy 预拉取：正常情况下，只有当用户拉取了代码，proxy 上才会有缓存，这样就导致一个新文件创建后用户首次拉取的时候是没有缓存的，会回源到 commit 给 commit 造成压力，我们可以在服务器上定时用程序来模拟用户拉取 P4 文件，生成缓存，这样真正用户访问的时候就没有回源压力了。

2. 架构优化-Edge

Proxy 在远距离（跨城、跨国）大批量小文件传输时表现不佳，因为仍需要访问 Commit 去执行大量的 metadb 数据读写操作，此时可以在用户本地部署 edge 来进行优化。Edge 把 workspace 和 shelve 数据放到了本地，这样拉取时只需要读写本地 Edge 的 metadb，大幅提升效率。



Edge 还可以搭配 proxy 使用，用户先访问 Proxy，Proxy 连接 Edge 进行缓存。



3.存储优化

p4d（commit/replica/edge）的数据文件主要分为 3 类：DB 文件（db.*）、事务日志（journal）、仓库文件（depot），三类文件有不同的要求。

1. DB 文件（db.*）

- 内容：版本控制与管理用数据表

- 特点：超低延迟高带宽的随机读写
- 设备选择：高性能 SSD

2. 事务日志（journal）

- 内容：事务流水日志，可搭配 checkpoint 用来恢复数据
- 特点：可持久地顺序写入，对顺序写入性能有要求
- 设备选择：高性能 SSD 或 HDD

3. 仓库文件（depot）

- 特点：高空间占用，对延迟要求不高
- 设备选择：大容量高带宽设备，最好能支持扩容，例如大容量 SSD、HDD 或 NFS

部署时建议把 DB 文件、事务日志文件和仓库文件分 3 块盘部署，以达到最佳性能。

Proxy 上的数据以缓存数据为主，对读写性能要求高，可以使用多块本地 SSD 盘组建 Raid0 阵列来使用。

四、多地访问优化

Commit 部署在用户最多的城市，CommitStandby 需要和 Commit 同城不同机房。

Proxy、Edge 需要在用户所在的城市就近部署

P4 仓库容灾

数据备份方式

热备份：

集群中的 replica 节点或 standby 节点

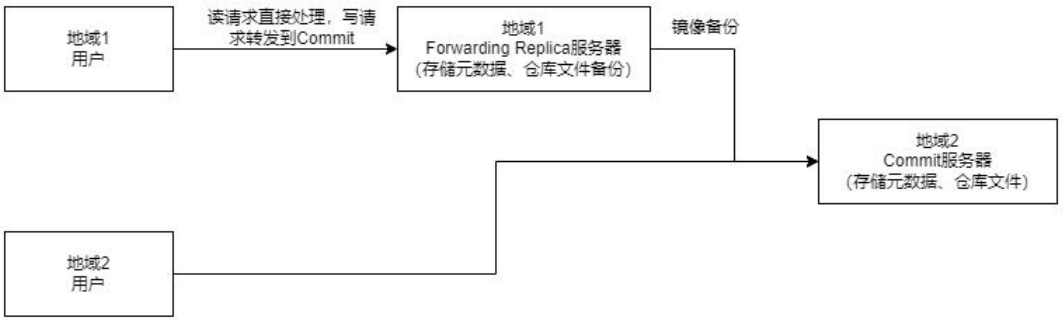
冷备份：

DB: 历史 checkpoint + 历史 journal

仓库：rsync 等文件拷贝机制

热备：镜像（replica）节点备份内容

- Replica 节点包含主节点的完整元数据 + 仓库数据镜像
- 可以帮助主节点分担处理读请求
- 灾备：随时可以切换为主节点
- 根据灾备机制完整性，分为 replica 和 standby
- 根据接受的请求类型，分为 forwarding 和 read-only
- 实际类型：forwarding-replica、read-only replica、forwarding-standby、(read-only) standby



热备：Standby 节点备份内容

standby 基本特性与对应的 replica 一致，具备对应 replica 的所有功能，standby 可以看作改进型的 replica。相同点：

- 热备节点
- 完整的数据同步备份
- 分担部分请求处理
- standby 提升了灾备场景的能力，不同点：
- 更完善的 journal 同步机制，避免数据丢失或集群状态不一致
- failover 机制，实现自动主备切换
- Mandatory 模式下同步确认机制降低集群性能

节点类型	读请求	写请求	故障切换	主节点类型	改进的同步机制
Forwarding Replica	本地处理	转发给主节点	手动	Commit	否
Read-Only Replica	本地处理	拒绝	手动	Commit、Edge	否
Forwarding Standby	本地处理	转发给主节点	自动 (failover)	Commit	是
(Read-Only) Standby	本地处理	拒绝	自动 (failover)	Commit、Edge	是

DB 数据冷备份

冷备对象：最近一个 checkpoint 和 checkpoint 以来的 journal

其他 checkpoint 和 journal 可以用来支持回滚等操作

方法：

- Journal 和 DB 放在不同的存储设备上
- 定期跑 checkpoint 和 journal rotation
- 历史 checkpoint 和 journal 进行可靠存储
- 可靠存储包括：

- 使用可靠的存储设备（Raid1 磁盘阵列，云存储，备份系统等方案）
- 多副本机制

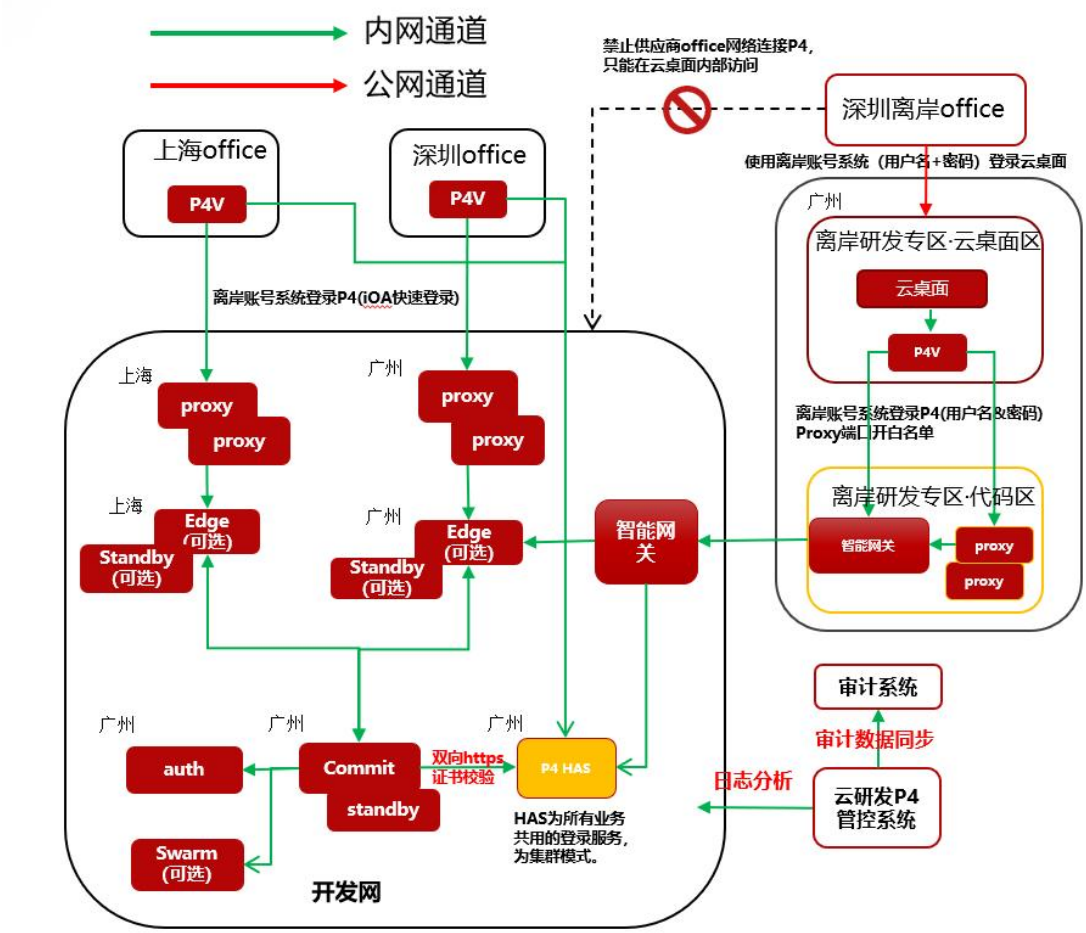
P4 仓库安全

我们制定了一些安全加固规范：

- 公网连接必须配置 SSL 加密
- 公网访问需要通过安全组对来源 IP 进行限制
- P4 密码安全级别必须设置成 3 以上：p4 configure set security=3
- 禁止自动创建用户：p4 configure set dm.user.noautocreate=2
- 强制新用户修改密码：p4 configure set dm.user.resetpassword=1
- 启用监控：p4 configure set monitor=25（这里至少要是 2，建议用 25 来开启锁 DB 进程监控）
- 开启审计日志：使用-A 参数开启审计日志，例如 p4d -A /data/perforce/auditlog
- P4 需要用非 root 账号部署，禁止监听高危端口
- 权限表里默认的所有用户写权限要去掉（write user * * //...）

P4 仓库总体架构

为了满足内部员工和外部供应商共同使用 P4 的需求，基于性能、容灾、安全三方面的优化方案，我们设计了如下 P4 集群架构。内部员工直接访问开发网的 P4 集群，供应商通过云桌面在离岸研发专区（专门为供应商设立的网络专区）访问 proxy，请求通过智能网关转发到开发网 P4 集群，以此来达到内部员工和外部供应商在安全、可靠、高效的情况下使用 P4 协同办公的能力。



SRE 可靠性提升措施：

1.使用统一账号系统，提供统一鉴权；

2.元数据（含权限表、用户表、trigger 等）自动备份，一键恢复；

3.集群化部署，主备容灾，使用云盘避免机器故障导致数据丢失；

4.代码拉取、提交、删除、变更记录接入公司审计系统；

5.人员离职、组织架构变更，邮件提醒权限删除；

6.禁止供应商 office 网络连接 P4，只能在云桌面内部访问；

7.p4auth 服务器单独架设，commit 异常，切换 replica；

P4 版本要求：

P4 HAS 要求 p4 版本>2019.2，若 P4 版本无法升级到 2019.2 以上可以不用 HAS 而是用公司内部 LDAP 鉴权。

P4 仓库扩展性

1. trigger

P4 的 trigger 机制可以在用户执行特定命令时触发对应的程序，从而扩展仓库功能。以下列举一个 P4 登录鉴权的 trigger 案例。

Triggers:

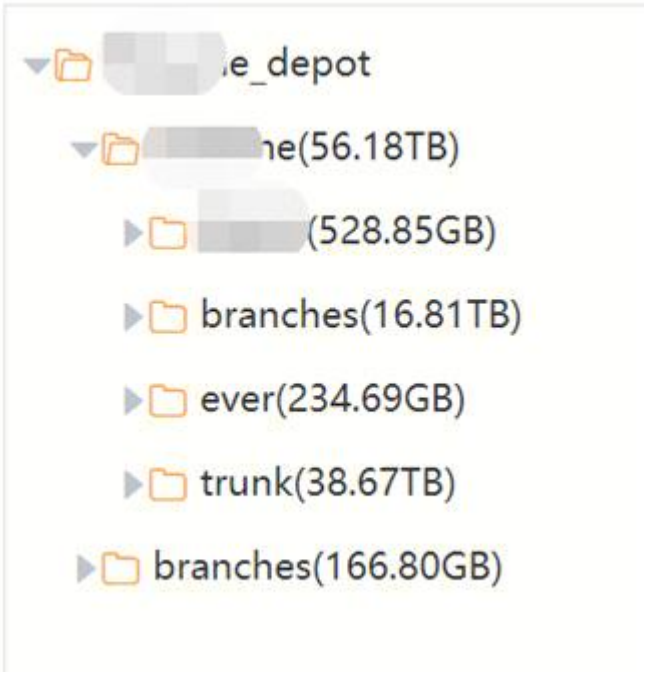
ldapAuth auth-check auth "/data/script/p4auth_ad-no_null ldap.domainname.com 389 %user%"

这个 trigger 的类型为 auth_check，会在用户登录时触发，调用/data/script/p4auth_ad-no_null 这个程序，把用户登录用户

名和密码传给 ldap.domainname.com 这个 ldap 服务器的 389 端口，校验通过用户即可成功登录 P4 服务器。

2. 容量管理

由于 P4 上经常会存储很多大的二进制文件，硬盘空间容易满。我们建设了容量管理系统，利用 P4 SDK 分析出每个目录、每个文件的所有历史版本占用的存储空间总大小。



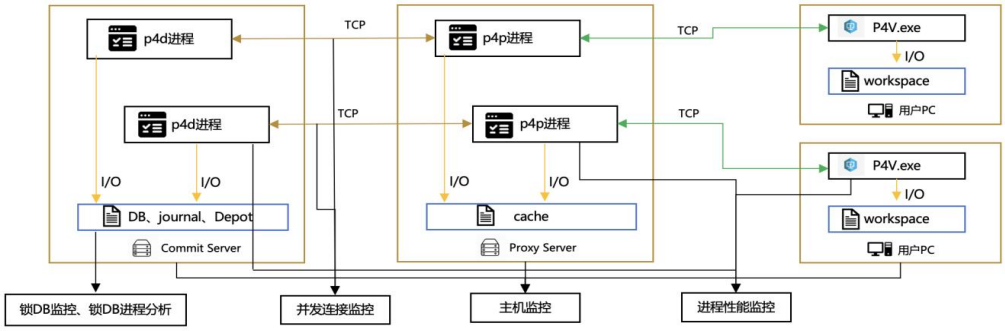
容量管理系统可以基于 P4 的 filetype +S 属性，配置每个文件在服务器上保存的历史版本个数，自动清理超出设置范围的历史版本文件。

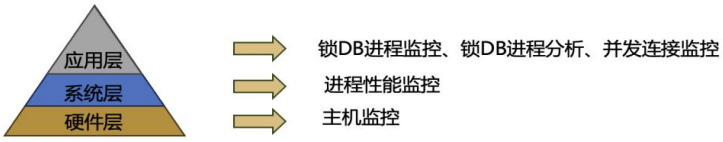


P4 仓库可观测性

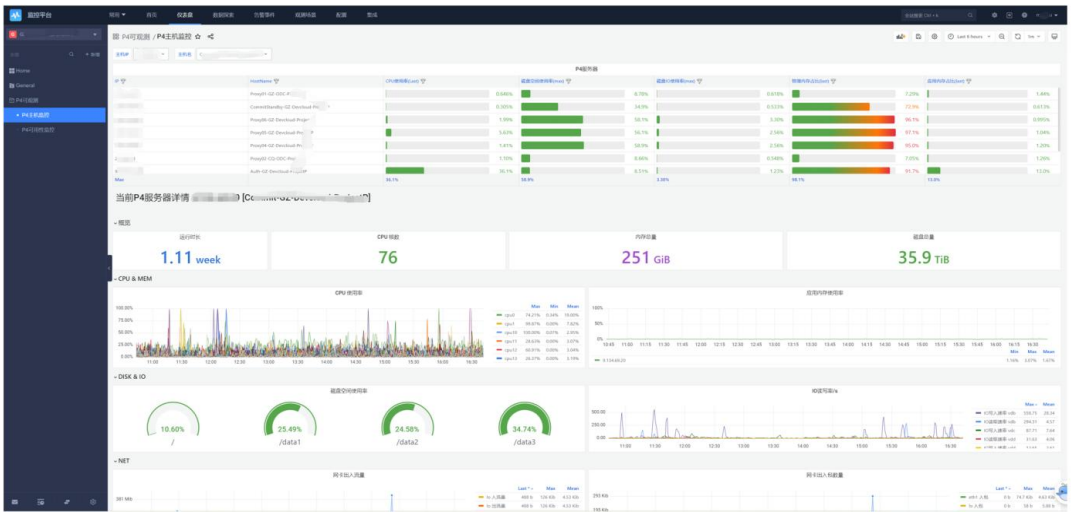
1. 监控建设

基于用户访问 P4 的进程网络、存储调用链，我们建设了主机监控、进程性能监控、并发连接监控、锁 DB 监控、锁 DB 进程分析等监控能力。





主机监控：监控 P4 集群各主机 CPU、内存、磁盘、网络等使用情况



网址：SRE-Elite.com 微信：SRE 精英联盟

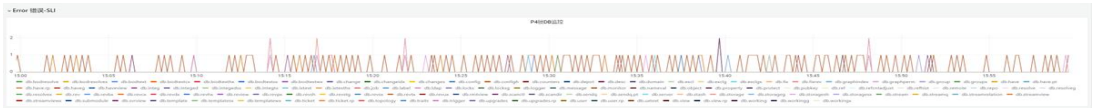
进程监控：监控 p4 连接进程（p4d、p4p）的 CPU、内存、IO 等使用情况



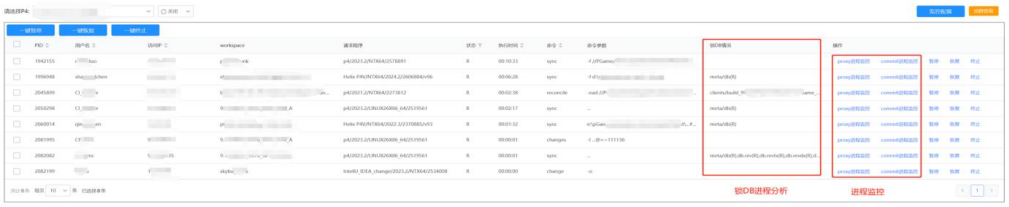
P4 并发连接数监控：监控 P4 的并发连接数情况



锁 DB 监控：监控各个 P4 DB 表的锁 DB 情况（不锁为 0，读锁为 1、写锁为 2）



锁 DB 进程分析：分析出锁 DB 进程以及该进程具体锁了哪些 P4 DB 表，并且能一键批量暂停、恢复、终止异常 P4 连接。



2. 可用性 SLO

首先设定 SLI

指标	目标
P4 并发连接数	<前四周的周峰均值*150%
P4 锁 DB 持续时间	<5 分钟

前四周的周峰均值定义：（前四周每周的并发连接数峰值的和）÷ 4

配置 SLI 监控项：



● 服务不可用：指标的值未达成目标，则该类别提供的服务不可用

● 服务可用性 =（1 - 不可用的总时长 / 总服务时长）* 100%

根据监控告警时长计算出服务不可用总时长，由此计算出服务可用性，仪表盘如下



（3）使用 Prebuild 提升构建效率的实践

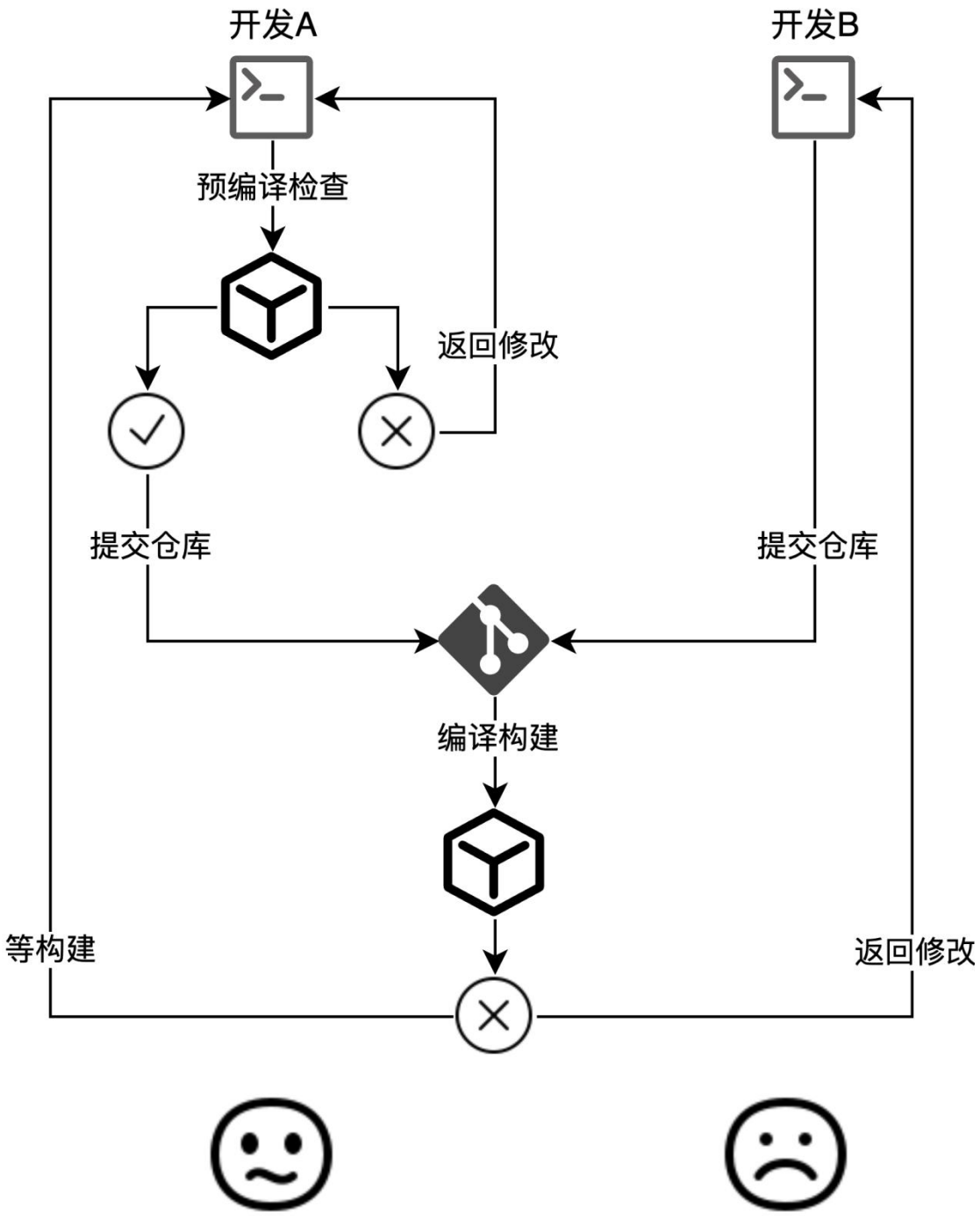
这是一个典型的通过预编译解决代码合并容易出错，提升构建成功率和效率的案例，适用于超过 10 人以上且高频做代码提交合并的开发团队。

该案例描述了一个超过 50 人的大型开发团队，对于如何提升编译速度做了详细的方案拆解，在本地构建资源有限的情况下，通

过利用云计算资源方案，还能大幅提升构建效率的问题，体现出互联网特点。

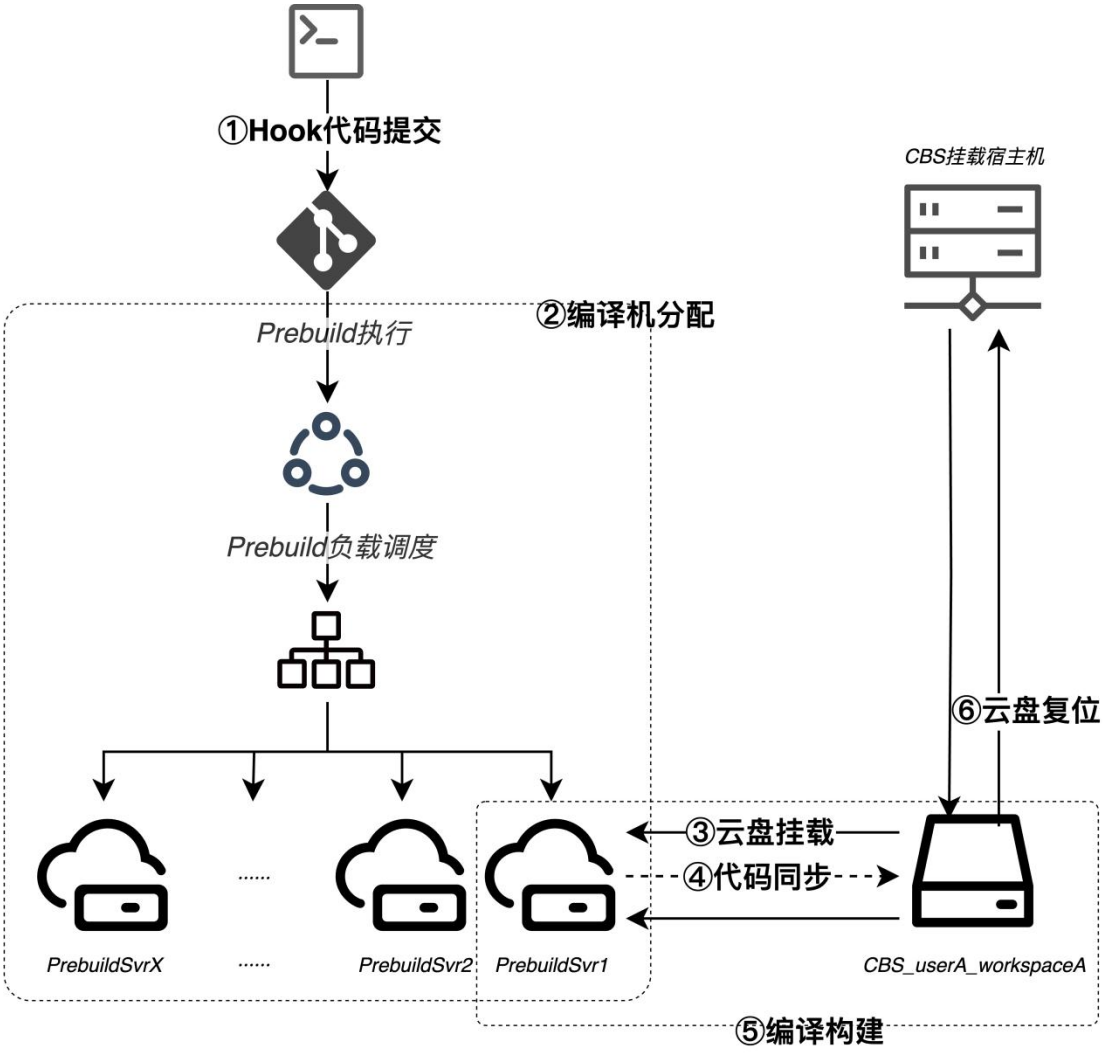
背景及设计原则

某项目的开发团队人数规模超过 50+，多个开发在合版时，出现合并失败的情况比较多见，特别是在人工执行检查时容易遗漏，并且引起本地编译卡顿。结果影响整体的版本出包和迭代效率，造成整个项目经常熬夜等构建完成，造成很大非必要的加班，拖累整个团队出包交付效率。



图：原始构建出包方案影响不同开发间协作构建出包效率

解决方案：通过使用 Prebuild 方案，把本机需要编译的资源同步到云上，利用云的强大计算资源能力，快速构建后，将编译好的文件同步到本地盘，提高编译速度。



图：Prebuild 实现流程图

设计思路和关键流程

1. Hook 代码提交：

代码提交：开发人员提交代码到代码仓库时，触发 Hook 机制。

2. 编译机分配：

（1）负载调度：预编译服务器（prebuildSvr）根据负载情况分配编译任务。

（2）预编译服务器：包括 prebuildSvr1、prebuildSvr2、prebuildSvr3 等。

3. 云盘挂载：

（1）存储初始化：根据初始化生成的存储映射表，将 CBS 个人工作空间目录 CBS_userA_workspaceA 临时移动到 prebuildSvr1 上。如果无映射，则临时分配并将映射存入数据库。

4. 代码同步：

（1）同步代码：userA 用户本地的 workspaceA 代码差量同步到 cbs 个人目录 CBS_userA_workspaceA。

5. 编译构建：

（1）蓝盾流水线：在 preSvr1 启动蓝盾流水线进行编译构建（preSvr1 也可以接入 tbs 算力）。

6. 通过办公机器人通知&云盘复位：

（1）通知用户：编译完成后，通过办公机器人将编译结果通知用户。

（2）云盘复位：然后将 CBS 目录 CBS_userA_workspaceA 从 prebuildSvrA 摘除并回 cbs 的宿主机 A 进行挂载。

网址：SRE-Elite.com 微信：SRE 精英联盟

主要亮点

1.存储计算分离：成本节约，无需一人一机；数百 G workspace 秒级同步，避免原模式分配新编译机的数百 G 文件同步

2.Hook 机制避免泄露：支持 Git/P4/SVN 仓库；支持按文件类型触发（例如 lua 无需触发）

3.不影响开发体验：prebuildSvrA 不阻塞提交；不消耗本机算力；构建异常会通过办公机器人通知失败详情

实践效果

通过实施 preBuild 方案，可以显著提升代码提交和构建的可靠性，减少构建失败的情况，提高整体开发和构建效率，并通过自动化机制优化开发流程。主要收益有：

1.保障编译正确性：保障所有开发人员提交代码仓库的代码都经过了本机编译正确性检查

2.构建成功率提升：代码合流后，构建成功率提升 30%以上

（4）出包加速构建可靠性研发保障实践

中大型游戏的研发周期一般来说都是比较漫长的，在和项目组沟通调研的过程中，发现在构建这一环节中，游戏包体的构建出包漫长导致很多的研发/测试流程会有效率上的阻塞情况。“构建出包慢”这种情况，几乎每个项目都会或多或少遇到这样的问题，尤其是在使用比较广泛的 UE 引擎构建出包过程中。针对以上的问题，

腾讯推出了 InGame 出包加速服务，提供 UE 引擎下 Cook 阶段加速服务

在同项目组深度调研沟通过程中，我们深挖了基于 UE 引擎开发的游戏项目，从构建出包再到真机安装验证整体流程中的各个环节，即 Build－Cook－Stage－Package&Deploy 与 Install/Test（注：Install/Test 不属于 UE 引擎出包环节)。

Build：代码编译（耗时占比 20%）

Cook：资源烘焙（耗时占比 70% 注：中型项目 5+Hr/ 特大型项目 20+Hr）

Stage：拷贝缓存（耗时占比 5%）

Package & Deploy：打包 & 发布（耗时占比 5%）



在 UE（Unreal Engine－虚幻引擎）项目中，“Cook”（资源烘焙）是一个至关重要的环节。它涉及将项目中的各类资源（如纹理、模型、音频等）经过精心处理和优化，转换成目标平台（如 PC、游戏机、移动设备等）能够顺利识别和使用的格式。这个过程可以比作烹饪食物，将原材料经过一系列加工处理，最终变成可以直接食用的美食。通过“Cook”，游戏资源得以优化，减少了内存

占用，同时确保了游戏能够在不同的平台上顺畅运行，从而扩大了游戏的受众范围。

然而 Cook 也是游戏构建出包全流程中最耗时的环节，且业内目前还没有比较好的解决方案，UE 官方直到 5.3 版本才有一个单机多进程的 Cook 方案；同时，其他阶段特别是代码编译 Build 阶段业界已有比较好的解决方案（如，商业化方案 Incredibuild，开源方案 FastBuild 等），Stage/Package 阶段总体耗时少，本身也只是简单的文件归档迁移等操作，优化的空间和价值并不大。因此，Cook 阶段的加速成为了游戏研发出包效率提升过程中的关键突破点。

Cook 阶段出包加速方案设计思路

为了解决 Cook 阶段出包加速的诉求，我们基于 Cook 阶段耗时进行了具体分析；

Init：

初始化阶段，主要是设置 Cook 过程的基本参数和配置，准备开始处理资源

Load/Save：

加载阶段，Cook 会加载项目中的资源文件，如纹理、模型、音频等。

保存阶段，处理后的资源会被保存为可被目标平台使用的格式。

Mesh/Anim：

Mesh 阶段，Cook 会对模型资源进行处理，包括顶点数据的优化、网格的简化等。

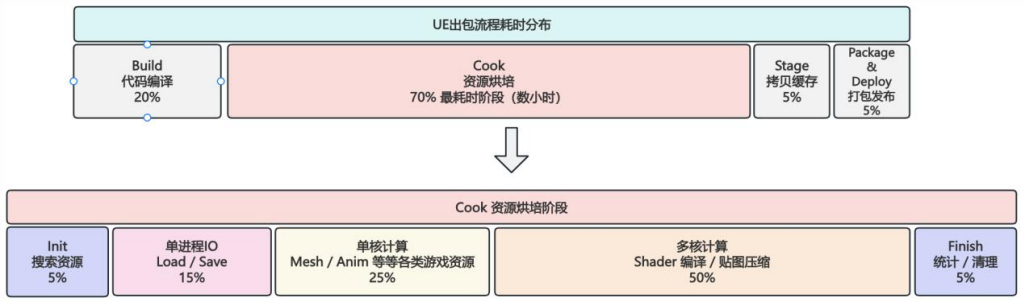
Anim 阶段，对动画资源进行处理，包括动画数据的压缩、优化等。

Shader：

Shader 阶段，Cook 会对着色器资源进行处理，包括着色器的编译、优化等。

Finish：

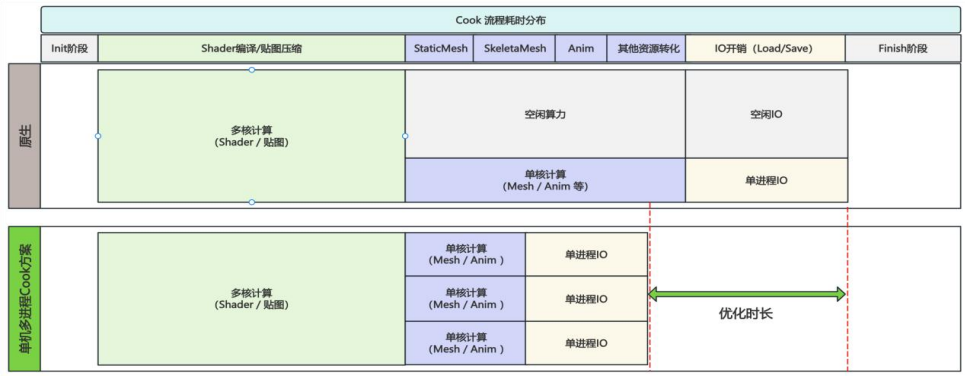
完成阶段，Cook 会检查所有资源是否已经成功处理，并生成最终的构建产物。



从上图可以看到，除了启动的初始化和结束的统计清理等工作，Cook 阶段最核心的工作是在单个 UE 进程下的 IO 和资源计算，包括单核计算和多核计算在内的一系列的资源计算。针对不同的场景，我们进行了多个优化方案的落地。

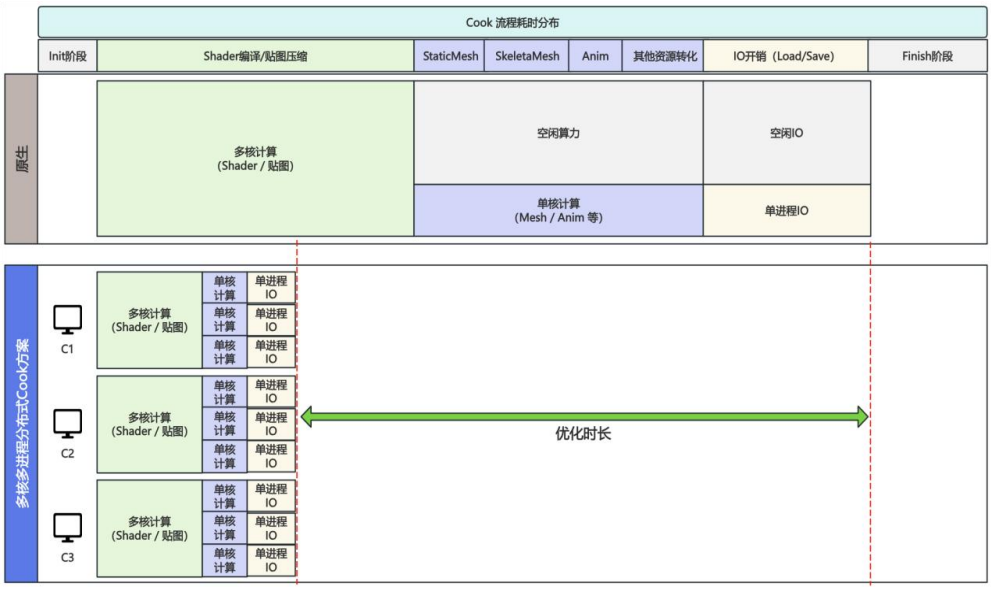
优化点 1：单机多进程 Cook 方案

针对单进程 IO/单核计算场景，通过并行处理充分利用了单机冗余算力资源，同时实现了 IO 瓶颈的突破；充分利用单机硬件资源进行计算提速；



优化点 2：多机多进程分布式 Cook 方案

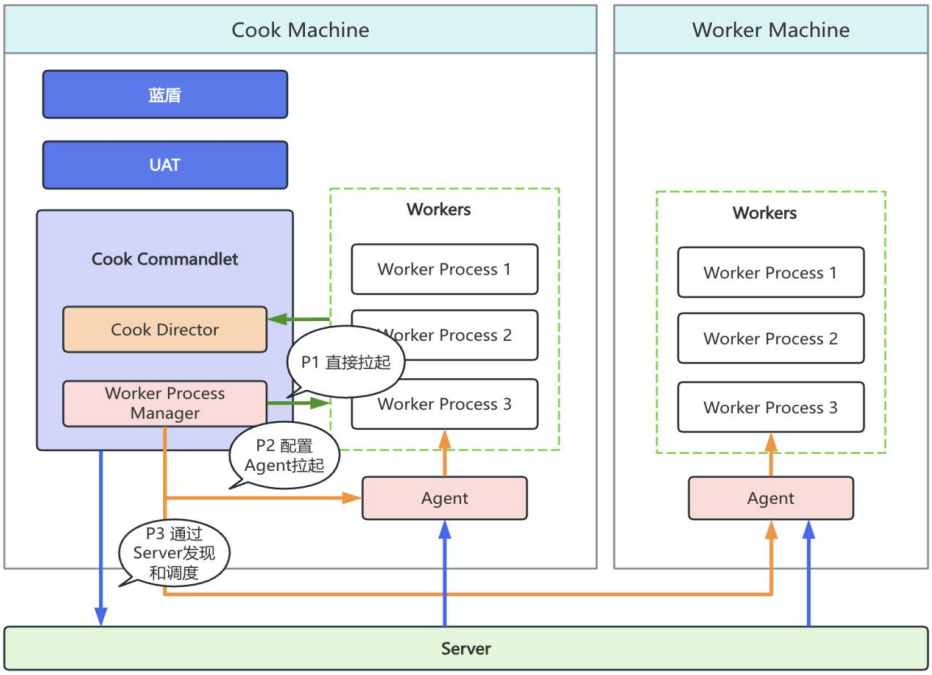
针对多核计算场景，通过物理机算力的分摊可以多倍扩展算力效能；实现了算力的极大拓展，提升 cook 阶段加速上限。



网址：SRE-Elite.com 微信：SRE 精英联盟

优化点 3：Agent 智能调度分布式 Cook 方案

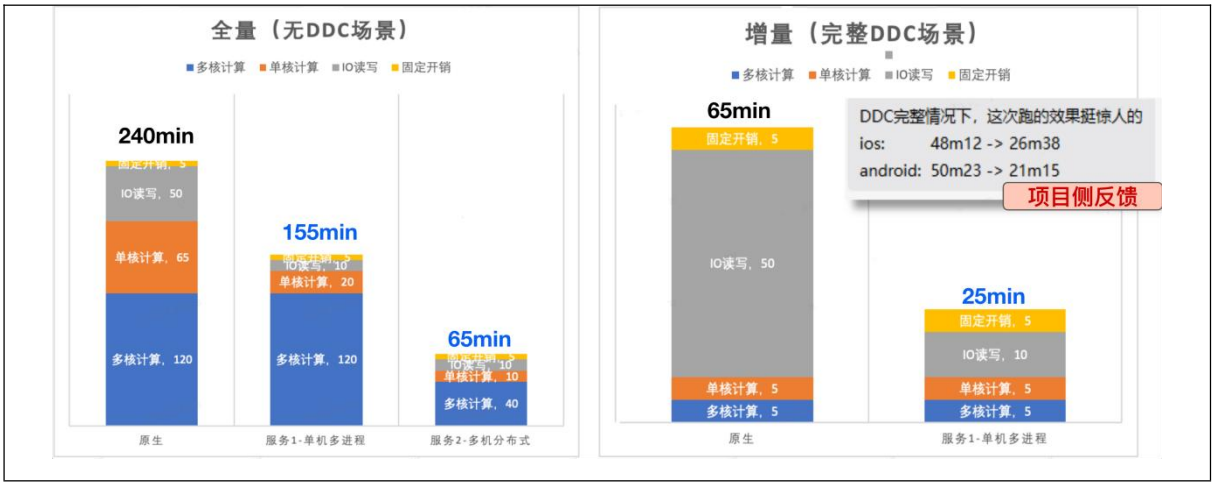
在整个 Cook 阶段，通过对 Agent 的灵活调度以及一系列算法的优化，可以对构建机的空闲算力进行合理调度，将 Cook 阶段耗时进一步压缩。其核心是通过对于 Agent 的灵活调度，对于多层级的进程进行管理，同时实现物理机负载均衡；



实践效果

当以上优化点都落实到项目侧（大中型项目，Cooksize 100GB+， Packages 15w+），在全量与增量加速场景下平均提效加速

3-4 倍，使得总体出包的速度大大地降低，为腾讯游戏研发效能提升做出了极大贡献。



(5) 制品晋级研发可靠性保障实践

为了应对行业的快速变化，企业的版本更新频率不断加快，制品的版本质量对生产环境的可靠性至关重要。通过建立一套制品晋级流程，可以有效地把控外发版本的质量。

一般情况下，制品的等级通常分为 Alpha、Beta 和 Release 三个级别，每个版本通过相应的测试后，才能晋级到下一个级别。

以“主干开发、分支发布”的研发模式为例，外发版本的特性和 Bug 修复在主干开发阶段经历 Alpha（开发自测通过）和 Beta（测试人员测试通过）两个阶段。当代码合入到外发分支时，将进行功能测试和制品检查，以确保制品达到外发版本的 Release 等级要求。

制品晋级主要包含构建检查、制品一致性校验、功能测试和关键指标巡检等 4 个考察维度。

1. 构建检查

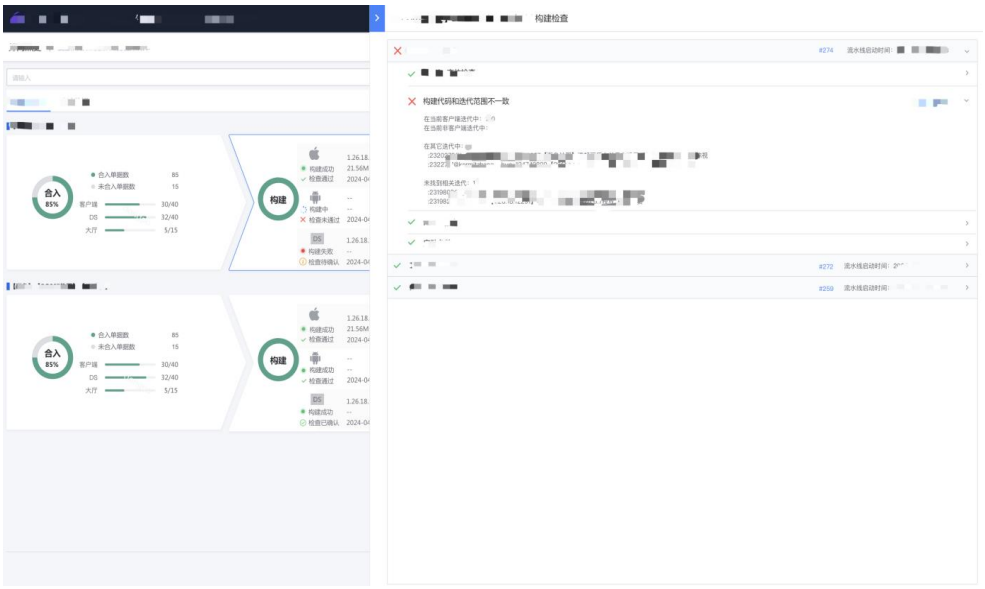
在构建过程中需检查构建原材料（即代码）是否符合预期。根据检查项类型，可以分为“致命检查项”和“二次确认检查项”，构建通过的标准是所有检查项全部通过或经过问题澄清。

(1) 致命检查项

当致命检查项不通过，必须终止构建流程，并排查问题直到解决。

以下是某业务致命检查项不通过案例：

在构建过程中发现构建原材料（即代码分支）和研发管理工具中规划的迭代范围不匹配，构建原材料中存在未关联迭代需求或 Bug 的代码。



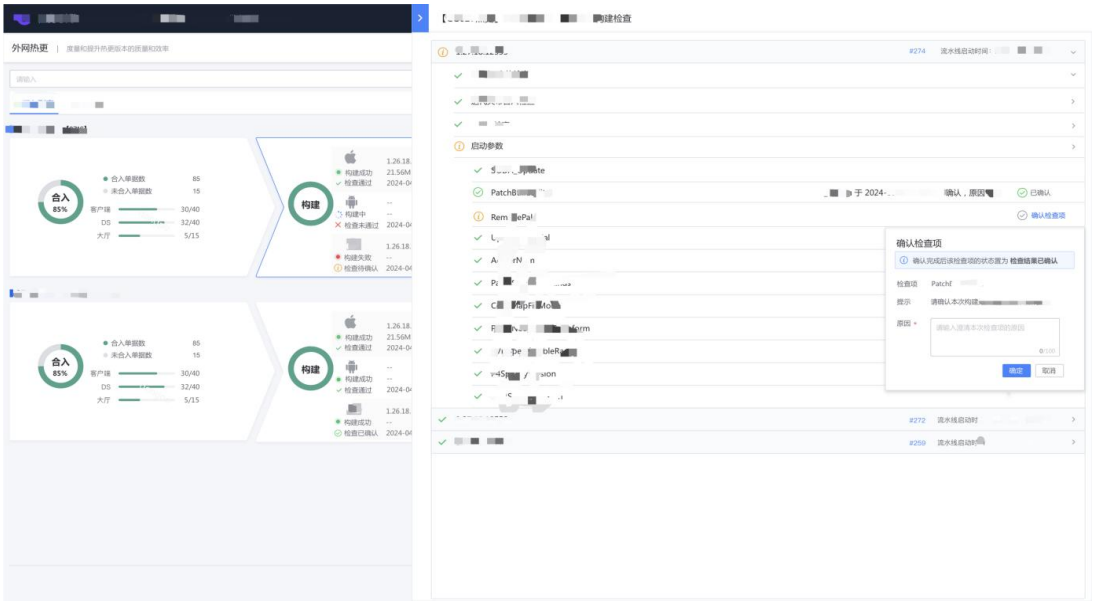
(2) 二次确认检查项

二次确认检查项包含非致命检查项和非明确的构建检查逻辑检查。

对于非致命检查项，构建检查人员根据实际情况，可选择继续维持“不通过”状态，或置为“已确认”。对于非明确的检查项，可以选择将其置为“已确认”或重新构建。

以下是某业务的实践案例：

在构建过程中存在构建参数与默认值不一致的场景，构建检查人员根据实际情况澄清，或当不符合预期时重新构建。



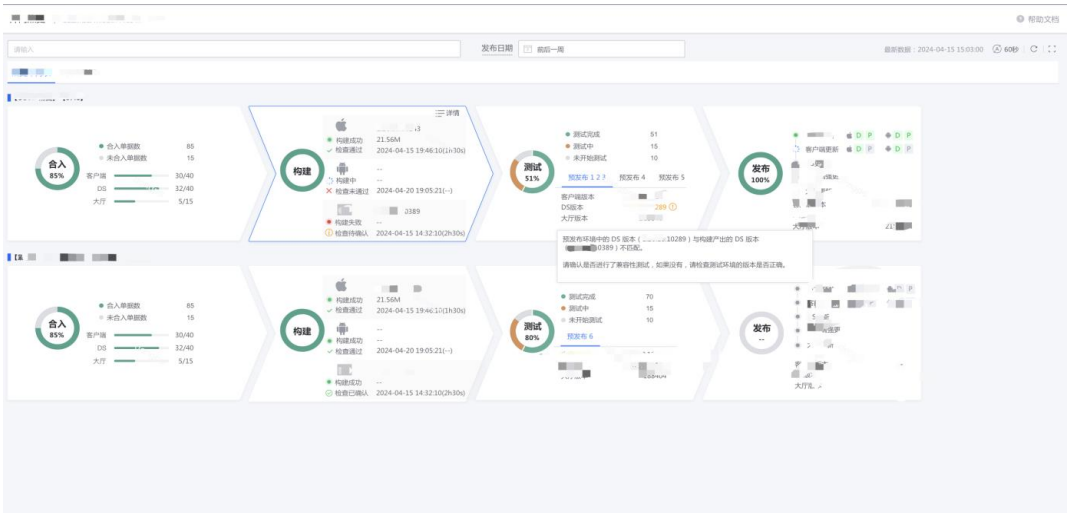
2. 制品一致性校验

当版本更新频繁时，可能会出现版本未正确部署到测试环境，而测试人员反馈测试通过的情况。这种情况下，测试人员测试的版本和构建制品不一致，导致测试未通过的版本被发布到生产环境，引发用户投诉。

因此，需要校验构建制品和测试环境中运行版本的一致性，以确保测试人员测试的版本与构建制品一致。

以下某 Top 业务的实践案例：

当测试环境的版本与构建制品版本不一致时，系统给出提示，指引相关人员检查并跟进。



3. 功能测试

功能测试主要包括 API 自动化测试、UI 测试等，相关内容在 3.3.2 章节已做详细介绍，此处不再赘述。

4. 关键指标巡检

构建的制品正确部署到测试环境后，需要经过功能测试。然而，功能性测试在特定时间段内完成，测试通过不意味着制品达到了外发标准。因此，需建立一套外发制品关键指标巡检流程，涵盖版本质量、服务器性能、关键路径用户体验等关键指标，以确保制品达到外发的标准。

1) 版本质量

检查客户端和服务端的崩溃数据，如程序崩溃次数、程序 Coredump 次数等。

2) 服务器性能

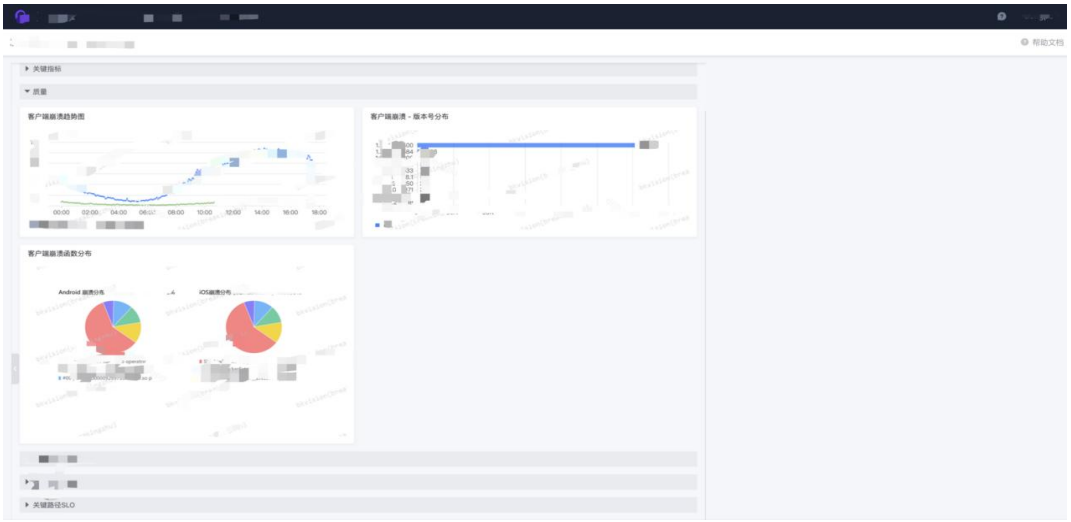
对比当前版本相对于上一个版本在单机性能维度的表现，评估单机承载能力是否下降。

3) 业务关键路径场景指标

观察业务关键路径指标是否符合预期，如注册、更新、登录、匹配、对局和支付流程的关键指标。

以下某业务的实践案例：

计划外发的制品通过了功能测试，但在版本质量维度发现客户端偶现崩溃、服务端 Coredump 情况，找到问题原因后，重新构建制品并走制品晋级流程。



当制品在分支发布的环节通过了构建检查、制品一致性校验、功能测试、关键指标巡检后，制品等级从 Beta 晋级为符合发布标准的 Release 级别，最终 SRE 将制品可靠地发布到生产环境。

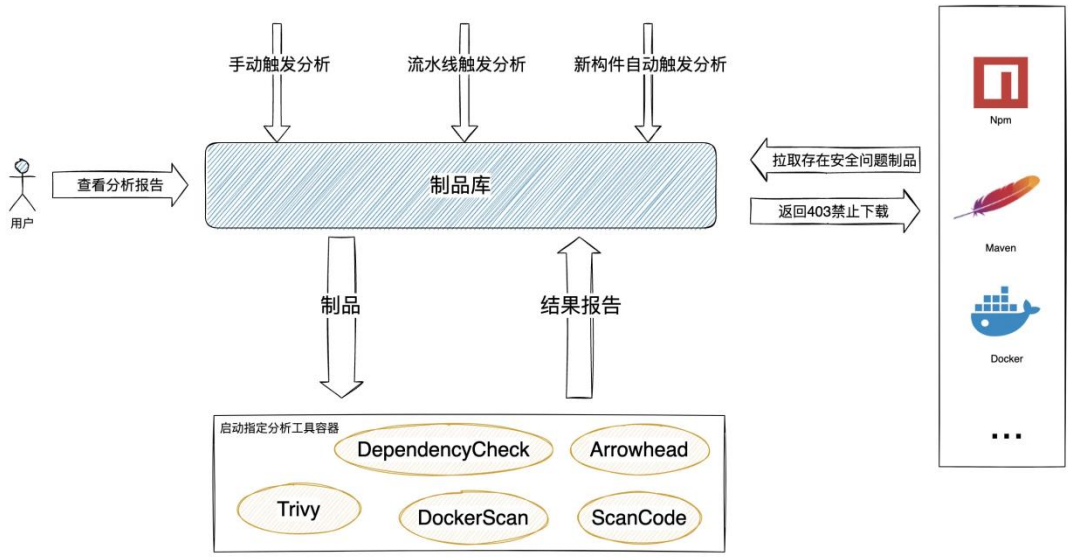
(6) 制品安全扫描和制品全球分发可靠性保障

制品如何安全、有效、快捷部署，是指通过技术手段，打通构建流水线，将构建完成的制品主动推送到用户多种类型终端的过程，用自动化的方式对制品进行安全扫描、分发、安装、更新，让用户以较低的时间成本获取到构建产物，减少临时下载制品对迭代效率的影响，尤其对于具备超大制品、全球发行的业务类型，研发保障制品库分发的可靠性尤为重要。

1. 制品库安全扫描研发保障可靠性实践

业务相关制品中往往包含了大量第三方组件，但是这些组件是否安全，使用的许可证是否允许商用等问题存在着很多不确定性。对海量制品进行分析，帮助用户发现制品存在的安全漏洞、敏感信息、第三方依赖许可证等信息

具体实现架构上，制品库管理员在将分析工具注册到制品分析平台后，用户即可指定要分析的制品创建分析任务，分析结束后会通过聊天机器人通知用户分析报告的地址。从分析任务的创建，到加载文件执行分析，再到分析结束后保存报告，整个流程都是在制品库集群内部完成，很大程度上避免了制品传输到外部带来的带宽压力与数据安全风险。



制品分析平台的设计与实现

分析任务的来源有三种，手动触发、流水线触发、新构件上传后自动触发。触发后就会根据任务指定的分析工具与搜索规则，开始搜索需要分析的制品，为每个制品创建子任务，提交到任务队列中等待执行。

同时针对制品安全扫码研发保障上，平台会根据现网的制品扫描速度监控数据，在制品库管理后台为不同的分析工具分别设置 MaxTime/MB，然后根据这个值为制品计算出超时时间。在创建子任务时也会设置一个最小允许执行时间，避免一些小文件计算出的超时时间过短导致分析工具启动后还没初始化好任务就超时的问题。

以下是某 Top 业务在研发制品保障的具体实践：

制品分析管理服务通过 Micrometer 将当前的任务执行情况数据导出，并对接到蓝鲸监控平台。业务侧可以通过监控可以查看当前的任务执行、扫描速率等信息，针对性地做一些优化工作。



1.业务快速通过流水线插件配置制品扫描

在“归档构件”插件中，可以配置使用制品库安全扫码的功能

网址：SRE-Elite.com 微信：SRE 精英联盟



a. 流水线添加归档构件或构建并归档 Docker 镜像插件，勾选启用制品扫描

b. 选择同步或异步扫描，同步扫描会阻塞流水线，可以结合质量红线使用，异步扫描会在制品上传完成后继续执行流水线

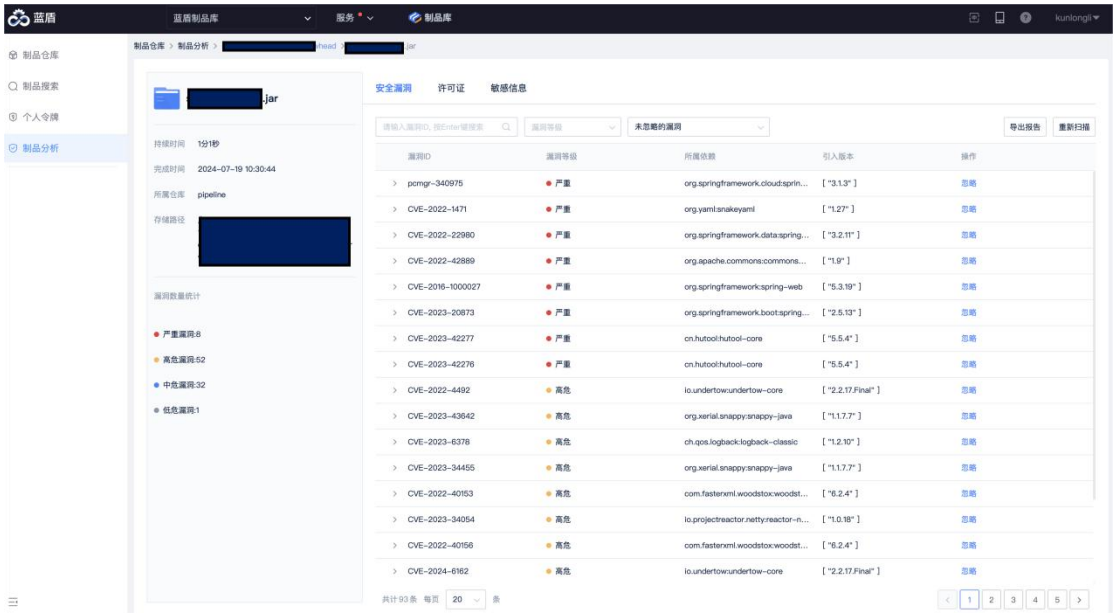
c. 扫描结束后可以通过聊天机器人通知构建触发人

2. 业务快速查看报告

插件里面可以开启扫描开关，开启后归档完成会触发扫描通过聊天机器人发送报告



通过具体链接页面可以查看报告，在报告页面，即可查看扫描方案已扫描的制品列表的安全情况，展示的风险等级为制品扫描出的漏洞最高风险等级。从而协助研发人员针对制品问题快速进行优化确认，以免在后续外发系统后，出现更大的安全事故。



3. 结合质量红线

同步制品库扫描可以结合质量红线来使用，对于安全扫描有问题的制品，可以通过质量红线来优化制品质量



具体步骤，可以在归档构建插件配置页点击立即配置。配置页面点击立即添加。选择单个指标，找到归档构件相关指标添加即可



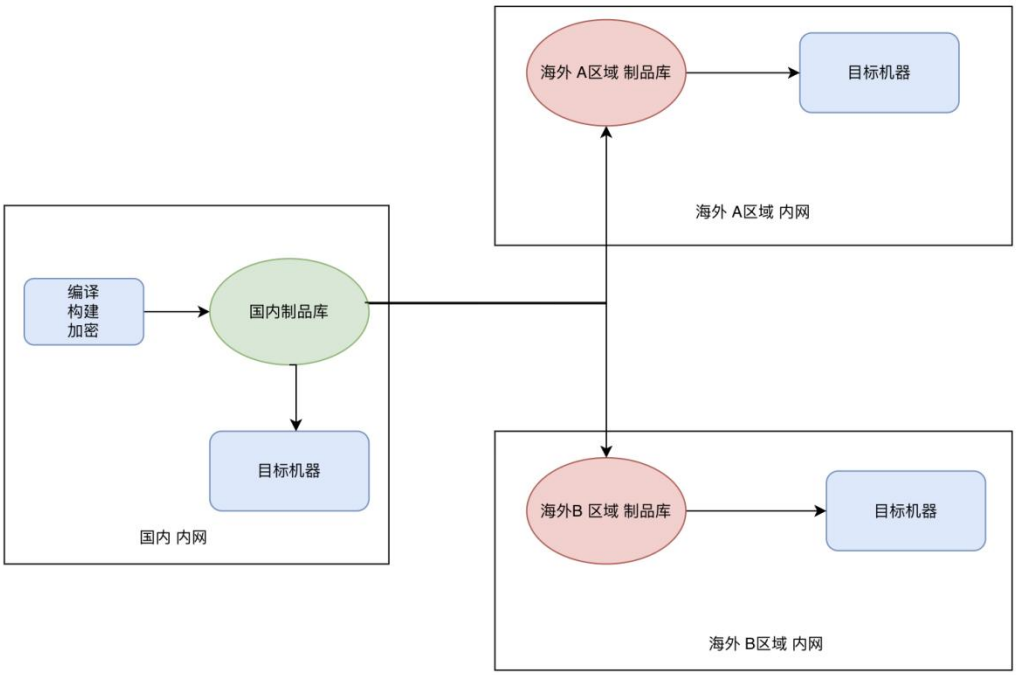
2. 制品部署研发保障可靠性实践

在产品出海的潮流下，业务全球统一发行已经是大势所趋，如何保障版本的一致性部署是在全球化统一发行过程中避不开的难点，以下是腾讯游戏全球发行中版本的一致性部署的解决方案：

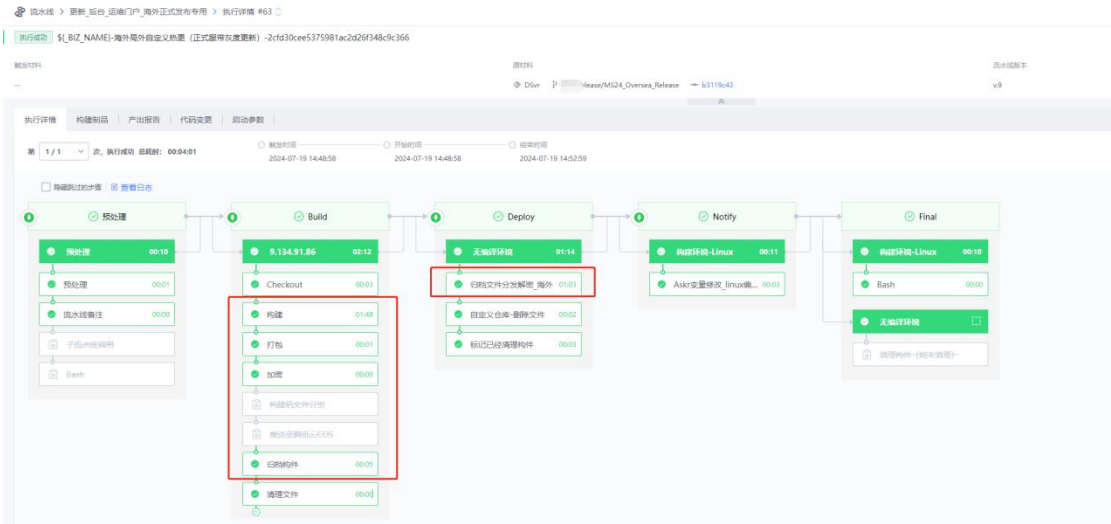
① 在国内编译机上从代码仓库(P4/Git/SVN)拉取最新的版本进行编译和构建，并对构建包进行加密后上传到国内的制品库；

② 海外制品库是国内制品库的镜像，在完整性方面有着强一致的校验；

③ 国内版本部署时从国内的制品库进行拉取；海外的版本部署从各个区域对应的制品库进行就近拉取，国内和海外共用新的构建包，保障了版本的一致性；

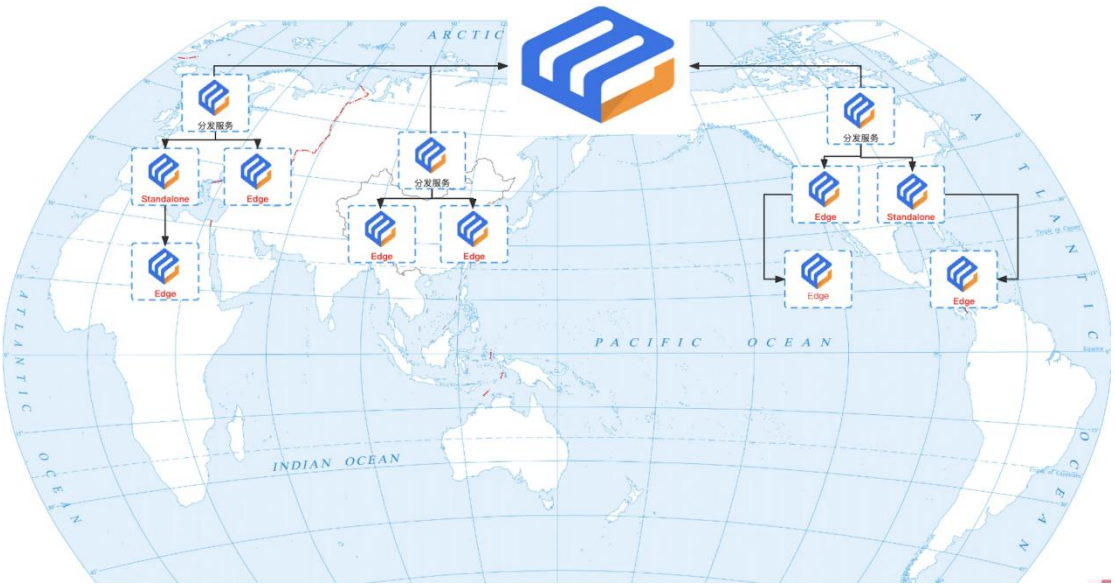


以下是某业务在研发制品分发保障中，平台流程的编排步骤如下：



制品库为了支持用户多地域、多节点使用制品库的场景，实现了制品分发功能，不仅支持多种部署模式以及集群间同步功能，而且还支持将制品分发到异构集群（第三方仓库）。

目前制品分发功能针对同构集群节点已实现 Generic、Maven、Docker、npm 等制品类型的分发；针对异构集群现已实现 Docker、Helm 类型制品的分发。制品分发服务已在多个环境中进行部署，主要使用场景包含将制品分发到海外不同部署环境中、将构建并上传到制品库的镜像分发到第三方的镜像仓库中。



通过 Micrometer 采集各种指标（请求数量、请求频率、线程池正在执行线程数、等待任务数量等），并对接到蓝鲸观测平台。通过查看配置的指标面板，可以查看当前任务执行速率，等待任务个数等；同时还可以查看整个服务的性能状况。

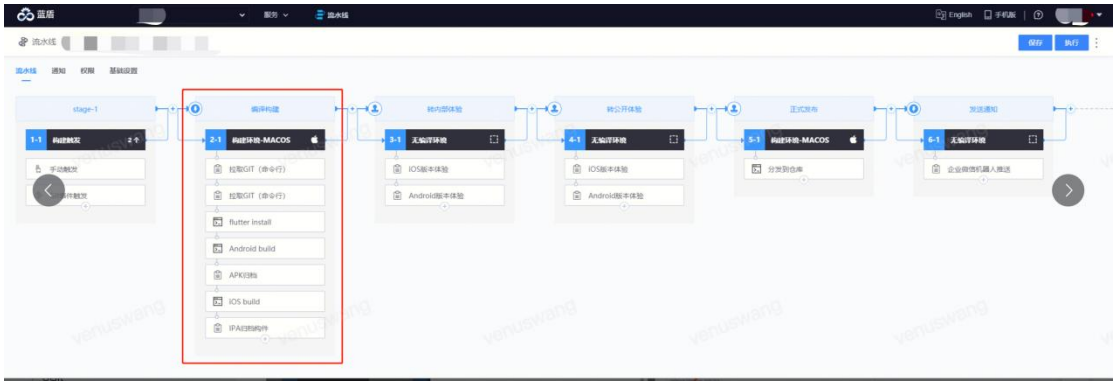


同时，根据研发保障报表，可以查看整个任务全流程中的各种数据，例如筛选出异常任务数据。每次任务执行都会有详细的记录，如任务起止时间、任务调度时间、任务对应制品实际开始传输起止时间、任务执行成功率、传输速率、失败任务数、失败任务详情、执行时长等数、任务失败原因等。依托于任务记录可对每次分发任务进行回溯或者当发生分发失败时可以快速定位问题并解决；同时依托于详情数据可以进行任务对账。



3.下载可靠性研发保障实践快速安装

业务可以将流水线构建出来的 IPA/APK 包发布到蓝盾手机应用，用户可以直接通过蓝盾 APP 直接快捷安装新的构建包，便于产品、测试团队验证产品新特性，发现产品潜在缺陷，让产品在内部验证流程上得到闭环，加快版本的迭代完善，并且配套着权限控制，让测试产品能够更安全的对内测试和体验。

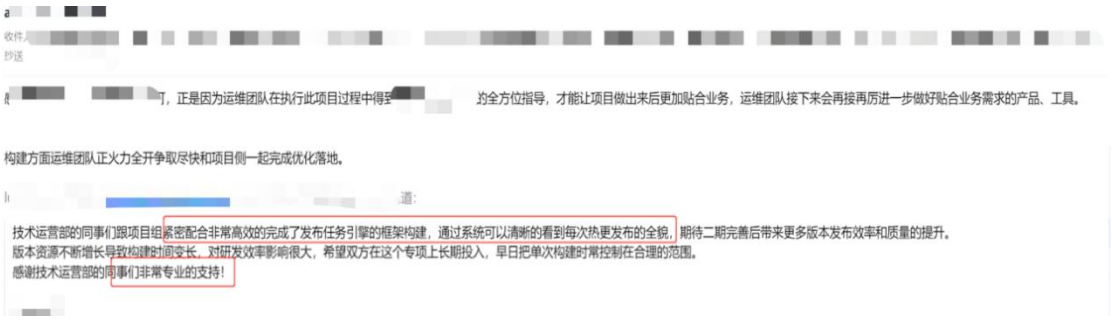


通过平台的制品库或者蓝盾 APP，业务就可以查看最近构建的新客户端包，产品和测试团队可以直接扫码安装，加快了产品的安装和验收进程，验收过程有问题，可以及时进行代码修复和重新编译构建，以此进行功能的快速迭代和 bug 的快速验收修复。



（三）总结及展望

在我们实践中，当 SRE 从代码可靠性、代码仓库可靠性、构建可靠性和制品可靠性的角度切入，解决了业务研发迫切的问题，专业度得到了业务研发的好评，以下选取部分的内部客户反馈。

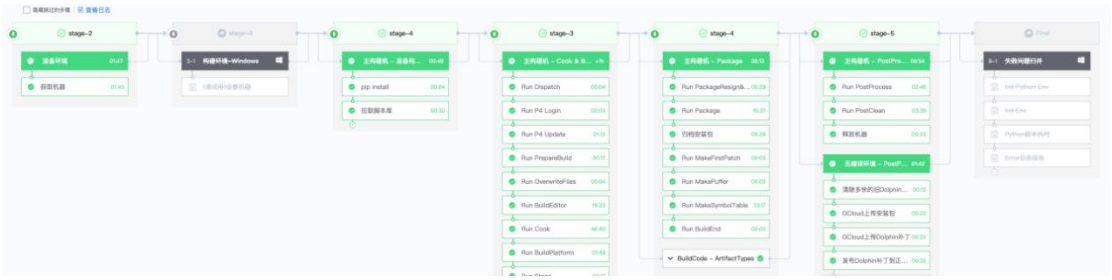


研发管线优化



编译加速，全量从 6 个小时降低到 3.5 小时左右

客观可量化数据上，有以下的整体业务收益：



步骤	优化前	优化措施	优化后
编译机部署	1、蓝盾 Agent日均掉线告警数 100+ 2、高强度构建 日均磁盘满告警数 30+ 3、170+构建机项目组自行维护，50多人都可操作，编译环境（开发、插件、Agent、命令、系统配置）千差万别 4、Win/Macos编译过程频繁死机	1、标准化管理 2、监控调优 3、系统参数调优	1、蓝盾 Agent日均掉线告警 0-1次（稳定性提升 90%+） 2、管控内机器日均磁盘满告警0次（完全解决） 3、构建机维护由运维全面接管，且全部实施标准化管理（完全解决） 4、异常实时监控，机器可用率99.9%以上
拉取代码（P4代码拉取）	1、P4代码拉取慢，不到100Mbps，问题定位慢，解决效率低	1、Proxy、Commit配置升级 2、启用云研发方案 3、开启并行拉取 4、员工硬件升级 5、实时告警	1、速度提升至500Mbps以上，云研发场景最高到4Gbps 2、异常响应15min以内，大部分问题5min解决
资源拉取	1、UE DDC访问延迟高，下载慢5MB/s 2、Untiy CacheServer下载非常慢2MB/s	1、本地部署、就近部署	1、UE延迟从 30~50ms，降低至 1ms，下载速度提升至25MB/s，提速5倍 2、并发下载已到 20MB/s，提速10倍
代码编译	1、编译构建慢，耗时超过预期	1、TBS多构建机并行能力、pump加速能力、云端加速能力 2、定制高配构建机	1、通过TBS多构建机并行能力、pump加速能力、云端加速能力，编译耗时降低48% 2、全量编译2h缩减至40min
客户端打包	1、签名耗时10min	1、本地签名	1、签名耗时降低至1min，签名效率提升90%
整体	1、核心流水线执行成功率不足 60%	ALL	核心流水线执行成功率提升至 90%+（提升 30%+）

● 业务收益-核心流水线执行成功率从不足 60%提升至 90%，成功率提升达 50%。

● 业务收益-通过代码可靠性保障，质量红线拦截比率 16%，代码缺陷数从 311 个下降为 0，代码安全漏洞数从 37 个下降为 0，包含了圈复杂度、代码规范等多维度的代码评分提升了 40%

● 业务收益-数据资产管理 P4，速度普遍提升 4 倍+，卡顿率降低 70%（几乎为 0），优化下载拉取链路，下载速度由 100Mbps 到 800Mbps

● 业务收益-通过优化代码仓库边缘加速，代码仓库拉取速度提升 100%，多人同时拉取代码仓库成功率为 99.99%；代码仓库每日卡顿次数从 10+次，下降到 0 次

● 业务收益-通过接入 Prebuild 方案，业务编译构建成功率提升：代码合流后，构建成功率提升 30%以上

● 业务收益-业务日构建综合提效 50% 以上。针对构建机软件流程优化，优化前，业务通过定制构建机，无法用镜像方式进行套件部署，N 台部署 10+构建套件需要 2*N 小时以上。SRE 介入后，通过批量自动化，安装 10+款软件，N 台部署 10+构建套件人工操作环节仅需 1 分钟

● 业务收益-通过蓝鲸持续集成平台中，BKTurbo 集群分布式编译，游戏业务构建出包效率提升 30%～70%；云渲染时间下降 98%

● 业务收益-通过制品可靠性保障，业务制品安全扫描接入覆盖度提升 100%，制品分发到全球，1GB 文件，保证在 5min 内完成分发，同时达到 99.9% 的分发成功率； 制品分享效率上，以头部业务为例，通过制品构建后快速应用安装，安装后占用存储空间约 10G 的游戏，节省单测试人力 30+分钟/次，对于 200 人的测试团队，单次即可节省 100+ 小时的人力消耗，从而保证业务每日出包每日即可完成快速测试的敏捷开发流程持续顺畅进行。

2.3.2 某语音直播公司研发过程保障实践

SRE Elite 精选原因

此案例展示了某语音直播公司在现代化软件架构下的研发保障实践。面对微服务、容器化和服务网格等新技术带来的挑战，该公司构建了全面的研发保障体系，涵盖快速发布、稳定性保障、代码可靠性和服务运行等多个关键模块。其中，采用服务网格进行环境隔离和金丝雀发布，属行业内的创新实践，体现了深入的云原生应用。此外，通过 IDE 插件对接环境进行调试，大幅提升了问题排查的效率。此案例实践性强、创新性高，具有广泛的借鉴意义。

（一）背景及设计原则

以人工智能、大数据、云计算、微服务、云原生等为代表的新型软件架构不断发展，这不仅重塑了整个软件产业的技术生态，还深刻地改变了软件技术的技术架构和商业应用，推动了软件设计、开发、生产、应用等各个环节的发展模式变革。在这种新形势下，软件的应用化发展呈现出数据化、平台化、智能化、云化等特点。

某语音直播公司在现代化的软件架构思想指导下，正是围绕微服务、容器化、服务网格 等现代化技术构建了内部的软件体系。现代化的软件架构思维给传统的研发过程保障带来了全方位的新挑战；同时直播场景在稳定性和软件架构方面与传统软件架构存在差异。

在现代化软件架构下研发具有以下新特点和挑战：

1.基于敏捷开发思想的快速发布与构建

在敏捷开发思想的引导下，团队致力于快速迭代和交付。每天的构建和发布次数达到百次级别。

在这种情况下，如何确保构建的快速和顺畅，以及发布过程的平稳和稳定，成为一个巨大的挑战。

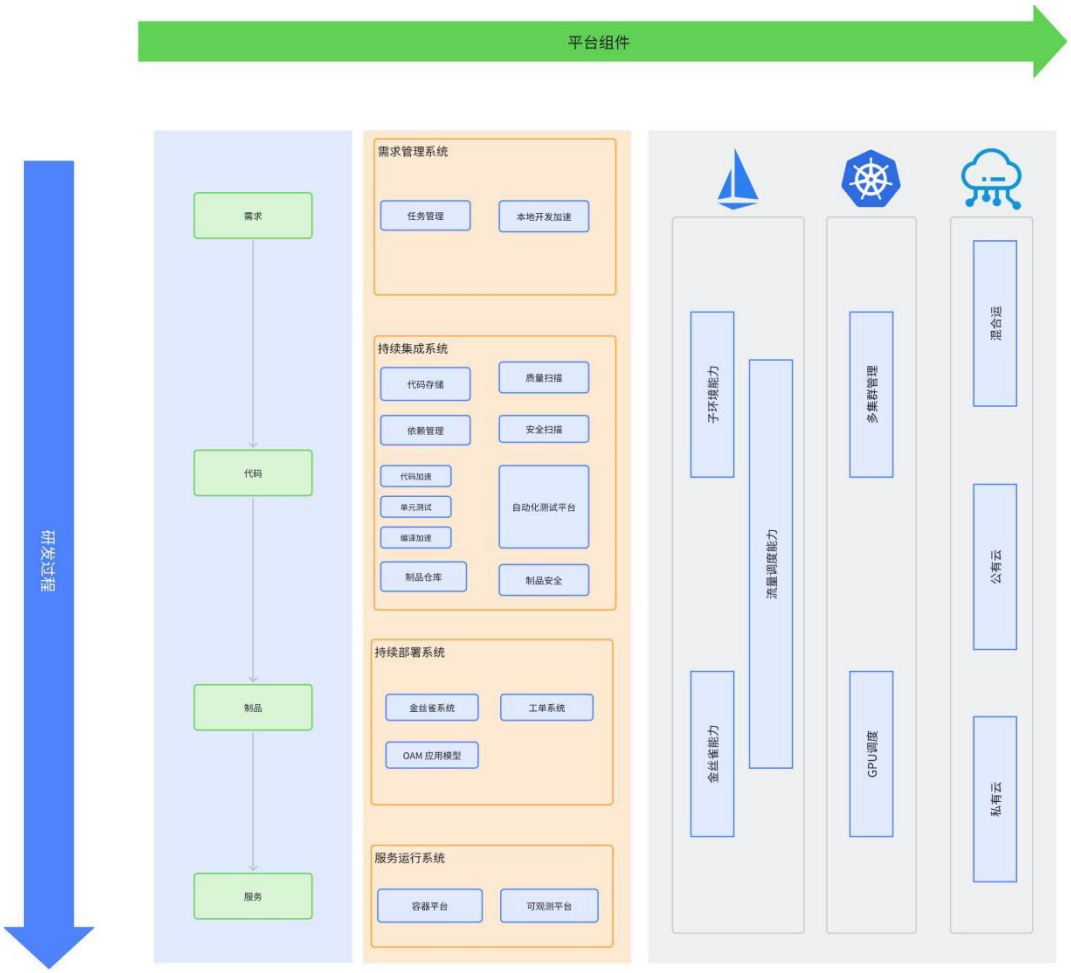
2.极高的稳定性要求

在如此快速的迭代情况下，如何保障业务发布的稳定性，尤其是直播场景对稳定性的要求更高。在出现问题时如何快速回滚恢复，以及如何避免影响大量用户成为保障稳定性的关键一环。

网址：SRE-Elite.com 微信：SRE 精英联盟

（二）体系设计及关键流程

体系设计



为了应对以上挑战，我们构建了四个相关系统来应对。

需求管理系统

为实践 BizDevOps 特将业务团队纳入研发过程，通过需求/任务管理能够及时反馈业务诉求的进度和流向。包含

①任务管理服务，针对需求的状态、自动流转，时效统计等；

②本地开发，能够实现研发本地 IDE 编写代码，自动关联需求，并能一键部署在云端集群本地 Debug。

持续集成系统

为使研发养成良好的代码持续集成习惯，通过流水线引擎自动将代码存储管理、单元测试、自动化测试、质量扫描、安全扫描等手段与工具结合起来。通过质量阈的有效代码提交，将进行自动化编译，最终生成制品。

持续部署系统

针对服务部署所关联的配置和交付过程进行统筹管理。包含 OAM 模型管理、工单流程、金丝雀发布等子系统。

服务运行系统

主要包括

①容器平台，针对服务运行时资源、路由资源、流量控制资源进行管理；

②可观测平台，对部署和应用在集群里的资源进行运行时数据采集、告警等稳定性保障。

基于 OverlayFS 的拉取源码加速实践

背景：流水线中的任务都需要基于代码执行，公司内部也有很多大仓库（数据量 3-6G），同时流水线本身也是容器化运行的，如

网址：SRE-Elite.com 微信：SRE 精英联盟

果每次都全量拉取代码，消耗时间就会很久很久。所以需要一套机制来解决基于云原生环境的仓库代码缓存方案。

需求分析：

- 基于大仓库的场景，需要做冷启动（预先 clone），同时需要适应代码提交做增量更新。
- 基于流水线的场景，我们需要提供针对 repo、branch、commit 提供代码，同时考虑到流水线的制品晋级和失败重试的情况，需要最大程度复用代码的存储。
- 基于容器化的场景，流水线任务都是在 Pod 里面运行，需要代码以文件系统的方式在 Pod 里面可以访问。

基于 OverlayFS+NFS 的 CSI 方案设计

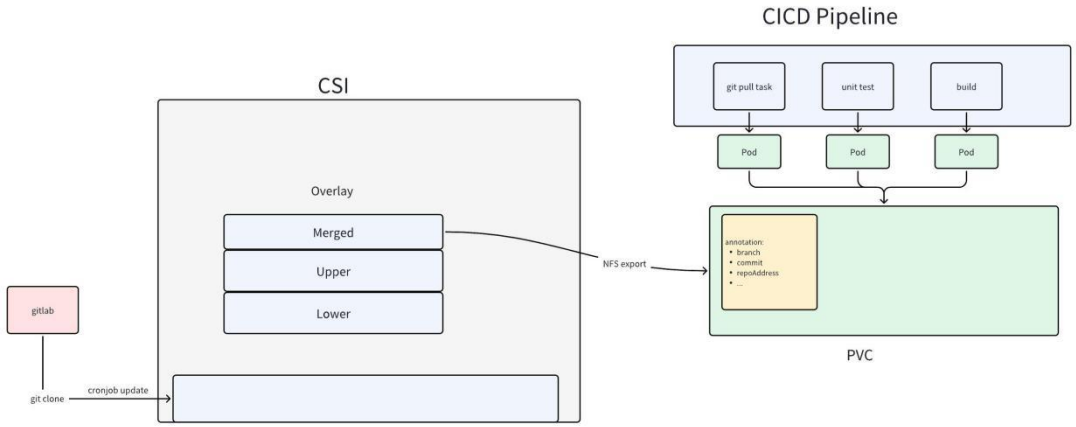
基于流水线的场景考虑，我们需要尽可能复用已拉取到存储介质里（PVC）的代码仓库，所以 OverlayFS 技术是适用于这个场景的。

通过定期将代码仓库中的代码全量制作快照数据，作为 OverlayFS 的 lowerdir，流水线任务拉取代码仓库的时候，只需要拉取少量变更代码（和上次快照之间的差异），可减少 90% 以上的代码拉取量，从而实现加速效果。通过 NFS 协议将 OverlayFS merged 之后的文件作为 PVC 的存储介质提供给流水线任务运行的容器。

K8s 技术选型

基于上面的结论，最简单的方式就是流水线任务 Pod 绑定一个 PVC，这个 PVC 对应于需要的代码文件。

在 CSI 二次开发实现侧，由于 K8S 内置了 NFS 挂载，所以我们的挂载协议基于 NFS 实现成本最低，只需要自己实现 Provisioner，PVC 声明 Provisioner 对应的 StorageClass 即可提供代码文件数据。



基于持续集成的代码可靠性实践

背景

在公司内部，随着敏捷开发和持续集成/持续交付（CI/CD）理念的广泛应用，开发团队需要频繁地进行代码提交和合并，团队也建立了每日集成的机制。这种高频次的代码集成如果缺乏有效的质量保障机制，可能会导致以下严重后果：

- 代码质量下降：未经严格检查的代码可能包含逻辑错误、代码风格不一致等问题，影响代码的可维护性和可读性。

- 合并冲突增加：频繁的代码提交和合并如果没有质量检查，容易引发更多的合并冲突，增加开发和维护成本。
- 安全漏洞：缺乏代码安全性扫描的代码可能包含潜在的安全漏洞，给系统带来安全风险。
- 测试压力增大：未经质量检查的代码流入测试环境，会增加测试人员的工作量，导致测试周期延长，影响项目进度。
- 发布风险增加：质量不过关的代码进入生产环境，可能导致系统不稳定，甚至引发严重的生产事故。

因此，为了确保代码的可靠性和稳定性，我们需要建立一套严格的质量红线机制，确保每一次代码提交和合并都经过严格的质量检查，只有通过质量检查的代码才能进入测试和生产环境。

保障机制

质量卡点任务

为了保障代码的可靠性，我们在 CI/CD 系统中集成了多种质量检查任务组件，具体如下：

- Sonar 扫描：集成 SonarQube 进行代码质量扫描，检测代码中的潜在问题，如代码异味、漏洞和技术债务等。用于确保代码的可维护性和可读性，减少技术债务。
- 单元测试：在代码提交和合并时，自动运行单元测试，确保代码的功能正确性。

- API 自动化测试：在代码提交和合并时，自动运行 API 测试，确保接口的功能正确性和稳定性。
- 作用：验证系统各接口的正确性，确保系统的整体功能稳定。
- 代码风格扫描：集成代码风格检查工具（如 ESLint、Pylint 等），确保代码风格的一致性。
- 代码安全性扫描：集成安全扫描工具（如 Snyk、OWASP 等），检测代码中的安全漏洞。

质量门禁设置

结合业务现状，有部分老服务代码问题较多，业务又在快速迭代，如果一刀切所有问题都修复才能上线，需要投入人力去修复老服务，投入产出比不高。因此，在 CI/CD 系统上，我们开发了质量门禁设置功能，允许研发管理者，在项目维度，将服务分组，不同分组适用不同的质量门禁，自定义以下质量指标：异味，bug，漏洞，单元测试覆盖率，来适配业务自身的情况，平衡代码质量和研发效率。

Sonar 扫描质量门禁常用分组如下：

- 老旧服务分组-质量阈值：bug：1，漏洞：1，异味：99，单测覆盖率：0.0%；也就是允许异味的存在，以及单测覆盖率为 0，但不允许 bug 和漏洞不修复

● 标准服务分组-质量阈值：bug：1，漏洞：1，异味：1，单测覆盖率：30.0%；不允许 bug、漏洞、异味不修复，不允许单测覆盖率小于 30%

● UI 界面示例

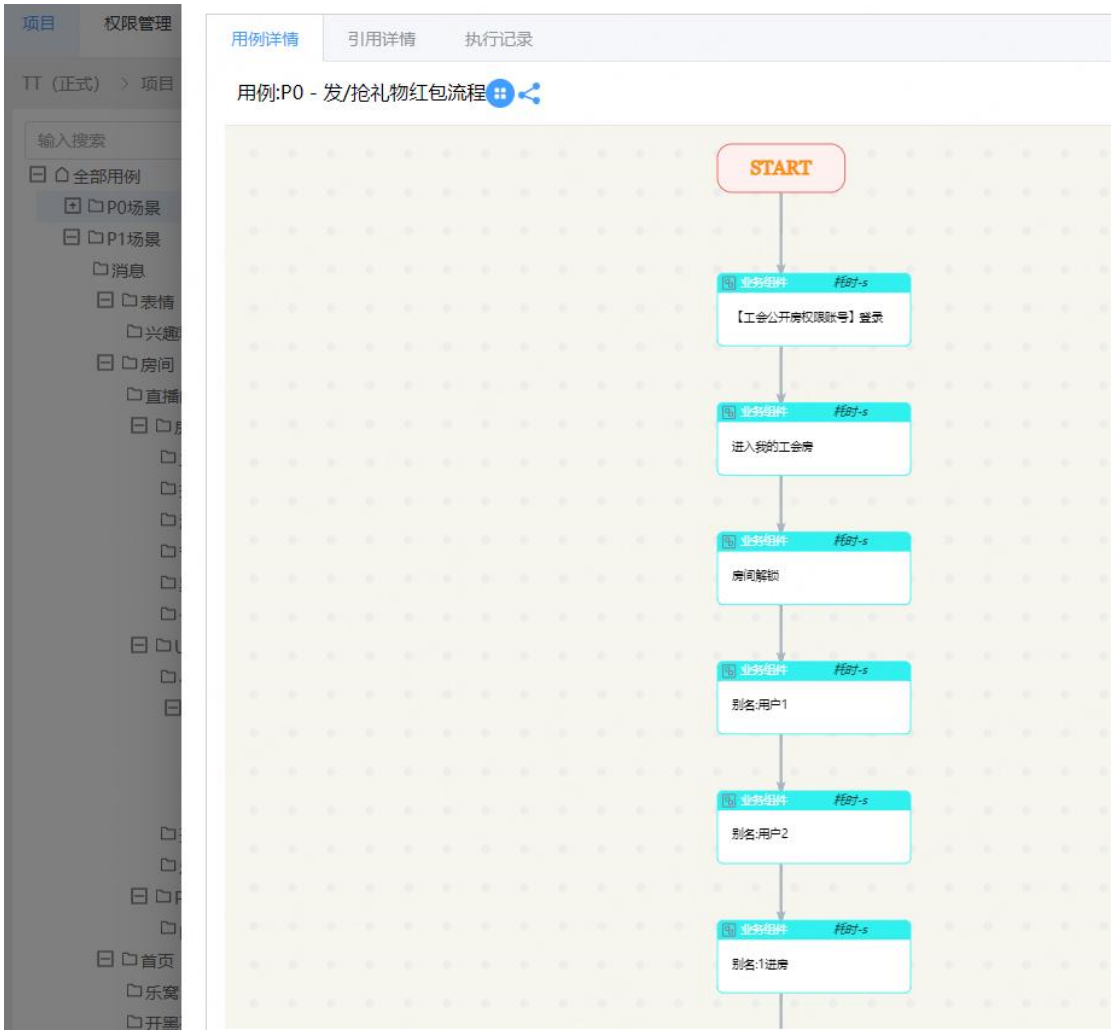


API 自动化测试

● 接口用例编写

在质量保障要求上，需要服务 100%通过所有的 P0-API 测试用例，不允许失败，需要说明的是，业务侧认定为 P0 级别的用例，都是 APP 最核心的流程，例如：登录，进房，上麦，送礼，等。

这些用例是测试人员通过自研的测试平台上编写，将业务流程，利用测试数据，按顺序调用不同接口来实现，编写界面 UI 示例如下：



若干这些用例组成的用例集合 100%通过，保证了单个服务的逻辑变更，对于 APP 主流程不会造成阻塞，依然符合预期，减少了测试人员回归测试的压力。

● 接口测试可靠性保障

同一个迭代往往涉及多个需求，多个服务的代码编写，如果所有正在开发调试的服务，都部署到了同一个测试环境中，在 API 自动化测试中发现有用例未通过，无法快速定位到是哪个服务的修改引起的，为了解决这个问题，在 API 自动化测试中，我们也使用到了后文中提到的“子环境”概念。

在执行 API 自动化测试之前，需将涉及的服务变更部署至独立的子环境。所有依赖的上下游接口服务请求将定向至稳定的基线环境。此举确保了在 API 测试执行期间，唯一发生变化的是当前服务。因此，若测试结果中出现未通过的用例，可以明确判定为当前服务变更所致，从而便于研发人员迅速定位并修复问题。

3.2.4 质量卡点任务运行效果

sonar 扫描，单元测试，代码风格扫描的结果，会在流水线中汇总体现，在整体质量卡点任务通过后，研发人员会收到飞书通知，提醒他本次质量卡点是否通过，也可以主动在流水线中查看，UI 示例如下：



飞书通知



流水线

API 自动化测试的结果同样也有飞书通知，以及流水线中展示。



红线机制

结合业务研发流程，在 CI/CD 系统上，我们设置了一系列的红线机制，具体应用在以下场景

识别到有问题的代码 Push：

在研发人员 push 代码时，自动触发流水线运行。如果有质量检查任务不通过，将会提醒研发人员提交有问题，请检查并修复，从而减少后续合并和发布的风险。

阻止不合格的 Merge Request：

在研发人员发起 Merge Request 时，自动触发流水线运行。如果有质量检查任务不通过，将会阻止研发人员完成 MR，直到问题被解决。这样确保只有通过质量检查的代码才能合并到主干，减少合并冲突和代码质量问题。

发布质量红线机制：

在运行发布上线的流水线时，如果本次编译无法通过质量检查，将无法进入提测阶段和上线阶段，建立了质量红线的机制，类似于安灯系统，确保流入提测和生产环境的代码质量，减少测试压力和发布风险。

通过以上手段和机制，我们建立了一套完整的代码可靠性保障体系，确保每一次代码提交、合并和发布都经过严格的质量检查，保障代码的可靠性和稳定性，减少测试压力，提高开发效率。

基于服务网格的研发过程保障实践

我司落地 Istio 四年以上，最初一年里流量管理能力仅仅只是应用在网关处通过 VirtualService 对指定 host 和 path 进行规则路由。随着 Istio 的稳定和功能加强，我司也在进一步探索 Service Mesh 在解决实际业务诉求方面的各种可能。

结合不同的应用场景与 Istio 的特定功能，分别在部署环境、金丝雀发布、开发过程三个方向取得了实质性的突破。

基于子环境的多环境稳定性保障实践

背景

各个业务团队出于对数据和网络的隔离，拥有众多的部署环境。包括：开发环境、集成环境、测试环境、灰度环境和生产环境，不同环境具有明显的差异。

- **开发环境：**由于频繁变更，存在不稳定的代码、服务覆盖等问题，从而导致环境不稳定;
- **测试环境：**直接影响 QA 环节，引入较繁琐的人工审批流程，以确保测试环境稳定;
- **跨部门服务因为依赖**，会进一步放大了环境不稳定的问题。

为保障各部署环境的稳定，特引入了基准与子环境的概念。

- **基准：**由相对稳定的服务/组件构建的可提供完整业务服务的系统，同一部署环境内只允许一套基准环境。默认标记名为“base”

- **子环境：**由至少拥有一个服务/组件构成，借助基准环境上游服务/组件，对有特定标记流量提供完整业务服务。一个部署环境可拥有多个子环境，同一服务/组件可存在于多个子环境，不同子环境具有标识唯一。

实现方案

资源部署：

单个 Namespace 下实现对单个应用多个版本的共存部署，在原有 Labels 的基础之上，对于子环境的资源（Deployment、Pod、ConfigMap、HPA）满足以下规则：

所有资源的命名.metadata.name 以子环境名字作为后缀，即{app}-{senv}，同时修改关联，包含：

Deployment 的.spec.template.spec.volumes[].configMap.name

HPA 的.spec.scaleTargetRef.Name

所有资源的 releaselabel 调整为以子环境名字作为后缀，即{app}-{senv}，同时修 Deployment 的.spec.selector.matchLabels 中的 release（基准环境的 deployment 必须保持原有的值不变：{app}）

所有资源，新增 senvLabel 用于标记环境，取值为子环境名字{senv}（基准环境的值为 base）

Deployment

YAML 自动换行 复制

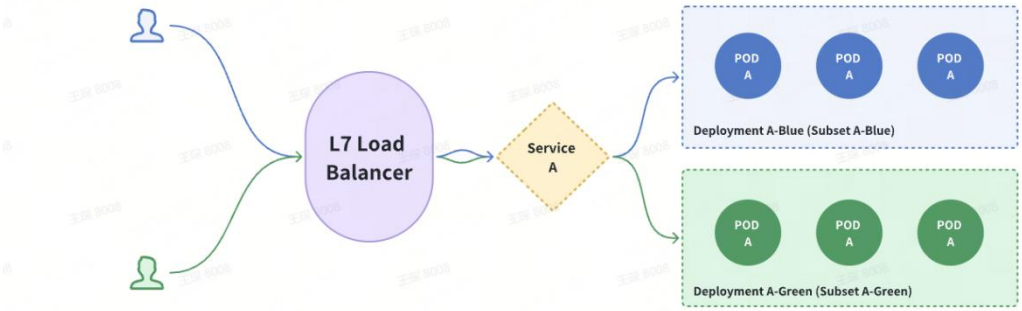
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: greeter
    release: greeter-canary
    senv: canary
  name: greeter-canary
spec:
  selector:
    matchLabels:
      app: greeter
      release: greeter-canary
  template:
    metadata:
      labels:
        app: greeter
        release: greeter-canary
        senv: canary
    spec:
      containers: ...
      // ...
      volumes:
      - configMap:
          defaultMode: 420
          name: greeter-canary
        name: config
```

ConfigMap

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  labels:
    app: greeter
    release: greeter-canary
    senv: canary
  name: greeter-canary
data:
  greeter.yaml: ""
```

路由管理：

通过单 Service+Istio Subset 的方式，实现流量分别到基准 or 子环境。



规则如下：

- DestinationRule：根据 senv 进行 subset 划分
- VirtualService：根据规则将流量路由到不同 subset 中

网址：SRE-Elite.com 微信：SRE 精英联盟

Destination Rule

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: greeter
spec:
  host: greeter.default.svc.cluster.local
  subsets:
  - name: primary
    labels:
      senv: base
  - name: canary
    labels:
      senv: canary
```

Virtual Service (Delegator)

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: greeter-delegator-80
spec:
  http:
    - name: canary
      match:
      - headers:
          x-qw-trafic-mark:
            exact: "canary"
      route:
      - destination:
          host: greeter.default.svc.cluster.local
          subset: canary
    - name: default
      route:
        - destination:
            host: greeter.default.svc.cluster.local
            subset: primary
```

基于流量染色的金丝雀发布实践

背景

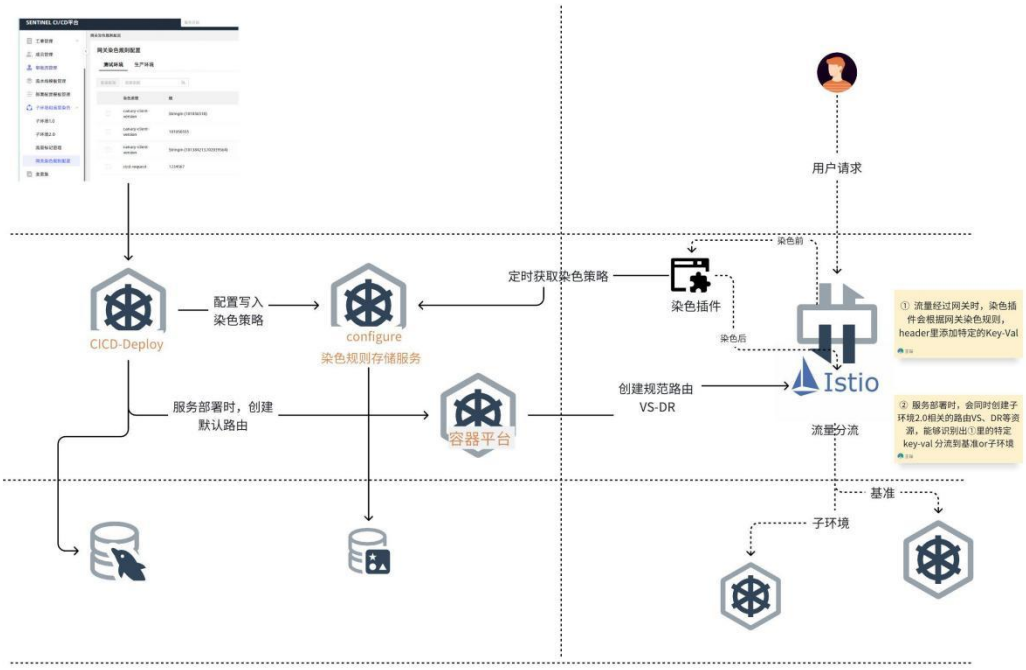
金丝雀发布（灰度发布）作为常见的稳定性保障手段，是指在生产环境正式发布之前，放入小比例的流量并逐步提升，旨在实现流量验证，避免程序缺陷引发大范围影响。以下是金丝雀发布常见的概念。

网址：SRE-Elite.com 微信：SRE 精英联盟

- 金丝雀环境：是在生产部署环境预创建的一个特别标记的“子环境”，其环境名字和流量标识均为 canary。
- 流量标记：HTTP/GRPC 请求头中携带的指定头（Key 为 X-Qw-Traffic-Mark）的值。
- 流量染色：于请求链路某一环节的中间件（网关、网关插件等），为请求设置特定流量标记的动作。
- 染色规则：用于网关插件中间件针对满足条件的流量，染色成对应的流量标记，所建立起来的一种关联关系。

实现方案

要实现金丝雀发布，在实操上是通过，配置特定的染色规则（用户设备号、用户 ID、App 版本号等）以流量标记的方式，附加到同学呢协议上，并由 istio 识别后，根据标记将部分小比例流量分流至相应的环境。



以下是相关实现金丝雀发布过程中，

控制面组件在整个方案中的作用：

- CICD 平台，Deploy 子服务，对 子环境 基本信息、流量标记、染色规则进行管理。
- 染色规则服务(Configure)，对网关染色规则进行实际的存储管理，作为染色插件的数据提供方。
- 容器平台，服务部署时管理具体的路由规则实例（VS、DR）。

数据面组件在整个方案中的作用：

- **Istio 网关：**作作为用户流量的入口，Istio 网关处理和管理进入服务网格的外部流量。它支持 HTTP、HTTPS、gRPC、TCP 和 UDP 协议，提供灵活的流量控制和路由能力。

- **染色插件：**纯自研的 WasmPlugin，通过 Istio Gateway 被加载。首先读取 Configure 服务的染色规则，然后根据规则对经过网关的流量进行“染色”，即，在 HTTP Header 中增加特定的 Key-Val。其中 Key 固定为 X-Qw-Traffic-Mark，Val 为染色规则绑定的流量标记。
- Istio（Virtual Service、Destination Rule）通过识别染色后的流量标记，采用 VS 或 DR 规则分流到具体环境的 POD。

通过上述组件的协同工作，系统实现了高效、安全的金丝雀发布流程。

基于子环境的开发过程加速实践

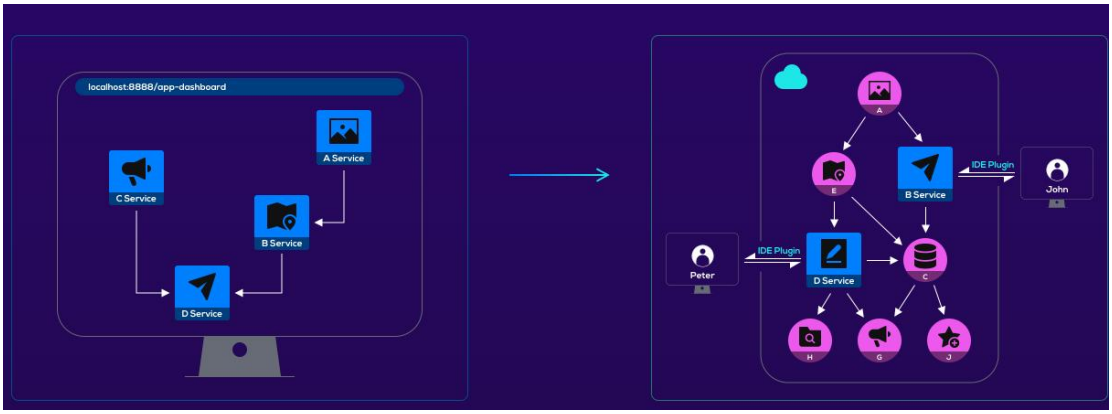
背景

在研发工序中，在需求-->开发-->联调-->测试-->发布整个流中，最频繁的动作就是开发-->联调的过程。特别在微服务、多人协作开发的模式下，往往会出现开发人员有多个版本需要在集群环境中调试的情况。而且在调试过程中也必须通过流水线发布才能发布到集群中，流水线的耗时等待更加加剧了这个过程。

实现方案

如果想要加速这个过程，最好的方式为：让开发人员拥有在本地 IDE 开发调试一样的方式，对集群中的服务进行调试。这样就减少了流水线的发布等待过程；借助子环境的隔离能力可以做到服务

版本的隔离，解决多人服务争抢的问题；借助子环境流量染色的能力，实现服务间的功能联调。方案前后对比如下所示：



既能降低开发人员的学习成本（与本地 IDE 调试一致），又能加速研发过程。





我们通过自主研发 Vscode、Jetbrains 插件的方式，为开发人员赋予了这样的能力。

IDE 插件的运行流程如下：

i.当开发人员需要进行远端调试时，会克隆当前服务在集群中的工作负载，并为其打上开发人员的标识（子环境、用户），使其作为服务的一个“子环境”在集群中运行；

ii.借助复用“CSI”的代码加速功能，将开发人员本地的代码增量同步到工作负载中；

iii.启动服务，并针对相应语言启动一个调试服务器（例如 golang 对应 dlv）；

iv.通过端口转发，将本地 IDE 的调试客户端连接到集群的调试服务器上；

v.调试结束后，回收集群中对应的资源。

如此一来，便实现了开发人员在本地远程调试服务的功能。

（三）总结及展望

目前，公司内部已有超过 95% 的业务实现了容器化，且 80% 的业务运行在服务网格中，覆盖了所有核心业务。基于 Kubernetes 和服务网格等云原生基础设施，公司构建了一套从

应用研发到发布上线的全流程保障体系，将效率和稳定性视为生命线。

面向未来，随着大模型与 AI 应用的兴起，研发过程保障迎来了新的机遇与挑战。例如，使用大模型辅助编码、进行代码审查（Code Review）以及问题排查等，大模型在研发领域展现出巨大潜力。我们将重点思考如何基于大模型构建一个能够保障各场景应用研发过程的体系，以进一步提升研发效率和质量。

3 入网控制

3.1 运行环境适配

3.1.1 运营环境设计

产品在经过了核心概念的提炼，通过对市场分析、确立用户定位、关注用户体验和产品风险等方面后，建立项目筹备小组，在完成了开发产品原型，验证技术风险，及通过相关评审确定进入量产阶段以完成产品的全部内容的开发。此时，将进入产品面向用户的试运营阶段，根据新产品评价体系每月对产品运营数据指标进行评测和汇报；确认产品质量是否已达到面向外部用户的标准；经过相关评审后，项目组可根据实际情况启动正式运营。

通过运营环境的规划与设计，可以根据过往的运营经验形成可运营规范，帮助项目组和研发规避相同的风险，同时结合运营策略进行推演正式运营后的情况，以此来提前发现潜在的风险和问题，便于和运营、研发团队进行评估和优化排期。同时，对于规模化测试所需人力、用户体验、成本预估、设备及机房选型、网络带宽资源及周边组件的容量，可用性等进行提前的评估和筹备。鉴于越来越多的组织采用多云环境，我们在规划过程中特别注重对公有云、私有云和混合云的供应商进行优劣势分析，同时充分考虑运营环境与云服务的耦合程度，做出恰当的选择和权衡。

从组织结构上来说，可以成立一个专家组，主要的工作项是横向提炼各业务在运营阶段中的各种共性问题、解决方案的沉淀和刷新，确立可运营规范。同时，在业务的关键节点参与技术评审。

对业务支撑团队，在进行运营环境规划与设计时，主要可分为如下几个步骤：

架构。业务决定量产并分配到支撑团队后，支撑团队需获取《业务架构说明书》《部署文档》，运营节点（建议近三个月为宜）等必要的业务相关文档从而了解业务架构、各模块功能和通信逻辑、容灾及技术指标等信息。以传统模式部署还是云原生模式部署，在这个环节会确定下来。

评审。运维专家及支撑团队组织项目组、研发团队进行可运营规范、评审标准、版本交付规范等的宣导;深入就业务架构，运营目标涉及的技术问题进行专项沟通，以此来对业务有个整体了解，梳理风险及问题。

验证。通过对业务测试环境的搭建，掌握业务搭建方法，并分析可自动化和改善的点，此阶段非必须建设自动化，因为业务架构成熟度和交付结构可能会经常变动。同时根据对外测试的的目标选定部署的机房、机型及合适的网络运营商等环境规划。

对齐。正式运营阶段筹备前根据节点时间进行倒推梳理整体支撑的检查列表，明确工作项、完成时间、负责人和细节部分，并与干系人逐一明确对齐，共同完善。和研发、运营共同确定业务稳定性目标 SLO 及相关的指标 SLI，以及围绕用户体验所需的场景和分析方式，并做好相关的数据埋点、呈现和监控。对于基线数据提前做好规划和收集。

迭代。配合业务压测，了解性能瓶颈，并重点进行方案的拟定和保障实施。根据运营中的问题刷新风险问题列表，并设计解决方案持续跟进，并在下次产品迭代更新后进行实际效果验证。

3.1.2 容器云适配

随着云原生技术的不断发展，容器以及 Kubernetes 等技术的长足发展给企业数字化转型带来了便利。容器技术将软件

运行环境打包成一个“集装箱”，方便在不同环节进行传递；而 Kubernetes 则将容器的调度和部署标准化，让开发运维人员不再关注资源层面的调度和容灾。容器云适配则是通过云平台、Kubernetes 和 Docker 等云原生技术帮助企业降低成本，加快业务迭代，对企业数字化转型提供强有力的技术支撑。

通过容器云适配可以简化了服务部署等方面的运维管理的复杂度，让业务团队更加专注于自身核心业务逻辑的开发，从而实现快速的业务迭代。同时通过虚拟化、资源池化以及弹性调度等技术，增加资源交互弹性，帮助业务团队不断降低资源成本，因此 SRE 应推动业务容器云的适配。

改造筹备。筹备相关的内容主要目的是梳理各模块之间的逻辑以及交互方式，便于针对容器环境进行适应性评估，明确业务架构与容器环境之间的差距，便于业务进行架构调整。例如按照业务进程本身是否缓存数据，可分为有状态模块和无状态模块，无状态模块。同时还需要关注进程的服务发现机制，如果业务自己有成熟的服务注册机制，例如业务自行设计的服务注册中心，那么在容器化的过程中，优先会有部分模块会落入 Kubernetes 集群中。

网络评估。一般情况下，Kubernetes 的集群内与集群外是两套独立的网络，一旦容器化后落入 Kubernetes 集群中，内网访问时网络链路就会发生变化，也就是 overlay 网络。在 overlay 方案中，我们可以认为集群中的所有容器，默认都可

以通过 IP 地址相互直连。集群内可以主动访问集群外，但是集群外看到的来源 IP 是集群节点的 IP 地址，并不知道容器 IP 地址。即使知道容器 IP 地址，集群外也不能直连容器 IP 地址基于这些条件，业务需要评估各模块访问关系，确认落入集群中后是否需要进行针对性适配。

工作负载评估。工作负载评估的基础原则是如何基于业务无损、不停机更新的情况下，可以选用哪种工作负载进行容器管理。对于各类更新方式：滚动更新、蓝绿发布、金丝雀发布，业务落地容器化必须至少选择一种方式进行管理，这是保障我们利用好容器特性的基础。常用的工作负载是 Deployment 和 StatefulSet，是 Kubernetes 原生提供的分别解决微服务部署，有状态服务部署的方案。建议优先评估是否可以采用原生工作负载进行部署。

弹性伸缩评估。弹性伸缩是云计算中一种常用的方法。通过设置伸缩规则来自动增加/缩减业务资源。HPA 是 Kubernetes 基于弹性伸缩进行的规则设计，Kubernetes 对 Workload 的资源使用（CPU，Mem）、自定义 Metric 指标进行负载评估：当负载超过设定高水位时，例如 CPU 平均负载超过 60%，就扩容 Pod 实例以降低负载到设定水位之下。当负载低于设定低水位时，例如 CPU 平均负载低于 20%，裁剪 Pod 实例以释放资源，降低成本。要实现水平扩缩，业务需要在业务逻辑与架构上实现优雅退出、负载均衡。优雅退出：业

务容器负载降低时，HPA 会针对模块实例进行缩容操作，业务容器必须要优雅退出，保障业务逻辑顺利完成退出动作。负载均衡：当负载增加时，HPA 会直接增加容器数量。无状态服务的处理方式较为简单，可以通过前置的负载均衡器有效均衡负载；有状态服务则需要业务动态感知实例/容量变化，合理分配玩家至新增服务实例。

容器云适配后的交付和管理。容器云适配后建议使用 GitOps 来进行持续交付。GitOps 是由 Alexis Richardson 在 2017 年首次提出的，该模型主张将 Git 作为“单一事实来源”来管理和同步开发和生产环境，而不是使用传统的基础设施管理和部署方式。GitOps 的核心思想是将应用系统的声明性基础架构和应用程序定义存放在 Git 版本库中。将 Git 作为交付流水线的核心，每个开发人员都可以通过提交拉取请求（Pull Request）并使用 Git 来加速和简化 Kubernetes 的应用程序部署和运维任务。每个环境都应该有一个代码仓库，用于定义给定集群的期望状态，然后 GitOps operator 会持续监控特定分支（通常是 master 分支），并在探测到 Git 发生变更后，将此次变更传递到集群，并更新 Kubernetes etcd 中的状态。通过使用像 Git 这样的简单熟悉工具，开发人员可以更高效地将注意力集中在创建新功能而不是运维相关任务上。

3.1.3 数据库存储适配

为确保项目数据库运行的稳定性、可靠性、可扩展性、安全性以及高性能，SRE 应该对数据库存储进行适配或优化，确保可以为应用程序提供高质量、高性能、稳定性强的数据存储和访问服务。主要包含以下几个方面：

选择合适的数据库类型。根据项目需求、项目架构设计和数据模型等，选择合适的数据库类型，如果数据具有固定的结构和关系，那么关系型数据库（如 MySQL、PostgreSQL、Oracle 等）可能是更好的选择。如果数据具有动态的结构，那么非关系型数据库（如 MongoDB、Cassandra、Redis 等）是更推荐的选择。除数据模型外，在评估数据库类型的时候还需要考虑项目的数据规模、业务场景、可扩展性、性能、一致性、可用性、成本等因素，综合考虑数据库选型。

数据库部署和配置。选择合适的服务器部署数据库服务，并进行最佳的数据库参数配置，如调整内存、连接数、缓存等参数，以满足项目所需的数据库运行的性能和资源需求。

数据库高可用和扩展。根据项目需求，实现数据库的高可用和扩展，如搭建主从复制、分布式设计、读写分离等。

数据库监控和告警。实施数据库监控，收集数据库关键性能指标（如 QPS、慢查询、磁盘空间等），并设置合适的告警阈值，以便在出现问题时及时发现并处理。

数据库备份和恢复。制定数据库备份策略，定期备份数据，确保数据安全。同时，要熟悉并掌握数据库恢复流程，以便在发生数据丢失或损坏时能够快速恢复。

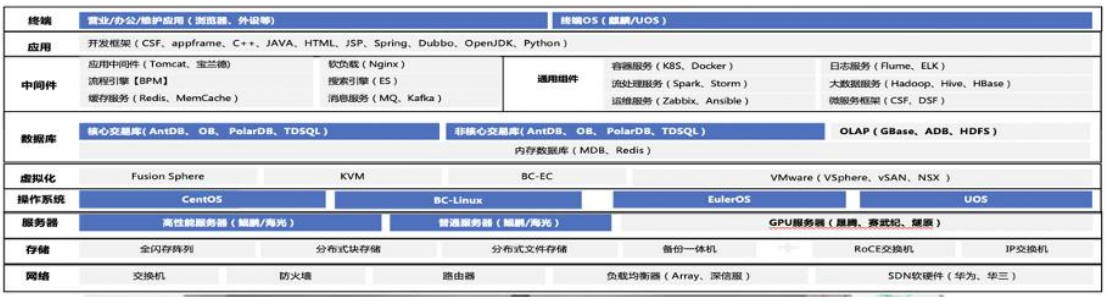
数据库性能优化。分析数据库性能瓶颈，优化查询语句、索引、表结构等，提高数据库性能。

数据库安全。保障数据库安全，如设置合理的权限、防止 SQL 注入攻击、加密敏感数据等。

3.1.4 信创适配

在政策推动下，自主可控成为当下热点。各大企业均在进行自主可控替代，加大信创布局。信创产品相较于传统商业产品，信创产品的技术成熟度、生态成熟度存在着客观差距。对于 SRE 来说，针对含有信创产品的运行环境适配，需要建立以下工作流程：总体目标制定、信创产品选型、产品验证测试、推动适配改造、业务测试、割接上线、运行保障。

总体目标制定。根据发展趋势，企业在整体 IT 投入中，明确对信创产品的采购比例。从而制定企业自主可控的总体战略目标，即年度含信创替代的项目计划，以及项目中进行信创替代的比例和对象，绘制全局信创替换后的架构图，重点标注端到端自主可控列表，包含硬件、数据库存储、容器平台及组件、终端等。每一次的信创项目完成后，均可以更新端到端的全局信创架构图以及具体的信创产品应用清单。



信创端到端全局示意图

信创产品选型。确定项目的信创替代目标后，具体在哪一层进行信创产品的替代也随之明确。根据项目的生产环境及业务场景，对替换对象进行选型。以数据库存储、硬件、中间件这些为例，SRE 有以下维度的选型策略。

1）数据库存储

在信创过程中，SRE 参与各类信创项目的开展，为确保生产运维的可维护性，SRE 需要在组件选型适配上重点关注。项目需要通过不同的业务场景完成数据库选型，从联机事务处理过程、后台批量事务处理过程、联机分析等业务场景构建出业务主要特征，根据操作特性、数据量、实时性、一致性和 SQL 依赖等主要特征来适配业务场景。

2）硬件（服务器、操作系统）

在操作系统选型上需要从管理清晰度、公司战略、使用成本和可维护性为出发点，综合考虑开源和商业的使用版本。

3）中间件

在中间件选型上要根据业务开发语言的适配程度、改造难度、可维护性等综合考虑，更多地了解实例基础配置的差异

性，以及各类中间件产品的特性，维护过程中的复杂性等确认选型产品。

产品验证测试。针对各个产品，搭建同步于生产规模的产品测试环境，主要包含计算环境以及网络环境。对产品自身的相关项目进行测试，以体现产品的能力。测试完后，输出详细总结报告，报告中应包含每一个测试项的结果指标、现存 bug 和原有的非信创产品的横向指标比对。

以数据库产品测试为例，会进行数据类型及语法测试、数据库性能测试（包含数据查询、更新、删除、聚合、多表关联等场景）、业务 SQL 测试、高可用测试、数据备份和恢复测试、可维护性测试（包含错误检测和提示、日志信息、在线升级等）。完成所有产品测试后，把测试过程中发现的问题和风险整合到产品测试报告，以供选型决策参考。

推动适配改造。确定好信创替代产品后，根据信创产品的特性，需要已有应用架构做关联适配，主要包含两类，一类是集成部署架构的适配改造，如一些国产化的操作系统，部署配置应按照其系统路径进行调整；另一类是应用的代码适配改造，确保兼容信创产品后不影响系统稳定性。以下列了分布信创的适配改造场景：

1）针对硬件（服务器、操作系统类）

进行编译适配：针对语言编译类型选择不同方案适配 CPU 架构；通过容器封装或者构建相似环境实现操作系统适配；

网址：SRE-Elite.com 微信：SRE 精英联盟

2）针对数据库

进行数据库解耦及开发框架升级：将与数据库多样化交互收敛到中间层，降低90%换库导致的应用改造工作量；将分布式事务控制在应用层；禁用存储过程和数据库链，禁止面向特定特性进行编码，实现面向对象编程。

3）针对终端

进行代码适配：业务SQL 类、函数类调整，CSS 布局，渲染适配调整；弹窗异常、回调事件不触发、按钮无效、无法投屏等适配。

业务测试。在应用架构根据信创产品进行适配改造后，进入业务测试阶段，首先在测试环境上，使用测试数据进行功能、性能测试，并进行切换演练。确认过程无误后，在真实环境搭建的创新集群上进行测试。

1）功能测试

第一轮采用生产核心的业务场景或回归测试用例，对信创集群进行接口调用。梳理报错场景进行分析改进。

2）性能测试

使用自动化压测平台，对信创集群进行压测，根据新集群和生产集群的比例，对比压测后的响应结果，评估集群的性能情况。

3）非功能测试及切换演练

针对评审中因引入信创产品后，可能引起的非功能性，比如由于数据库中间件的新增，需要重点测试批量数据操作的性能，由于开发框架的调整，需要重点测试故障转移、接口超时设置生效、服务熔断能力等。

此外，由于信创集群的引入，在割接中当晚会使用流量倒换的模式，把部分流量切换到信创集群/平面。所以需重点测试切换能力是否可以快速生效。

割接上线。通常情况下，为确保生产业务的稳定性，涉及信创产品替代的项目，采用先单平面改造上线，通过流量控制，把一部分生产流量引导信创平面进行验证，并行运行稳定后，再进行全量替换的方案。

割接当晚首选不停服发布，首先把部分流量引入到信平面，进行功能、性能及其他非功能性测试验证。验证无误后，把新老平面并行运行，并进行相应的测试验证。重点测试平面间切换开关的演练，如果次日业务量高峰时新平面出现问题，可随时启动切换开关，把流量全部引导到原有的老平面。

运行保障。针对信创项目割接，建设应急处理的三板斧，切流、重启、扩容，首先应保证割接次日系统异常时可通过切流快速应急恢复，其次信创平面的容器云平面，可实现自动重启。此外，还应具备弹性扩缩容或人工接入时的一键扩容能力。确保生产业务系统的稳定性。

网址：SRE-Elite.com 微信：SRE 精英联盟

在可观测维度，重点关注将信创领域的关键指标整合进统一的运维监控体系中。鉴于信创涉及从端到端的技术栈改造，其磨合过程可能较为漫长，因此，我们可以考虑采用 eBPF 技术来实现对网络、系统和应用的全链路监测，从而为系统故障诊断和持续稳定运行提供坚实的支撑。

3.2 运行环境交付

3.2.1 基础资源服务

围绕稳定性系统下，SRE 需要提供一个可恢复性、可扩展性的基础资源服务。基础资源包括：服务器、存储、网络、数据库、中间件等。SRE 应推动通过服务化方式交付基础资源。主要需要完善如下方面的内容：

推动可恢复性的基础设施环境建设。可恢复性的基础设施环境是指信息系统依赖的硬件基础设施环境、应用平台、中间件、数据库等资源能够有效地从故障中恢复。为了达到可恢复性，这些基础设施在技术选型、构建、运行保障中，需持续考虑到各种风险，包括自然灾害、技术灾难、请求激增、人为错误或外部攻击等，并采取预防性、冗余性、高可用架构等以减少潜在的损失。它还具有在发生灾难后快速恢复的能力，这包括备份、冗余、快速恢复服务和关键信息系统的保护。可恢复性的基础设施环境还强调在设计和规划阶段就考虑到恢复能力，而不仅仅是灾害应对和恢复阶段。

以云的方式交付基础设施环境。SRE 应推动混合云的平台服务能力建设，以支持按需选择一个能够满足需求的云服务。SRE 根据应用程序的要求、用户的数量、数据流量等因素，在云平台的服务目录，在线选择资源池的 CPU 、内存、存储、网络、数据库、中间件、平台应用服务等资源，并提交相关申请。SRE 可以在线查看已分配的资源和运行的应用程序，并可以通过管理终端对应用程序进行管理和维护，例如，启动、停止、重启、删除等操作。当不再需要某项资源时，可在线释放该资源，并将其返回到资源池中。同时，SRE 应在云环境部署应用程序，包括安装和配置应用程序、配置相关的网络、安全设置，以及在部署完成后，对云环境进行管理和监控，以确保其正常运行和安全性。

量化评估基础资源环境。SRE 应建立量化基础资源服务的稳定性与成本管理指标，包括平台性能（资源服务的平均响应时间、吞吐量、处理能力等）、可靠性（资源服务的可用性、容错性、故障率等指标）、可扩展性（资源服务的系统扩展性、负载均衡能力、资源管理能力等指标）、安全性（数据安全性、访问控制、漏洞管理等指标）等。成本指标，以及资源成本管理（资源的利用率、调用次数等指标）。

3.2.2 可观测策略

可观测主要是为了应对 IT 运行环境与技术架构复杂性，重点解决 SRE 在故障发现、故障诊断与故障恢复环节的应急

过程管理。要求 SRE 需要从原来只负责可用性被动保障的角色跳出来，站在白盒角度看系统运行状况，剖析系统层面的运行信息。SRE 在推动可观测试策略时，需建立以下工作流程：

制定可观测标准化规范。为有效实施可观测性策略，需推动监控、日志、链路追踪等相关技术规范的制定，确保信息系统满足这些技术要求。规范应聚焦于增强主动监控性能指标数据上报能力，优化日志的标准化和可读性，实施数据埋点以建立链路追踪，并提供必要的基础设施服务支持，如系统监控数据上报和日志采集分析。技术规范应适用于新建和现有系统的持续优化，并明确配套的工作流程，确保可观测性工作贯穿软件的需求、设计、开发、测试等各环节。此外，组织内需要加强规范宣讲，推动系统架构师和研发工程师等人员认识到可观测是系统建设的必要条件。

建立可观测工作小组。可观测在生产保障中应用，但具体工作涉及产品、研发、测试、运维多个团队的协作，SRE 需根据可观测规范建立以系统或业务为单位的可观测工作小组，以提升协同效率。SRE 是可观测工作小组的牵头人，负责推动具体系统可观测能力的持续建设。因此，SRE 需要清楚的知道系统的业务运行状态、应用服务状态、批处理状态、性能管理、容量水位、依赖环境等黄金指标，明确相关黄金指标的数据计算逻辑，推动相关功能的建设与验收。同时，需要提前参与到系统设计阶段，梳理指标异常监控策略，以及系统依赖的应用

平台、系统软件、基础设施、上游系统的监控策略，以及如何通过应用日志辅助故障定位。另外，需将系统的上下游、交易请求链路、资源依赖等关系作为软件交付物之一，系统能够为感知链路节点的健康状况提供数据。

融入需求设计评审阶段。SRE 在系统的非功能性设计评审阶段，应参与可观测方案的制定，提出软件可观测需求，并明确具体的监控、日志、链路的数据埋点，以及工具支持的需求。

跟踪可观测的技术实现。在入网控制阶段，SRE 应构建完善的数据采集、加工与处理平台，涵盖日志管理（如 ELK）、性能指标监控（如 Prometheus、Zabbix）、链路追踪（如 Zipkin、Apache Skywalking），持续性能剖析（如 Pyroscope），eBPF 无埋点观测等能力。

为了处理和分析不同工具产生的可观测数据，我们建立了一个统一的数据关联框架，该框架包括数据收集器、处理器、存储系统以及查询和可视化工具。这些组件共同作用，实现了多种观测数据的关联融合，并支持多层次的数据下钻和追踪，构建起了一个可观测的拓扑结构。

此外，还需建立一套全面的数据质量监控机制，覆盖数据的生成、收集、加工和使用各个环节，以确保数据的准确性，可靠性及实时性。

验收可观测策略交付。SRE 在入网控制阶段应验收可观测相关数据埋点、可观测工具、生产环境配置等工作，确保系统上线后，配套的可观测工作机制能够顺利开展。

3.2.3 自动化策略

SRE 应尽可能的推动自动化一切的工作期望。自动化策略是将事件驱动思维模式融入到运维的方方面面，可以从思维、技术两个角度发力。思维角度，即 SRE 组织从一线操作、二线运维、管理岗位，要对重复性、操作性的工作琐事有天然的排斥感，并想方设法用软件方式代替手工操作。技术角度，一是从 SRE 工具层面建立以原子脚本、编排任务、任务调度的自动化操作能力；例如，通过编写脚本来自动化日常运维任务，并使用如 Terraform 等工具来实现基础设施即代码（Infrastructure as Code，IaC），确保环境的一致性和可重用性。二是将 SRE 手工操作标准化，并将标准的运维操作场景化，基于场景将自动化操作与工作机制相结合；这可以通过编写标准操作手册（SOPs）并将其转化为可执行的代码。三是 SRE 工作前移，推动应用系统自身自愈或无人值守的可靠性设计。

推动定时任务集中管理策略。为了解决定时任务的作业调度可靠执行问题，在没有集中作业调度策略前，SRE 需要各显神通，采用类似 crontab 、Windows 定时作业，以及基于软件系统程序的定时作业等解决方案。由于多个定时作业执行状态

可能会相互影响，分散式的定时作业管理容易引发风险、效率、管理等问题，比如：遗漏某些重要的业务调度步骤引发账务风险，手工任务周期设置有误导致任务漏做或重复做风险，调度批次异常无法及时发现的风险，人工重复执行不可重复操作作业任务，重复建设调度管理工具扩大成本等。因此，SRE 应推动集中式的作业调度自动化管理，例如部署 Kubernetes CronJobs 等工具，将规律性的任务固化由机器执行，支持多种目标端对象的任务执行、支持统一采控的远程任务执行、支持低代码的流程编排、支持快速应对作业异常处置等能力，解决系统稳定性保障涉及的生产力、安全控制等问题。

推动软件交付的自动化策略。软件交付是 IT 的关键价值创造，持续交付是提升软件交付速度的一个工程实践。持续交付的软件发布方式提倡全自动化，即围绕软件程序的部署，编排发布各环节的自动化操作。发布流水线是自动化策略落地的关键技术，流水线将一个软件发布环节串联起来，让软件交付过程中不同的角色可以透明地看到整个过程。同时，通过线上化各环节的执行步骤能够量化持续交付的水平，比如自动化测试覆盖率、缺陷数量、每天构建次数、发布平均时长等，量化数据能让团队清晰地看到低效环节并进行改进。对于不同的应用系统，SRE 应建立针对性的自动化发布策略，比如针对敏态应用，SRE 还要推动蓝绿、灰度/金丝雀、滚动、红黑的自动化部署策略等。

推动故障自愈策略。自动化的故障自愈是指通过自动化的方式，在系统出现故障时快速、准确地恢复到正常状态，减少人工干预的必要性，提高系统的稳定性和可靠性。SRE 在推动系统的故障自愈时，可以从应用程序应对异常时自愈的自动化，包括应用程序健壮性涉及的降级、熔断等，以及在应用程序以外配置的自动化自愈策略。从技术实现角度，主要需要解决自动化故障检测策略涉及的预设的故障检测规则和算法，系统可以自动检测故障，并根据预先设置的动作自动修复。

推动应急预案自动化策略。SRE 首先需推动应急预案的线上化，线上化应急预案有助于将标准化动作自动化，比如针对主机、应用服务、容器等最小运行对象单元的应急。线上化应急预案后，可推动预案的策略管理，支持多种策略组合、策略发布与组装、场景编排、通用场景等。接下来，SRE 应推动预案策略与自动化工具、接口和流程执行，关联场景绑定。

3.3 测试策略

3.3.1 连通性验证

通常在项目集成部署完成后，SRE 会联合开发团队对部署的环境先做连通性测试，目的是保证各个系统或模块之间的数据传输正常、请求和响应的格式正确，并且确保接口在不同条件下的稳定性和可靠性。主要可以分为以下四个步骤：确定

测试目标、构建测试请求、单调用连通性测试、压测连通性测试、记录和报告测试结果

1）确定测试目标

根据部署的架构图，确定要测试的系统范围以及调用通路，列出各个系统模块要测试的接口清单，包括系统模块名称、URL、IP 地址、端口号等信息，以及被测试接口的功能、参数、请求方法（如 GET、POST 等）和预期响应格式（如 JSON、XML）等。

2）构建测试请求

根据接口的特点和需求选择合适的测试工具。一般选取最核心的业务功能创建连通性的测试请求，包括请求的 URL 或 IP 地址、请求方法、请求头、参数等。

3）单调用连通性测试

利用选定的核心业务功能测试请求，对各系统模块接口发起单笔调用，检查返回的数据格式、数据内容是否满足预期结果。用此方法遍历各个系统模块之间的边界连通性。

4）压测连通性测试

完成基本的单笔调用连通性测试后，为进一步评估接口的性能。利用压测工具，对核心业务场景用力发起生产 100%的压力测试（目前实际使用过程中，有流量回放能力的公司，直接取核心业务的生产流量即可）。检查成功率，对错误响应进行核实，明确是功能参数问题还是性能问题，并予以改正。

5）记录和报告测试结果

记录测试过程中的关键信息和结果，并生成测试报告。测试报告应包括测试目的、测试环境、测试脚本、测试结果、问题和建议等内容。

3.3.2 功能测试

SRE 功能测试主要指在开发已完成交付测试，提交可交付代码版本的基础上，正式接入生产环境进行的测试，其功能测试主要包括版本功能测试（针对当次版本变更功能的测试，验证可用性、准确性）和回归测试（针对系统现有核心功能进行回归验证，确保非当次版本变更内容的准确性）。整体功能测试可以分为以下几个步骤：测试流程和计划制定、生产验证测试执行、当晚缺陷处理及次日故障跟踪、测试故障分析及评估、测试用例持续设计和维护。

1）测试流程和计划制定

制定上线功能测试管理流程，贯穿测试过程中缺陷的提出、处理、复测、结束等各个阶段，按照工作任务规范职责，形成测试计划清单，及时高效地关键节点进行监控，从而保证上线验证测试的有效性，以提高系统上线的整体质量。

2）生产验证测试执行

第一阶段：测试数据准备；

第二阶段：测试执行。将自动化用例通过平台一键式发起执行，在确认所有自动化计划发起完成之后，开始执行手工用

用例。上线当晚上测试组根据上线范围进行自动化测试，如遇到异常情况影响测试进度，及时通知客户并协调相关人员处理。测试组严格执行上线测试流程，明确红线时间，完善联动升级和通报机制，确保上线测试过程顺畅。

3）当晚缺陷处理及次日故障跟踪

生产当晚问题处理：主要分为四步，对于生产实时出现的问题，及时上报上线或变更管理员，协调相关资源协作排查问题，同时实时通报问题现象、进展、解决方法和最终处理结果。

测试报告编写：功能测试组编写系统验收测试报告，测试报告内容包括缺陷发现时间，测试进度，缺陷的准确描述，以及后续缺陷修复情况等。

测试进度通报：功能测试组对生产验证测试报告进行通报，若当晚未发现缺陷，则发送 60 分钟核心用例测试阶段性通报；若当晚发现缺陷，则发送问题发现通报并且整点时要发送问题进度通报。

测试缺陷处理：当测试发现缺陷，先做专业性的判断，是否为真缺陷还是业务理解问题，或者是电脑、手机、网络等客观因素导致的缺陷，若为真实缺陷，联合开发团队共同解决当晚的测试缺陷。

测试任务结束：如果当天上线依然存在遗留问题或故障，需要交接给次日保障人员持续跟踪、解决和通报。同时，生产

上线当晚测试内容进行总结，内容主要包含准发布验收测试问题分析、生产验证问题分析、生产次日相关故障分析、生产验证测试分析，并输出上线生产验证情况总结。

1）测试故障分析及评估

对生产上线当晚测试发现的问题或故障进行还原，并将故障现象和测试抓包数据反馈给保障值班人员。通过保障值班人员处理之后，将故障进行记录，包括故障现象、解决方式、故障分析等信息。以每周为维度统计上线次日故障，罗列出SRE相关故障总数，划分出重大故障、重要故障、一般故障的个数，进而细分未覆盖故障数与已覆盖故障数目，分析得出故障未覆盖原因以及问题归属系统，评估可进行优化覆盖用例数目。

经生产验收核心业务回归测试后，系统或平台未发现任何重大故障。这意味着系统在回归测试过程中成功通过了核心业务功能的验证，并且没有发现对系统关键功能或数据造成重大影响的故障。减少了重要故障的风险。重要故障指的是那些可能导致系统崩溃、数据丢失、功能不可用或对业务流程产生重大负面影响的故障。然而，需要注意的是，虽然没有发现重大故障，但仍可能存在一些较小的问题或不太常见的故障情况，因此仍需对系统进行持续的监控和改进。

2）测试用例持续设计和维护

网址：SRE-Elite.com 微信：SRE 精英联盟

用例库基于业务量、投诉量、故障标准和企业考核标准将业务分成四个星级，星级越高，业务越重要。并定义二星级以上为核心业务，在生产回归测试中予以覆盖。对用户有独立入口操作的页面定位为业务场景，作为用例建设的基础。通过调研用户使用系统的习惯，录制前台 UI 操作用例。并通过对核心参数进行多枚举值覆盖的方式，建设多路径覆盖的测试用例，对 UI 用例进行补充。每一轮生产验证测试复盘完后沉淀的改进方案和补充用例，都会纳入测试用例库。

自动化用例维护均在上线前 1 至 2 天内完成，可以确保系统上线前的最后一轮检查和修复。自动化用例维护涉及对已有的自动化测试脚本进行更新和修复。当系统发生变更或更新时，现有的自动化用例可能会因为页面结构或功能变动而失败。因此，在上线前，测试团队会仔细检查自动化用例的稳定性和有效性，并根据需要修复脚本中的问题。演练平台创建计划演练发起周期性测试，提高测试效率和准确率。

3.3.3 性能压测

性能测试包括压力测试与负载测试。全链接压力测试通过超负荷的负载条件来测试系统的稳定性和可靠性，以确定系统在极限负载下是否能够正常工作。负载测试则模拟用户访问系统，检测系统在不同负载下的响应时间和资源使用情况通过。在进行全链接压力测试时，定义性能指标和测试场景，设计合适的测试用例，并可使用 Jmeter 工具执行测试。通过监控系

统的性能指标和记录测试结果，可以分析系统的性能瓶颈并提供优化建议。总结而言，性能测试是一种评估系统性能和稳定性的测试方法。通过模拟真实场景的用户行为和系统负载，它可以评估系统在不同负载下的性能指标，并发现性能问题和瓶颈。

1）全链路压测工作流程制定

制定上线压力测试管理流程，贯穿测试过程中方案设计、缺陷的提出、处理、复测、结束、上线后的复盘改进等各个阶段，按照任务分类明确职责，且各任务对应到具体方案及工作清单，保证生产压测管理流程可以切实得到执行，从而保证生产压测的有效性，以提高系统上线的整体质量。

2）全链路压测方案设计

针对某次项目压测，根据不容的业务场景，设计压测方案，主要确定实施范围和实施方法。

（1）确定实施范围和压测指标

按照生产实际业务大类，可以分为查询类和受理类，因模拟受理类业务容易产生脏数据，工作量成本和代价较大（受理占比较大的一些场景可以考虑模拟受理类）。通常情况下我们的压测实施范围首先定位为查询类业务。针对生产系统如此多的查询类场景，既要能够压测出生产瓶颈，同时又能够将影响面降到最低，较好的方法是选取 TOP 查询类的场景作为压测范围。压力测试无论是生产环境还是测试环境，都需执行相关

网址：SRE-Elite.com 微信：SRE 精英联盟

的指标来衡量压测的效果，也便于后续统计评估同一个接口在各月的性能趋势。

（2）确定实施方法

集中化场景压测：指单纯的查询类生产业务压测，跟真实生产业务场景存在一定的差异，为了更真实地拟合生产实际，考虑通过对单集群的压力测试来反映真实的生产场景。

混合性场景压测：针对爆发式增长的特殊业务场景，比如“充值送话费”活动，“充值”等，他们主要的特征是在某个特定时间段集中迸发而导致业务受理的瞬时高峰，因此，我们需要模拟混合业务场景对系统进行压测，测试系统的性能短板，并指导资源的最佳配置。

3）全链路压测执行

（1）执行生产验证测试必要条件

各系统都连接好，可以测试全流程的业务；生产环境测试结果能够真实反映系统运行状况；生产环境能测试一些入网环境无法测试的场景；生产验证测试能够发现一些由于环境差异导致的软件缺陷。

（2）测试执行的工作项

其一是测试案例执行，其二是执行结果汇总，指第三方测试组对测试案例执行结果进行汇总；处理缺陷即生产测试验证支撑组修复测试过程中缺陷，提交测试用例让执行组进行回归

测试；其四是测试管理工具的管理与维护；最后是协调与管控，对执行过程中遇到的问题协调多方进行处理与解决。

（3）生产验证测试执行策略

生产验证测试执行就是根据测试案例编写阶段编写的生产环境测试用例来执行，不过在执行测试时有几个地方需要注意：仔细检查软件生产环境是否搭建成功；注意测试用例中的特殊说明；注意全面执行测试用例，每条用例至少执行一遍。执行测试用例时，要详细记录软件系统的实际输入输出，仔细对比实际输入和测试用例中的期望输入是否一致，不要放过一些偶然现象。

（4）实时监控

对于生产压测主要监控内容有：云平台监控、日志监控、应用监控、CSF 接口监控、网络监控、服务治理监控、主机监控等等。

4）生产当晚问题处理以及次日故障跟踪

对于生产实时出现的问题，及时上报上线或变更管理员，协调相关资源协作排查问题，同时实时通报问题现象、进展、解决方法和最终处理结果。

测试报告编写：测试组编写系统验收测试报告，测试报告内容包括测试结果，缺陷修复情况等。

测试报告核实：测试组对生产验证测试报告进行核实，主要核对测试执行状况、测试结果、缺陷修复情况。

网址：SRE-Elite.com 微信：SRE 精英联盟

测试报告发布：入网测试报告核实通过后，测试管理组组长发布入网测试报告。

生产准出确认：根据测试计划和测试报告分析各项指标作为评判依据，如案例覆盖率、入网测试通过率、关键业务测试通过率、端到端测试通过率等，测试结果是否满足准出条件。

最后是生产次日故障跟踪，主要是用于确定压测当晚的实施效果是否存在不足或遗漏，跟踪人员通过次日跟踪也能够更加熟悉生产，对保障生产稳定性以及全链路压测和生产环境的紧密型都有极好效果。

5）压测总结报告分析与评估

对测试结果进行分析，根据测试目的和目标给出测试结论。通过对接口流量的业务成功率、系统成功率和生产耗时等重要指标进行详细的分析，这些采集到的数据被视为本次上线压测接口的基线，用于评估系统在压力下的表现。在性能回归测试的压测过程中，性能测试组成员会将实际的接口耗时、业务和系统成功率与基线进行比较。这样的比较能够清楚地显示出接口性能的升降幅度，进而判断出系统可能存在的性能瓶颈和潜在问题。

同时，性能测试组成员也会积极抛出问题并持续追踪这些问题。在追踪的过程中，他们会对各个问题进行深入的分析，并提出切实可行的建议和优化方案。这些建议和优化方案的目的是为了解决相应的问题，并最大程度地提升系统的性能和稳

定性。通过持续的压测和性能优化工作，性能测试组成员能够不断改进系统的性能，并确保系统在高负载情况下仍能够正常运行。为业务的顺利进行提供了坚实的基础，也保证了用户能够获得更好的体验和服务质量。

6）全链路压测自动化与维护

基于开源的 Jmeter 工具做定制开发，形成全链路压测自动化工具。其架构是控制台、压力生成器、分析器，主要作用是配置测试场景，通知代理器进行数据初始化或清理，搜集测试过程中被测系统各个环节的性能数据，并根据要求模拟一定数量的虚拟用户对被测系统发送业务请求，实现对被测系统的压力测试，其中一个虚拟用户对应一个业务并发；将会发送测试的过程及结果数据信息给控制台进行采集汇总。最终分析并展示测试结果，同时对测试结果进行保存。

每月例行对生产全链路压测结果、测试账号、测试数据和测试场景用例进行检查，对场景变化的用例进行及时维护，对出现问题的测试账号、测试数据等及时地修复，以确保生产压测的准确性和安全性，确保各种数据可持续、可继承、可追溯。

3.3.4 数据迁移

1）数据完整性和准确性测试

由 DBA 采用数据库稽核工具，支持同构/异构数据库间的数据稽核比对，包括 count 稽核和表全字段稽核，通过监控可

以快速定位两端的差异，保证两端数据库同步数据的完整性和准确性；并且支持割接后增量数据回传，保障割接后发生故障能够快速回切恢复。

2）可用性测试

由 SRE 进行迁移可用性测试，目的是确保数据在迁移后能够正确地被使用。测试所有核心及非核心的功能，并确保它们达到预期的结果。另外，在迁移完成后的一段时间内（如每日早晨的开门测），针对之前已经测试过的功能进行再次回归测试，以确保在迁移后的初期没有引入新的错误。

3）性能测试

由 SRE 进行迁移性能测试，测试系统的在迁移后的性能是否与之前相同。同全链路性能测试的方案，按步骤进行验证。

3.4 变更评审

变更评审主要是为了降低系统变更投产带来的风险，并让变更如期交付业务。变更评审中，不同的 SRE 组织会在系统交付生命周期的不同阶段建立相对应的变更评审机制，比如项目立项环节的可用性评审、技术或部署架构可用性评审，设计阶段的非功能性、可运维性评审，上线环节的 CAB 评审等。

3.4.1 稳定性架构设计评估

SRE 组织为了推动稳定性架构管理，建议在整个技术线层面建立跨团队的技术架构管理组织，负责制定稳定性架构管理规范、技术组件规范，以及相应的技术管理与技术评审流程。SRE 应重点从高可用、故障恢复、可扩展性、数据完整性、部署环境角度推进相关评估工作。

高可用是运维管理的一条底线保障要求，运维主要工作是消灭单点风险，提升系统韧性，比如数据库中提到的主备、主从、分布式，数据中心的两地三中心、分布式多活，以及将一个应用系统同一个服务组件部署在多个数据中心机房、不同物理机的多个虚拟机上、为应用的负载均衡提供网络硬件或软件负载均衡器、提供具备高可用架构的消息中间件等 PaaS 云服务等。为了更好地推进评估工作，SRE 需要提前提供架构高可用的规范，制定通用组件、信息系统架构高可用参考模式，将高可用要求更早地落地在系统设计过程中。

故障恢复可借鉴最佳实践、具体信息系统的特点等，制定相应的故障恢复能力要求。在应用系统层面，需关注应用拆分、服务或系统交互解耦、服务无状态、减少总线节点服务依赖、增加异步访问机制、多层次的缓存、数据库优化、限流与削峰机制、基础设计快速扩容等。在基础设施层面，具备可恢复性的基础设施环境必须能够高效地从自然灾害或人为灾难中迅速恢复。这意味着需全面考虑各类风险，涵盖自然灾害、技

术故障、人为失误及网络恶意攻击等，并采取相应的预防措施以最大限度地降低潜在损失。具体措施至少应包括数据备份、系统冗余、快速恢复服务以及对关键信息系统的严密保护。

技术架构的可扩展性是指在不影响现有系统功能和性能的前提下，系统能够扩展或增强其功能的能力。通常涉及对系统设计、架构和模块化的考虑，以便于未来的扩展和升级。这包括将系统拆分为多个独立的子系统或模块，每个模块负责特定的功能和业务逻辑，降低整个系统的复杂度与模块间的耦合度，提高扩展性；支持横向和纵向的扩展能力，通过增加服务器数量或提高每个服务器的处理能力，来提升整个系统的处理能力，或通过升级单个服务器或组件的硬件来提高其处理能力；系统支持弹性伸缩，即根据系统的负载情况自动调整计算和存储资源，以实现系统的动态扩展和缩减；减少对总线节点服务的依赖，将多节点组成的逻辑交互改为端对端访问方式，以减少交易影响因素，避免单节点性能问题影响其他应用系统；增加异步访问机制，因为同步机制在性能出现问题时，会在短时间内耗尽最大连接数，即使这个最大并发数是正常情况下的 10 倍。将同步连接改为异步通信或引入消息队列都是解决上述问题的方案；支持多层次的缓存，可以从前端、应用内部、数据库等层面建立缓存。

数据完整性是运维保障的底线要求，持久化数据的生命周期通常会比系统与硬件的生命周期长很多，很多新系统上线或

架构调整都考虑数据迁移工作。同时，还要关注一些复杂性数据处理，比如：批次、清算、对账等操作，这些操作极易受数据问题影响，运维侧需要关注数据处理的异常中断原因定位、哪些环节是可以应急中断、中断后是否支持多次重试、与第三方系统约定数据不一致时以哪方为基准等应急处置机制。

选择合适的部署环境需要考虑多种因素，包括应用程序的特性、性能要求、可扩展性、可靠性、安全性、服务提供商的支持和成本效益等。不同的因素将对部署环境的选择产生重要影响，比如某些应用程序可能需要大量的内存和计算资源，有些需要选择能够提供高吞吐量和高数据处理能力的平台服务，有些需要选择云服务使应用程序能够随着需求的变化而扩展，有些需要选择高度稳定与安全性的基础设施环境。

3.4.2 非功能性技术评估

运维的非功能性设计是主动应对可运维性问题的切入点，直接决定系统在生产环境的成本与收益，甚至决定系统生命周期的长短。以下罗列运维侧需要推动的非功能设计。

系统运行状况可观测。云原生提出可观测的监控指标、日志、链路三要素同样适用于传统以主机为代表的技术架构。运维是在一个黑盒子的成品上进行监控、日志、链路完善，像 NPM、APM、BPM 等是运维侧发起的一些解决方案。从非功能设计看可观测，需要运维前移，推动必要的监控、日志、链路相关的研发规范，提升主动上报监控性能指标数据的能力，

优化日志的可读性，并提供必要的基础设施服务支持，比如支持系统监控数据上报、日志采集分析等。

故障隔离与服务降级。故障隔离和服务降级的目的是以牺牲部分业务功能或者牺牲部分客户业务为代价，保障更关键的业务或客户群体服务质量，是防止连锁性故障蔓延的方法。在设计中，运维侧需要从系统或业务角度，梳理应用系统所调用的各个服务组件，对各个服务组件出现故障时的假设，及应对措施。

性能评估。性能容量管理主要基于响应时间（系统，功能，或服务组件完成一次外部请求处理所需的时间）、吞吐率（在指定时间内能够处理的最大请求量）、负载（服务组件当前负荷情况）、效率（性能除以资源，比如 QPS=并发量/平均响应时间）、可扩展性（垂直或水平扩容的能力）等黄金指标开展。性能问题对稳定性的挑战不仅仅是单组件不可用，更大挑战是某个组件性能问题不断扩散到别的组件，导致大规模的故障。性能问题极易引发复杂的异常问题，SRE 需要加强相应的技术评估。

终端版本向下兼容。移动化后终端版本的管理越来越重要，在架构上一方面需要尽量保证升级后的版本要向下兼容正在流通的低版本的可用性；另一方面要对流通的版本进行收敛管理，支持多种在线更新机制。

基于基础平台运行。无论是基础设施平台，还是 PaaS 层的应用平台，或持续交付工具链，系统均需要尽量与公司现有基础平台对接，避免引入新的技术栈。

可配置而非硬编码。在应急时，发现有些“参数”硬编码在程序中，无法快速调整，需要推动配置管理。另外，IP 的 DNS 域名改造也是可配置的一个方向，减少人工在后台修改。

日志规范化。日志分析是认识应用系统的关键手段，SRE 可关注以下几点：一是“存储”，因为行业或企业内部有保存日志数据的要求，需要一个成本可控、可横向扩容、支持海量日志数据的管理平台；二是“查询”，日志是感知线上系统运行状况，了解代码级问题的一个窗口，在出现业务问题、故障应急等场景下，可以利用日志查询问题，需要建立实时的数据采集、高效的日志检索能力；三是“监控”，基于日志分析关键字、正则检索、模式匹配等方式，能够提供监控能力；四是“分析”，对日志的非结构化数据进行加工处理，生成结构化数据，用于运行数据分析，实现异常发现、故障定位、性能分析、容量评估等分析工作。

测试方案完备。在测试方案评审中，SRE 应重点围绕性能、容量、压力、稳定性架构等测试方案的评审。评审的角度包括，测试用例的描述是否清晰，测试用例的执行结果是否符合要求，测试结论涉及遗留问题的应对措施等。

3.4.3 变更保障准备工作评估

变更带来系统稳定性风险。从生产变更故障引发率看，来自变更的故障遥遥领先其他因素引发的故障，变更后通常是运维组织最为繁忙的时段。不管是大的软件基线，还是小到一个功能迭代，甚至一个参数或配置的调整，也可能引发一个重大故障。变更带来的风险点很多，有设计上的程序缺陷类的问题，也有管理上的版本管理的问题，也有操作层面的执行问题，也可能是协作层面上下游系统沟通不畅引发的相互影响的问题等，解决变更带来的风险问题是一个极为复杂的系统性工作。SRE 可考虑从以下工作中加强变更保障的评估。

监控和告警是否就绪。建立完善的监控覆盖面，包括基础设施、平台软件、数据库、中间件、操作系统、性能等技术指标监控，与特定系统相关的业务功能、客户体验指标监控，以及与系统上线后重要业务功能首笔出现的监控。

运行健康检查方案是否就绪。需建立完善的监控和预警体系是保障系统可运维性的重要手段，能够实时监控系统的各项指标，及时发现问题并进行预警，以便及时处理问题。

应急预案是否就绪。故障应急工具是否就绪：系统出现故障时，需要能够快速恢复并进行处理，避免故障扩大化。

技术运营工作是否就绪。系统上线后，需要进行全面的运营管理工作，包括用户服务、首笔业务发生的跟踪、异常数据

或逻辑问题的处理、安全管理等方面的内容，以确保系统能够稳定、高效地运行。

文档是否完善。根据组织软件交付的协同机制，明确新系统上线的文档交付清单，比如：项目与需求、技术架构图、测试结论（含压力测试）、安装部署（与环境相关）、应用/业务监控、首笔功能监控及保障、重要功能清单、重要配置与参数、运行效能评估等文档。

知识库是否就绪。建立完善的知识库是保证系统可运维性的重要基础，需要对系统的各项功能和操作进行详细的描述，并建立知识库，以便在需要时能够快速查找相关信息。

3.4.4 新系统或新业务上线保障评估

新系统或新业务上线是从 0 到 1 的过程，具体的工作通常包括：上线准备工作、制定技术方案、评估测试管控、上线文档准备、风险评估、安全保障措施、运维监控准备、上线过程协同、上线发布、系统验收、上线试运行等。

新系统或新业务上线给企业带来机会与挑战。一方面，作为项目重要里程碑，新系统将为企业业务发展或运营管理助力，通常业务需求方会投入大量精力在上线后的运营推广工作，以期望更好地给业务及客户带来价值；另一方面，新系统上线带来众多的不确定性因素，需要对不确性因素进行管理。

不确定性包括：新业务流程的合规性、数据安全、隐私安全等问题；新业务对现有上下游系统在业务及性能层面的影

响；新系统在设计上是否满足业务活动的性能、容量要求；配套的稳定性相关的监控、日志、应急、数据备份等非功能性需求是否就绪；功能已知缺陷的影响是否评估；发布上线、环境部署、下线方案是否就绪；对系统业务运行情况的观测能力等

新系统或新业务自身的业务风险可能会对存量业务流程产生影响。在加强风险评估工作上，SRE 应加强以下风险的评估：

新系统业务带来的风险防范、监管合规、数据安全、隐私安全等问题；

新系统业务流程可用性、终端体验、数据准确性等风险；

新系统业务对现有上下游系统在容量、性能方面的影响，以及上下游配合改造对原有业务带来的影响；

新系统资源、架构、重要功能是否满足业务期望，以及接下来业务活动的性能、容量要求；

系统压力测试、容量评估方案、功能遗留缺陷等是否评估到位等；

另外，SRE 还应主动推动以下工作：

标准化先行，建立业务与技术层面的合规、风险、隐私、安全等方面的管理要求，并辅助相关技术检测手段。

业务系统非功能性需求的建设，比如系统回退、版本切换、灰度发布、终端体验感知等配套功能的实现。

业务系统技术运营需求的建设，比如对首笔业务感知、业务流水监测、关联系统的技术运营监测等。

系统性能与容量管理，包括相关评估指标、基线容量设计、指标数据加工、容量评估分析报告、压力测试等工具建设。

同时，针对可能突增的业务模式，SRE 需要建立限流、削峰机制。架构优化上，需要前端系统做交易并发控制开关，必要时进行前端限流、削峰，以及后端服务降级，通过一些前端交互设计减少客户的体验影响，重点保障系统的核心服务。

SRE 通过聚焦新系统或新业务上线阶段，做好运维工作左移，提前做好资源交付能力建设提高系统上线速度，利用运维平台能力建设帮助业务研发专注业务逻辑，提前参与到架构及非功能性需求的研发与验收，能够让系统上线后融入平台化管理模式。

4 变更管理

根据业界经验及 Google 多本 SRE 书籍的论述，约 70% 左右故障是由变更引起的。然而，业务发展中的变更是不可避免的，因此，通过变更管理来控制变更的风险，尽可能降低由变更导致的故障率和影响范围，是提升系统稳定性的一条可持续且高 ROI 的路径，也是每个 SRE 团队面临的关键课题。

4.1 发布管理与变更管理关系阐述

发布是软件工程术语，变更是系统工程术语，在各自领域，这两个概念常常相互包含对方。SRE 关注的核心问题是业务应用的稳定性，SRE 所涉及的“发布”大多特指代码变更，而“变更”的范围则包含了硬件变更（网络设备、主机、存储等一切 IaaS 硬件）、系统软件变更（各类 OS 等）、应用软件变更（与 SRE 相关）、安全设施（机房配套设施及安全防护设备）等一切对 IT 系统调整的动作。因此本白皮书将“发布”定义为“变更”的一部分，特指版本发布。

发布管理是指整个软件开发周期中，软件从开发阶段转移到生产环境的过程。这个过程包括规划、调度、控制以及测试软件版本的构建、部署和交付。发布管理的重点是确保软件的新版本能够顺利、有效地发布，同时减少对现有系统运行的影响。在 SRE 领域，发布管理通常强调自动化部署流程，以减少人为错误，提高发布速度和可靠性。

变更管理则更加宽泛，涵盖了所有对生产环境可能造成影响的活动，包括软件更新、运行环境、配置修改等。变更管理的目标是确保所有变更都是经过计划、测试和批准的，从而最小化任何负面影响，确保系统的稳定性和可靠性。变更管理流程通常包括变更的申请、评估、批准和监控。

关于两者的关系：

发布管理是变更管理的一个子集：在 SRE 框架下，所有的软件发布都应视为变更，因此需要遵循变更管理的流程和原则。这意味着每一个软件发布都需要经过严格的审查和批准过程，以确保它不会对生产环境造成不可预见的负面影响。

相互支持的目标：尽管两者的焦点不同，但发布管理和变更管理的最终目标都是提高服务的稳定性和用户满意度。通过精细的发布管理，可以有效地推出新功能和修复，同时通过严格的变更管理可以确保这些发布不会导致服务中断或性能下降。

共享的实践和工具：在实际操作中，SRE 团队会使用各种工具和实践来支持这两个过程，例如版本控制系统、自动化测试、持续集成和持续部署（CI/CD），以及监控和警报系统。这些工具和实践不仅支持快速安全的代码发布，也帮助团队管理复杂的变更。

发布（Release），指将经过测试的、验证正确的软件版本（包括但不限于客户端、服务端程序，相关配置文件、数据等）导入到目的变更地点的行为。通过规范有效的发布管理流程，可以使发布实施更快，成本更优，风险更小，保障业务的稳定性和可用性。

发布管理（Release Management）的目标是确保所有线上生产环境的版本发布行为具备可控性和可追溯性，具体包括但不限于以下几个方面：

（1）合理定义发布计划，以确保对业务影响最小化，并降低风险。

（2）确保发布到生产环境的版本统一，并经过必要的版本质量保证过程。

（3）确保包含安装、测试、验证和回退等发布过程中的各个环节操作的规范性。

（4）确保发布相关的利益干系人（如SRE/研发/测试/产品/运营团队和用户等）的行动一致性。

在国内业界的实践中，不同职责的团队所肩负的变更任务会不一样，例如，基础架构部门，云平台部门SRE更多的是进行基础架构层面的变更管理，而业务SRE更多负责发布管理，所以以下的内容侧重点，差异性也会因此而出现。

4.2 变更体系设计

4.2.1 变更体系设计原则

SRE关注的核心是业务应用系统的稳定性，SRE描述任何问题都是从软件工程的视角出发，围绕业务系统做出某方面调整改造，或者开发了某些管理系统。Google SRE针对变

更管理强调自动化操作的技术实践，包括渐进式发布、过程和结果的反馈/检测、回退等。

4.1 中明确了在 SRE 视角下变更包含发布，发布特指代码变更，以及发布与变更在工程层面应该共享实践及工具，因此 SRE 变更体系设计的原则包括两点：严谨的变更流程以及通用的工程化体系。

4.2.2 变更及发布流程设计

4.2.2.1 变更发布准备

由于变更/发布涉及到产品、开发、测试和运营等多个职能团队的参与，因此在正式实施操作之前，必须进行一系列充分的事前准备工作。作为线上环境的第一责任人，SRE 在整个发布过程中担负着“总协调”的角色。在发布前，SRE 需要进行以下关键任务：

1. 审核变更发布产品功能和服务的可靠性。
2. 评估变更发布的风险，并制定相应的应对策略。
3. 选择合适的变更发布方式和方案，以最大程度降低影响。
4. 协调各职能团队完成相应的准备工作，确保协同合作。
5. 主导制定具有可操作性的发布流程清单，确保流程的顺畅执行。

6. 预先制定针对各种可能的变更发布异常的快速执行应急预案，以确保对突发情况的迅速响应。

（1）变更风险评估

变更风险评估是指在进行软件发布或系统变更前，对可能存在的风险进行评估和分析，以便在变更过程中及时采取措施来降低风险，确保变更的稳定性和可靠性。在业务全生命周期中，产品希望在用户侧得到快速验证和反馈，这就要求每次的功能迭代都能快速发布到线上环境，而 SRE 围绕业务的性能和稳定性，更加关注线上环境的持续稳定运行，变更带来改变，改变带来风险，变更与稳定在某种程度上存在冲突，因此，针对线上环境的每一次变更/发布（包括常规发布和紧急发布），引入风险评估环节变得尤为重要。由于变更概念较为宽泛，为使读者深入理解后续环节，本章节后续内容将以变更中最为重要的版本发布（代码变更）为例，介绍风险评估之后的环节。

1. 版本发布容量风险评估

通常情况下，新功能的发布往往会带来临时用户访问量的急剧增长。因此，SRE 在执行发布之前需要做好容量风险的评估工作。为了做到这一点，一方面可以依据发布前压力测试的数据，结合运营数据模型（包括发布过程中和之后的流量及增速预测），提前准备足够的发布容量冗余，并尽量避开在业

务高峰期进行发布；另一方面，可以选择将首次发布限制在某一单独区域的用户，通过该区域的容量增长数据的分析，建立后续大规模发布时的信心和预期。

2. 版本发布依赖风险评估

发布依赖风险评估主要包括基础设施、程序引用、下游依赖以及上游调用几个关键方面。在基础设施方面，涵盖计算、存储、网络等要素，需要确保相关基础设施负责人积极参与发布过程，进行全面的风险评估和制定相应预案。在程序引用方面，包含第三方类库、代码、数据等，必须避免偶发性故障、程序 Bug、系统错误或安全问题对发布正确执行造成影响。对于下游依赖，应设定合理的超时时间，提前有效预估和沟通下游流量，防止对下游系统造成过度负担，确保强依赖变成弱依赖，同时高优先级应用不得依赖低优先级应用。至于上游调用，需要明确各调用方的身份，根据不同优先级制定合适的分配、限流和熔断策略，以确保整体系统的稳定性和可用性。

3. 版本发布异常风险评估

并非每次发布都能按计划如期完成，作为 SRE，我们需在发布前进行细致的异常风险评估，主要涉及发布延时、发布故障以及发布回退（包括程序、配置、数据回退）这三个方面，以应对潜在的不确定性。发布延时指的是由于停机发布实际所造成的停机时长超出原计划时长，或者由于热更新不停机发布新功能用户无法使用，从而对业务可用性产生不利影响。发布

故障指的是在发布过程中人为操作不当或不可预知原因而引发的故障，例如文件、配置发布出错，主机、网络设备故障等。至于发布回退，指的是在发布过程中发生无法正常推进的情况，导致线上环境的用户数据受到污染，必须将业务的程序、配置和数据回退到发布前的某一时刻。为确保系统的稳定性，我们必须在发布前充分评估这些异常风险，以准备好应对可能的挑战。

4. 其他风险

在新功能上线初期，面临着业务竞争、无法控制的用户行为等因素引发的诸多风险，其中包括大量的 DDoS 攻击、客户端滥用行为等。这种情况可能导致服务过载，甚至新功能的崩溃等问题。因此，SRE 在新功能发布上线之前需要通过有效的压力测试来规划业务应对这些挑战。最佳的做法是为新服务设计具备“功能开关”的机制，并灵活运用灰度和阶段性发布的方法。

通过详尽的压力测试，可以模拟和评估在真实环境中可能发生的各种情景，从而有效预测系统在高负载和攻击情况下的表现。同时，引入“功能开关”机制使得在遇到问题时可以迅速关闭或切换功能，以减轻系统负担。此外，采用灰度发布和阶段性发布的策略，可以逐步将新功能引入生产环境，降低潜在问题对整个系统的冲击，使问题的范围得到有效控制。

通过这些规划和策略，SRE 能够更好地保障新功能的上线稳定性，降低因各类风险而导致的服务中断和性能问题的风险。

效果评估：

SRE 理念倡导拥抱风险，对于发布服务来讲，不可能做到 0 风险，发布的可靠性并不是一味地追求越高越好，因为极高的可靠性需要付出巨大的成本，与实际的回报不成正比。在错误预算（Error Budget）的允许范围内，根据当次发布服务的具体情况，评估风险容忍度，并预留处理风险突发情况的时间。此外，在尽可能减少计划外停机时间（unplanned downtime）的基础上，寻找质量、效率、成本和安全之间的平衡点，是一种有效的发布风险评估方法。

（2）发布方式确定

发布方式是软件开发和维护过程中采用的一系列策略和方法，旨在确保应用程序的性能、可靠性和可用性。SRE（Site Reliability Engineering）发布方式的核心目标是最小化应用程序停机时间和风险，并确保新版本应用程序经过充分测试和验证，具备良好的性能和可靠性。

典型的 SRE 发布方式涵盖蓝绿部署、金丝雀发布、滚动发布和灰度发布。此外，在发布过程中，还存在着多种具体的

部署方式，如整包部署、增量部署和容器化部署，它们主要区别在于所采用的技术实现方式。

零风险发布是SRE 不断追求和实践的目标，为此有多种发布方式可供选择，以满足不同的发布需求。在选择SRE 发布方式时，主要考虑以下几个方面：

1. 基于应用程序的复杂性

在面对高复杂度的应用程序时，可能需要采用蓝绿部署或滚动发布等发布策略，以确保整体应用程序的连续可用性。蓝绿部署允许同时运行两个版本的应用程序，其中一个版本是当前稳定版本，而另一个版本则是待验证的新版本。经过充分的测试和验证后，可以逐步将流量切换至新版本，直至完全替代旧版本。相较之下，滚动发布则以逐步更新应用程序不同部分的方式来保障整个应用程序的可用性。

2. 基于可靠性需求

如果应用程序对可靠性要求很高，可能需要使用蓝绿部署、金丝雀发布或灰度发布等方式，以确保新版本应用程序的性能和可靠性得到充分测试和验证。金丝雀发布可以逐步增加新版本应用程序的流量，以测试其性能和可靠性。如果出现问题，可以立即回滚到旧版本。灰度发布可以将新版本应用程序的流量引导到一小部分用户，以测试其性能和可靠性。如果出现问题，可以立即回滚到旧版本。

当应用程序对可靠性的要求极高时，常常需要考虑采用蓝绿部署、金丝雀发布或灰度发布等发布策略，以确保新版本应用程序的性能和可靠性是经过充分测试和验证的。金丝雀发布允许逐步增加新版本应用程序的流量，从而评估其性能和可靠性。在发现问题的情况下，能够迅速回滚至旧版本，确保系统稳定性。另一方面，灰度发布则能够将新版本应用程序的流量引导至一小部分用户，以进行性能和可靠性测试。同样，在发现问题时，也能够立即回滚至旧版本，以最大程度地降低潜在影响。

3. 基于 SRE 团队能力和经验

在团队经验不足或缺乏自动化工具支持的情况下，推荐选择更为简单的发布方式，例如滚动发布。滚动发布以逐步更新应用程序不同部分的方式确保整个应用程序持续可用。相较而言，当团队拥有丰富经验和自动化工具支持时，可以考虑采用更为复杂的发布策略，如蓝绿部署、金丝雀发布或灰度发布。这些高级发布方式能够更精细地控制新版本的推出，从而最大程度地保障系统的可用性和稳定性。

4. 基于时间压力

在面临时间压力时，选择合适的发布方式至关重要，以确保应用程序更新的及时性不以牺牲质量为代价。滚动发布和灰度发布是在此类情况下推荐的策略，因为它们能够提供快速且连续的部署能力。

滚动发布允许按计划逐步替换应用程序的旧版本，这种逐步更新的方法有助于维持应用程序的持续可用性，同时允许在必要时快速回滚。

灰度发布则通过将新版本引入给一小部分用户，以实现对新版本性能和可靠性的快速测试。这不仅加快了反馈获取的过程，还降低了潜在风险，因为任何问题都可以在影响扩大之前迅速被识别和解决。

选择这些策略有助于团队在紧迫的时间框架内有效地推进项目，同时通过及时的反馈和验证确保新版本的性能和可靠性。这种灵活性在处理紧急部署或快速迭代时尤为重要，确保了即使在时间压力下，也能保持应用程序的稳定性和用户满意度。

效果评估：

SRE 发布方式的评估标准主要涵盖故障率、可重复性、回滚能力、时间效率以及用户满意度等多个方面。通过对这些关键指标的综合考量，我们能够全面评估不同发布方式的优劣，并选择最为适宜的发布策略来有效地管理应用程序的更新和维护。

这一系统化的评估方法有助于深入理解各种发布方式在实践中的表现，从而为决策者提供明智的选择。通过对 SRE 发布方式的精准度量，我们能够更加精细地调整发布流程，以满

足应用程序更新的需求，同时最大程度地确保系统的稳定性和性能。

（3）发布协调

发布协调是SRE 团队在软件发布全过程中协调各个团队、统筹推进各发布环节，以确保发布过程的顺利进行。每次业务上线发布都牵涉到产品、开发、测试、运营、市场等多个职能团队，其成功依赖于这些团队的紧密合作。由于SRE 团队在日常工作中积累了丰富的产品知识、技术架构能力，并建立了良好的跨团队关系，使其成为最适宜协调各团队的角色。在发布前、发布中和发布后，SRE 以全局视角统筹整个流程，引导团队构建可靠、可扩展、稳定且性能卓越的产品。

（1）制定详细的发布计划，与开发团队和其他相关团队协商，明确发布时间、版本号等信息。借助功能发布上线日期，逆向规划代码提交截止时间、测试打包、发布前准备和发布停机时长等排期，并与所有相关团队成员确认方案和排期，输出发布电子流和工单。

（2）审核发布产品的新功能和内部服务，确保它们与原有业务的可靠性标准一致，并提供提升可靠性的具体建议。对于发布的业务模块，通过SRE 技术手段检测和预防可能增加停机时间或妨碍性能目标的单点故障，制定高可用性和快速灾难恢复方案。

（3）在发布过程中作为多个团队之间的联系人，考虑当前发布的影响范围。可以迅速组建一个虚拟团队，联系相关职能人员，包括但不限于产品、运营、开发、测试人员，通过线上线下会议的沟通方式，持续检查发布版本所涉及的各团队工作进展。

（4）跟进发布所需任务的进度，负责处理发布过程中的所有技术相关问题，包括但不限于构建发布自服务模型的平台工程，提升自动化程度，解决由资源不足引发的容量问题，以及通过服务降级处理过载的技术预案等。

（5）作为发布全流程的“守门人”，决定所有发布项是否都是“安全的”，确保发布过程进入可控状态，降低发布失败率，缩短发布时间周期。

效果评估：

SRE 团队通过与其他团队（如开发、测试团队等）的协调合作，以确保发布过程的顺利高效，并最大程度地减少对用户的影响。对其效果进行评估可从以下几个方面进行衡量：

发布效率：通过协调发布流程，降低了团队间的沟通成本，缩短了相互等待的时间，提升了发布效率。

发布稳定性：SRE 团队与其他团队协作，始终将业务稳定性视为首要衡量指标，确保发布版本在生产环境中稳定运行，减少故障和停机时间。

发布质量：通过协调发布流程，SRE 团队能够保证发布版本的质量，涵盖代码、文档和测试等多个方面的质量保证。

用户满意度：通过协调发布工作，最大程度地减少对用户的负面影响，包括减少停机时间、降低用户投诉数量，提升用户体验，从而提高用户的满意度水平。

（4）发布执行清单

发布执行清单是指在正式操作之前由业务 SRE 团队制定的详细清单，其中包含了软件或系统发布的所有步骤和操作。其主要目的在于确保发布操作流程的顺利和成功。该清单通常由业务 SRE 团队编写，并在发布过程中按照既定的操作方案进行实施。完善且准确的发布执行清单能够有效保障发布过程中避免由于人为操作导致的发布步骤缺失或执行不准确而引发的系统中断。

（1）发布执行清单包含但不限于以下要素：

a. 各步骤的具体执行内容及预估耗时。详细的执行命令能够降低发布过程中人员操作失误的可能性，而预估耗时则有助于 SRE 发现执行步骤是否存在异常，并评估发布总耗时。

b. 执行成功确认手段。清晰地执行成功确认手段有助于在发布过程中尽早发现系统中断，并增强发布过程中的信心，例如，通过观测请求量指标是否恢复到 7 天前的环比水平。

c. 操作回滚步骤。清单必须包括回滚步骤，以便应对发布过程中可能出现的问题和故障。

d. 测试与验证步骤，至少要包含新功能的测试与验证，以及用户关键链路的基础功能的测试。确保该次发布新增加的功能和关键链路的验证符合预期

e. 发布操作人员协同，对于跨团队或多实施人员的发布，需要相关的实施人员要做到知会和协同，尤其是对于有前后顺序的步骤，需要前置步骤实施完毕并且结果符合预期后才能进行后置步骤的实施。

（2）发布执行清单按执行顺序分为发布前、发布中、发布后三个阶段。发布前的检查清单应关注于检测评估发布前置环境是否符合发布条件，发布中的执行检查清单则包括具体的发布变更操作，发布后的执行检查清单主要用于验证此次发布是否成功。

（3）利用工具化模板将发布流程清单线上录入，通过可视化的图形界面进行任务流程编排和执行。采用插件对接与开发的方式，实现与发布操作所需工具和平台的调用，从而实现发布流程清单的跨系统调度自动化。这样可以减少发布过程中的人为切换等待时间，并降低人工操作的失误。

（4）发布清单应在团队中共享，以避免人员单点问题。在发布过程中，应当在发布清单中记录发布执行起止时间、实施

内容、实施人员等关键信息，作为后续发布流程回顾及优化的依据。

效果评估：

SRE 发布执行清单是一个旨在规范发布过程的详细清单，其目标在于确保发布过程的高效性和流畅性，以最小化故障和停机时间。对 SRE 发布清单的效果评估可以从以下几个方面考虑：

发布速度：SRE 发布清单提供了一个标准化的发布流程，有助于团队更迅速地将新功能和问题修复版本发布到生产环境中，从而显著提高整体发布效率。

发布稳定性：SRE 发布清单中涵盖了各种验证步骤，可在发布前进行充分的测试和验证，以确保发布的版本在生产环境中运行稳定。这一方法大幅减少了故障和停机时间，从而增强了系统的整体稳定性。

发布质量：SRE 发布清单中包括回归测试、性能测试等步骤，确保发布版本的高质量。通过这些措施，不仅提高了系统的整体质量，还提升了用户的满意度。

（5）应急预案制定

发布应急预案是一项用于指导发布过程中处理紧急情况和发布事故的战略性计划。通常由专业的业务 SRE 团队编写并在发布过程中备用，旨在确保发布的系统在任何情况下都能保

持安全和稳定。应急预案在发布工程中扮演着业务连续性管理的关键角色。即便在充分准备和严格执行的情况下，每次发布变更仍然存在不可预知或无法评估的风险，SRE 通常通过应急预案来应对这些挑战。在系统非预期中断的情况下，SRE 可能面临巨大的压力，并且在短时间内难以全面了解业务系统的上下游状态。这种情况下，临时决策的恢复方案可能因未全面考虑而引入新问题。因此，为避免引发更严重的问题，SRE 有必要在发布前充分准备应急预案。

制定发布应急预案的整体原则应遵循以下几个步骤：

（1）明确应急预案范围和目标：详细列出此次发布应急预案的覆盖范围和应对目标，包括需要关注的模块、业务指标、观测手段、预案启动条件、实施方案以及实施后的预期表现。这样有助于协助 SRE 在发布过程中做出准确的判断和应急操作。

（2）发布过程和完成后的监测：在发布过程和完成后，SRE 应通过监测指标来判断是否发生了非预期状况，并确定是否需要启动应急预案，以提前发现系统中断。

（3）与发布协调步骤的联动：当出现超出预期的情况时，应快速组建故障团队，制定团队人员清单和分工计划，以缩短系统中断的恢复时间。

SRE 发布的应急预案制定步骤如下：

（1）确定应急预案的范围和目标：明确此次发布应急预案的覆盖范围和应对目标，包括哪些紧急情况需要应对，以及应急预案的主要目标是什么。同时，对可能发生的紧急情况进行分类和分级，制定相应的应急响应计划，包括行动步骤、通信流程、资源调配等。

（2）组建应急响应团队的成员和职责：确定应急响应团队的成员和职责，包括开发团队、测试团队、运维团队等。建立紧急联系方式，包括电话、邮件、即时通讯等，以确保在紧急情况下能够及时联系。同时，需要定期组织团队审查和更新应急预案，以保持其有效性和适应性。

（3）制定详细的应急预案内容：包含但不限于：

a. 预发布测试：在正式发布之前进行预发布测试，确保版本的稳定性和兼容性。

b. 限流措施预案：在版本发布期间采取限流措施，以避免系统过载和崩溃。

c. 监控和报警预案：加强监控和报警机制，及时检测和响应系统异常和故障。

d. 回滚计划预案：如果版本发布出现问题，立即启动回滚计划，将系统恢复到之前的稳定状态。

（4）应急预案演练：根据应急预案制定应急演练计划，并进行演练，评估应急预案的有效性和实用性。

效果评估：

效果评估方面，可以从以下几个方面来衡量：

（1）有效性：应急预案是否能够有效地应对发布事故，包括是否能够快速响应、准确识别问题、及时采取措施、有效恢复等。

（2）可操作性：应急预案是否易于操作和使用，包括是否能够提供清晰的指导和说明、是否能够让发布操作人员快速掌握和操作、是否能够减少人为错误等。

（3）可维护性：应急预案是否易于维护和更新，包括是否能够及时更新和修订、是否能够保持最新的技术和流程、是否能够在下一次的发布中继续复用等。

4.2.2.2 版本发布实施

（1）发布前确认

发布准备是指在正式进行发布操作之前所需要操作的一系列步骤，以确保发布流程的顺利和成功进行。在执行正式发布之前，SRE（Site Reliability Engineering）工程师需执行如下工作，但不限于：环境备份、制品预分发、告警屏蔽以及各项准备工作的最后确认。

（1）发布流程确认。需要在发布前与相关干系人对齐整体发布内容，需要明确各项发布的时间及执行内容。

（2）风险评估。需要对发布内容进行全面评估，以确定发布对现有系统的安全性、机器性能、环境依赖等是否会产生影响，通过风险评估可以避免在变更过程中出现无法回滚、安全漏洞等问题。

（3）环境备份。在发布前，必须对系统的数据、配置以及业务程序进行备份，以确保在出现问题时能够快速恢复。数据备份范围包括但不限于业务数据库、文件系统等，而配置备份则包含业务配置文件、证书文件等。选择适当的备份方式（手动或自动备份），以确保备份的及时性和完整性。

（4）制品预分发。制品预分发是指在发布前将需要更新的文件提前分发到目标服务器，以便在发布时能够快速更新。这一步骤有效缩短了发布时间，降低了发布过程中出现问题的风险。完成预分发后，需要进行验证，确保预分发的制品能够正确地更新到目标服务器上。

（5）监控告警屏蔽。为防止发布停机时引发不必要的告警，SRE 在前置工作中需要屏蔽部分监控告警指标，以防止告警风暴的发生。设置合理的屏蔽时长，并记录屏蔽监控指标的详细信息，包括指标名称、屏蔽时间范围、屏蔽原因等。同时，需要避免屏蔽敏感指标或对系统安全性产生负面影响，以确保非发布模块的稳定性和安全性。

（6）各项准备工作的最后确认。作为发布前准备工作的把关人，业务 SRE 需确认测试是否充分、发布环境是否准备就

绪、备份和恢复是否有效、风险评估和应急预案是否充足、通知是否及时准确、发布执行清单是否完整清晰等相关事项。

（7）效果评估。发布前置工作的效果可从以下几个方面考虑：

发布成功率：衡量发布前置工作效果的重要指标之一是发布成功率。较高的发布成功率表示前置工作效果较好。

发布耗时：另一个衡量效果的指标是发布时间。若前置工作充分，发布时间应缩短，提高发布效率。

问题处理速度：若出现问题，评估前置工作效果的一个指标是问题处理速度。充分的前置工作应该能够加速问题解决，减少系统故障时间。

用户反馈：用户反馈是评估前置工作效果的另一个关键指标。不良的用户反馈表明前置工作未达到预期效果，需要进一步改进。

（2）发布执行

发布执行是SRE 团队负责执行一系列发布操作，包括停服、更新、起服等，以将新功能成功发布到线上正式环境并对用户开放。为确保每次业务新特性稳定上线，SRE 团队需制定可靠的发布执行规范和发布意外防范机制，以确保各类对象（如二进制程序、配置文件、数据库及依赖环境）以可重现、自动化的方式发布到业务生产环境。

发布执行的主要步骤包括：

部署：执行部署脚本，将软件部署到指定目录。

测试：在部署完成后进行功能、性能和安全等测试，以确保新版本正常运行。

监控：通过监控告警手段监测发布后运行状态，及时发现并解决问题，保障系统稳定性和可靠性。

回滚：如果出现问题，按照已制定的应急预案和操作流程，及时回滚到上一个版本。

发布执行过程中还应包括以下内容，包括但不限于：

自服务模型：SRE 发布工程师通过开发工具、制定最佳实践，使整个发布执行过程自动化，不再需要 SRE 工程师干预。研发团队或其他第三方人员可以自主掌握和执行发布操作，构建发布执行的自服务模型。

发布操作的简单化：用户可见的功能应尽快发布上线，以减少每个版本之间的变更。发布执行时应按小批次进行，易于理解每次发布对系统的影响。通过逐步改变系统，同时考虑每次变更对系统的改善和退化，寻找最佳方案。

发布操作的标准化：统一运维操作入口，降低复杂度，提高可管理性。避免手工误操作，防范悲剧发生，将原本复杂易错的手工操作实现标准化运维，使运维操作更加可靠。同时，减少由运维人员手工误操作所带来的业务风险。

发布执行的稳定性与灵活性：平衡稳定性和灵活性，通过构建通用发布流程、实践和工具，提高发布执行的可靠性。最小化发布执行流程和工具对开发人员灵活性的影响。

最小化意外复杂度：SRE 团队需不断努力消除发布执行过程中的必要复杂度，发现并提出抗议，以减少引入的意外复杂度。

效果评估：

对发布执行效果进行准确性、成功率和效率三个方面的评估：

（1）准确性：操作是否按照计划和目标进行，是否准确无误地完成了发布操作。

（2）成功率：发布执行的成功率是否高，是否达到预期目标，是否影响了业务的稳定性和可用性。

（3）效率：发布执行的效率是否高，是否在规定时间或提前完成任务。

（3）发布验证

SRE 发布验证是指在软件发布后对已部署的软件进行验证和测试，以确保软件能够正常运行，并满足预期的功能和性能要求。发布验证贯穿于SRE 的整个发布过程中，而不是仅仅存在于部署后。根据不同的发布计划，存在不同的验证流程，其中包括但不限于以下几个方面：

1. 稳定性验证

稳定性指系统要素在外界影响下表现出的某种稳定状态。对于应用程序而言，稳定性是保障系统能够满足SLA（服务水平协议）所要求的服务等级协议的关键。在应用程序发布过程中，应持续关注关键的：基础指标、应用指标及业务指标。

基础指标：所依赖的系统和基础设施层面的指标

例如：系统负载、内存容量、网卡流量

应用指标：能够反馈业务依赖的接口服务是否可用的指标

例如：请求速率、请求时延、请求成功率

业务指标：能直观地反馈业务或者系统功能是否可用的指标

例如：在线人数、订单量、评论量

确保应用程序能够以一个安全的状态持续运行。如果观察到系统整体处于非稳定状态，应及时采取相应的应急措施，以将发布的影响降到最小。

2. 确定性验证

应用程序的发布通常旨在修复缺陷或提供新特性，因此发布的结果应该是确定的。在发布过程中，对应用程序进行持续观察，只有在一个或多个周期内应用程序的运行状态符合预期，功能性或缺陷性的改动能够生效时，才能将发布视为有效。一旦发现应用程序在发布过程中出现了非预期的情况，应及时记录并评估是否需要继续发布，以规避风险。

3. 依赖性验证

依赖性指系统中不同组件之间的相互依赖关系。一个复杂的系统通常由多个组件组成，系统中某个组件的变动可能对整体系统产生影响。在发布过程中，应关注与发布程序存在依赖关系的其他组件的运行情况，以确保应用程序的改动不会对其他组件造成预期外的影响。最好在发布前对依赖组件的运行情况进行评估，尤其是对性能或时延要求较高的组件，以避免对业务造成雪崩效应。

4. 效果评估

SRE 发布验证的效果可以从以下几个方面进行评估：

发布验证的平均时间：衡量发布验证的平均时间，即发布验证开始到完成的时间。

发布后的稳定性：衡量发布后系统的稳定性和可靠性。

发布后的性能：衡量发布后系统的性能，包括响应时间、吞吐量、并发量等。

发布后的用户满意度：衡量发布后用户的满意度，包括用户的反馈和投诉等。

通过对这些标准的综合评估，可以优化发布验证的流程和方法，提高系统的稳定性和可靠性。

（4）应急预案实施

SRE 发布应急预案实施是指在软件系统发布过程中，对于突发紧急情况，SRE 团队采取一系列紧急措施以确保发布过程的稳定性和安全性。以下是 SRE 发布应急预案实施的详细步骤：

（1）紧急通知：在出现紧急情况时，SRE 团队应立即通知相关人员，包括但不限于开发团队、测试团队和运维团队，以便共同制定应急方案。

（2）应急方案选择：SRE 团队根据事先制定的应急方案，选择最适合情况的解决方案。这包括确定问题的性质和影响、采取的紧急措施、修复时间等关键因素。

（3）紧急回滚：在紧急情况下，如果发布过程出现问题，SRE 团队应立即执行回滚操作，将系统还原到之前的稳定版本，以最小化对用户的影响。

（4）问题排查和解决：SRE 团队应立即进行问题排查，采取必要的措施来解决问题，以确保发布过程的稳定性和安全性。

（5）事后总结：在问题解决后，SRE 团队应进行全面的事后总结。这包括对应急过程的分析，问题原因的详细剖析，以及提出改进措施，以便在下次发布中更有效地执行。

通过SRE 发布应急预案实施，团队能够迅速、有效地应对紧急情况，保障发布过程的稳定性和安全性，最终提升用户满意度。

效果评估：

发布应急预案实施的效果可以从以下几个方面进行评估：

（1）预案执行情况： 应急预案是否能够按照预定计划执行，是否能够应对突发情况做出及时调整。

（2）故障恢复时间：应急预案是否能够快速、有效地解决故障，缩短故障恢复时间，减少业务影响。

（3）用户投诉数量。应急预案实施后，用户投诉数量是否下降，是否能够有效地保障用户的服务体验。

（4）实施效益。应急预案是否在合理的成本下，能够限制发布故障的影响范围，最大限度地减少业务损失。

通过对这些方面的全面评估，团队可以更好地了解SRE 发布应急预案实施的实际效果，为未来的发布过程提供有针对性的改进建议。

4.2.2.3 发布总结

SRE 发布总结是指在软件系统发布后，SRE 团队进行的一系列总结和评估活动，旨在深入挖掘问题、总结宝贵经验，并提炼出切实可行的改进措施，以优化未来的发布过程。SRE 发布总结通常包括以下详尽步骤：

（1）数据收集：收集涵盖发布过程各个方面的数据，其中包括但不限于发布时间、故障和停机时间、用户反馈等多维度信息。

（2）数据分析：对所收集的数据进行深入分析，以明确识别发布过程中存在的问题和瓶颈，确保全面了解系统性能和用户体验。

（3）经验总结：精炼并深度总结发布过程中所获得的经验和教训，包括成功实践的要点以及需要改进的方面，为未来的发布活动提供宝贵指导。

（4）改进措施提出：根据对经验和教训的深入总结，明确提出一系列切实可行的改进措施，以不断提高发布过程的质量和效率。

（5）改进实施：将提炼出的改进措施有机地融入到发布计划中，并在下一次发布活动中有序实施，以确保团队在实践中持续演进。

通过SRE 发布总结，团队得以持续优化发布流程，提升发布效能和质量，有力保障业务的稳定性和可用性，最终为提高用户满意度奠定坚实基础。

4.2.3 变更的工程体系设计

为了减少因变更引发的故障，SRE 团队应通过软件工程体系设计一套通用的，可持续性的变更系统。

4.2.3.1 面向变更管理的 ITSM 设计

ITSM 全称（Information Technology Service Management），面向变更管理的 ITSM 工单设计要以“变更流程管理，变更风险评估，变更窗口管理，变更自动化，变更知识库”作为核心的功能，从而确保更严谨、有效地实施变更操作。

(1)变更流程管理

ITSM 系统需提供变更流程管理的基础能力，以承载变更管理流程规范落地，其中包括：

- 变更定义：支持变更流程负责人对变更管理的业务表单和工作流进行自定义。
- 变更申请：支持变更申请人填写变更工单需求并提交变更申请。
- 变更审核：支持变更复核人对变更内容进行复核及修订。
- 变更审批：支持对变更申请进行审批，包括单人审批、多人会签审批、多级审批等。
- 变更实施：支持对变更实施的结果进行记录。

(2)变更风险评估

对变更的风险进行评估是变更管理中的重要一环，不同的变更风险级别会对应不同的变更管理策略。因此，ITSM 工具

应该能辅助提升变更风险识别的准确度，减少因风险误判而引发的故障，包括：

- 变更风险级别定义：支持风险级别自定义。
- 变更风险评估规则：支持风险影响因子和风险计算规则的定义。
- 变更影响面分析：支持与 CMDB 进行联动，获取变更直接影响或间接影响的资源对象，以分析本次变更可能影响的关联资源。
- 变更资源保障分析：支持与存储资源系统，云资源系统等进行联动，以分析本次变更所需要的基本的硬件资源和人力资源。
- 变更冲突检测：支持与人员工作日程联动，提供可视化的变更事件“日历日程”，结合变更日程分析可能存在的冲突。

(3)变更窗口管理

变更窗口的定义是变更风险管控的重要手段，在影响较小的时间段实施变更，即使变更过程中出现异常也能将损失降低到最小。ITSM 工具需提供变更窗口维护和合规性检测的能力，包括：

变更窗口维护：支持对可执行变更的时间段进行定义，以供变更申请时进行选择。

变更窗口合规性检测：支持将变更实施的时间计划与变更窗口的定义进行联动校验，以确保不会出现实际变更实施在变更时间窗口之外的情况。

(4)变更自动化

企业业务的敏捷、研发的敏捷都会导致运维变更的频率增加，仅靠人工已难以满足变更效率的要求，不管是基础架构的变更还是应用版本的发布变更，都已经在寻求更高效的自动化执行工具。因此，ITSM 的变更管理能够与自动化工具便捷的集成也相当重要，复杂变更逐步抽象为多个低风险的标准变更，并将标准变更实现端到端的自动化。为实现与自动化的联动，ITSM 的设计应具备：

1) 第三方集成能力：支持如 API、脚本等系统集成方式，以获取自动化执行前所需的基础数据，或者合适的时机调用自动化动作。

2) 自动化流程节点：ITSM 的流程编排需支持自动化节点，以便于管理流和工程流之间衔接的编排。

(5)变更知识库

知识库可以将过去的变更经验和知识进行整理和归类，在面临类似的变更需求时，可以用来学习培训，提供辅助决策，从而降低错误风险，减少重复工作，提高工作效率。

知识库一定要进行分类整理，变更知识库要把“变更类型、变更原因、变更范围”作为主要的分类维度，再配合“关键字检索”的功能，就可以让变更知识库发挥重要的价值。

- 变更类型：硬件变更、软件变更、配置变更、流程变更等
- 变更原因：故障修复、性能优化、功能增强、合规需求等
- 变更范围：单个系统内、多个系统等

4.2.3.2 面向变更管理的 CMDB 设计

CMDB（Configuration Management Database，配置管理数据库）是一个存储 IT 基础设施中所有配置项（CI，Configuration Item）信息的数据库，包括硬件、软件、网络设备、服务等。

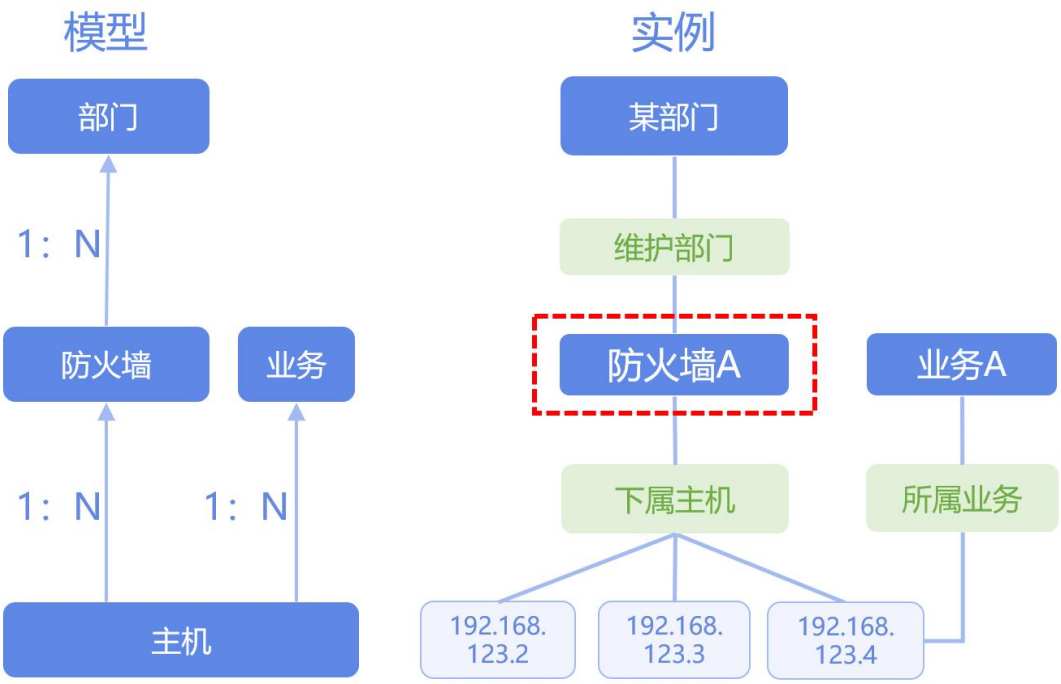
CMDB 不仅存储配置项的属性信息，还存储配置项之间的关系。在变更实施过程中，CMDB 可以通过“配置模型、配置实例、场景联动”等能力，识别准确的“变更信息”，支持这些流程的运转、发挥配置信息的价值，帮助实施人员执行变更任务。

(1)配置模型定义与管理

1）模型管理：根据元数据之间的关联关系，将变更操作常用的一组数据定义为一个模型，比如主机模型，包括主机的

名称，操作系统，CPU，内存，磁盘，MAC 地址，CPU 架构，内网 IP，外网 IP，IPv4，IPv6，云主机实例 ID，云主机状态，创建时间，创建人，固资编号，设备 SN，主要维护人，备份维护人，所在省份，所属运营商，质保年限，SLA 级别，寻址方式等。按照这个方式，根据变更场景定义“业务模型，项目模型，交换机模型，负载均衡模型，防火墙模型”等。需要提供一个把这些元数据建立连接的“模型管理”的功能。

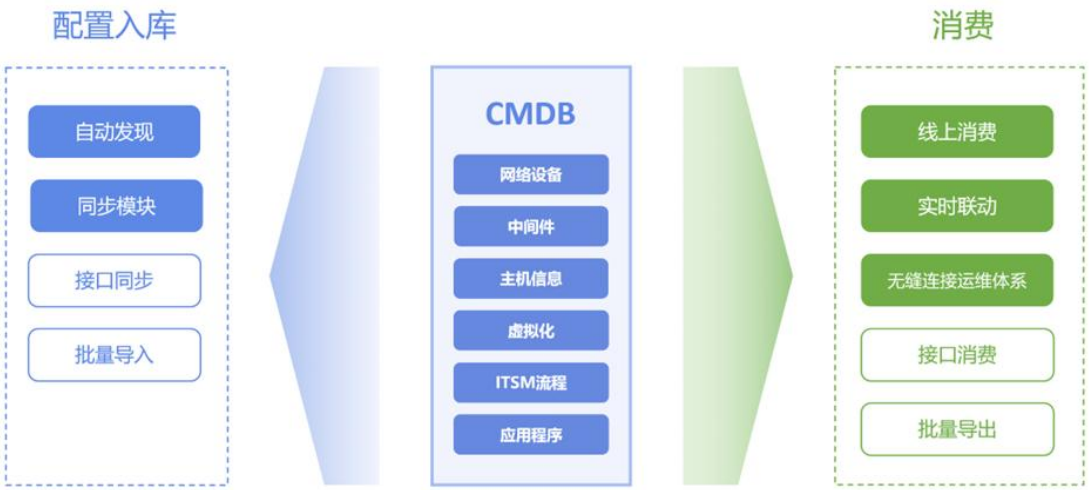
2）模型关系：变更的配置数据之间，天然存在关系。使用“模型关系”定义模型之间是如何关联。要提供可视化的拖拽页面，通过关联关键字，在不同模型之间建立一对一，一对多的约束关系，所有实例支持无差异关联。要提供关联关系的创建，删除。为了确保模型的基础功能，部分内置关联不可以删除和修改。



(2)配置元数据入库与消费

变更实施前，需要将与变更相关的配置项元数据录入CMDB，包括网络设备、中间件、主机信息、虚拟化、流程信息、应用程序、维护人员、系统环境等，要支持批量录入、更新、导入等功能，支持通过 API、SNMP、日志分析等模式的自动发现入库元数据。变更发起后，周边系统可以通过订阅和推送的方式获取消费数据，不仅要做到消费数据的实时联动，还应该做到无缝连接运维体系，从而通过变更管理，实现CMDB 配置数据的全生命周期管理。

1. 资源入库：CMDB 的数据新增入库应支持与变更管理中的资源申请、应用投产等变更场景进行关联，在相应流程结束后，数据可自动新增录入 CMDB。
2. 资源变更：CMDB 的数据变更应支持与变更管理中的资源变更（如：主机扩容、设备迁移）、应用版本变更等变更场景进行关联，在相应流程结束后，数据可自动更新 CMDB。
3. 资源退库：CMDB 的数据变更应支持与变更管理中的资源退库（如：主机下线、设备报废）、应用下线等变更场景进行关联，在相应流程结束后，数据可自动更新 CMDB。



支持更高效地变更管理流程是 CMDB 重要消费场景之一。变更管理流程主要是对 CMDB 中的数据信息进行查询消费，其中包括：

1. 配置项的基础信息查询：变更流程的发起环节，需要从 CMDB 获取变更对象相关的配置信息，以识别变更对象是否准确。因此，CMDB 工具的设计应考虑提供实例数据的信息查询接口。
2. 配置项的拓扑结构查询：变更流程的评估环节，需要从 CMDB 获取变更对象的关联信息，通过支撑、依赖、上下游等关联关系来识别变更的影响面，以辅助评估变更风险。因此，CMDB 工具的设计应考虑提供基于特定对象实例的拓扑结构信息查询接口。
3. 配置项的管理信息查询：变更流程的审批环节，需要通知对应的运维负责人。尤其针对重大变更，需要组织变更的干系人一同进行审批。此时，需要从 CMDB 获取变更对象的组织、负责人等信息，以实现变更管理者进行准确的通知和会议组织。因此，CMDB 的配置项模型设计应考虑管理信息字段的设计和维护。

(3)配置实例管理

1）支持配置实例的创建、更新和删除等能力，信息更新时，通过空值判断，长度校验，正则表达式等，确保数据准确。

2）在变更发起时，要对实例进行准确的查询，支持全局查询，多维度联合查询，管理实例查询等。

3）为了避免变更越权操作，所有的变更操作要按照角色，业务，实例，层级等多维度控制。

4）提供操作审计的功能，对变更操作可以追溯，降低变更风险。



(4)CMDB 与变更场景的联动

CMDB 的重要性上升，是因为不仅仅需要支撑变更管理流程更高效地运转，同时还需支撑后续的自动化变更执行。而自动化场景中，如何支撑好应用发布层面的变更是比较复杂的，CMDB 的模型设计需要重点考虑如何支撑好应用发布自动化的场景。

CMDB 应用层的模型设计主要采用以业务为中心的设计理念，我们称之为“业务拓扑”：

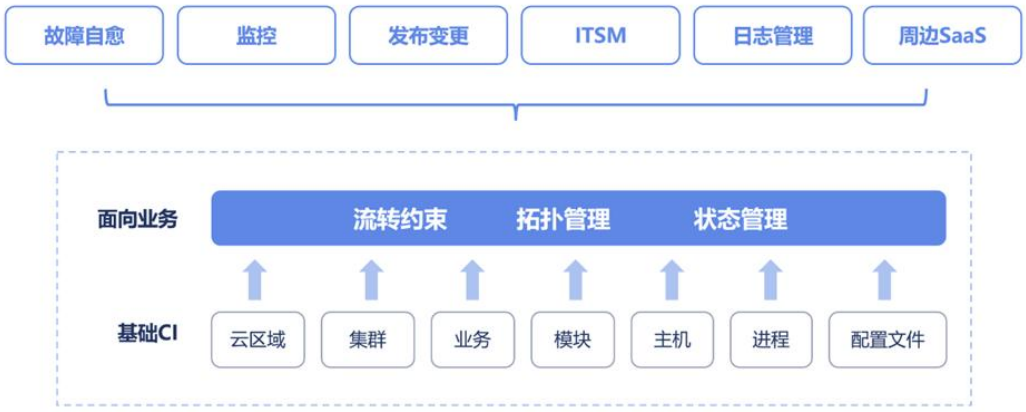
业务拓扑是对业务应用系统所提供的业务功能进行领域的划分，纵向来看，业务拓扑描述的是一个应用系统的划分规则，其最小颗粒度为单独对外提供服务能力的模块。

第一，模块为业务拓扑末端节点。模块在功能上对外提供独立的服务能力，在应用系统中为最小的组成部分，往下，与基础架构资源直接关联（中间件，主机，调用的数据库等）。

第二，业务拓扑一般不要超过 3 层。超过三层的业务拓扑太深，容易出现管理上的混乱，不利于场景的消费和管理。

第三，支持多环境管理。在应用从需求到发布的阶段中，需要在不同环境中进行进阶，开发环境，构建环境，测试环境，预发布环境，发布环境等。环境影响的不仅是基础架构资源的划分，还是影响应用配置的变动。不管是在 DevOps 还是 SRE 理论体系中，应用环境的管理都强调了其重要性。

第四，支持多集群管理。这里的集群指的是分布式的集群。每个集群都对外提供相同的业务服务，集群中包含着一个或多个模块。集群一般在互联网应用架构下出现得比较多，如：广东集群，广西集群。



4.2.3.3 面向变更管理的作业平台设计

变更是一个高频操作，作业平台（Job Platform）的目标是提供一个统一的软件工程体系，能够将变更中使用的“脚本执行，文件传输”等操作规范化，标准化，无需远程登录目标主机，就可以完成变更操作。通过这种设计，可以不断沉淀优秀的变更经验，为变更操作的自动化，与周边系统的联动，提供有力的功能保障。

(1)作业通道能力

设计一个先进的作业平台，首先在底层需要有一个能面向海量、异构、跨区域的 IT 基础架构的高性能、高可用、多功能化的作业通道能力。这些能力包括以下方面：

1）任务服务。可以目标管控节点上的远程作业任务执行能力，如 Bash、Perl、Bat、PowerShel、Python、SQL 等。

2）文件服务。提供文件快速传输能力，实现在不同的文件大小和节点规模下，动态调整组网策略以此达到最佳的文件传输效率，为业务提供高效稳定的底层文件传输服务。

3）代理（Agent）。可以安装在业务需要管控的实体机、虚拟机或者容器里面，是任务执行和文件传输的实际执行者。代理还需要具备灵活的、可自定义的插件扩展功能，以支持通过 snmp、ipmi 等各类协议对各类 IT 设备或接口进行命令操作或数据获取。

4）平台服务（Proxy 集群）。由于被管控的设备往往分散在各个网络区域，作业通道还需要支持在各个物理或虚拟网络区域部署高可用的 Proxy 集群，以实现高性能、高可用的跨区统一管控能力。

(2)变更脚本执行与变更文件分发

脚本执行和文件分发是作业平台最基本能力，工程化落地需要考虑以下几个方面：

1）快速执行脚本能力。快速执行脚本提供的使用场景是针对需要快速执行的一次性任务，并且可能会反复调整脚本逻辑来执行得到用户想知道的结果，类似于在终端跳板机 Console 执行的效果。脚本内容应支持手工录入和脚本库内容引用，并且支持各类的脚本语言；脚本需要支持传参，并且支持敏感参数的屏蔽；支持系统执行账号的指定或选择，并可支持执行超时等设置。

2）快速文件分发能力。快速分发文件的目的也是让需要进行一次性传输文件的用户，可以快速地执行或调试并得到结果。除了基本的文件源地址与源目录、目标地址与目标目录的配置外，也应该支持传参、超时、执行账号选择等配置，还需要考虑分发的上传/下载限速以避免对生产网络造成业务影响。

3）实例对象管理。作业平台是 SRE 体系下的一个能力模块，作业平台不应该维护一套独立的 IT 对象配置信息，而应该与 CMDB 进行对接，脚本执行与文件分发中的实例均应该来自 CMDB 而不是在作业平台中单独维护。

4）脚本库管理。脚本库是作业平台的重要资产，最常见的分类有两种，以 IT 对象类别以及场景类别，IT 对象包括从 IaaS 的硬件、网络、OS、虚拟化、容器等，到 PaaS 层的中间件、数据库、消息队列、缓存等，再到 SaaS 层的各类 Web 服务器，应用等。场景则是基于生命周期考虑的从资源交付、

安全合规、巡检，到日常变更相关的启停、扩缩容、迁移、应急自动化、灾备切换、下线等操作场景。当然，脚本库的建设也需要考虑参数、版本、账号、调试等能力，这里不再一一赘述。

5）文件管理。从场景上看文件分发相关的场景包括软件安装、软件更新、配置文件更新、操作系统补丁下发等；从类型上看文件可简单分为二进制文件和可编辑的文档类配置文件。这两类文件有参数、版本、分类等通用的管理诉求，可编辑的配置文件还需要具备在线编辑、生产比对、KV 提取与发布的管理功能。

(3)变更作业模板与执行方案

通过流程编排能力，将运维操作场景中涉及的多个脚本执行或文件分发步骤组合成一个作业模板，这个作业模板尽可能把场景相关的共性逻辑都包含进去，然后再根据实际使用场景衍生出相应的执行方案，那么作业模板和执行方案的关系即为"一对多"。

1）作业模板。作业模板需要具备基础信息、全局变量、作业步骤、作业编排四个基本能力。基础信息主要是描述作业模板名称、场景标签。全局变量需要支持字符串、命名空间、实例、加密字符串等类型。作业步骤需要支持执行脚本、文件

分发、人工确认三种类型。作业编排则是可以将这些步骤进行串并行组合最终形成一个作业模板。

2）执行方案。作业模板创建完后，即可以从模板中创建出一个或多个根据场景需求定制的“执行方案”；每个执行方案可以从模板中勾选自己所需的步骤，修改全局变量的变量值。可以根据变更范围设置执行方案的策略和机制。

滚动策略：按照百分比（10%，按每 10%台一批直至结束）或者具体的数值（10，按每 10 台一批直至结束）设置。

滚动机制：如执行失败则暂停、忽略失败，自动滚动下一批、不自动，每批次都人工确认。

3）作业调度。作业执行方案可支持定时、立即开始、周期等多种调度能力。

4）执行控制。作业执行过程中，根据业务需求需要提供相关的开始、暂停/恢复、终止/强制终止、人工确认等功能，并实时展示相关的运行状态。

(4)变更操作安全(权限与审计)

1)高危语句拦截。作业平台应提供可自定义的高危语句自动检测与拦截功能，以避免高危变更命令操作对生产环境造成故障。

2）权限管理。作业平台作为企业统一作业变更操作底座平台，需要具备足够细颗粒度、多维度、灵活适配各类组织架

构的权限设计，以保证安全的变更操作。从工程化落地的最佳实践讲，需要从整体 SRE 工程体系上进行权限中心的设计，作业平台的权限申请、审批、控制将由权限中心进行集中控制管理。

3）作业审计。作业平台应提供所有通过页面执行、定时执行和 API 调用等渠道触发的任务历史查看记录以便于进行审计或在故障诊断中分析 IT 变更对故障的影响。

4.2.3.4 面向变更管理的微服务容器编排设计

随着业务发展，需要不断地对软件系统进行更新和优化，一些快速迭代的业务场景逐步使用微服务的架构，就需要设计针对微服务架构的变更管理工程体系。

(1)微服务容器集群管理

1)统一集群管理，纳管不同种类的 K8S 集群

在容器管理平台纳管不同来源的 K8S 集群，在业务持续建设过程中，在不同地域会综合考虑云厂商的能力、价格，以及业务特性，选择不同云厂商的容器服务。比如国内使用 TKE 和 ACK 容器服务，在海外使用 AWS 和 GCP 的容器服务。但业务是一个团队在维护，为了避免每个业务运维方都单独对接不同厂商的容器服务，额外增加适配和管理的成本。

2)支持不同的集群管理模式

容器管理平台要能够支持不同的集群管理模式，来应对不同业务场景需求。

容器管理平台需要支持独立集群模式，以便特殊业务应用能够独享该集群资源，并提供对等的权限控制；对于自建独立集群需要提供配套的集群自动化运维方案；对于托管独立集群需要对接第三方管理平台。

容器管理平台需要支持共享集群模式，在大型单个集群内提供基于 Namespace 级别或者 vCluster 级别的资源隔离能力，以应对业务在无需关心集群及节点资源情况下使用容器技术进行发布的场景。

容器管理平台需要支持联邦集群模式。大型业务往往在分布式处理能力和容灾能力上有多地多中心部署的需求，容器管理平台需要提供联邦集群管理能力，便于业务像向单集群发布应用一样向多集群发布应用。

3)集群容量管理

容器管理平台在不同集群管理模式下，提供一致的容量管理方案。集群容量管理，能够实时查看集群容量，能够手动或者自动对集群进行扩缩容操作来满足容量设计要求。

在集群容量管理自动工作时，应该具备按照一定的可配置策略，通过 CA 从节点池或者资源池获取节点，并自动加入对应集群，达成容量设计目标。

4)统一对外暴露的 API

通过建设统一的容器管理平台，对外暴露统一的集群管理 API 接口，对业务方屏蔽底层的 K8s 集群版本和厂商差异。同一个资源在不同的 K8s 版本中 APIVersion 是不一样的，为了规避发布业务方做持续适配，或者让第三方业务系统消费时单独适配，容器管理平台需要适配差异，对外提供统一的 API 服务。

5)表单化的集群资源管理能力

在不同集群模式下，对 K8s 集群提供表单化管理能力，能够对 权限范围内的 Node、Cluster、NodePool 等资源进行统一管理。能够实时查看权限范围内的集群状态、资源用量、CPU 利用率、装箱率、节点列表等基础信息；能够通过页面对权限范围内的集群进行增删改查。

(2)微服务应用部署及变更

1)管理微服务应用部署及变更配置和流程

以表单模式和 YAML 模式构建容器应用部署配置，包括但不限于编排 Deployment、StatefulSet、Service、ConfigMap 等资源对象。表单编排模式，是将 YAML 文件中的重要变量和常用变量抽取出来，做成让用户选择的表单，降低运维工程师使用门槛。

表单模式的局限性在于，不是所有的容器配置都会支持在表单中支持，因此也需要支持以原 YAML 文件方式去编排容器应用，支持用户导入 YAML 文件，从 K8s 直接同步 YAML 文件等方式。

不论是表单模式还是直接编辑 YAML 文件，均应提供不同部署及变更配置的版本管理能力，并支持不同版本间的比对、升级、回退。

微服务应用的发布变更应该支持通过任务模板，创建多个步骤/节点执行微服务发布任务。发布任务模板中可以编排容器发布节点，不同容器发布节点支持串行发布、并行发布等方式。针对微服务场景，每次版本发布要灵活发布业务系统中的不同服务，因此节点需要支持灵活的开关，满足微服务特性要求。

2)支持多样化的应用部署及变更方式

不同业务形态下，应用的特性有很大差异。容器管理平台应该具备多样化的编排调度能力，来支持不同类型应用的部署和变更。

支持无状态应用的部署和变更。支持通过 K8s 原生 Deployment 方式或者其他扩展 CRD 的方式，支持无状态应用的部署和变更，并提供过程可视化和结果的实时反馈。

对有状态服务的支持。有状态应用往往是业务中的核心服务或者特殊服务，在业务系统中往往具有重要的价值，容器管

理平台应当通过扩展 CRD 方式对有状态服务提供更贴近真实场景的部署和变更能力。这些能力应该包括但不限于以下方面：

a)容器原地更新能力；

b)支持变更前置或后置处理；

c)支持按照一定的步骤或者顺序完成部署或变更；

d)支持部署或变更交互式过程控制；

不管是无状态服务还是有状态服务，容器管理平台应该能够使用上述能力，依据用户设定的方式来动态调整应用实例个数来达成应用容量设定。当满足扩容条件时，自动拉起新的服务实例；满足缩容条件时，能够按照一定策略驱逐容器，并缩容容器实例，以达成应用容量设定目标，并兼顾减少碎片和装箱率要求。

微服务业务部署和变更，在灰度策略上有更灵活的选择。容器管理平台应该实现这些灰度策略，并管理灰度过程，以确保业务部署和变更的顺利。这些灰度策略，包括但不限于以下种类：滚动升级、蓝绿发布、金丝雀发布。在进行灰度变更时，能通过南北向流量调配来控制灰度过程。

3)容器应用状态管理

容器管理平台提供页面，展示 Namespace、Workload、Pod、Container 级别的状态数据，使得用户能够以直观方式实时查看应用运行状况。

除此之外，K8s 提供的 Service，以及其他自定义控制器，可能提供横向维度的映射关系，在应用状态管理页面，也需要对这类映射关系进行展示。

在实际 SRE 工作中，往往同时涵盖传统物理机、VM 和容器，为便于统一管理，在 CMDB 中需要维护容器应用拓扑关系，以及容器与母机的映射关系。在 CMDB 中的容器应用拓扑关系，可以精确到 Pod 级别，以便于精确维护不同资源间的关系。

(3)微服务容器的可观测

微服务容器在变更时的，支持查看容器服务相关的事件、日志、监控。会调用 API 实时拉取事件和日志，采集监控数据，查看容器资源对应的详细监控数据。

查看容器发布状态：会调用 K8s 接口查询容器资源的状态，比如 deployment 是否处于 running 状态，副本数是否符合预期

查看容器发布事件：会调用 K8s 接口查询容器资源的事件，pod 调度到哪个节点，镜像拉取成功，pod 启动成功等事件记录

查看容器发布日志：容器变更之后，运行中会产生容器日志，查看容器标准输出日志中是否存在较多 error 字段

查看容器发布监控：容器变更之后查看容器的 CPU、内存、网络等指标监控是否正常。

4.2.3.5 面向变更管理的跨系统调度编排设计

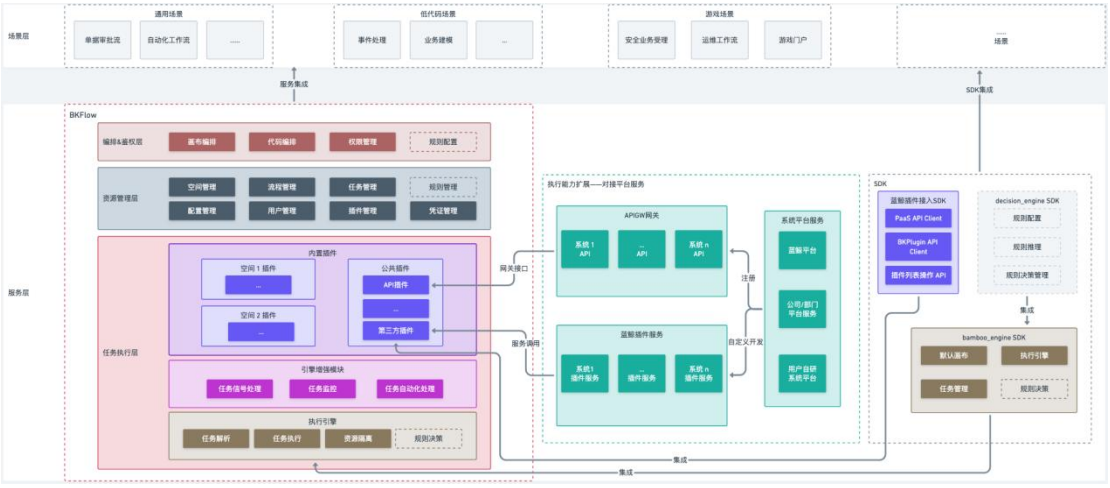
完整的变更过程，往往会涉及对各个系统进行变更执行，例如变更前对监控告警策略的调整，变更过程完成对应用的更新和网络策略的变更，变更后进行全面的业务验证等。有些变更还需要考虑多环境、多应用、多实例、多机房、多可用区的灰度变更流程设计，此时需要不仅是作业平台，还需要提供跨系统调度编排能力来实现变更全过程的自动化。

(1)通用调度引擎

通用调度引擎是整个跨系统调度编排设计的核心部分，它负责管理和调度不同系统之间的数据和业务流程，即变更的流程执行能力。在变更流程执行过程中，也要支持决策引擎和 FEEL 表达式解析器，让变更的操作流具备自动化的逻辑判断能力。

- 流程引擎：可以解析，执行，调度由用户创建的流程任务，并提供如启动，暂停，继续，撤销，跳过，停止，强制失败，重试和重入等灵活的控制能力，支持立即、定时和周期性启动的调度方式，支持并行、子流程等进阶特性，并可通过水平扩展来进一步提升任务的并发处理能力。
- 决策引擎：基于 Python 的 DMN(Decision Model Notation) 库，使用 FEEL(Friendly Enough Expression Language) 作为描述语言，可以作为决策引擎，可用于流程引擎中分支网关条件表达式的解析，解决实际业务场景中的决策问题。

流程调度引擎在设计的时候，可以分为任务执行层，资源管理层，编排和鉴权层。



通用调度引擎需要具备并发处理能力、容错能力和可扩展性，以适应不同类型变更操作，如脚本执行、API 调用、基于通用网络协议的变更操作等。

(2)编排能力

编排能力是将变更过程中各个系统的数据和业务流程进行有序组织和协调的能力。

支持通过“可视化画布”的方式，拖拽变更流程中需要的节点，在变更流程中添加逻辑网关，如串行，并行，分支，汇聚，条件并行。支持嵌套编排，设置变更的子流程。提供基于流程即代码的自动化流程编排。

变更流程节点中支持内置一些插件，这样制作一个变更流程将更高效。

- 变更操作的基础插件：定时插件、HTTP 请求插件、发送通知插件、人工确认/暂停插件、审批插件、消息展示等。
- 常用变更系统的插件：CMDB 的插件（删除主机，主机加锁，修改主机服务状态，按照规则修改主机自定义属性，故障机替换，转移故障机至待回收/故障机/空闲机模块，批量更新主机/模块/集群属性，上交主机到资源池等），作业平台的插件（快速执行脚本，快速分发文件，获取任务日志等），ITSM 插件（更

新变更日历，关闭变更工单等），监控插件（告警屏蔽插件，解除告警屏蔽插件等）。

通过编排能力，在一个变更流程中，就可以编排多个系统的变更步骤，集中化地实现变更操作的自动化和批量处理。

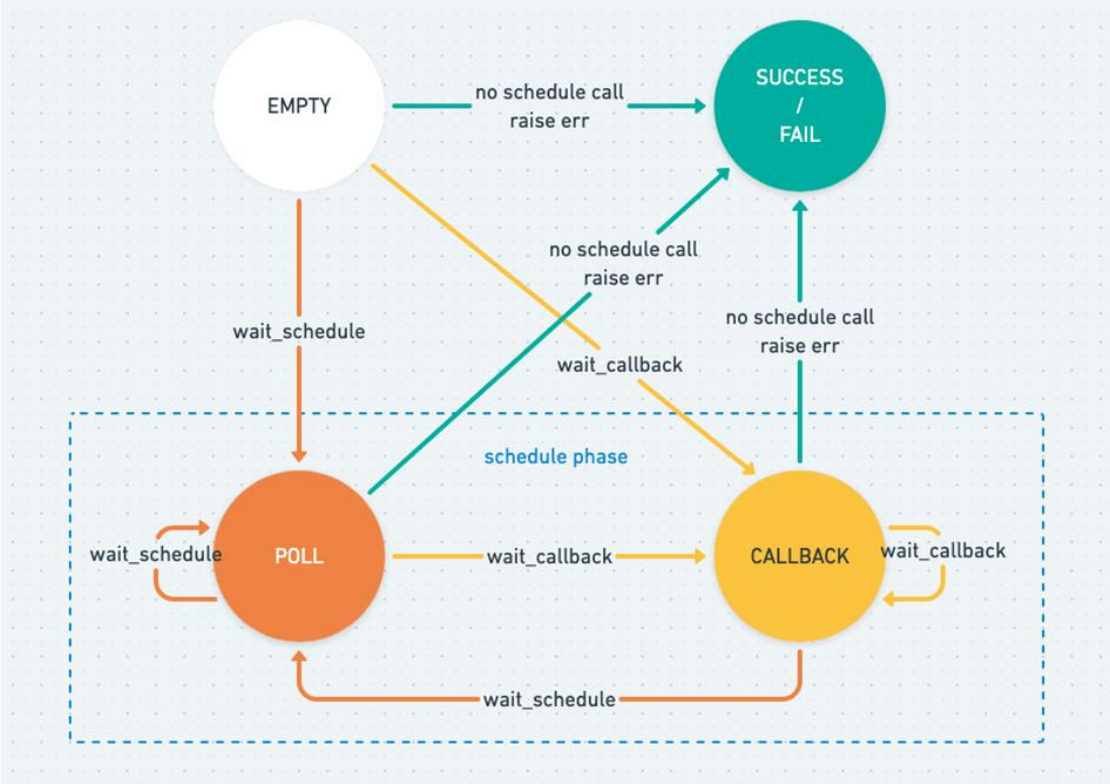
(3)扩展插件市场

当某个系统的变更操作过程较为复杂时，可以通过丰富的表单界面和验证逻辑将企业内部各个系统、各个平台的变更操作过程组装成一个标准插件模板，来实现体验更佳、效率更高的调度编排能力。

插件的开发需要具备完整的开发流程规范，包括高效的可视化低代码的开发方式、插件调试和快速发布能力。

建设插件市场，要使用统一的、轻量化的插件开发框架（plugin-framework），开发者使用该框架进行插件开发，通过调用 plugin-framework 暴露出来的标准接口，完成系统插件功能的实现，并将其部署到变更管理的系统上，就可以使用插件的能力。

一个插件在一次执行生命周期中可能会经过下图所示的状态转换，通过框架提供的方法，开发者可以在插件逻辑中进行任意合法的状态流转，以适配不同的业务变更场景。



通用性的插件可以发布到“插件市场”，共享插件资源。企业可以根据自身需求，从插件市场中选择合适的插件，以实现对不同系统的支持和扩展。

(4)调度编排模板与任务

编排后形成的流程称之为模板，模板应具备全局变量、版本管理、版本审批发布等功能。企业将一系列通用的变更操作和流程节点创建成调度编排模板。企业就可以快速构建和管理变更流程，提高变更管理的效率和准确性，降低变更风险。

支持模板“创建，删除，更新”等管理功能，支持立即执行、定时执行、周期性执行的任务实例，支持“模板克隆”，“导入导出”等共享模板的功能，所有任务实例的执行记录均需要被保存，以便于搜索查看、安全审计和运营分析。

常用的低风险的变更操作，如针对测试环境的变更，可以将变更模板授权给非专业人员执行，即提供生成“轻模板，轻应用”的功能，限制模板的参数，生效范围等，将只能由运维/开发执行的变更操作，授权给测试/运营/策划等职能岗位使用。

变更调度编排模板在跨系统调度编排设计中可以帮助企业快速实现特定场景下的变更操作。企业可以根据自身需求，选择合适的调度编排模板，并根据实际情况进行定制和优化。

4.2.3.6 面向变更管理的可观测设计

变更是导致生产故障的主要原因，因此在变更执行过程中，实时观测并感知因变更触发的各种业务异常如告警、故障等事件，是变更执行者主动并及时解决问题，降低故障影响半径的有效方式之一。为了实现变更的可观测性，需要具备以下几个能力：

1. 变更影响分析：实时感知变更对系统或服务的影响，需先建立系统的依赖关系，包括业务依赖关系、应用调用关系和资源关联关系。业务依赖关系指变更对象与周边业务服务的依赖关系；应用调用关系是指微服务之间的调用关系；资源关联关系则涵盖数据库、虚拟机、物理机等资源的关联。有了这些依赖关系，可以在变更过程中对变更对象及其关联对象进行观测，实时感知生产业务运行状态。
2. 全栈可观测：建设基于 Trace、Metric、Log 的可观测能力，并将这些能力与变更影响分析结合，实现对变更相关对象在变更过程中的全面可观测。
3. 灰度验证：变更通常以灰度方式进行，结合自动化验证确保每批次变更无异常后再进行下一批次变更。根据不同业务与应用特征，可设计不同的批次变更时间间隔。
4. 流量闭环：全链路灰度是一种在微服务架构中实现应用版本平滑发布的技术。它不仅需要在网关处做灰度策略，还需要在微服务调用之间实现灰度。通过观测系统，我们应该能观测到在进行全链路灰度时，相关的流量在相应的泳道内闭环，不产生逃逸。而相关实现上，需要使用到链路跟踪数据。

5. 版本质量对比：通过观测手段，分析应用版本在性能、可用性、用户体验等方面的数据对比，以持续改进 CI/CD 链条质量。
6. 变更运营分析：从组织、对象、运维场景等多个方面对变更的成功率和服务质量进行统计分析，发现变更脆弱或问题环节并持续改进优化。

变更流程中，通常会采取研发/SRE/运维人员盯盘监控的方式进行业务分批变更及发布上线，这样可以在业务出现变更故障时，及时发现并由 SRE 进行业务止血操作或回滚，防止变更引入的风险进一步扩大，从而影响大量用户。

以下是几个重要的监控指标，需要关注：

1. SLO（Service Level Objective）指标：
   - 定义和监控 SLO，明确服务的目标水平（如响应时间、可用性等），并在变更过程中实时监控这些指标，确保服务质量不下降。
2. RED（Rate, Errors, Duration）指标：
   - 请求速率（Rate）：监控每秒处理的请求数量，确保变更不会导致请求速率异常波动。
   - 错误率（Errors）：监控请求的错误率，及时发现并处理变更引入的错误。
   - 请求持续时间（Duration）：监控请求的响应时间，确保变更不会导致响应时间显著增加。

3. 核心业务指标：

- 业务关键性能指标（KPIs）：根据具体业务，定义和监控关键性能指标，如交易成功率、用户登录成功率、购买成功率等。
- 用户行为指标：监控用户行为，如页面访问量、点击率等，评估变更对用户体验的影响。

4. 错误日志比例和数量：

- 日志收集和分析：集中收集变更前后的错误日志，分析错误类型和数量，及时发现和解决问题。

特别要提醒的是，除了在变更时需要监控数据外，在回滚后的验证场景也需要进行密切的观测，以确保系统恢复到预期状态。

4.3 变更管理案例

4.3.1 B 站变更防控的设计与实践

SRE Elite 收录点评

该平台利用了 trace 和 CMDB 资源拓扑信息，以关联和聚合应用服务的变更，能够追踪并识别变更对整个服务生态系统的潜在影响，有助于提高故障排查的效率和准确性。

在企业内不同部门存在多套发布变更系统，且短期难以推倒统一重建的情况下，从变更防控的视角，补充建设防控平台，通过统一模型、熔断控制等方式，实现自动化的集中式变更管控。

（一）背景及设计原则

目前，在SRE 的业界，对变更的重视程度已然达到了顶峰。在实践过程中，基本上是围绕变更方案评审，变更发起前控制（重大活动/节假日等关键时间控制），变更事件记录这些开展变更管理工作。

这种做法应对传统的IT 架构是足够的，但是在目前围绕云原生+微服务的这套复杂技术体系来看，存在着诸多问题，诸如：

评审围绕流程驱动，在评审环节依靠人的经验来确保方案的正确性，过于依赖经验并难以持续保证效果

由于资源的稀缺性，难以确保每个变更都能够纳入评审，仅能覆盖部门高优重保的大型变更计划

评审行为仅能约束变更的发起，无法保障变更执行过程按照要求切实有效执行，比如强制分区灰度要求，强制观察时间要求等

基于人操作的变更和平台操作的变更，围绕技术平台的基础设施和业务平台的配置变更，散落在各个地方，各自对于变更用了各自的流程定义和描述，难以收归一起管理和分析

单纯的变更记录，难以支撑当下复杂场景下由某个边缘变更、底层变更导致的级联故障定位

由于缺乏完整的变更管控技术框架，难以实现公司内部所有变更的平台托管，即无法知晓到底存在多少变更行为，哪些地方会发起变更，每次变更具体是变了什么东西，变更影响具体影响了什么对象和范围等

面对以上诸多局限，秉承技术问题还需要技术解法的原则，将变更作为一个独立的技术域，抽象和定义了变更的对象概念，规划和设计了变更管控技术框架，围绕这个框架打通从变更元信息定义、变更平台/场景接入、变更防控到变更分析的全流程链路，来实现变更的一体化和多层次管控。

（二）体系设计详情

B 站的变更防控方案基于行业最佳实践，构建了工程化的变更管控体系。通过标准化描述模型和过程模型、分层策略、全方位托管、灵活感知、准入准出检查和熔断机制，实现高效变更管理和价值挖掘。

（1）变更定义

统一变更描述模型

针对不同角色对变更的理解以及不同业务部门在语义的定义上是一定存在差异的。在B 站，应用变更、配置变更和基础设施变更分别由不同平台承载。由于这几种变更场景对变更信息的定义存在差异，因此实现不同类型变更统一管控的目标需要抽象出一个通用的变更信息模型。这个通用的变更信息模型将会作为一个中介层，跨越各种平台和部门，以统一的方式描述和定义变更信息，确保所有参与变更的个体和系统都有共通的理解和认知，从而提高变更管理的效率和准确性。

变更描述模型应包含以下基本信息和管控信息作为元数据：

基本信息

- 变更来源：用以标识具体的变更平台和平台唯一标识
- 变更对象：包括变更对象类型(应用/数据库/服务器等)和变更对象唯一标识
- 变更时间：变更具体执行的时间
- 变更人员：变更实际的操作人员
- 变更环境：变更所生效的环境，比如 UAT/ PRE/PROD 等

- 变更内容描述：对变更操作的详细描述，包括 Release Notes 和详情描述，对于传达变更目的和实施细节非常重要。

管控信息

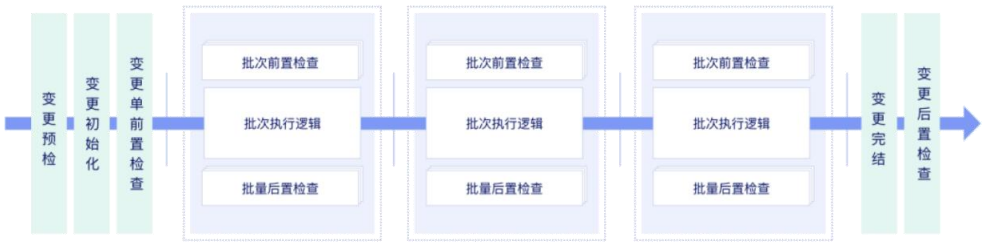
- 变更目标：即期望通过变更来达成的目的，包括日常迭代、BUG 修复、故障处理等；
- 变更场景：指具体的变更动作，包括迭代发布、配置发布、服务器上下线等；
- 变更范围：即变更在线上环境的生效范围，包括机器分批生效模式以及流量比例分批生效等；
- 变更影响：指本次变更影响的服务和基础设施

标准化的信息模型使得后续的变更感知、变更预检、变更防御以及变更审计等操作都能够基于统一的信息结构进行，有助于降低管理复杂性，提高协同效率。此外，对于其他技术风险领域的能力建设，这个模型也为故障定位、根因分析等方面提供了通用的数据结构和框架。

统一变更过程模型

统一变更过程模型旨在提供标准化方法，以一致的方式描述和管理从提出到实施的整个变更生命周期。

这个模型通常包括以下几个关键组成部分：



一个变更的生效往往是基于一定范围和流程的，因此我们需要围绕这个过程进行变更流模型的定义。通过流模型定义，我们可以抽象出两类基本的变更流模型，分别适用于逐步生效的变更和无法拆分的变更：

1、无法拆分的变更流模型：

即一次性生效模型，适用于某些无法拆分或拆分成本较高的变更场景，例如 DB 配置类型变更。在这种模型下，变更控制防御能力主要集中在配置变更前后。

2、逐步生效的变更流模型：

按机器分批生效：将变更分为若干批次，在每个批次中逐步将变更应用到不同的机器上。这种方式允许在不同的批次上实施更多的风险管控，以便更精细地监控和评估变更对系统的影响。

按流量比例分批生效：允许按照预定的流量比例逐步引导用户访问到经过变更的系统。这种方式同样可以在不同阶段上实施风险控制，适用于需要更细粒度控制用户流量的情况。

3、变更等级分层

变更等级分层策略根据潜在影响、风险和重要性将变更分级，以便更精确地控制和管理变更过程。这样的分层策略有助于组织更加精确地控制和管理变更过程，确保关键系统和应用的稳定性和可靠性。

等级	含义	数量	生命周期定义	使用场景
G0	仅做事件同步，无管控能力	一个	仅有一个事件同步节点	适用于无任何风险的变更，但数据需要提供给相关人员做检索、审计等场景
G1	仅做单节点的变更前后切面管控	一个	生命周期中仅有一个变更节点	适用于低风险变更，无需复杂风险管控能力，仅做事前准入性和事后完整性检查的场景

G2	有完整的变更工单，且工单下关联了至少 1 个批次的子节点	多个	生命周期分为四个阶段(工单开始>各批次节点开始>各批次节点结束>工单结束)	适用于多数逐步生效的变更，并需要配套进行风险管控的场景
G3	在完整的变更工单感知基础上，增加了变更提单阶段感知	多个	生命周期在 G2 等级的基础上，前置增加了变更提单阶段	
G4	在变更提单感知的基础上，增加变更无人值守的决策能力	多个	生命周期在 G3 等级的基础上，变更提单后增加无人值守决策阶段	适用于需要系统进行无人值守代理执行变更的场景

以上某组织变更参考标准

每个管控等级的变更场景在满足前一级别的基础之上衍生出更多的变更要求和管控切面。基于标准的变更信息模型以及变更等级，我们就可以将变更管控单独抽离出来,在不同的平台业务上构建统一的变更管控平台,使得 SRE 团队能够更专注

于变更的风险防控与治理，同时平台业务团队更专注于平台研发本身。

（2）变更管控平台架构设计



变更管控平台面向不同用户提供丰富的功能，包括面向研发和 SRE 的变更信息感知、检索和订阅，以及面向平台方提供的变更场景接入、托管等能力。以下是对每个功能的具体描述：

变更信息接入：通过标准变更信息模型，对整个变更场景进行完整定义，确保变更场景的一致性和可管理性。支持对变更场景的全生命周期进行编排，包括计划、执行、监控和反馈，提高变更管控的可追溯性和效率。

变更信息感知：覆盖范围广泛，平台涵盖了服务器、网络、数据库、中间件、Pass、系统发布、系统配置、业务数据

等多个领域的变更信息。提供变更搜索、变更订阅、影响面分析等功能，使用户能够快速准确地获取关于变更的信息。

变更过程防控：根据变更管控等级和防御能力，实现对变更风险的发现和阻断。提供灵活的变更防控策略，确保对不同等级的变更采用适当的防控手段。

变更过程分析：通过深入分析变更实施的各个环节，及时发现潜在隐患，优化变更策略，确保变更的顺利进行和业务稳定。

变更结果度量：通过对变更防控效果进行度量，包括场景覆盖率、风险检测质量、变更阻断准确率、变更防御执行效率等方面的指标。通过分析结果，持续提高变更防控的各项指标，优化防控策略和执行效率。

通过这些功能，变更管控平台不仅提供了全面的变更信息管理和感知能力，还支持对变更场景的标准化托管和灵活的防控策略。同时，通过变更分析功能，平台能够持续优化变更防控效果，提高整体的变更管理水平。

(3)变更平台/场景接入

变更场景是整个变更管控的核心概念之一，因为所有管控动作都是基于变更场景的，这里也可以理解为变更的作用域。平台层面的接入情况，主要是元信息的录入和描述过程。





在技术层面，我们主要将一个变更场景整体做了如下定义：

```
type ChangeScene struct {
    // 归属变更平台
    Platform
    // 关联变更资源类型
    SourceType
    // 基础信息(变更内容)Schema
    ContentSchema
    // 管控信息(批次内容)Schema
    StepSchema
    // 检查项
    []Navigation
    ...
}
```

```
type ChangeControl interface {
    // 变更初始化
    InitChange()
    // 变更结束
    FinishChange()
    // 变更批次开始
    StartChangeStep()
    // 变更批次结束
    EndChangeStep()
}
```

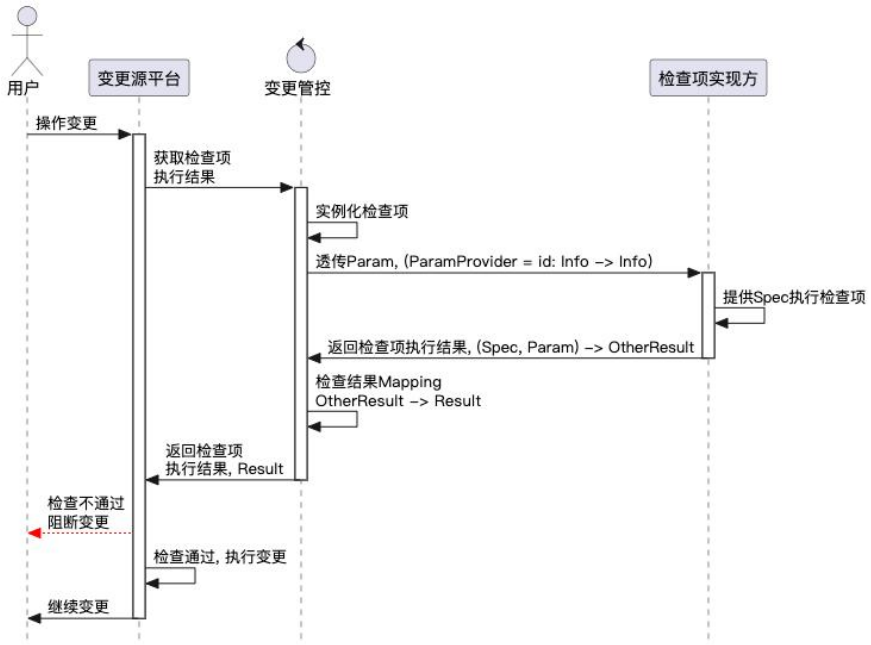
可以看到变更场景定义了一类变更，包括归属的变更平台，操作的变更资源，管控时所需的基础信息 Schema 和管控信息 Schema 以及检查项等。每一个变更都基于具体的变更场景进行实例化，执行变更场景定义的管控动作和信息交互。因此,进行变更管控的第一件事就是定义需要管控的变更场景，变更源平台在接入自身平台后需要在 ChangePilot 上托管自身的变更场景后才能进行实际的变更管控。

(4)变更元信息托管

实现变更过程的精细化拆分，拆分前后阶段，拆分批次，拆分批次前后阶段等

基于变更单、批次单的前后阶段增加切面的检查项能力嵌入

通过检查项实现过程的检测和干预



（5）变更感知

围绕全域变更信息的整合，系统构建了分词与关联机制，通过多元化的检索与触达手段，以实现人与系统间变更信息的灵活流通。

变更检索

在变更检索方面，系统设定了多种场景，包括针对单一对象或平台的时间跨度搜索，以及针对单个变更行为的搜索。

检索场景

- 围绕单一对象的时间跨度搜索

- 围绕单一平台的时间跨度搜索
- 围绕单变更人的变更行为搜索
- 检索的维度涵盖了变更标题和变更状态的修改，变更的源平台、对象和环境，以及操作人员和影响范围。

检索纬度

- 变更标题、变更状态
- 变更源平台、变更对象、环境
- 操作人、影响范围

（6）变更防控

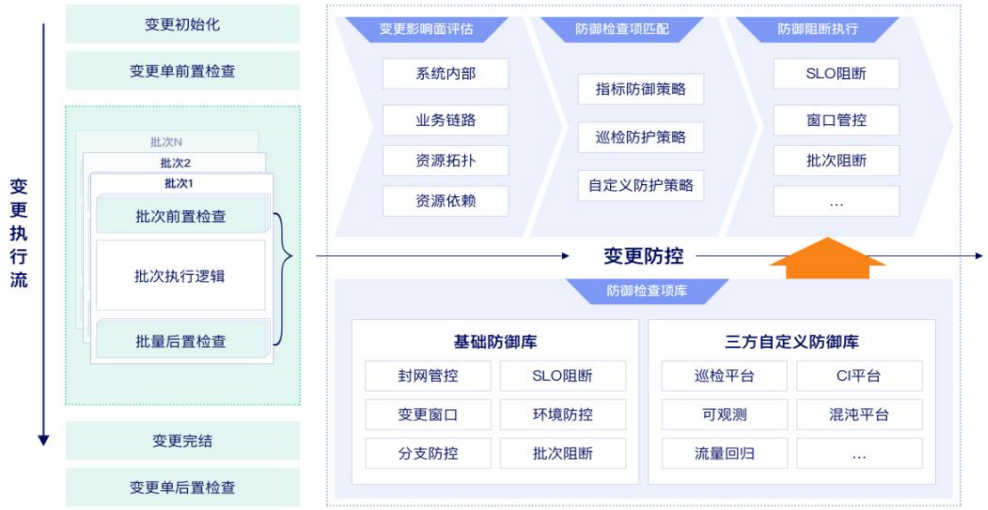
防控技术方案

变更防控方面，系统依托于变更流模型，实现了链式执行，并通过设置检查项进行准入准出。

执行策略包括观察者模式和执法者模式，以及阻断变更过程、预警提示异常、升级审批阻断变更过程等防控策略。

系统还建立了熔断机制，以应对变更系统请求失败的情况。

通过以上的设计，达成了变更防控的技术方案。

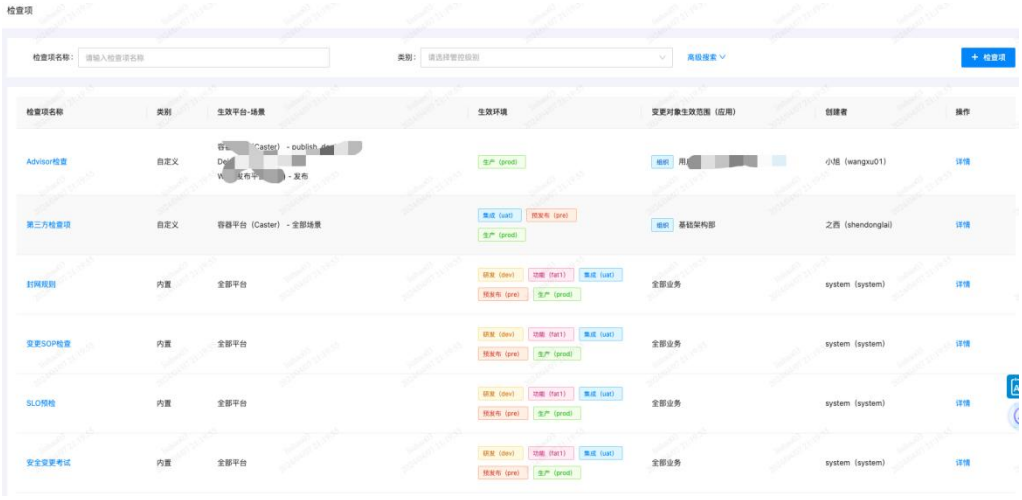


防控能力概述

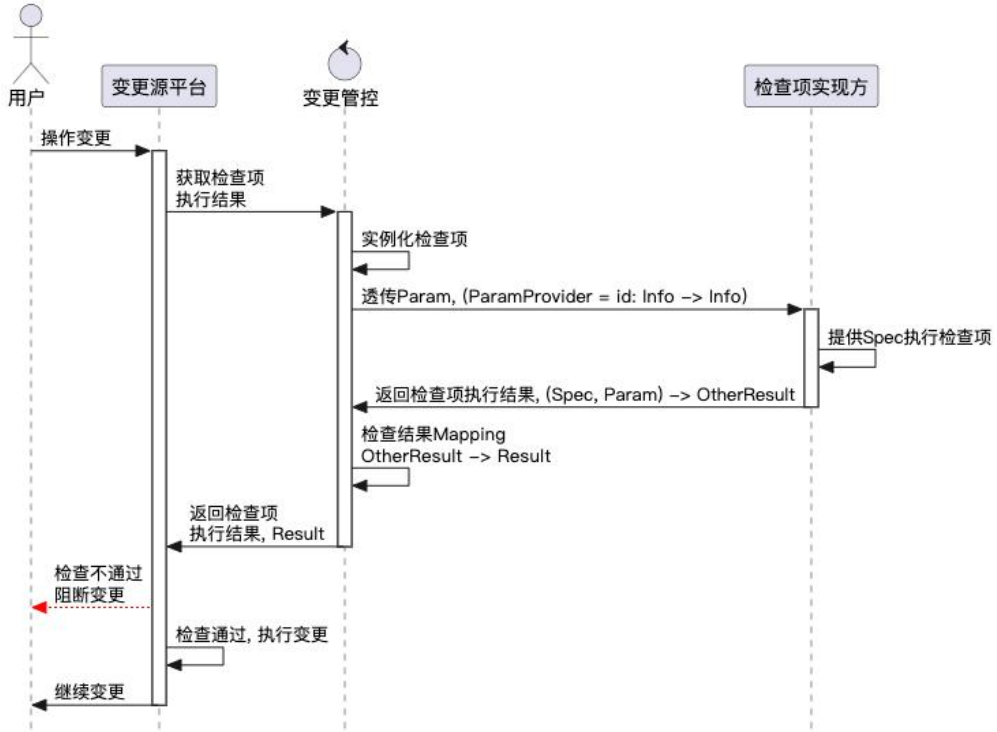
变更系统的防控能力包括通用检查项如封网拦截、SLO 检查、批次检查、环境监察等，以及自定义检查项如安全扫描、业务代码发布 SOP 等。这些检查项旨在防止不符合标准的变更进入生产环境。

举例一，有业务进行灰度变更之后，SLO 迅速下降，这个时候系统就会进行拦截，阻止业务继续进行发布，避免导致更大的故障。另外，可能的场景包括，批次检查避免用户不进行任何的灰度等。

举例二，安全团队，希望对发到线上的制品包有一个安全扫描，则可以使用构建自定义检查项，避免有安全风险的制品流入到生产环境中。


检查项
检查项名称：
类别：
高级搜索
+ 检查项
检查项名称
类别
生效平台-场景
生效环境
变更对象生效范围（应用）
创建者
操作
Advisor检查
自定义
小旭（wangxu01）
详情
第三方检查项
自定义
容器平台（Caster） - 全部场景
之西（shendonglai）
详情
封网规则
内置
全部平台
全部业务
system（system）
详情
变更SOP检查
内置
全部平台
全部业务
system（system）
详情
SLO预检
内置
全部平台
全部业务
system（system）
详情
安全变更考试
内置
全部平台
全部业务
system（system）
详情

技术交互模式


用户
变更源平台
变更管控
检查项实现方
操作变更
获取检查项
执行结果
实例化检查项
透传Param, (ParamProvider = id: Info -> Info)
提供Spec执行检查项
返回检查项执行结果, (Spec, Param) -> OtherResult
检查结果Mapping
OtherResult -> Result
返回检查项
执行结果, Result
检查不通过
阻断变更
检查通过, 执行变更
继续变更



（7）变更价值挖掘与可视

通过多维度视角对变更数据进行挖掘，以发现风险点，明确优先级，并可视化项目价值。



运营面板和检查项面板为团队提供了实时的数据视图。


2024-04-10 14:09:58
变更管控数据大盘
2024-04-01
至
2024-04-30
平台个数
封网管控:
过程管控:
场景个数
检查项个数
内置 100.0%
内置
防御情况:
绿通放行:
封网放行:
拦截率（总）
占比
0%
0%
0%
0%
0%
0%
2024-04-01
2024-04-08
绿通申请原因分布
100.00%
绿通申请趋势
绿通申请量
手动绿通列表
申请人
触发平台
申请应用
拦截率（按平台）--未实现
拦截率（按管控阶段）--未实现

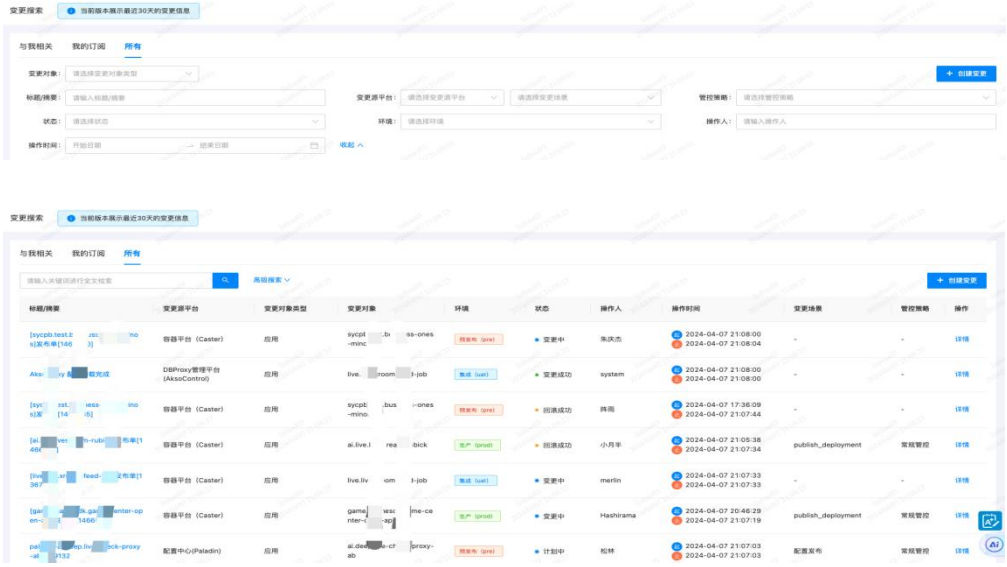
运营面板


发布前SLO拦截方式
发布前异常SLI分布
封网白名单业务分布
发布前SLO检测列表
规检单
appid
操作人
操作时间
sli
可用区
SLO当时值
阻断阈值
检测阈值
preflight-...
99.416058
99.803401
99.77838
99.830066
99.943455
共 26 条数据
封网白名单
关联封网策略
appid
原因
开始时间
结束时间
发布中SLO拦截方式
阻断 3.57%
提示 96.43%
发布过程-异常SLI分布
"HTTP可用率" 25...
"GRPC可用率" 28...
"HTTP请求可用率...
"GRPC请求可用...
"SLB可用率" 25.0...
"APIGW可用率" 1...
提示 100.00%

检查项面板

（8）开放能力

开放能力包括全量或按纬度的变更消息投递，便于让上下游系统对变更相关消息进行记录并消费。例如，可以与监控系统或审计系统进行联动。



1. 变更订阅

订阅服务允许用户根据服务、业务、组织等多维度进行订阅，包括自身变更、上下游变更，以及变更的平台和场景。触达机制涵盖个人、群组、webhook 等多种方式。



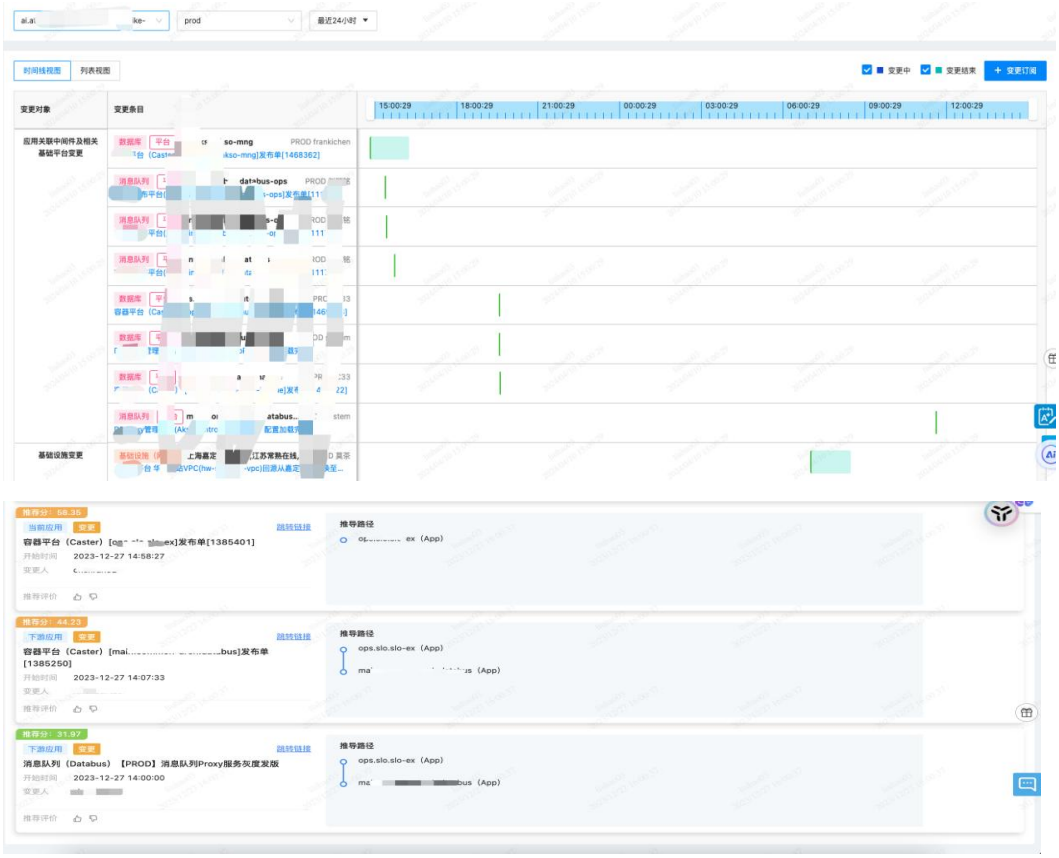


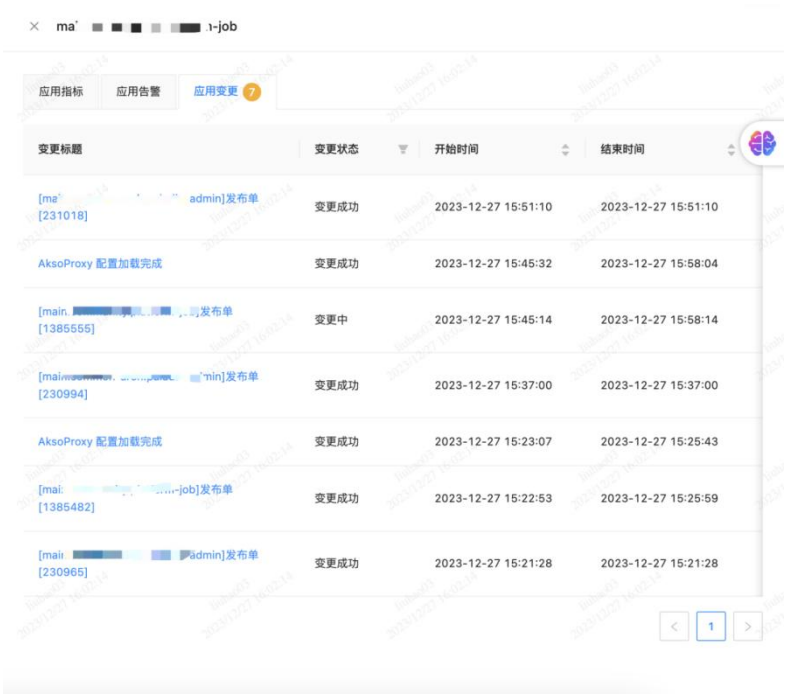
2. 变更分析

通过对变更故障的深入分析，我们发现很大一部分异常并非源自本服务的变更，而是由上下游依赖或更基础的设施变更所引起的。为了解决这一挑战，ChangePilot 利用了 trace 和 CMDB 资源拓扑信息，以关联和聚合应用服务的变更。

具体而言，我们利用 trace 分析获取了应用服务的上下游关系，同时通过 CMDB 资源拓扑信息分析得到了应用服务所管理的资源架构，包括存储层（如数据库集群、缓存集群等）、计算层（包括容器、物理机实例）以及网关层等关键资源信息。随后，通过统一的资源标识符进行检索，我们能够有效地聚合变更信息。

这种方法使我们能够更全面地了解应用服务的生态系统，包括其上下游关系和底层资源架构。通过整合这些信息，ChangePilot 能够追踪并识别变更对整个服务生态系统的潜在影响。这不仅有助于准确定位变更引起的异常，还提供了更深层次的可视化和理解，有助于提高故障排查的效率和准确性。




应用指标
应用告警
应用变更 7
变更标题
变更状态
开始时间
结束时间
变更成功
2023-12-27 15:51:10
2023-12-27 15:51:10
AksoProxy 配置加载完成
变更成功
2023-12-27 15:45:32
2023-12-27 15:58:04
变更中
2023-12-27 15:45:14
2023-12-27 15:58:14
变更成功
2023-12-27 15:37:00
2023-12-27 15:37:00
AksoProxy 配置加载完成
变更成功
2023-12-27 15:23:07
2023-12-27 15:25:43
变更成功
2023-12-27 15:22:53
2023-12-27 15:25:59
变更成功
2023-12-27 15:21:28
2023-12-27 15:21:28

（三）总结及展望

经验教训


充分设计
在各个领域中抽出来一个独立的领域做建设，存量的替换成本是极高的
平台愿意折腾的次数可能只有一次
变更领域内的核心要素一定要思考清晰，反复沙盘，再推进业务接入
标准定义
模型定义
交互协议
找准切入
前期阶段
寻找高频易错变更场景进行落地
寻找有合作意愿的团队
自己手中有场景最好，先打样
中期阶段
通过高频场景进行泛化，从组织层面进行辐射覆盖
寻找低频高危变更场景进行推进覆盖
后期阶段
为接入场景提供数据价值，基于数据驱动更多场景覆盖和接入&提升管控级别
兼顾效率
故障情况
绿色通道
白名单
延迟批次检测
牺牲批次的风险，减少用户体感
基于历史变更，动态减少变更过程的冷静期
自动盯盘，推进变更
系统自动观测指标
减少用户在变更过程中的盯盘和点击操作
保持效用
广度控制
执行力度，摊子别铺太大
到底要管控多细
深度控制
到底要对一个变更场景做多细致的防控
是不是瞎折腾了
变更不是解决新领域的问题，更多是旧领域的从新建构和构建
需要一定的时间，新方案引入的价值才能原有隐性变更建设的成效
日拱一卒
决心问题
需要持续性的投入建设，形成系统建设的架构约束
量变引起质变
变更的提升在中后期很枯燥无味，不见成效
形成机制、习惯
从强迫接收到主动托管

在变更管理的实践中，我们总结出了五个关键日拱一卒的经验教训：充分设计、找准切入、兼顾效率、保持效用。以下是针对这些教训的优化建议：

充分设计：在实施变更管理前，必须进行彻底的规划和设计。这意味着需要确保变更设计的可持续性和兼容性，避免频繁修改导致合作方的抵触。设计阶段应考虑软件的建模标准，确保内部逻辑的变更不会影响外部接口。

找准切入：在变更管理的初期实施阶段，应选取高频且容易出错的场景作为切入点，以此构建成功案例并推广。中期阶段，将成功模式推广到整个组织，包括低频但高风险的变更场景。通过展示数据驱动的成效，增强变更方案的吸引力。

兼顾效率：在变更过程中，确保不过度干预日常业务和开发工作。通过设置如绿色通道和白名单等机制，优化变更流程，减少对业务运行的影响。其中，绿色通道，主要用于快速止损，仅豁免 1 个小时，即开即生效。而白名单主要用于长时间豁免管控规则，一般用于活动期间的保障场景下一旦出现问题需要快速止损的情况，需要事前提交申请，并审批通过。

保持效用：变更管理应注重实际效用而非理论上的改进。避免过度设计检查项，应聚焦于实际需要，确保变更控制的举措既有效又实用。同时，防止在没有明确需求的情况下引入不必要的变更。

日拱一卒：持续的投入和改进是确保变更管理成功的关键。应建立长期的变更管理机制，将变更管理的标准和实践融入日常操作，逐步培养组织内对变更管理的认可和依赖。

未来展望

愿景：实现 B 站变更防控体系的充分性、高效率和有效性。持续保稳提效，在复杂多变的业务情形下借助数据&智能建立可持续的变更防控机制。



4.3.2 携程云平台基础设施变更管理实践

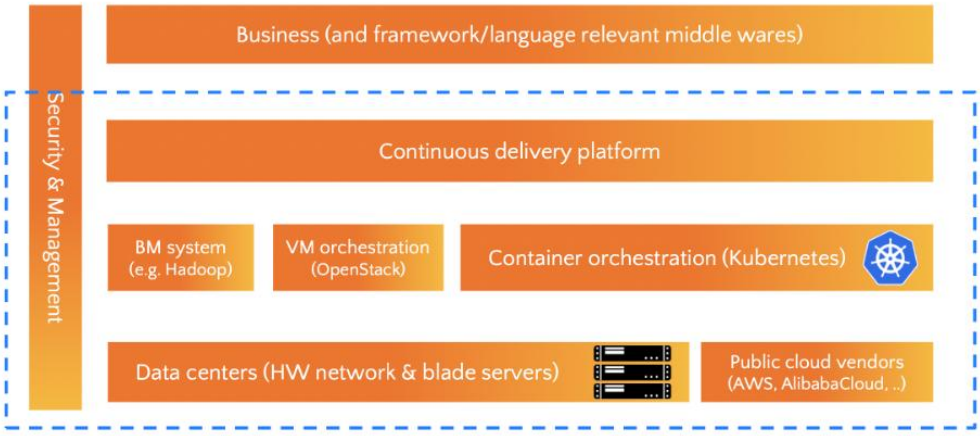
SRE Elite 收录点评

1，这是一个由混合云 SRE 团队提供的案例，作为基础设施质量的把控者，对变更的计划性和标准作业流程要求严格，对于企业内私有云 SRE 管理团队有一定的参考价值。

2，大量使用了 IaC 的方式对基础架构进行管理及变更，其中使用 SaltStack 和 StackStorm 管理配置变更，使用 Kustomize 和 Git Workflow 管理多个集群和环境的基础组件配置文件，对公有云资源使用 Terraform 进行管理，实现了全栈 IaC 落地。

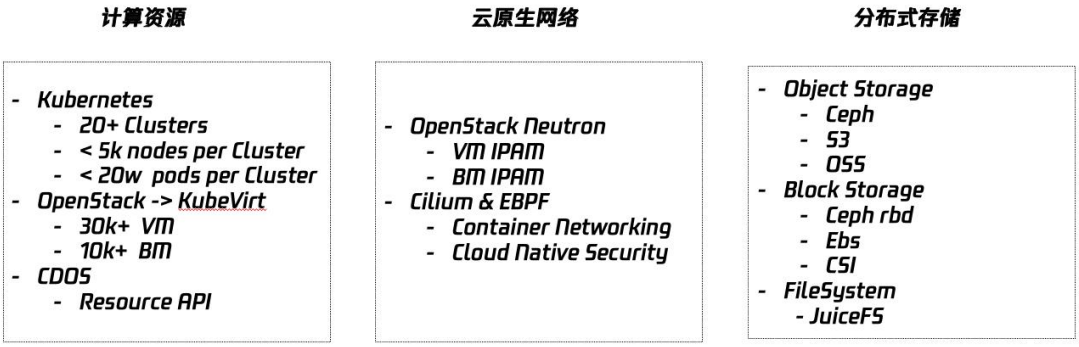
（一）背景及设计原则

携程云平台概览



携程基础设施采用混合云架构，云资源分布在自建数据中心和公有云供应商。携程混合云平台的 IaaS 层提供裸金属管理、虚拟化宿主机（KVM），以及容器宿主机等多种底层资源类型。其中容器通过 Kubernetes 进行编排和交付。更上一

层则是各种业务 PaaS 平台，诸如在线应用、数据库（DB）、ElasticSearch、Kafka 等服务。



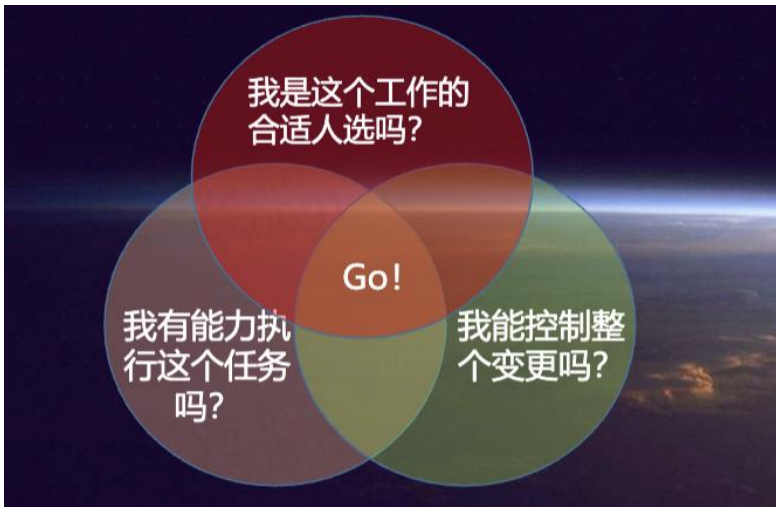
在携程的技术架构中，由云技术团队负责主要计算、网络和存储三大资源类型的管理和交付。在计算资源方面，携程云团队管理超过 20 个 Kubernetes 集群，每个集群的节点数控制在 5000 个以下，以减少单一集群故障对整体的影响。在虚拟化技术方案，携程云团队从使用 OpenStack 过渡到 KubeVirt 进行 VM 资源交付，目前运维管理约 3 万台虚拟机实例。

网络虚拟化技术方面，携程云团队使用 OpenStack Neutron 和 Calico 来管理虚拟机和裸金属服务器的网络。容器网络则主要基于 Cilium 和 eBPF 技术。携程存储解决方案在私有云中主要使用 Ceph，而公有云存储则采用 S3、OSS 等对象存储。文件存储方面，携程采用 JuiceFS 和 FastDFS 来满足不同的业务需求场景。

（二）基础设施变更管理详情

变更失败主要由两大因素引起：人为因素和流程因素。

首先，人为因素并非指操作人员故意造成故障。很多时候，故障是因为变更人员未能遵循既定规范和操作流程，导致变更失败。因此，携程要求变更实施人员在操作之前，自问自答如下三个核心问题，也称为“三步法则”：



- 是否为合适人选：确保变更人员对任务有充分的理解和背景知识，而不仅仅是盲目执行上级的指令。
- 是否具备必要能力：变更人员需评估自己是否有能力设计出完整的变更方案，包括发布计划和应急恢复计划。
- 是否能控制整个任务：在变更过程中可能会遇到意外问题，变更人员需要有能力主导问题的解决和恢复。

- 讨论这三步法则的目的，并不是完全消除人为因素导致的故障，而是为了最大限度减少人为因素的影响。
- 流程因素也极为重要。在设计变更时，应遵循几个基本原则：
- 计划性：技术设施的变更应严格按照计划执行，与普通应用的变更略有不同。
- 灰度原则：灰度执行变更，先小范围后大范围，以控制风险。
- 可回滚性原则：确保每次变更都可以安全回滚至变更前的状态以应对可能的故障。
- 避免循环依赖：变更过程中应避免循环依赖问题。
- 风险评估机制：建立系统的风险评估流程，确保变更的安全性。

计划性原则

与普通应用的变更相比，基础设施变更应该严格按照计划执行，典型例子如 Kubernetes 4 个月的发布模式，在此参照下，基础架构进行变更前，更应该制定更严格的计划。

灰度原则

- 严格按照 Availability Zone (AZ) 定义，控制变更的灰度
- 不同 AZ 的服务，不依赖同一套基础设施服务

- 控制基础设施变更的爆炸半径
- 生产不同 AZ 的基础设施变更，时间上必须间隔 1 天
- “慢即是快”

携程采用了混合云架构，并建设了国内的同城三中心高可用体系。这意味着我们的基础设施变更必须严格遵循三个可用区（AZ）的原则，确保每个 AZ 的服务都具备高可用性。即使一个 AZ 出现故障，也不会影响到其他两个 AZ 的正常运行。此外，为了控制变更的风险范围，每个 AZ 所依赖的基础设施必须是独立的，不能有共享的部分。

最关键的一点是，不同 AZ 之间的基础设施变更需要有足够的观察期。携程云团队规定，任何基础设施的变更后，必须至少经过一天的时间来观察其效果。变更后需要经历至少一个流量高峰期来验证变更的稳定性，是否能抗住峰值流量压力。尽管看似放慢了变更速度，实际上这种方法能更有效地确保变更的质量和安全，避免了因急于求成而可能带来的风险。这种做法体现了一个重要原则：慢是快，即通过细致周到的计划和执行来实现更快、更稳定的长期发展。

可回滚原则

- 基础设施变更比较复杂，工程师容易忽略回滚策略

- 测试环境需要验证回滚步骤是否可执行
- 部分极端情况，可以将新建集群作为回滚方案
- 依赖快速构建环境的能力+数据备份+配置备份
- 典型的回滚策略
- 代码和配置文件都需要版本管理，杜绝仅回滚镜像
- 引入中间版本
- 谨慎删除旧字段

关于变更管理中的可回滚原则，这是一个在实践中经常被忽视的重要方面。在进行变更时，团队会制定一个回滚计划，但仅在测试环境或小范围灰度发布时未发现问题，往往不会去实际验证回滚计划的可行性。因此，携程云团队强调在测试环境中必须验证回滚策略的有效性。在一些极端情况下，比如之前升级 Ceph 集群时遇到的问题，集群无法启动，不得不重建集群并迁移数据，显示出在紧急情况下能够重建整个集群的能力是多么重要。

携程云团队推崇代码与配置的分离原则，在进行变更时，尽量避免同时修改代码和配置，避免定位困难。

另外，考虑，引入中间版本，在数据库变更中，最佳实践是先添加新字段，迁移数据，然后再删除旧字段，而不是直接修改字段属性。这样做可以减少回滚的复杂性和出错的风险。

总之，确保变更可回滚性不仅是为了应对可能的失败，也是确保整个系统可维护性和稳定性的关键措施。通过这些策略，能够更有效地管理技术变更，减少潜在的业务中断风险。

杜绝循环依赖

- 核心基础设施间的依赖需要定期梳理
- 核心基础设施自愈能力
- 网络&虚拟化控制器，做到无外部依赖自启动
- 容易忽略的依赖项：
- DNS
- 分布式存储

在管理基础设施时，强调减少循环依赖的重要性，因为这些依赖关系往往会导致系统复杂化和潜在的运维问题。为此，携程云团队定期梳理核心组件的依赖关系，并确保这些关键组件具备自愈能力。特别是在网络和虚拟化控制器这两个方面，历史上曾经经历过断电事件、电力恢复后，理论上业务应该立即回复，但实际上却因为这两个核心组件未能及时启动而导致整个系统无法运作。这一经历强调了底层系统的自主性和弹性的重要性，它们需要能够在没有外部依赖的情况下自启动。

此外，在进行技术变更时，经常被忽视的两个关键领域是 DNS 和存储。这些系统虽然在日常操作中看似稳定，但在变更过程中，任何对这些系统的忽视都可能导致严重的业务影响。

杜绝循环依赖

梳理基础设施管理服务，定义 P0 / P1 / P2 服务标准

针对 P0 业务，建立定期风险 review 机制，review 内容包括：

- 故障爆炸半径
- 业务架构
- 业务部署规范
- 业务故障预案

针对 P0 业务，进行故障演练，验证是否满足服务 P0 级定义，并建立常态化演练机制

为了确保核心应用和基础设施的稳定性和可靠性，携程云团队建立了一个详细的风险评估机制，并对服务进行了等级划分，包括 P0、P1 和 P2 级。这些服务级别的定义基于过去几年中对系统故障影响的分析，其中 P0 级指的是对业务影响最大的系统。

对于 P0 级服务实施定期的审查，包括几个关键方面：

故障爆炸半径：评估单个故障点对系统的影响范围，确保高可用性，即单个可用区（AZ）的故障不会影响到其他区域。

业务架构：分析业务依赖的系统和组件，确保这些依赖项的 SLA 也符合高可用要求。

部署规范：检查部署的规范性，确保所有操作不会发生预期外的行为和影响。

标准操作流程手册（SOP）：经过定期演练的标准操作流程手册 SOP，确保应用的所有者或管理者在发生故障时，可以依据 SOP 迅速恢复服务。

此外，携程云团队每年会进行实际的突击演练，例如选择性断开一个机房的网络连接，来测试其他机房是否能够独立承担起 100%业务流量。这种演练帮助携程云团队验证系统的实际表现与预期是否一致。

除了这些集中演练外，携程云团队还鼓励应用的所有者通过混沌工程的方式，定时注入模拟的网络故障（如断网、丢包、网络延迟增大），不断验证并提升自身应用的韧性，确保真正满足 P0 级服务的要求。这种主动的风险管理和应急演练是确保业务连续性和系统稳定性的关键措施。

（1）基础设施变更计划 - 日常运维变更实施方法

云组件部署机器调整、底层服务参数调优、DR 调整、域名及访问入口调整、备份机制优化、日志轮转调整、内核参数变更等 IAAS 组件日常运维管理相关变更，适用于如下变更流程及变更原则：

1）生产环境计划内变更，需要在每周四的周会前提交 Release Plan，周会上 review 变更计划。采用一票否决制，有任何不确定因素及反对意见的，变更搁置；

2）变更计划需要有明确的回退步骤；

3）对生产环境应用有一定概率产生网络中断等影响的变更，需要安排 outage 窗口、通过评审后实施；

4）对发布系统有较大概率产生影响时间大于 15 分钟的变更，提前一天做好用户通知报备工作。

基础设施变更模板

- 变更计划由变更委员会 Review
- 关联 Review 通过且 Merged 状态的 Merge Request
- 详细的 Rollout & Rollback Plan
- 需要在发布窗口期内变更

- 实时通知 NOC 变更状态



@sys-ccb （注意：部分项目是 @ sys-dp）

产品

- 填写变更的项目名称（必填）

Changelog

- 填写变更内容（必填）

merge_request

- 填写 merge_requests 链接 （必填）

Migrations

- 根据是否需要做 migration 填写

关联系统

- 影响到的其它系统

发布环境

- 填写发布的环境

Rollout Plan

- 填写发布操作步骤 （必填）

Rollback Plan

- 填写回退步骤 （必填）

发布窗口

- 填写计划发布的时间

实际开始日期

- 填写实际开始发布的日期（必填）

实际结束日期

- 填写实际结束发布的日期（发布周期大于一天时，必填）

预计发布时间

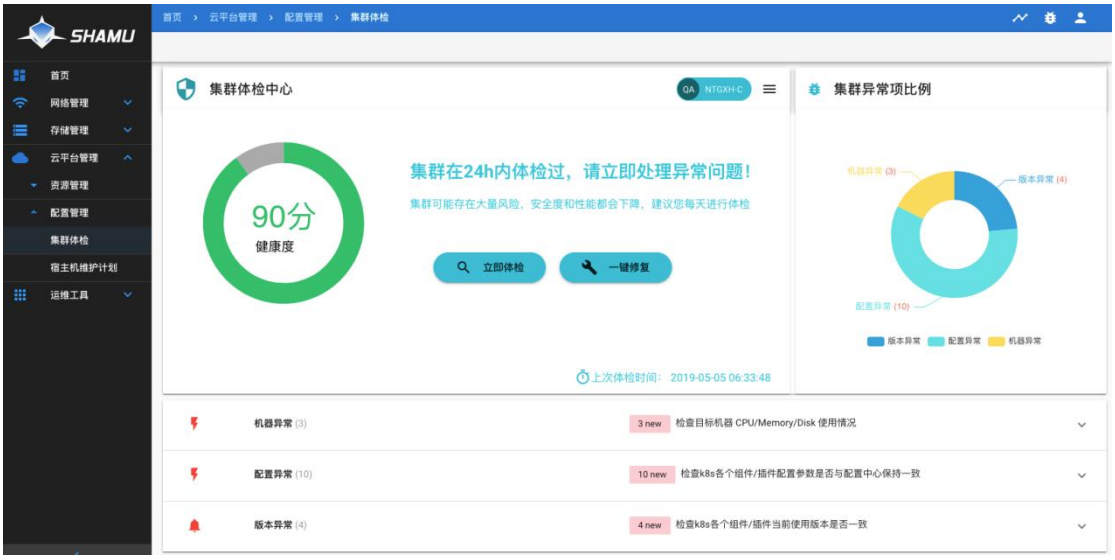
- 20 分钟

发布状态

- [ ] 发布中
- [ ] 成功
- [ ] 回退

携程变更模板示例

一致性巡检



配置不一致是万恶之源，为了解决这一问题，携程云团队投入了大量精力确保所有系统配置的一致性。携程云团队开发了一款基础工具，专门用于监控和检测宿主机上的 Kubernetes 组件配置是否一致。检查项不仅包括各组件的配置，还涵盖了内核参数以及各服务的配置设置。

该工具会定期进行巡检，并将检查结果反馈给相关团队，以便及时发现并解决任何配置不一致的问题。

基础设施变更计划 - 基础设施大版本升级

升级注意事项

https://docs.cilium.io/en/v1.12/operations/upgrade/#id3

1. cilium-operator/cilium-agent 默认 Prometheus metrics 端口号发生了变化

- cilium-agent 9090->9962
- cilium-operator 6942->9963

2. 在 Azure IPAM 模式下，--azure-use-primary-address 默认是关闭的，即不使用 ENI primary ip，aws --aws-use-primary-address 也是一样的，需要注意开启

3. 公有云 operator 能够支持通过 instance-tags-filter 配置过滤 EC2 实例

4. cilium-operator pprof 相关的配置进行了重命名

- pprof -> operator-pprof
- pprof-port -> operator-pprof-port

5. kube-proxy-replacement 配置里的 probe 选项被标记成废弃，将会在 1.13 中移除

- https://docs.cilium.io/en/v1.12/gettingstarted/kubeproxy-free/#kube-proxy-hybrid-modes

以上是 Cilium 组件 v1.11 升级 v1.12 的差异分析报告

要进行技术基础的大版本升级，需要非常的谨慎，以下是升级过程中的关键点。

变更内容概述：首先明确升级中哪些内容发生了变化。

参数更新：列出与之前版本不同的参数设置。

功能新增：介绍升级后新增的功能。

功能减少：说明此次升级中减少的功能。

在实施变更时，携程云团队采用持续部署（CD）模式，确保每项变更首先在测试环境中验证，并进行回滚测试。之后，变更会在生产环境中逐个可用区（AZ）实施，每次变更后观察一天以确保稳定性，然后继续在其他 AZ 执行。

另外，如 ECTD 版本升级，K8s APIserver 版本升级，也可以遵循此逻辑。

这样的管理策略旨在确保每次技术变更都是可控和安全的，同时通过严格的测试和分阶段部署来最小化风险。

（2）IaC 实践：StackStorm & SaltStack

为了保证配置和变更的一致性，携程云团队使用了多种配置管理工具，如 SaltStack 和 StackStorm，来管理配置变更。通过 SaltStack 管理基础组件的配置，并通过 Git 追踪配置版本的变化。为了增强系统的安全性和稳定性，携程云团队要求不同的可用区之间不得共享配置文件，这样即使发生配置错误，也只会影响单一的 AZ，而不会波及整个系统。

以下是我们的一些实践

通过 SaltStack 管理云平台基础设施服务

SaltStack Formula & Pillar 机制，标准化配置和敏感信息分离

不同 AZ 不共用 SaltStack Master，缩小变更操作的爆炸半径

Cilium Agent 不依赖 K8s DaemonSet, 用 SaltStack + docker-compose 管理 Cilium 服务

通过 SaltStack 管理云平台基础设施服务

使用 StackStorm 提供的事件驱动和 runbook 技术栈，实践 IaC

Chatbot & Stackstorm 工作流对接

机器配置更新依赖 SaltStack 提供的能力

对 Stackstorm 执行情况进行深入分析，持续优化运维变更的 workflow

（3）IaC 实践：云基础组件变更管理 GitOps

Layout structure

The layout of configs as below

```
- base
-   | appnameA
-   | appnameB
-   | appnameC
-   ...
- overlays
-   | clusterA
-       | appnameA
-       | appnameB
-   | clusterB
-       | appnameA
-       | appnameB
-       | appnameC
-   | clusterC
-       | appnameC
-   ...
```

代码分层结构

使用 Kustomize 来管理多个集群和环境的基础组件 Yaml 配置文件

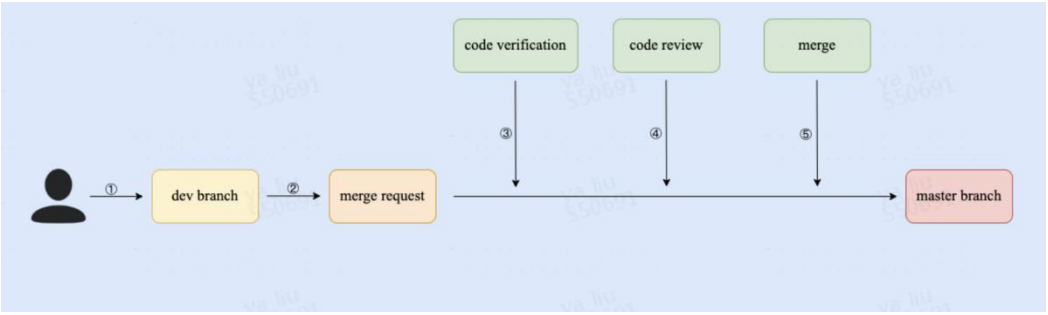
使用 Git Workflow（Tag/ MergeRequest）进行版本控制

基于 GitlabCI，CI/CD Pipeline 将基础组件部署到集群中

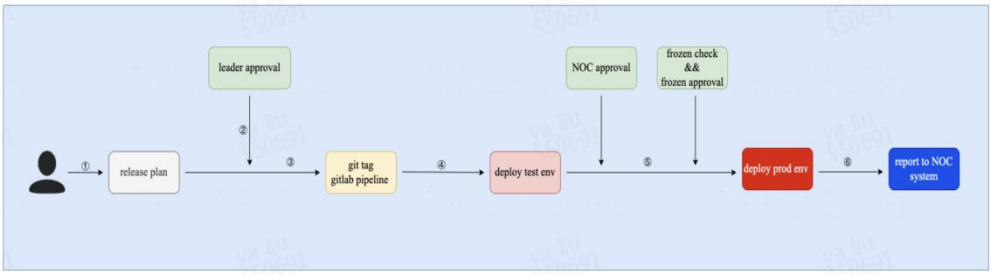
随着云原生技术的推进，携程云团队将更多基础组件转变为 Operator 化部署，几年时间里从最初的十几个增长到现在的五六十个。这些 Operator 的发布和变更管理，采用了 GitOps 技术来落地。

在携程云团队的实践中，所有组件统一存放在一个主仓库中，然后根据集群的维度对每个组件进行 layer 拆分。携程云团队使用 Kustomize 来管理不同集群和环境下的基础组件 Yaml 配置文件，确保配置的灵活性和可维护性。

版本控制方面，携程云团队采用 Git Workflow，包括使用标签（Tag）和合并请求（MergeRequest）来管理代码的版本。这样的配置管理（CM）模型较为简单：开发人员开发完毕后提交代码，进行代码检查和审查，之后提交发布计划供上级审批。



CI 流程



CD 流程

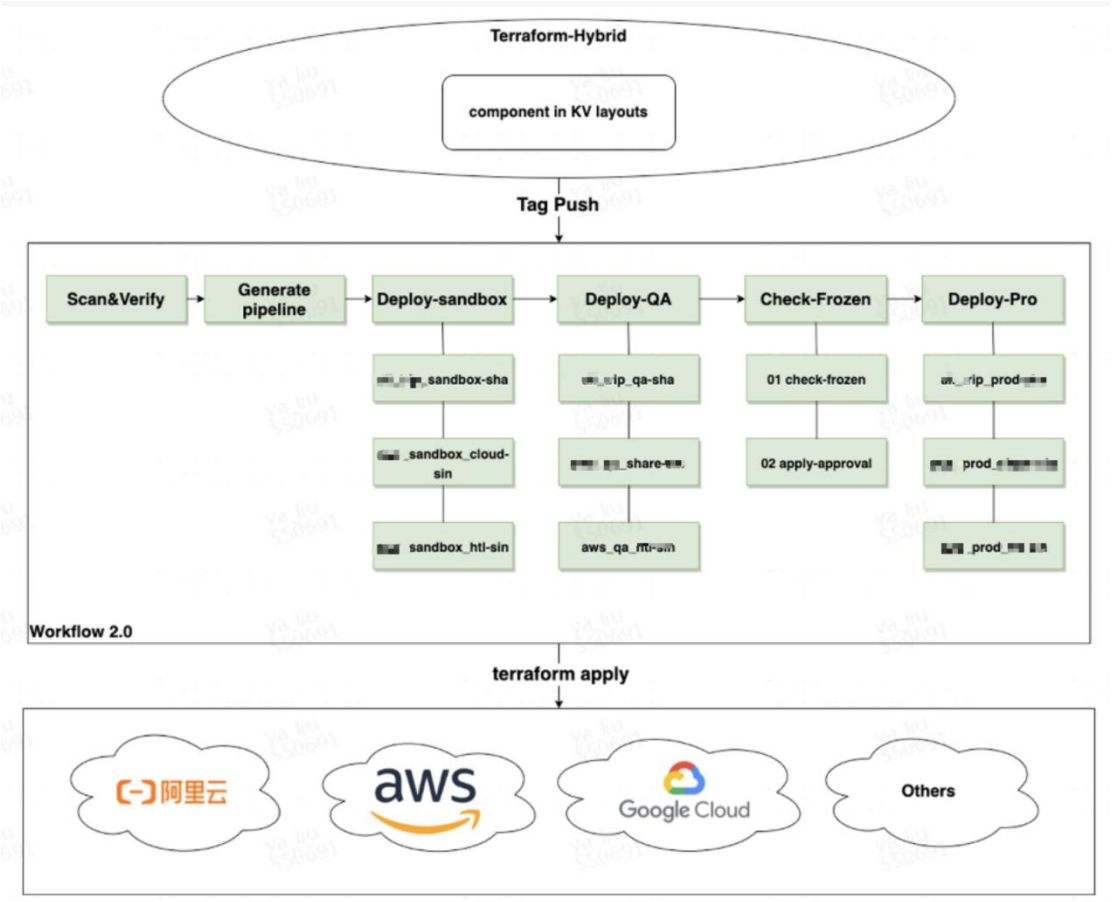
在自动化部署方面，携程云团队基于 GitlabCI 实施 CI/CD Pipeline，将基础组件部署到指定的集群中。在发布过程中，除了需要上级的审批，还必须确保发布操作在预定的维护窗口内进行。最终，产品的部署严格按照单个可用区（AZ）逐一进行，确保发布流程的安全性和可控性。

这样的策略通过分步骤、逐一区域的方法降低了整体风险，确保了系统的稳定性和安全性。

（4）IaC 实践：Terraform

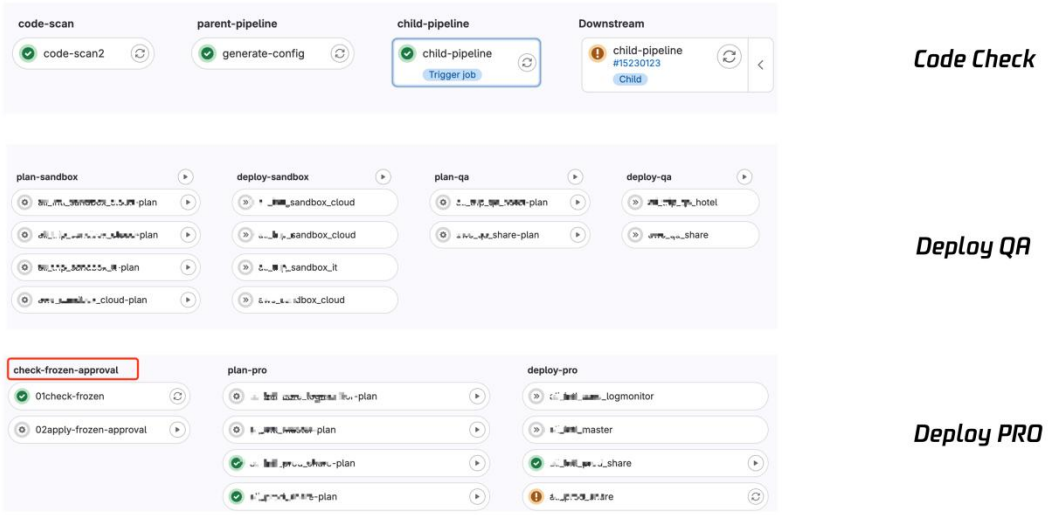


Terraform 目录结构



Terraform 部署流程

携程云团队在公有云资源管理中采用了 Terraform，并结合 GitOps 进行管理。这种方式与之前讨论的 Operator 管理类似，携程云团队按照组件的维度来组织 Terraform 配置文件的目录结构，这样做可以有效避免多人同时开发时产生的冲突。



Terraform Release Pipeline

以上的 Terraform 的工作流程包括几个关键步骤：首先是代码检查，确保配置的正确性和最优化；接着是在 Sandbox 和 QA 环境中进行部署。Sandbox 环境的使用场景主要是在引入新资源时，不应直接在测试环境部署，而应先在 Sandbox 环境中进行验证。验证无误后，再将资源部署到测试环境和生产环境。

在生产环境部署时，携程设有一个“冻结审批”机制。如果部署在维护窗口期内进行，通常可以直接通过；如果在窗口期外，则需要直属领导的审批才能执行部署。这样的安排旨在确保部署的安全性和责任明确，以便在出现问题时能有明确的责任归属。

（三）总结及展望

切记灰度，切忌不做回滚方案，面向灾难设计架构

AI + ChatBot 可以承担更多的日常工作，对于变更可能造成的风险进行智能化提示

对于核心组件定期 Review，主动升级，避免跨多个大版本的升级操作

不畏惧基础设施变更的前提：

具备持续变更能力

对于基础设施有深入的研究和理解

实践出真知

首先，灰度发布是基础设施组件变更不可或缺的环节。在逐步推出新功能的同时，降低风险可能带来的影响范围。其次，必须制定并验证回滚方案。这确保在遇到问题时，变更操作人员可以迅速恢复到之前的稳定状态。

此外，基于 AI 技术，比如聊天机器人，可以帮助工程师进行更多的风险分析和提供智能化的提示。这些工具的应用能够更有效地管理和预测潜在的问题。

第三，对核心组件的定期审查和主动升级非常关键。工程师应避免跨多个版本进行升级，就像之前遇到的从 Kubernetes

1.3 升级到 1.19 过程中遇到的各类挑战。目前，团队面临的挑战是如何将 1.19 升级到 1.28，因为长时间未升级导致积累了许多待解决的 bug。

在进行基础设施变更时，持续的变更能力、深入研究和理解基础设施是必需的。此外，实际操作比理论讨论更为重要，故障演练是一种很有效的风险识别方法。

在人为因素不可避免地影响变更时，工程师应当正视错误，而不是试图掩盖。作为变更执行者，如果出现问题，应该坦诚承认。同时，作为领导者或管理者，应当避免简单责怪下属，而应反思系统设计是否足够健壮，以减少人为错误的发生。

4.3.3 某银行变更管理设计与实践

SRE Elite 收录点评：

这是一个银行变更管理的案例，需要在符合监管的前提下保障可靠性，并对风险控制进行了量化。案例定义了应用和基础设施等变更风险的评分模型和积分制度，并结合评审进行准入控制。

该银行业务变更涉及多方企业人员关联，流程环节复杂，对变更管理所涉及的过程，设立了配套的组织架构闭环治理，

例如变更自动化率、耗时、风险评估准确性及流程审批效率等。

(一) 背景及设计原则

背景及挑战

通常银行业务信息系统的业务研发团队和 SRE 团队分属于不同的部门，对应的持续集成（CI）和持续交付（CD）也是由两个团队分别建设，中间通过流程管理平台（ITIL）联动，任何变更，通过审批后，才可以部署到生产环境。

相比于互联网行业以快速响应市场变化和用户需求为核心，强调技术敏捷、系统安全和数据保护的变更管理，银行行业受到金融监管机构的严格监管，银行信息系统的变更发布关系到银行业务的连续性和系统稳定性，因此，传统银行的变更管理通常参照 ITIL 标准开展实践，以变更流程的合规性、变更风险管理的谨慎和严格为主。

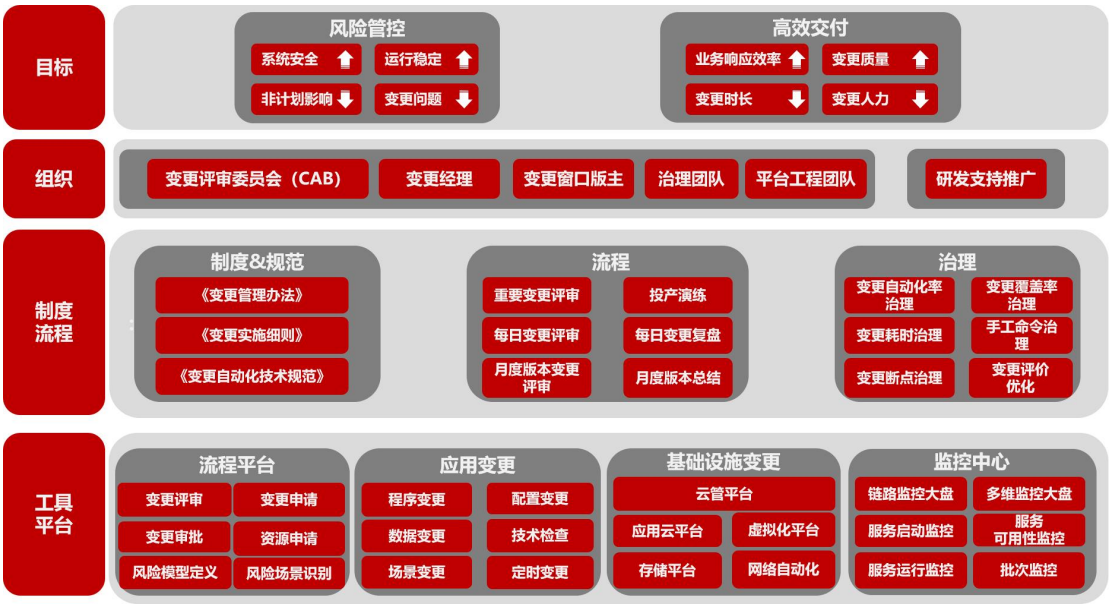
然而，随着互联网金融的兴起和商业银行数字化转型的加速，银行业务部门对业务系统的变更管理提出了更高的要求。他们期望变更管理不仅要确保变更的稳定性和安全性，同时也要能够快速响应市场和用户需求。这种“既要稳、又要快”的

双重要求，无疑给银行信息系统的变更管理带来了更大的责任和挑战。

面对这种挑战，银行需要在保持流程合规的基础上，利用先进的技术手段提高变更自动化水平，提升变更的效率和质量，同时，进一步优化变更管理流程，实现变更风险线上化、数字化，强化变更风险的识别和控制，以满足不断变化的市场和用户需求，支持业务快速创新。

设计原则

某银行生产环境，约 700 多个应用，3 万多台 OS，2 万多个容器。从 2013 年开始进行应用变更自动化的建设，最初目的是减少手工操作性工作。**平台能力从最初的应用程序更新自动化，逐步扩大到配置更新、数据更新、技术检查自动化，再到与流程管理、配置库、监控等生态平台的联动，资源交付自动化。**当前，变更管理以变更风险控制、高效交付作为变更建设的主要目标，通过组织、流程、技术平台的持续演进以及常态化的变更治理实现对变更全生命周期的闭环管理。

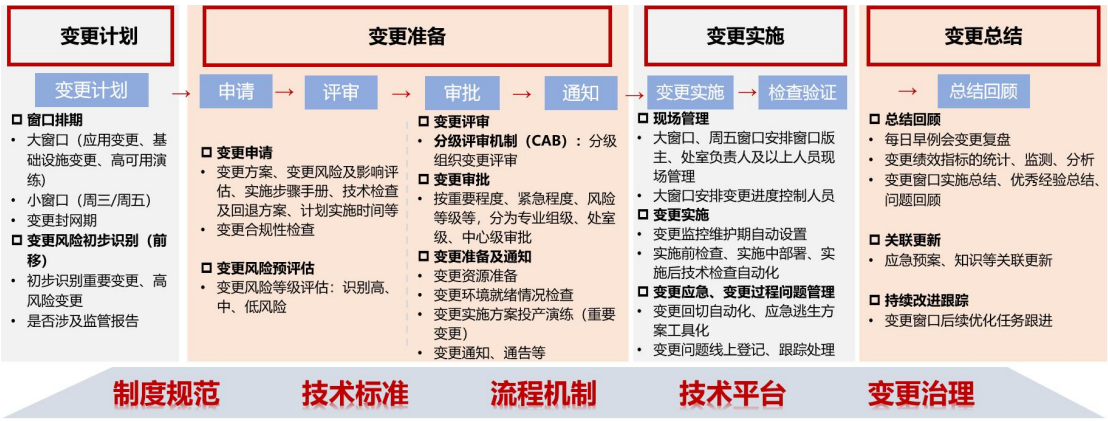


某银行变更管理总体思路和设计如上图所示

1. 变更组织围绕变更的核心业务流程进行设计，由数据中心牵头组建，研发中心支持推广团队（注：研发支持推广团队是数据中心与研发中心融合的桥梁）共同参与，主要职能是制定变更相关的制度、规范，组织变更计划制定、评审、总结，进行变更管理相关工具平台建设，开展变更相关的治理活动，驱动变更流程的高效、高质量执行；
2. 变更相关制度流程在满足监管要求的前提下，主要目标是指导变更管理、变更治理工作开展，制定变更流水线编排等技术实施规范，是变更技术实现的补充；
3. 工具平台则是以风险管控自动化、变更执行自动化、检查自动化为目的，并尽可能将管理手段转变成技术实现。

（二）体系设计详情

某银行将变更流程分为计划、申请、审核、审批、通知、实施、验证、总结 8 个阶段，并围绕这 8 个阶段重点开展变更风险管控以及全流程自动化治理。



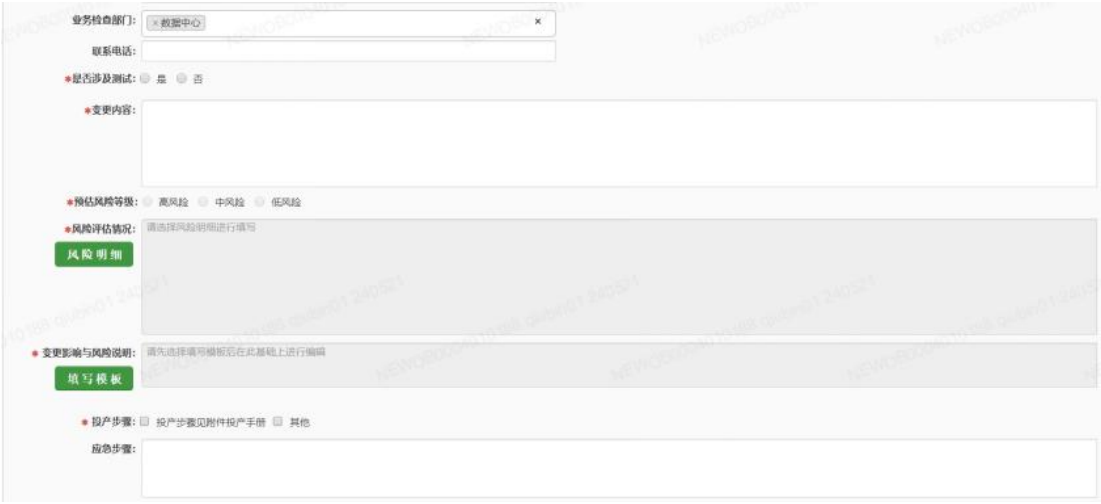
下面主要介绍变更风险管控、全流程自动化的建设及管理思路：

（1）变更风险的识别和控制

1.变更风险清单化、场景化、线上化管理

通过定义应用变更、基础设施变更风险评分模型，并纳入变更申请的风险预评估环节。对于重要变更，还需要对变更准备情况、变更准入条件、非功能性需求进行评审。

下图为变更申请工单风险评估部分，预估风险等级根据风险评估情况自动生成。



应用类变更内容的往往更加复杂、动态，因此，应用变更的风险预估模型适合采用积分制。某行从**系统重要性、变更自动化程度、部署方式、投产时长、与其他变更的关联度、变更影响范围、高风险变更内容**等 7 个方面进行风险预评估。下图为应用变更风险评估预览：



基础设施类变更内容比较标准、固定，但容易发生全局性影响的生产事件，适用将风险评估场景化，方便评审委员会、

审批人员快速判断变更风险等级。下图为基础设施变更风险场景预览：

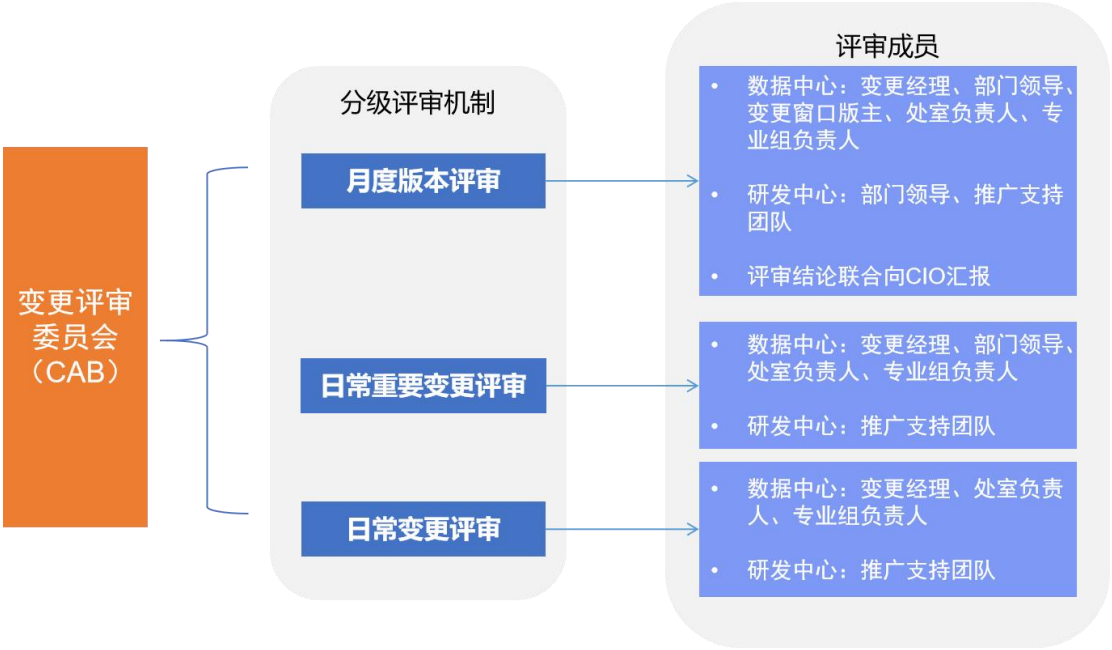


针对重要变更，某某银行在变更申请阶段增加了针对重要变更的变更准备情况、变更准入条件、非功能性需求实现情况的评审。按照变更分类分为应用类、系统类、机房类、网络类、安全类 5 个大类 80 多个子类，每个子类评审要点分为 A、B、C 三个级别，A 级属于严格执行要求，评审未通过则不允许申请变更。下图为重要应用变更评审要点预览：



2.组建变更评审委员会（CAB）

由数据中心 SRE 团队和研发支持推广团队共同组成，按照变更分级评审机制组织开展变更评审。评审内容包括变更材料、变更方案、变更预评估风险及变更影响。



3.应用变更风险管控方案

某某银行应用变更通过变更风险评分、制定新建准入、与研发建立联合变更管理机制，同时，采取变更窗口集中管理、变更时点控制、投产时长控制，避免发生重大业务影响的生产事件，并且通过 SRE 左移落地，确保非功能性运维需求的实现，确保业务信息系统投产后的可靠性、可运维性。



4.基础设施变更风险管控方案

基础实施类变更重点是严防严控基础设施变更引入风险，避免发生全局性业务影响的生产事件。某行对基础设施类变更按场景进行风险等级分类，标准化基础设施变更手册、严格要求配套变更应急措施/逃生措施，并严控基础设施小变更。



（2）变更自动化与治理

1.应用变更自动化

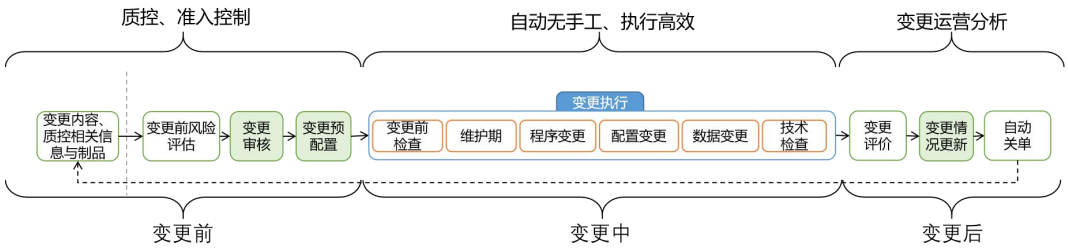
银行业务系统涉及的技术栈多且复杂、三方系统关联多，变更场景复杂，是一个多关联方、多流程、多环节的系统性工作。应用变更除了要面对风险引入控制问题，还要面对投产时间长和工作强度高两方面痛点。表层原因主要是自动化程度不足、投产中断多、手工检查验证多，集中体现在【变更中】环节，根源多产生于【变更前】，包括研发侧引入的制品质量问题、脚本质量问题、部署流水线质量问题；环境因素引入的环境变量、用户/网络权限、文件系统容量问题；自动化平台引入的代理可用性、变更自动化能力、检查自动化能力问题。以下是某银行全流程自动化应用变更实践：

1）**变更流程联动：**通过建立流程审批、制品库、应用变更自动平台，API 网关之间的联动。运维管理平台接受研发管理平台提交的变更申请，并同步变更制品；应用变更自动化定

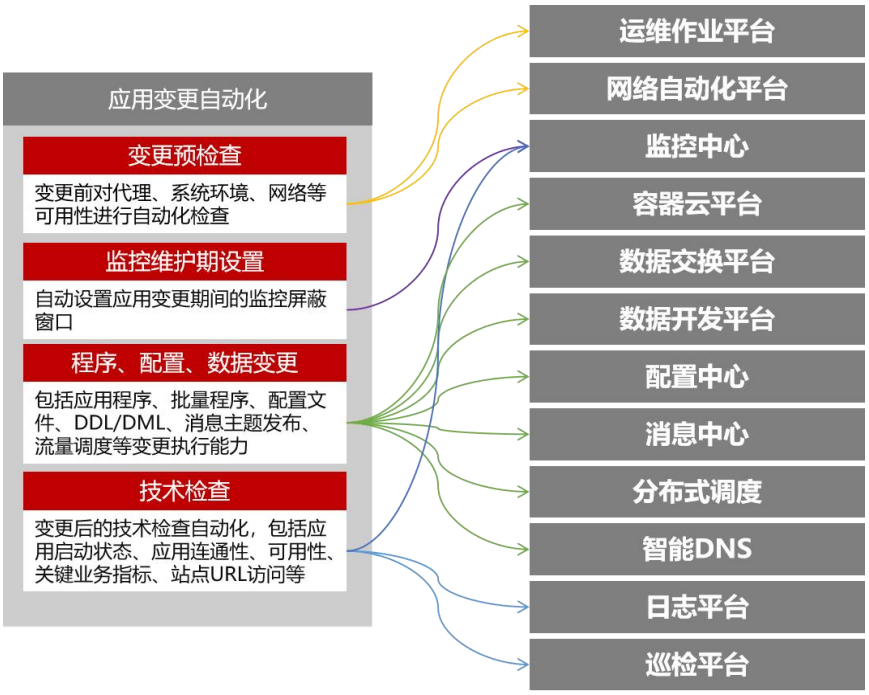
期同步运维管理平台状态为“实施中”的变更清单，对于定时执行变更单，应用变更自动化平台根据变更执行计划定时发起，并在实施完成后通知 SRE，其他变更单，由应用管理员在应用变更自动化“变更工作台”完成变更单查看、发起、确认更新等操作。



2）生态对接实现变更执行自动化：应用变更执行过程由变更前检查、监控维护期设置、程序变更、配置变更、数据变更、技术检查等 6 个环节组成。



6 个环节的自动化执行，主要通过自动化变更平台本身的流程引擎、代理执行能力，以及通过 API 网关与其他运维支撑平台的生态对接，实现平台自动化能力的扩展。



① 变更预检查自动化：变更发起后先进行预检查，在变更前提前暴露问题、解决问题，减少变更过程异常中断，有利缩短变更耗时。

	检查内容
代理	代理是否在线
	代理权限是否正确
系统环境	指定路径是否存在
	数据库密码是否准确
	环境变量是否存在修改
	临时下载目录空间是否足够
网络权限	平台到目标服务器网络连通性
	目标服务器之间连通性
	新增业务网络连通性
	第三方对接平台网络连通性

② 监控维护期设置：支持按默认维护期和人工选择个性化维护期，避免变更期间监控误报。



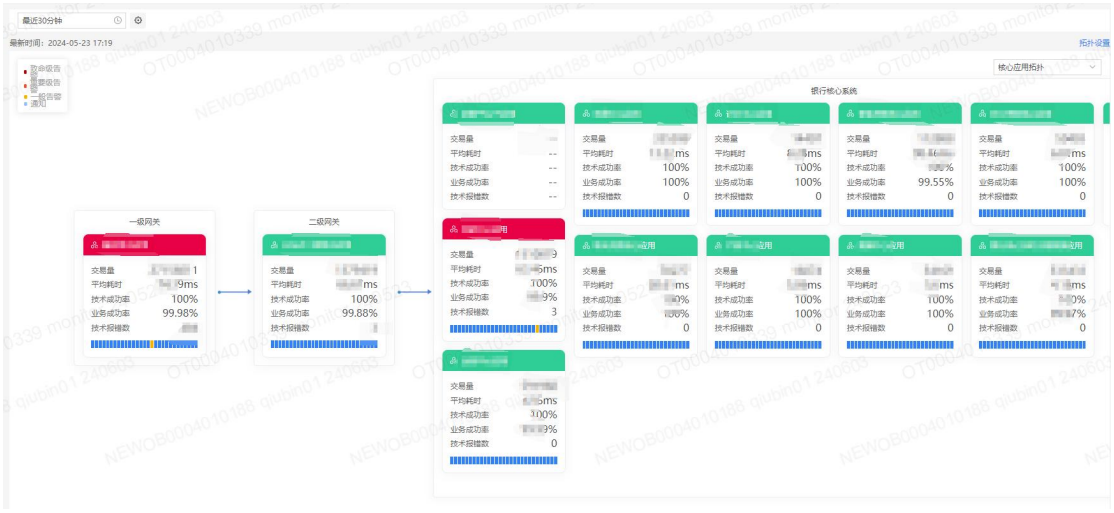
③ 变更执行过程可视化：包括执行进度可视化、异常可视化，此外，对于重要业务系统变更，还采用双屏变更方式，一屏用于变更操作、一屏用于业务运行观测。



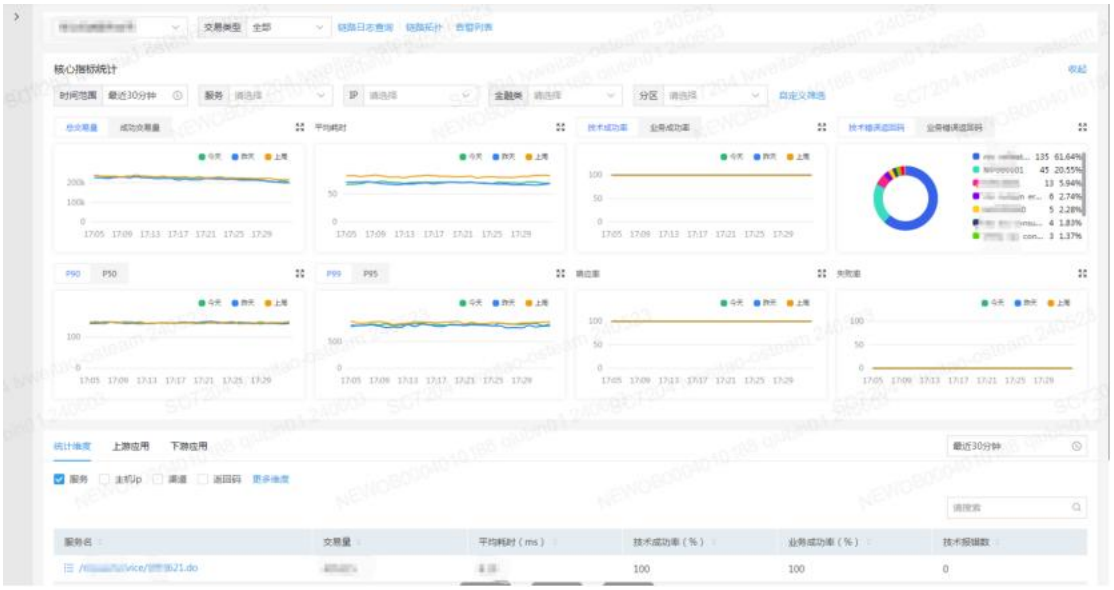
图：云上变更过程可视化



图：变更异常信息回显

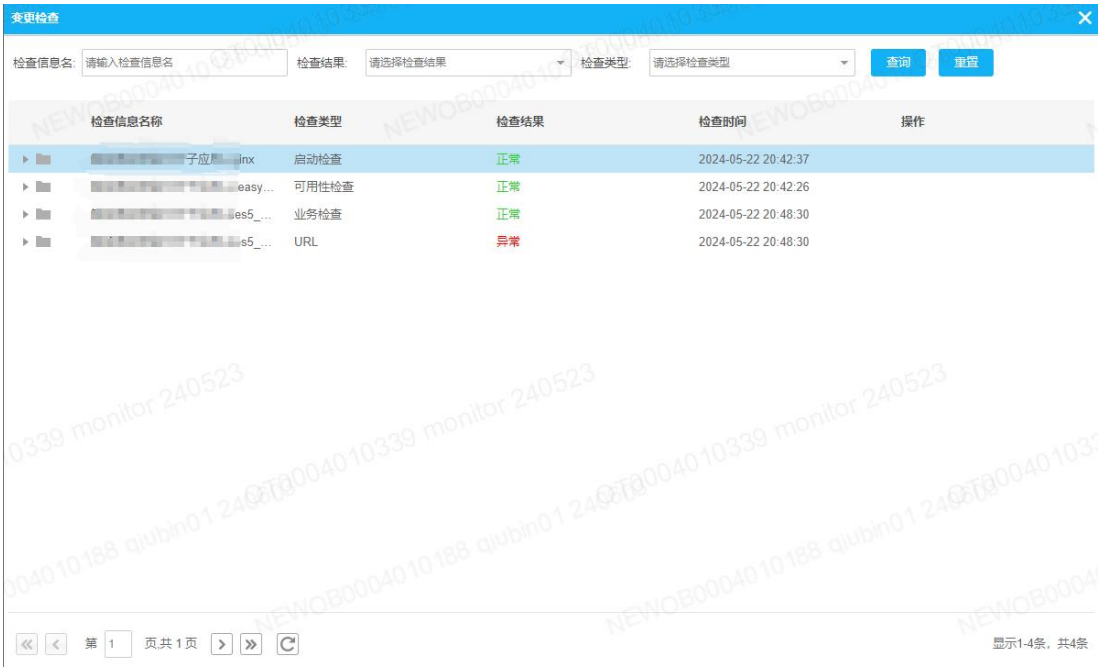


图：应用群监控大盘



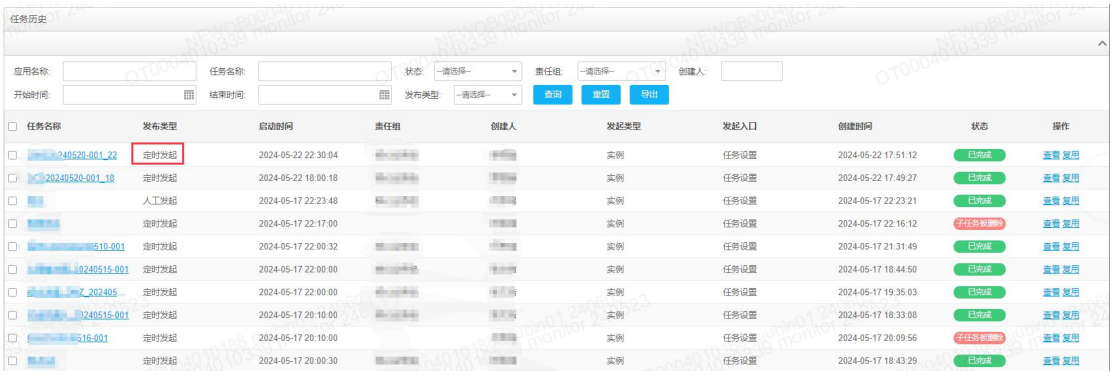
图：应用多维分析大盘

④ 技术检查自动化：通过平台自身代理，以及对接监控中心、日志平台、巡检平台实现变更技术检查项的自动执行。

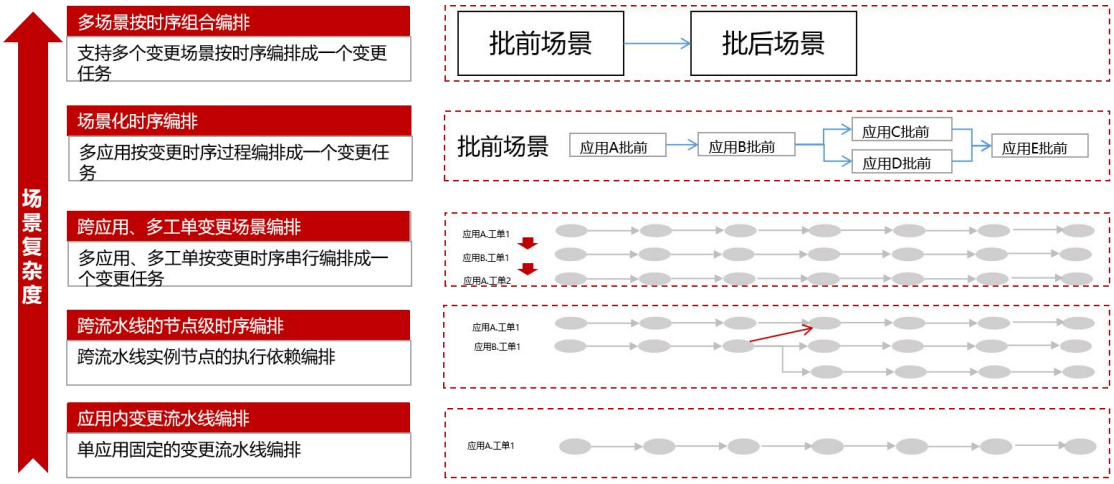


⑤ 变更任务定时执行：对于已实现全流程变更自动化的应用，支持配置变更任务自动发布，变更执行结果通过蓝信通知到 SRE。





3）灵活的变更策略支持：支持跨发布流水线的节点依赖、场景化变更，减少变更次数、加强变更时序控制。

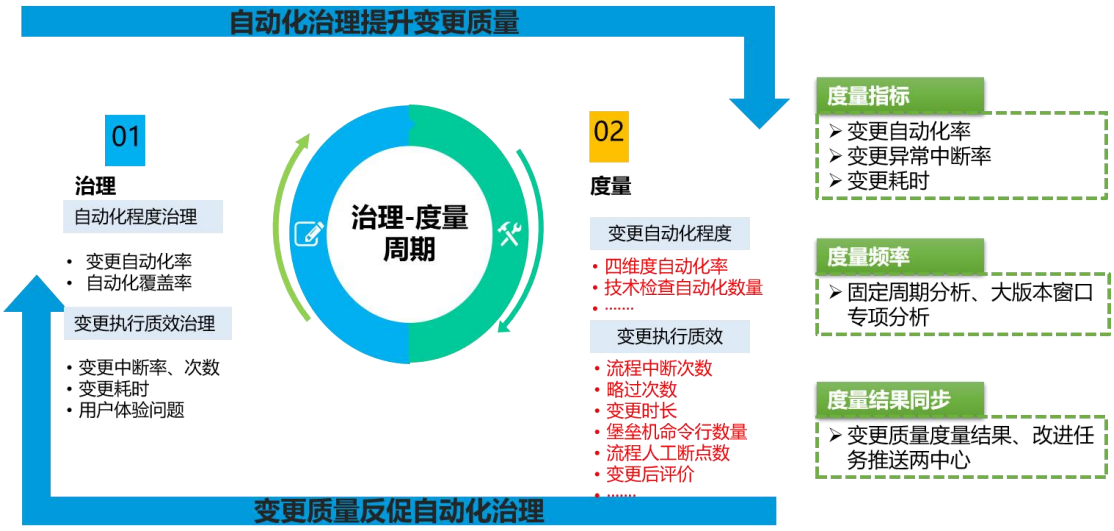


场景化变更实现国产化分布式核心应用群按批前、批后时序，单双机、生产同城环境分批部署，大大提升核心变更效率，缩短核心变更时间：



2.应用变更自动化治理

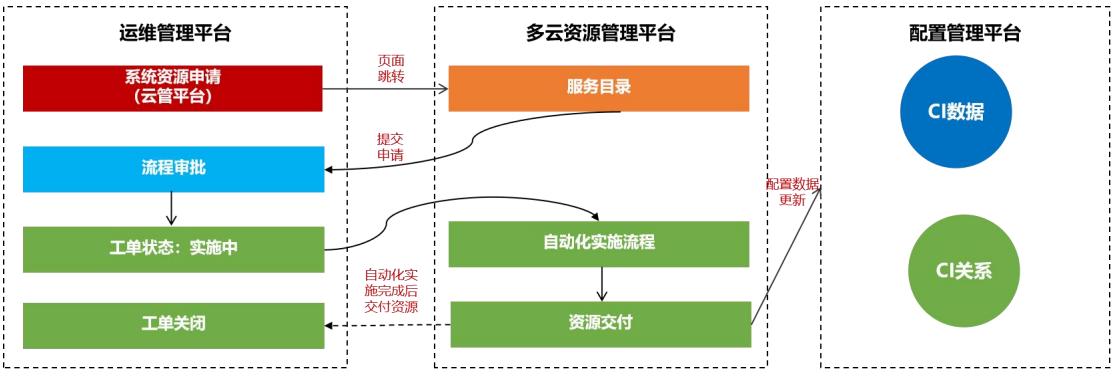
应用变更自动化的治理主要面向 6 个变更执行环节的执行自动化、异常中断、执行耗时以及变更后评价（变更执行相关用户体验、需求、问题等信息收集）进行治理，治理团队包括变更经理、平台工程团队、研发支持推广团队。



3.基础设施资源交付自动化

某某银行通过流程平台与多云资源管理平台、网络自动化平台对接，实现基础设施资源申请流程审批通过后自动交付、配置数据自动更新。

1）系统、软件、存储资源交付自动化



为实现资源的自动化和标准化交付，基于基础实施即代码（IaC）技术，使用代码为每一类资源定义标准的最小化的资源模块，实现资源的代码化抽象：

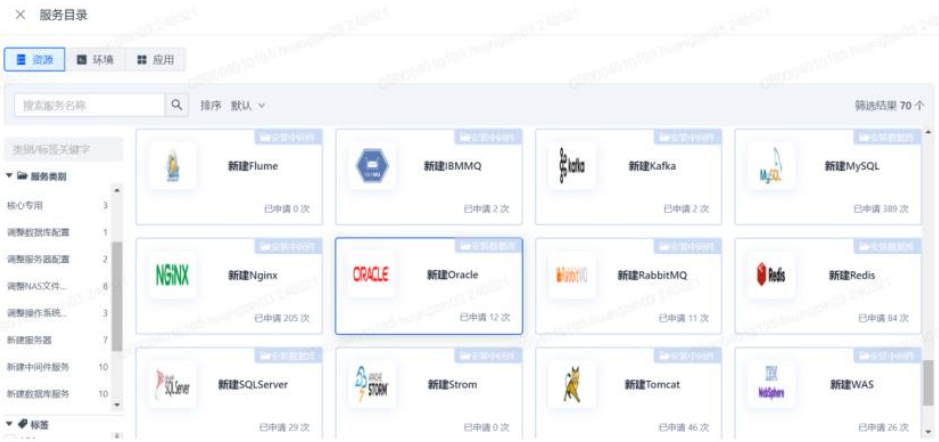
① 按照不同的资源类型建设标准的、可复用的资源模块，包括虚拟机/物理机、操作系统配置、代理、数据库、中间件、存储资源和网络资源等。

② 各资源模块分别管理，不同技术团队按标准配置规范负责对应组件部分的开发和调优，并进行统一管理。

③ 资源交付流程以服务目录的形式展现，通过对标准的资源模块进行组装，编排形成通用的部署模板，将部署模板映射发布为服务项，为用户提供资源交付的自助服务；

④ 按需组合资源模块，描述资源模块间依赖关系，实现复杂资源定义，满足任意丰富的业务场景需求。

⑤ 提供套餐化的自助服务，简化和标准化申请参数，提升服务体验。

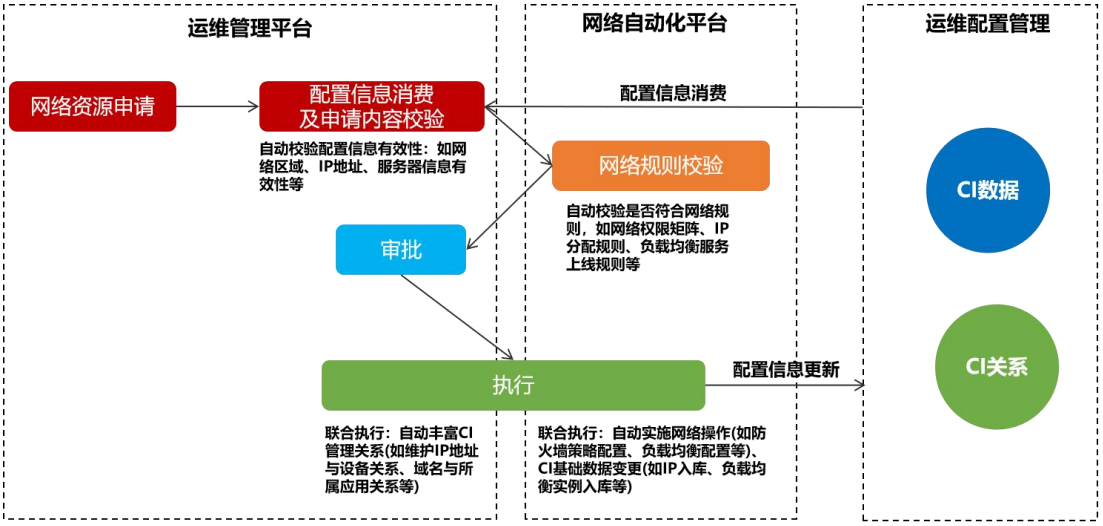


云管平台服务目录示例



容器云资源服务项交付示例

2）网络资源交付自动化



流程平台对接网络自动化平台、CMDB 实现网络资源申请工单快速填写，域名发布、IP 分配、网络访问权限、负载均衡服务上线等网络资源申请项的自动交付。

网络自动化平台支持任务编排、人工/自动执行、遇错停止、继续执行、结果回显、健康检查和备份前后对比、变更涉及设备的告警一键清理等功能。



网络资源、负载均衡自动交付示例

（三）总结及展望

变更是一个多关联方、多流程、多环节的系统性工程，在安全、数字化、降本增效为主题的当下，变更管理面临最大的挑战，其实是安全、效率、成本的既要又要，在未来 1~2 年，在变更管理方面，将持续深耕变更风险控制自动化、操作执行自动化、治理分析自动化，提升变更风险管控能力和交付效率，降低变更人力投入，推动业务信息系统的标准化、服务化、容器化改造，以此降低变更管理的难度和成本。

4.4 发布管理案例

4.4.1 中移互联网敏捷发布平台建设实践

SRE Elite 收录点评

1，“中移互联”这个名字就体现了传统行业与互联网的结合，传统 IT 模式与敏捷互联网业务间的矛盾，促使其从方向上选择卸下历史包袱，推倒零散的工具系统，一步到位建设一体化运维平台。并且对物理机、虚拟机、容器等不同类型的资源及上层应用设计了混合编排模式，为后续的综合算力调度做了储备。

2，在一体化运维建设之后，为了追求更高的质量效率，中移互联 SRE 团队向研发服务左移，保障了测试环境及生产环境的发布一致性，并且承建了研发工具链，从运维一体化扩展至研运一体化。

（一）背景及设计原则

中移互联网有限公司（以下简称“中移互联”）是中国移动集团公司面向互联网领域设立的专业子公司，目前运营着多款拥有亿级用户规模的产品，包括移动云盘、移动认证、超级 SIM 等。从业务上看，这些业务有持续迭代发版的内在需求，然而，各业务线的发布变更能力存在显著差异，手工与工

具并存，工具平台多样化，不仅效率低下，且容易出错，发布能力与业务发展水平不匹配。



为了解决上述问题，中移互联提出了推动构建公司级统一应用敏捷发布平台。该平台旨在实现传统主机和云原生环境的全场景自动化发布变更，构建覆盖测试、灰度和生产环境的一体化应用发布能力，从而推动发布工作从手工到自动化、发布工具从分散化到集约化的转型，提升生产环境的变更成功率和发布效率，为业务团队提供稳定、安全的发布保障，进一步推动公司的数字化转型和业务发展。

（二）体系设计及关键流程



敏捷发布平台基于蓝鲸底座构建，旨在提供 CI CD 集约化一体化的变更管理能力。

平台的架构设计包括以下几个层次：

资源层：涵盖分布在不同机房的物理机、虚拟机及各类的 K8s 集群。目前公司内有 8 个数据中心，包含：广州萝岗、南方基地、北京机房、移动云无锡机房等。

公司产品有全量接入移动云，使用 KCS 容器服务；部分产品接入浪潮 PaaS 平台，使用浪潮提供的 K8s 容器服务；中小产品则自建 K8s 集群进行部署运营。

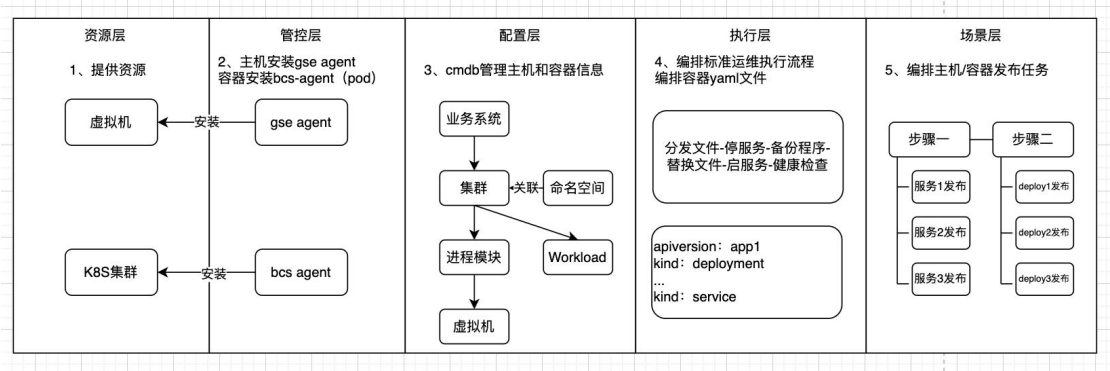
管控层：使用 GSE Agent 统一管理所有产品的物理机和虚拟机，命令下发、文件分发、脚本执行、数据采集都通过统一的 GSE Agent 去完成，规避了业务线各自使用不同工具和管理方式的弊端。

敏捷发布平台通过 BCS 将不同来源的 K8s 集群统一管控，并对外暴露统一的 BCS-API。这一做法一方面屏蔽了不同容器服务（如 KCS 接口或浪潮云 PaaS 接口）的 API 差异，避免了每个产品团队需要单独适配；另一方面也解决了不同版本 K8s 接口的差异问题，因为 BCS-API 本身兼容了不同版本的 K8s 接口。

配置层：统一管理主机和容器资产，并通过应用拓扑将二者关联。在发布变更时，只需选择目标业务系统和服务，系统会自动从 CMDB 拉取目标主机及 K8s 集群的 Namespace，提升配置的准确性和唯一性，降低变更出错的概率。

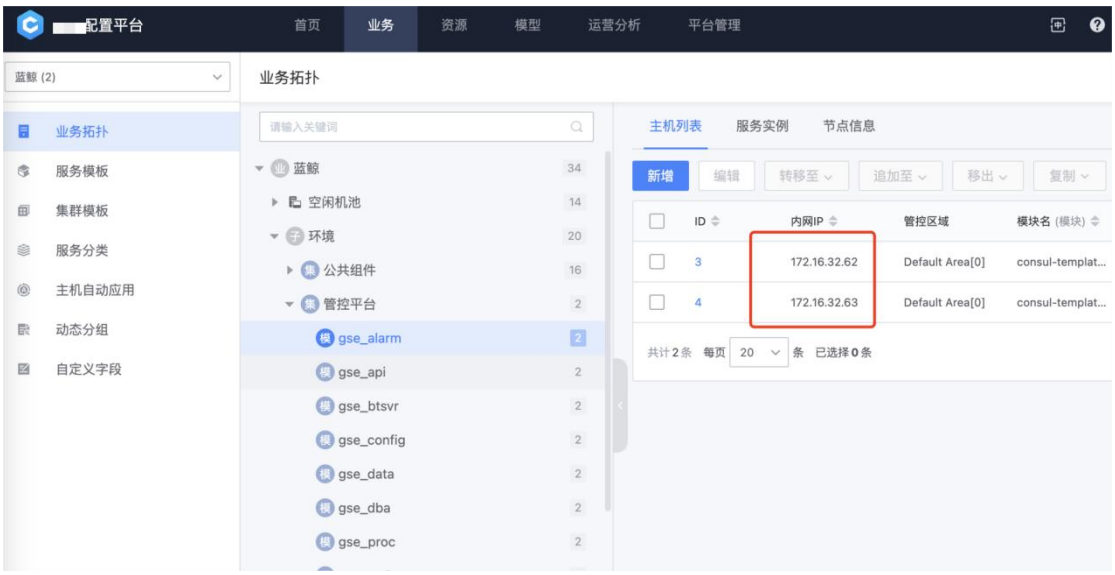
执行层：通过标准运维编排规范的通用执行流程，各业务线可以复用此流程完成传统主机服务的变更发布，并通过调用 BCS-API 完成容器服务的变更发布。

场景层：提供应用发布自动化统一编排任务，包括自动滚动分批发布任务、主机和容器混合发布任务、蓝绿发布任务等。在测试环境中，通过 DevOps 流水线自动完成编译、构建，并调用应用发布自动化接口完成发布；在生产环境中，集成 BPM 工单审批系统，生产环境的每次变更任务创建时会自动生成 BPM 审批工单，需运维领导审批通过后方可执行。



配置层统一管理主机和容器资产，并且通过一个应用拓扑将主机资产和容器资产关联起来。在发布变更时，只需要选择要发布的业务系统和服务，自动从 CMDB 拉取服务要发布的目标主机，以及目标 K8s 集群的 Namespace。统一的变更配置层，提升配置的准确性和唯一性，降低变更出错的概率。

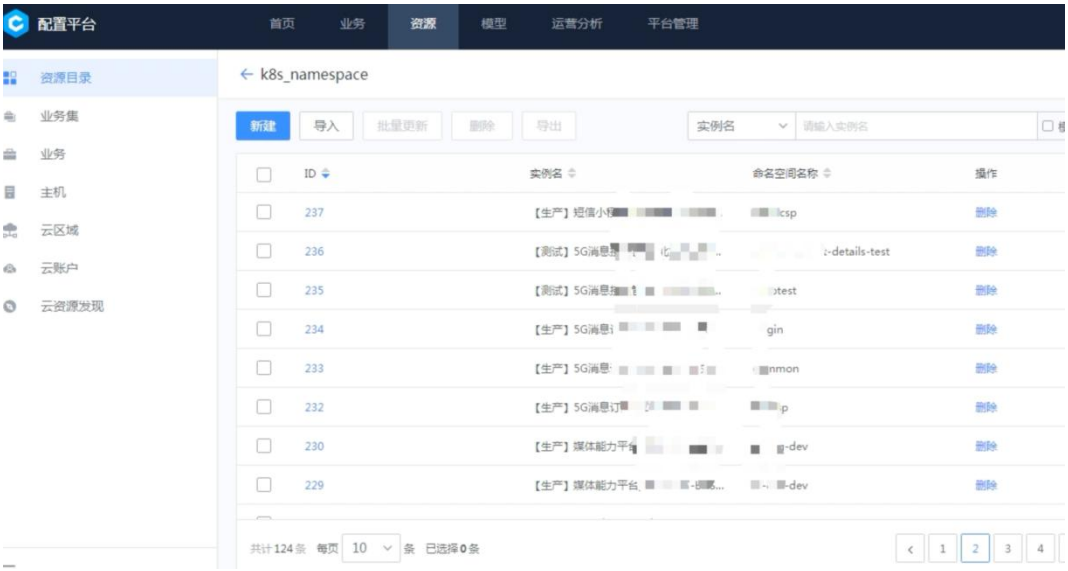
CMDB 维护传统应用拓扑，业务-环境-集群-模块的四层应用拓扑维护数据，管理下业务系统数量，每个业务系统下服务模块数量，每个服务模块部署在哪些主机上。在 CMDB 中维护好这些数据，业务方在发布时，消费这些数据，需求单独维护“应用”和“主机组”的关系。



针对容器应用，同样复用这套拓扑关系数据，利用 CMDB 中模型的关联关系，建立集群模型和命名空间模型的关联关系。



在 CMDB 中维护 K8s 的命名空间数据，方式是周期性调用 BCS-API 实时拉取 K8s 集群数据，做比对和更新。



Namespace 资源

将命名空间和应用拓扑中的集群层级关联之后，就可以用相同的应用拓扑管理容器资源，无需额外维护一套拓扑关系。



执行层通过标准运维编排规范的通用执行流程，每个业务参考复用此流程完成传统主机服务的变更发布，通过调用容器管理平台 BCS-API 完成不同容器服务的变更发布。

在场景层，则通过应用发布自动化统一编排发布任务，包含自动滚动分批发布任务、主机和容器混合发布任务、蓝绿发布任务。



虚拟机发布节点配置



发布 10 台主机，选择随机滚动分批

在测试环境，通过 DevOps 流水线自动完成编译、构建，调用应用发布自动化接口完成发布；在生产环境，集成 BPM 工单审批系统，生产环境每次创建变更任务，自动创建 BPM 审批工单，运维领导审批通过之后，才能执行生产变更任务。

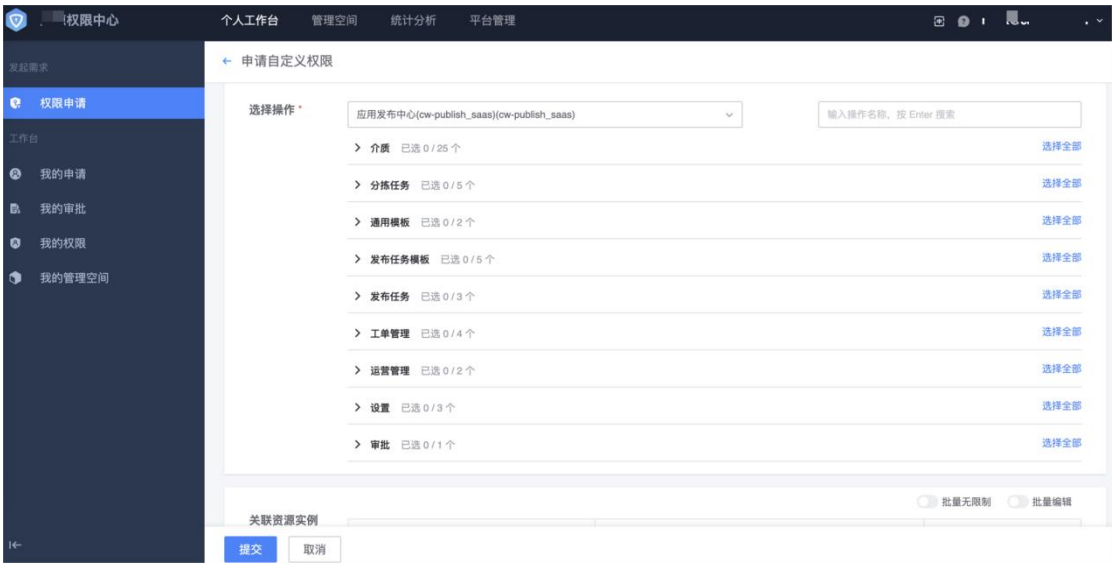
研发部和系统部密切协同，构建全网业务从需求、项目、代码、构建、制品、发布等端到端闭环管理，接入公司多个部

门 30+业务线，累计纳管近万台主机和 60＋K8s 集群，生产环境周发布近 600 次。

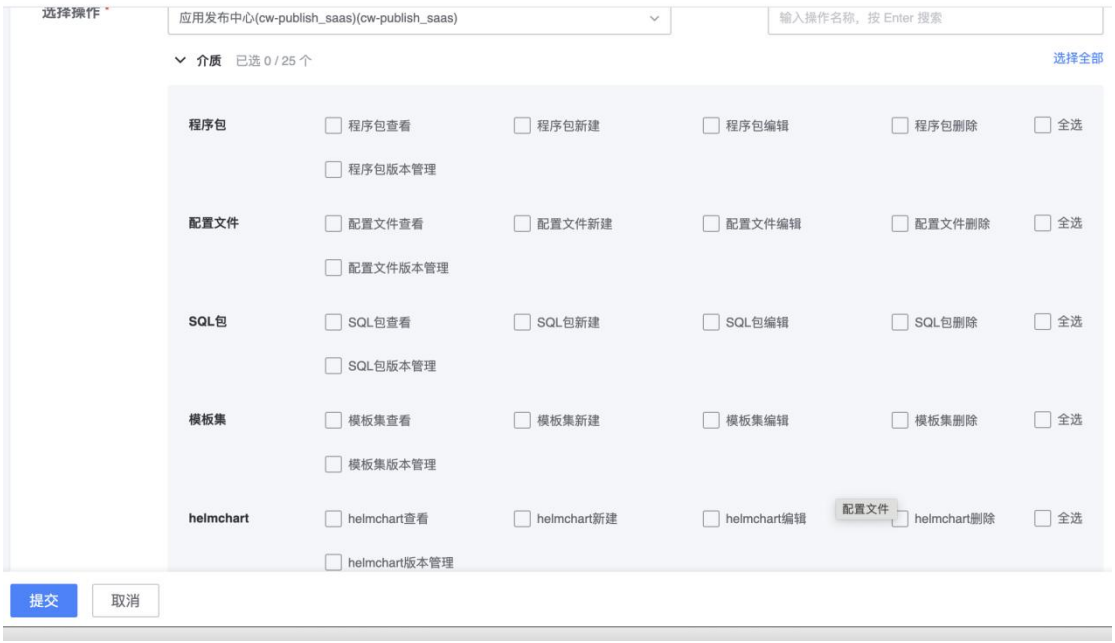
（1）统一权限管控

无论是自研战略业务，还是中小业务，统一使用敏捷发布平台，这样可以进行统一权限管控，严格控制不同角色的生产操作权限，极大收敛变更权限风险敞口。

将发布权限统一注册到权限中心，核心是介质、发布任务模板、发布任务三大模块。



介质模块包含：程序包、配置文件、SQL 包、模板集、Helm 包的增删改查和版本管理。这些操作的授权范围都是业务系统，即授权用户某业务系统下程序包的增删改查、版本管理权限。



发布任务模板和发布任务，核心是控制执行能力，授权范围也是业务系统，即授权用户某业务系统下发布任务执行权限。



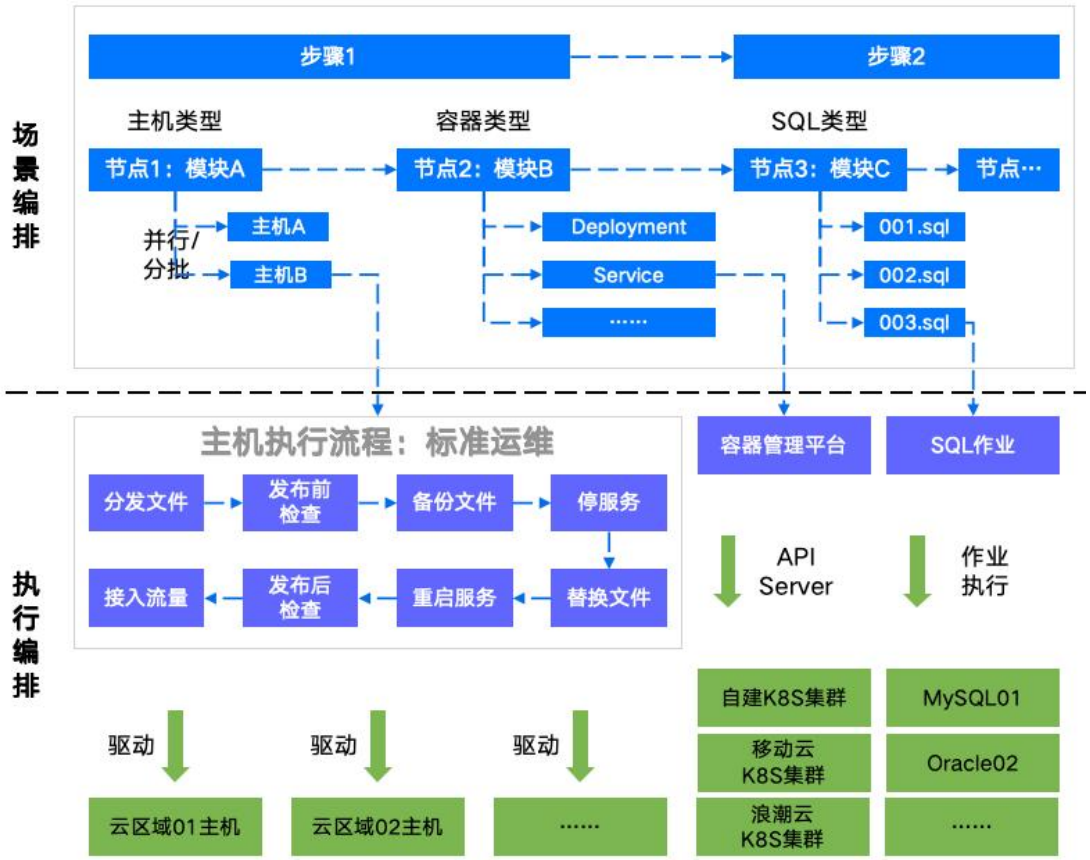
统一变更入口之后，可以对不同业务团队的变更操作进行约束，推动变更操作规范化通用化，降低发布失败率。

此外，统一不可删除的变更记录，既能快速定位问题，缩短变更异常的回滚时间，还能用于度量分析和复盘改进，以及能提供给公司审计。

业务	CI/条	构建次数	总构建次数	CD/条	发布次数	总发布次数	周生产环境发布次数	主机/台	K8S/套
[redacted]	420	504	14227	386	491	13399	118	253	0
[redacted]	560	283	5243	363	294	4679	113	48	5
[redacted]	718	161	13494	429	256	7998	80	623	2
[redacted]	111	300	1101	139	342	1248	74	545	0
[redacted]	187	347	6040	151	312	5473	47	108	0
[redacted]	231	333	4215	150	309	3325	37	188	1
[redacted]	998	1283	19867	670	988	13140	29	3642	29
[redacted]	100	50	3187	108	47	2595	10	24	0
[redacted]	110	49	1415	79	9	145	9	817	2

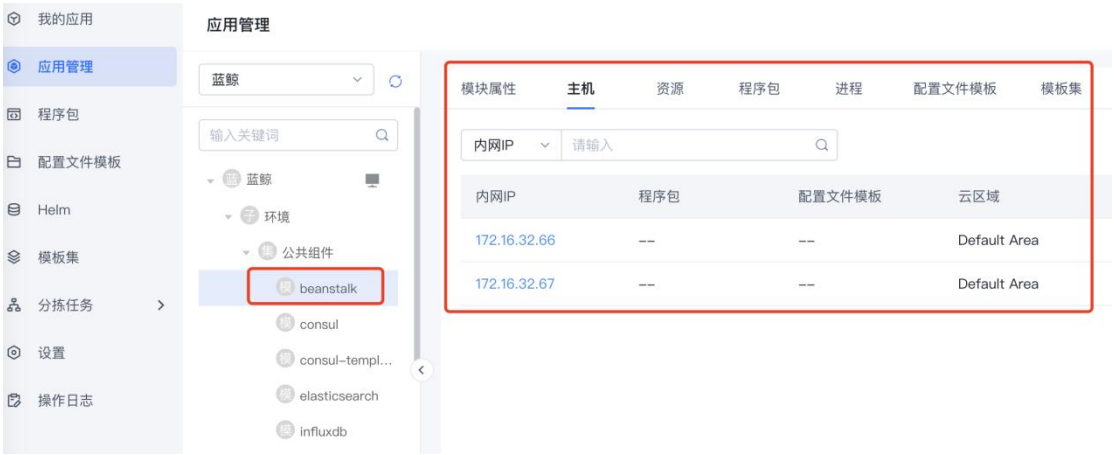
（2）多场景任务模板

中移互联目前处于云原生转型的过程中，但各团队进度不一：有些业务已全面云原生化，全部部署在 K8s 环境中；有些业务因历史技术原因仍运行在虚拟机上；还有一些业务处于转型中，部分服务在 K8s 环境，部分在虚拟机环境。为实现传统主机服务和云原生服务的变更一体化，敏捷发布平台通过集成标准运维和容器管理模块，提供一套兼容主机和容器任务的变更模板，屏蔽异构环境差异，提升生产环境变更成功率。

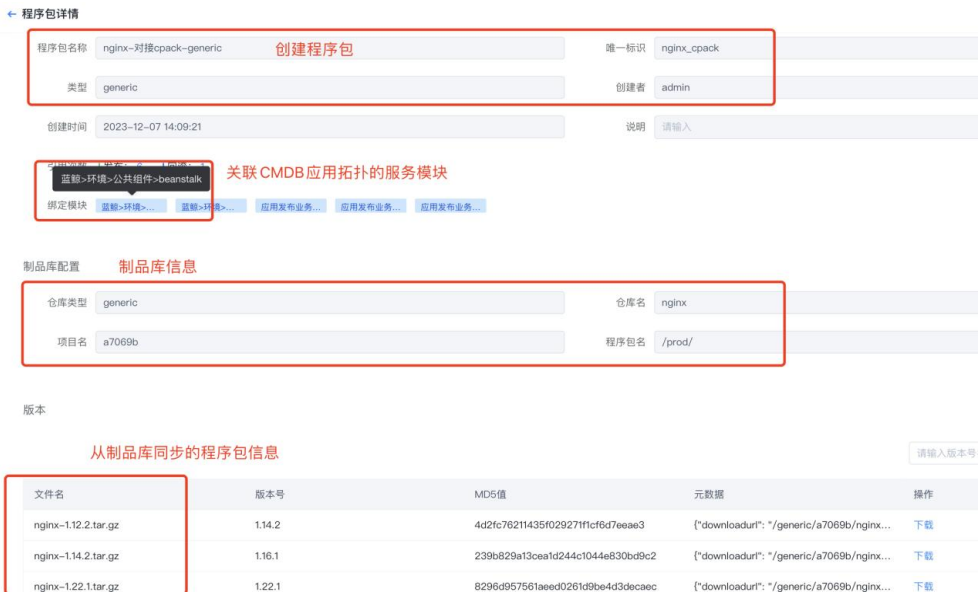


完成一套混合发布任务编排的流程：

步骤一：维护传统应用拓扑关系，服务模块关联主机



步骤二：创建程序包，从制品库同步程序包版本，并关联应用拓扑中的服务模块



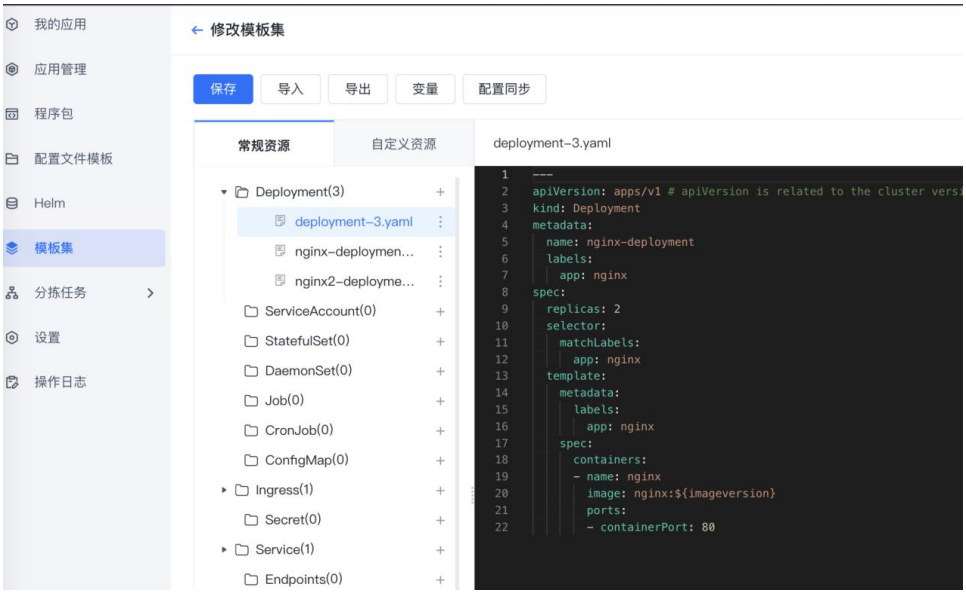
步骤三：在标准运维编排主机执行流



步骤四：创建传统应用发布节点，选择业务、模块、程序包、主机



步骤五：创建容器模板集，导入 YAML 文件，或者从生产 K8s 集群直接导入 YAML 文件。将容器模板集和 CMDB 的服务模块绑定

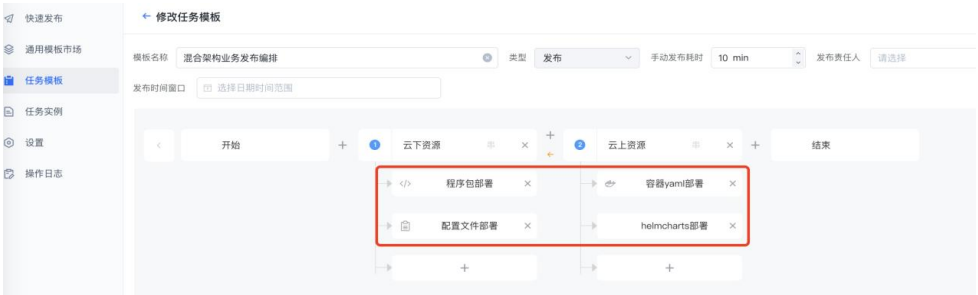




步骤六：创建容器应用发布节点，选择发布的业务系统和服务模块，选择关联的模板集，选择模板集中版本，以及版本内 Workload、Ingress



步骤七：创建完整的混合编排发布模板，包含主机程序包发布、容器模板集发布，以及配置更新和 Helm 发布（未展开）。



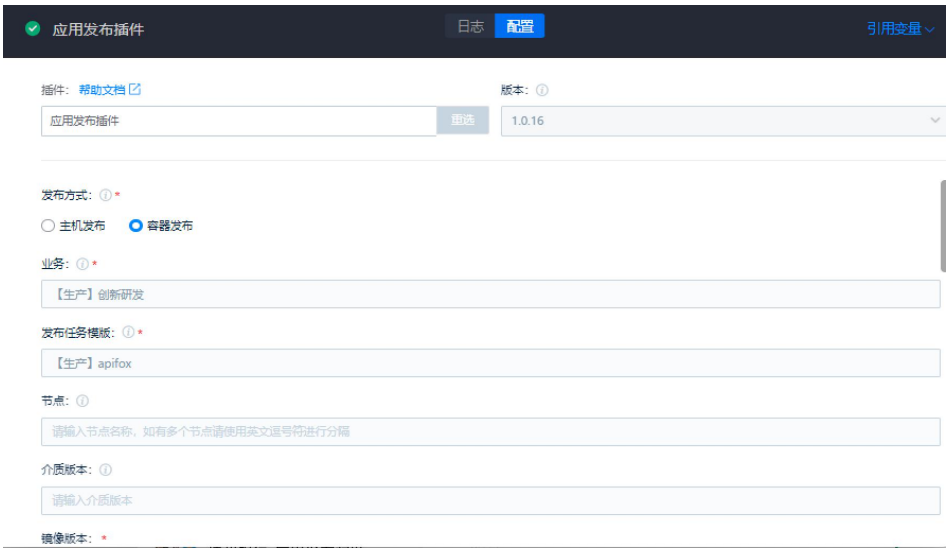
（3）测试环境/生产环境发布流程打通

研发人员在测试环境发布服务时没有问题，但在生产环境发布时却出现故障。这主要是由于测试环境和生产环境的架构和配置不一致，以及两者使用不同的发布流程。为了提高生产变更的成功率，需要在研发、测试和生产环境中使用相同的部署流程。然而，研发/测试环境强调敏捷性，要求自动构建和自动部署，而生产环境强调安全性，要求变更安全稳定。因此，需要平衡不同环境和团队的需求，才能实现研运一体化。

为解决这一问题，研发产品部和系统工程部密切协同，在一站式敏捷发布平台上实现测试和生产环境的统一流程管控。在测试环境中，使用 DevOps 流水线自动完成编辑、构建、代码扫描、制品入库、持续部署和自动测试，并在持续部署节点调用应用发布自动化接口进行部署。



研发/测试环境流水线



对接应用发布插件

生产环境，BPM 工单系统和应用发布自动化对接,达到生产变更任务的强审批执行。



对接工单系统的审批内容

（三）总结及展望

建设总结

敏捷发布平台接入业务 156 个，累计 1400 多个用户接入使用，平台累计共完成配置 CI 流水线 4402 条、CD 流水线 4044 条，CI 总构建次数 9 万余次，CD 总发布次数 6.5 万次；纳管 1 万余主机节点，60 多个 K8s 集群。

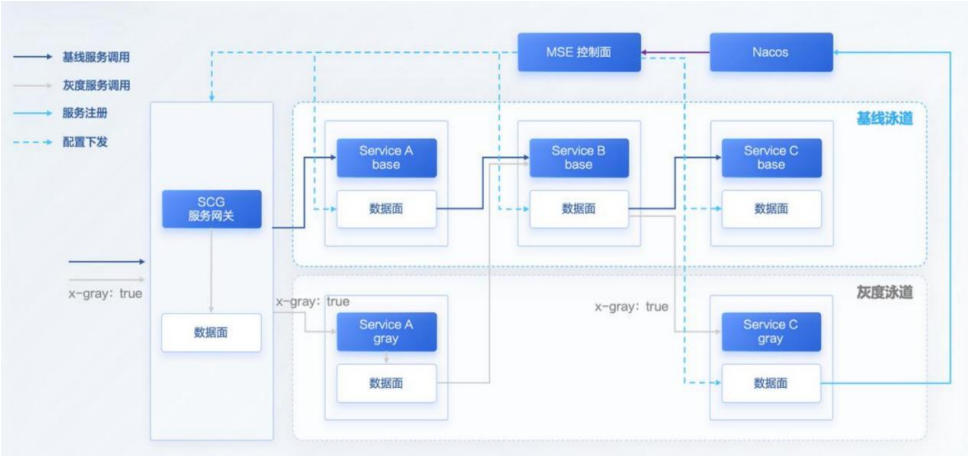
同时，逐步接入各业务线研发/测试以及灰度/生产环境，目前全环境覆盖率为 54%，部分环境覆盖率 44%。

展望方向

由于架构方面原因，目前大部分应用的发布并不能实现灰度，违反了“可灰度，可监控，可回滚”的实践原则，爆炸半径无法最小化。但是由于架构升级改造成本高昂，要进行全量改造并不现实。

因此，研发部和系统部目前正在联合推动基于 Java Agent 的无侵入全链路灰度发布方案的落地，目标是构建灰度泳道管理体系，为业务提供逻辑泳道隔离和泳道流量投递等发布能力，并计划与业务微服务网关对接，实现跨集群、多应用的灰度发布。同时，灰度发布规范体系也在制定及落地中，设置了业务灰度准入规则，旨在扩展发布窗口，帮助业务团队实现不停机的无

损发布。



4.4.2 某证券变更一体化平台建设实践

Elite 收录点评：

本证券案例的变更一体化平台建设目标是解决四种场景的变更统一：自研和外购应用的发布/变更一体化、传统和云原生应用的发布/变更一体化；测试与开发环境的发布/变更一体化；敏捷与安全的工程一体化。覆盖场景多，平台整合度高，规划参考性较强。

(一) 背景及设计原则

某证券是国内资产规模 Top 10 的证券公司，每年 IT 投入超过 10 亿，有庞大的客户基础和丰富的金融产品，有成规模的 IT 团队和匹配业务发展的变更一体化平台，满足了证券类业务 SRE 在变更场景中关注的特点。

部分传统业务系统是外购，由厂商进行维护和变更；新的业务系统大部分都是自研，通过流水线进行发布。变更平台需要同时兼容外购业务系统和自研业务系统的变更。

目前大部分业务系统是在虚拟机上通过二进制包方式部署，随着云原生的发展，部分新业务系统部署在 TKE 容器集群上，通过 Yaml 文件或者 helm 包进行部署，未来计划增加 ACK 容器集群。变更平台需要同时兼容二进制部署，和云原生的 yaml/helm 方式。

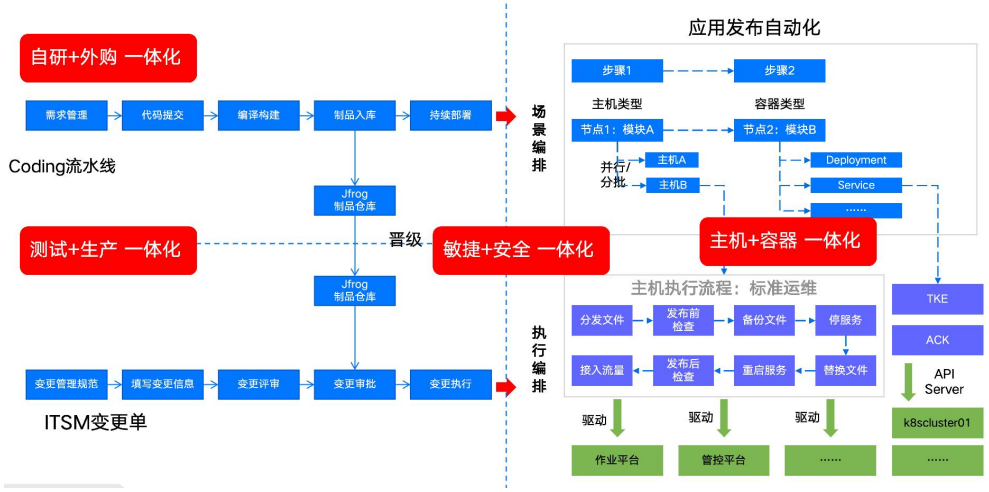
有存在测试环境使用一套发布流程，在生产环境使用另外一套发布流程，可能出现一个应用版本在测试环境通过测试，但实际在生产环境发布失败。主要原因是测试和生产环境工具和流程的差异性。因此变更平台需要支持同时在测试环境和生产环境使用，支持资源和流程的晋级，保证两个环境的操作一致。

在测试环境中，基于 DevOps 理念，要做到持续部署，达到“敏捷”的要求；在生产环境中，基于 ITIL 理念，要做到

变更管理，达到“稳定”的要求，变更平台需要同时兼顾“敏捷”和“安全”。

（二）体系设计详情

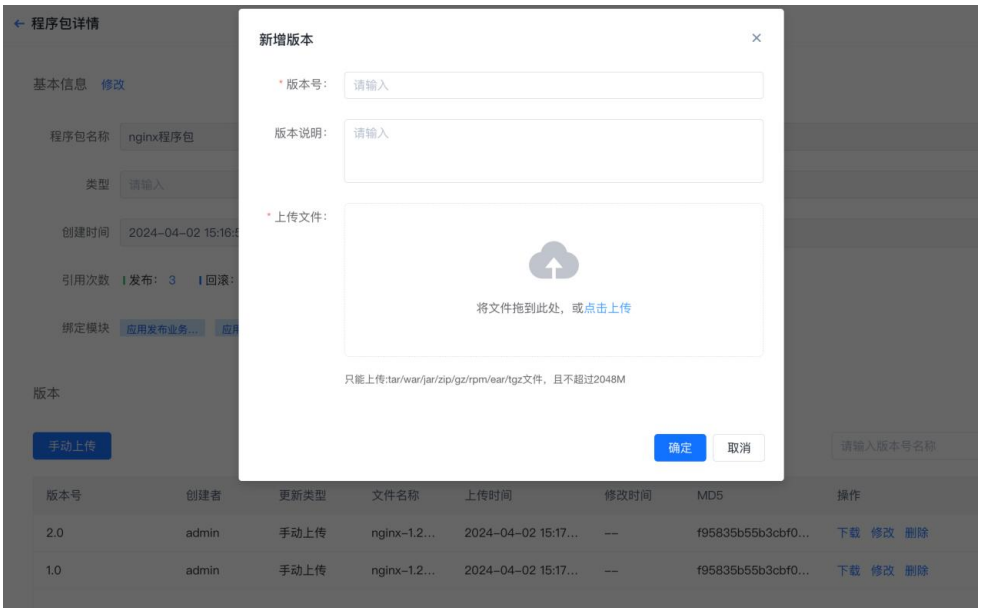
基于上述背景，IT 运维部一直在思考如何建设一个变更一体化平台，抽象出四个“一体化”的思路：自研和外购一体化、传统和云原生一体化、测试和生产一体化、敏捷和安全一体化。



（1）自研和外购一体化

外购业务系统是由厂商维护的，厂商提供的就是一个制品包，因此需要支持厂商运维人员在平台维护程序包，并完成发布动作。

变更一体化平台支持运维人员在界面上手动上传程序包，存储在共享存储服务中。



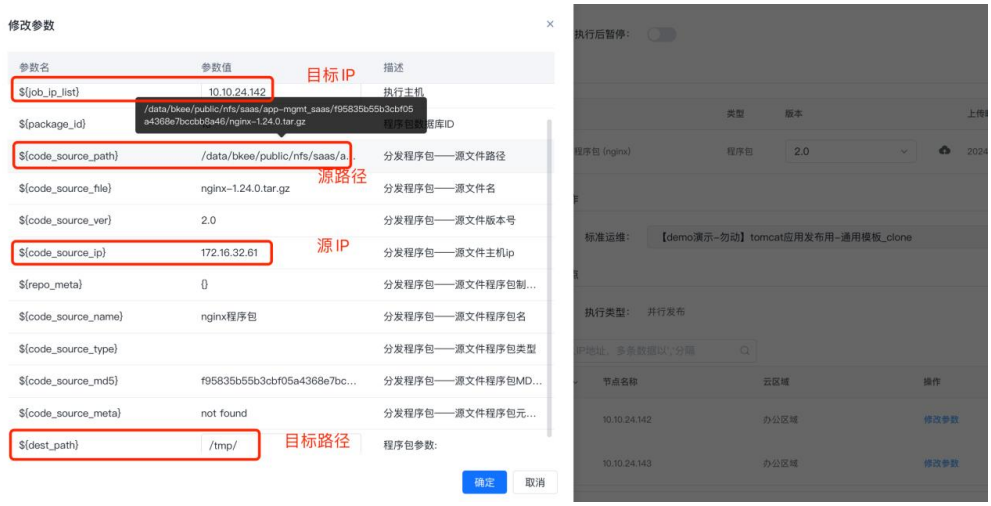
为了避免存储堆积，可以自定义设置存储清理策略：

- 永不清理
- 按天数清理
- 按版本清理



在发布任务中，会将程序包存储的源服务器 IP 和源地址，以及分发的目标 IP 和目标路径，作为参数变量传给标准

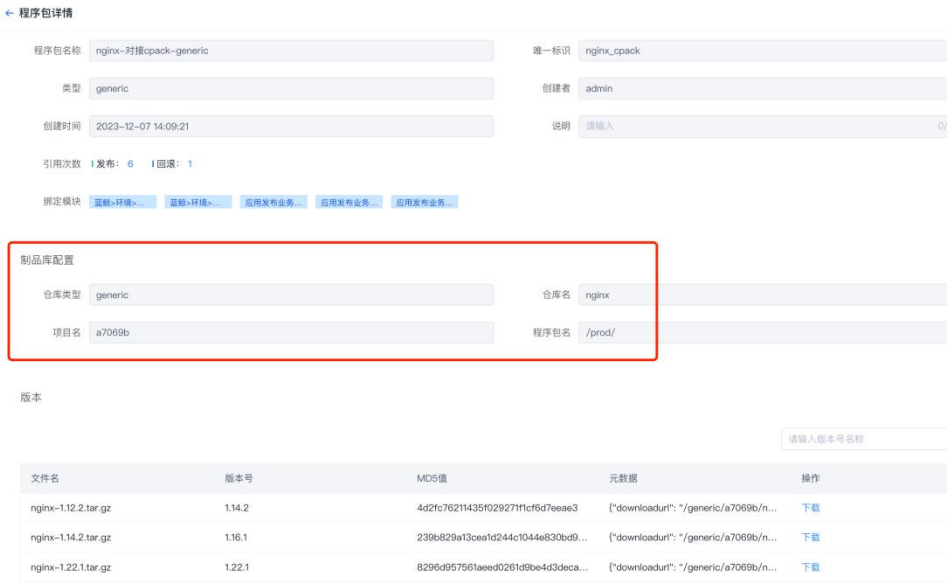
运维流程，这样可以通过作业平台的原子实现，文件的快速分发。



标准运维中引用快速分发文件的原子，利用 GSE 的大文件分发能力，可以快速自动将文件分发到目标机器上。



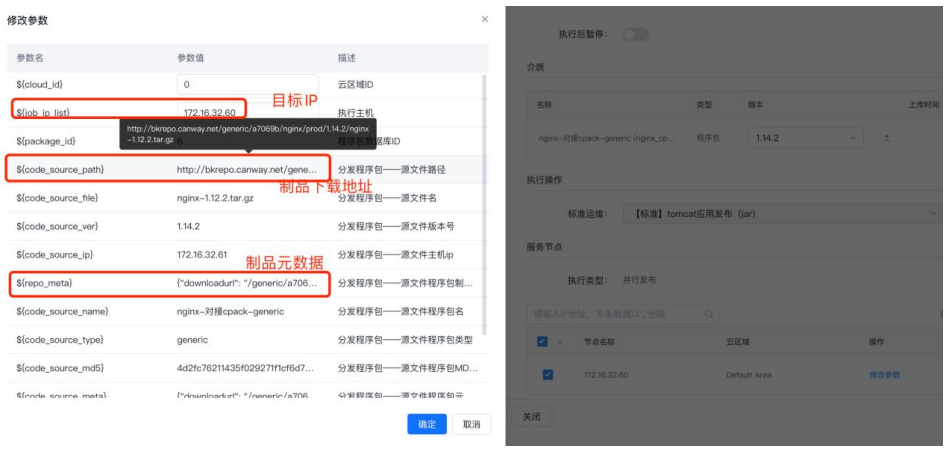
自研业务系统是开发团队维护，通过流水线打包之后，自动入库到标准的制品仓库中，在测试环境和生产环境中都是存储在制品仓库中。为了维护制品包的一致性，变更一体化平台支持通过制品库插件，对接制品仓库同步程序包信息。



上传制品库插件



针对这种类型，变更平台不会再存一份程序包，因此是调用制品库接口获取程序包的名称、版本、MD5、下载地址，用于做程序包分发。



考虑到直接开通所有主机到制品仓库的网络策略，存在较大的风险敞口，以及各个云区域到制品仓库的网络速度限制，不会在目标机器上直接执行 curl 命令下载程序包。

SRE 同学编写了自定义的文件分发原子，用一台代理机器，和制品仓库在同一个云区域，先执行 curl 下载命令，将程序包从制品仓库下载到本地，再通过作业平台的文件分发功能，实现大文件的快速分发。



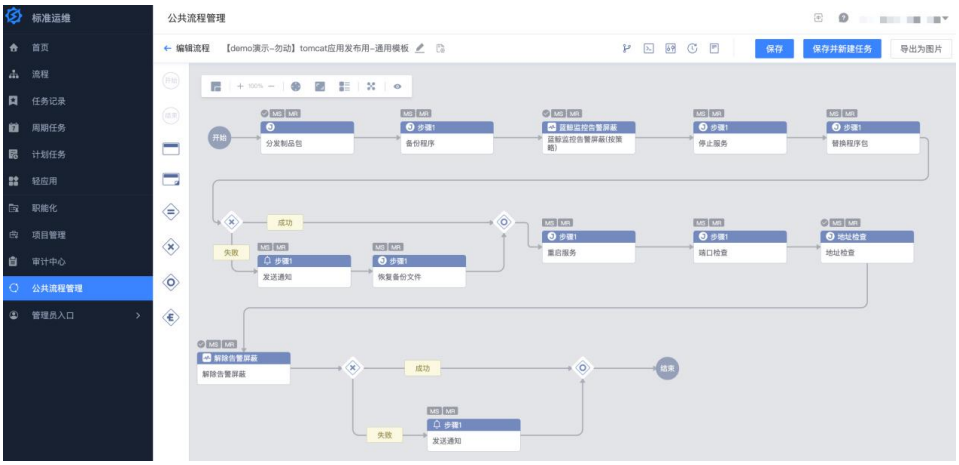
通过兼容设计，将两种类型的制品包统一管理起来，用同一套发布逻辑，完成制品包的分发和部署，实现自研系统和外购系统的一体化。



（2）传统和云原生的一体化

公司的传统应用部署在主机上，发布时需要登录主机操作，而云原生应用部署在 K8s 集群上，发布时需要执行 kubectl 命令完成。另外部分工具的主机发布任务和容器发布任务存在割裂，无法统一执行。而应用发布中心支持一个任务同时兼容主机发布和容器发布，实现一次审批一次编排。

针对主机场景，使用管控平台纳管所有主机，标准运维编排所有主机发布任务。并且针对同类型的业务系统，规范建议复用相同的标准运维流程，无需每个业务团队自己建设。通用化规范化发布流程建议：



(1)分发制品包：将具体的制品包名称、分发 IP、分发路径设置为变量，由业务系统在发布时传入，这样所有业务系统的制品分发复用这一个规范分发节点



(2)备份程序：完成原有位置的程序备份

(3)屏蔽告警策略：传入目标主机 IP，或者屏蔽策略 ID，停服务前自动完成告警消息屏蔽。

由于变更时存在服务启停，变更期间必然产生大量无效告警，因此需要将变更目标主机的进场告警屏蔽。此操作也是引用一个通用的监控平台原子。



(4)停止服务：执行停服务脚本

(5)替换程序包：替换程序包

(6)重启服务：执行启服务脚本

(7)端口检查：检查进程端口是否启动

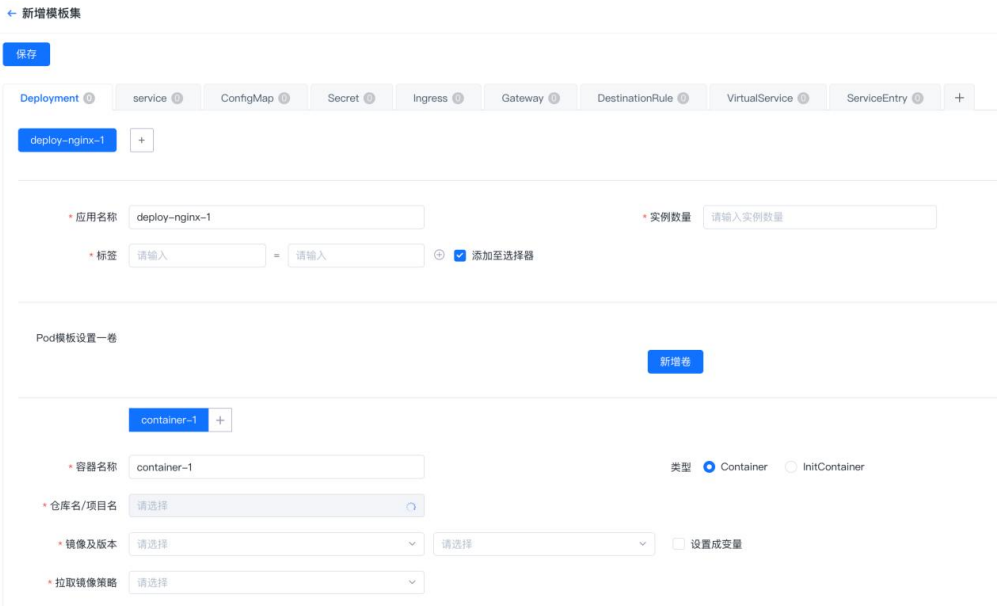
(8)地址检查：检查业务访问地址访问是否正常，编排一个通用检查脚本，具体 URL 通过参数变量传入，多业务系统复用此脚本



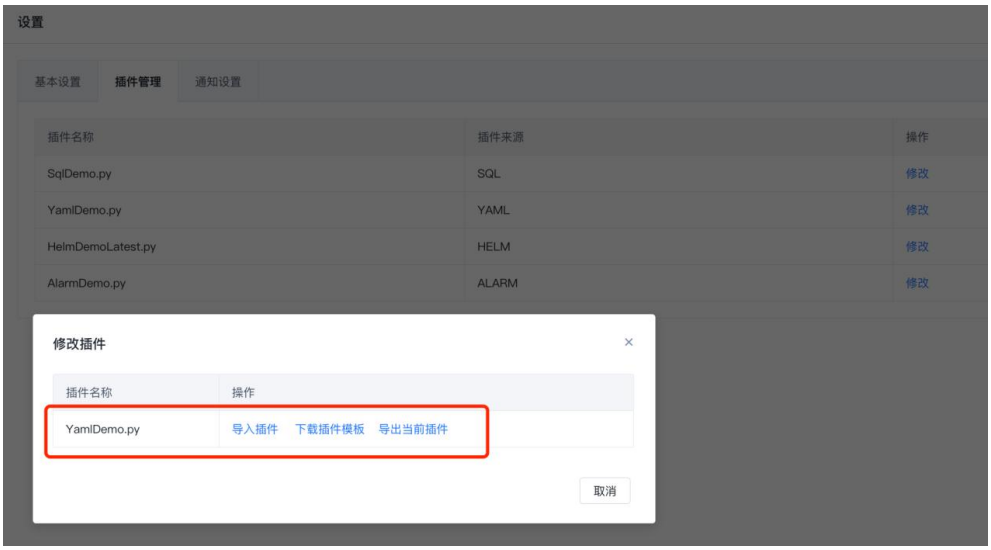
(9)解除告警屏蔽：变更完成之后，自动解除告警屏蔽

(10)发送通知

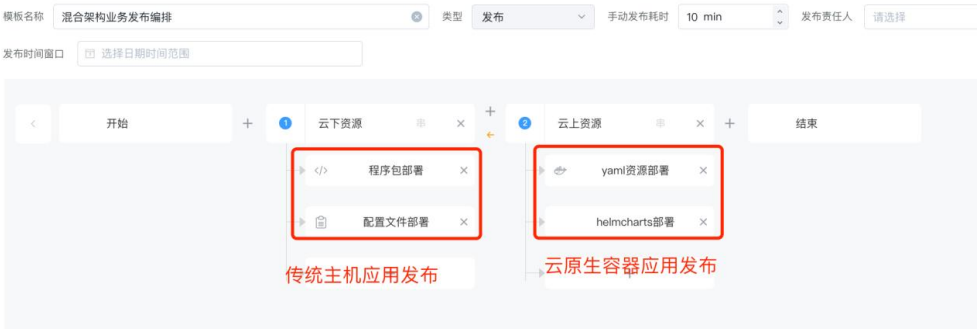
针对云原生的应用，变更平台支持编排容器应用的 yaml 文件，提供两种编排方式，一种是原生管理方案，支持运维人员直接导入 yaml 文件进行管理；一种是将 yaml 文件中的关键参数抽取成表单参数，运维人员只需要下拉选择即可，降低容器应用的编排门槛。



做容器发布时需要调用 K8S 接口，变更平台通过 yaml 插件集成接口实现容器发布。



因此变更一体化平台，支持将传统主机发布节点，和云原生发布节点，编排到同一个发布任务中，实现“传统和云原生的一体化”。



此外考虑到还存在数据库的 SQL 更新、纯配置更新等不同场景，变更平台在调度引擎时，单独设计的插件模块，支持通过可扩展的插件去接入更多发布类型。

（3）测试环境和生产环境的一体化

为了提高生产环境的发布成功率，要求生产环境的发布操作，尽可能 1 比 1 复刻测试环境的发布操作，达到测试环境和生产环境的一体化。

IT 运维部的 SRE 专家将发布操作拆解成三个环节，每个环节都需要保证测试环境和生产环境的一致性。

环节一：应用拓扑

在测试环境一般只会有一套集群，而生产环境两地三中心，至少存在三套集群，一般会认为测试和生产的环境没办法保持一致，从而忽略测试环境的维护。但实际上可以在应用拓扑层级保持一致，并且按照同一套规范去维护。

公司在测试环境、预发环境、生产环境都使用“业务-子系统-集群-模块”四层拓扑架构，减少架构层面的疏漏。



环节二：制品管理

在测试环境和生产环境都部署独立制品仓库，开通从测试环境仓库到生产环境仓库的单向定点网络，并创建定期制品同步策略。

开发人员在 CI 流水线阶段完成构建打包之后，就会自动入库，将版本包存储到制品仓库中；在测试人员完成测试后，自动触发制品晋级任务，自动将制品从测试环境的制品仓库同步到生产环境的制品仓库，用于生产变更。

这样确保生产环境要发布的制品包，是经过测试，并且和测试环境的版本是一致的，减少运维人员在过程中的手动下载上传干预，全流程自动化流转。

环节三：发布流程管理

发布流程时生产环境变更中的最核心的部分，更要求生产环境的发布流程，是在测试环境中完整执行过的。且为了降低人工重新配置的复杂度以及出错概率，变更平台中的发布任务模板和标准运维执行流程，都是支持从测试环境导出，直接导入到生产环境，保证发布流程的一致性。

变更一体化平台从应用拓扑、制品管理、发布流程管理三个维度，实现测试环境和生产环境的一致性，提高生产环境变更的稳定性。

（4）敏捷和安全的一体化

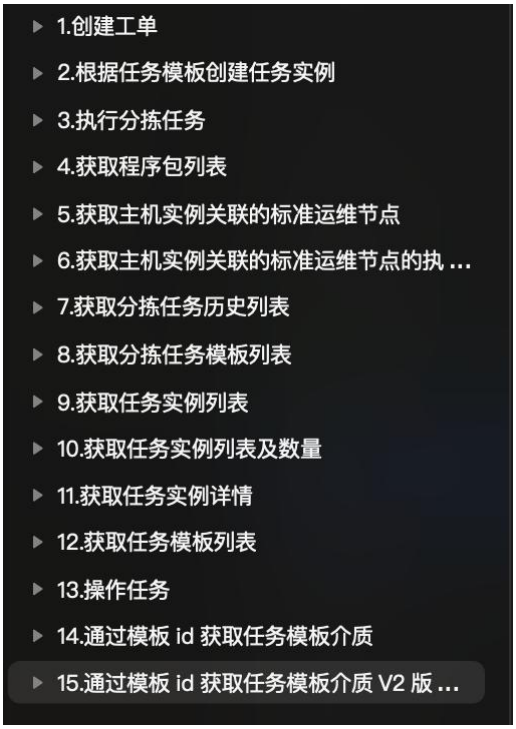
在测试环境，开发人员通过 CI 流水线做持续集成和持续部署，实现快速开发迭代，因此对变更平台的要求是：能够作为 CD 节点完美到 CI 流水线中，实现应用的快速部署。

变更平台开放了丰富的 OpenAPI 去满足 CI 流水线的要求：

search_template 接口支持传入业务系统查询发布任务模板

create_task 接口支持传入任务模板 id 创建发布任务

operate_task 支持操作发布任务，包括执行、暂停、灰度、停止



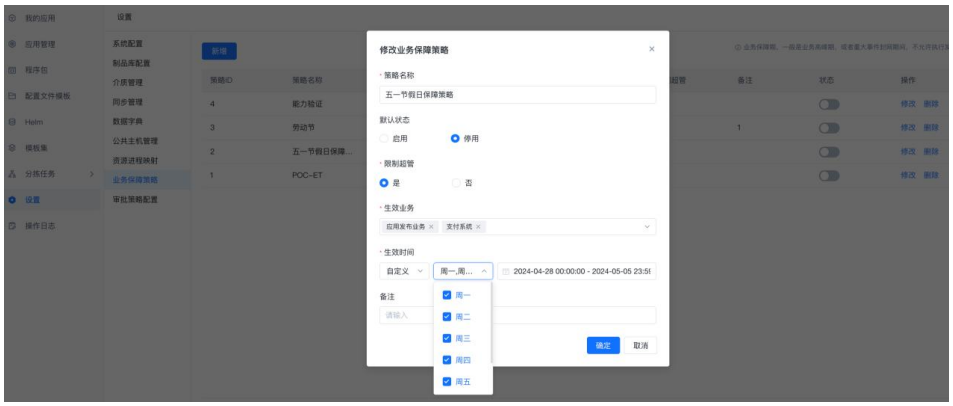
在生产环境，最重要的要求是稳定安全，因此需要对业务变更操作进行更多管控。一方面是在重要时间段内不允许变更，比如证券交易期间，重大会议期间；另外一方面所有变更

操作需要和工单系统对接，运维领导审批通过之后才能执行发布。

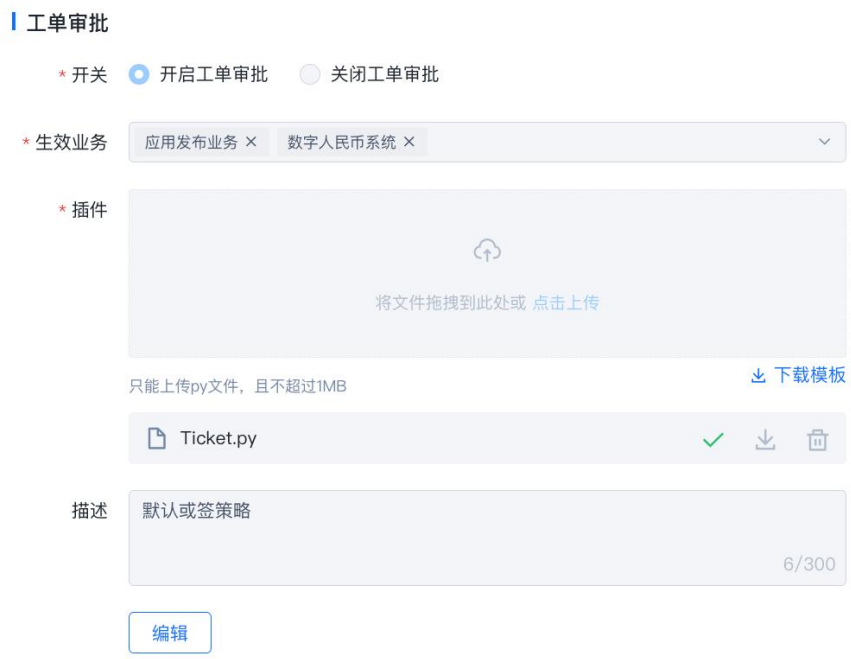
创建业务保障策略，在重大会议期间、节假日期间，从技术上限制运维人员对生产环境进行变更操作。目前大部分金融企业都是采用管理规范进行约束，在此期间不允许进行变更，但无法落地到技术层面进行强管控，还是存在业务团队冒险变更的风险。深层原因是，没有一体化的变更平台，每个业务团队使用各自的变更工具进行操作，无从限制。

业务保障策略支持灵活配置满足不同场景需求：

- 支持设置策略状态，默认启用/停用
- 支持限制超管
- 支持选择生效业务，不同业务的安全级别不同，不同时间管控要求不一样，C 端业务强管控，内部系统弱管控；以业务系统级别管控，而不是具体 IP 维度管控，降低运维复杂度
- 支持自定义设置生效时间，日常证券开市时间是周期性周一-周五，节假日是固定时间段



通过工单插件对接现有的变更工单系统，实现生产变更操作的审批。



在测试环境，通过 OpenAPI 集成到 DevOps 流水线中，实现快速发布，达到“敏捷”要求；在生产环境通过业务保障策略和工单审批策略，实现发布管控，达到“安全”要求。

（三）总结及展望

变更一体化平台建设是一套长期动态的系统工程，随着应用系统及周边配套平台联动，还有很多工作在规划中，有很多能力在持续完善，例如度量、复盘、规范化等等。另外一些源自互联网的新方案例如 argoCD 等相对于传统的 K8s 部署方式集成度更高，可以探索其与合规跟质量的结合点。

4.4.3 游戏 GitOps 发布管理实践

SRE Elite 收录点评：

1，这是一个典型的 GitOps 发布案例，对于互联网企业来说很常见，对于传统行业 SRE 具备一定的参考价值。它适用于流程和权限管控相对宽松而执行频率较高的测试环境、预发布环境、体验环境及低社会敏感应用的生产环境等。

2，该案例描述了一个大型 SRE 团队全球化应用部署管理的场景，对不同细分职能的分工和协作模式做了阐述，对不同类型权限的操作者提供了差异化入口及审计方案，同时也兼顾到了不同使用习惯的开源组件集成，体现了鲜明的互联网风格。

(一) 背景及设计原则

为了更好地加速业务创新和解决业务规模化的挑战，云原生应用架构与开发方式应运而生，与传统单体应用架构相比，分布式微服务架构具备更好的、更快的迭代速度、更低的开发复杂性，更好的可扩展性和弹性。然而，正如星战宇宙中，原力既有光明也有黑暗的一面。微服务应用在部署、运维和管理等方面的复杂性却大大增加，DevOps 文化和背后支撑的自动化工具与平台能力成为关键。

在容器技术出现之前，DevOps 理论已经发展多年。但是，如果“研发”与“SRE”不能用相同的语言进行交流，用一致的技术进行协作，那就永远无法打破组织和文化的藩篱。Docker 容器技术的出现，实现了软件交付流程的标准化，一次构建，随处部署。结合 IaC 和 Kubernetes 声明式的 API，可以通过声明式的方式实现自动化的持续集成与持续交付应用和基础设施，大大加速了研发和 SRE 角色的融合。

此外 GitOps 也是一种符合 DevOps 思想的运维方式，GitOps 以 Git 仓库作为唯一的事实来源，储存声明式配置，并通过自动化工具实现环境和应用的自动化管理。Git 实现了版本控制、回滚、多人协作；声明式配置保证了配置的可读性和事务性；自动化部署消除了人为错误，调高了部署效率和准确性，同时也保证了多环境的一致性。所以 GitOps + 声明式配置能够很好的解决传统运维的痛点，提高部署效率，保证部署的一致性。

（二）体系设计及关键流程

(1) SRE 分层服务模式

在实施 GitOps 的过程中，我们需要对 SRE 团队进行分层的能力建设，相距业务团队由远及近可以划分为“平台

SRE”、“定制 SRE（可选）”、“服务 SRE”、“On-call”四个团队：

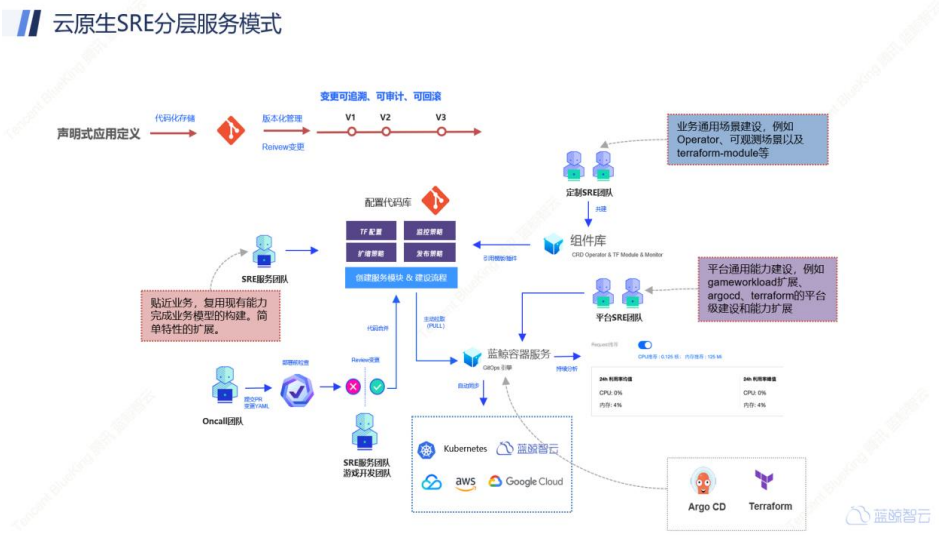
1）“平台 SRE”负责通用能力建设，针对游戏的特性定制和构建容器服务，提供一站式游戏微服务云原生能力为核心，完成游戏场景的 workload 定制、ingress 扩展、GitOps 以及 Terraform 等能力与容器平台的集成。

2）“定制 SRE”是可选组织，人数较少，视企业应用的异构程度及数量，以及“服务 SRE”团队的能力情况酌情设置，其职能可以合并到上下两层，如果设置该组织，可以负责通用的声明式自助服务场景建设，开发基于业务场景抽象的 Kubernetes CRD Operator、声明式全栈可观测场景，以及可复用的 Terraform module 等。

3）“服务 SRE”紧密结合业务需求，利用“定制 SRE”提供的方案、工具和技术对业务的上云改造方案以及预期效果进行评估，并完成业务应用模型构建。实现业务基础设施的编排，业务应用的运行管理，以及业务的扩缩容策略以及流量管理策略的配置。同时将声明式的配置代码存储在代码仓库中，基于 Git 实现配置的版本化管理和权限管理。

4）全球 On-call 团队则基于统一的 Git 变更流程，对应用以及应用所依赖的基础设施配置进行变更。变更内容在经过

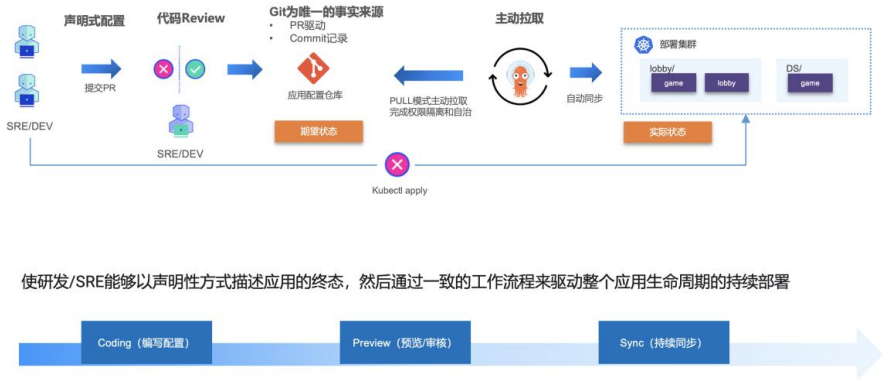
代码审核后将被合并到主干，GitOps 主动监测到配置的变更并同步变更到业务环境中，并确保环境配置的一致性。



(2) GitOps 持续交付

GitOps 能力基于 Argo CD 实现。GitOps 作为一种应用交付模式，极大地简化了传统 CI/CD 流程。在持续集成（CI）阶段，SRE 团队的焦点集中在代码编译和容器镜像的构建上，避免了在蓝鲸 CI 平台（蓝盾）中存储 Kubernetes 的认证信息，从而增强了系统安全性。进入到持续交付（CD）阶段，运维人员仅需通过 Pull Request (PR) 或 Merge Request (MR) 向代码仓库提交变更，GitOps 引擎会主动地从 Git 仓库拉取配置，并自动地将这些配置与当前集群的状态进行对比和同步。这种“拉取（Pull）”模式相较于传统的“推送（Push）”模式的最大优势在于，它允许集群主动拉取配置，

避免了编写和维护复杂的流水线脚本或手动触发更新流程的需求，简化了操作并提高了自动化水平和系统的安全性。

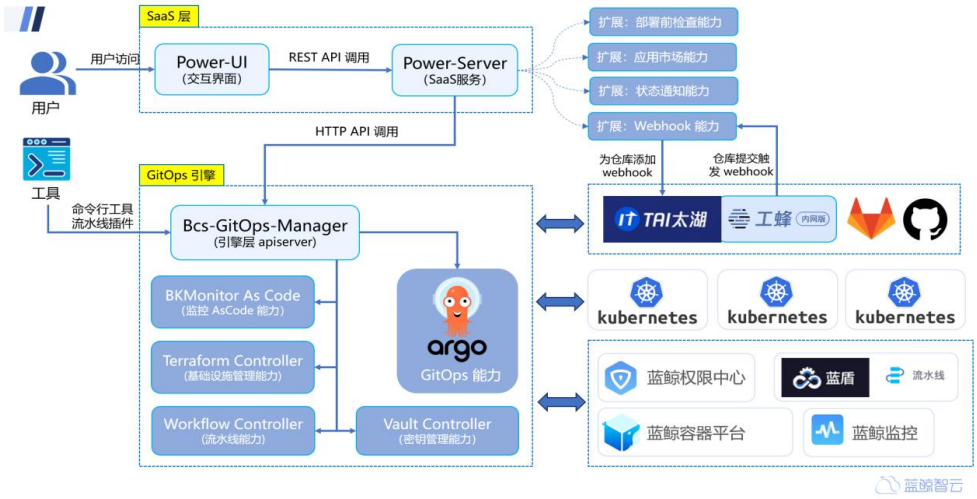


（3）声明式自助服务能力

针对游戏 SRE 团队而言，我们围绕游戏生态构建了一系列声明式和自助式的服务场景。通过向蓝鲸平台工程靠拢，我们实现了服务的可衡量、可管理和可复制性，同时不断进行优化和创新，以适应不断变化的需求和挑战。

游戏 SRE 团队根据游戏场景提出了云原生架构下应用的三层架构模型，运维能力层声明了应用的交付流程、扩缩容策略，应用运行层声明式了应用的基础元素，包含服务、服务配置和数据库脚本，基础设施层则声明了应用运行时的环境和基础设施，基于 Everything as Code 理论将应用模型存储在代码仓库中，实现统一演进，可审计、可回溯。

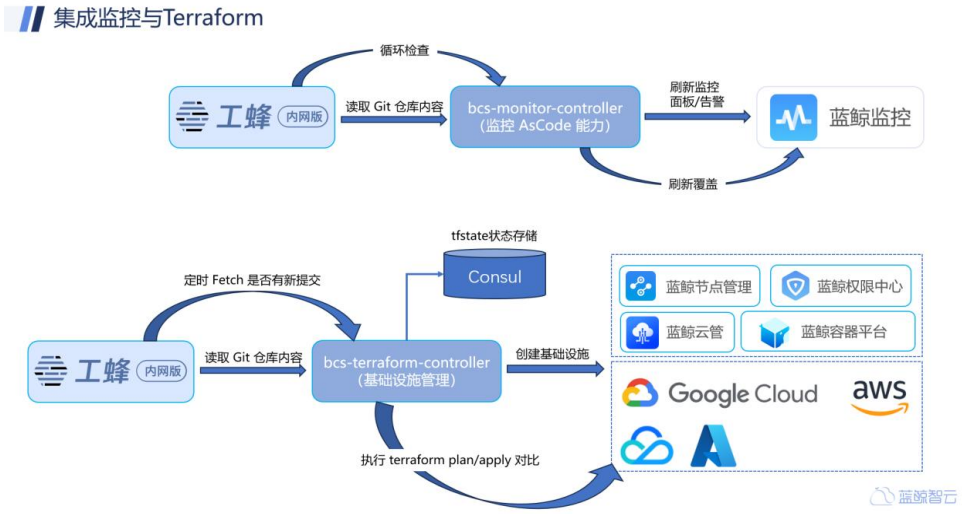
在面向用户层，为不同团队设计了多种入口，以蓝鲸 SaaS PowerApp 为例，为业务运维人员提供了更加友好的操作界面和更加完善的权限控制。



除此之外，游戏开发人员可以通过蓝鲸 CI（蓝盾）触发，合作企业工程师可以通过标准运维 App 触发，平台 SRE 工程师可以通过 BCS-SaaS 触发，同时也保留了命令行操作的入口。

最后，我们增加了 ArgoCD 的扩展，例如密钥管理、代码库协议转换、Dry-Run、历史差异对比、动态资源信息等，

在基础设施管理层面，集成了监控与 Terraform 等基础能力。



（三）总结及展望

权威机构 Gartner 认为，在云原生环境下，我们需要建立面向服务的 DevOps 理念，通过平台级能力为服务团队提供 DevOps 的支持。这种模式被认为是最适合微服务的 DevOps 服务模式，其核心价值主张就是“X as a Service”。

在这个背景下，游戏 SRE 团队的任务就是为游戏研发团队提供一个稳定的工程平台，帮助他们屏蔽复杂的底层基础设施。这个平台包含开发者所需的各种工具、自助的开发者服务界面、以及可复用的模板，以满足开发团队的日常需求。

在这里，平台的编排构建可以由 IaC 实现统一的编排能力。而基于 GitOps 则解决了声明式配置管理以及持续交付的问题，实现全球地域的一致性游戏应用交付和管理。

而平台工程的出现，其实是 IaC 和 GitOps 技术发展的必然结果。在现代软件开发中，由于技术复杂度和变化速度的加快，以及微服务架构的广泛使用，研发团队不再能够有效地关注所有的技术细节。平台工程就是将这些通用、底层的工作抽象出来，由 SRE 平台团队来提供，从而让研发团队能够专注于他们的主要目标。

5 故障应急

5.1 故障体系

5.1.1 故障定义

故障的定义如下：在日常运营中，无论任何原因导致我们的服务中断、服务品质下降或用户服务体验下降，都被视为故障。但不包括用户环境活自身操作引起的问题。

5.1.2 故障等级

故障等级是衡量故障影响度的标准，数字越小级别越高。通常业界是以 P0 P1 P2 P3 P4 P5 等作为序列， P0 为影响度最高的故障（也有部分公司以 P1 作为最高级别）。

影响度参考指标包括客户投诉量、资金损失金额、受影响用户数，故障时长，业务重要性等。

通常情况下， 故障等级由 GOC、业务开发、质量团队、业务 SRE 和业务稳定性等部门共同制定。而且相关故障等级需要提交业务负责人审批，确保业务方对相关故障级别标准的认可。并定期由 GOC 统一审核故障定义和等级，并公示，确保大家按统一标准执行。

故障等级的作用，我们可以基于这些定义，开展可观测性建设（配置针对性的告警体系以及预案）和平台能力构建（包括而不仅限

于可用性架构设计及评估，压测，混沌演练，容灾演练），目的是提前识别潜在问题，进行预防和保障。

在故障发生后，需要对故障进行评估，并采用就高原则，如果一个案例符合多条故障定义，通常以最高等级为准。

5.1.3 故障序列

高稳定性团队可能会面临因非技术问题（如业务运营方配置失误，网络运营商的抖动或其他外部因素）引起的故障，因此需要明确技术故障与外部原因之间的区别，因此需要引入故障序列。

故障序列是根据故障的根本原因和实际影响进行分类的方法，区分“故障根因”和“实际影响”。

在定义故障序列时，主要考虑以下几点：

技术责任：判断故障是否由技术原因引起，如是，则可能扣减故障预算。我们关注 P1 到 P5 等级的故障，因为它们通常涉及技术责任并影响业务。

故障记录与配额：评估年度稳定性需考虑故障记录和配额。许多公司限制每年故障次数，超出后可能影响绩效。因此，非技术原因的故障，如业务操作失误，不扣减预算。例如，外部原因的故障标记为“W”，业务操作引起的标记为“S”。公司可根据这些分类制定定义更明确相关的稳定性指标。

5.1.4 关键业务场景

组织有众多业务，究竟该优先保障哪些？我们应聚焦于最重要的业务，而非面面俱到。因此，我们定义了关键服务接口和结果点，即 关键业务场景， 也称 GOC（Global Operation Center）场景，也有公司叫 NOC (Network Operation Center）)。只要确保这些 GOC 场景的稳定，组织就不会出现重大问题。

关键业务场景通过以下三个领域定义：

- 场景描述：明确需要保障的业务场景及其功能。
- 场景影响评价：设定衡量指标（如成功量、成功率、响应时间）来评估场景的稳定性和性能。可以理解为 SLI /SLO 体系的构建。
- 故障等级：明确场景故障时可能涉及的最高等级。重要业务需重点保障以避免高等级故障。

实例说明

以登录功能为例：

免密登录：使用成功量作为衡量指标，若成功量下降，则可能触发相应故障等级。

账号密码登录：使用成功率进行衡量。

GOC 场景定义的目的：

通过定义 GOC 场景，聚焦关键服务接口和结果点，**驱动和确保公司整体稳定性。**

围绕 GOC 场景，开展容量评估、压测、限流、监控、变更管理等各项工作。

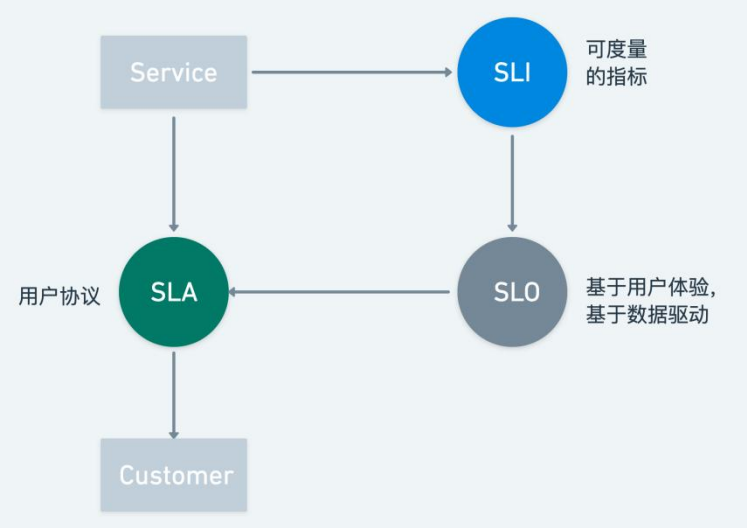
- 业务链路：清晰了解所保障业务的核心目标和完整的业务链路。明确稳定性保障团队，当问题发生时，能够迅速召集相关人员共同解决。
- 容量模型：从接入层到中间件，再到下游数据库，全面了解系统架构；建立性能基线，明确资源成本；描绘流量模型，识别不同时间段的流量特征（如早高峰、午高峰、晚高峰）；定期进行全链路压测，评估系统容量，确保在高负载下的稳定性；根据容量模型，设置合理的限流策略。防止突发流量超出系统承受范围，保障服务稳定。
- 可观测性：明确业务的关键监控点和指标（成功量、成功率、响应时间等）。建立实时监控和预警机制，及时发现和处理异常情况。
- 变更防御：建立变更感知体系，构建变更流程机制，对核心业务实施严格变更管理，确保安全可控，保证所有变更可被定位，有迹可循。
- 应急能力：设定应急响应率目标，设定应急组织整形，提升团队快速定位和解决问题的应急止血能力，确保故障迅速恢复。

- 历史故障：对历史故障进行分析复盘，找出稳定性薄弱环境，并进行跟进。

在明确了关键业务场景后，需要进行关键业务的 SLI / SLO 体系的构建。

5.1.5 应用服务 SLI/SLO/SLA

5.1.5.1 什么是 SLI/SLO/SLA



（1）SLI（Service Level Indicator，服务水平指标）是衡量系统或服务性能的具体指标，直接反映服务的质量或稳定性。SLI 通常用来监控特定的服务性能数据，如可用性、延迟、错误率、吞吐量等。它是实现 SLO 的基础，通过 SLI 的实时数据监测，可以判断服务是否达到了预期的 SLO 目标。

一般把这些指标分为 3 层：用户体验、中间件、基础设施。用户体验主要是指和用户强相关的一些功能监控，比如登录、购买等

功能，这些往往用户的关键路径，也是我们应该关注的点；中间件主要指在架构中用到的开源组件等；基础设施就是指网络、物理服务器、物理电源、磁盘阵列卡等，在建设 SLO 时优先梳理用户体验层指标。

（2）SLO（Service Level Objective，服务水平指标）是对服务质量的量化目标。它定义了系统或服务在一段时间内需要达到的性能或可靠性标准，通常用百分比来表示。例如，如果设定了 99%的 SLO，那么服务在一个月内至少要有 99%的时间可用，或其响应时间需要满足一定的标准。SLO 通常基于用户体验和业务需求来设定，既要满足用户期望，又要在技术和成本上实现可行。

（3）SLA（Service Level Agreement，服务水平协议）是服务提供商与客户之间达成的正式协议，规定了服务的期望水平以及未达标时的补偿措施。SLA 通常会包括具体的 SLO 目标，例如服务可用性为 99.9%，以及服务中断的赔偿条款。SLA 的目标是明确责任和预期，确保服务提供方和客户之间的透明度和信任。

在特定情况下，可以考虑将 SLA / SLO 与绩效挂钩，这样可以确保团队的目标与组织的服务质量目标一致。团队成员可能更加积极地工作以满足或超越这些目标。同时这样也为团队设定了明确的期望，并能实现数据驱动的评估方式。但是，与绩效挂钩，团队成员可能会过度专注于短期指标，而忽视长期目标和创新。建议每个组织根据客观的实际情况进行使用。

5.1.5.2 如何建设 SLI/SLO

SLO 建设步骤：确认 SLO 预期 -> 梳理关键链路 -> 梳理 SLI（1 原则+1 方法+关键链路） -> 上报 SLI -> 创建监控策略 -> 计算 SLO -> SLO 指标呈现

1. 确认预期

和业务负责人、产品、策划、开发等相关干系人一起确认该功能的预期，即 SLO。需要注意的是，SLO 的设定需要根据服务实际需求来制定。趋近于 100%的 SLO，将会消耗巨大的成本，但是用户可能并感受不到变化。过低的 SLO，将会让用户的感受变差。我们需要确定在不影响用户体验的情况下，确定一个合理的 SLO。

2. 梳理关键路径

梳理用户从功能入口到使用该功能的关键路径，比如：

app 启动 --> app 更新检查 --> app 更新 --> 登录 --> 获取昵称 --> 展示昵称

3. 梳理 SLI

原则：优先梳理用户体验层 SLI，即用户可以感受到的功能、按钮

方法：VALET，从 V（容量）、A（可用率）、L（延迟）、E（错误）、T（人工提单，保底分类）5 个维度来梳理，避免漏掉 SLI

将这 2 个原则套在第 2 步关键路径的每个节点，即可梳理出一份相对完整的 SLI 列表

4. 上报 SLI

对于没有 SLI 需要客户端或服务端从新上报到监控系统，对于已有的 SLI 需要校验其准确性

5. 创建监控策略

针对每个 SLI 需要创建一条监控策略，策略的告警阈值根据第一步定的 SLO 来确定。举个例子，若 SLO 定为 99%，则“app 更新检查成功率”指标的告警阈值为＜99%

6. 计算 SLO

围绕业务稳定性衡量，一共设计了 5 个指标, 多个维度，并基于监控系统计算出相关指标的值。以下的表格是一个指标和维度的事例的组合,供参考，通常 SLO 的指标维度，一定会包含时间维度，服务维度，其他可以根据自己的业务特性自行定义。

维度信息

维度	含义	枚举值
VALET	VALET 方法论，分别代容量、错误、延迟、可用率、工单	Volume Availability Latency Error Ticket
range_time	固定的时间范围，单位：天	1 7 30 180 365
strategy_name	监控策略名	策略名

strategy_id	监控策略 ID	策略 ID
bk_cc_biz_id	业务 ID	
bk_biz_name	业务名	

指标信息

指标	含义	计算公式	包含维度
slo	汇总的 SLO 指标，代表当前场景的稳定性	SLO=（总时间-不稳定时间）/ 总时间 不稳定时间=多个 SLI 告警时间并集	range_time
slo_error_time_info	各个 SLI 错误消耗详情，根据该指标，可看出哪个指标的不稳定时间最长 每个 SLI 独立计算	slo_error_time_info =SUM（SLI 告警时间）	velat range_time strategy_name strategy_id bk_cc_biz_id bk_biz_name
slo_error_time	汇总之后的错误消耗	不稳定时间=多个 SLI 告警时间并集	range_time
mttr	告警平均恢复时间，值越小越好	mttr = 故障总时间 / 故障次数	range_time
mtbf	平均告警间隔时间，值越大越好	mtbf = 正常时间 / 故障次数	range_time

7. SLI/SLO 指标呈现

SLO 的展示一定要基于对应 SLI，目的在于让相关干系人能直观的看到当前 SLI 下的 SLO、错误消耗等指标。展示的思路根据不同的用户，大致可以分为三种方式：

（1）业务场景视角的展示。针对业务某一个 SLI 或者某个场景做深度的 SLO 数据展示。可以分为多个仪表盘，每个仪表盘再按照 VELA 维度展示该功能所有 SLI 指标。达到效果：每个 SLI 的实时数据、可自定义时间范围展示 SLI 指标的曲线及具体值。更聚焦在某个场景或某个 SLI 的深度展示和分析。例如：某业务登录场景的 VIP 延迟、鉴权成功率、用户信息拉取延迟等登录场景核心链路的 SLO 详情展示。大多用于故障定位，排查问题。

（2）业务整体视角的展示。从业务的角度，展示多个 SLI 的 SLO 数据，将多个功能的 SLO 指标汇总在一起展示。达到效果：通过该仪表盘查看各个功能过去不同时间维度的 SLO，用于判断各个功能的稳定性（slo）；当某个 SLO 低于我们预期时，可以定位到具体影响的 SLI 指标（slo_error_time_info、slo_error_time）；各个功能的告警发生频率（mtbf）；各个功能的告警恢复时间（mttr）。例如：某业务的登录、支付、上传、浏览、语音等核心功能的 SLO 整体展示。大多用于单业务稳定性评估。

（3）多业务整体视角的展示。从整个公司或集团的更高的视角展示多业务的 SLO 数据，将公司或集团主要业务模块 SLI 和 SLO 定义清晰，并展示在一个大屏上。例如：公司所有业务登录成功

率、公司所有业务信息发送成功率、所有业务添加购物车成功率等。用于评估整体技术中台能力，整体服务质量。

整个 SLI/SLO 的展示，从微观业务场景细节，到宏观业务基本状况，根据不同的使用场景，不同的用户需求，可以拆分为不同展示模式，汇总不同量级的数据，灵活展示。

5.1.5.3 如何持续迭代 SLI/SLO

SLO 出来后，可以用实际值和之前的预期做比对，分为以下 2 种情况

1. 实际 SLO < 预期 SLO。比如实际值为 95%，预期为 99%

此时可观察 SLI 的错误消耗，先对错误消耗最大的 SLI 进行调查，如果是指标质量问题导致 SLO 低于预期，那则修复 SLI 后再进行观察

如果指标质量一些正常，也无错误数据，那说明指标的实际情况就是这样，确实达不到预期值。此时又可分为 2 种情况：

业务架构不合理或部署方式不合理导致的 SLI 指标达不到预期：比如架构单点导致用户访问不稳定，集中部署导致用户访问延迟加大，此时应该利用 SLO 数据推动开发进行架构调整，推动运维进行部署优化，从而提升用户体验

SLO 预期高于现实：可能现实条件确实达不到我们的要求，此时应该要放宽 SLO，降低 SLO 预期。否则我们的 SLO 指标会一直处于不达标状态，开发、运维也无能为力，这样的结果会造成团队恐慌，一定不是我们想要的。

2. 实际 SLO > 预期 SLO。比如实际值为 99%，预期为 95%

SLI 覆盖率低：确定关键链路 SLI 是否梳理完整，如果有缺失，则补齐，先提高 SLI 在关键路径的覆盖率

SLO 预期太低：这么导致我们对用户体验的衡量不够准确，可能会出现用户体验很糟糕，但是我们 SLO 数据一切正常，导致与我们目的背道而驰。此时正确的措施应该是收紧 SLO，提高预期

3. SLO 迭代小结

从上面的分析来看，迭代 SLO 主要分为 2 大类：实际 SLO < 预期 SLO 和实际 SLO > 预期 SLO。

实际 SLO < 预期 SLO：一般情况下我们首先要做是看为什么没有达到预期的 SLO，而不是调整 SLO 的数值。只有有充分理由说明之前的 SLO 制定存在不合理，那么我们才能放宽 SLO。

实际 SLO > 预期 SLO：实际的 SLO 比预想的要好，我们可以分析是之前的目标设置的低了，还是我们投入了超过预期的成本。如果这个服务不需要这么高的 SLO，而我们也需要降低一些成本，那么可以适当放宽 SLO。

对于 SLO 的持续迭代，可以从下 5 个步骤执行：

- SLI 质量优化
- SLI 覆盖率提升
- 架构调整
- 部署优化
- 放宽或收紧 SLO

5.2 故障应急体系

5.2.1 故障应急体系设计原则

故障应急体系设计应以快速恢复业务为核心目标。故障应急生命周期通常包括故障发现、故障诊断、故障恢复、故障复盘四个阶段。通过完善的监控与发现机制、有效的故障诊断与定界、及时的恢复措施、全面的复盘与改进，再配套适当的组织流程、定期演练和持续优化，以及相关工程平台的落地，是实现故障应急响应能力提升、保障业务连续性的关键。

5.2.2 故障应急流程设计

5.2.2.1 组织支撑

5.2.2.1.1 组织支撑的重要性

跨部门协作的必要性：故障管理涉及多个团队和部门，单凭某一部门无法独立完成。组织层面的支撑能够打破信息孤岛，促进跨部门的高效协作。

明确职责与权限：通过组织结构的设定，明确各级人员在故障应急中的角色和职责，确保在紧急情况下，所有人都知道自己的任务和权限。

资源的有效调配：组织支撑能够保障在故障发生时，快速调用和调配所需的资源，包括人力、技术和物资。

文化与制度的保障：组织支撑还体现在文化和制度建设上，通过制度的制定和文化的宣导，提升全员对稳定性和故障应急的重视程度。

5.2.2.1.2 构建应急组织架构

1.设立专门的应急管理委员会

建立一个跨部门的应急管理委员会，负责统筹公司的故障应急工作。该委员会应由各业务单元的代表组成，具备以下职责：

制定应急策略和政策：包括故障定级、责任划分、应急流程等。

协调各部门的应急工作：确保各部门在应急过程中步调一致。

推动应急文化建设：通过培训、宣导等方式，提高全员的应急意识。

2.明确应急角色和职责

在组织内部，需要设立不同级别和职责的应急角色，因每个公司所在的行业以及业务特性不一样，没有办法一概而论， 因此以下给出的部分参与角色设计仅供参考。

应急总负责人：负责全公司的应急指挥和协调，具备决策和资源调配的权限。

业务线稳定性负责人：专注于特定业务线的稳定性保障，负责快速响应和解决问题。

GOC 工程师：需要 7x24 小时应急盯屏值班工作，针对全站各类风险事件，及时发起应急响应流程，进行有效的应急调度，负责初步的故障处理。根据公司规模， 可能还有分为部门级，公司级等多级的 GOC。

OnCall 工程师：需要确保自己的电话或其他通讯工具在值班期间始终保持畅通，以便随时接收和处理来自系统或同事的告警信息。在收到告警后，OnCall 工程师需要迅速确认问题的性质和严重程度，并调用相关的方面进行故障应急处理。参与 OnCall 的人员，可能是各个产品线 技术负责人，又或者是研发，OnCall 会参与日常问题的响应。与 GOC 工程师类似，OnCall 工程师同样会根据公司及业务实际情况，设立分级的 OnCall 制度并配备相应的人力。

技术应急人：简单来说，就是研发、质量、SRE，负责本业务的故障跟进及处理；并负责本业务的风险场景识别、监控覆盖、应急快恢能力建设、应急人员培养、风险意识培训。

此外，蚂蚁集团在长期的工作实践中积累了宝贵经验，并首创了**“应急值班长”**制度。值班长是公司应急响应体系中的关键角色，可以理解为对业务有一定了解，有组织力、决策力，且已经过 3 次以上符合预期的应急处置（包括线上的真实问题和演练问题）的专家，负责全公司的应急指挥和协调，具备决策和资源调配的权限。在应急情况下，值班长需要组织和协调各个团队（如 GOC 工程

师、Oncall 工程师、研发、质量、SRE 等），确保快速响应和有效解决问题，实现应急止血。由于相应的要求很高，压力巨大，因此能担任值班长，是相关领域专家的荣誉，公司也会进行相应的公示及表彰。

3. 建立高效的协同机制

明确的 OnCall 值守体系：设定一线、二线、三线值守人员，明确各自的职责和交接流程。

完善的应急响应流程：制定标准化的应急预案，确保在故障发生时有章可循。

畅通的信息共享渠道：利用即时通讯工具、故障管理平台等，实时共享故障信息和处理进展。

资源保障机制：确保在紧急情况下，能够快速获取所需的技术和人力资源。

5.2.2.1.3 强化文化与制度建设

1. 制度建设

制定应急预案和规范：包括容灾架构设计、演练章程、故障处理流程等。

明确奖惩机制：将稳定性目标纳入各部门的绩效考核，激励全员重视故障管理。

建立知识库和文档：总结故障处理经验，供全员学习和参考。

2. 文化宣导

开展安全生产月等活动：通过专题活动，提高全员的稳定性和应急意识。

培训与分享：定期组织培训、技术分享会，提升团队的整体应急能力。

新人培训：在新员工入职培训中，强调稳定性的重要性和应急流程。

5.2.2.1.4 持续改进与演练

定期开展应急演练：模拟真实的故障场景，检验应急预案的有效性，发现并改进不足之处。

故障复盘和分析：每次故障处理后，进行深入的复盘，分析原因，提出改进措施。

持续优化应急流程：根据实际经验，不断完善应急响应流程和机制。

5.2.2.2 故障发现

5.2.2.2.1 监控发现

监控发现是指 SRE 通过监控主动发现系统运行异常的情况。为此，SRE 应考虑以下内容：

1. 监控指标体系设计

监控指标应按层次结构设计，形成一个监控体系金字塔。通常情况下，越靠近塔尖的指标越与业务紧密关联，用于感知用户的真

实体验；越靠近底层的指标则与系统运行环境相关，用于故障定位和根因分析。

SLI 定义：SRE 应优先考虑确定关键服务的 SLI（服务级别指标），这里的关键服务在不同类型业务会有很大的行业特性，如，电商行业会高度关注购物，支付等业务，游戏行业会关注登录，对局等业务。这些业务的指标选择有两大原则，一，优先选择与用户体验强感知的业务对应的指标，如请求成功率、响应时间等，二，能识别一个主体是否稳定，如错误率，容量指标，人工接入工单数等。

应用层指标：另外，对应用层的监控非常有价值，常见的应用层指标有：延迟、流量、错误率、饱和度，线程池、连接池等指标。

技术组件监控：对系统中各个组件的监控是确保系统稳定性的关键。组件监控应包括数据库、缓存、消息队列等关键组件的性能指标，如查询时间、命中率、队列长度等。

基础架构监控：基础架构监控覆盖服务器、虚拟化平台、容器云平台、网络设备、存储设备等硬件设施的健康状况和性能指标。常见的基础架构监控指标包括 CPU 使用率、内存使用率、磁盘 I/O、网络带宽等。

在覆盖率上，SRE 要确保所有关键服务和组件都在监控范围内，包括第三方服务和 API。

2. 指标标准化

SLO 的设定：与开发团队合作，基于 SLI 设定明确的 SLO（服务级别目标），如 99.9%的请求成功率或 200ms 以内的响应时间。SLO 是衡量服务质量的具体目标，设定目标有利于团队进行整体的协作。

统一框架：提供一个统一的开发框架，简化指标收集和监控配置的过程，确保关键性能指标的规范化。建议使用 OpenTelemetry 协议及 SDK 。

指标规范：制定明确的指标规范，促进跨应用和 IT 基础设施的协调一致。例如合适的指标名字以及合适的指标标签。

3. 告警触达

SRE 配置的监控策略，应以“故障影响程度”进行分级，不同级别的监控告警应通过电话、短信、IM、邮件等不同的渠道触及 SRE 人员，避免单一渠道告警堆积导致重要告警未被及时关注。当短时间内出现大量告警时，SRE 应对告警实现收敛能力，以便 SRE 可以对告警分类，及时梳理出故障范围。

4. 告警轮值

为了降低 SRE 长期面对告警的心理压力，在团队规模允许的情况下应实施告警轮值。当值 SRE 作为告警的第一责任人，负责组织开展后续故障处理事项。告警触达在第一责任人（A 角色）未在约

定时间内响应的时候，可以继续通知第二负责人（备份角色、B 角色）或者第三负责人（C 角色）等。

良好的 SRE 实践应实现所有故障通过监控主动发现，以期达到 SRE 优先介入处理，降低对用户的影响。SRE 可以通过故障主动发现的占比，来衡量监控发现的覆盖程度，该比例越高越好

5.2.2.2.2 巡检发现

在业务系统实际落地过程中，难免存在预期以外的故障发生。所以在监控发现手段以外应安排人员对系统各个环节进行定期巡检，作为监控系统的补充。巡检与监控的区别主要有以下几点：

1）指标差异。监控更关注状态、性能、容量类指标，巡检可根据 SRE 自身工作需要或者业务需要设置更为个性化的指标（如 Linux 内核版本、Linux 内核参数、系统启动时长、文件系统用户权限等）

2）指标阈值不同。巡检往往设置比监控阈值更苛刻的指标，以便发现一些隐藏的风险。

3）周期不同。巡检一般按小时、按天、按周、按月甚至年度执行，监控一般在分钟或者秒级。

4）深度不同。业务系统、技术组件的深度巡检，通常可覆盖更复杂的隐患检查场景，例如，系统上下游处理容量、关键依赖的健康状态、关键技术组件的性能指标，可作为重要活动、重要节假日前的保障措施，也属于故障预防的一种手段。

SRE 可以考虑通过以下三个方面组织巡检：

1）巡检周期

针对不同业务模块的重要程度，SRE 可以对各个模块实施不同的巡检周期。对于直接影响用户使用体验的核心模块巡检频率应至少一天巡检一次，非核心模块可按业务特性考虑降低巡检频率。

2）巡检手段

SRE 应将需要巡检的内容提前整理为仪表盘或其他可视化内容，巡检人员应通过图表可直接了解当前系统指标是否满足 SLO。同时，SRE 应可方便地对历史数据与当前数据进行对比，可直接了解当前指标是否存在异常走势或变化

3）巡检目标

巡检是为了查找系统中未知的故障模式或者隐患，因此 SRE 除了关注 SLO 是否达标以外，尤其需要关注重要指标的历史趋势是否发生变化、是否存在指标毛刺等，以期通过主动巡检的方式及早发现系统故障或隐患。

良好的巡检可以让 SRE 早于用户发现系统异常，及时止损。因此，可以通过巡检发现的问题与人工上报问题的比例来衡量巡检发现的质量，巡检发现的比例越高质量越好。而从另一个角度，巡检发现是作为监控发现的补充，巡检发现问题与监控发现问题的比例越低表示监控发现越为完善，此时比例越低越好。

总的来说，监控则更侧重于实时的响应和问题解决，而巡检更侧重于预防性的维护。两者都是有效的保证稳定性的方式，通常同时使用。

5.2.2.2.3 人工上报（舆情，客服，运营人员等）

在实际运营中，尽管自动化监控系统可以覆盖大部分的故障发现场景，但总有些情况是自动化监控所无法捕捉的，比如用户体验上的微妙变化、业务逻辑上的复杂问题等。因此，人工上报成为监控系统的重要补充。

人工上报通常涉及以下几个方面：

1.舆情监测：通过社交媒体、论坛、客户反馈等渠道，对用户的讨论和意见进行监听，从中发现可能的系统问题或用户不满。

2.客服反馈：客服团队是与用户直接沟通的窗口，用户遇到的问题通常会第一时间反馈给客服。因此，客服团队需要有一套机制，将用户报告的问题快速传达给 SRE 团队。

3.运营人员反馈：运营团队在日常工作中也可能发现系统的异常情况，他们对业务的了解可以帮助 SRE 团队更快定位问题。

5.2.2.3 故障诊断

5.2.2.3.1 应急协同

系统架构与逻辑的复杂性越来越高，应急协同通常是一个跨团队、跨地域的协作过程，建立高效的应急响应机制是应急协同的重要工作。

1. 协同机制

- OnCall 值守：一二三线值守，分工明确；
- 应急响应：建立完善的应急响应流程；

- 环境资源保障：各环节资源投入及升级流程清晰，能快速提供需要的资源。

2. 岗位设置

1）应急总负责人（二/三级部门主管）

- 统筹协调应急响应工作
- 发起研究重大应急决策和部署
- 决定实施和终止应急预案

2）应急指挥小组（三/四级部门主管）

- 接受应急总负责人的领导，传达和落实应急总负责人的各项指令，汇总和上报应急信息
- 负责应急执行小组成员的协调沟通，协调应急事件处置工作中的重大问题
- 向业务及其他相关方同步必要的应急事件处理相关信息

3）应急执行小组（四级部门主管、工程师）

- 落实应急总负责人及应急指挥小组布置的各项任务
- 组织制定应急预案，并监督执行情况
- 掌握应急事件处理情况，及时向应急总负责人和应急指挥小组报告应急过程中的重大问题

向应急指挥小组确认事件恢复效果，关闭事件

4）值班人员（OnCall/值班人员）

- 响应所有事件，包括通过电话、自动监控等渠道上报的事件

- 完整记录所有接收的事件信息，包括：用户信息、事件描述、发生时间和地点等
- 为事件进行适当的分类、为事件分配优先级等属性
- 使用知识库等手段对事件进行初步诊断和分析，并进行尝试解决
- 必要时联系相关方参与事件处理
- 如果不能解决事件，应当将事件分配给合适的二线支持并及时升级
- 检查事件记录的处理进度，保持与用户的联系，适时通知事件处理状况

3. 协同过程

- 启动应急响应机制
- 组建应急指挥部
- 信息共享与同步
- 资源调配
- 决策与指挥
- 持续监测、评估、反馈

5.2.2.3.2 故障定界

在微服务架构下服务通常会有多个模块组成，当其中一个模块异常时往往会导致整个服务多个模块产生异常告警。而故障定界时指 SRE 确认故障是由哪个模块异常导致整体故障的过程。SRE 在进行故障定界的过程中，应注意以下事项：

1. 故障定界并非故障定位

故障定界时，SRE 应将精力放在确认故障模块上，为后续故障恢复提供操作依据，过程中主要追求更高的时效性。故障定位则主要偏向确认故障根因，为后续代码级或架构级别的调整提供依据，过程中追求的是准确性。因此，SRE 在故障定界的过程中需要时刻谨记首要任务是恢复中断的业务，而非对故障原因刨根问底，绝大多数的故障定位是可以事后做的。

3. 辅助定界手段

1）告警信息丰富

告警是 SRE 触达故障的关键渠道，也是直接有效又容易忽略的提升故障定界效率的第一步。告警信息除了异常本身的信息描述，（例如，CPU 使用率 100%）还可通过丰富告警对象信息、上下文信息，让 SRE 更快分析、定位产生告警的源头，例如，SRE 是否已经做到所有告警都能展示对应的主机信息、IP/容器信息，是否有告警对象的集群、主备信息，交易类监控指标是否能够同时在一条告警信息里，查看当时的技术成功率、延迟、流量、错误率、饱和度等信息。

2）CMDB 辅助定界

CMDB 是生产机房镜像库，SRE 需构建以配置为中心的运维一体化建设满足各个运维场景对象可以追溯到 CMDB CI。每条告警信息中携带了 CI，当有告警产生时，可以初步定界监控直接的问题对象，注意准确的定界还依赖于快速验证。

3）链路跟踪手段

链路跟踪通过对符合 OpenTelemetry 标准的原始链路日志进行实时采集与流式计算，还原精准链路拓扑，应用于全链路实时观测、基于链路的业务指标计算、故障定界。

链路跟踪准确定界故障源，比如应用通常会因在组件异常问题、交易相互依赖、因数据库造成上层应用队列堵、多只交易缓慢的问题等通过链路跟踪定界故障。

4）可观测故障决策树

在链路拓扑的基础上整合 CMDB 数据关联业务系统具体组件的日志、指标信息，故障定界时可以进行多层下钻挖掘故障源头，并通过关联故障排查脚本作业的方式支持在链路上做进一步的故障信息收集。结合 CMDB 拓扑、链路拓扑的组件关系、服务关系，将常规的故障排查步骤编排成故障决策树，将专家经验沉淀为系统判断规则基于可观测数据和排查任务反馈的信息进行故障定界判断

5）快速验证设想

当 SRE 有怀疑的故障模块时，应该通过一定的手段（如：调整流量、配置参数、增加实例数量等）对设想进行验证。SRE 需要注意验证过程中应只影响模块的部分实例、流量或请求。操作实施后，SRE 应通过故障观测手段来确认设想是否符合预期，如不符合预期应考虑是否操作参数需要调整或模块猜测存在偏差。

3. 明确定界故障边界

筛选已知故障定界场景，提前确定定界组件，SRE 明确在明确微服务架构下各个组件后，常见的如云 PaaS、数据库、中间件、主机、网络、存储等，整理筛选出可能的组件故障点，依赖于事先人工分析和以往故障案例分析形成故障边界。故障边界的确认是尽可能减少不确定性，把定界点考虑在前。

4. 复杂情况定界故障

对于复杂的故障场景，可能是多种组件故障混合的。一般可以通过 CMDB 拓扑链路关系确认最下层组件，首先分析最下层组件并进行快速恢复验证。

5. 详细记录操作步骤

SRE 应记录定界验证的操作内容，作为后续故障恢复或切换的依据。因此，SRE 应通过运维工具进行操作，避免由于上机操作导致操作记录散落无法进行归档。同时，详细记录的操作步骤可以避免当验证产生非预期效果时，或多个操作叠加产生负效果时，可作为应急回滚的依据。

故障定界是 SRE 故障处理中的核心过程，只有准确的故障定界才能保障后续有效的故障恢复方案。而且由于过程中会涉及操作行为，SRE 需要对一切的操作行为保持谨慎避免由于操作导致额外的故障发生。可以通过准确性、耗时和操作次数来衡量定界质量

5.2.2.3.3 影响评估（影响人数，范围，上报级别）

故障影响评估是多方面的，主要看对服务功能的影响、影响时长、故障发生的时段、对用户影响的范围。

（1）如果对应用服务无任何影响，可以快速恢复的生产问题的，这些问题 MTTR 为 0 且对用户无感，事后报告直接领导。

（2）对于应用服务有影响的，且影响到客户但范围可控的，定性为生产事件，需要及时上报到系统运维中心生产运行情况沟通群并报告处室领导（互联网行业一般到部门总监），并在事件问题群跟进事件解决进度。如果影响时间过长，需要进一步报告中心领导。

（3）对于应用服务有影响，且影响到客户且范围过大的，定性为生产事件，需要及时上报到系统运维中心生产运行情况沟通群并报告中心领导。在互联网行业，通常上报至部门总经理。

国家网信办就《网络安全事件报告管理办法（征求意见稿）》公开征求意见，按照《网络安全事件分级指南》，属于较大、重大或特别重大网络安全事件的，应当于 1 小时内进行报告。网络和系统归属中央和国家机关各部门及其管理的企事业单位的，运营者应当向本部门网信工作机构报告。属于重大、特别重大网络安全事件的，各部门网信工作机构在收到报告后应当于 1 小时内向国家网信部门报告。

5.2.2.4 故障恢复

在 SRE 中，故障恢复是一个重要的概念。当 SRE 收到监控告警信息并确认故障发生（排除误告警）后，应第一时间确认评估故障影响范围，同步故障到相应干系人，并根据相应的优先级对故障

进行恢复，并在恢复后需对故障原因进行根因分析，做好排除隐患，尽量避免二次故障。最大程度减少系统停机时间，降低 MTTR，保障业务可用性，保证业务连续性。

故障恢复通常包括架构自愈、应急预案、应急维护和恢复验证等方面内容。

5.2.2.4.1 架构自愈

故障是不可避免的，架构自愈的理念是在系统设计之初以及迭代过程中考虑系统的健壮性，使其具备完善的逃生能力，通过一系列技术手段来实现系统的高可用性，架构高可用是满足架构自愈必要条件，包括如下两个方面。

1. 部署架构高可用

在业务部署上通过跨交换机/机房/可用区/地域等粒度的容灾，如通过云的置放群组策略控制业务部署的亲和性，实现 IaaS 层部署架构的高可用；当 IaaS 层发生故障时，无需人工干预，可快速自动切换调度。常见的部署架构有两地三中心，即系统划分为三个数据中心，两个位于同城，一个位于异地，同城的两个数据中心分别承担主备角色，异地数据中心则作为备份，保证高可用性和容错能力。

2. 业务技术架构高可用

业务应通过不断迭代，优化业务架构，排除单点模块，实现业务本身高可用，如典型的负载均衡、多活、热备自动切换等实现故障转移。或者业务架构可以充分利用云的弹性伸缩 AS 以及云原生

能力（Workload、HPA/GPA/VPA），使用扩容、限流、降级等自动解决系统过载、流量增加等场景的故障。

5.2.2.4.2 应急预案（已知故障场景）

基于已知的常见故障场景，SRE 复盘历史上出现的生产环境故障，提取故障特征，梳理业务相关依赖（关联哪些设备、哪些组件、相关依赖）沉淀成专家经验的故障处理模板，落地到连续性管理平台（运维手册）中。SRE 通过对手动故障处理操作流程进行总结提炼，开发故障处理工具脚本，将其编排落地到自动化故障处理系统中。通过不断的迭代，丰富故障识别场景，实现当故障发生时，通过监控、链路跟踪等可观测指标匹配识别故障，执行自动化处理流程，无需人工干预进行故障恢复。

由于系统会经常地发生变化，所以相应的应急预案会可能出现老化失效的问题，因此，需要进行持续的版本迭代并进行演练。

5.2.2.4.3 应急维护（未知故障场景，人工干预）

针对无法自动化处理的故障，需要 SRE 快速人工介入处理，及时卷入周边关联团队协助处理，并按照故障响应处理的流程规范对故障进行升级，同时定期对故障处理进展信息进行同步。在故障恢复后对问题根因进行分析复盘，并评估是否落地到自动化故障处理中。

5.2.2.4.4 缓解及恢复的措施

缓解及恢复措施是在生产环境中，当系统发生故障或重大问题时，为降低用户影响、恢复业务功能而采取的紧急行动与恢复策略。其目标是通过快速“止血”减少问题影响，并在此基础上迅速实现服务的全面恢复。

核心原则：

1. 快速止血：优先采取缓解措施，减少用户感知的影响，而不必立即解决根本问题。
2. 服务恢复：在缓解措施生效后，尽快通过稳定的方法恢复系统正常运行。

典型缓解与恢复方法：

- 应用回滚：将服务恢复到已知的稳定版本。
- 数据回滚：还原数据库或配置到健康状态。
- 重启：重启相应的业务服务器及服务
- 熔断：通过启用熔断机制，在调用失败率过高时自动停止调用故障服务，避免连锁反应导致更大范围的系统问题。
- 限流：动态限制用户请求速率，防止高并发或恶意请求导致系统过载，确保关键功能的正常运行。
- 降级：关闭非必要功能，集中资源保障核心业务运行。
- 扩容：增加系统资源以应对高负载或资源瓶颈。
- 切流：将流量引导至备用系统或健康实例。

- 屏蔽：暂时屏蔽问题模块或功能，降低故障扩散风险。
- 隔离：将故障模块隔离，保护其他服务的稳定性。

以上仅仅是部分缓解与恢复的手段，实际应用中需要根据具体场景灵活选择或组合使用，以最大化降低影响并恢复服务。

宗旨：缓解及恢复措施的核心是通过临时性手段迅速减少故障影响，并争取更多时间来分析问题根因与设计长期修复方案。它强调行动的高效性与精准性，以最大化减少对用户和业务的干扰。

5.2.2.4.5 恢复验证

故障解决后，SRE 团队应对业务进行健康检查，验证故障恢复情况；持续观察一段时间，查看 SLI 各项指标是否正常，包括观察性能指标（延迟、吞吐量、处理速率、时效性），可用性指标，质量指标，运营指标等，确认服务已恢复正常。

5.2.2.5 故障复盘

复盘源于围棋术语，也称「复局」，指对局完毕后，复演该盘棋的记录，以检查对局中招法的优劣与得失关键。这样可以有效地加深对这盘对弈的印象，也可以找出双方攻守的漏洞，是提高自己水平的好方法。故障复盘是指生产故障发生后，对故障处理过程、影响和原因分析、制定有效改进措施的过程。故障复盘是一次宝贵

的从历史故障中汲取经验教训的机会，通过及时复盘，举一反三，确保下次类似的问题不会出现甚至扩大化，故障复盘对业务稳定性建设非常重要，技术团队需要不断加强和培育事后总结的文化，理清所有的根源性问题，最关键的是落地有效措施避免未来重复发生或降低故障重现的概率及爆炸半径。

故障解决后，需要及时针对引发本次故障的根因及处理过程做一次详尽的复盘，识别到其中的风险，并制定及跟踪后续对这些纰漏的修复及警示。故障复盘作为稳定性建设一个非常重要的环节，需要管理层自上而下的认同和支持，不能把事后总结当成例行公事，需要让工程师们真正认同事后总结是把系统服务变得更可靠的一次机会。任何故障发生后，都会产生一篇故障复盘的文档记录到故障管理系统中。针对复杂或者严重的案例，需要通过正式的会议形式组织故障复盘，提升工程师们重视度和面对面讨论引发更多深入思考。正式复盘会议之前通过标准化的故障总结模板收集复盘需要的信息，由故障相关的团队一起协作参与完成复盘材料的准备，会上主要还原事实，聚焦改进，针对故障关键信息达成共识，复盘过程中要避免相互指责，建立“对事不对人”的事后总结文化。

下面主要针对正式故障复盘会议的组织、流程和平台工具进行详细介绍说明。

5.2.2.5.1 复盘组织

故障复盘已成为一项标准的稳定性服务，需要关注服务的时效性，一般建议重大事故的复盘在根因明确后的第 1 个工作日内完成

复盘，非重大事故建议 7 天内完成复盘总结。正式故障复盘会议由质量管理团队组织和主持，会前发起正式的会议邀约，建议邀请故障相关方的技术研发、测试、SRE 等角色参加，同时建议按需邀请：网络、DBA、UED、业务运营、客服、值班长等角色。如关键角色因时间冲突不能到现场，建议更换时间，避免因关键角色的缺席导致信息的不对称，有重开的风险。重大故障的复盘务必邀请故障相关方有决策权的管理角色参加。

1. 复盘流程

下面我们将从盘前准备、盘中主持、经验传承三个方面来介绍故障复盘的流程，尤其是盘前准备对整体复盘质量至关重要。

2. 盘前准备

复盘组织人在故障复盘会正式召开前需按照标准的复盘模板准备好故障的复盘材料，增加正式故障复盘会议条理性，复盘模板包含但不局限于以下内容，复盘材料可由故障涉及相关团队一起按照统一复盘框架协同完成。

过程回溯：回溯问题是一个抽丝拨茧的过程，一般包含几个关键时间节点的信息回溯：故障注入、故障发生、故障发现、故障通告、故障响应、初因定位、故障止血、影响消除、根因定位等。

原因分析：回溯完成过程之后，可以为分别从故障根因、触发原因、造成影响放大原因三个层面进行剖析。通过 5-Why 方法抽丝剥茧层层扒开迷雾，找到问题的根源，从根源上制定有效改进措施，避免问题再次发生。

影响分析：故障影响分析帮助我们从更全面视角了解故障带来的损失，需要更关注每一起故障对客户和业务造成的影响，如是否产生了社会舆情及客诉，是否产生了业务的资损等；其次是技术视角，系统的可用性是否受损，关键技术指标下跌程度。

经验总结：为了避免核心关键优化措施遗漏，可以按照故障发生的整个生命周期进行讨论分析，并落地执行：一方面从系统质量层面关注需求设计和评估是否充分，代码是否经过 Code Review 并充分测试，变更执行过程是否有效灰度和观测，是否有回滚预案；另一方面关注故障处置方案是否合理，如故障是否监控发现，是否及时应急，是否有更快的恢复方案等。为避免类似的故障重复发生，需要给出短期治标方案，又需要长期治本方案，以及沉淀经验和教训。

3. 盘中主持

故障复盘主持人需要对故障信息有全局性的理解，在会议中能有效识别并记录故障关键信息和改进点。针对大家的讨论进行全程的控场，能收能放，当大家针对某一个问题的讨论过于发散，要及时控场，引导发言的同学聚焦故障相关核心问题讨论；对于重大责任争议故障，建议复盘主持人邀请高级管理层参加，建立“聚焦改进、对事不对人”的复盘氛围，避免相互指责，鼓励各自关注自己团队的问题和改进。

4. 经验传承

针对一些经典的故障分析，可以通过内部的运营平台或者技术论坛，进行更大范围的公布和分享，目的是希望大家都能够“以史为鉴”，对共性问题进行反思，形成故障深入复盘的文化氛围，更多的部门和团队可以从经典故障案例中汲取教训，从而获益，共同推动线上业务的稳定性建设。

5. 平台工具

我们首先需建立故障总结的标准模板，通过流程管理工具将故障模板中的关键节点信息，如故障的描述、过程节点、影响及原因分析等结构化沉淀到系统中，我们一般该系统为故障管理系统，基于故障管理系统沉淀历年的故障复盘记录，基于故障管理系统沉淀的历史故障，数据结构化和线上化，可以进行更多维度的故障数据分析，将分析结果分享和赋能给工程师们，进一步提升业务的稳定性。同时基于故障管理系统我们可以做更多场景的拓展，如通过自动化数据收集，一方面提升故障记录效率，也可以将故障复盘记录和变更信息、定位和恢复信息，以及业务应用等信息进行关联，提升变更质量，提前做好变更观测，也能推进定位及恢复系统建设，提升恢复时效。故障管理系统虽然只是一个流程管理工具，却为我们很多系统稳定性工具建设提供了丰富实战数据来源。

5.2.2.5.2 根因分析

1. 根因分析的目的

根因分析指为了确定适合的解决方案而探查问题根本原因的方法，用以逐步找出问题的根本原因并加以解决， 而不是仅仅关注问

题的表征。与解决临时问题和被动应对相比，以系统化方式确定和分析问题原因，找出问题解决办法，并制定问题预防措施等，使得对解决根本性问题更为行之有效。

2. 根因分析的步骤

因为引起问题的原因通常有很多，物理条件、人为因素、系统行为、或者流程因素等等，通过科学分析，有可能发现不止一个根源性原因。首先我们需要找出：定义问题（发生了什么）、找出原因（为什么发生）、改进措施（什么办法能够阻止问题再次发生）。

1）如何定义问题

在根本原因分析之前，必须确定并定义发生的问题，可以通过提出以下几个问题来探寻根因：

- 谁首先发现了这个问题？
- 到底发生了什么，尽可能详细地描述？
- 在这个过程中的什么时候发现了这个问题？
- 问题是什么时候发现的？
- 到目前为止发生了多少次？
- 有没有发生的模式？
- 问题是如何被检测到的？

通过收集这些信息，找出具体的细节，以便对问题有更全面的了解。此时可能需要采取短期遏制措施或纠正措施。将需要审查所

有收集的信息，以便根据事实和数据确定可能存在的问题，一旦定义清楚了发生的问题，就开始了根因分析过程。

2）如何找出根因

进行根本原因分析时，一种较为常见的技巧是“5 问法”，得到每个问题（“为什么”）的答案后，继续提出更深入的问题（“好，那为什么”）。通常情况下，经过大约五次提问即可找到最根本的原因，但具体次数取决于实际情况：有时 2-3 次便足够，有时可能需要多达 50 次。

举个例子：淘宝商品详情页面无法展示库存数量。

首先立刻可以反应出的一个问题：为什么库存数无法展示？这是第一个“为什么”。

第一个答案：因为库存调用链路异常。

第二个“为什么”：为什么库存调用链路异常？

第二个答案：因为 A 服务调用 B 服务未得到响应。

第三个“为什么”：为什么 A 服务调 B 服务会出现无法响应？

第三个答案：因为 A 服务的机器异常导致服务不可用。

第四个“为什么”：为什么 A 服务的机器会异常？

第四个答案：因为 80%左右机器出现了 FullGC。

第五个“为什么”：为什么机器会出现 FullGC？

第五个答案：因为今天的代码发布引入了一条全表扫描，慢 SQL 导致请求堆积，导致机器出现 FullGC。

3）制定改进措施

我们找出问题根因的核心目的是针对根因制定改进措施，避免同样的问题再发生。

5.2.2.5.3 制定改进

1. 举一反三

故障改进是故障复盘中的关键一环，故障改进不仅仅是处理当下问题，一定要去看本质的问题，目的是降低故障再次发生的概率，但降低概率不等于不会发生，当故障真的来临的时候，也需要有足够的能力和定力来应对。

从问题入手，是治标；从根源入手，不仅可以解决眼前的问题，还可以解决更多隐性的未知问题，是治本。故障改进工作就是一次对系统和业务问题的治本思考。

2. 如何做好故障改进

首先重点要关注“谁？什么时间？完成什么任务？交付什么结果？”

改进项必须有负责人，一个任务可能多人配合，但必须指定一个主要负责人，负责人对任务的过程、结果负责。

改进项必须明确完成时间，需要明确给出改进项在某个具体时间节点的完成时间。

明确改进的输出，任务的目标、交付结果必须明确，作为任务完成的验证标准，且可衡量。如设计方案的改进措施，要求输出方案文档，可以进一步明确方案文档必备的内容等。

同时一个好的改进措施需要符合 SMART 原则：

Specific：即改进项。我们需要改进的事项或指标。

Measurable：即验收标准。改进完成后通过评审、演练、主管确认等方式验收。

Attainable：即改进项是否可落地。需要避免出现一些喊口号、无法落地的改进。

Relevant：即和故障本身有相关性。避免已在计划内的或非故障识别的改进项纳入充数。

Time-bound：即预期解决时间。建议最长不要超过三个月，避免改进期间故障再次发生。

3. 改进措施的记录

为确保改进项得到有效跟踪和验收，需要结构化地记录在系统中，一个完整改进措施的内容包括以下内容：

标题

计划完成时间

负责人（及其团队或协助处理人）

验收方式及验收人

跟踪人

改进措施的类别

具体改进内容描述及验收标准

4. 改进措施的分类

措施的分类主要有基础架构类、产品设计类、发布类、变更执行类、编码类（前端代码、客户端代码、服务端代码）、测试类、监控类、容量/灾备类、安全类、流程规范类、数据库类、中间件类等。在每次故障复盘的过程中，监控类是每一起故障必考虑的改进项，技术同学会从精益求精的角度提升系统监控发现，及时有效预警，为业务的恢复争分夺秒。

5.2.2.5.4 问题跟踪

故障复盘梳理出来的故障报告需要细化事件问题跟踪项，包括事故情况、根因、责任人和后续改进事项。首先距离落地还需要持续跟踪明确处理人，预计完成时间等，避免事故跟进事项成为空谈。其次，SRE 建立定期提醒机制，确保事故跟进事项的落地。一些事后问题修复的跟进可以通过文档管理平台落地开发任务。

5.2.3 故障应急工程体系设计

5.2.3.1 面向故障应急的监控设计

在 SRE（Site Reliability Engineering）中，面向故障应急的监控设计是确保系统高可用性和业务连续性的重要组成部分。故障应急监控设计的核心目标是能够及时发现潜在问题，迅速定位故障根因，并且通过有效的告警机制和监控覆盖，保障故障响应的及时性和准确性。

以下是对面向故障应急的监控设计的详细扩展说明。

(1) 监控体系结构设计

监控体系应按层次结构设计，形成一个完整的监控金字塔模型，涵盖从底层基础设施到用户体验的各个层级。具体设计包括：

用户体验层监控：在监控体系的顶层，监控应聚焦用户体验相关的指标，如请求成功率、响应时间、系统可用性等。此层监控旨在直接感知用户的真实体验，通过 SLI（服务级别指标）和 SLO（服务级别目标）来衡量服务的整体质量。

应用层监控：中层监控覆盖应用服务的性能指标，如延迟、错误率、吞吐量和饱和度等黄金指标。这些指标能够反映应用的健康状态，并帮助识别潜在的性能瓶颈和资源限制。

组件和中间件监控：进一步下层，监控应涵盖关键组件和中间件，如数据库、缓存、消息队列等。常见的监控指标包括数据库查询时间、缓存命中率、消息队列长度等，这些指标对于保障系统的稳定性至关重要。

基础设施层监控：最底层的监控体系应覆盖服务器、网络设备、存储设备等硬件设施的健康状况。此类监控包括 CPU 使用率、内存使用率、磁盘 I/O 和网络带宽等指标，确保基础设施的稳定运行。

(2) 指标标准化与统一框架

为了确保监控指标的可操作性和可比性，必须在组织内部推动指标的标准化：

SLI 和 SLO 的设定：SRE 团队应与开发团队合作，基于 SLI 设定明确的 SLO。这些 SLO 不仅要符合业务需求，还要考虑用户体验的敏感性，如请求成功率的目标设定为 99.9%或更高，响应时间限制在 200ms 以内。

统一监控框架：通过使用统一的监控框架（如 OpenTelemetry），简化不同系统和服务的指标收集与监控配置过程。统一框架有助于促进跨团队的协同，避免数据协议不一样而构成的成本。

指标规范化：制定组织内部的监控指标命名和标签规范，以确保不同团队和系统间的监控指标能够互通，并且便于后期的分析与调优。

(3) 多层级告警策略

为了在故障发生时快速响应，告警机制应具备分级处理和多渠道通知的能力：

告警分级策略：根据故障影响的严重程度，将监控告警分为多个级别，并定义不同级别的响应策略。例如，最高级别的告警可能需要通过电话和短信同时通知值班 SRE，而次级告警可以通过 IM 或邮件通知。

多渠道告警通知：为了防止单一通知渠道的拥堵或失效，告警应通过多个渠道同步触达相关人员。SRE 团队可以利用电话、短

信、IM、邮件等多种手段，确保告警信息在第一时间得到关注和处理。

告警收敛与分类：在高峰期，系统可能会产生大量告警。此时需要通过告警收敛技术，将同类告警合并，以减少告警噪声。SRE 团队还应对告警进行分类，快速识别出影响范围最大的故障，从而优先处理。

(4) 轮值与应急响应机制

为减轻 SRE 团队成员的负担，建议实施告警轮值机制：

告警轮值：设立轮值制度，确保团队成员轮流作为当值 SRE，负责第一时间响应告警。轮值制度不仅可以减轻个体的心理压力，还能保障每个故障都有专人负责。

应急升级：当值 SRE 在规定时间内未能响应告警时，系统应自动升级通知至备份 SRE 或更高级别的管理人员，确保告警不被遗漏。

(5) 综合监控覆盖与冗余设计

为了实现故障的全面监控，SRE 团队应确保监控的全面覆盖：

关键服务与组件的全覆盖：所有与业务密切相关的服务、组件和第三方 API 都应在监控范围内，确保无论何种故障发生，都能在监控体系内被及时发现。

冗余监控设计：针对特别重要的业务系统，建议设计冗余监控路径，例如使用多个独立的监控系统交叉监控同一业务，以防止单一监控系统失效导致的漏报。

通过以上设计，SRE 团队能够构建一套面向故障应急的高效监控体系，从而显著提升系统的故障发现与响应能力，确保业务的连续性和高可用性。

5.2.3.2 面向故障应急的作业平台设计

面向故障应急的作业平台设计是 SRE 体系中的核心环节，旨在通过自动化、标准化和流程化的手段，提高故障处理的效率和准确性。一个高效的故障应急作业平台应能够整合监控、告警、诊断、恢复和复盘功能，为 SRE 团队提供一站式的故障处理支持。以下是对面向故障应急的作业平台设计的详细扩展说明。

(1) 作业平台的功能模块设计

作业平台应包含多个功能模块，分别负责不同的故障应急任务：

监控与告警集成模块：此模块负责与监控系统（如 Prometheus、Grafana）和告警系统（如 PagerDuty、Opsgenie）进行集成，确保所有告警信息能够实时同步到作业平台中。平台应支持

告警的自动分类和优先级排序，便于 SRE 团队快速识别和处理最紧急的故障。

自动化作业模块：该模块提供自动化操作的支持，包括自动化脚本执行、批量操作和预定义的应急流程调用。自动化作业模块可以大幅减少手工操作的错误概率，提高故障处理的效率。常见的自动化操作包括服务重启、资源扩容、流量切换等。

恢复与切换模块：当故障定位完成后，平台应支持一键恢复或切换操作，帮助 SRE 迅速执行应急预案。恢复与切换模块应与自动化作业模块高度集成，确保关键操作的安全性和可控性，同时支持回滚机制，以应对未预期的操作失败。

(2) 自动化与流程编排

自动化是提升故障处理效率的关键，作业平台设计应支持高度自动化的流程编排：

预定义应急流程：平台应支持 SRE 团队预定义各种常见故障场景的应急流程，包括诊断步骤、自动化操作和通知路径等。预定义流程应可以灵活调整，适应不同故障的特定需求。

动态流程调整：在故障处理过程中，作业平台应允许 SRE 根据实际情况动态调整应急流程，添加或跳过某些步骤，以应对复杂或

未预见的故障场景。流程调整应具有日志记录功能，确保所有操作均可追溯。

多级流程审批与控制：对于高风险操作，平台应支持多级审批机制，确保在关键步骤执行前获得必要的授权。此功能有助于避免因误操作导致的故障扩大或其他次生问题。

(3) 可视化与交互设计

一个高效的作业平台应具备良好的可视化与交互设计，使 SRE 团队能够直观地了解系统状态并高效执行操作：

仪表盘与实时监控：作业平台应提供综合的仪表盘，展示关键性能指标、告警信息和故障处理进展。实时监控数据应与仪表盘集成，使 SRE 团队能够实时观察故障处理对系统状态的影响。

交互式操作界面：平台应设计为用户友好的交互界面，支持拖拽式的流程编排和一键执行操作。复杂的操作步骤应通过可视化流程图展现，帮助 SRE 团队理解操作顺序和影响。

移动端支持与跨设备操作：为了提升应急响应的灵活性，作业平台应支持移动端操作，使 SRE 团队可以在任何地点、任何设备上

执行紧急操作。跨设备的无缝切换确保在紧急情况下，团队成员能够迅速接管并继续故障处理。

(4) 安全与权限管理

作业平台必须在设计中充分考虑安全性，确保故障处理过程中的操作安全、权限控制合理：

分级权限管理：平台应支持精细化的权限管理，确保只有授权人员才能执行关键操作或访问敏感数据。权限管理应根据角色进行分配，确保不同级别的 SRE 和管理人员能够在权限范围内执行相应操作。

操作日志与审计：所有在平台上执行的操作应被详细记录在操作日志中，支持事后审计和回溯。日志数据应被妥善保存，并具备查询和导出功能，以便在安全审计和复盘中使用。

安全加固与防护机制：平台应具备多重安全防护机制，包括但不限于身份验证、双因素认证、IP 白名单等，确保作业平台的安全性不受外部威胁的影响。

(5) 平台集成与扩展性

为适应不断变化的技术环境和业务需求，作业平台应具备良好的集成与扩展能力：

第三方工具集成：平台应支持与多种第三方工具和服务集成，如 CI/CD 工具、云服务平台、配置管理工具等。通过 API、Webhook 等方式实现数据和操作的无缝衔接，提升平台的功能多样性。

模块化扩展：作业平台应采用模块化设计，允许 SRE 团队根据需要添加或移除功能模块，以适应不同规模和复杂度的系统需求。这种设计使得平台能够随着业务的增长和技术的演进进行快速扩展。

微服务架构与容器化部署：平台应支持基于微服务架构和容器化的部署方式，确保平台自身的高可用性和可扩展性。在云原生环境中，作业平台应能够自动伸缩，适应不同规模的负载需求。

通过以上设计，SRE 团队能够打造一套高效、可靠的作业平台，有效支撑故障应急响应的各个环节，从而大幅提升系统的稳定性和业务的连续性。

5.2.3.3 面向故障应急的 ITSM 设计

在 SRE 体系中，面向故障应急的 ITSM（信息技术服务管理）设计是确保系统稳定性和业务连续性的关键环节。故障应急的 ITSM 设计应围绕“快速响应、有效处置、持续改进”的目标，提供故障处理流程的全生命周期管理。以下是对面向故障应急的 ITSM 设计的详细说明：

(1) 故障处理流程管理

ITSM 系统应具备完善的故障处理流程管理能力，确保故障从发现到解决的整个过程都有清晰的操作路径和责任分配。关键要素包括：

故障报告与记录：支持故障自动检测和人工上报，记录详细的故障信息，包括故障类型、影响范围、发生时间、相关组件等。

故障分类与优先级设定：依据故障的影响程度和紧急性进行分类，如关键故障、严重故障、普通故障，并设定处理优先级，确保关键故障优先响应。

故障处理流程：预定义故障处理流程，涵盖故障的初步诊断、详细分析、修复操作、验证与关闭等步骤。流程应具备灵活性，允许根据实际情况动态调整。

(2) 故障应急响应机制

为了在故障发生时迅速响应，ITSM 系统应具备高效的应急响应机制：

多层级响应机制：根据故障的严重程度，启动不同层级的应急响应。一般分为一线响应、二线技术支持、三线专家支持，确保复杂问题能够快速升级并得到有效处理。

故障升级与通知：在故障未能在规定时间内解决时，系统应自动升级通知至更高级别的支持团队或管理层，并通过多渠道（电话、短信、IM 等）确保关键人员及时响应。

协同处理与资源调度：支持跨团队的协同处理，提供实时的沟通工具和资源调度功能，确保在故障处理过程中各相关方能够有效合作。

(3) 故障知识库与决策支持

知识库是 ITSM 系统中的核心模块，记录了历史故障的处理经验和最佳实践，支持在故障处理中快速参考和应用：

知识库构建：整理和归类历史故障处理案例，涵盖故障原因、影响范围、解决方案、恢复时间等信息。知识库应按故障类型、影响范围、组件类型等维度进行分类。

智能推荐与自动化：利用 AI 技术，从知识库中智能推荐最优的故障处理方案，并支持自动化执行常规操作（如服务重启、流量切换），提高故障处理效率。

持续更新与优化：故障处理结束后，系统应自动将处理过程和结果更新至知识库，确保知识库内容的实时性和实用性。

(4) 故障复盘与改进管理

故障复盘是提升系统稳定性的重要步骤，ITSM 系统应支持对故障的全面复盘与改进管理：

复盘流程：在故障解决后，系统自动启动复盘流程，收集故障处理的所有相关数据，包括操作日志、故障影响评估、处理步骤等。

根因分析与改进措施：采用 5 Whys 等分析方法，深入探究故障根因，制定并追踪改进措施，确保类似故障不再发生。

改进执行与验证：针对复盘确定的改进措施，系统应生成具体的任务清单，分配责任人，并设定完成时限。完成后，通过验证机制确认改进效果。

(5) 故障应急演练与预案管理

为了确保应急响应的有效性，ITSM 系统应支持故障应急演练和预案管理：

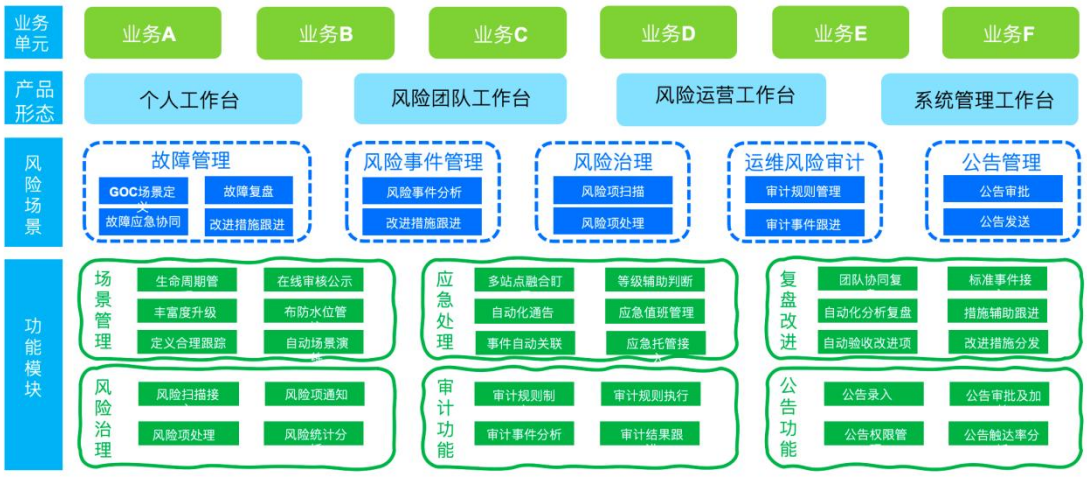
应急预案管理：建立涵盖不同故障场景的应急预案库，定期更新并与实际故障处理经验相结合，提升预案的实效性。

演练计划与执行：定期组织故障应急演练，通过模拟真实场景测试预案的有效性，演练结果应记录在案，并用于优化应急预案和处理流程。

演练结果分析：对演练结果进行分析，识别潜在的问题和改进机会，确保在实际故障中能够快速而有效地应用演练中的经验。

通过上述设计，SRE 团队能够构建一套面向故障应急的高效 ITSM 系统，有效提升系统的故障响应能力和业务连续性。

5.2.3.4 面向故障应急的风险管理平台



技术风险管理平台应面向各个业务部门，涵盖了众多业务单元（BU）。他们均能在该平台上进行操作。产品形态主要服务于普通的 SRE（Site Reliability Engineer）人员以及业务接口人员。稍后我

会介绍我们的整体应急响应架构。针对不同的用户角色，我们设计了多种工作台形式。例如，一些负责 GOC 的同学或类似职能的同事可能会在系统中输入特定故障定义的场景，为此我们建立了风险运营机制。

故障管理：故障场景、故障定义等信息用平台沉淀起来，同时通过故障复盘、Action 跟进之类的的管理能力整体形成功能闭环

风险事件管理： 风险事件是指未被定义为故障、影响较小的业务事件，可以归类为风险事件。

风险治理：集中处理一些风险巡检系统的风险产出物，业务稳定性同学可以基于风险项信息进行风险的跟踪与治理

运维风险审计： 对违规操作，进行审计。

公告管理：平台提供公告功能，用于发布重要的稳定性信息。

5.3 故障应急案例

5.3.1 小米故障应急响应经验分享

SRE Elite 精选原因

小米拥有很强的硬件基因文化，因为如果硬件出现质量问题，相关的修复成本将会非常巨大。所以其质量有独特的要求，小米拥有独立 QA 团队，对运维质量进行考核及管控，构成了其独特的故障管理体系以及复盘的体系,可供有类似业务特性的组织进行参考。

（一）背景及设计原则

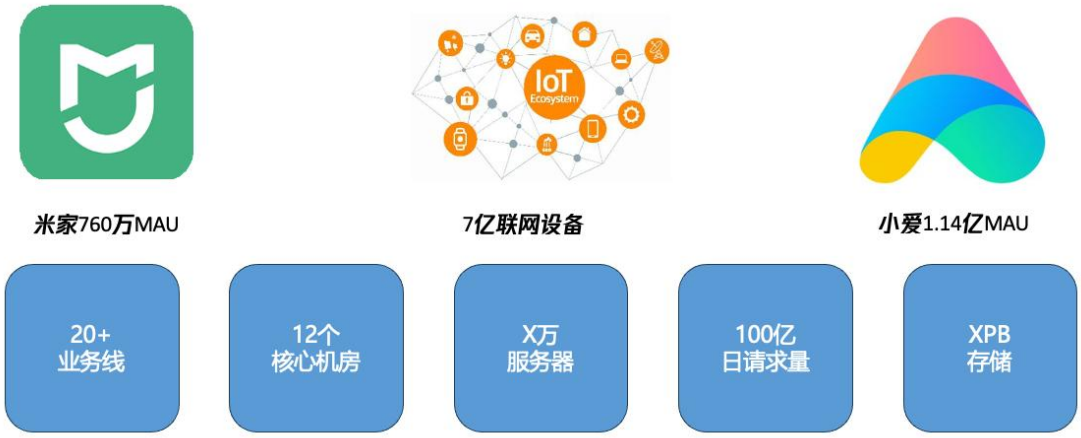
小米的最新战略目标是构建一个“人-车-家”全生态系统（Human x Car x Home），旨在满足全行业的需求。我们希望在这个生态系统中，能够像星辰大海一样广泛覆盖，做好每一个细节。为了体现我们小米 SRE 工作的复杂度，以下是一些关键数据：



- 我们拥有 6.99 亿台智能设备。
- 涵盖 200 多个智能硬件品类（仅小米品牌，不包括小米生态链）。
- 小米生态链大约覆盖了用户生活场景的 95%。

小米 SRE 面临的挑战

小米作为全球第三的手机制造厂商&全球领先的消费级 AIoT 平台，SRE 挑战巨大



● 目前有 7 亿联网设备（国内和国际）。

● 米家平台的月活跃用户数（MAU）为 760 万，作为互联互通的中枢。

● 小爱的月活跃用户数为 1.14 亿。小爱不仅仅是一个音箱，其后端也在适应 ChatGPT 的大潮流。

● 我们有 20 多个业务线，可以认为是集团级的组织单元。

● 全球范围内有 12 个核心机房，服务于各个地区。

● 使用了七家公有云，阿里云，金山云，腾讯云，火山云，AWS，Azure，GCP，积累了丰富的多云管理经验，但是在过程中也因为各云标准不统一在对接过程中非常痛苦

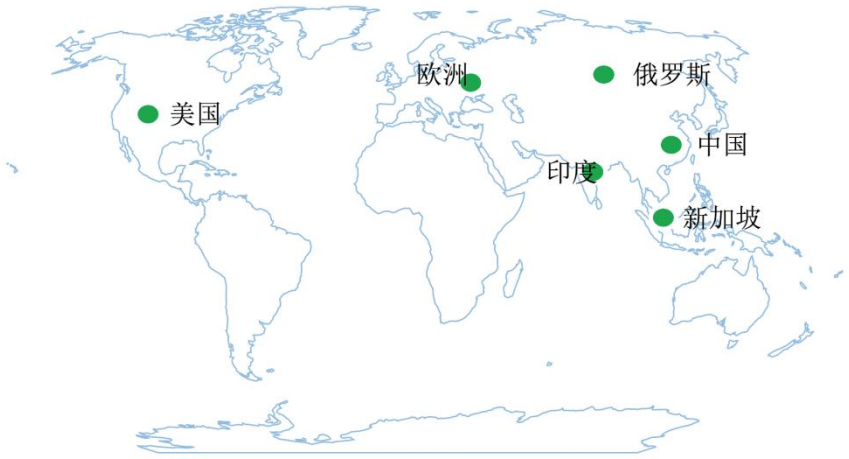


在故障应急方面，我们的目标是做到快速发现、快速定位和快速控制损失。具体而言，阿里的专家提出的经典目标是：1 分钟内发现故障，5 分钟内定位故障，10 分钟内控制损失。这也是我们 SRE 团队的追求目标。

关键能力点拆解

为了实现上述目标，我们需要拆解出几个关键的能力点：

- 故障发现：及时发现故障是应急处理的第一步。
- 故障诊断：在发现故障后，迅速进行故障诊断，确认故障场景。
- 故障恢复：在确认故障场景后，快速进行故障恢复。
- 闭环管理：在故障处理完成后，进行故障复盘，做好问题管理和缺陷管理。



（二）解决方案详情及案例

构建哪些能力应对故障响应

1. 故障发现

故障发现是应急响应的第一步，包含监控、巡检和故障上报三个方面：

监控：

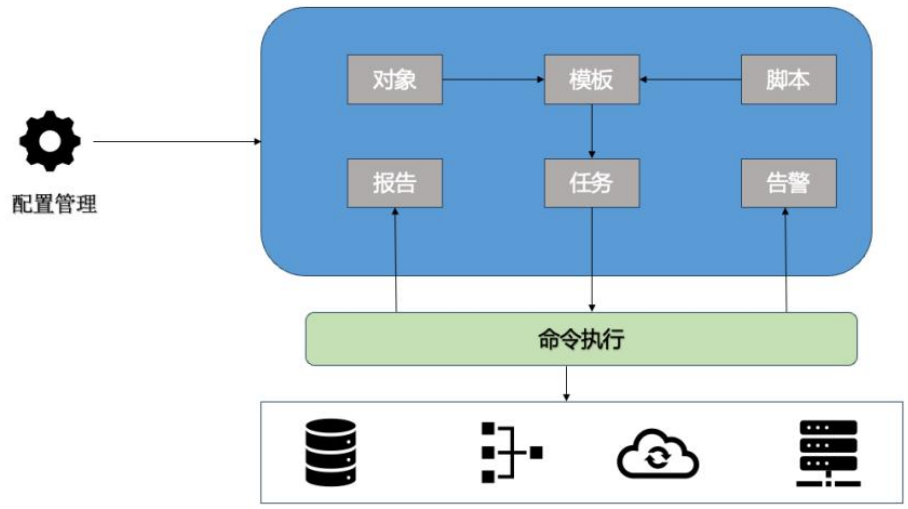
使用各种监控工具如 Prometheus、Grafana、DataDog 等进行基础设施、程序、日志、用户体验和网络性能的监控。

- 基础设施监控：混合云环境下的 IT 基础架构信息实时监控（性能/可用性）
- 应用程序性能监控：应用程序的功能/运行状况监控，APM，跨微服务，主机，容器和 Serverless

- 日志管理：为应用，系统和云服务提供日志管理，可创建索引，查询可视化
- 报警：支持机器学习，实现预测和阈值自定义等能力
- 用户体验监控：模拟客户监控产品的体验和可用性的真实用户监控
- 网络性能监控：云或者混合环境的网络流量进行分析和可视化处理

业界中监控的相关工具很多，但关键在于将监控系统与企业的 CMDB 资源数和业务流程系统协同，把产品用好，不停提高产品的易用性。

巡检：



以上是一个经典的巡检模型

抽象巡检配置：首先，我们需要定义和抽象出巡检的配置。

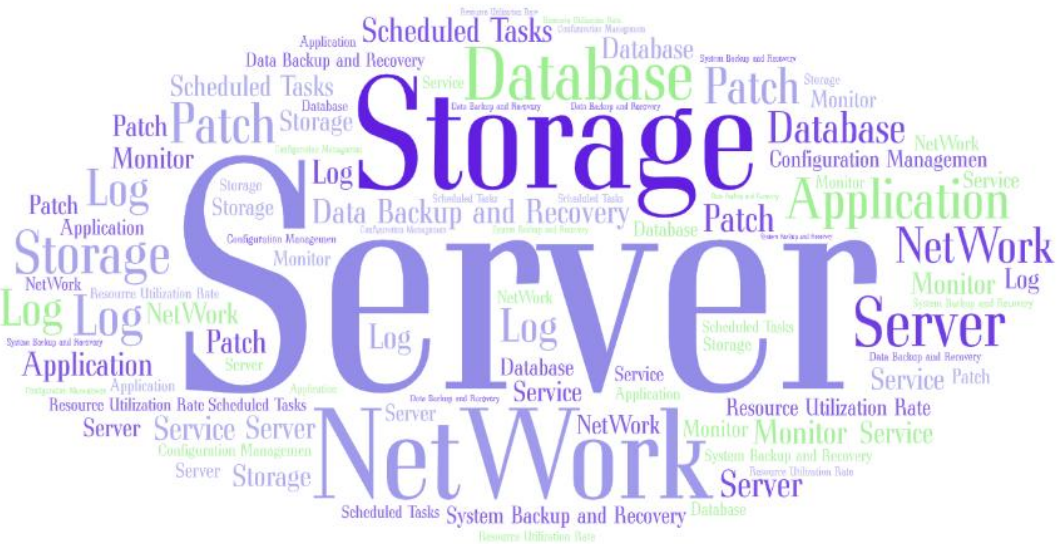
抽象巡检对象：接着，明确巡检对象的范围和具体内容。

抽象巡检模板：然后，制定巡检的模板，以便标准化巡检流程。

在实际操作中，我们可以借助命令执行系统（例如蓝鲸的作业系统）来进行巡检。这些巡检对象包括各种资源、数据库、网络、云服务以及基础设施主机等。

通过这些指标，我们可以全面监控和维护系统的健康状态，确保服务的稳定性和可靠性。

以下是我们根据我们巡检指标所构建的词图，包括数据备份，数据库， 网络， 日志，补丁等等。



故障上报：

自动化监控无法解决所有问题，用户体验上报是重要的补充手段。关注舆情监测、用户投诉和客服统计数据，与运营角色保持信息互通。

- 舆情监听
  - 媒体信息收集
  - 论坛讨论热点
- 客服反馈
  - 用户投诉上报
  - 客服统计数据
  - VIP 重点服务关注
- 运营反馈
  - 新功能上线检查
  - 日常操作反馈

2.故障诊断

每一位资深的 SRE 同事都必须牢记这句话：如果在发生故障的第一时间无法立即恢复，那么在故障排查过程中，一旦想到能够恢复故障的方法，应优先进行故障恢复，而不是急于找到故障的根本原因。在故障诊断阶段，我们应从以下四个方向进行拆解：

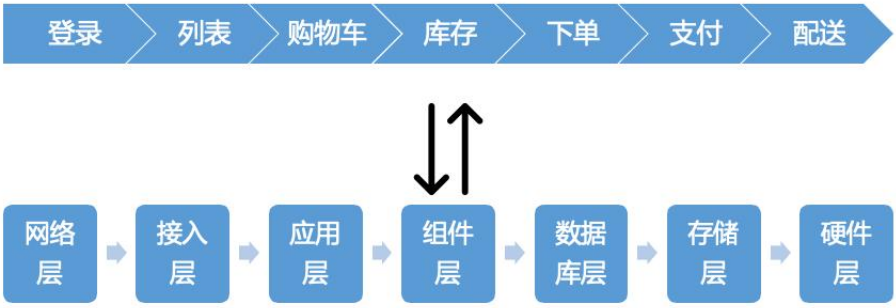


- 故障范围：确定影响模块及其逻辑功能依赖和上下游服务依赖。
- 根因确定：通过关键告警和日志分析，找到故障的根本原因。
- 影响评估：评估故障的影响范围，及时向上级报告关键信息。
- 组织能力：在故障场景下需要强大的组织协调力，确保各部门协同工作。

故障范围界定

作为 SRE 团队的基本理念，每位同事脑海中都必须清晰地记住两个关键概念：横向和纵向的依赖关系。

关键词：SRE向上要理解业务功能模块，向下要精通云基础技术组件

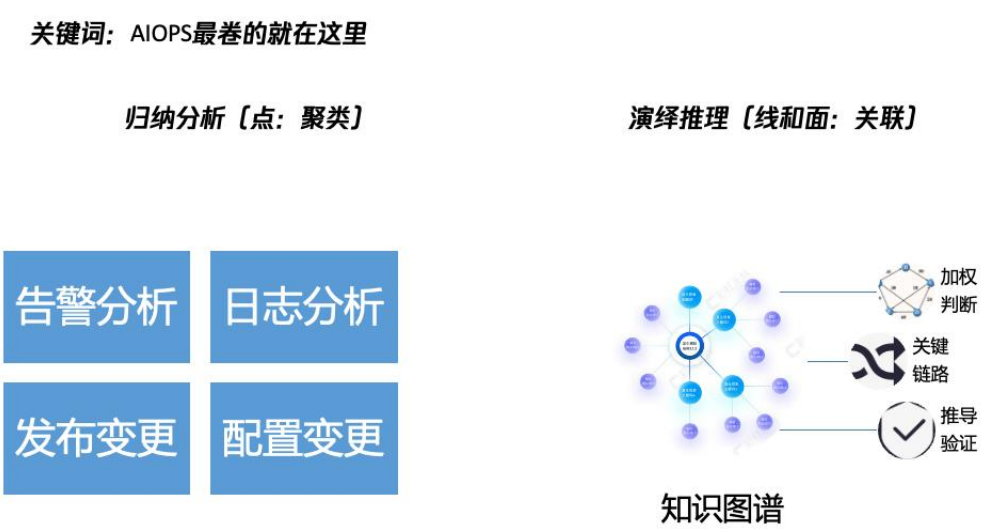


横向依赖指的是业务层面的逻辑关系。例如，在一个典型的商城系统中，业务逻辑的依赖关系包括登录、商品列表、购物车、库存管理、下单、支付和配送。这些模块之间存在着前后端和上下游的依赖关系，理解这些关系有助于我们快速识别和定位故障。

纵向依赖涉及系统的技术层级，从底层到顶层的依赖关系，包括网络层、接入层、应用层、组件层、数据库层、存储层和硬件层。每位 SRE 同事应熟悉这些层级的依赖关系，了解每一层之间的交互和依赖。

基于上述横向和纵向的依赖关系，我们能够更准确地界定故障范围。在发生故障时，清晰的横向业务逻辑和纵向技术层级的结构图能够帮助我们迅速确定故障点，评估影响范围，并采取有效的恢复措施。

根因确认



在故障诊断和根因确认方面，无论是依靠人工还是借助 AIOPS，核心的方法其实可以归结为两大类：归纳分析和演绎推理。

归纳分析的核心在于将分散的信息聚合起来。例如，当我们收到大量的告警日志时，首先需要通过归纳分析将这些零散的告警信息进行汇总和筛选。这个过程的目标是过滤掉无关紧要的噪声，提取出有价值的告警信息。通过这种方式，我们可以更高效地识别出潜在的问题区域。

演绎推理则是基于归纳分析得出的点，通过逻辑推理将这些点串联起来，形成一个完整的故障路径。知识图谱是演绎推理的一个典型应用。通过知识图谱，我们能够将归纳分析得到的关键点有机地结合在一起，构建出一个系统的故障模型。这种模型不仅能够帮助我们找到故障的关键路径，还能最终确认故障的起因。

总的来说，归纳分析和演绎推理是故障诊断过程中的两个重要步骤。归纳分析帮助我们从大量的告警信息中提取出有价值的数

据，而演绎推理则帮助我们将这些数据进行逻辑串联，找到故障的根本原因。通过这两种方法的有机结合，我们可以更加准确和高效地进行故障诊断和根因确认。

影响评估

关键词：及时升级！



在故障处理过程中，影响评估是一个至关重要的环节。对于那些没有多年经验的 SRE 来说，可能对这部分的认知还不够深刻。然而，影响评估实际上决定了故障处理的优先级和策略，影响到整体的决策过程。

首先，影响评估帮助我们确定故障的严重程度。这包括评估故障对业务运营的影响、对用户体验的影响以及对公司声誉的影响。例如，一个影响支付系统的故障可能比一个影响非关键功能的故障

更为严重。通过评估故障的影响范围，我们可以决定是否需要将故障上升到更高的优先级，并调用更多的资源来处理。

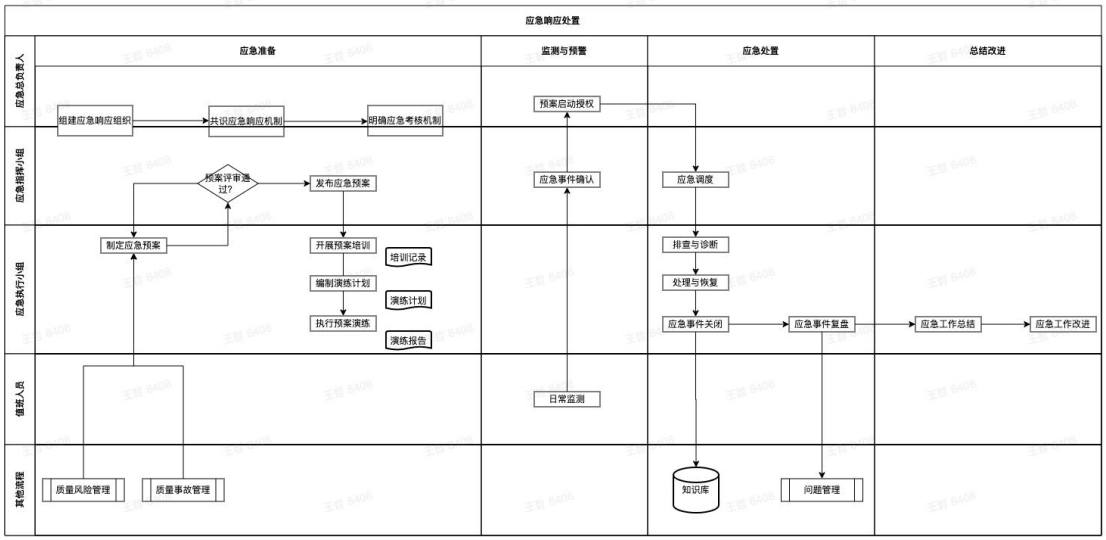
其次，影响评估也涉及决策的层级。根据故障的严重程度，我们需要决定是否需要通知高级管理层，甚至是公司高层。这一过程需要综合考虑多方面的数据，包括政府监管的要求、市场舆情的反馈、故障对经营的影响、受影响的用户数量以及影响的持续时长等。

例如，如果故障可能导致数据泄露或违反监管政策，就需要立即通知相关的法律和合规团队。如果故障影响到了大量用户，可能需要启动市场公关团队来管理用户的反馈和舆论压力。

此外，影响评估还帮助我们合理分配资源。在资源有限的情况下，了解故障的影响范围可以帮助我们优先处理最关键的问题，确保在最短的时间内恢复关键业务功能。越大的故障，处理时间可能越长，因此需要更精确的资源调配和时间管理。

应急响应组织能力

不同的层次都由不同的团队来负责运维管理，同层次不同的硬件/系统/应用都由不同的小组来负责运维管理。



在小米的应急响应流程中，监测预警和应急处置是关键环节。值班人员负责初步处理，确保问题在萌芽阶段得到控制。应急执行小组处理复杂故障，需高效沟通和定期演练。应急指挥负责整体协调和资源调配，总负责人负责对外发布信息，确保一致性。建议优化信息流，使用统一平台，引入 AIOPS 技术，提升预警和处理效率。定期演练和评估，建立详细文档和知识库，确保信息时效性和准确性。

3. 故障恢复

故障恢复的目标是迅速恢复系统功能，减少故障影响：

三板斧：重启、回滚和扩容。重启适用于单个或多个机器上的服务恢复，回滚用于撤销最近的变更，扩容用于增加系统承载能力。



升级版三板斧：限流、降级和熔断。



限流的目的是控制入口流量，减少超出后端承载能力的请求，并丢弃新请求。限流可以在不同层次上实现，例如在四层和七层网络协议上，或者在引擎层和数据库层上。通过限流，我们可以有效地防止系统过载，确保系统稳定运行。

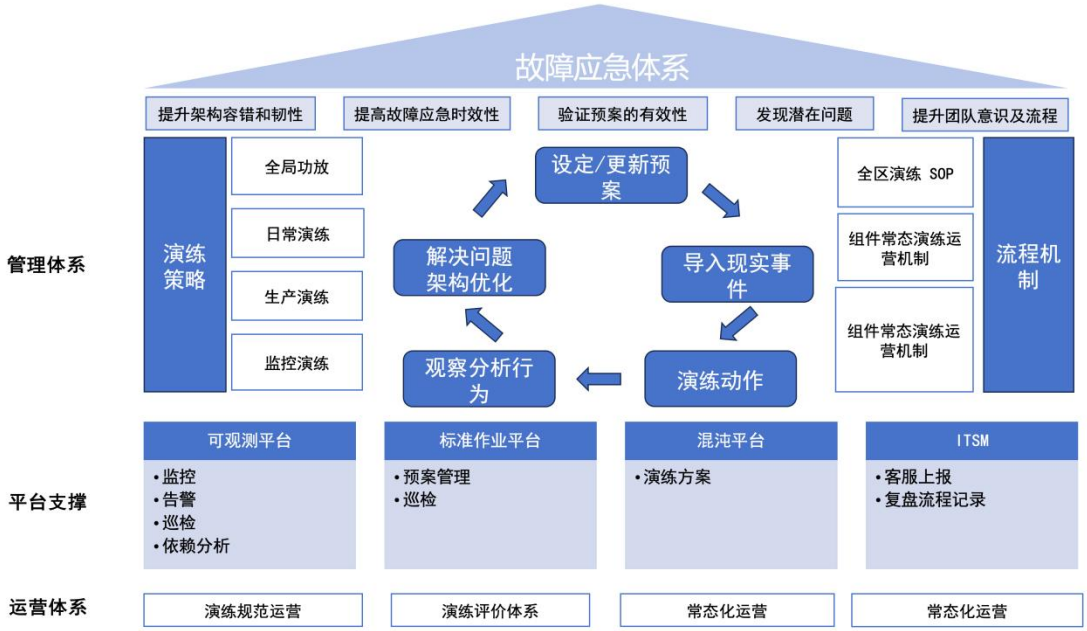
降级是指有意识地降低系统部分功能和服务质量，以确保核心功能和关键服务的持续运行。例如，在大型促销活动（如 618 和双十一）期间，京东和淘宝通常会对非核心链路进行降级，以确保核

心链路有足够的容量承载流量。实现降级通常通过配置管理来完成，需要业务 SRE 和研发团队共同配合。

熔断是一种较为危险的操作，只有在明确了解关键业务核心链路的情况下才应实施。熔断的目的是在依赖的第三方服务出现异常时，主动认为自身服务也异常，或者返回固定值，从而避免因响应时间过长导致的级联效应。熔断技术可以有效防止系统因单点故障而导致的全面崩溃。

这三项技术是我们在传统“三板斧”之外，与研发团队协作提升系统可靠性的关键措施。通过限流、降级和熔断，我们可以更好地保障系统在高负载和异常情况下的稳定运行。

故障恢复要以练养战

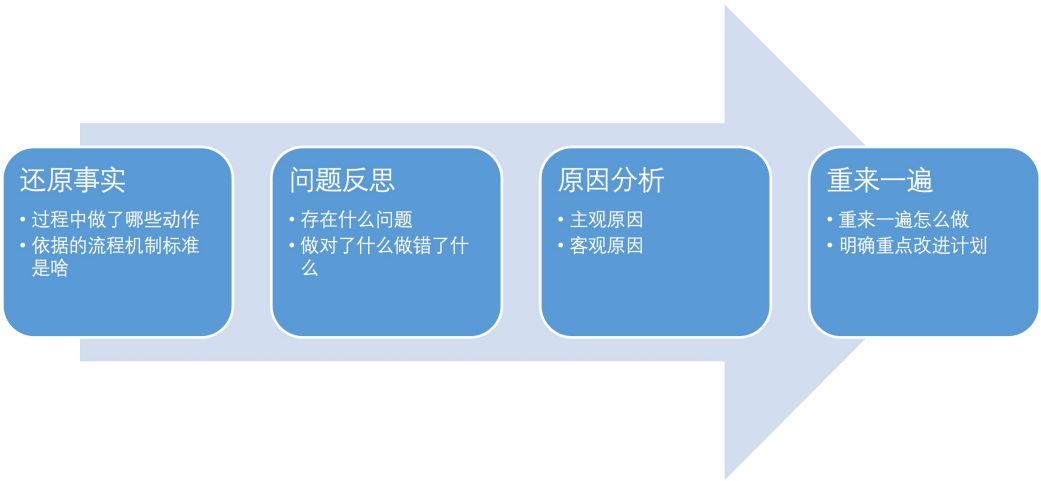


在此次会议中，我们讨论了故障管理和恢复的核心，即制定和执行预案以确保系统稳定性。预案有保鲜期，需定期更新和验证其有效性，通过故障演练实现。演练大盘涵盖演练策略、流程机制、管理体系、工具体系和运营体系。具体包括日常演练、生产演练、监控演练、全局演练 SOP、组件常态演练、重点业务演练、监控报警、故障恢复和复盘改进。通过这些措施，提升系统在故障情况下的应对和恢复能力，确保业务的连续性和稳定运行。

声明：上图为作者从网上引用材料，如有版权问题，请联系 SRE 精英联盟。

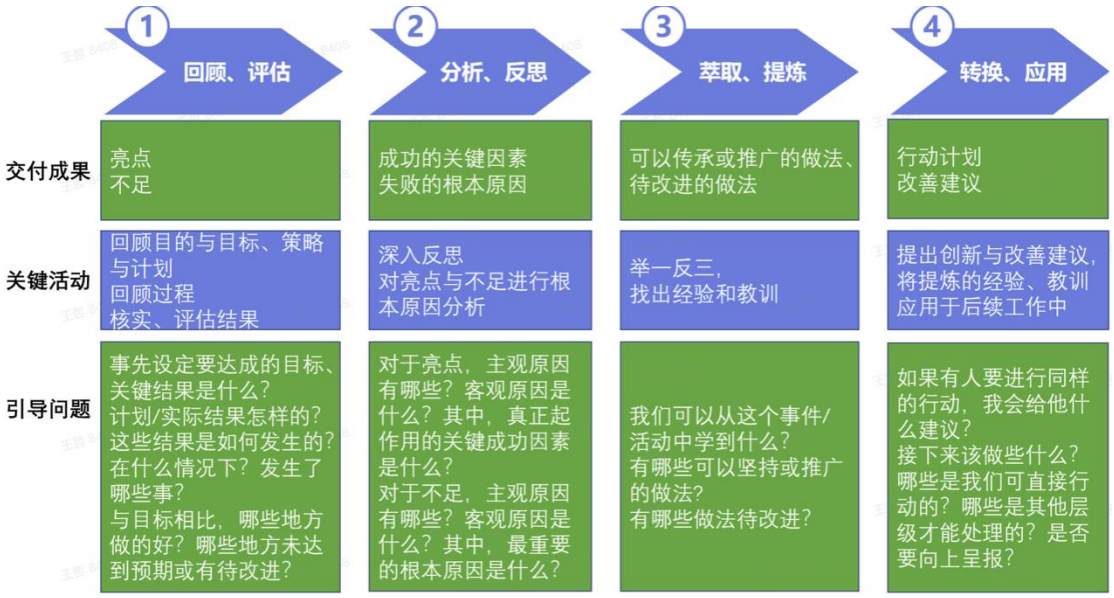
4. 故障复盘

故障复盘是防止同类问题再次发生的关键：



- 还原事实：记录故障处理过程中所有核心动作和依据的流程机制标准。
- 问题反思：反思故障中存在的问题和解决措施。

- 原因分析：分析主观和客观原因，强调管理动因。
- 重来一遍：思考如果重来一次，哪些环节可以加速或避免。



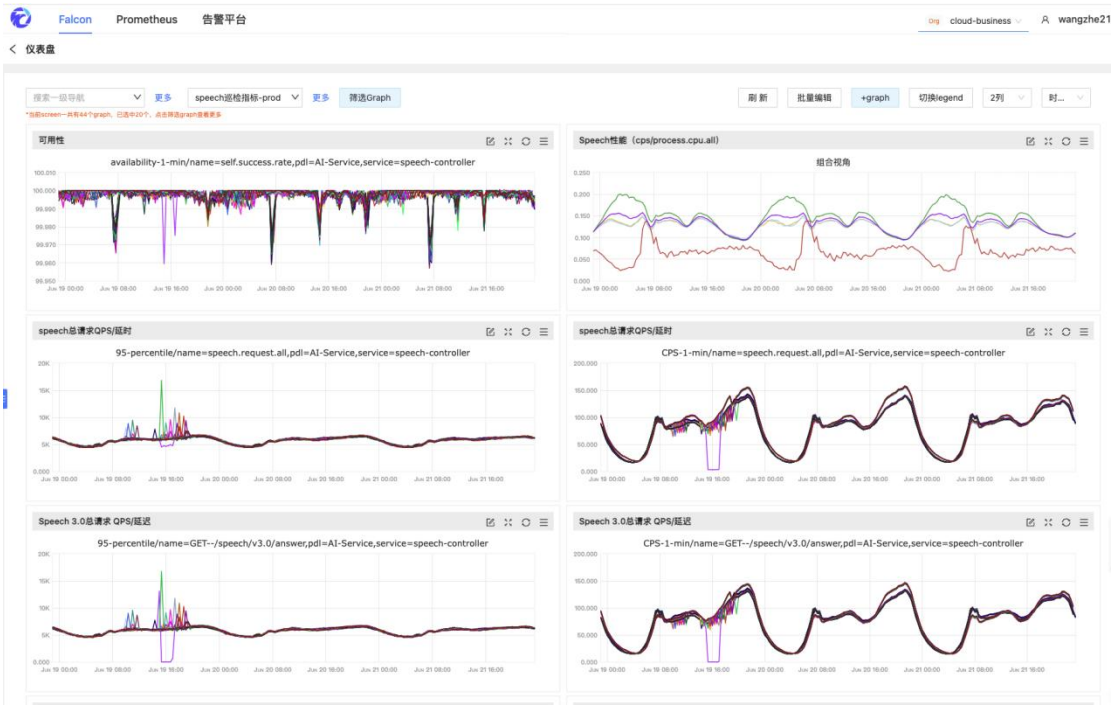
故障复盘是确保系统稳定性的重要环节。许多公司在故障复盘方面做得不足，但在小米，故障复盘被视为核心工作，尤其是在智能硬件制造企业中，质量问题的召回成本和舆情影响巨大。小米将故障复盘作为质量体系的重要组成部分，并在互联网及基建组织中推广。

在复盘过程中，强调每个步骤的执行和记录，确保每一个动作都有明确的结果和影响。通过格式化复盘问题反思、原因分析以及假设重来一遍，确保问题不再重现。使用关键引导问题（如 5 个为什么）深入骨髓地进行复盘，确保每个故障都能得到全面分析和改进。

通过以上措施和体系建设，我们能有效提升系统在故障情况下的应对和恢复能力，确保业务的连续性和稳定运行。

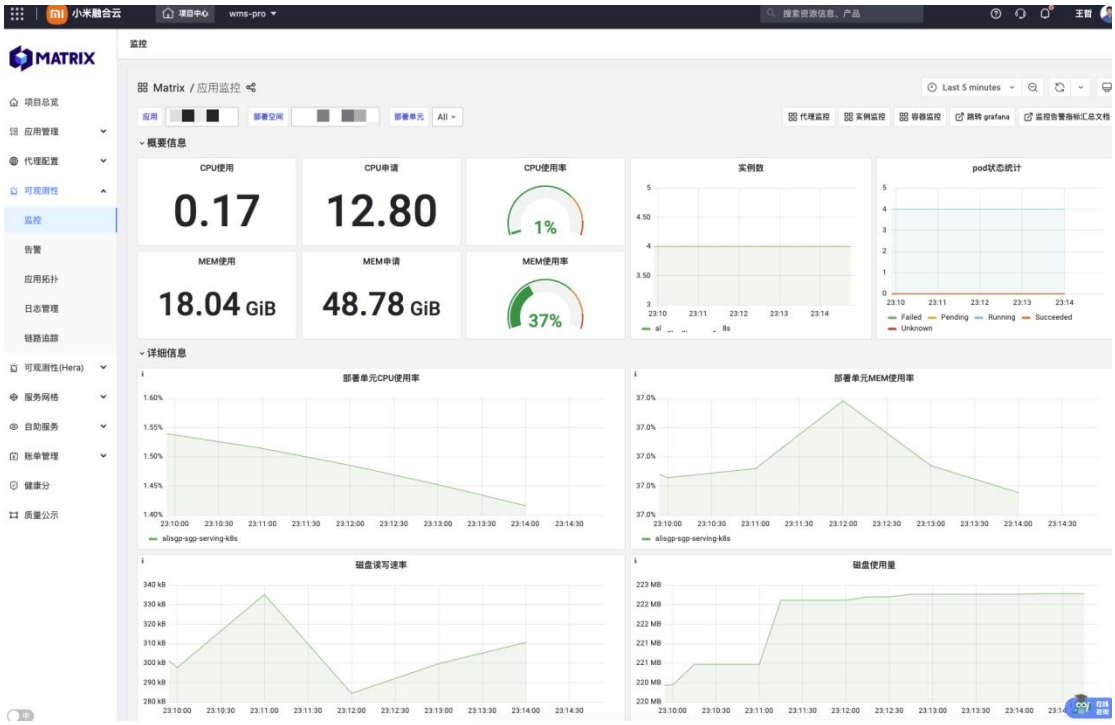
5.工具和平台

监控



众所周知，著名的开源软件就是在小米进行孵化的，这就是 Open-Falcon 进化到 2024 年的样子，目前在小米监控体系中主要定位用于基础监控。

另外我们的系统中还整合了 Prometheus，利用了 Prometheus 的生态支持众多监控指标，进行容器监控， 业务监控等。



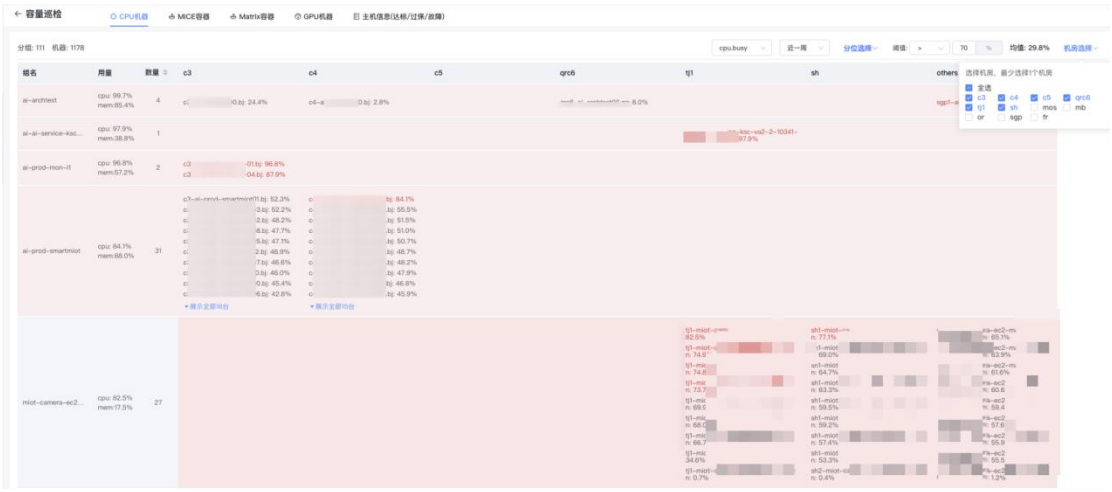
我们在小米内部实现了一个一站式的平台工程，包括容器管理，应用管理，服务治理等能力，并将部分可观测能力进嵌入。

告警



在告警方面，我们使用飞书作为基础平台，并基于其能力开发了自有的告警功能。我们的告警系统支持图表查看、关联分析和告警路径编辑。当告警触发后，工程师可以认领告警，其他人无需重复处理。此外，告警还可以被屏蔽，以避免重复通知。

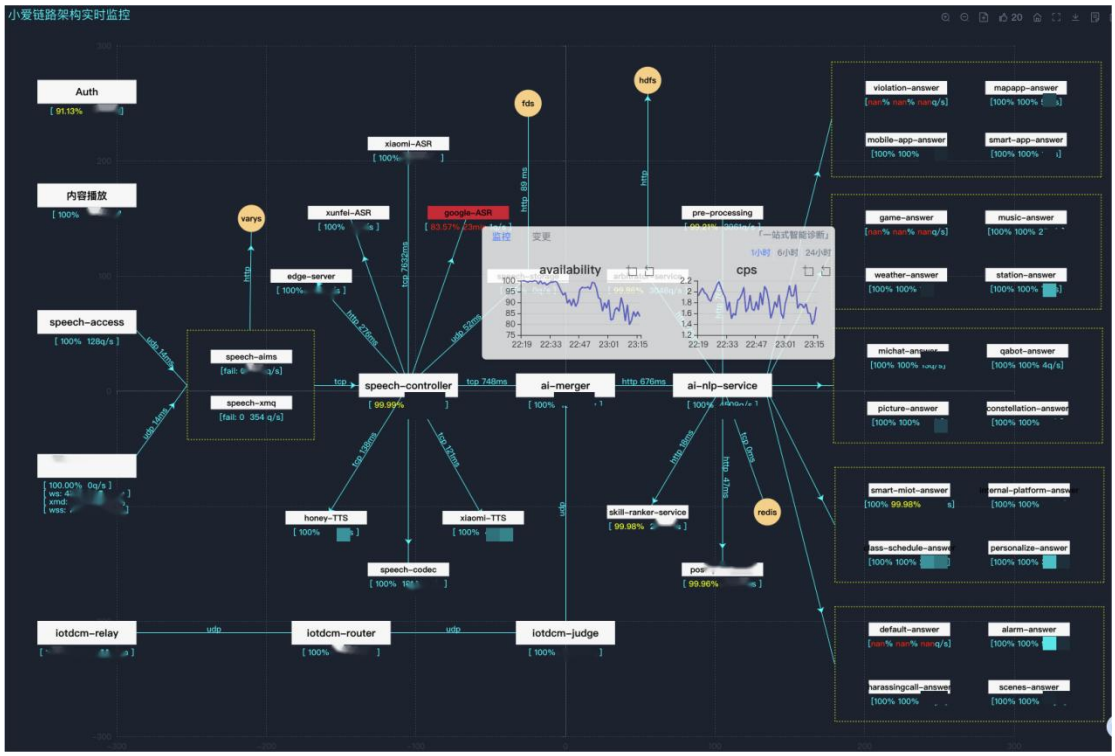
巡检



巡检是我们日常工作的重要组成部分。目前，我们内部有多个巡检平台。昨晚我截屏时发现其中一个平台无法使用。这种情况在 SRE 工作中时有发生。虽然我不打算详细展开，但可以肯定的是，做好巡检是避免故障的最有效手段。

建议大家充分利用监控工具，并合理编排各种巡检任务。只要管理好巡检和变更，你们公司的系统就能保持稳定，减少故障的发生。

链路架构

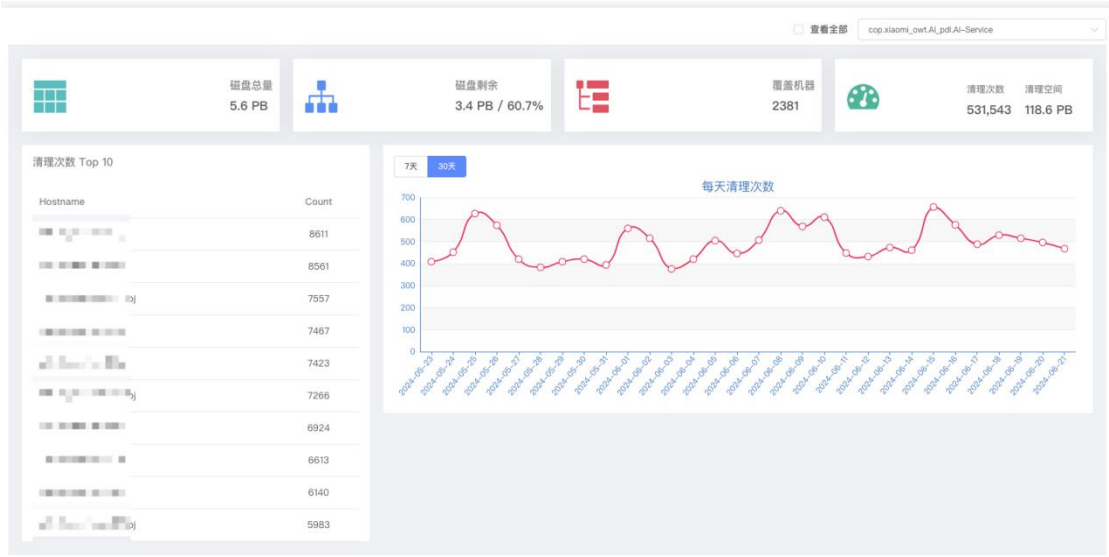


目前的链路监控系统在设计和实现上存在一些理想化的成分。在 SRE 精英联盟的讨论中，部分同事建议使用 Skywalking 进行抓包，并自动生成业务逻辑图。然而，除非公司拥有非常强大的语言框架并统一应用，否则只通过配置管理实现抓包并生成业务架构图会较为困难的。

对于那些允许使用多种框架和语言的开放型公司而言，包括小米，从研发侧生成架构链路并不现实。更为可行的做法是由 SRE 团队基于业务逻辑架构绘制核心链路图，并将其与监控系统进行对接，提供支持和全局监控能力。这是目前大多数公司在链路监控方面较为理想的解决方案。

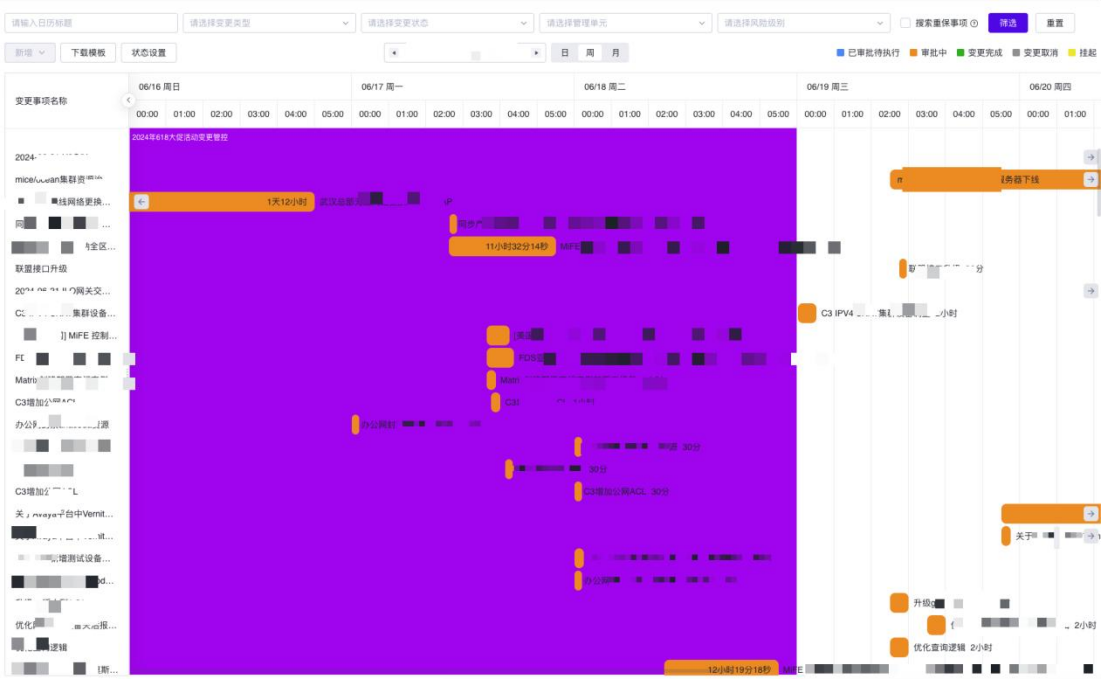
接下来，我们需要解决的问题是如何在功能发布后，确保链路监控与现有系统保持一致，形成一个闭环。实现这一点后，链路监控系统将能够更好地满足公司日常监控和异常处理的需求。

磁盘自愈



以上截图展示的是是小米一个磁盘自愈的小工具。

变更日历



我们的变更管理体系非常严格，质量组织对变更的要求极其严苛。例如，周一和周五不允许进行某些级别的变更，特定时间点也禁止变更。此外，我们还对双十一、618、4 月米粉节等重要节点进行严格管控。

从图中可以看到，我们在特定时间段实施了变更管控，所有变更需要总监级以上审批。我们还根据时间维度记录了所有变更，便于在异常发生时快速查找根因。SRE 团队通常会查看这些时间节点，确认公司在特定时段内进行了哪些变更，以决定是否需要回滚并进一步查找问题根源。

故障上报

我们建立了一个故障上报系统，要求在故障发生并初步判定影响后，立即通过该系统上报故障。系统具备一键生成故障复盘文档的功能，按照标准化的复盘格式记录。

在故障处理完成后，例如云平台要求在故障处理后一天内完成内部复盘。接着，我们会与业务方进行第二轮复盘。根据故障的严重程度，可能还需要质量委员会进行进一步复盘。最后，完整的故障报告将提交给集团相关的质量组织。



故障处理

为提高故障处理效率，我们统一了故障上报入口，并建立了大部及公司级故障应急响应中心。小米拥有数万名工程师，我们设立了一个成员超过 5000 人的故障服务通报群。当遇到无法判断的故障情况时，可以在群里发布信息，确保所有相关人员能够及时看到。



质量管理

在小米，质量管理是独立于基础设施的，有独立的组织作为监管方，监督技术、运营和产品。质量管理团队有自己的目标和执行计划，尽管他们的工作可能对日常工作效率和人员舒适度带来挑战，但他们遵循行业标准，如信通院的标准，确保安全、隐私、合规和质量要求。

我们首先满足质量管理团队的诉求，再在这个框架下寻找妥协和执行的空间。这种独立的质量管理视角，尽管带来挑战，却是确保系统安全、合规和高质量运行的关键。



（三）总结及展望

故障应急是对 SRE 体系综合能力的考验，涵盖信息采集、快速反应和决策、稳定准确的处置、资源整合能力及日常演练形成的流程体系。

关键能力

信息采集：及时、准确地获取系统状态和故障信息。

快速反应和决策：迅速分析并做出决策，确保问题及时解决。

稳定处置措施：执行稳定、准确的应急措施，确保系统尽快恢复。

资源整合：整合内外部资源，迅速调动所需资源处理故障。

日常演练：通过日常演练，形成规范的流程体系，提高团队应急效率。

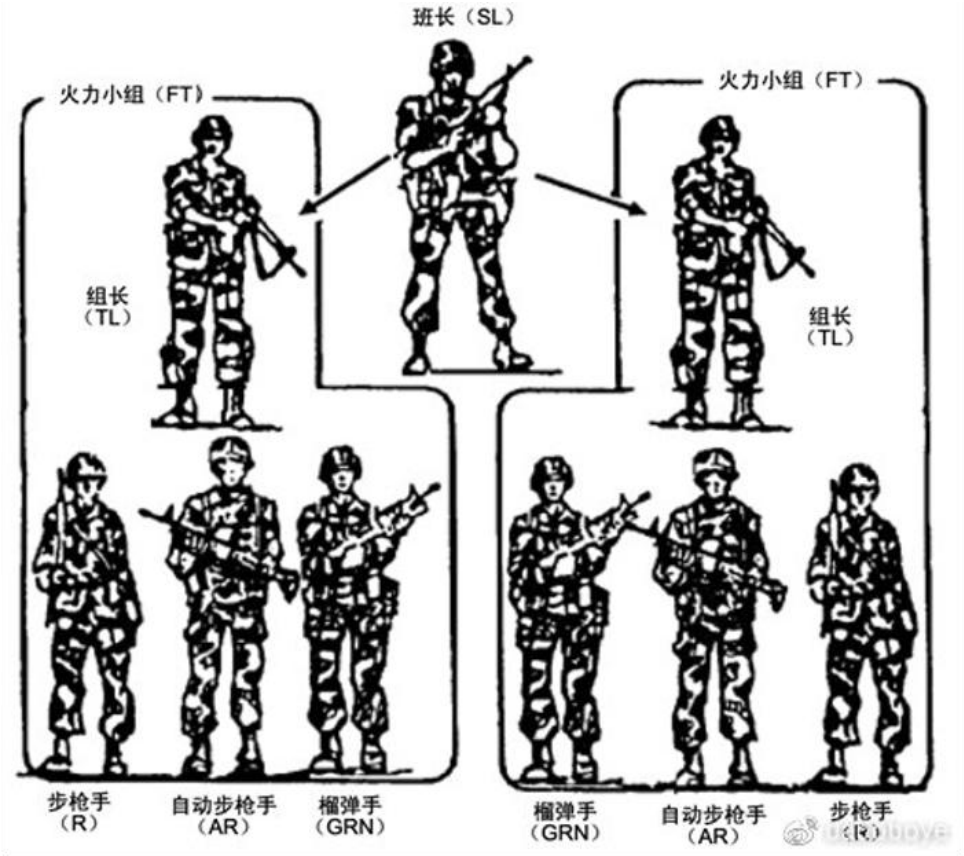
团队的重要性

经验丰富、能力强大的专家团队是故障应急的关键。每位成员在故障应急中扮演重要角色，确保系统稳定运行。



2004年在伊拉克的一个美国海军陆战队步兵班，里面有3挺M249。照片中有12人，第13人是拿相机的

类比 SRE 团队之于海军陆战队的步兵班：



分工明确：团队中有专门负责信息采集和执行操作的成员，形成三角形决策团队。

资源整合：如步兵班呼叫精准打击，SRE 团队迅速调用资源解决故障。

日常训练：通过日常演练提升应急能力，每位成员都是关键。

通过这些措施和体系建设，提升系统在故障情况下的应对和恢复能力，确保业务连续性和稳定运行。

5.3.2 中国联通数字化监控平台稳定性保障实践

SRE Elite 精选原因

中国联通作为国家重点央企，长期以来以其庞大的业务体系和稳健的运营著称，面对数字化转型的浪潮，中国联通积极推动核心业务系统向云原生架构大规模演进，面临着技术革新的复杂挑战，还需确保转型过程中的系统稳定性。此案例探索并构建了一套符合稳态企业的稳定性保障方案、策略及平台，并深度融合可落地的智能化 AIOPS 能力，提升运维效率与故障处置速度，为其他稳态企业提供了可供参考的路径。

（一）背景及设计原则

中国联通是国内知名的四大运营商之一（原三大运营商现加入广电成为四大运营商），中国联合网络通信有限公司软件研究院（简称“中国联通软件研究院”）是中国联通的子公司，成立于 2015 年，是中国联通集团总部提升自主研发能力的重要战略规划，主要承担中国联通集团的内部业务支撑系统、管理支撑系统、大数据以及前瞻性技术研究与应用，负责 B 域、M 域、D 域系统的建设和运营，

这里稍微解释一下，各个域名是什么意思。

B 域（业务域）= business support system，

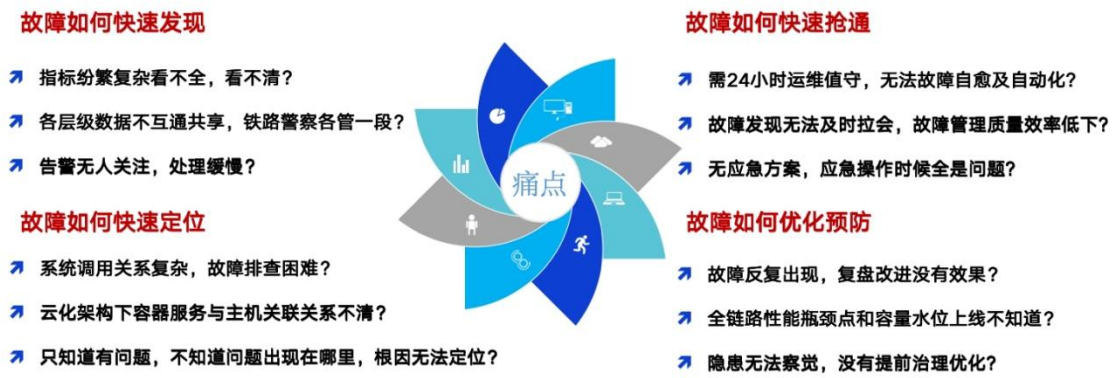
M 域（管理域）= management support system。

D 域（数据域）= Data support system

其中，B 域系统主要包括营业厅办理业务的系统。例如，用户去营业厅办理开卡业务，使用的系统就是我们的业务系统。此外，用户每月交话费和出账单的系统也是由我们公司开发和运营的。

本案例提供者主要负责数字化生产运营保障体系的建设，以及天眼数字化监控平台的架构设计和演进。中国联通作为四大运营商之一，拥有 4.3 亿用户，每天有千万级的调用量，而且联通的系统是一个集约化的系统，31 个省份都在同一个系统上运行，作为国计民生的基础设施，对系统的稳定性和故障处理提出了极高的要求。

运维人员的问题痛点



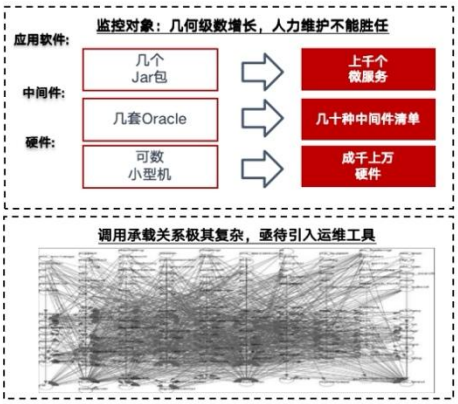
例如在故障过程中，如何快速发现问题。我们面临的一个挑战是指标众多，但难以全面掌握。不同层级——基础设施层、云平台层和应用层——之间的数据不共享，如何实现它们的统一共享？是否存在各自为政的情况？即使告警已经发出，但无人处理的情况又该如何应对？在微服务和容器化的环境中，服务之间的调用非常复杂，我们是否有快速排查故障的手段？服务和主机之间的关系是否

明确？当问题出现时，我们能否迅速定位？

我们是否有 24 小时值班的运维人员？是否具备自动化快速自愈和止损的能力？在故障过程中，能否迅速调度故障专家进行处理？应急预案是否有效？故障复盘后，除了整改措施，是否有进一步的优化？我们需要挖掘和治理哪些风险隐患？以上均是我们的运维人员面临的挑战。

云原生下运维的问题挑战

随着云原生技术的不断成熟，企业数字化转型也在不断加速，企业 IT 架构进入云原生时代，多云多集群部署已经成为常态和趋势，几何增长的云资源、微服务以及复杂化的调用关系与业务场景，传统人肉运维难以为继，如何保障系统的全面稳定，保证业务流程的高效运转，为系统运营提出了不小的挑战。





接下来，我将举一些典型案例，说明白不同故障在不同层面的办理一些案例截图



故障根因在 SaaS 服务下的实例，具体某个主机上的服务失败调用增加。



故障根因在 PaaS 组件，有时候，服务本身没有问题，但是依赖的下游组件出现了故障。例如，依赖的 Redis 突然出现了雪崩，或者调用的 ES 突然查询变慢。这些问题又该如何处理？



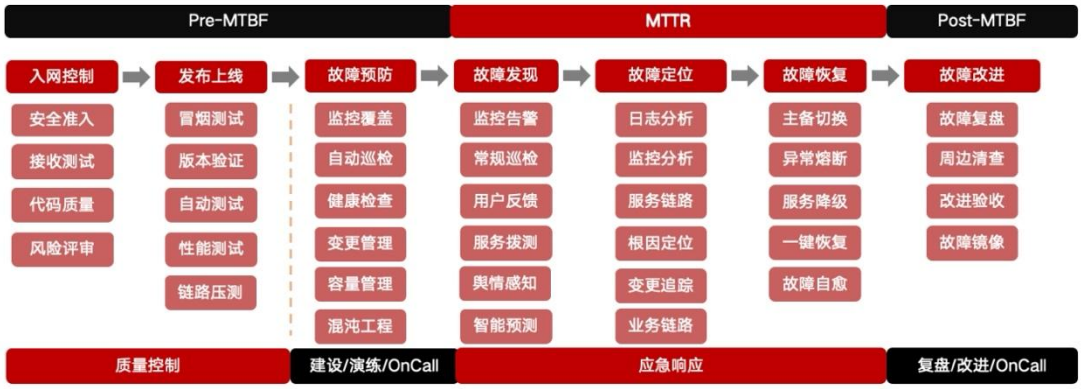
故障根因在外部接口案例：外部接口故障：除了自身系统内部的问题，还可能会遇到外部因素的干扰。例如，依赖的第三方接口出现了问题。这样的问题又该如何快速发现，快速感知，快速排查呢？

（二）解决方案详情及案例

数字化监控平台的核心能力三个解决方案

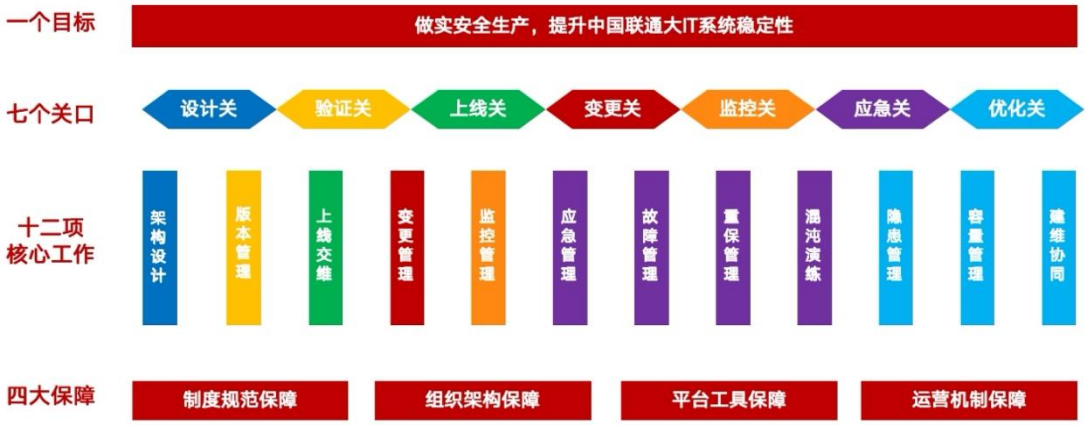
稳定性保障解决方案

解决方案，将安全生产稳定性保障左移，在入网控制时介入，对入网控制、发布上线、故障预防、故障发现、故障定位，故障恢复、故障改进提供端到端工具支撑。



生产运营保障体系结构框架

生产安全保障体系：一个目标，依托四大保障，聚焦研运流程中十二项核心工作，严格把控七个关口。



设计关关键点：架构设计阶段，确保包含冗余设计、灾备设计、熔断限流等稳定性设计。

验证关关键点：上线前进行严格的版本管理和验证，确保系统符合稳定性和安全性要求。

上线关键点：确保研发与运维之间的无缝衔接，规范上线流程，确保系统平稳上线。

变更关键点：变更前进行流程审批，变更中严格执行规范，变更后进行全面检查，确保变更过程安全可控。

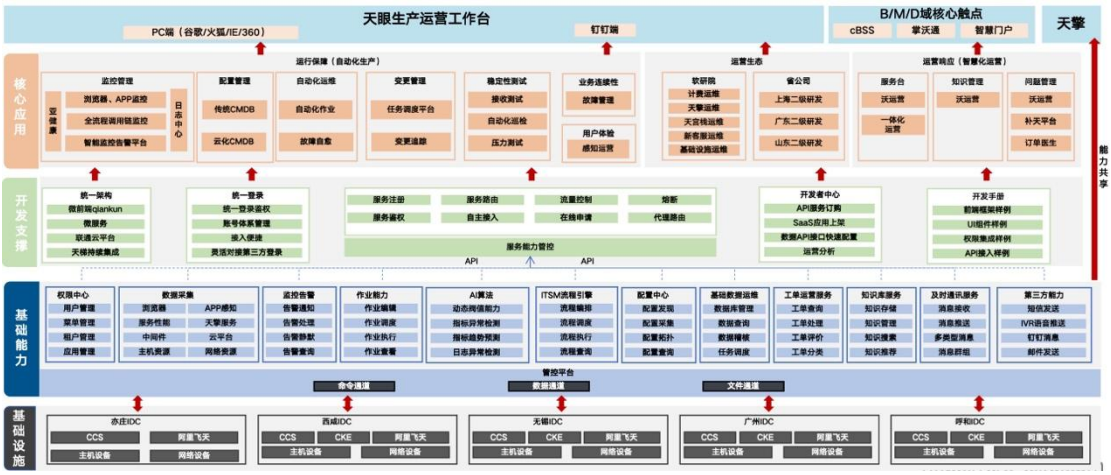
监控关键点：建立核心的可观测性能力，实时监控系统运行状态，及时发现和处理异常。

应急关键点：制定并演练应急预案，涵盖故障上报、调度和复盘，确保在故障发生时能够快速响应和恢复。

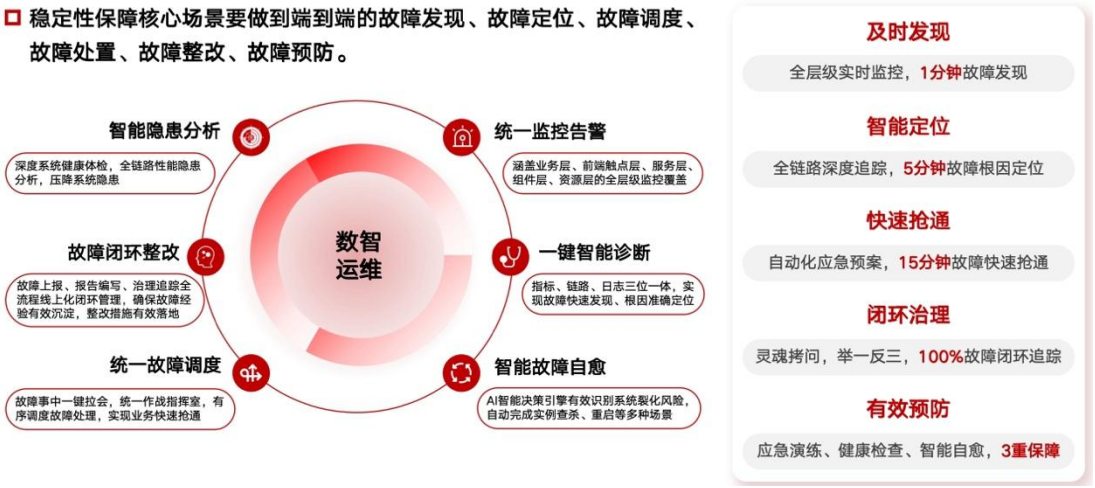
优化关键点：持续优化系统，定期进行隐患排查和治理，评估和管理系统容量，确保系统稳定性和高效性。

数字化监控平台架构

基于云原生下的生产运营支撑平台，以全局运营视角解读 IT 运维，提供端到端、全层级的运维工具支撑，依托大数据与人工智能技术，助力企业数字化业务高效、稳定运行，从传统运维向自动化生产、智慧化运营转变。



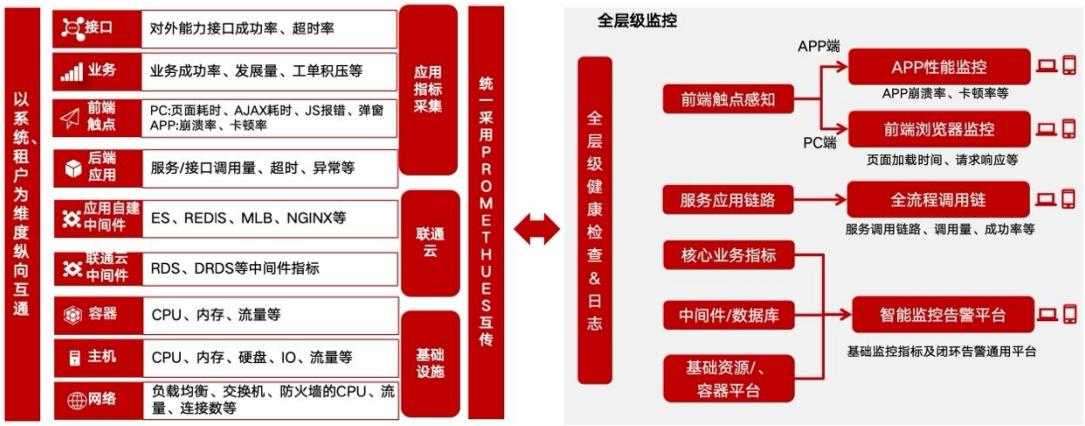
稳定性保障的场景应用实践



1-5-10 目标是由阿里提出的，联通则设定了“1-5-15”目标，即 1 分钟内发现故障，5 分钟内判断故障根因，15 分钟内快速恢复。虽然尚未完全实现 15 分钟内恢复，但通过多年的努力，目前故障 15 分钟内恢复的比例已达到 65%。

为了实现这些目标，我们构建了六个核心场景，来指导我们的工作。

1.统一监控告警

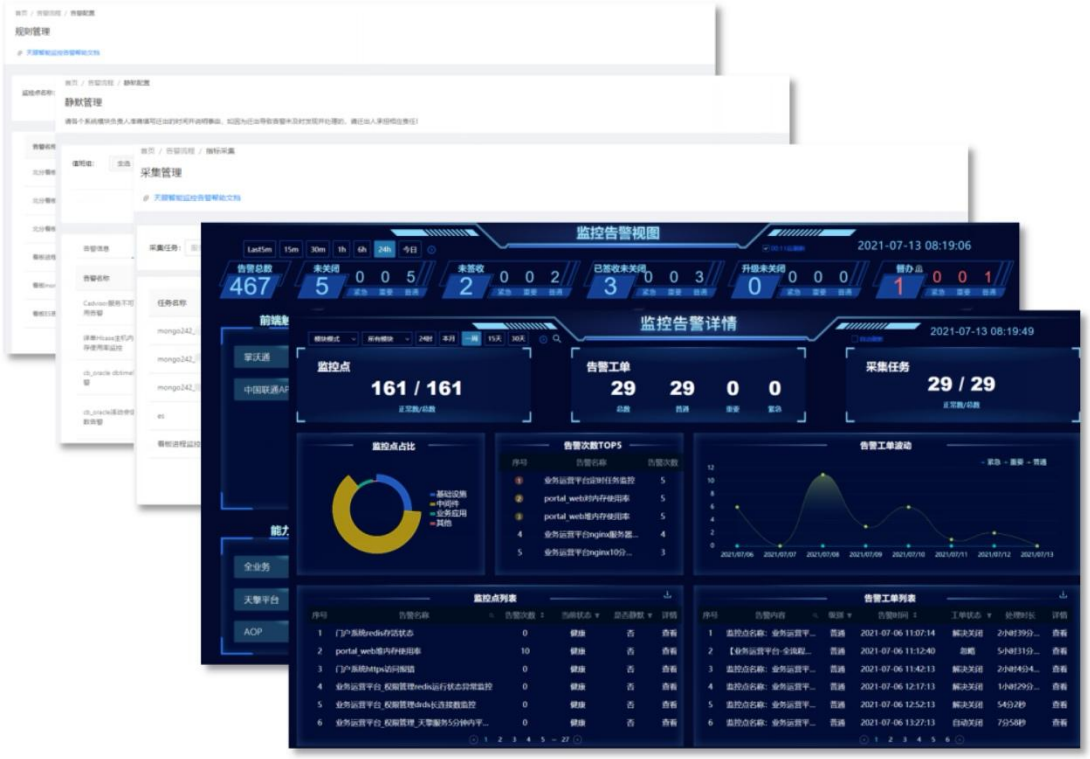


要进行统一监控,我们统一采用 Prometheus 协议进行监控数据的采集，并按照系统和租户的维度进行监控，以实现不同层级指标的统一管理，所有这些监控数据最终通过 CMDB 进行关联和汇聚，确保监控的全面性和准确性。

图示中展示了全层级监控标准，打破分散割裂格局，实现全层级、全链路、端到端的性能监控和链路追踪。

智能监控告警平台

平台提供 IaaS、PaaS、SaaS 各层级监控能力，实现多层级运维数据互通，支持全流程可视化配置，多渠道告警通知，工单闭环管理，用户快速实现监控接入，为系统日常生产运行提供保障。



数据采集：采集组件管理、私有数据仓库接入、租户自定义采集

- 监控配置：告警规则、收敛条件、告警内容
- 静默管理：多维静默管理（全量、监控点、监控实例）
- 告警通知：告警工单推送、电话催办
- 告警处理：双终端工单处理、工单闭环管理
- 告警大屏：系统监控告警全景图、告警工单处理进度

我们的告警平台具备全面的统一纳管能力，涵盖基础设施层、平台层、组件层、服务层、触点层和业务层。通过统一的数据拉通和监控告警，我们实现了全面的系统监控。

联通具有天然的优势，例如短信和外呼能力。这些能力已集成到我们的短信中心、通知中心和外呼中心中，能够快速通知相关值班人员。与互联网公司不同，我们更强调责任问题，确保故障通知到具体值班人员，而非简单地发到群里无人认领。

具体流程如下：

- 故障通知：故障发生时，系统会自动通知相关值班人员。
- 故障处理：如果值班人员在 10 分钟内未签收处理，系统会自动升级通知给其负责人。
- 逐级升级：若负责人仍未处理，系统会继续升级通知至部门经理，直至故障被处理。
- 7x24 小时值班：我们有专门的调度值班组，确保外呼能及时联系到值班人员。如普通外呼未能联系成功，将换值班人员进行外呼，确保告警及时签收处理。

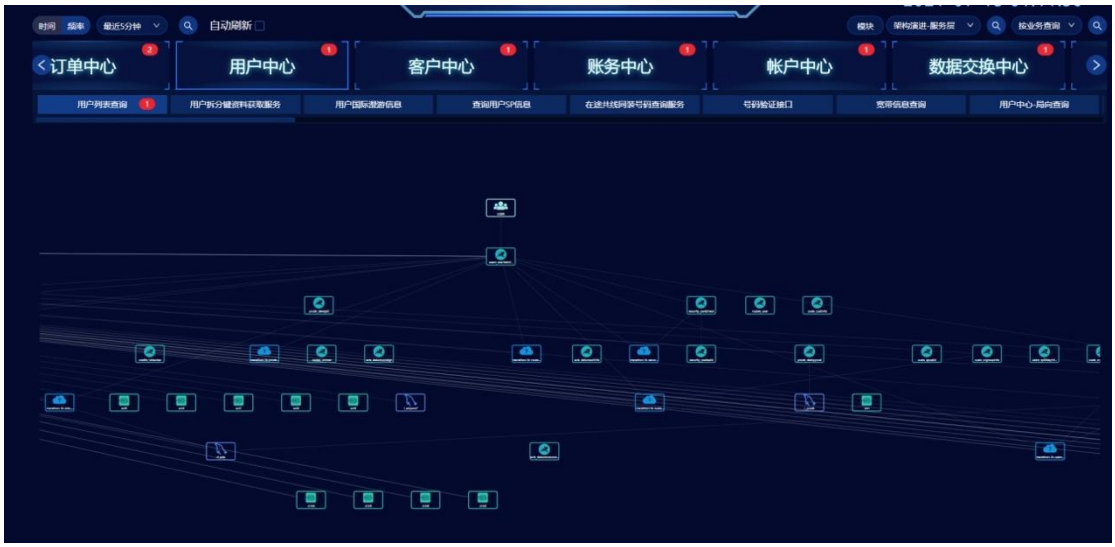
此外，我们有一套运营机制，通过这套机制压降签收处理时长，确保告警得到及时处理，保障故障 1-5-15 目标的实现。



针对这些全层级的监控指标，需要了解每个指标的定义和标准。为此，中国联通软院制定了 414 项的全层级指标规范，包括每个指标的推荐配置、必须配置的指标、每个指标的配置阈值等。

1.全流程调用链监控

我中国联通的集中系统包含 5000 多个服务，链路拓扑复杂，需依赖技术手段进行梳理，我们通过全流程调用量监控(即 APM)实现对系统的全面监控。最初采用 PinPoint，去年开始升级为基于 OpenTelemetry 的系统，兼容两种形式，采集黄金指标并绘制链路拓扑。通过相关数据，在出现问题时，我们能迅速定位问题指标，如调用量下降、失败率上升或接口超时，并及时发出告警。通过 CMDB，我们可以定位到具体实例，快速重启实例以恢复故障，并查看应用所在的容器和主机的详细指标。



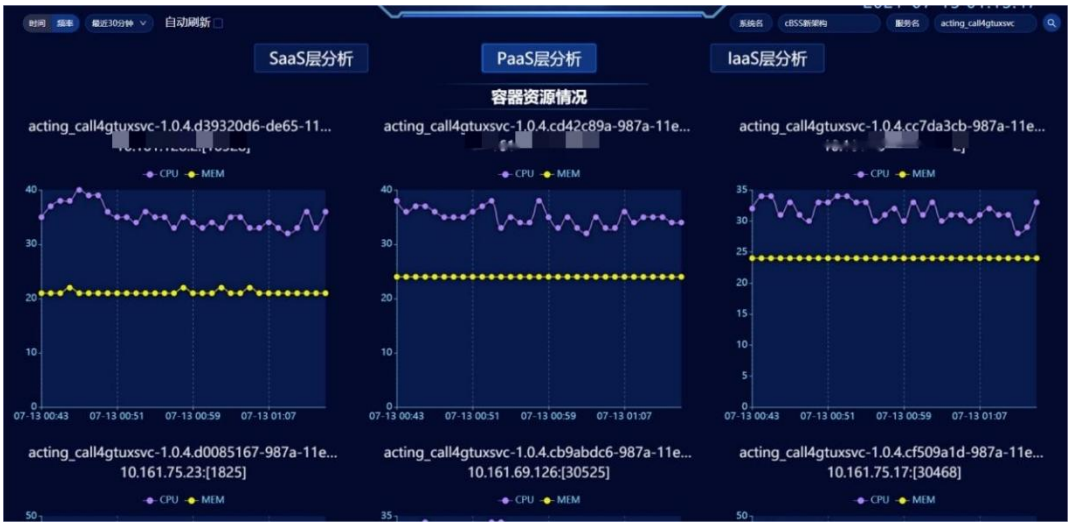
调用拓扑 全流程调用链拓扑自动生成，分租户管理



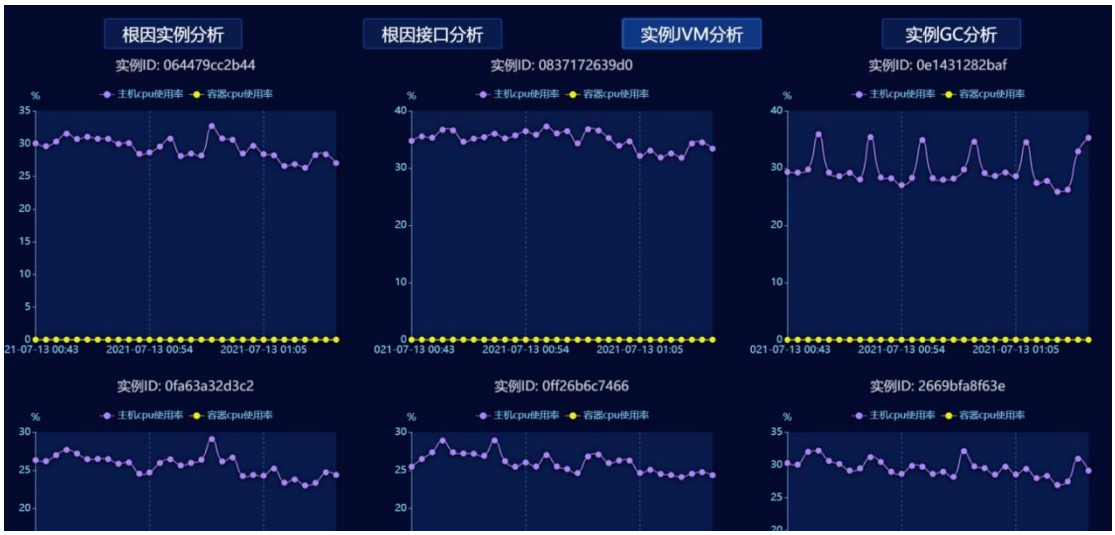
服务趋势/报错异常：服务调用关系、趋势图、报错分类（系统/业务）



实例/接口分析：调用链与云化 CMDB 做关联，关联到容器与主机



SaaS/PaaS/IaaS PaaS 层组件、平台容器资源情况，IaaS 层主机资源



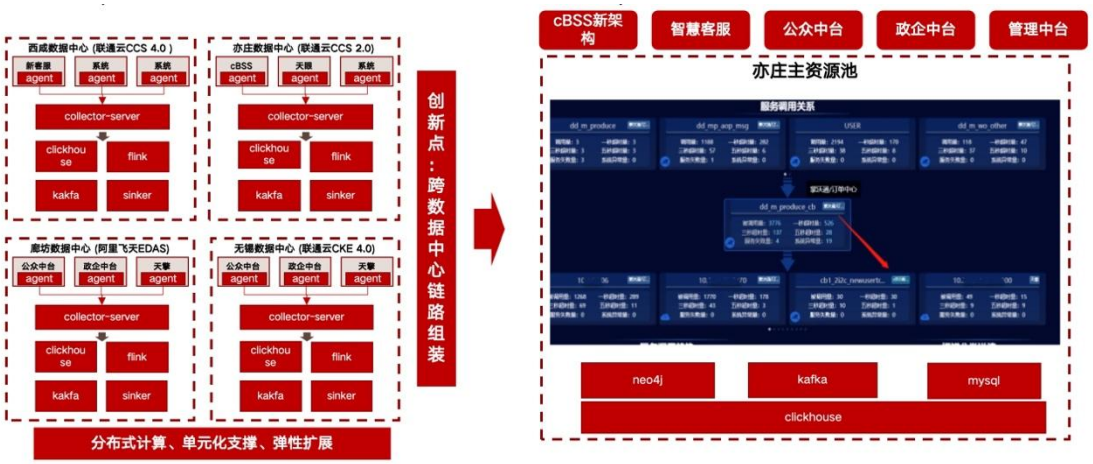
JVM/GC 分析 服务实例 JVM 与 GC 情况分析



告警配置：调用量、超时、异常黄金指标多指标自由组合

跨系统分布式追踪

通常的链路跟踪系统串联范围通常只局限在某一云平台之上，但我们通过分数据中心汇总串联，实现了分数据中心汇总串联，支持跨系统、跨云平台（CKE/CCS/EDAS）、跨数据中心（亦庄、西咸、廊坊、无锡）链路拓扑，完成跨系统调用实时追踪和方法清单级根因定位，日均处理近千亿数据。



前端触点监控

采用 JS 埋点的方式，采集用户访问过程的性能指标，获取浏览器端的真实用户行为与体验数据。包括页面加载、点击、弹窗、JS 报错、ajax 等用户全轨迹跟踪，通过大数据分析，应用于院内故障定位、安全分析、终端分析、感知分析、异常分析等场景。


系统总览


页面性能分析


弹窗分析


AJAX分析

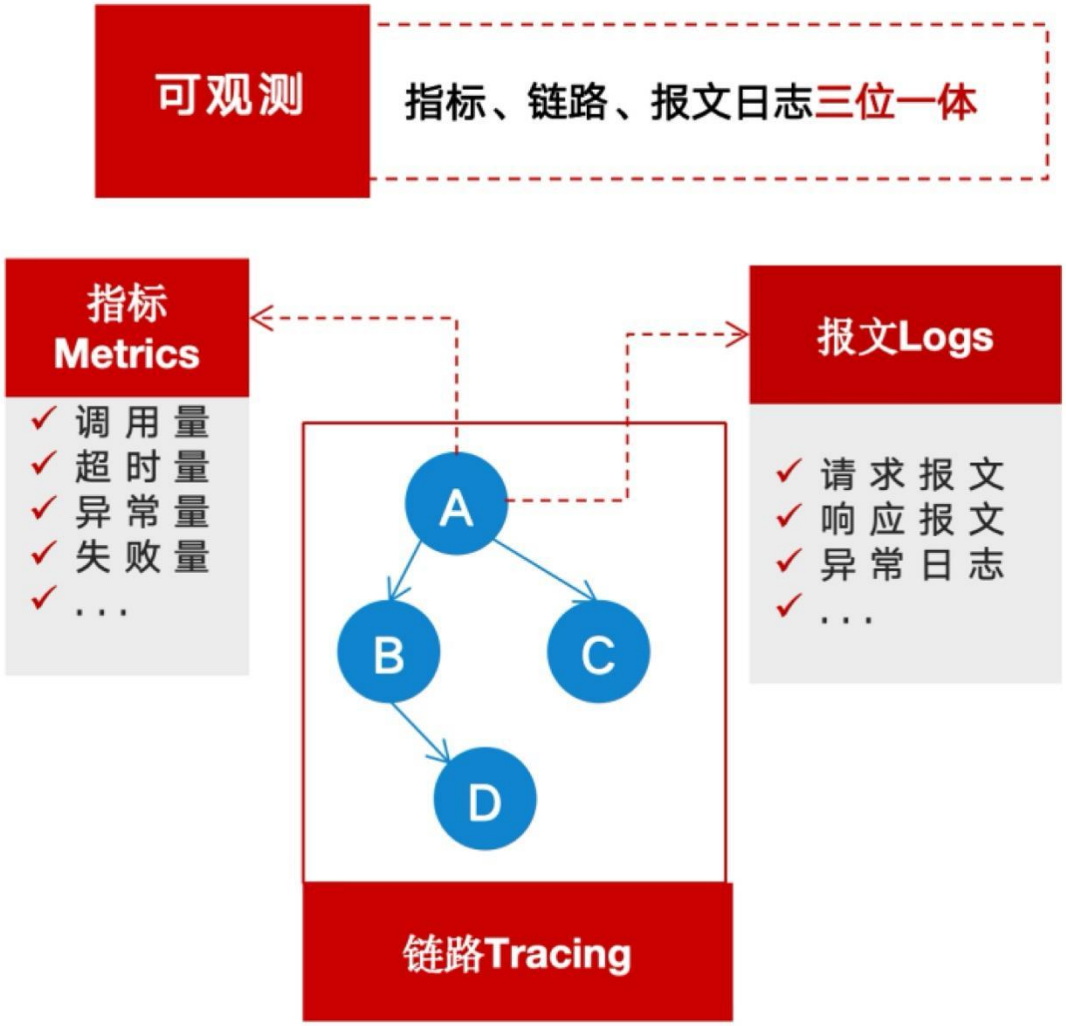

用户轨迹分析

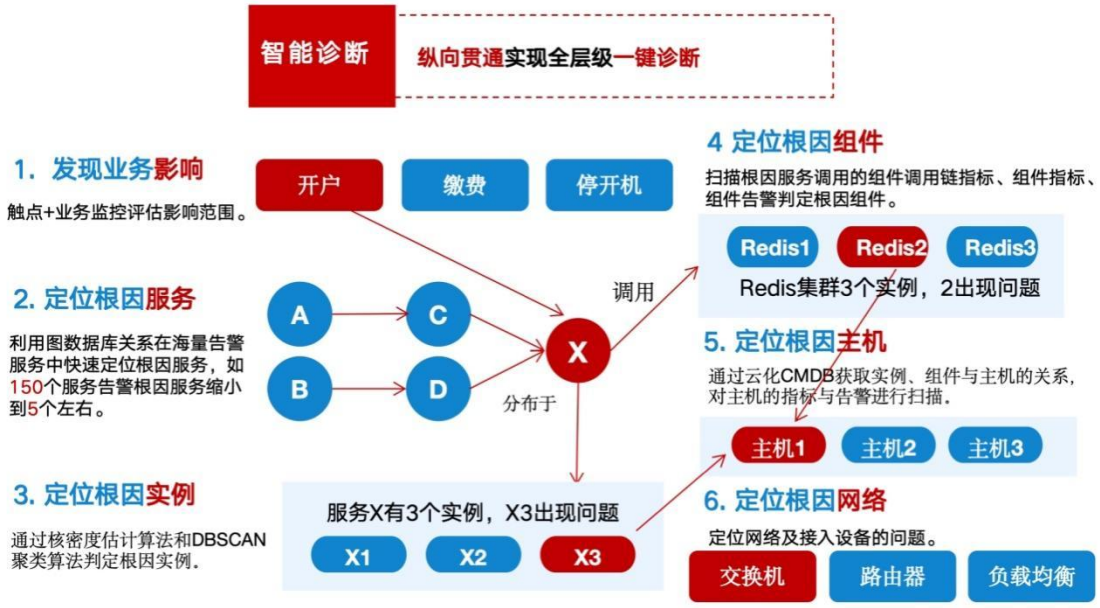

工号稽核

2.一键智能诊断

通过采集指标、链路、报文日志，实现三位一体的可观测性，在系统纵向全层级方面实现触点层、服务层、组件层、平台层、主

机层、网络层纵向贯通，结合云化 CMDB 关联定位，实现全层级一键诊断，端到端快速定位问题根因。





故障诊断案例

依托全层级监控指标数据、全层级链路调用、云原生 CMDB，建立故障传递模型，以服务层为故障起点进行纵向串联，配以规则+AI 的能力实现全层级一键智能故障诊断。目前，中国联通一键智能诊断准确率大概在 70%左右。





服务实例异常：根因服务实例耗时突增实例 GC 引发故障



Oracle 异常：Oracle 会话数突增导致服务连接超时增多



ES 异常：ES 进程负载率突增导致上游服务连接超时



RDS 异常：RDS 慢 SQL 突增导致节点状态异常



Redis 异常：Redis 耗时波动引起上游服务连接超时



快立方异常:根因服务下游调用快立方告警异常



主机宕机异常：LB 所在主机宕机导致 lb 实例销毁重启服务波动



网络异常：网络带宽使用率指标打满引起访问受限

3.智能故障自愈

将“监”、“管”、“控”工具能力融合，告警信息结合 AI 判定算法，触发自动化作业能力，实现故障自愈流程，有效缩短故障处理、恢复时间。



以下是案例：

16:20:45 应用告警



16:20:45 自动触发诊断





16:20:47 推送实例查杀、重启工单



16:21:29 运维人员确认操作

16:21:34 应用恢复

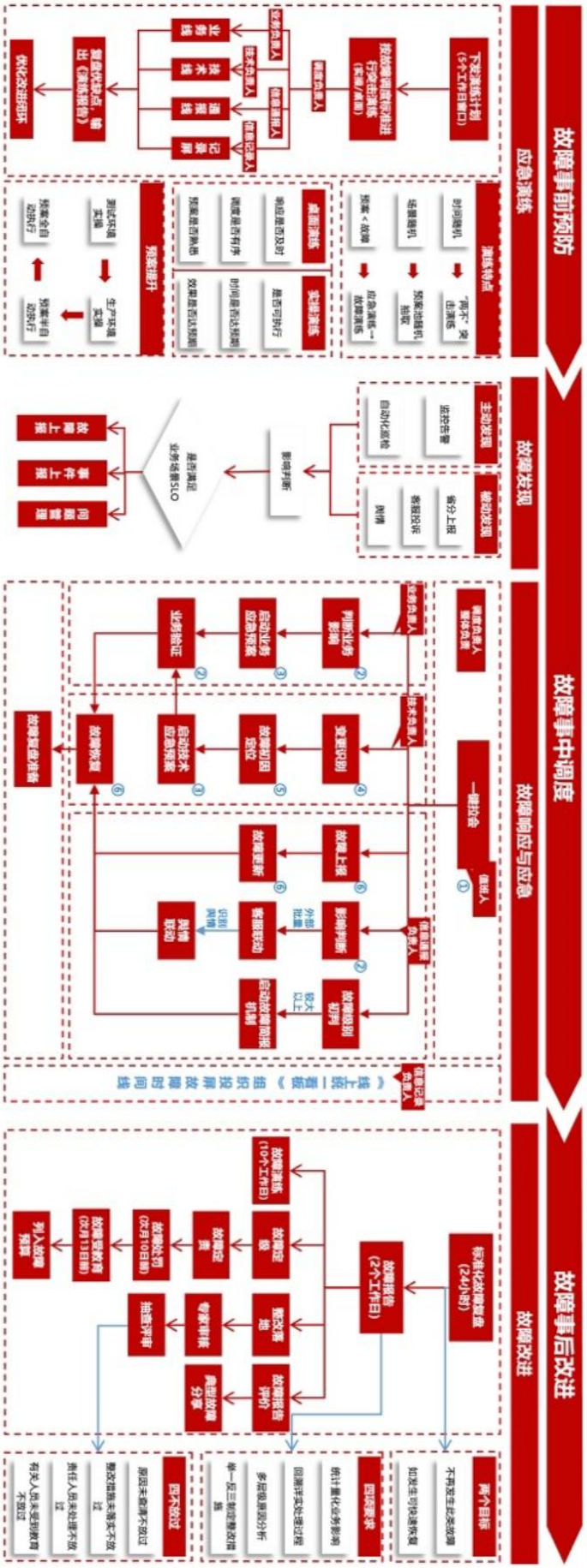


可以看到，从收到告警到恢复仅用 47s，与手工操作相比减少了 4 分钟故障恢复时间。

4.故障闭环管理

故障事前、事中、事后全流程线上闭环管理，提升故障管理质量和效率，降低故障时长及次数，提升业务连续可用率。

在事前阶段，我们会编制全面的应急预案，并定期进行应急演练。这些演练不仅有助于我们预防故障，也能确保应急预案非一纸空文，而是真正可执行的。我们会进行桌面演练和实操演练，前者用于检验流程的有效性和响应速度，后者则用于验证应急预案的执行能力和预期效果。通过这些演练，可以提升整个应急预案的可用性。


故障事前预防
应急演练
故障事中调度
故障响应与应急
故障事后改进
故障改进

在故障发生阶段，主要通过监控告警、巡检、省分报告、客户投诉和舆情等方式发现故障。一旦发现业务 SLO 被触发，将进行故障上报，并通过一键拉会功能召集相关值班人员进行会议。这些人员包括故障调度负责人、业务负责人、技术负责人、信息通报负责人和信息记录负责人，他们各司其职，确保故障得到及时、有效地处理。

在故障结束后，我们会进行故障改进。在 24 小时内进行标准的故障复盘，在 2 个工作日内完成故障报告编写，包括故障信息、根因分析、操作处置流程及问题整改措施等内容。在 10 个工作日内，我们将对故障进行演练，并根据出现的问题进行整改闭环追踪。

作为央国企，我们严格遵循“四不放过”原则，即：

1.原因未查清不放过：必须彻底查明故障原因，确保问题根源明确。

2.整改措施未落实不放过：确保所有整改措施得到有效落实，避免问题再次发生。

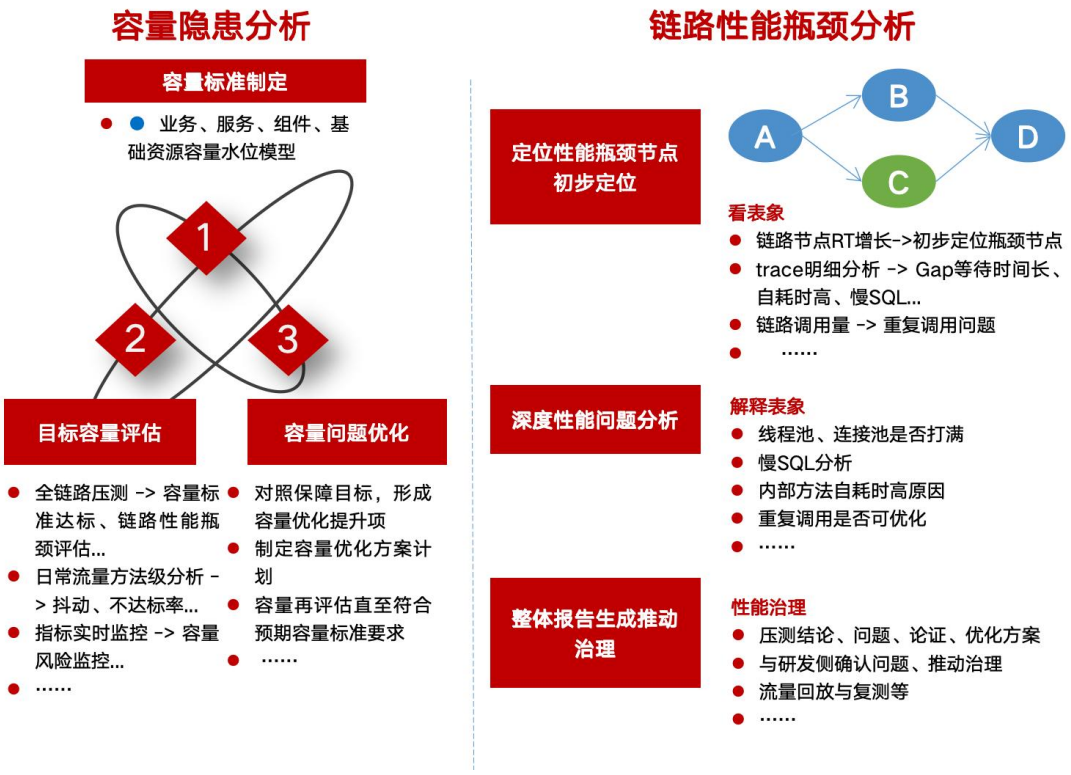
3.责任人员未处理不放过：明确每个故障的责任人，确保责任人员得到应有的处理。

4.相关人员未受到教育不放过：对相关人员进行必要的教育和培训，提高整体防范和解决问题的能力。

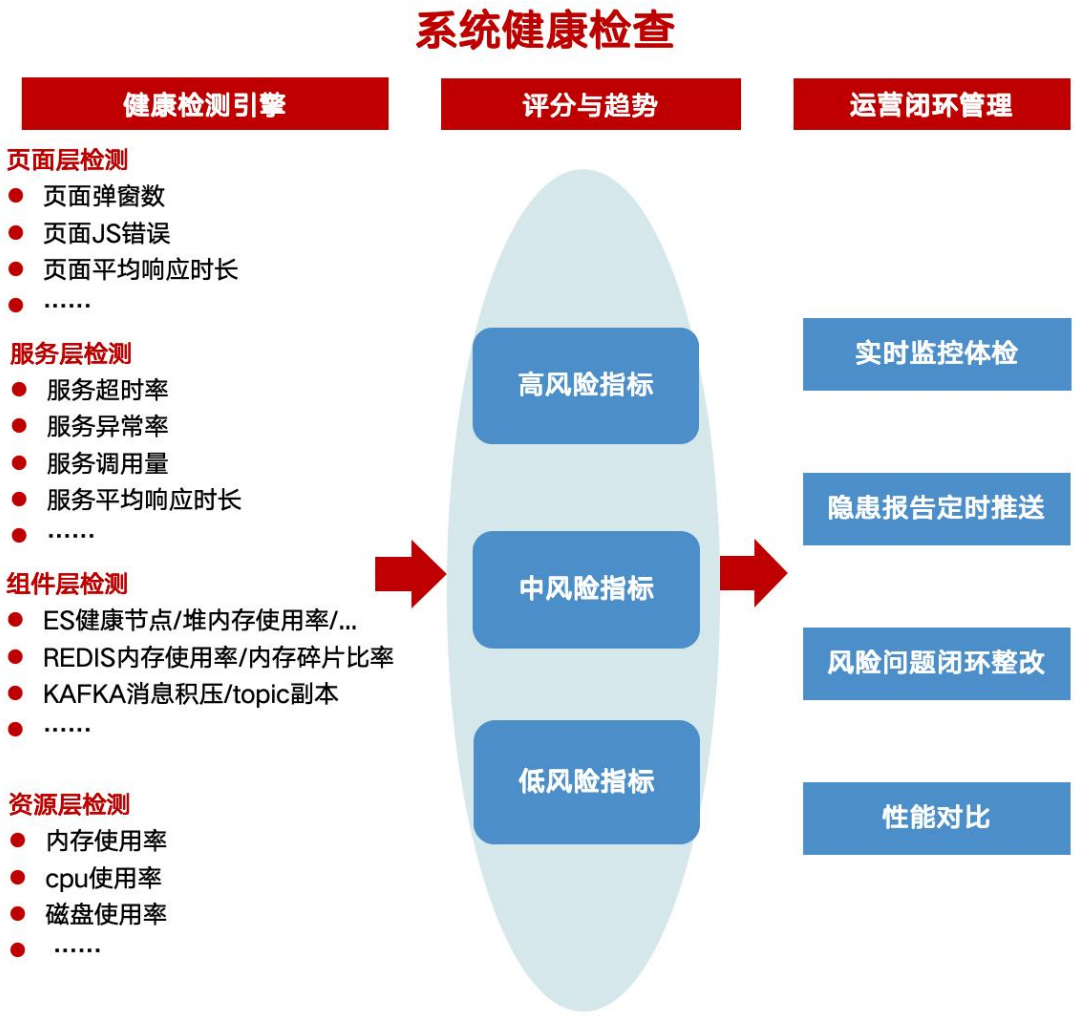
以上就是我们在故障的事前、事中、事后全生命周期的线上闭环管理方式。

5.智能隐患分析

结合监控指标与容量指标，定期开展容量隐患评估，通过核心业务链路的全链路压测，分析链路性能瓶颈，建立健康度算法模型，识别与治理系统潜在风险隐患，保障系统健康稳定。



网址：SRE-Elite.com 微信：SRE 精英联盟



亚健康检查案例



上面是一个系统亚健康的实践案例分享。通过全层级指标，包括网络层、平台层、组件层、服务层及触点层等不同层级的黄金指

标，运用 AI 算法分析来建立健康度算法模型。例如，如果 CPU 使用率超过 90%，将发出告警。但如果 CPU 使用率一直维持在 80%左右，即使没有达到告警阈值，也能通过风险等级判断其为高风险隐患，从而识别出是否为亚健康状态。

通过全面的隐患分析，可以实现系统健康状态的档案化管理。可以通过日、周、月等不同的时间维度来统计系统问题和风险情况，观测系统在各个阶段的运行情况。也进行实时的系统体检和计算，通过不同的阈值判断指标异常以及风险程度。

还可以进行版本前后的性能对比。例如，如果版本更新前有 10 个风险隐患，更新后增加到 20 个，那么此次更新就是失败的，因为它增加了风险隐患。我们需要提前规避这些问题，提前治理。如果风险隐患减少到 5 个，那么这次更新就是有效的，可以看到它解决了哪些风险隐患。通过不同时间段的对比，可以看到整个系统性能变化的趋势。

还可以定期自动生成体检报告，通过工单形式发送给相关系统的负责人，让他们明确了解系统的风险隐患以及如何进行治理。体检报告会列出相关风险指标，并提供解决方案，助力提升整个系统的稳定性。

（三）总结及展望

数字化监控平台依托生产运营保障体系，构建稳定性保障解决方案，实现了从统一监控告警到智能隐患分析的六大核心场景闭环，显著提升了系统稳定性与运维效率。不仅实现了故障的快速发现与精准定位，还通过一键智能诊断与自愈功能，大幅缩短了故障恢复时间。统一故障调度与闭环整改机制，确保了问题得到彻底解决，有效预防了类似故障的再次发生。

展望未来，随着 AI 技术的深入融合，聚焦生产运维核心应用场景，充分发挥大模型优势，基于大模型+AI Agent 能力，通过提示词工程、检索增强和智能体构建等技术手段，打造生产运维智能体系，通过 AI 工具+大模型能力的赋能生产运维新模式，基于 AI 技术的产品重构和系统之间的拉通，平台将更加智能化，实现故障的提前预警与主动干预，为业务连续性保驾护航，提升生产保障效率，从而提升系统稳定性，引领运维管理迈向新高度。

5.3.3 腾讯全球化游戏故障管理实践

SRE Elite 精选原因

腾讯游戏在全球运营的多个游戏业务中，统一使用了 SLO /SLI 方法论，对业务进行业务导向的监控可视化，并使用了 eBPF 等技术，对业务进行无死角的观测，实现了业务服务的标准化度量，故障的快速感知及定位。并能通过蓝鲸平台，实现了部分固定场景的自愈，实现了监控与批量作业的联动，降低了 MTTR，相关实践具备较强的落地性及可参考性。

（一）背景及设计原则

在腾讯全球化的游戏业务中，故障管理面临着许多独特的挑战。这些挑战主要集中在网络复杂性、文化差异、云资源使用以及法律法规的合规性上。

范围	网络	监控指标	文化差异	云资源	法律政策	时区差异
全球化业务	网络环境复杂	指标和维度更多	文化多样	混合云部署	法规多元	多时区
国内业务	网络环境相对稳定	指标和维度相对单一	文化差异不大	单云部署	法规统一	单时区

首先，网络问题是全球化游戏的主要挑战之一。由于各国的网络质量和运营商服务差异显著，导致监控指标需求增加。例如，在国内游戏环境中，专线质量通常由专门部门保障，而在海外，SRE 团队必须自行监控并分析玩家的网络质量，包括国家、运营商和城市等多个维度的网络性能差异。

文化差异也对故障管理带来挑战。例如，在印尼，由于大部分人口信奉伊斯兰教，玩家在特定的祈祷时间段内会集中掉线。这种现象在特定时间段内频繁发生，最终通过了解当地文化得以解释。同样，在泰国，每天早上 9 点的升旗仪式期间，学校的学生和老师全体站立，无法使用手机，导致游戏的在线人数显著下降。这些案例表明，了解并适应当地文化习俗对于解决全球化游戏业务中的故障至关重要。

此外，使用多种云资源也是一个重要的挑战。为了满足全球合规要求，游戏公司需要在不同地区使用多种云服务，如 GCP、AWS 和微软云。这不仅增加了资源管理的复杂性，也对 SRE 团队提出了更高的要求，需要他们具备强大的数据分析和处理能力，以应对多维度的监控指标。

法律法规的差异也是一大挑战。各国的用户隐私保护法规，如欧盟的 GDPR 和美国的 CCPA，要求游戏公司在数据管理和上报时间的一致性上严格遵守相关规定。这需要 SRE 团队具备深刻的法律合规意识，并采用相应的工具来确保合规性。

因此我们建立了事前监控，事中应急，事后总结复盘的机制，来应对相关挑战。

（二）解决方案详情及案例

事前：如何以 监控、自愈、处理的流程的角度，构建故障管理体系



大规模全球监控体系构建实践

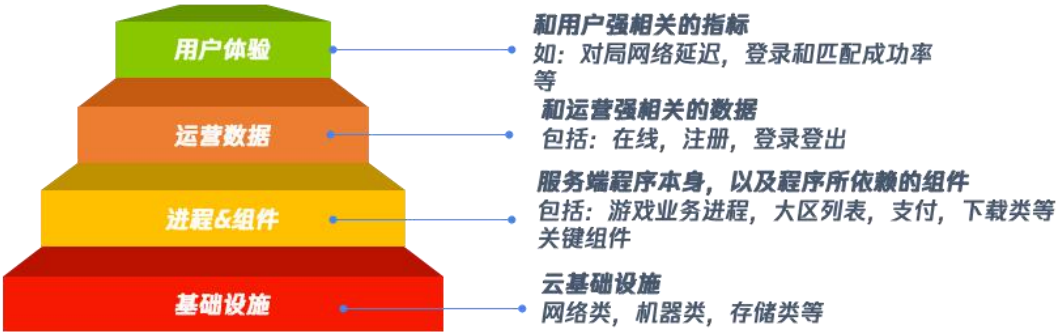
全球化监控的挑战实现快速的故障发现的挑战：

- 复杂的系统多样性：指标多，游戏登录配对对局支付成功率耗时，网络延迟、丢包率，带宽使用率抖动，技术设施可用性、性能
- 数据渠道碎片化：对象多，公共服务，游戏模块，组件模块，网络质量。
- 多维度指标：地理位置，运营商类型，策略类型，问题类型，用户类型，玩法模式类型，地图类型。
- 严格的安全合规要求：操作合规， 数据合规，代码合规……
- 渠道多，云监控，自定义监控，组件监控，日志监控，

首先，复杂的系统多样性和大量数据需要强大的数据分析和处理能力。此外，多维度的指标拆分需要 SRE 对游戏业务有深刻理解，否则上报数据无意义。严格的安全合规要求需要有强大的安全意识和工具来避免风险。

全球指标体系的建立思路

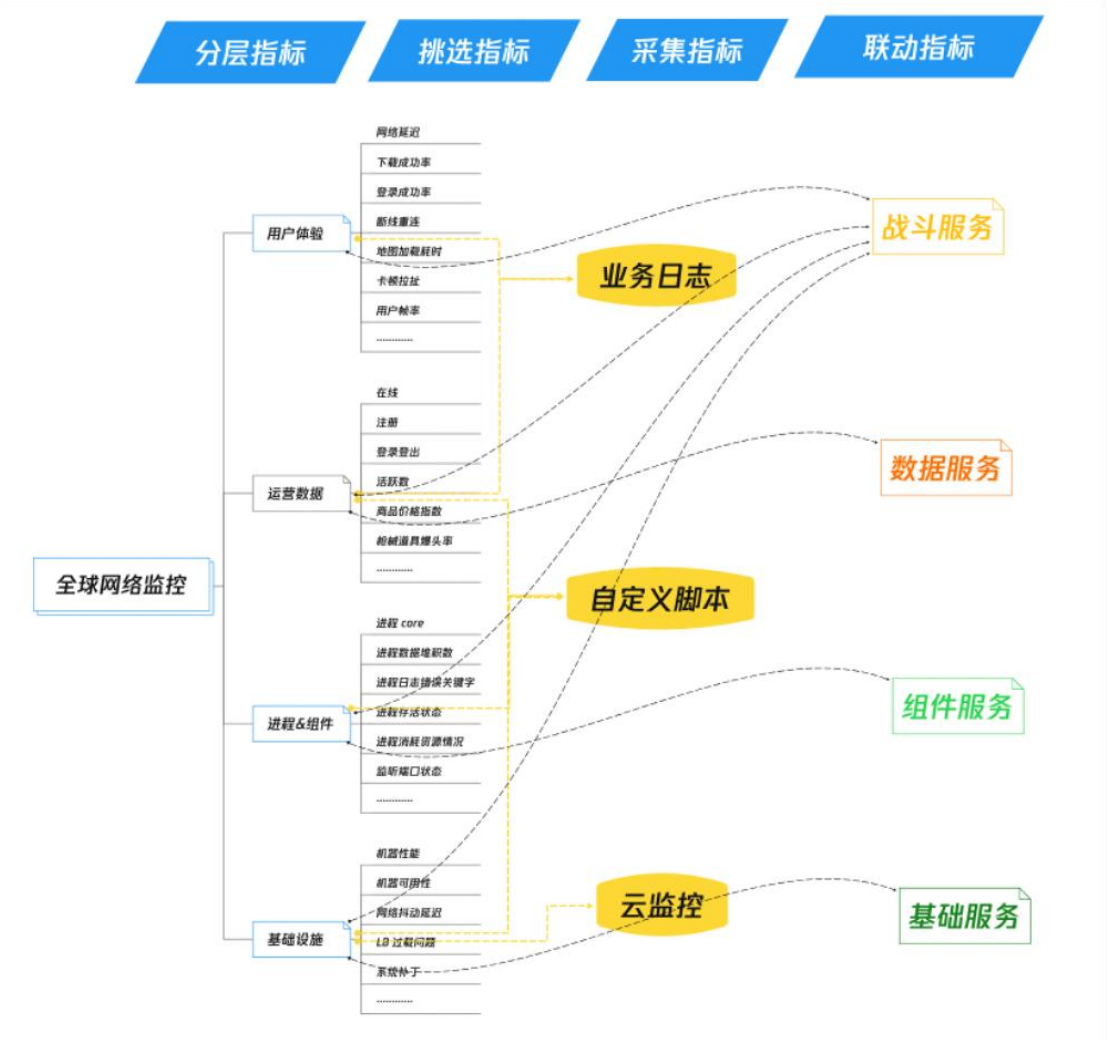
Step1：基于蓝鲸监控体系下，业务指标如何管理-指标分层



我们把业务监控指标做分层，从最下面的基础设施层到最上面用户体验层，总共四层，基础设施主要涵盖网络，机器类，存储类，进程和组件是游戏，或者组件程序本身的，运营数据包括在线，注册，最上面用户体验包括用户对局，匹配网络延迟

我们把业务监控指标做分层，通过这样的分层，我们能清晰的了解每一层的指标数据，能更立体，更全面地掌握业务质量和健康度

Step2：建立监控指标体系



有了上面的分层指标后，接下来就要把二级指标最重要的挑选出来了，哪些指标能直接反应该层的性能和健康状况，而且可衡量，关联性强的，我们就选择。

下一个就是把指标采集下来，这些指标很多来自于业务日志，包括程序日志，运营数据日志，自定义脚本上报数据，还有云监控拨测数据等

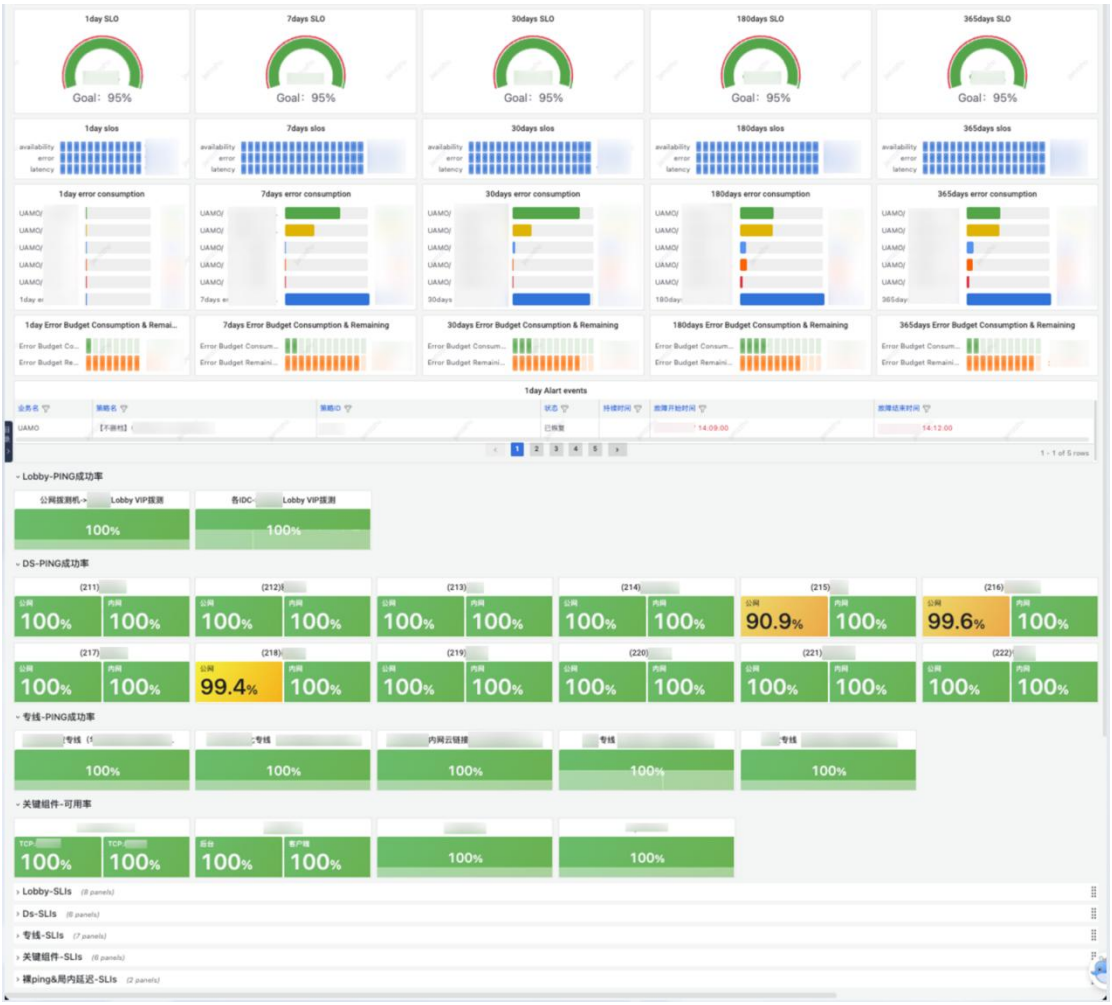
接下来需要将指标服务关联起来。这意味着你需要理解每个指标如何影响你的服务，以及服务之间的依赖关系。这可以帮助你更好地理解你的服务的性能和健康状况。

Step3：数据采集，统一管理



我们把全球的不同渠道的数据通过上报到计算平台和监控后，利用计算平台强大的数据计算，分析和存储能力，提高数据质量的安全性，提高效率。

Step4：建立网络质量的可观测大屏，快速发现和定位问题



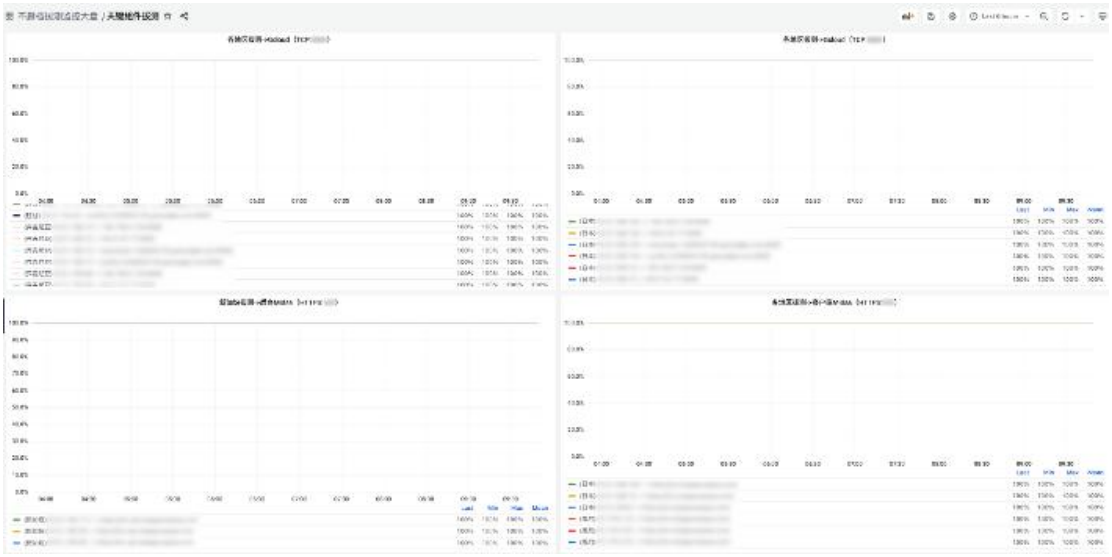
蓝鲸监控仪表盘显示的 SLO

最后，我们通过把指标和服务联动起来，通过可观测大屏呈现给用户

这是一张全球网络质量的可观测大屏，能快通过这张大屏很容易了解到关键组件，关键模块，专线的网络质量

最上面的 SLO 是代表整体，按照一天，天，一个月的时间来展示。出现问题后，可以通过下面的视图，快速定位到哪里到哪里的外网或者内网的 PING、是否有问题

Step4-1：基础设施可观测大屏



这里的基础设置，值得是游戏的基础组件，通过蓝鲸拨测解关键组件网络质量，相关组件可用性指标可能包括

- 下载服务可用性
- 登录，大区列表服务可用性
- 支付服务可用性
- 排行榜服务可用性

Step4-2：业务对局体验可观测

通过对游戏关键组件服务的端口拨测，来监控服务的可用性



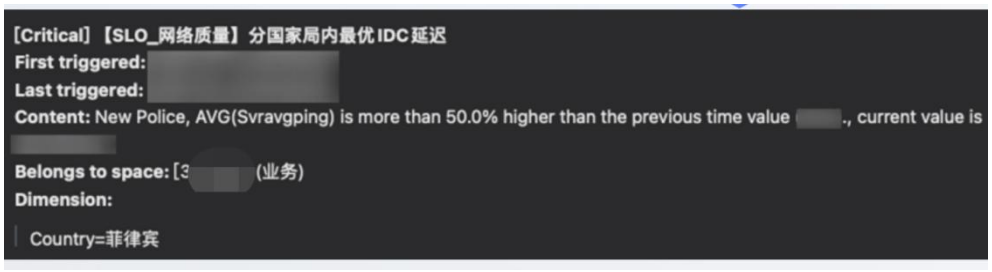
热力图是一种非常有效的数据可视化工具，可以帮助我们理解和解释复杂的数据集。

识别最佳和最差的 IDC 网络：热力图可以清晰地显示出哪些 IDC 网络的性能最好，哪些最差。颜色的深浅可以代表性能的好坏，比如深色可能代表性能好，浅色代表性能差。这可以帮助你快速地识别出最佳和最差的 IDC 网络。

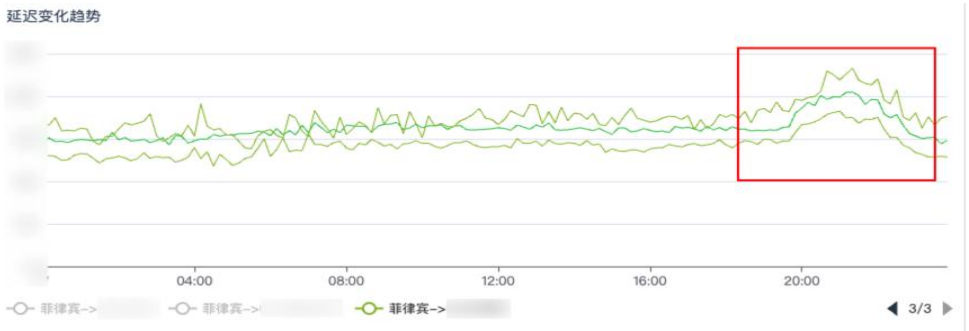
- 提供决策支持：通过识别最佳和最差的 IDC 网络，可以做出更好的决策。例如，可能会选择将更多的资源分配给性能好的 IDC 网络，或者优化性能差的 IDC 网络。
- 跟踪性能变化：热力图不仅可以显示当前的性能，还可以显示性能随时间的变化。这可以帮助你跟踪和理解 IDC 网络的性能趋势。

案例 1：维度下钻和数据关联，定位菲律宾网络问题

e.g. 晚上黄金时分，菲律宾玩家网络延迟为何突然升高？

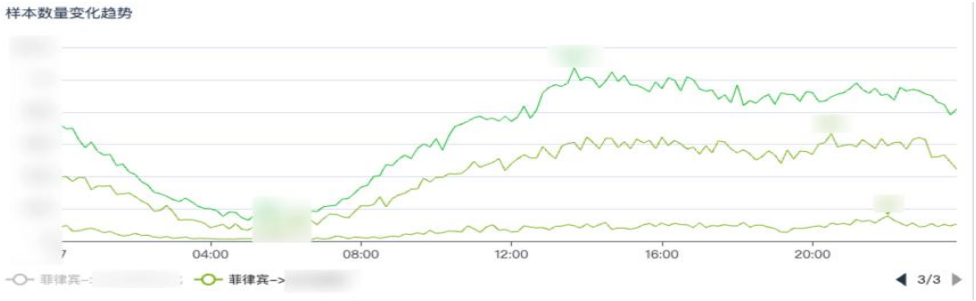


1.1. 告警触发：菲律宾最优 IDC 对局延迟升高 -> **触发监控告警**



2. 维度下钻：**检查分 IDC 对局延迟时序图** -> 对局分配重点 IDC 中，延迟均升高且趋势一致

-> 基本排除 IDC 问题



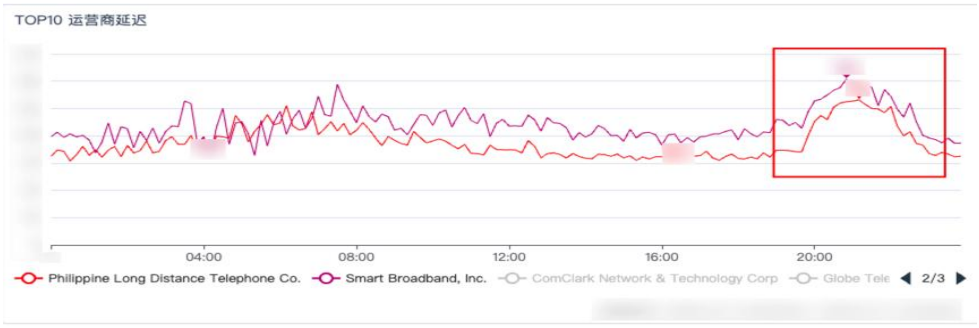
3. 关联检查：检查对局样本数量 -> 样本数量无异常

检查网络质量拨测数据 -> 无异常

检查菲律宾在线数量 -> 在线无影响

检查裸 ping 延迟 -> 有同样上升趋势

-> 基本排除业务网络链路问题，怀疑是公共网络故障



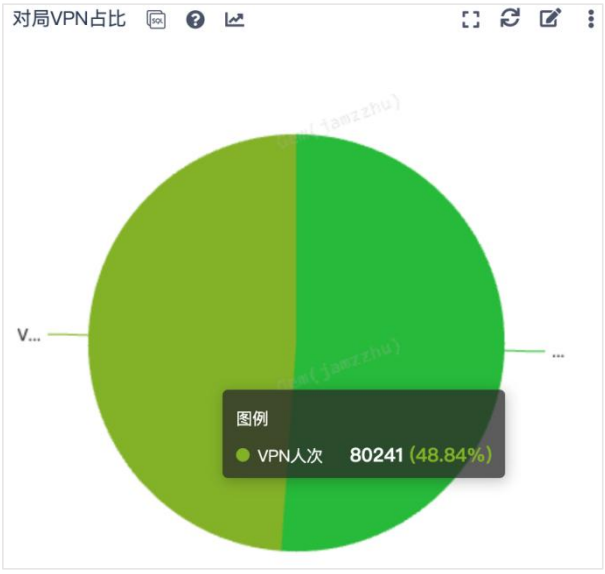
4. 换一个维度下钻：检查分运营商对局延迟时序图 -> 有两个 TOP 运营商延迟上升趋势和总体上升趋势一致

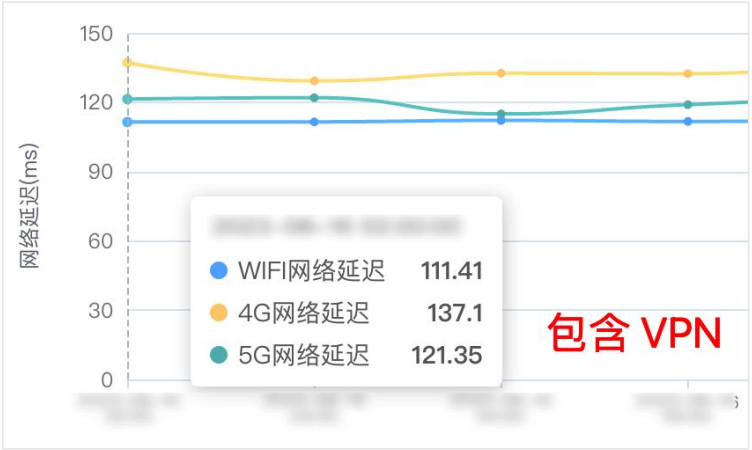
-> 初步判断为运营商问题

5. 公共网络问题求证：找云厂商核实 -> 确认为运营商光缆故障导致

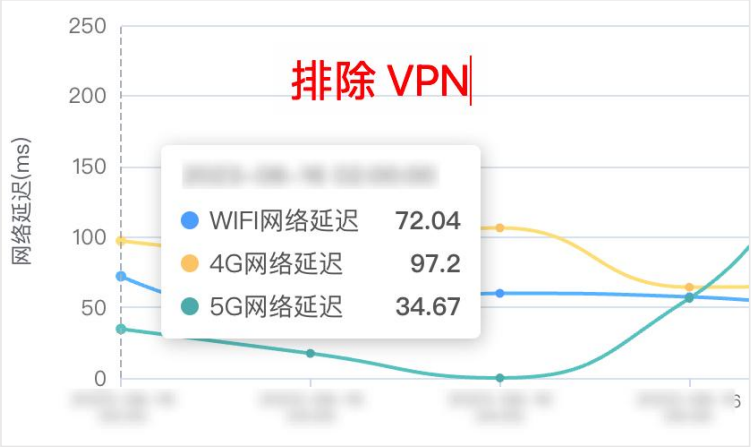
案例 2：网络可观测，数据层层剥离，破解网络疑难杂症，提供运营决策

1.新加坡玩家明明网速流畅，为何大盘延迟居高不下？



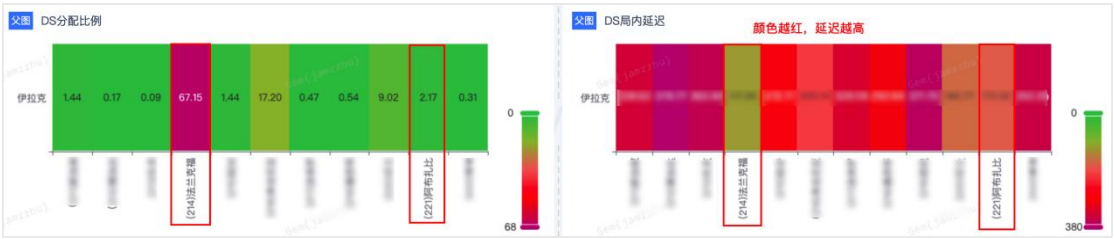


第一个例子是关于如何通过网络可观测性来解决网络疑难问题。我们以新加坡的玩家为例，新加坡的网络环境很好，但延迟却居高不下。通过分析，我们发现很多玩家使用 VPN 来访问游戏，这些 VPN 用户的 IP 地址显示为新加坡，但实际上来自其他国家。排除这些 VPN 用户后，发现实际的网络延迟显著降低，Wi-Fi 延迟只有 72 毫秒，4G 网络为 97 毫秒，5G 网络为 34.67 毫秒。



这表明问题在于使用了 VPN 的用户，并为我们的运营决策提供了依据，如针对 VPN 用户数量多的国家进行不同的运营策略。

2.伊拉克玩家为何不就近匹配，反而绕路去了欧洲节点？



伊拉克是中东国家，但它的玩家为何不连接到最近的服务器，而是连接到法兰克福？通过 trace 路由和节点分析，我们发现伊拉克

的运营能力较弱，且与周边中东国家没有直接的海底电缆或点对点连接系统。结果，网络流量被绕道到法兰克福。尽管如此，伊拉克到法兰克福的延迟约为 100 毫秒，这对某些游戏来说是可接受的，因此许多伊拉克玩家选择连接到法兰克福的服务器。

案例 3：业务关键路径质量可观测，快速发现登录问题

用户旅程与关键路径——首先，我们按照用户旅程来分析业务关键路径。这个概念源自谷歌的用户旅程分析。以游戏应用为例，用户旅程通常包括以下步骤：

下载游戏->登录->匹配->对局->支付



SLO 大盘-下半部分 对应指标的实时曲线，明确不稳定的点具体信息

问题发现与告警机制

在左上角的图表中，我们可以看到用户登录的成功率（SO）。当登录成功率出现红色警示时，我们的告警系统会立即触发告警。



指标分析

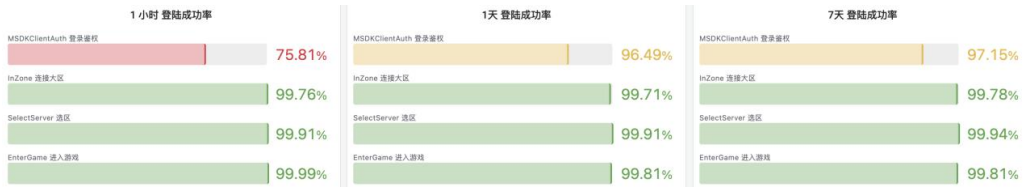
收到告警后，我们首先检查左下角的各种指标，包括在线用户数和系统容量等。确认这些指标没有问题后，我们进一步分析右侧的细节。

图表显示了登录组件（MSDK）的成功率。我们注意到，成功率从 96%骤降至 93%，并继续下降至 80%左右。这表明 MSDK 组件出现了问题。

关键节点

Time	1、msdk鉴权成功率	3、Lobby进程存活性	4、Lobby主机CPU使用率	4、Lobby主机MEM使用率
2023-02-10 11:30:00	98.68	1.00	7.32%	15.28%
2023-02-10 11:25:00	98.38	1.00	7.26%	15.27%
2023-02-10 11:20:00	98.01	1.00	7.38%	15.25%
2023-02-10 11:15:00	96.54	1.00	7.33%	15.22%
2023-02-10 11:10:00	88.37	1.00	7.01%	15.20%
2023-02-10 11:05:00	86.03	1.00	6.83%	15.20%
2023-02-10 11:00:00	86.57	1.00	7.34%	15.21%
2023-02-10 10:55:00	86.72	1.00	6.83%	15.22%
2023-02-10 10:50:00	87.73	1.00	6.95%	15.22%
2023-02-10 10:45:00	93.19	1.00	7.02%	15.22%
2023-02-10 10:40:00	98.32	1.00	7.07%	15.20%
2023-02-10 10:35:00	98.89	1.00	6.96%	15.18%





时间轴与监控图

在下方的五分钟监控图中，我们可以看到问题发生的具体时间段，大约在 10 点到 11 点之间，尤其是 10:15 左右，成功率显著下降。

问题定位与根因分析

通过进一步探查，我们发现最底层的监控图显示成功率通常在99%以上，但在问题发生时，成功率降至 75%。这再次确认是登录组件的问题。

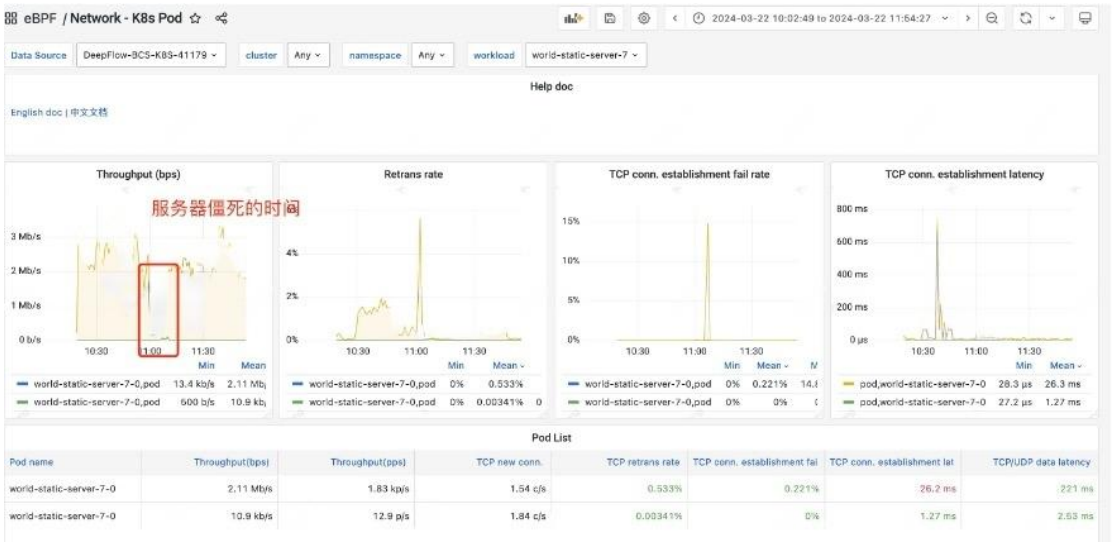
最终，我们确定问题的根因是由于 QQ 平台的周年庆活动导致系统过载，影响了我们的游戏登录鉴权服务。通过这种用户旅程的分析方法，我们能够快速定位并解决问题。

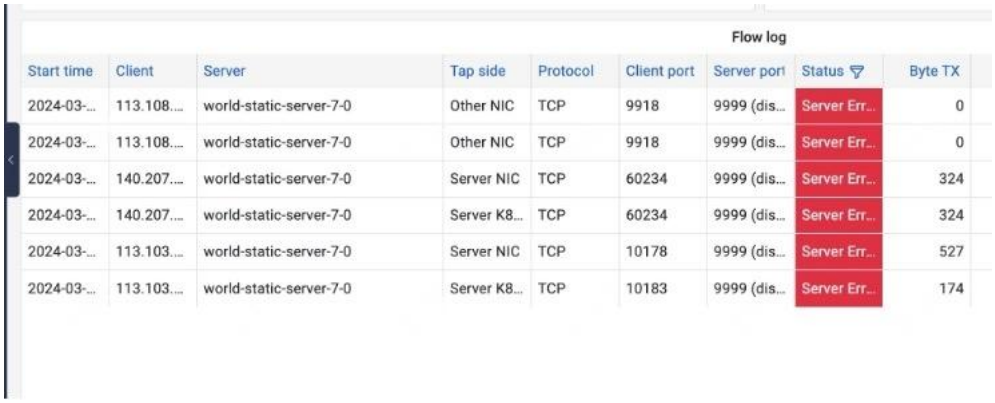
案例 4：使用 eBPF 可观测，快速发现服务器端程序问题

1.在某个副本中进行战斗的玩家突然集体卡住

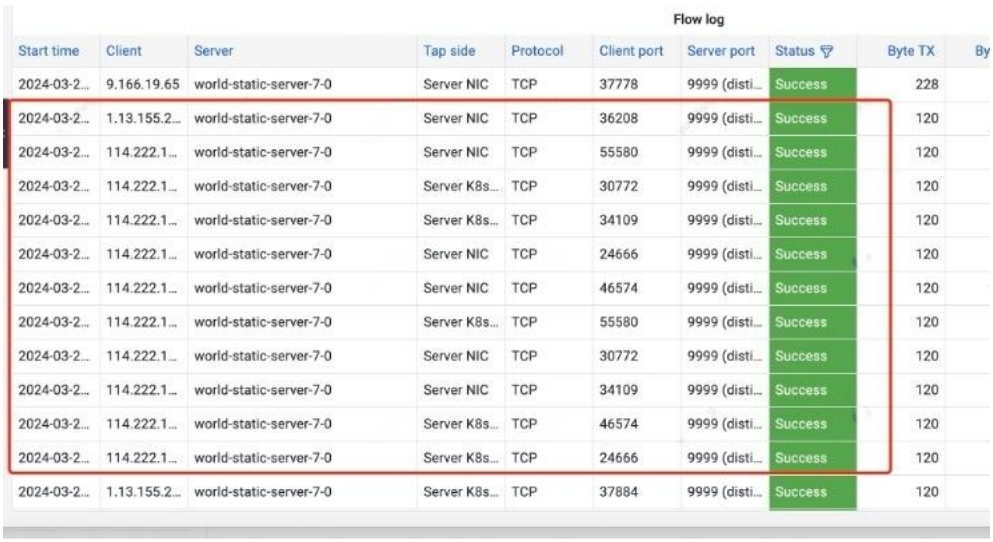
确定服务器是否在运行中：正常运行

确定服务器出入流量是否正常：流量掉 0





- 从玩家到服务器的业务流量全部是 server Error 状态
- CLB 到服务器的健康检查流量全部是 success 状态
- 是不是可以说明，只有涉及到业务协议的请求包才会出问题？



Flow log

Start time	Client	Server	Tap side	Protocol	Client port	Server port	Status	Byte TX	By
2024-03-2...	9.166.19.65	world-static-server-7-0	Server NIC	TCP	37778	9999 (disti...	Success	228	
2024-03-2...	1.13.155.2...	world-static-server-7-0	Server NIC	TCP	36208	9999 (disti...	Success	120	
2024-03-2...	114.222.1...	world-static-server-7-0	Server NIC	TCP	55580	9999 (disti...	Success	120	
2024-03-2...	114.222.1...	world-static-server-7-0	Server K8s...	TCP	30772	9999 (disti...	Success	120	
2024-03-2...	114.222.1...	world-static-server-7-0	Server K8s...	TCP	34109	9999 (disti...	Success	120	
2024-03-2...	114.222.1...	world-static-server-7-0	Server NIC	TCP	24666	9999 (disti...	Success	120	
2024-03-2...	114.222.1...	world-static-server-7-0	Server NIC	TCP	46574	9999 (disti...	Success	120	
2024-03-2...	114.222.1...	world-static-server-7-0	Server K8s...	TCP	55580	9999 (disti...	Success	120	
2024-03-2...	114.222.1...	world-static-server-7-0	Server NIC	TCP	30772	9999 (disti...	Success	120	
2024-03-2...	114.222.1...	world-static-server-7-0	Server NIC	TCP	34109	9999 (disti...	Success	120	
2024-03-2...	114.222.1...	world-static-server-7-0	Server K8s...	TCP	46574	9999 (disti...	Success	120	
2024-03-2...	114.222.1...	world-static-server-7-0	Server K8s...	TCP	24666	9999 (disti...	Success	120	
2024-03-2...	1.13.155.2...	world-static-server-7-0	Server K8s...	TCP	37884	9999 (disti...	Success	120	

进一步分析 Server 的错误日志，发现了一个关键的错误类型：Server half close flow。

错误类型分析

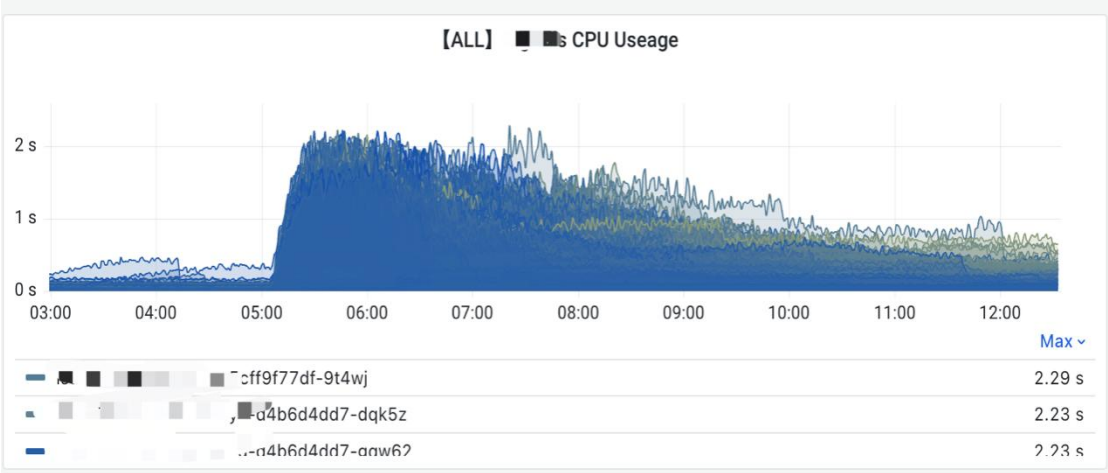
通过分析错误日志中的关键字，我们发现这是一个典型的 TCP 协议中的半连接异常。具体来说，这种情况发生在 TCP 连接的关闭过程中。

我们将错误码和分析结果反馈给开发团队，指出这是一个 TCP 协议处理中的问题。经过进一步排查，发现问题源自发行商在处理协议时的错误实现。

2.整个链路每个组件的 CPU 使用率过高

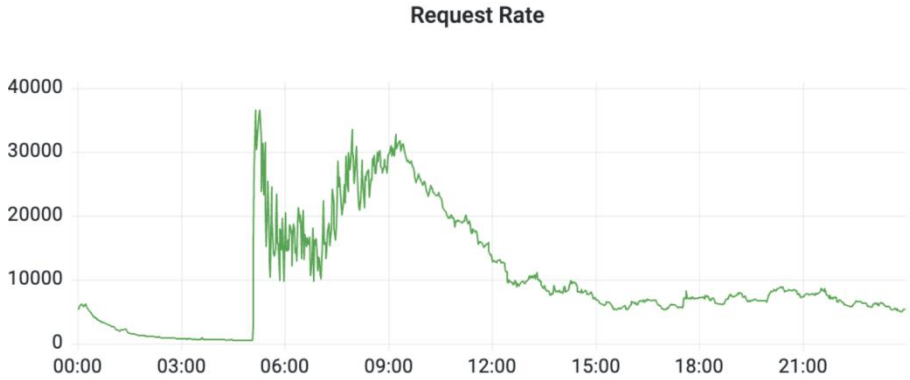
某一天，我们发现不仅仅是游戏登录模块，整个游戏链路，包括匹配模块和战斗模块等，都出现了 CPU 使用率突然增高的问题。以下是详细的分析和解决过

- 怀疑玩家数增长：相较同期无明显增长
- 怀疑服务器产生过多 gc：不符合整个链路均增长的表现



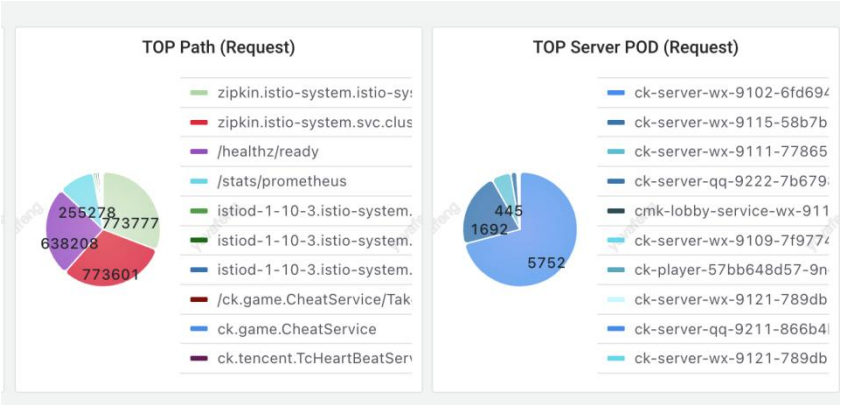
- 在玩家总数没有增长的情况下，是否有可能是单个玩家发起的请求数增多？

整个链路均增长，说明导致问题的原因与网络有关



- 通过 request rate 确认，确实存在过高的 qps

我们使用 eBPF 的 Top Server Pod 面板，找出请求数最高的前 10 个 Pod。



- 通过 top path 定位到某 pod 的接口异常非常多

我们将问题反馈给研发团队，并核实日志，定位到问题的根因如下：

- 某个 SDK 的发布新版本时，未能正确兼容游戏版本。
- 由于不兼容，SDK 不断重发请求，导致整个链路的流量激增，最终引发 CPU 使用率飙升。

eBPF 能够在内核态进行高效的网络数据包过滤和监控，帮助我们分析协议层的问题。它类似于高级的抓包工具，能够替代 Wireshark 等传统工具，极大地提高了问题定位的效率。如果没有

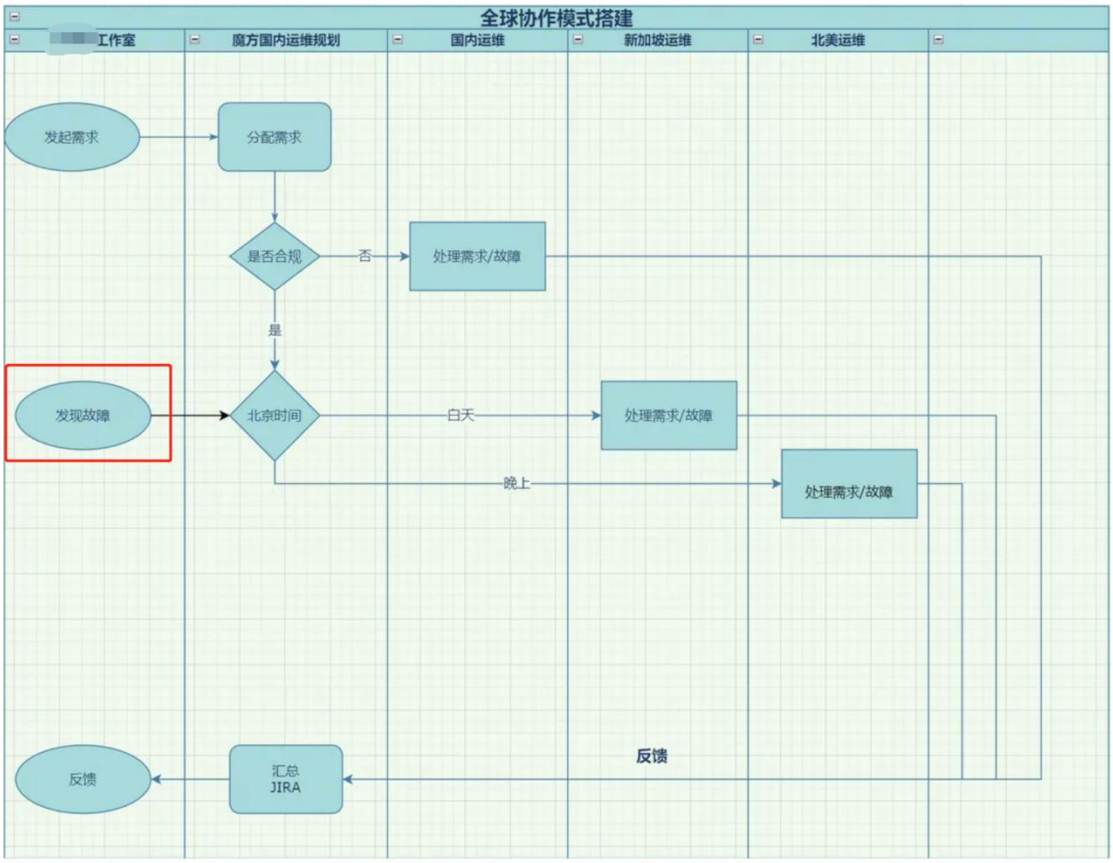
eBPF，我们可能需要逐个检查每个服务和模块的流量，耗时且难以迅速定位问题。

事中：故障快速响应并处理

建立 ONCALL 模式，保证故障得到快速响应

全球化协作流程，保证业务能得到 7*24 小时全天候服务





在我们的 oncall 模式中，告警信息的管理和收敛是关键。以下是我们如何优化告警系统，减少信息洪流，并提高告警的有效性。

建立有效的分级告警机制，减少告警信息洪流

问题

每天收到 N 多告警，等于没告警

收到告警，该谁接收和处理？

如何解决告警风暴问题：

- ✓ **区分告警等级，使用不同告警策略**
- ✓ **告警责任人自动识别，关注点分离**

大多业务噪声降低 90%+

告警等级	判断依据	ONCALL	告警渠道	告警主要责任人
致命	生产环境为主：严重影响游戏用户体验，收入类等问题。比如大面积网络和组件类故障，关键模块机器重启，程序严重 BUG 类	SG（8：00-21：00） NA（21:00-8:00）	企微致命告警群+电话告警+短信+邮件	基础设施：SRE 程序类 ：对应模块开发 公共数据类：SRE 和开发
预警	问题堆积后，可能会产生致命问题，比如生产环境容量，性能类告警	SG（8：00-21：00） NA（21:00-8:00）	企微预警群+短信	基础设施：SRE 程序类 ：对应模块开发 公共数据类：SRE 和开发
提醒	信息事件为主，比如日志错误统计类，非生产环境告警	SG（8：00-21：00） NA（21:00-8:00）	企微提醒群	基础设施：SRE 程序类 ：对应模块开发 公共数据类：SRE 和开发
过滤	无效信息，不告警			

1. 告警等级分类：

- 致命告警：涉及生产环境的重大问题，如大面积网络故障、服务器重启、关键模块故障、严重程序 bug 等。
- 预警类告警：短期内不会产生致命影响，但潜在风险较高的问题，如磁盘使用率超过 50% 或 60% 开始预警，Kubernetes 中 Pod 调度策略的预警等。

- 提醒类告警：主要是一些错误日志，开发人员在排查问题时需要查看，但不影响游戏正常运行。

2. 告警渠道优化：

- 致命告警：通过全渠道通知，直到告警接收人确认。例如，主责任人未响应时，通知备份责任人，备份责任人未响应时，通知团队领导，逐级上报直至总监或 GM。
- 预警类告警：在企业微信群中同步即可。
- 提醒类告警：仅在需要时查看，不做紧急处理。

3. 责任人区分：

- 基础设施类告警：由 SRE 团队负责处理，如服务器重启、网络故障等。
- 程序类告警：由对应的后台开发人员负责处理，提前预设告警名单，有针对性地通知相关人员。
- 公共数据类告警：根据具体情况分配给相关团队或人员。

4. 故障治愈小程序：

- 在故障治愈小程序中，可以直接 @ 相关的 SRE 或开发人员，提高响应效率。

建立故障应急处理分工协助流程

Google 的故障指挥体系



谷歌的故障指挥体系的话，它是参考 1968 年这个美国森林大火的这样的一个体系，

属于腾讯游戏的故障指挥体系

当一个问题被定性为故障，这时我们就要成立 War Room，如果是在办公时间，大家可以聚集到同一个会议室，或者同一块办公区域集中处理；

如果是非办公时间，可以是视频、电话或微信会议的方式，形成虚拟的 War Room。

但无论是真实还是虚拟的 War Room，根本目的是快速同步信息，高效协作。在特别时期，比如业务上线这样的关键节点，咱们做法一般是在一个会议室，运维和项目组坐在一起。在已经形成双十一文化的阿里，他们把这样的办公地点称为“光明顶”，各大高手齐聚于此，共同保障业务和系统稳定。

我们一般要求以团队为单位，每隔 10～15 分钟做一次反馈，反馈当前处理进展以及下一步行动。

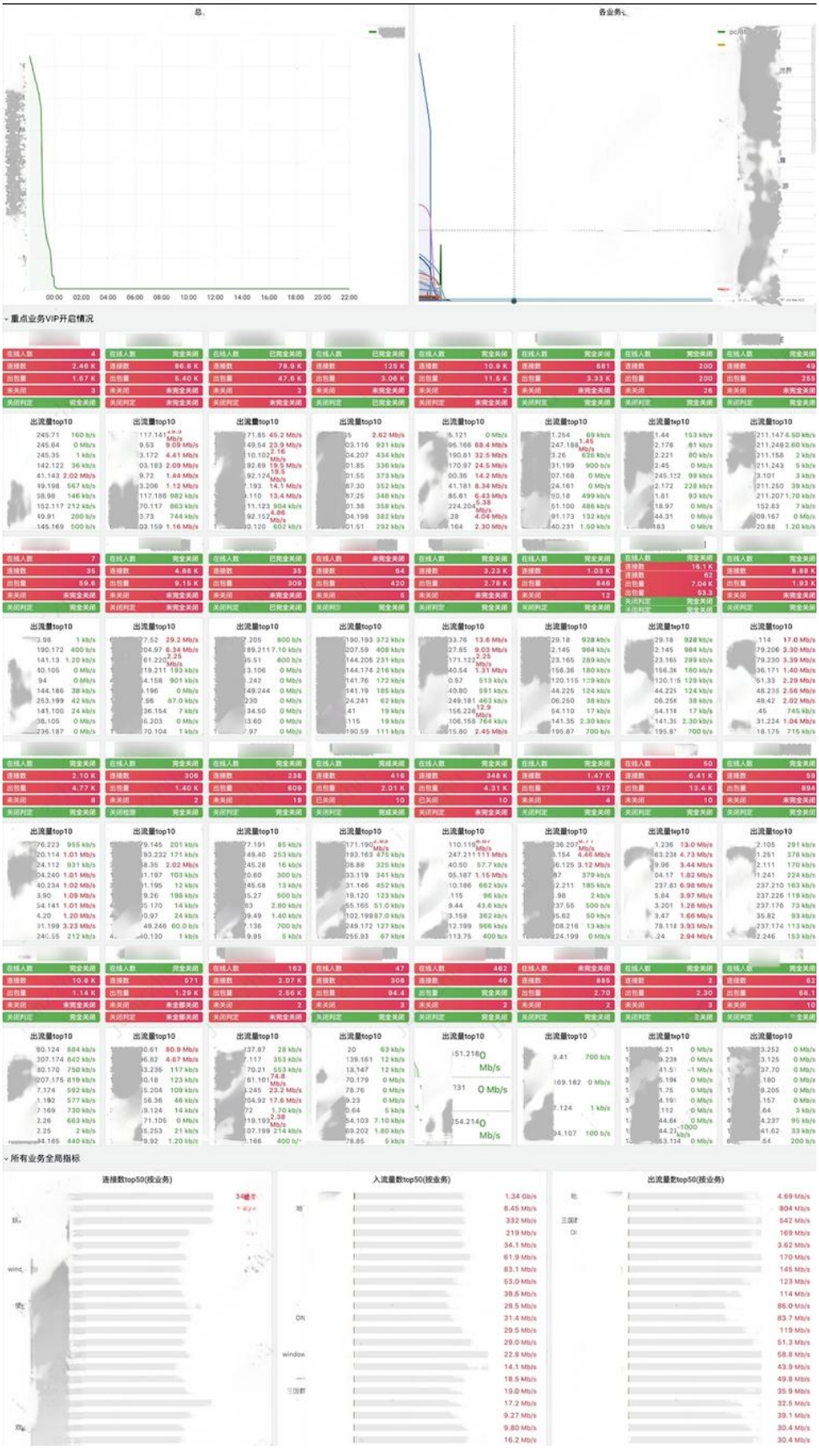
如果中途有需要马上执行什么操作，也要事先通报，并且要求通报的内容包括对业务和系统的影响是什么。最后由 IC 决策后再执行，避免忙中出错。

强调一点，没有进展也是进展，也要及时反馈。

很多团队和成员往往会抱怨，我专心定位问题还没结果呢，有什么好反馈的呢？

但是没有进展也是进展，IC 会跟 OL 以及团队主管决策是否要采取更有效果的措施，比如 10 分钟之内还没定位结果，可能就会选择做有损的降级服务，不能让故障持续蔓延，这个时候，反馈就显得尤为重要。

War Room 成员组成				
角色	IC	CL	OL	IR
负责岗位	总监	Leader/运营规划	运营规划	各系统开发/运维/SRE


总.
各业务.
重点业务VIP开启情况
所有业务全局指标
连接数top50(按业务)
入流量数top50(按业务)
出流量数top50(按业务)

从实践经验来看，如果是大范围的故障，一般就是总监来承担 IC 职责，接下来他就可以从更高的层面组织和协调资源投入并有效协作。

这时运维回归到 OL 的职责上，负责组织和协调具体的执行恢复操作的动作。

利用蓝鲸故障自愈快速恢复业务故障

故障自愈是一个非常重要的特性，它可以使系统在出现故障时自动恢复，而无需人工干预。这是一个非常强大的功能，可以显著提高系统的可靠性和可用性，同时也可以减少运维人员的工作负担。

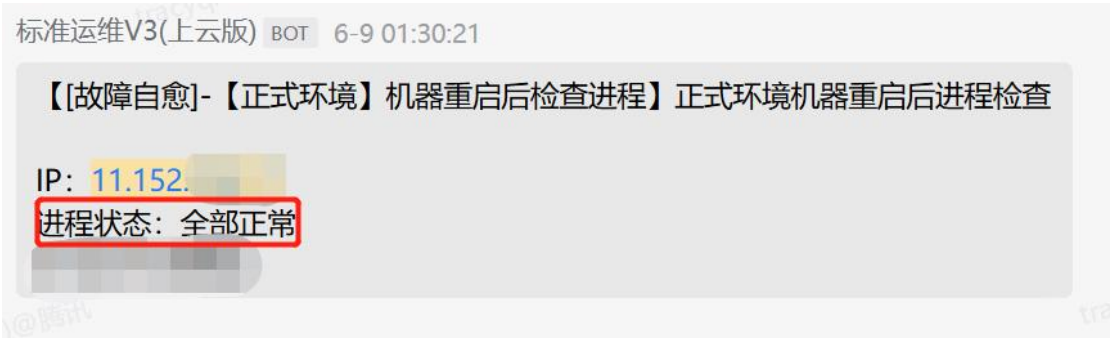
为了应对这些挑战，需要建立一个完整的故障管理流程，从故障发现到改进，特别强调故障治愈，优化 MTTR 时间，通过系统自动化减少人工干预。这些措施能够有效提升全球游戏运营的稳定性和安全性。

某业务使用故障自愈恢复业务进程

1 分钟自动处理，减少 MTTR 时间

ID	套餐名称	套餐类型	关联策略	触发次数(近 7 天)	最近更新人
#107001	【标准运维】【现网】屏蔽集群扩容告警策略	标准运维	1	10	
#99592	【标准运维】【现网】zonesvr通知在线人数	标准运维	1	--	
#90275	123	一键拉群	--	--	
#67588	正式环境致命告警ITSM建单	流程服务	198	--	
#69188	【标准运维】【现网】故障自愈拉起单主机进程	标准运维	2	2	
#69163	【标准运维】【预发布】故障自愈拉起单主机进程	标准运维	1	--	
#65697	【通知套餐】【运维】xwork捕获CVM运行异常通知	标准运维	1	--	
#63429	【快捷套餐】转移主机至空闲机模块	标准运维	--	--	
#63187	【快捷套餐】重启网管Agent	标准运维	--	--	
#63186	【TCM】重启idipsvr和midas_ntf_svr（指定IP地址...	标准运维	--	--	





利用蓝鲸故障自愈快速扩缩容

在复杂的游戏运营场景中，尤其是全球游戏的运营中，快速响应和资源管理至关重要。我们通过故障治愈系统，实现了自动化的扩缩容，确保游戏在不同区域的资源使用最优。

场景背景

游戏上线后，不同区域在不同时间点的资源需求会有所不同。例如，一个区域在某个时间点可能资源富余，而另一个区域可能需要扩容以应对玩家增长。为了高效管理这些资源，我们引入了 AIOPS 能力，通过智能监控和预测模型，实现自动化的扩缩容。





每天定时跑检查结果



记录扩缩容，节省成本直接展示

自动扩缩容实际执行结果

时间	执行状态	大区	实际扩缩容数量	执行时常	缩减成本/月/天
2024-05-10 15:46:00	成功	弗吉尼亚	-2	3 hour	¥-2
2024-05-10 15:47:00	成功	弗吉尼亚	-2	3 hour	¥-2
2024-05-10 15:48:00	成功	弗吉尼亚	-2	3 hour	¥-2
2024-05-10 15:49:00	成功	弗吉尼亚	-10	3 hour	¥-1

事后：复盘总结，从故障中学习并改进

复盘是我们在运维和故障处理中不断改进和提升的重要环节。**简单重复 10000 次，不如有效复盘 1 次，把经验转化为能力，将失败转化为“有意义的失败”**

故障复盘底层逻辑

- 故障是无法完全避免的
- 杜绝重复犯错，包容失败
- 根因分析要全面

在故障复盘中，发生故障后分析原因，总结经验教训，并制定改进措施，以防止类似故障再次发生。

一个经典的复盘实践案例来自航空业。几十年前，美国航空事故频发，死亡率高达 25%。通过引入“黑匣子”记录飞行数据和语音通信，专业团队对事故进行详细复盘，找出问题根源并改进。如今，航空业的安全性大幅提升，每年因空难死亡的人数降至 3-400 人，飞行已成为地球上最安全的交通工具之一。举个例子，如果有人需要每天坐一次飞机，那么大概 3200 多年才会遇上一次空难。

航空业能想到这么细致的原因，不是因为他们想象力丰富，而是能够及时把过去曾经发生过的错误，最大程度转化为未来可以小

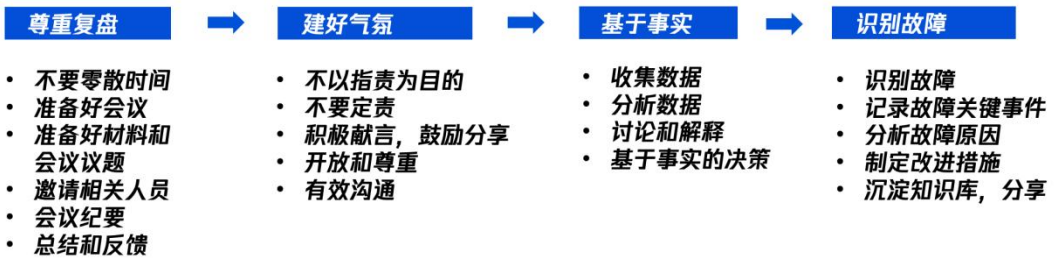
心和避免的方法，这类行为其实就是我们今天要重点讨论的故障复盘。

故障复盘的目标



复盘的主要目标是通过系统性的分析和总结，找出故障原因，降低故障率，提高系统的稳定性和可靠性。

故障复盘关键要素



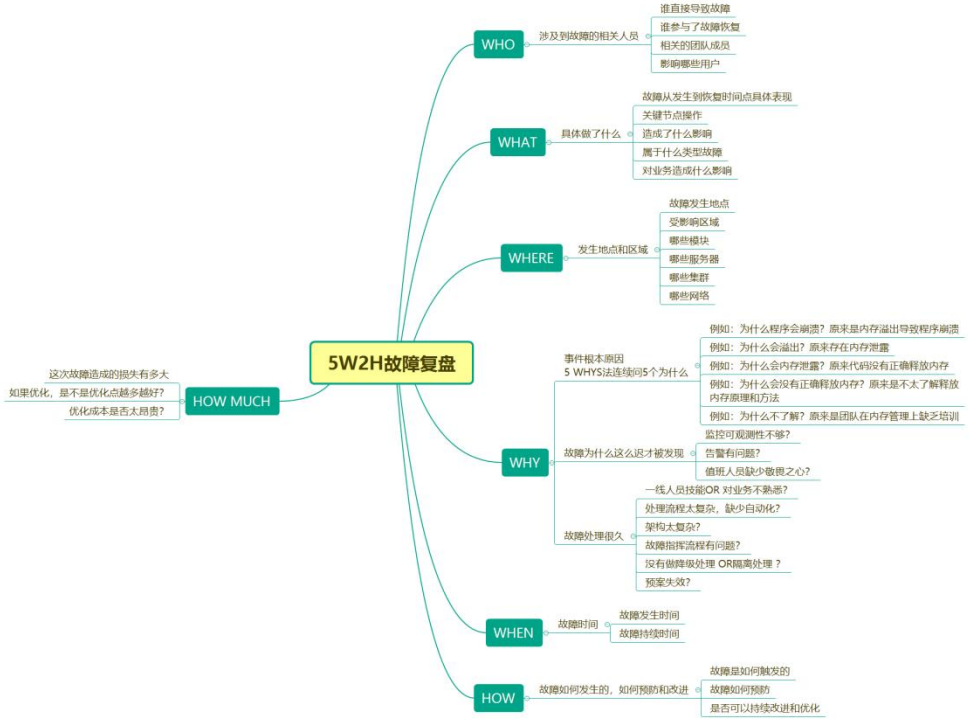
复盘方法

我们采用“5W1H”方法进行复盘：

1. Who（谁）：涉及哪些人？
2. What（什么）：发生了什么事情？
3. When（何时）：事情发生在什么时候？
4. Where（何地）：事情发生在什么地方？
5. Why（为什么）：为什么会发生？

在实际操作中，我们会：

1. 记录故障数据：详细记录故障发生时的所有数据和日志。
2. 召开复盘会议：邀请相关人员参加，按照“5W1H”方法进行讨论。
3. 制定改进措施：根据复盘结果，制定和实施改进措施。
4. 跟踪改进效果：持续跟踪改进措施的效果，确保问题得到解决。


5W2H故障复盘
WHO
涉及到故障的相关人员
谁直接导致故障
谁参与了故障恢复
相关的团队成员
影响哪些用户
WHAT
具体做了什么
故障从发生到恢复时间点具体表现
关键节点操作
造成了什么影响
属于什么类型故障
对业务造成什么影响
WHERE
发生地点和区域
故障发生地点
受影响区域
哪些模块
哪些服务器
哪些集群
哪些网络
WHY
事件根本原因
5 WHYS法连续问5个为什么
例如：为什么程序会崩溃？原来是内存溢出导致程序崩溃
例如：为什么会溢出？原来存在内存泄露
例如：为什么会内存泄露？原来代码没有正确释放内存
例如：为什么会没有正确释放内存？原来是不太了解释放内存原理和方法
例如：为什么不了解？原来是团队在内存管理上缺乏培训
故障为什么这么迟才被发现
监控可观测性不够?
告警有问题?
值班人员缺少敬畏之心?
故障处理很久
一线人员技能OR 对业务不熟悉?
处理流程太复杂，缺少自动化?
架构太复杂?
故障指挥流程有问题?
没有做降级处理 OR隔离处理 ?
预案失效?
WHEN
故障时间
故障发生时间
故障持续时间
HOW
故障如何发生的，如何预防和改进
故障是如何触发的
故障如何预防
是否可以持续改进和优化
HOW MUCH
这次故障造成的损失有多大
如果优化，是不是优化点越多越好?
优化成本是否太昂贵?

复盘模板参考

某故障复盘模板

故障概述

1. 主题：〔简述什么时间点发生了什么动作，做了什么操作，造成什么影响〕
2. 业务：xx业务
3. 时间：xx时间-xx时间
4. 责任归属：xx
5. 事故影响：xx

故障原因

〔分析故障的直接原因和根本原因，包括人为因素、权限因素等，有针对性得预防类似事故再次发生〕

1. 平台权限控制xxx问题
2. 安全操作流程执行不严格
3. xxx

事件回顾-时间线描述

〔详细描述事件从需求产生、发生事故、处理过程、故障恢复的整个流程〕

1. xxx时间点，收到XX告警
2. XXX时间点，收到XXX告警
3. XXX时间点，运维开始处理故障
4. XXX时间点，确认是XX模块故障
5. xxx时间点，运维重启该模块进程，故障恢复

故障影响

〔对事故过程中项目影响与损失进行评估，了解事故严重程度〕

1. 影响小区掉线xxx人
2. 影响收入XXX

整改措施

1. 针对事故发生的原因提出整改措施，完善制度

总结与反思

1. 对事故反思与总结，分析事故中的问题和不足，举一反三，提出改善建议

（三）总结及展望

总结

最后，我们回顾下今天主要分享的内容。全球化业务所面临的挑战主要包括网络环境的复杂性，指标和维度的增多，文化的多样性，多云部署，以及法规的多元化。这些问题也给全球业务的故障管理带来了诸多挑战，如系统的复杂性，数据渠道的碎片化，多维度指标，以及更严格的安全合规要求。

为了解决这些问题和挑战，我们提出了四步法：分层指标，挑选指标，采集指标和联动指标。通过可观测性，我们能够展示关键路径和关键网络质量大屏，以便于发现和定位问题。我们还分享了一些典型案例，通过关联检查，维度下钻，以及 eBPF 等技术手段来发现和定位问题。

在故障快速响应和处理方面，我们采用了 Oncall 机制，在全球三地协作中，根据告警时段和告警严重等级进行分类处理。我们建立了故障知会体系，一旦出现故障，我们会建立 War Room 作战室，多人协同快速同步和处理故障，以尽量减少影响范围。

最后，我们需要养成故障复盘的好习惯，明确故障复盘的目标和目的，使用 5W1H 复盘法，以提高复盘的效果。

展望

随着人工智能技术的不断发展，如何通过 AI 模拟人类思维，通过机器学习，深度学习，大模型等能力来理解系统的复杂性，从

而预测故障，以及快速发现和定位故障，处理故障将是一个非常有挑战性，也非常有价值的方向。

目前腾讯游戏结合蓝鲸平台 AIOps 能力，已经初步落地了资源智能扩缩容，业务全球化智能选择节点，异常检测，故障维度推荐等场景。另外，我们也在积极尝试利用 eBPF 全链路关联数据的特点进行根因定位。相信技术的发展，人工智能在故障治理场景上落地场景将会出现一个百花齐放的局面，期待未来能与大家有更多的分享。

5.3.4 XX 银行应急管理一体化平台建设实践

SRE Elite 精选原因

XX 银行是中国乃至全球规模排名前列的商业银行，业务众多，客户群体遍布全球，且适逢整体 IT 架构升级，数字化转型深入，技术挑战巨大。在这种背景下，XX 银行构建了符合金融行业强监管特性的三个一体化的应急管理平台：通过“一体化技术平台”实现了底层能力平台 PaaS 化，满足各种底层操作原子化包装的需求；通过“管理操作一体”，实现应急管理思想和自动化操作的同步；通过“数据融合一体化”，实现应急决策所需配置数据、执行数据、性能数据、变更数据的统一管理和展示 。通过以上三个一体化，降低了业务稳定性的风险，值得广大金融行业参考。

（一）建设背景

xx 银行是中国最大的商业银行之一，在全球范围内运营，提供包括存款、贷款、信用卡、财富管理、外汇交易等多样金融服务，客户群体遍布各个经济领域。在信息化投入方面，XX 银行一直走在国内银行业的最前列，最早进行核心银行系统升级，最早建立多数据中心，最早推广云计算技术，提升业务的弹性和响应速度等。通过这些信息化的投入，XX 银行提高了自身的运营效率，也在客户服务、风险管理、市场反应速度等方面取得了显著的进展，保持了行业领先地位。

随着 XX 银行信息化建设和数字化转型深入推进，IT 部门数字化系统规模不断扩大，各类 IT 设备数量呈指数级增长，信息网络和系统架构复杂度不断提高，运行风险持续增加。为提高业务系统可用性和持续性，公司建设了应急管理一体化平台，支撑业务运维及应急响应体系高效运转。

在此之前，XX 银行存在着大量业务系统应急处置以线下维护为主的特点，并且各系统应急处置组织无系统固化，事件应急预案管理不完善，对应急处置实操性不强；缺少常态化的培训和演练机制，业务支撑人员应急技能掌握不足，导致在应急处置时过程协调复杂、难度大，应急处置时间长，故障易上升到高程度的事件级别。故障事件处置过程没有形成知识经验，无法及时查找应急操作指引，事后复盘没有彻底执行及改善。基于此，建立线上、流程化

的应急管理响应一体化平台，实现应急事件的全生命周期管理，为内部整体信息系统稳定运行提供基础保障。

（二）解决方案详情及案例

基于上述背景，数据中心相关领导一直在思考如何建设一个应急管理一体化平台，通过提升 MTBF（平均无故障时间）时间，降低 MTTR（平均恢复时间）时间，就可以有效地管理应急全流程。这里我们总结了以下“1-2-3”设计原则：



图.应急整体业务流

一个目标：快速止血，第一时间恢复生产。

两大场景：覆盖“事件应急”和“灾备应急”两类场景。

三个一体：“技术平台一体”，实现底层能力平台 PaaS 化；“管理操作一体”，实现应急管理思想和自动化操作的同步；“数

据融合一体化”，实现应急决策所需配置数据、执行数据、性能数据、变更数据的统一管理和展示。

一个目标，快速止血，保障业务稳定性

应急管理的总体原则：

（一）第一时间恢复生产：生产事件发生后，事件发生单位应第一时间确认影响，根据应急预案组织开展故障恢复。

（二）统一领导、分级负责：各应急小组接受应急领导小组统一指挥，按照事件不同影响程度，分别启动应急组织架构。

（三）预防为主、演练结合。

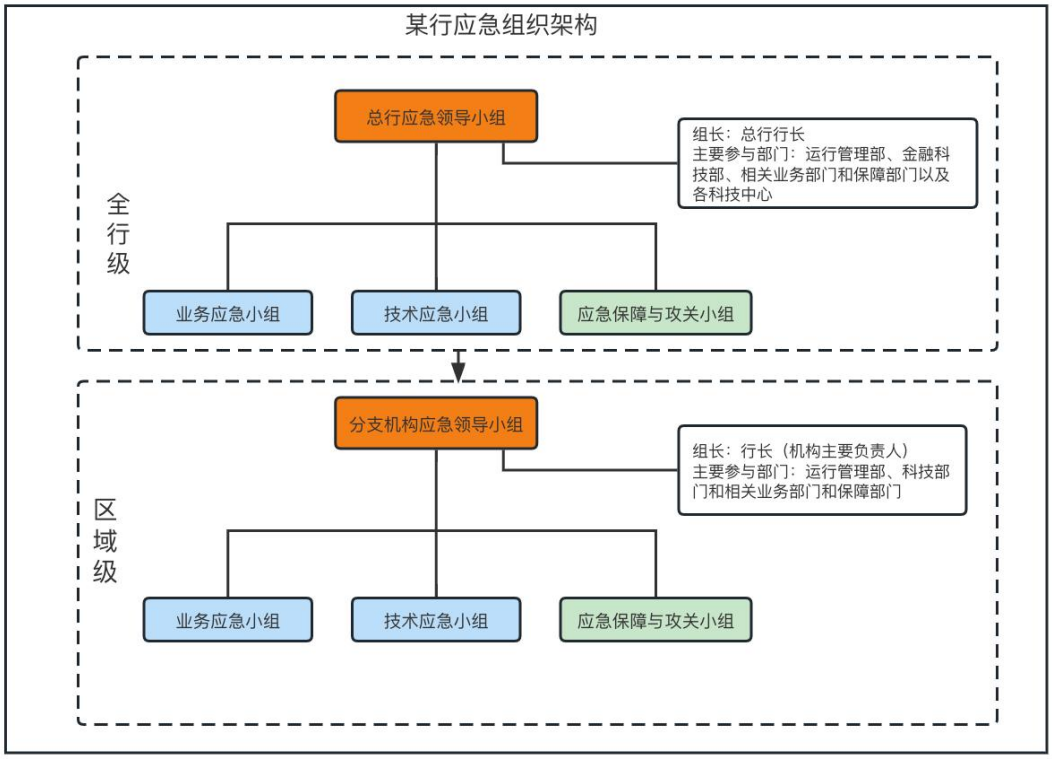
在应急事件产生时，可以按照不同维度设置不同群组管理应急人员，通过多种方式通知到应急人员，一、二线协同应急，该应急组织相关角色会迅速返回数据中心，通过移动端签到等方式了解人员到位情况，针对业务故障，采取暂不问责，暂不问询根因的原则，进行业务恢复快速处置

利用故障根因和故障处置的应急专家库匹配，根据不同故障类型，推导决策提供用户故障处置建议，常见的故障处置类型如下：

- 点状故障:某容器问题、某模板问题、某宿主机问题、某服务问题，则可通过容器重启、宿主机隔离、服务管理平台禁用服务
- 面状故障:园区故障、K8s 集群故障、Ceph 故障，则通过 F5/DNS 切换处理;

- 变更故障：版本升级问题，则直接提供 PaaS 平台回退接口；
- 容量不足：CPU/内存不足/ DSF（分布式服务框架）日志中存在连接池满，则提供 PaaS 扩容接口；

根据《商业银行业务连续性监管指引》外部要求，XX 银行制定了行内应急组织架构。应急处置组织架构分为全行级级应急组织架构和区域级应急组织架构。



应急组织各角色职责：

1. 应急领导小组：负责制定应急备份与恢复的目标，负责应急决策，负责应急过程的总体指挥调度和组织协调，负责督导应急处置实施；
2. 技术应急小组：包括应用、系统、网络、硬件、安全等专业，负责技术应急处置，负责应急过程中提供决策依据和技

术支持，负责做好相关专业之间应急处理工作的沟通协调，负责应急过程处理步骤的记录，负责汇报应急处理进展；

3. 业务应急小组：负责业务应急处置实施工作，负责做好相关专业之间应急工作的沟通协调，负责组织账务修复、验证和确认等；
4. 应急保障与公关小组：负责与其他小组的沟通和协调，负责提供应急所需的人力、物力、财力等资源保障工作，负责信息对外报告、披露和解释、媒体公关、安全保障、秩序维护、法律咨询等。

两大场景，覆盖“事件应急”和“灾备应急”两类场景

应急管理一体化平台覆盖事件应急和灾备各自有其独特的作用和流程，二者紧密相连，共同构筑起企业应对突发事件和灾难的完整防线。事件应急通常是灾备应急中提取的部分处置步骤，灾备应急也同样是事件应急的最后一道防线。事件应急旨在快速响应和处理突发的信息系统事件，如网络攻击、数据泄露、系统故障等，以尽量减少对业务的影响，场景多样，因此需要不断完善运营，丰富其预案下的不同场景，以便在故障事件发生时，能够快速检索推荐处置策略，实现业务快恢。

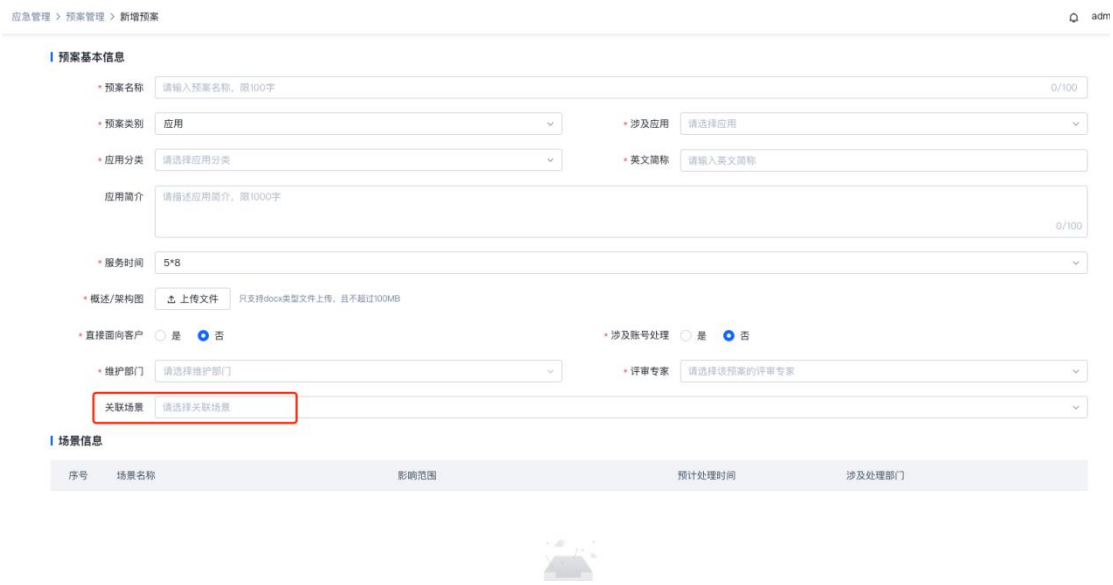


图.应急预案

应急场景是对事件的模拟，旨在通过具体的情景测试应急预案的有效性和可操作性。应急场景可以帮助识别预案中的漏洞和不足，提供实践经验。应急场景的演练结果可以反馈到应急预案中，使预案不断优化和更新，确保其在实际事件中更加有效。



图.应急场景

经过多年的应急演练和生产应急实践，沉淀了面向应用运维和基础运维的各类应急场景 3000+，如信用卡应用系统相关的应急场景：

序号	预案名称	应用分类	场景名称
1	信用卡产品应急预案	A 级	信用卡产品调整：服务限流与服务禁用
2	信用卡产品应急预案	A 级	信用卡缓存异常
3	信用卡产品应急预案	A 级	信用卡产品联机交易异常调整日志级别
4	信用卡产品应急预案	A 级	信用卡产品日切前批量处理未完成作业
5	信用卡产品应急预案	A 级	信用卡产品数据库服务器异常
6	信用卡产品应急预案	A 级	信用卡产品数据库开关切换
7	信用卡产品应急预案	A 级	信用卡产品调用服务开放切换
8	信用卡产品应急预案	A 级	信用卡产品联机信用卡某服务大范围报错
9	信用卡产品应急预案	A 级	信用卡产品异常切换连接集群
10	信用卡产品应急预案	A 级	Agent 进程异常
11	信用卡产品应急预案	A 级	重新信用卡产品应用的容器
12	信用卡产品应急预案	A 级	发卡应用修改信用卡产品数据库配置
13	信用卡产品应急预案	A 级	缓存异常
14	信用卡产品应急预案	A 级	批量控制台手工提交作业
15	信用卡产品应急预案	A 级	……

面向各类基础资源对象的应急场景，以 MySQL 举例，如下：

序号	MYSQL 数据库应急场景名称
1	自动清理作业异常
2	MySQL5.5 数据库主备不同步
3	数据库文件损坏或丢失
4	MySQL 数据库目录文件系统使用率高
5	Prometheus 监控状态异常

6	MySQL5.5 主备切换
7	存储监控数据过大导致磁盘使用率高
8	MySQL5.7 米同步备库切换
9	MVSQL5.7 异北备库计划内切换
10	MySQL 容器 CPU、内存扩容
11	MySQL 宿主机配置 SRIOV
12	MySQL 使用 CSI 存储扩容
13	分布式中间件进程状态异常
14	MySQL 数据库基于时间点恢复
15	MySQL 容器重建

灾备应急主要关注在遭遇重大灾难（如自然灾害、严重网络攻击、重大硬件故障等）后，如何恢复和重建信息系统。因此，在创建应急预案和灾备预案时，略有差异。应急预案关联各种场景，灾备切换关注整个系统或业务的切换过程。

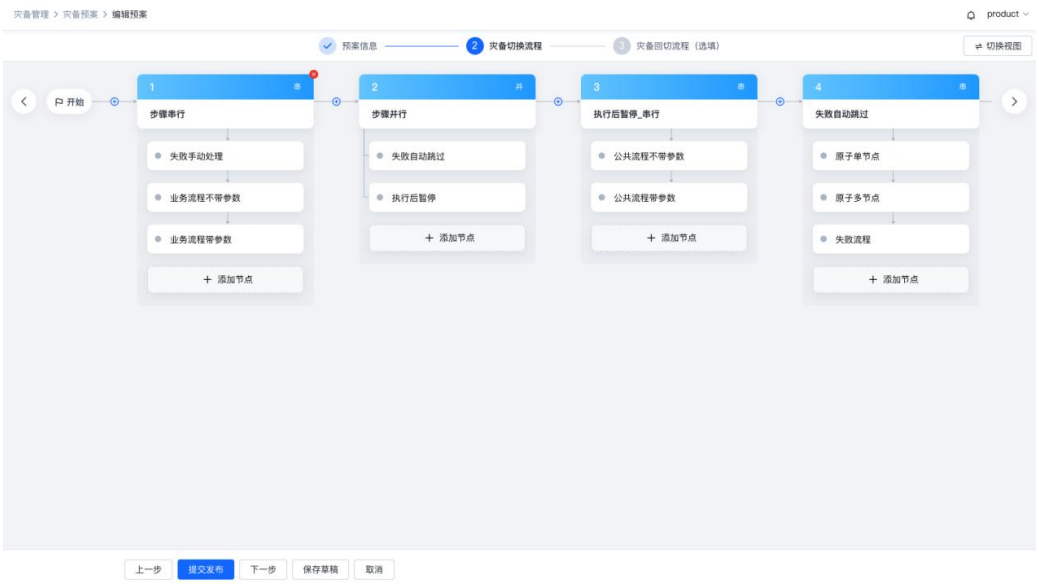
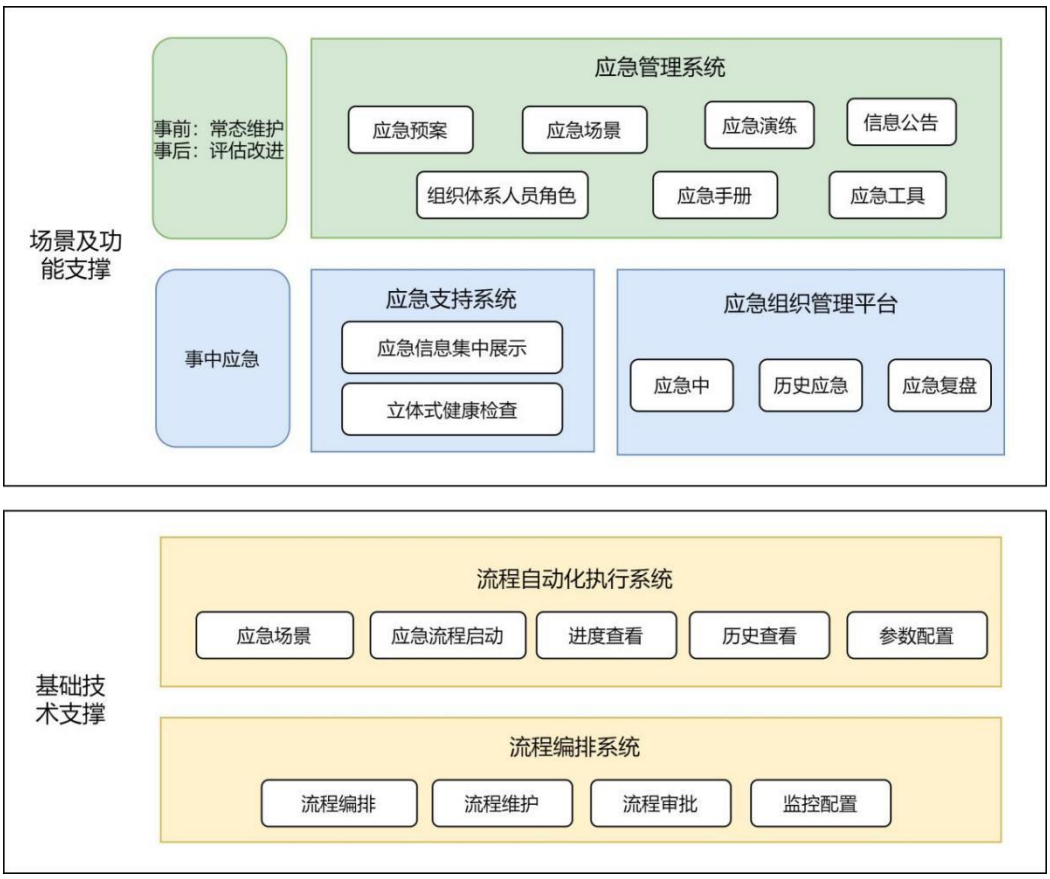


图.灾备预案

三个一体：技术一体、管理操作一体、数据一体

➢“技术平台一体”实现底层能力平台 PaaS 化；

为有效管理繁多的应急场景及应急预案、快速组织开展各类演练、高效实现不同角色、不同专业人员的线上协同，该行研发了一整套支撑应急的工具平台，用于支撑日常事件应急管理实施工作的开展。可以看出，整个平台的底层基础技术支撑实现多种能力的融合和统一调用，包含流程编排、自动化任务、监控配置等。



应急管理系统包括以下几个核心组件：

(1) 流程编排引擎，提供流程编排能力

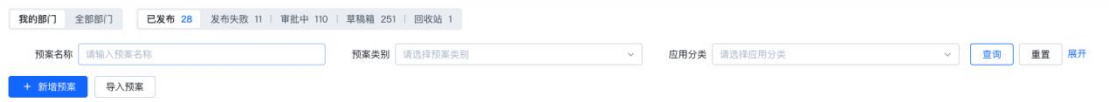
(2) 流程执行系统，提供对流程执行的启动、控制、审计等能力

(3) 事前应急管理，提供应急预案、应急场景的定义，并管理日常应急演练工作

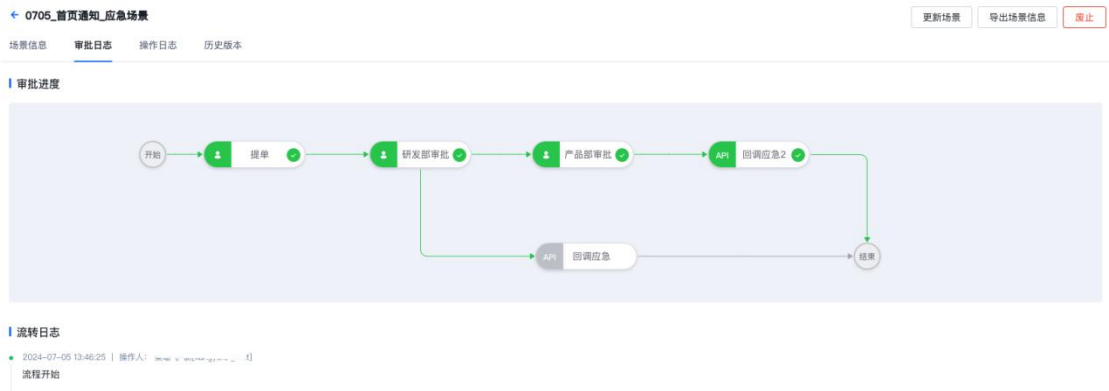
(4) 事中应急管理，提供应急启动后的组织协同、观测平面信息展示、应急故障分析决策和应急处置、应急验证和应急复盘能力，也包括应急事件的历史回溯、应急运营分析能力。

➢“管理操作一体”，实现应急管理和自动化操作的一体化管理；

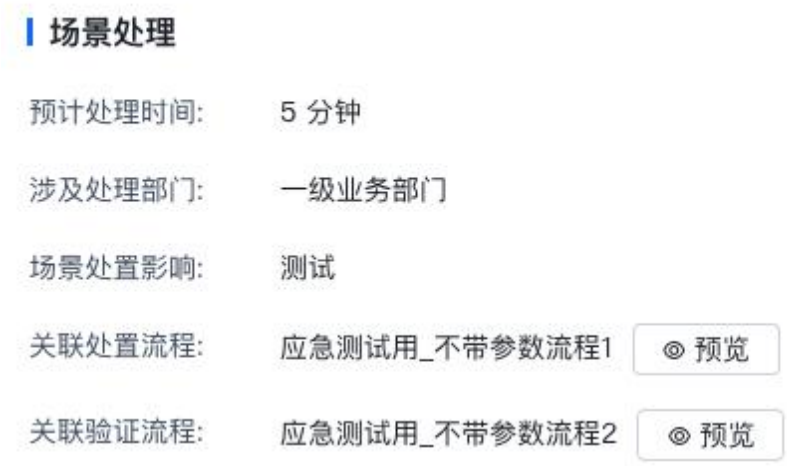
为了解决各系统应急处置组织无系统固化，事件应急预案管理不完善，对应急处置实操性不强的问题，在场景层建设了应急管理系统。通过不同 tab 状态页，管理不同组织范围下不同状态的应急预案、场景和演练计划等。



针对应急预案/场景的发布，应急演练计划的申请制定相应的审批流程，审批完成后即可生效。



针对应急场景预先制定好处置和业务验证的标准运维流程，与场景进行关联即可



灾备切换由于复杂性，需要制定好标准运维流程，并按照业务/系统切换和回切的步骤和子节点逻辑进行编排

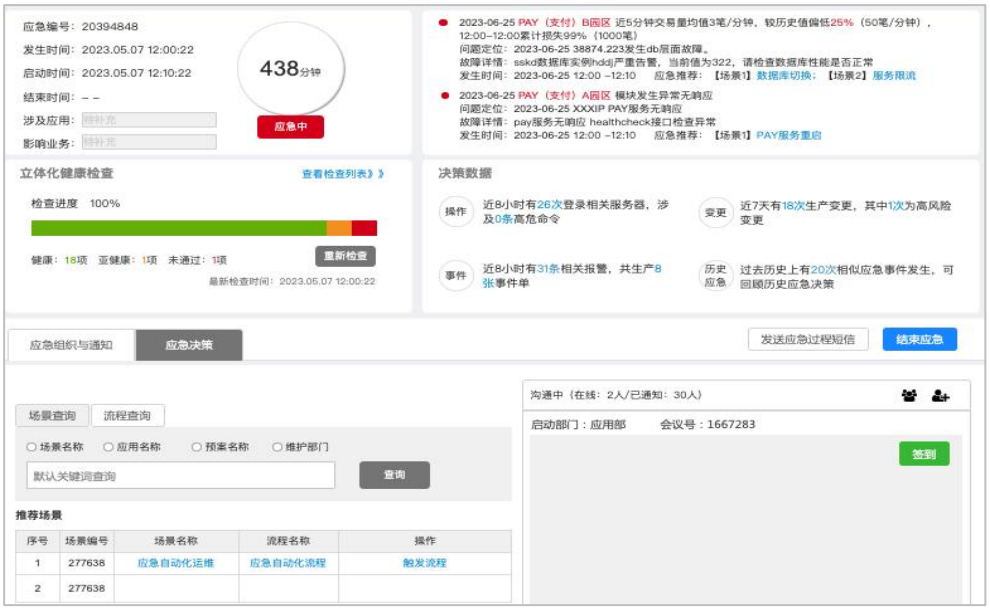


➢“数据融合一体化”，实现应急决策所需配置数据、执行数据、性能数据、变更数据的统一管理和展示。

在事中应急环节，需要一个统一的应急信息展示平台，展示以下信息：

- 故障业务的可观测链路数据（向上的故障影响分析和向下的故障溯源分析）
- 近期的变更单据数据
- 历史故障相似事件数据
- 健康检查数据
- 故障诊断决策分析数据

- 基于知识库的故障根因推荐和处置建议
- 应急小组在线协同信息



通过应急信息聚合，以便于应急组织能够更快的结合各类型数据进行处置动作的决策。

（三）总结及展望

总结

应急平台记录应急演练和应急处置的任务记录，可详细的查看每个任务的执行结果与详情。



围绕应急场景进行的演练和处置都可以按照场景要素中记录的所属业务进行应急演练和应急处置的运营统计。



通过图表，可以看出哪些应用组织相关应急演练活动较多，代表着该业务应急人员的应急能力掌握程度越高；哪些应用应急处置的次数越多，说明该业务系统稳定性较差，且事后复盘没有进行彻底。



畅想展望

展望未来，在大模型和 AIOps 人工智能的能力加持下，真正实现根因快速定位，组织的应急能力能够更加直观地通过运营数据计算得出，从而进一步提升不同部门和专业间的协作和配合能力，以应对更复杂的问题和事件，使整个数据中心的应急能力更上一层楼，为业务提供更好的保驾护航能力。

5.3.5 美图故障管理体系搭建实践

SRE Elite 精选原因

美图在这个案例的分享中，展示了非常完整的 SRE 体系及故障管理体系，以故障生命周期管理为核心，引入了由人员、流程、技术和愿景构成的“PPTV 框架”；并强调数据驱动的决策，倡导定期复盘和持续改进，通过构建稳定性运营平台，实现对故障事前、事中及事后的全方位管理，全面且扎实，很值得大家进行研读。

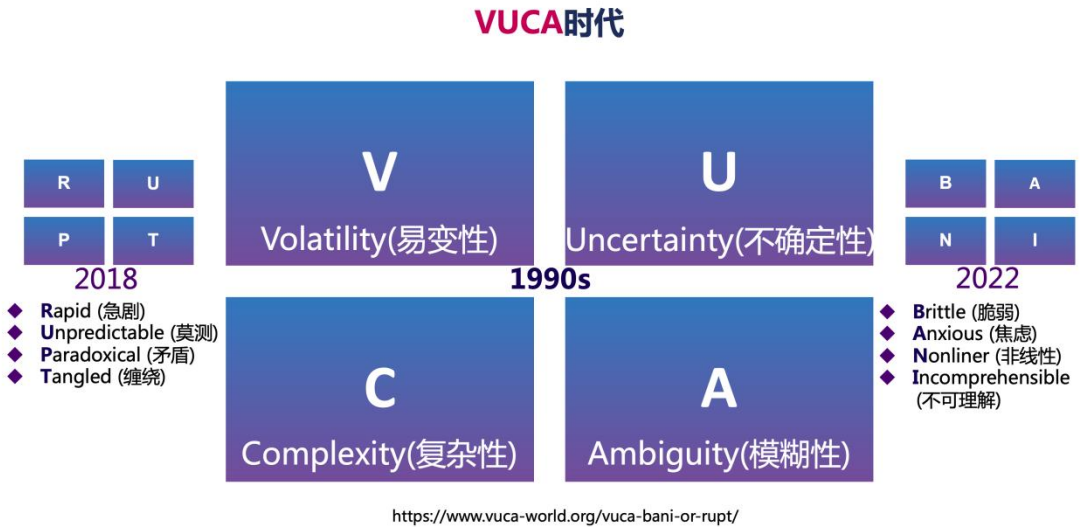
(一) 背景及设计原则

案例说明：

本文由线下 MeetUp 分享的材料整理而来，因此在行文结构上跟常规的文章组织形式会有些差异。本文开篇先介绍了外部大环境的 VUCA(BANI)特性，作为引子，进而落脚到 SRE 的岗位来讲解 SRE 的岗位职责以及所面临的挑战。而后，针对「稳定性保障」的部分展开讲解，最终聚焦在「故障管理」相关的内容上。为了保证完整性，引子的部分虽略有删减，但整体还是保留的，各位读者在阅读的时候，按需索取即可。

此外，因为「故障管理 | 应急响应」在稳定性保障的工作中，类似于一个用于检验组织稳定性建设水平的考场，在上考场之前正常都是需要做大量的准备工作的，针对开展这些准备工作的框架和方法，在分享中也做了扩展、归纳，为了行文方便，对内容的先后顺序做了调整，将文章整体结构调整为总分、层层递进的形式。

引子：VUCA 时代 与 稳定性保障



VUCA 这个词起源于 20 世纪 90 年代的美国军方，指的是在冷战结束后出现的多边世界，其特征比以往任何时候都更加「复杂」以及「不确定」；因为这个特征是具有比较强的「普遍性的」，后来这个词的应用范围就越来越广泛。VUCA 中的几个特点在我们的 IT 环境、在稳定性保障过程中也无处不在。

甚至，我们可以说：“我们在做的稳定性保障工作，其实就是要和 VUCA 做对抗”。即在一个复杂和不确定的环境下，如何去追求一个确定性、稳定性的结果——保障服务安全稳定地运行。

那我们具体需要怎么做呢？

0，首先在心态上要保持开放和接纳，接受现实的 VUCA 特性；

1，理解和运用稳定性保障工作中，可能涉及的各种「因素」和「规律」；

2，为可能发生的各种情况尽可能地做好准备和预案；

3，做好稳定性的规划和设计，构建一个符合企业实际情况的整体框架，并持续更新迭代；

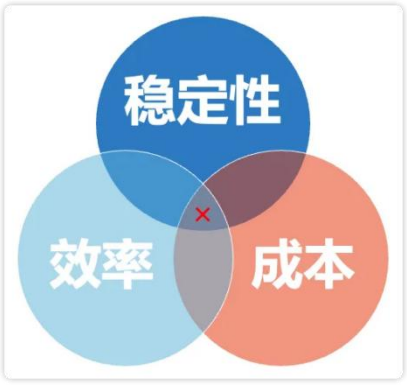
SRE 的岗位职责 与 面临的挑战

我们在公司内部是这么来定义 SRE 岗位的核心职责的，即：

- 保障线上服务的稳定性
- 建设工具、平台、基础设施来提升效率
- 用技术手段来控制、优化服务的运行成本

归结起来，其实就是几个关键字：稳定性、效率(包括职能支撑)、成本。其中「稳定性」是一个基础，也是 SRE 中的这个 R 的内涵。

当然，这三个职责也不仅仅是 SRE 或者运维同学需要承担的，诸多其他的岗位也有类似的职责目标。

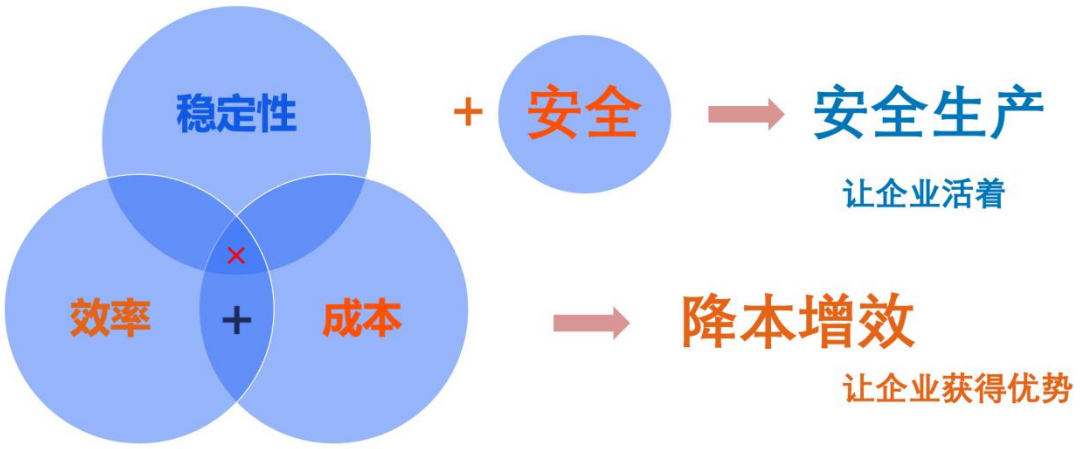


SRE 核心职责的不可能三角

愿望很美好，现实很“骨感”。因为如果你去细看，会发现这三个职责(或者称为目标)其实是一个「不可能三角」，我们无法同时在这三个方向上做到最优。

我们要做的、能做的事情其实是在这三者之间寻求一个平衡，一个动态的平衡。因为在公司、业务的不同发展阶段，在不同的场景下，我们的侧重点会有所不同。

那这三个核心职责，跟企业发展之间的关系又是什么呢？



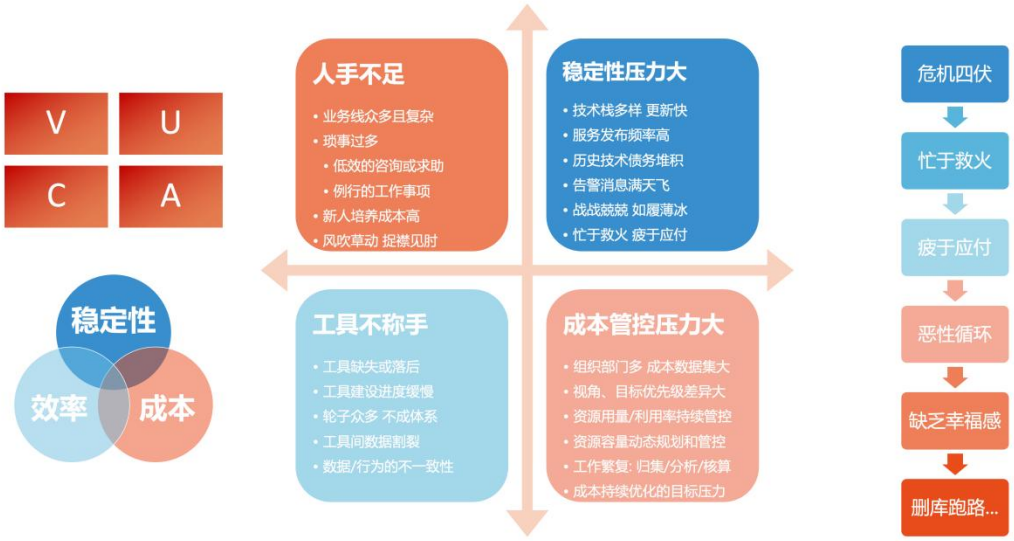
在我的理解下是这样的：

- 稳定性+安全 = 安全生产，即质量方向
- 效率+成本 = 降本增效，即效能方向

「安全生产」和「降本增效」这两者的主要目标分别是“让企业活着”和“让企业获得/保持优势”。

因为安全生产出问题，导致企业蒙受巨大损失甚至是倒闭的案例也有不少，这里我们就不做详细的展开。如果在相当的产品功能、质量、性能下，一个企业可以把单位成本控制得更低、可以有更高的交付效率，那么它就可以获得更高的竞争优势。

当然，这些事项或目标也是相辅相成的，这里我们只是探讨权重占比更高的部分。比如，如果一个企业的生产成本居高不下，在激烈的行业竞争中可能就会缺少或没有优势，最终也可能会影响到企业的生存问题。



SRE 的挑战和困境

那具体到 SRE 这个岗位，其面临的困境都有哪些呢？

除了前文提到的因几个岗位核心职责之间的相互制约所带来的困难之外，我总结了以下 4 个方面的困境：

1.人手不足

在很多公司内部都会有“研发：测试比”、“研发：运维比”这样的数据，通常情况下运维/SRE 的占比不会太高。这里我们取另外的一个指标，“SRE 人数：公司全员人数”，通常情况下这个数值不会超过 1%，甚至不会超过 0.5%。也就是说在一个 1000 人左右的中小型企业中，SRE 的人数通常不会超过 10 个人，甚至不到 5 个人。

然而，随着公司业务的发展、持续地探索，公司的业务线会越来越多，并且日益复杂。同时 SRE 岗位还需要处理非常多的琐事，比如各种信息问询、问题排查等求助，还有很多例行的常规事项，比如服务巡检、OnCall 值守、响应&处理线上告警或故障、编写各种文档等等。最终会导致 SRE 的负载非常高，难以实现 Google 对 SRE 岗位的时间花费的建议：即有 50%的时间来做工程建设类的工作。

还有一个需要关注的点，考虑到 SRE 的岗位特点，需要对线上服务的稳定性负责，是一个责任非常重的角色，培养一个新同学的成本是非常高的。一位新同学要想顺利接手线上服务的稳定性保障工作，除了必备的基础技能，还有诸多的背景知识或信息需要掌握，一般都需要经过一段时间来建立真实的体感才能完全进入状态。

在这样的背景之下，SRE 的人手时常会比较窘迫，尤其是遇到一些人员的变动，会更加捉襟见肘。

2.稳定性压力大

随着行业和技术的快速发展，SRE 所面对的工作环境和管理对象也发生着巨大的变化。所管理的服务架构从单体到分层、分布式，再到微服务、云原生，一路快速甚至跨越式发展。这时候 SRE 就会面临技术栈多样且更新迭代快的现状，这些持续攀升的复杂度和急剧的变化都会对稳定性带来巨大压力。

此外，企业为了快速响应市场，服务发布、变更的频率也是越来越高，由此引发的稳定性问题也是层出不穷。与此同时，在业务快速增长阶段，为了快速地交付产品功能，在一些质量、稳定性方面的投入就会有所忽略或舍弃，最终会导致相关的技术债务不断堆积，进而给稳定性带来更大的威胁。

以上的几个问题，如果不能及时关注和处理，就会导致 SRE 同学每天面临着爆炸式的告警消息，忙于救火、疲于应付，也会导致 SRE 等同学对线上服务(操作)的畏惧，战战兢兢，如临深渊、如履薄冰。

3.工具不称手

与前两个点相对应的就是 SRE 在工具规划和建设上的投入了，因为人手不足、线上服务的稳定性压力大、日常工作负载高，就会导致在工具建设方面缺乏足够的投入，或者是为了快速地解决某些

问题引入或建设了众多的轮子，这些工具(轮子)集缺乏顶层的规划和设计。

最终形成的局面可能就是：

- 工具的缺失或者没有做及时的更新适配
- 工具建设的进度缓慢，功能一再延期，无法交付
- 引入或制造了众多的轮子，各个轮子尺寸不一、不成体系，无法形成很好的合力
- 工具之间的数据相互割裂，每块数据都不够完整和新鲜，导致工具的易用性、可用性不足
- 严重的情况下可能会因为多个工具之间的数据、行为不一致带来灾难性的后果

以上 3 点，人手不足、稳定性压力大、工具不称手，也是存在相互依存、相互促进、相互制约的关系，如果协调不当可能就会进入上图所示的“死亡螺旋”：危机四伏 → 忙于救火 → 疲于应付 → 恶性循环 → 缺乏幸福感 → 删库跑路(开个玩笑，但愿不会这样)。

此外，还有第 4 点，成本管控相关的压力。

4.成本管控压力大

随着企业数字化转型的持续深化，IT 成本在企业成本支出中的占比也越来越高。在“降本增效”的大背景下，企业对于 IT 成本的管控和优化的诉求也就愈发迫切。

因为 SRE 岗位需要对业务的 IT 资源交付负责(这个点可能会因公司而异)，因此也就自然需要承担对 IT 资源管控的职责。比如对业务的资源需求合理性做评估，对资源用量、资源利用率、资源容量做管控，还需要对业务的 IT 成本数据去做归集、统计、核算分析，最后还需要识别出资源使用的不合理之处，并推动优化。

成本管控工作的难点有：

① 工作内容是非常繁杂的，琐碎且注重细节；

② 这是一个长期工作，需要周期性持续跟进和投入；

③ 很多工作是需要多个部门、组织来相互协作才能达成的，会涉及跨部门沟通中的一些阻力或困难。

以上就是我司 SRE 岗位的核心职责，以及我所理解和归纳的 SRE 的挑战和困境。

稳定性运营的一些实施准则

要走出上述困境，我们是需要多管齐下，协同发力的。聚焦于 SRE 最核心的岗位价值，「稳定性」相关工作的优先级需要提高。为了确保系统的稳定性运营，建议 SRE 团队遵循以下几项准则：

1. 宏观框架与理论支撑：

在进行任何操作时，必须有一个宏观的框架和理论指导。这并不是要我们拘泥于条文或产生教条主义，而是这些理论和框架是经过大量实践总结出来的宝贵经验，能够帮助我们规避许多弯路。因

此，理论支撑是非常重要的，当然，也需要我们结合实际情况灵活应用。

2.数据思维与度量先行：

所有决策和操作都必须有数据支撑。只有通过足够的数据，才能做出科学的决策。比如，在应急响应时，需要依赖监控数据和大盘，了解系统状况和问题严重程度。因此，数据思维和度量体系的建设是关键。

3.定期复盘与流程闭环：

定期的复盘和持续改进是稳定性运营中的重要环节。通过复盘，发现问题，总结经验，改进流程，形成闭环管理，从而不断提升系统的稳定性。

4.紧扣价值与持续输出：

所有的工作都要明确其核心价值所在，尤其是要提升岗位的核心价值，如稳定性、效率、成本和安全。任何行动都应该有明确的目标感和价值感，这样才能确保工作的高效性和针对性。

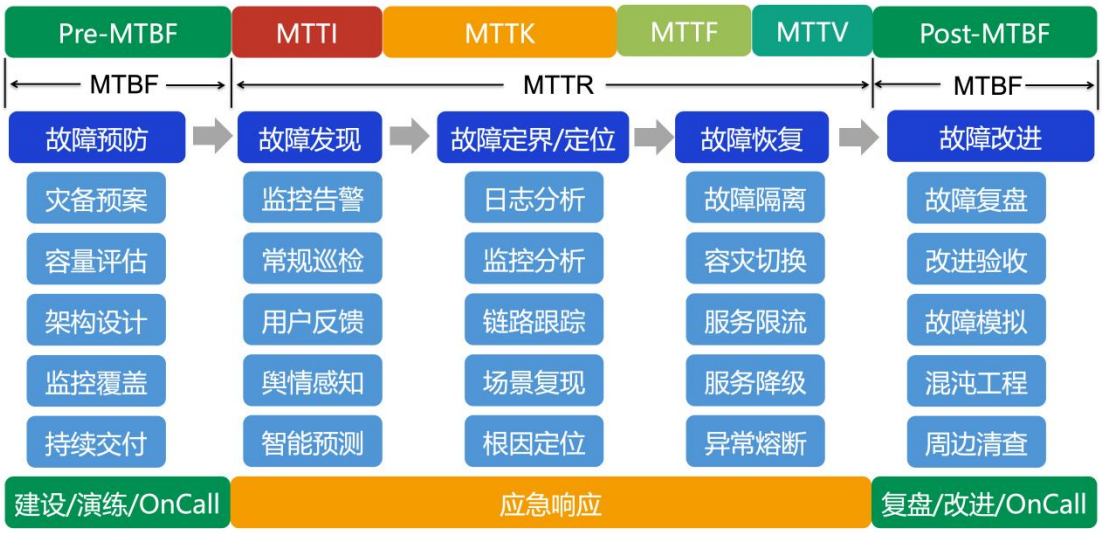
在推进稳定性建设的过程中，遇到职责划分(模糊)问题时，要主动出击，推动正确的事情取得进展，而不是因边界条件限制而放弃。这不仅是提升系统稳定性的关键，也是个人成长的重要途径。同时，这也是一名称职 SRE 的必备特质--「责任心」的体现。

通过以上准则，SRE 团队可以更好地保障系统的稳定性，实现高质量的运维管理。

(二) 体系设计详情

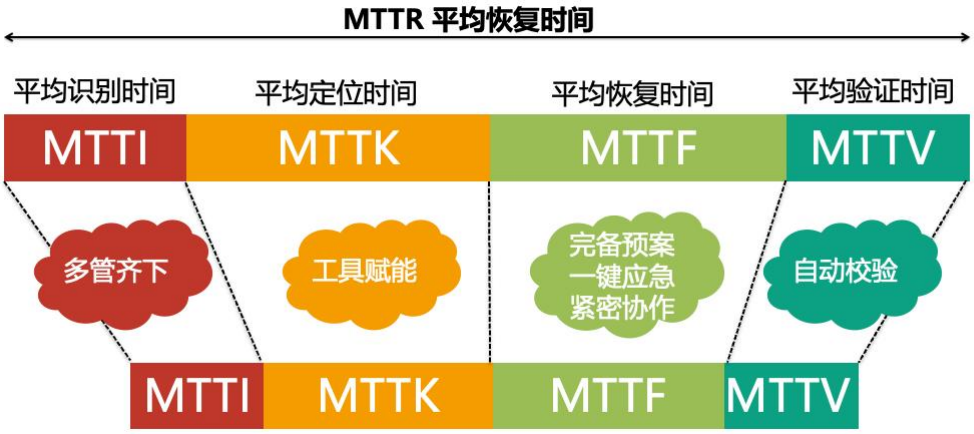
一、SRE 体系建设

1、稳定性建设全景设计



这张图应该是国内 SRE 同学比较熟悉的，即基于“故障生命周期管理”的思路来盘点和审视需要做的稳定性建设工作，核心目标是保障和提升服务的稳定性，框架中涉及的内容还是非常丰富的。

我们稳定性保障的目标，就是要尽可能缩短 MTTR，进而提高 MTBF。其中 MTTR 的指标是可以被细化拆解的，在细化之后，我们就可以分而治之、各个击破了。



我们可用的手段，总结如上图：

● MTTI 阶段：多管齐下；尽可能用更多的手段来覆盖可以发现、暴露问题的通道，包括完善监控点位的覆盖，建设体系化的观测能力和系统。

● MTTK 阶段：工具赋能；这个过程中需要更多地借助工具、平台的能力，建设健全运维系统，比如监控平台、日志平台、链路跟踪平台等，如果能力允许也可以去尝试建设“根因自动诊断定位”系统。

● MTTF 阶段：完备预案、一键应急、紧密协作；这个是在故障处理过程中最核心的部分，是真正可以让服务恢复正常的阶段；这个过程有比较多的前置依赖，也就是需要我们在平时做好储备的，比如建立健全预案体系，将预案的执行手段尽可能沉淀到工具平台中，应急动作尽可能做到一键直达，缩短预案执行的路径。其次，这个过程中可能还会涉及部门内部、部门之间的协作，尤其是在处理重大故障的场景中；这时候就需要有一套可以

让大家紧密协作的流程或共识，让大家可以信息互通、各司其职、有条不紊、高效协同。

● MTTV 阶段：自动校验；这个过程也是需要尽可能依赖自动化的手段去实现，相关能力通常可以内置在「可观测相关平台」或「故障快恢系统」中。

关于持续交付能力的重要性

「持续交付」能力这个部分，我要稍微展开讲一下，美图公司目前在大力推进 AIGC 相关的业务，在 AIGC 类服务的保障工作中，持续交付能力显得尤为重要。以下是一些具体原因及相关细节的简要说明：

首先，AIGC 类业务具有传播速度快，容易形成热点、业务流量增速迅猛等特点，对资源的用量、性能、弹性效率都有非常高的要求。

其次，AIGC 高度依赖 GPU 算力资源(虽然也有其他解决方案，但 GPU 目前还是更通用的资源)，而 GPU 服务器的初始化时间远长于常规的 CPU 业务场景。初始化过程包括多个阶段：机器初始化、GPU 卡的初始化、驱动加载、容器镜像拉取、算法模型加载等，这些步骤复杂且耗时。有些场景下的算法模型和容器镜像体积可达到几十 GB，这将进一步增加网络和存储的吞吐压力，进而增加资源交付的耗时。相比于传统纯 CPU 算力场景下的秒级容器扩容，

在当前的技术条件下，即便做了极致性能优化，GPU 算力资源也很难实现跟纯 CPU 同样的交付效率。

同时，GPU 资源是非常昂贵和稀缺的，尤其是受限于多种因素，相关资源非唾手可得，且企业通常较难容忍大量 GPU 资源的长期闲置，因此需要提前规划和准备。

我们目前采用多云战略，涵盖了国内主流云厂商和线下机房资源，并利用统一的「容器管理平台」整合纳管这些资源，可以在很大程度上解决仅依靠单一厂商无法满足资源需求的问题。不过，在资源调度过程中，也会涉及许多细节问题，例如：

- 不同 AI 算法所适用的 GPU 卡类型不同；
- 不同 AI 算法所依赖的周边资源(如网络、存储等)、周边服务会有差异；
- 同一个 AI 算法，在不同的 GPU 卡上的性能表现、性价比可能会有巨大差异；
- AI 算法/业务玩法，迭代速度快、丰富多变；
- 不同厂商之间的基础设施会存在差异；
- 在成本管控的压力下，我们还要保证算力成本最优；

这些现实细节都对我们的资源交付平台(容器管理平台)提出了高要求，必须具备强大的能力来支持高效的资源调度和优化，这也是我们近期重点建设的方向。

当然，除了上述 AIGC 的业务场景，在常规业务的稳定性保障过程中，「持续交付」能力也非常关键。比如在大量请求短时涌入的应对场景，在服务单点(服务组、AZ、Region 等)异常需要做快速容灾切换的场景等。

2、PPTV 框架



“PPT”应该是大家相对熟悉的一个概念，包含了人员、流程和技术，这是 ITIL 中的 IT 管理三要素，SRE 体系建设同样需要依赖这个模型。要做好故障管理，必须有适当的组织结构和流程保障；仅有组织和流程还不够，还需要强有力的技术支撑。

在此基础上，我增加了一个词，Vision（愿景、目标），与 PPT 结合，构成一个“PPTV”的框架。只有在稳定性建设上有共同的目标期许，才能做好 SRE。不同的参与部门应对服务稳定性有一致的

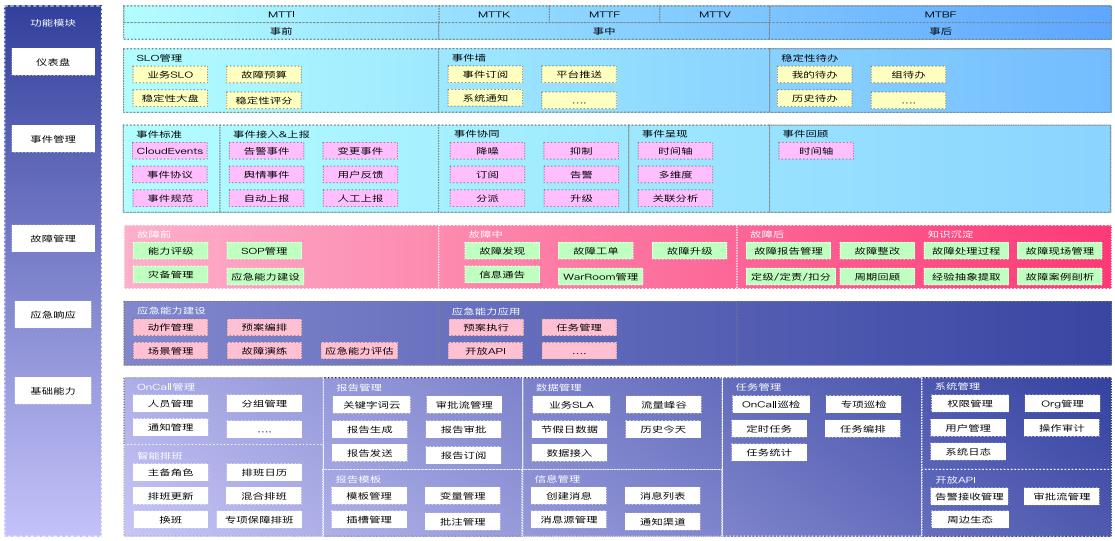
愿景，不论是出于行政压力还是自身对服务稳定性有追求，大家都需要有统一的目标。

“PPTV”这个框架模型，涵盖了目标对齐、组织流程保障和技术支撑等几个重要方面，是一个相对完整的框架，在做 SRE 体系落地的过程中可作为参考。

3、稳定性运营平台

除了上述大家可以用来参考规划工作内容的「稳定性建设全景图」、用于推进稳定性建设工作的宏观框架「PPTV」，我们还需要建设一个可靠的平台来承载和实现相关能力。

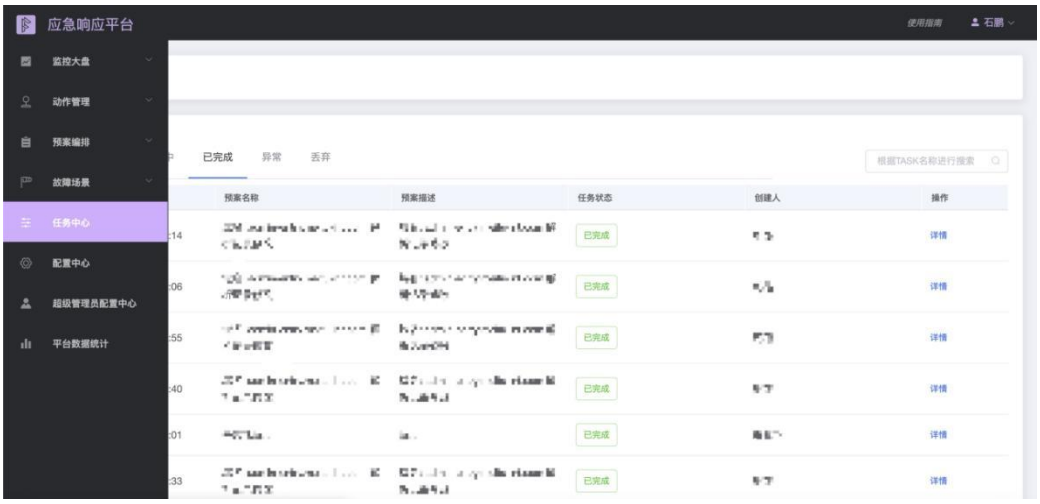
我们团队目前建设了一个「稳定性运营平台」，主要实现了对一些日常稳定性运营工作的管理，包含 OnCall 轮值、自动化巡检、事件管理、稳定性运营报告、业务关键指标数据管理、历史数据分析等；还有一个独立的「应急响应平台」，主要覆盖针对线上服务异常、故障的应急处理。目前我们正在将这两个平台做功能合并和刷新，下面是规划和实现中的「新版稳定性运营平台」的主体框架。

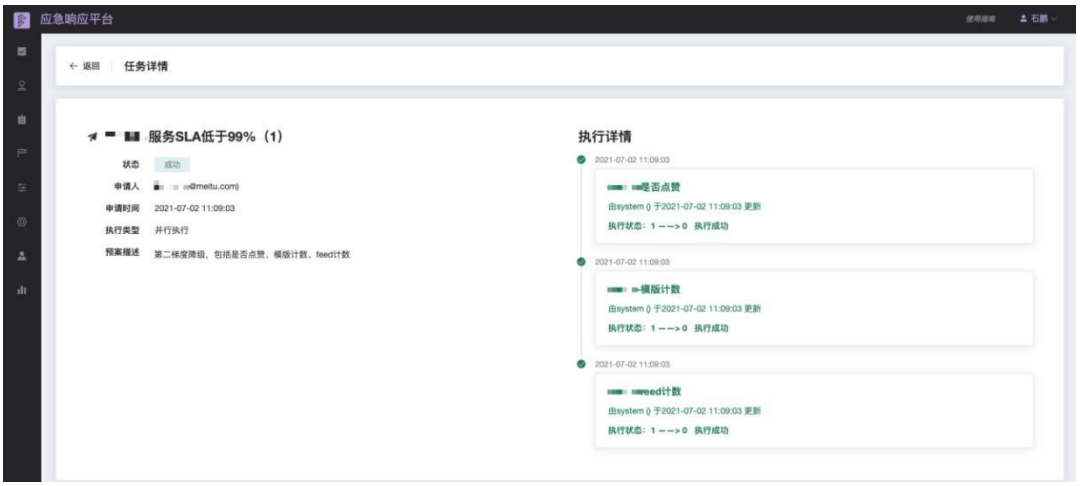


平台的主体功能按照故障生命周期的三大阶段进行拆解设计，以期覆盖故障事前、事中及事后的完整过程。整个平台的功能模块可分为三个层：由仪表盘构成的「呈现层」、由事件管理、故障管理、应急响应构成的「核心能力层」，以及平台的「基础能力层」。

我们期望以此平台作为 SRE 日常工作的底座，完成稳定性运营、应急响应、稳定性建设工作推进等核心工作。

针对故障应急的部分，我们现有的「应急响应平台」已经实现了必要需求的覆盖，在此简单讲解。







平台的核心逻辑是：

①将对服务干预的手段抽象为可执行的原子「动作」，如脚本或 HTTP 接口调用等；

②将这些原子动作进行组合编排，形成有序的应急处理「预案」；预案中的动作执行可以是带有依赖关系的，支持串行执行、并行执行、逻辑判断等；

③将预案绑定到预设的各个故障「场景」中，一个场景中可以绑定多个/多级预案；

当故障发生的时候，如果在对应的场景下有匹配的预案，我们就可以快速执行以实现止损；如果没有现成的预案，但是可以通过临时组合已有动作来覆盖的场景，也可以在平台上快速完成编排和执行的操作。

二、故障管理体系

通过前面的段落我们了解了 SRE 体系建设的主体内容，下面我们聚焦在「故障管理」的部分，聊一聊如何搭建故障管理体系。



我们前面一直在提“故障的生命周期”，我们的故障管理体系也要围绕故障的生命周期来展开：服务稳定运行、故障发生、故障

恢复、故障复盘、故障改进、改进验收，这个过程会是周而复始的一个「环」，我们服务的稳定性就会在这一个又一个「环」的迭代下变得“更健壮”或者“更脆弱”。

为了做好故障管理工作，我们需要从指标体系、流程规范、考核体系、组织支撑等几个方面展开建设。

1、可用性体系



上述所提及的即为标准化的 SLIs/SLO/SLA 体系，该体系融入了 VALET 方法，通过五个核心维度——容量（V）、可用率（A）、延迟（L）、错误（E），以及人工提单与保底分类（T）——来系统地梳理 SLI。这一步骤是构建系统可用性的基石，若无此体系，则无法有效地衡量系统可用性，后续的工作亦将缺乏坚实的基础。

2、故障定级、定性、定责

我们对故障的判定，通常包含 3 个方面：定级、定性、定责。

- 定级：故障的严重程度；
- 定性：故障属于什么类型；
- 定责：故障的责任要如何划分；

这几个方面的判定，都依赖于我们在组织内部建立标准规范、达成共识、形成制度。

1）定级：通用标准

美图故障等级衡量指标说明			
故障对服务功能的影响	权重占比xx%	备注	级别分值(阶梯式)
1级	主要功能完全不可用		100
2级	主要功能部分不可用		75
3级	次要功能完全 不可用		50
4级	次要功能部分不可用		25
故障影响时长	权重占比xx%	备注	级别分值(阶梯式)
1级	xx分钟以上		100
2级	xx分钟~xx分钟		75
3级	xx分钟~xx分钟		50
4级	xx分钟~xx分钟		25
故障发生所处时段	权重占比xx%	备注	级别分值(阶梯式)
1级	最活跃时段		100
2级	次活跃时段		75
3级	非活跃时段		50
4级	最不活跃时段		25
对用户的影响范围	权重占比xx%	备注	级别分值(阶梯式)
1级	影响用户占比xx%以上		100
2级	影响用户占比xx%~xx%		75
3级	影响用户占比xx%~xx%		50
4级	影响用户占比xx%以下		25

针对故障定级，建议形成“可以量化”的规则。

上图是美图定义的故障等级的衡量指标说明，图中列出四点都是常规的考核项，主要看对服务功能的影响、影响时长、故障所发生的时段、对用户的影响范围。

我们会根据相关考核项的具体影响程度进行量化打分，最终加权计算得出一个故障评分，并对故障进行定级（不同分数对应不同级别，具体可参考下文「故障预算」的段落）。

2）定级：个性化标准



除了通用定级标准中的四个考核项，在一些业务场景下可能还会有其他的考核项。比如某些电商类、商业化、广告类或金融类的业务，会重点考核资产损失；比如有些社区类应用，可能会重点考核舆情、公关事件等。因此，不同的业务部门会有不同的故障定级策略。

因为这些个性化故障定级的诉求是由业务真实的特性所决定的，因此是合理地存在，并且是无法直接简单粗暴地用通用规则所覆盖的。不过，为了维持公司全局的故障管理工作的公平、公正，

我们需要把所有业务部门的故障定级标准拉齐。我们的做法是跟有个性化故障定级诉求的业务部门做定级标准的映射，各业务部门对故障的考核项、考核项的权重可以不同，但是故障级别的定义需要跟通用规则保持一致，并且定级规则需要公示并通过「故障管理委员会」的审议。

3）定性：有效分类

除了对故障定级，我们还需要对故障做定性的分析和归类，由此可以发现一些故障的共性、特点或规律，并指导我们的稳定性建设工作。以下是美图的部分分类示例：

- 代码质量
- 测试质量
- 流程规范
- 变更操作
- 容量规划
- 产品逻辑
- 硬件设备
- 局方故障
- 云厂故障
- 第三方
- ……

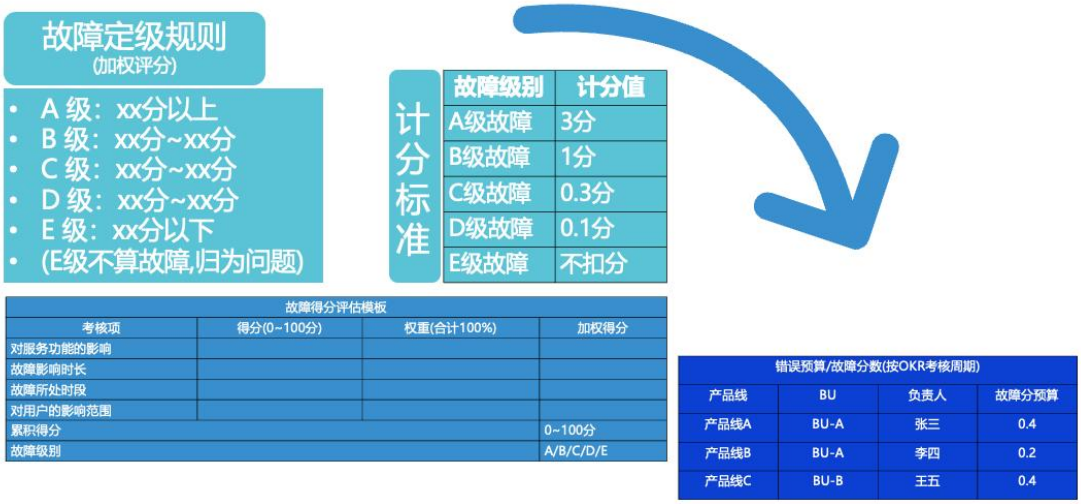
4）定责：判定原则

定责任并不意味着处罚，更多的是要承担故障整改的责任。

我们整体推崇和践行的故障文化是“不指责”，但不指责不代表着可以持续犯错误，尤其是一些低级的错误。我们在做故障定责的时候，有一些比较重要的判定原则，简述如下：

- 高压线原则：一些企业内部的红线(比如数据安全)不能触碰；不能犯一些低级的错误；不能持续犯某些错误。
- 上下游原则/健壮性原则：需要根据服务的上下游依赖集及健壮性判定。每个部件自身要具备一定的自愈能力，比如主备、集群、限流、降级和重试等；对依赖的管理方面，原则上核心应用对非核心应用的依赖必须有降级和限流手段。
- 第三方默认无责：主要指的对内做故障判定时，默认第三方无责；稳定性责任一定是内部角色承担。这条规则是为了避免各种问题都甩给第三方、云厂，久而久之会失去责任心。当然，对于第三方该追责、索赔的还是要正常进行。
- 分段判定原则：一个故障的链路可能会比较长，原因也可能不止一个。因此我们要分段做分析，这样更有利于让故障问题更聚焦，改进措施也会更有针对性。
- 自由裁量原则：具体问题具体分析，Case by Case。同时针对一些边界定义尚模糊的场景，要灵活把握、留有合理的空间。

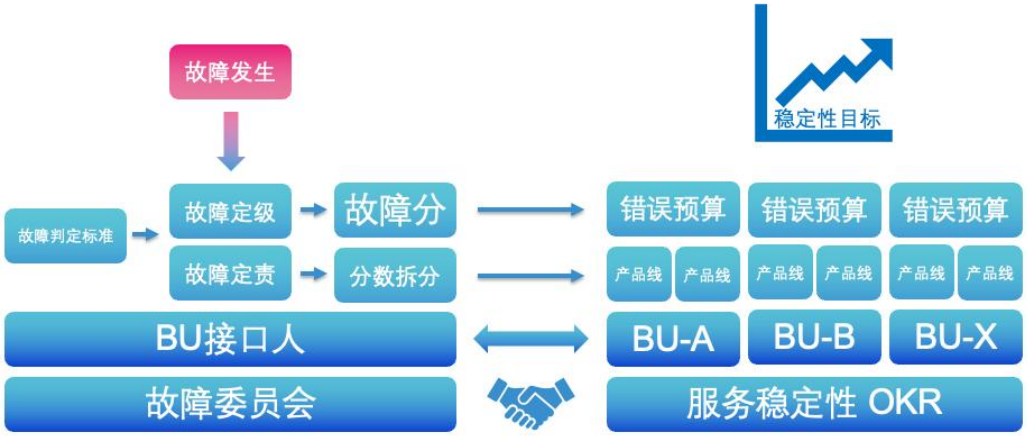
3、错误预算



我们在 SRE 相关的工作中会经常听到「错误预算」这个词，那故障预算的机制要如何落地呢？在不同的企业里会有不同的实践思路，下面我简单讲一下我们的做法。

根据前面讲的故障定级规则，不同的故障会基于最终的故障评分被定为 ABCDE 几个级别，不同的故障级别会被扣除不同的分数，这个用于抵扣的分数则是「故障预算」。我们每年为一个 OKR 考核周期(每季度、半年会做期中/期末的核算)，在一个考核周期里面每个 BU 会被分配一定的“故障分”，如果发生故障，则要根据规则抵扣相应的分数，如果故障分被扣完了则意味着错误预算耗尽，会对应着相关的限制策略。同 Google SRE 的做法类似，即可用性达到多少标准，发布将受到限制。我们将错误预算和 OKR 做了绑定，让大家在目标驱动之下达成这个目标。

4、组织结构支撑



故障管理特别依赖于组织的支撑，因为故障管理不仅仅是运营部门或者技术保障部门能单独完成的事情，需要拉通不同的团队进行协作。为此，我们成立了故障管理委员会。故障委员会是一个虚拟的组织，各 BU 指定接口人参与故障管理委员会，或参与故障管理委员会的一些日常工作，比如参与对故障判定的讨论和审议、传达相关信息给 BU 内部。

我们可以根据上图完整串一遍这个故障管理工作的流程。当故障发生了，基于判定规则给故障定级、定责。定级意味着每个级别产生对应的故障分；定责会涉及故障的拆分，比如发生了一个故障，它由 A、B 两个部门共同承担；比如，根据定责发现责任上 A、B 部门实际是五五开，因此 A、B 平分故障分，从 A、B 部门各自的错误预算里扣除。

通过这种方式就能约束好 BU 吗？当然不是，还要辅以行政管理的手段来保障，在我们内部就是 OKR。在各 BU 的 OKR 里需要有服务稳定性的目标项才行，当把服务稳定性的目标写进 OKR 之后，各 BU 才会更有压力和动力在故障管理及稳定性保障上做投入。

三、故障管理之「三段式拆解」

根据故障发生的时间顺序，我们把故障管理分为故障前、故障中和故障后，在每个环节都有一些核心的工作内容和目标。

故障前：故障预防、灾备预案；目标是尽可能地做足准备工作，毕竟有背方可无患；

故障中：发现、定位、解决故障；目标是尽可能的提高效率，缩短故障恢复的时间；

故障后：故障复盘、故障改进；目标是尽可能多的从故障中吸取教训，反思和学习，完善和修复故障中暴露出来的问题。

1、故障前

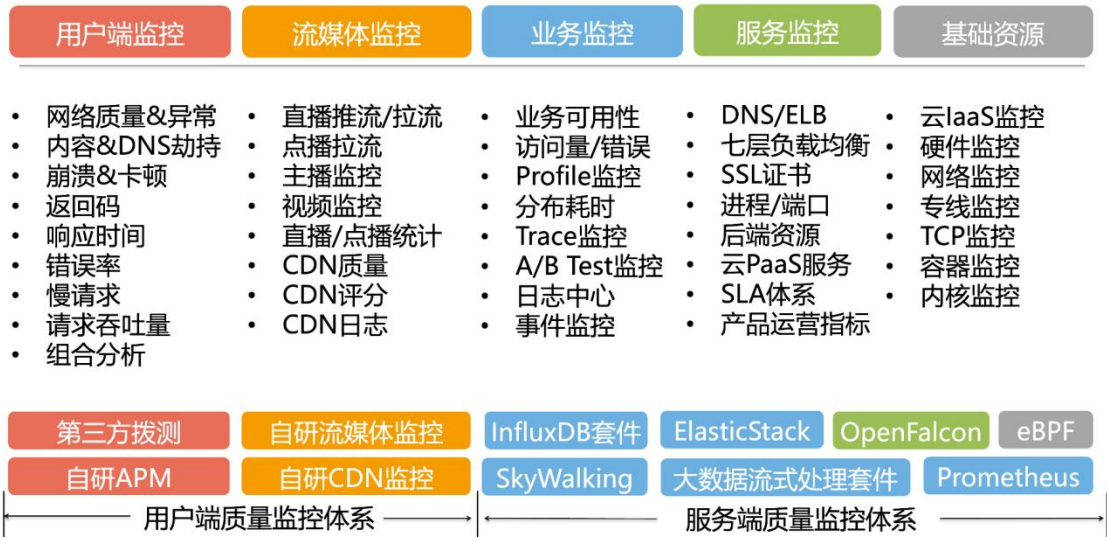
故障前的故障预防和灾备预案，我们应该怎么做呢？

这里列出了这么几个比较关键工作内容：



1）监控覆盖

监控覆盖的话比较容易理解，服务上线后，只有拥有足够的监控手段，并且尽可能多的覆盖服务的各个环节，才有可能在后面发生问题的时候，快速地感知到。也就是说，我们的目标是尽可能有更多的监控手段去覆盖我们服务，保障各个场景。下图是我们之前用到的一些监控组件：



我们把监控体系分为两块：客户端质量监控和服务端质量监控。

①客户端质量监控

我们在客户端质量监控里边去感知客户端的状态。

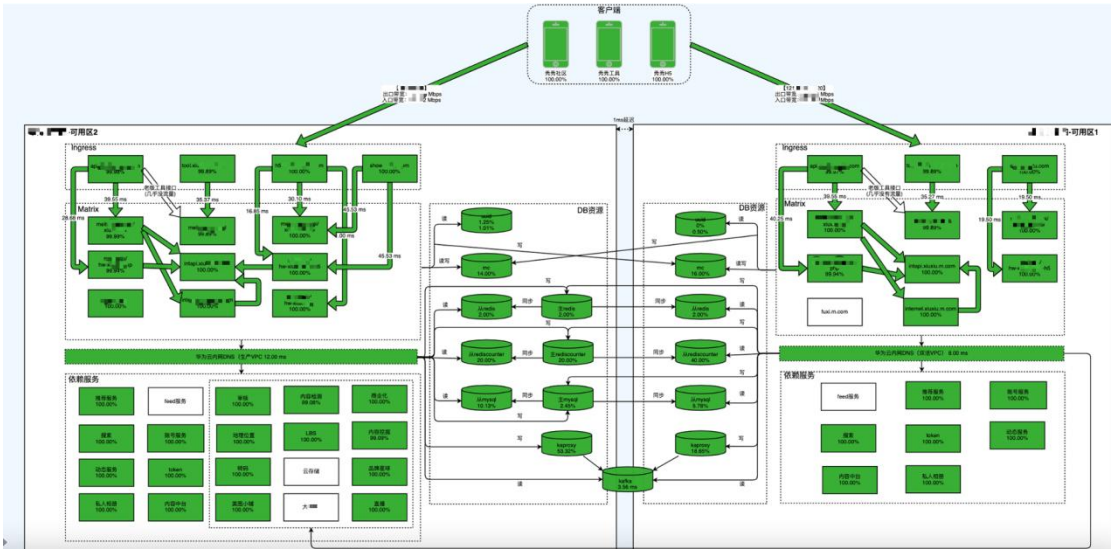
客户端质量监控我们都使用了哪些手段呢？

常规的有一些第三方的拨测、第三方的 APM。除了一些商用的产品，当然也有一些免费的工具是可以用的。

应对一些流媒体的应用，我们自研了一套融合 CDN 的监控还有流媒体的监控。这个其实也比较关键，因为现在业务体量大了之后，CDN 是不可避免地会用到，但如果没有这部分监控的话，CDN 质量就要依赖于用户反馈，链路就会比较长。所以说要有手段去主动监测 CDN 的质量。我们建设了这样的 CDN 的质量监控，然后会去给不同的 CDN 厂商去打分，再去做智能的调度。

②服务端质量监控

服务端的质量体系是用到了一些比较通用的，像 InfluxDB 套件、ELK、Prometheus、Open-Falcon，Skywalking 等等。除此之外，我们还建设了一些基于大数据流式计算的系统，主要是用来做我们 SLA 的体系。



通过可观测平台，构建了一个全景的链路拓扑下图是我们偏运维侧的一张数据流向图，主要是说明一个业务包含了哪些域名，大概经过了什么链路，经过了哪些中间组件，最终到达了服务端。但是这个拓扑偏向静态的关系构建，目前有点不是太符合现在的要求。

因此，目前在研发动态的拓扑展示。我们将服务的链路拓扑存储在图数据库中，以实现高效的全局链路信息管理。这不仅为上层业务提供支持，还在问题排查时能够快速定位和分析。数据的来源和整合过程如下：

首先，数据来源包括 CMDB（配置管理数据库）、资源层数据和业务层数据。这些数据从 CMDB 中提取并存储到图数据库中。同时，我们对机器、容器的节点（Node）、Pod 以及各种网络连接进行抓取和筛选，并将结果存入图数据库。此外，还从日志中提取相关信息，最终汇总到图数据库中。

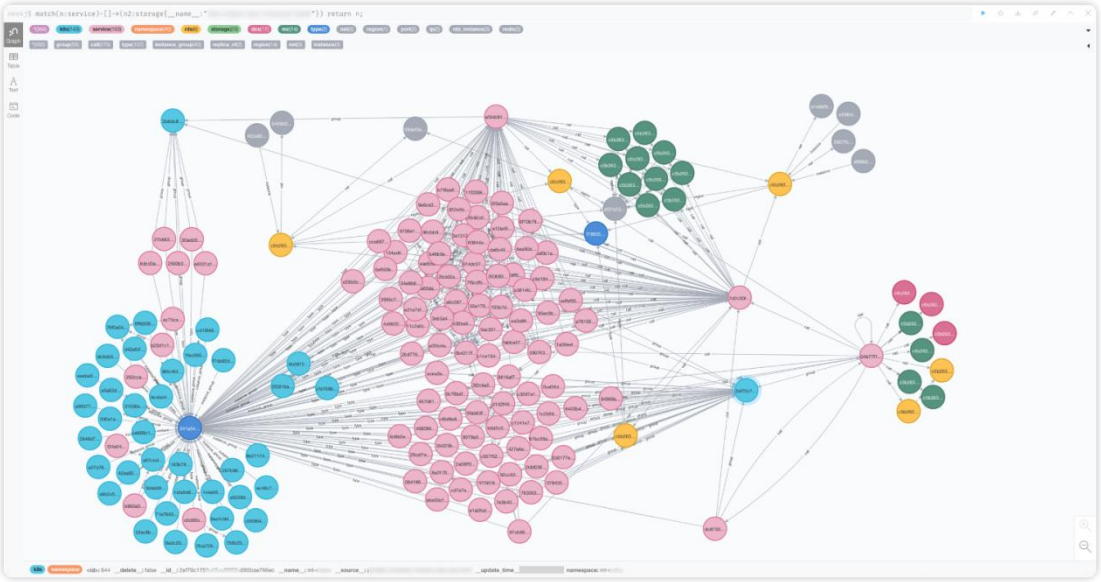
在建设过程中，需要注意以下几点：

数据多样性：链路信息复杂，单一手段无法获取全局信息，因此我们采用了多种数据来源的组合。

容器化环境：由于服务已全面容器化，扩缩频繁，数据量巨大，传统手段难以捕捉所有信息。因此，我们选择基于 eBPF（扩展的伯克利包过滤器）的网络连接数据抓取方法。这种方法尽管会产生大量数据并具有时效性，但能有效捕捉动态环境中的链路信息。

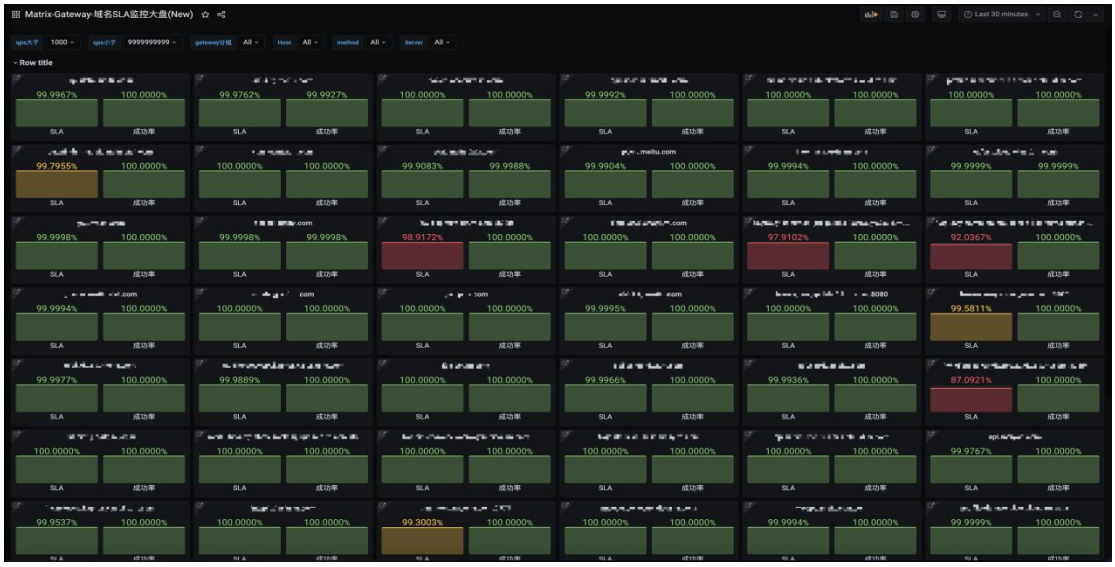
时效性管理：容器（如 Pod）的生命周期较短，可能仅存活 10 分钟。因此，在链路排查时，需要设计机制以查看特定时间段的数据，并确保过期数据不再展示。

通过这些方法，我们初步实现了对复杂链路信息的高效管理和利用，为业务支持和问题排查提供了强有力的工具。

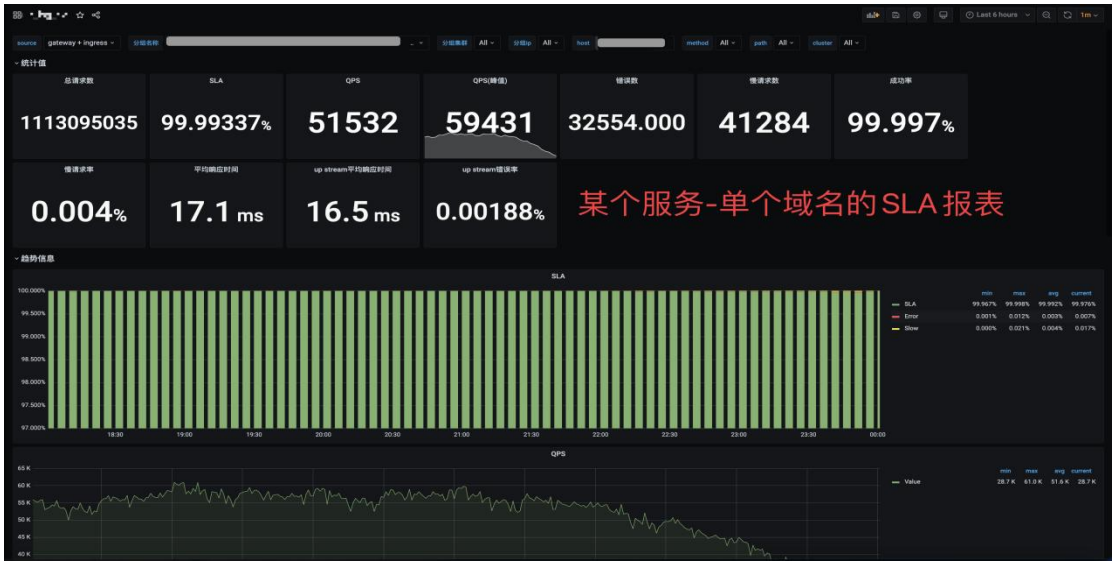


接下来展示一些我们现在正在使用的监控大盘案例，都是需要我们在日常工作中完善的：

下面这张图是我们的域名 SLA 监控大盘，每个域名一张卡片，我们可以在一个 Dashboard 里看到全局的信息。这个大盘在大面积服务故障的时候会非常有效，尤其是在故障恢复阶段，可能有个别服务会因为种种原因存在长尾问题、迟迟没有恢复正常，通过这个大盘我们就可以快速发现异常点。



下图是我们 SLA 的监控：



这是我们服务端质量体系里边最重要的一张报表，请求数、错误数、慢请求有多少、服务整体平均响应时间、后端响应时间、成功率等 SLA 的指标都会在这张报表里边体现。

下图就是前文提到的客户端的质量监控：



其实在这之前我们有用过一些商业产品，但后来发现商业产品不能很好地满足我们的监控需求，所以后来就决定自己研发了「哈勃监控」。

在这里面，我们就可以对客户端的状态有一个感知能力。比如说有时候客户端的网络质量不好，或者说因为一些网络错误、SSL 证书的错误，连接到 CDN 之类的失败了，导致最终请求没有到达服务端。

此时，你看 SLA 报表里面显示一直是 OK 的，但其实这时候用户真实的体验并不是这样的，他对服务的感受其实已经是很糟糕

了。所以我们就需要有客户端的监控体系，在这个报表里，就可以用一些指标，将用户真实的用户体验反馈出来。

2）架构设计

架构设计可能会更偏业务侧一些。我们在故障之前，要尽可能做好服务的架构设计，同时在做一些预案之前，也要把服务架构做好梳理。只有当我们把服务的架构了然于胸，才更有可能在故障发生的时候从容不迫，更好地定位问题。此外，要更多去加入柔性设计，也就是说你的服务要具备一些像降级熔断、故障隔离这些手段，要有这样的柔性设计在里边。这样架构可以提供这些能力，后面才能更好地去做服务的保障。再有一点就是，要尽可能地去规避常规的风险点，比如说单点之类的；同时还要尽可能地去规避架构里面常见的坑。

3）容量评估

下图是我们压力测试的一个页面：



页面 /... / 性能方案

【Report 2.0】压力测试报告v2.0

创建：了，最新修改于：

- from XXX压力测试报告~模版
- 变更信息
- 压测结论
- 压测方案
  - 压测目标
  - 压测范围
  - 压测环境
  - 压测工具
  - 压测依赖
  - 压测风险
  - 压测备注
- 资源评估
- 压测过程

from XXX压力测试报告~模版

变更信息

作者	日期	版本	操作	修改原因(内容)
[illegible]	[illegible]	v1.0	首次压力测试报告	本地mac压力测试
[illegible]	[illegible]	v2.0	首次[illegible]压力测试	项目持久化选型及云瓶颈

编辑

变更信息

作者	日期	版本	操作	修改原因(内容)
XXX	20190815	v1.0	首次压力测试报告	新项目上线前压力测试
XXX	20190915	v2.0	复测报告	项目上线后XXX部分压力测试

压测结论

- 经过多次压测可见，XXX服务的QPS为XXX，瓶颈在XXX，所以建议安全资源水位配置为XX
- 因压测环境与线上环境的XXX不通，所以XXX。
- 在XXX情况下建议重复压测XXX。

压测方案

压测目标

- 评估XXX服务的性能短板以及单个容器最低配置标准。
- 评估XXX容器环境下网络链路的最优配置。

压测范围

- http://XXX.m.com/XXX.XXX
- http://XXX.m.com/XXX.XXX
- http://XXX.m.com/XXX.XXX

压测环境

- 被压端：
  1. 域名：XXX.XXX
  2. 配置：XXX C XXX G

以这个图为例，容量评估是指在服务上线前，通过压测来评估服务的承载能力，从而更好地了解其上线后的状态。基于这些评估结果，我们可以根据业务量级来规划服务的容量。(此外，目前我们已经建设了能力更强的自动化压测平台来覆盖日常的压测需求。)

然而，容量评估并不能完全依赖压测。在压测之前，有没有其他方法可以辅助我们更科学地进行容量评估呢？答案是肯定的，但这往往被忽略，即数学计算。

在进行容量评估时，不能完全依赖压测。我们需要基于对服务的深刻理解，包括每条请求大致消耗多少资源等，进行基础认识的建立。然后，运用这些认识来评估所需的后端资源、CPU 和内存等。这些评估可以通过数学计算的方法来实现。虽然非常精确地计算可能比较困难，但这种方法不可忽略。我们应先进行粗略计算，再结合压测，最终得出一个科学的容量评估。

4）灾备预案和灾备演练

这两点是比较关键的，跟故障会更强相关。下图是我所整理的关于怎么做灾备预案和灾备演练的架构图：



我们把这个过程分为了五个环节，基本按照这五个环节来做，灾备预案就差不多可以完全覆盖了。下面来具体地说一下：

服务梳理：在服务梳理阶段，要基于前面的架构图等来梳理请求链路，要分段分层地梳理请求都经历了哪些层次、有哪些阶段、经过了哪些设备、周边有哪些依赖（包括内部的服务依赖、后端的资源依赖、第三方的依赖等），然后还要梳理架构目前是不是有什么风险、流量有没有经过一个单点、有没有哪个点可能是存在瓶颈的…这些都是我们在服务梳理阶段需要去梳理清楚的。

预案梳理：了解各种情况后，就可以梳理相应的预案了。仔细分析应该用什么样的手段来覆盖，将风险各个击破；然后要尽可能地做多级预案，因为预案如果只有一级的话，容灾能力是不够柔性的；此外，还要借助一些智能调度的手段；最后就是柔性设计，前面也有提到，是更偏业务侧一些，也就是说在服务里要有尽可能多的手段来做自己的一些 failover，可以去做一些降级、熔断等。

沙盘推演：梳理出来预案后，并不是直接做演练。要了解预案是不是真的有效，不应该也不需要直接做演练，而应该是先想清楚。我的建议是去做沙盘推演。在这个过程里，我们要尽可能去发动多的部门协作起来，来审视我们的预案是不是有效。毕竟人多力量大，并且不同团队的人对业务可能有不同的理解，在这种头脑风暴下，就有可能碰撞出更多的可能性。然后在这个过程里，会基于一些可能的故障场景、case 等来做推演，当故障场景 A 出现了，梳理出了预案 A’，那 A’能不能把故障 A 完全解决掉？在解决故障场景的时候，有没有引入一些其他的风险点或问题？在这个过程里面，我们都要想清楚，当得出一个大家都认可的结果后，预案才推演完成。

预案落地：这一步是需要做落地，包括文档输出、功能实现、架构适配、工具建设。

预案演练：落地之后，要通过演练的方法来验证预案是不是真的有效。在这里，我大概列了几种。比如无损演练和轻损演练：我们做故障演练要尽可能地做到对业务无损，但有些预案本身应对的故障场景就是会损害业务的，那这种时候我们要尽可能降低这种演练带来的损失，比如说选不同的时间段、流量的控制、灰度之类的，尽量去做轻损的演练，既然我们是通过故障演练的方式来确保预案有效，那肯定不能因为故障演练而演练出一个大的故障，这就有些得不偿失了；还有就是单点演练跟组合演练：你的演练到底是要一次演练某个模块，还是说要把一个大故障场景里所有涉及的点都演练一遍？这个也是我们在预案演练里需要考虑的。

下图是美图内部灾备预案和故障演练的例子：







大家可以看到，我们是分了几级来做的预案，包括数据的冷备、同城的双 AZ、异地灾备、热备等。我们在做故障梳理的时候，

大致也是按照前面讲到的环节来做：先梳理（分为研发侧和运维侧梳理），然后盘点（盘点这个过程里有哪些故障点），再做 case 推演，而后是预案的输出，最后是演练。

5）持续交付

持续交付也是需要我们在日常建设中就做好的事情。



十几年前，我们主要使用物理机。当线上发生故障，原因是大流量导致性能瓶颈时，首要的应对策略是什么？是扩容。只有通过扩容才能完全承载流量，减少损失。否则，只能进行有损的降级，即丢弃请求。这就涉及到持续交付能力。

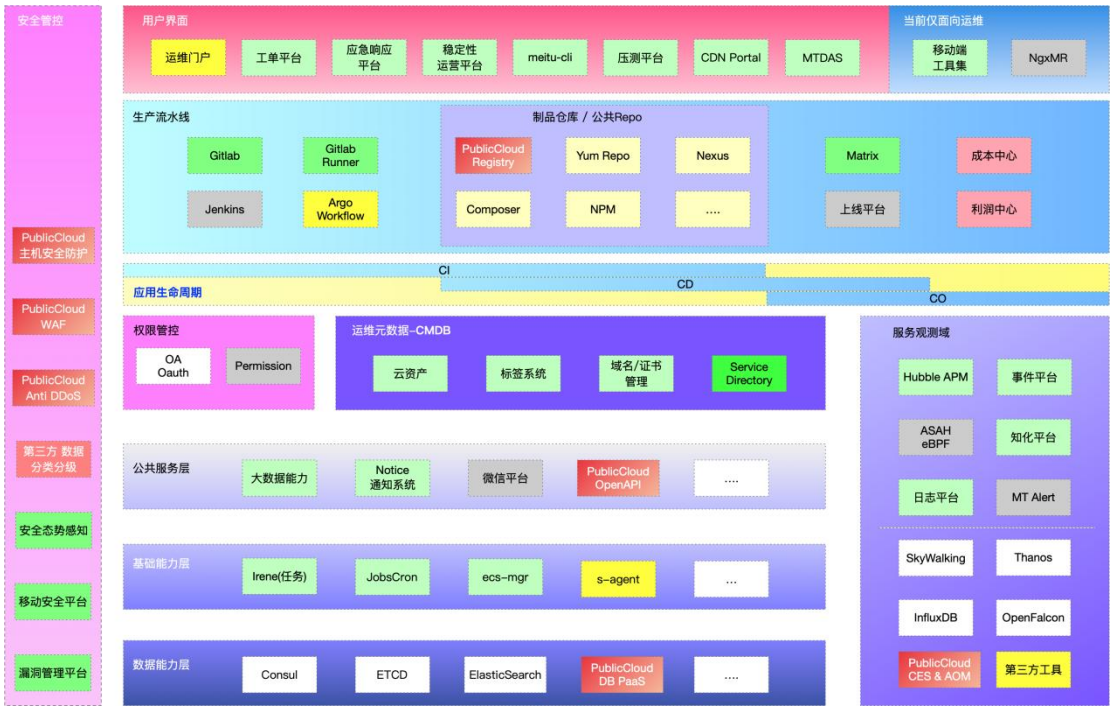
现在，随着云计算和容器技术的发展，基础设施架构具备了弹性伸缩的能力，能够更快速地进行扩容。那么问题是否解决了呢？

并不完全。持续交付相关的场景还有很多，如果发生故障时，持续交付能力无法跟上，故障恢复时间仍会被拖长。

上图展示了我们公司目前使用的一些持续交付体系组件，大家可能比较熟悉。右上角是我们公司内部开发的基于 K8s 的云管平台，支持多集群管控，内部代号为 Matrix。与其他外部云管平台类似，但我们开发得较早。基于这些基础设施，我们能够保持良好的持续交付能力。

上述内容讲述了我们在故障管理之前需要做的事情：在故障来临之前，如何做好预防和准备，以及应对能力的储备。

6）工具体系建设



除了前文有提到的几个跟稳定性保障强相关的几个平台或系统，还有非常多的工具平台对于稳定性保障、安全生产都是不可缺取的，我们需要构建一整套工具体系来实现相关需求并相互协作，最终共同保障服务的稳定性。

2、故障中

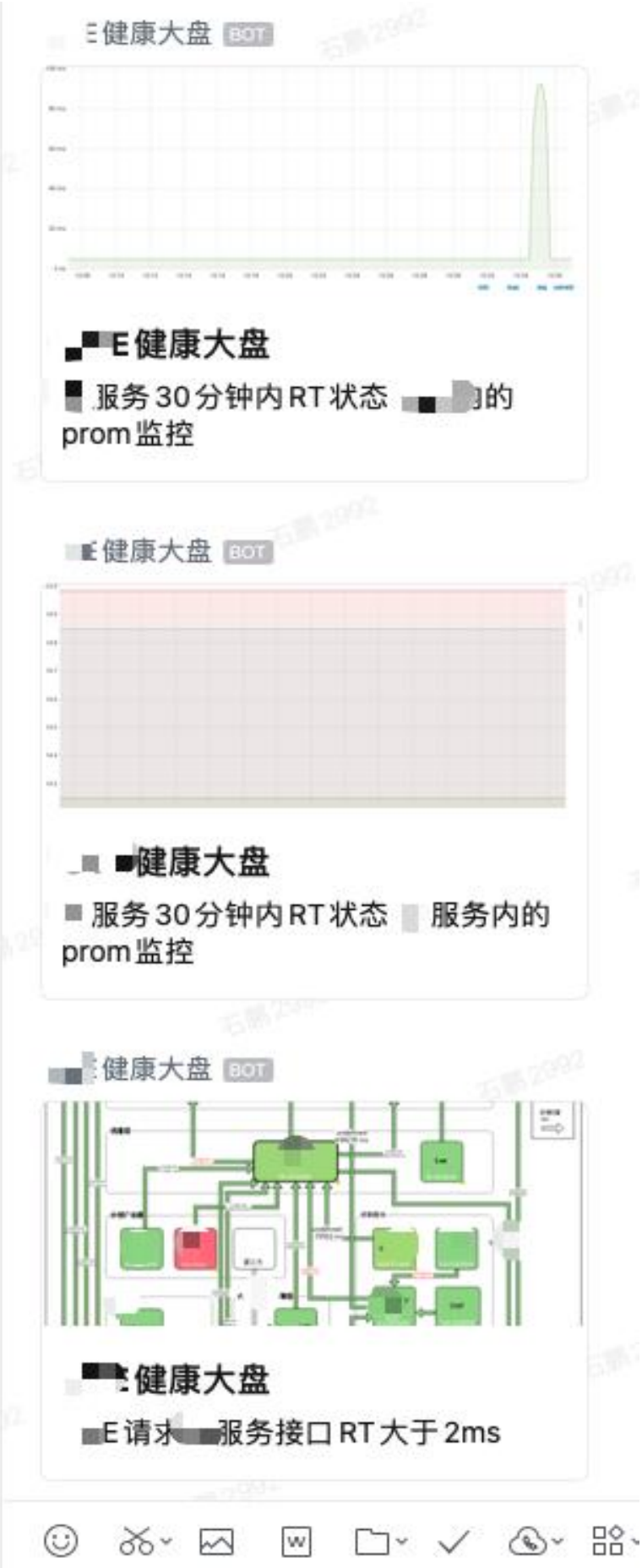


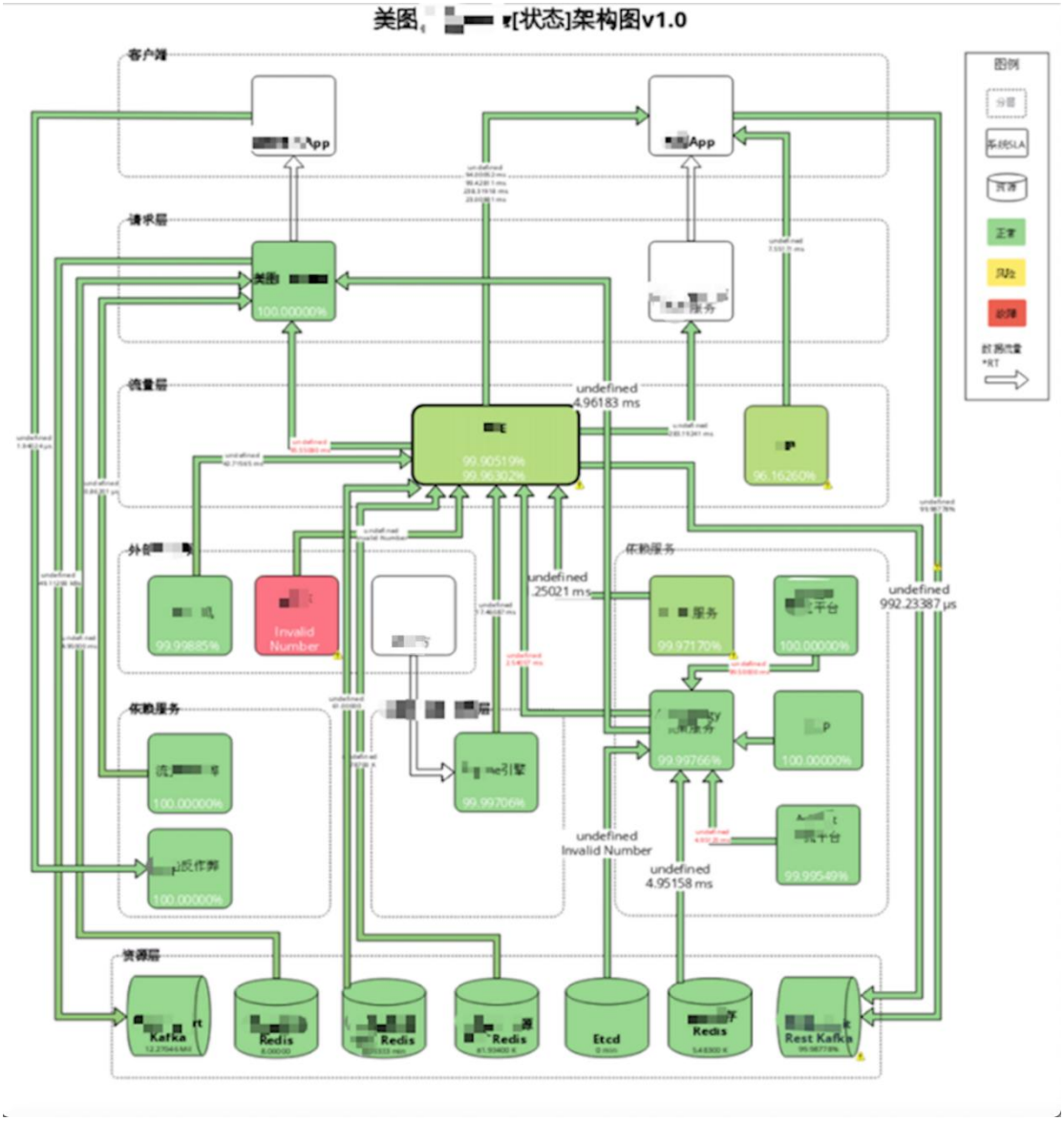
当故障真的发生了，我们应该怎么做？去发现定位和恢复。这里列举几个比较核心的手段：监控告警、日志分析、链路跟踪、故障隔离、容灾切换、降级熔断。只看字面意思也很好理解。下面我将以图结合案例分享具体做法。

1）监控告警


【[告警][级别-高]elb_elb流出流量5分钟突降30%】
忽略告警
time:2020-06-09 10:51:00
instance:9ee2443f-a
-adf94081c66f
__name__:elb:name_instance:m
3ps:rate5m
name:
proxy外网代理vip
value:-0.3497750513814819
warnName:elb流出流量突降30%
规则描述:elb流出流量5分钟突降30%
告警持续时间:1分钟


【[告警][级别-高]elb_elb流出流量5分钟突降30%】
忽略告警
time:2020-06-09 10:53:00
instance:9ee2443f
)c-adf94081c66f
__name__:elb:name_instance:r
ut_Bps:rate5m
name
proxy外网代理vip
value:-0.39166423518381527
warnName:elb流出流量突降30%
规则描述:elb流出流量5分钟突降30%
告警持续时间:1分钟


健康大盘 BOT
E健康大盘
服务30分钟内RT状态 的
prom监控
健康大盘 BOT
健康大盘
服务30分钟内RT状态 服务内的
prom监控
健康大盘 BOT
健康大盘
E请求 服务接口RT大于2ms



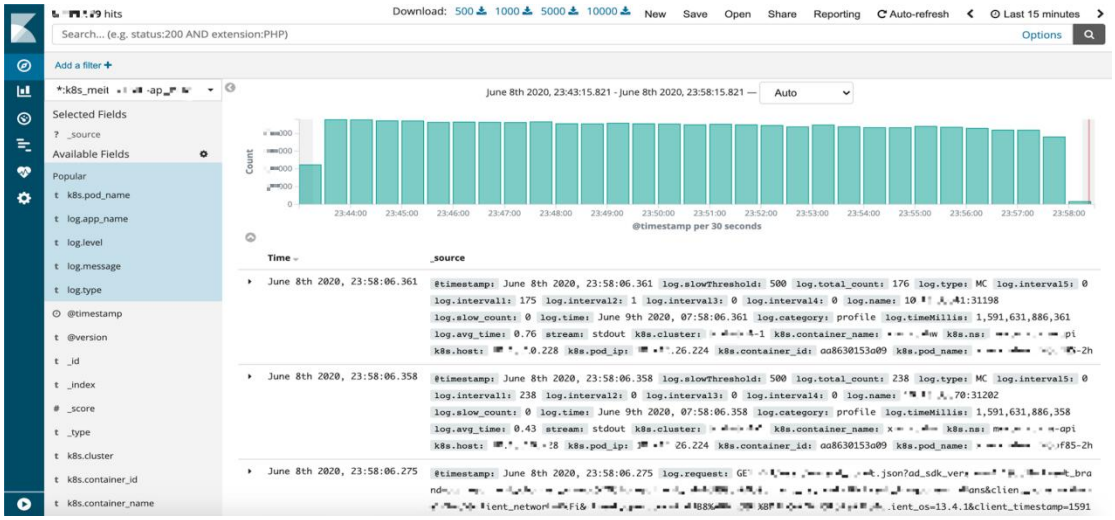
左图是一个流量突变的告警，可能是大家常见到的一种，通过这种文字的手段把报警发给你。这里有一个点需要注意，突降 30%，它其实是不同于常规那种告警，说目前的某个指标超出某个值或者低于某个值，或达到什么水平线那种阈值的告警。两者区别在于，突降是通过对比值得出的结论。这个示例说明在做监控报警时，可能需要考虑的建设点，想想是否有这类需求，有没有这类场景。

最右面的图看起来比较酷，它是怎么实现的呢？它的意义是什么？对比前面这两张文字图，文字图只是告知流量突降 30%，这是一种告警方式。中间的图就是由名为“监控大盘”的机器人发送出来的，也是我们服务的架构图。放大图看，我们的业务模块、交互链路已经呈现出来了；图中有一块是红色的，红色代表模块异常，与异常模块关联的某些组件的 SLA 降低到有个阈值，就变成淡绿色。

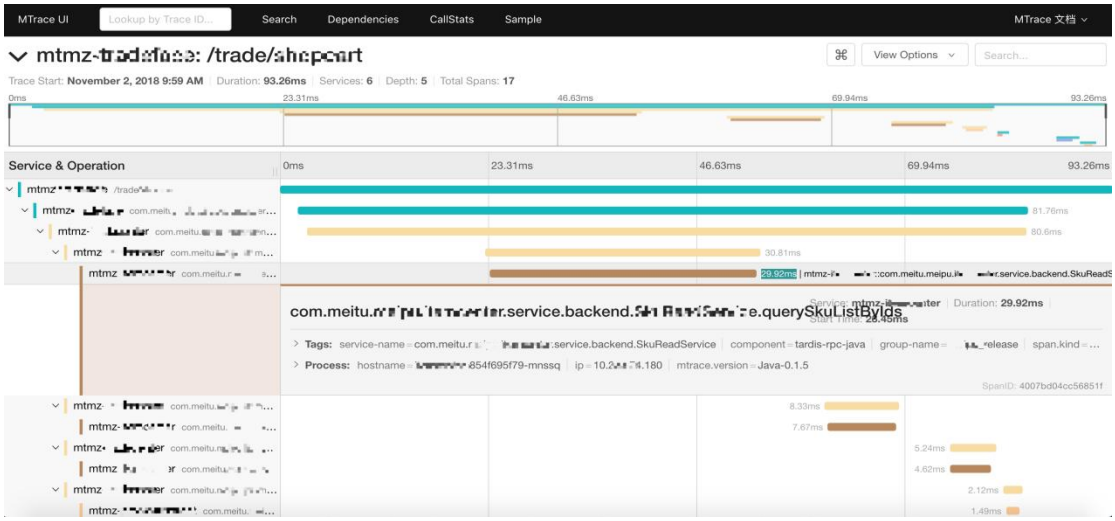
由于红色这块故障引发了另外一个异常，接连引发了一个又一个的异常。通过这张告警图，收获了更多的信息：首先知道系统挂了，或者系统异常了，然后这次异常大概是由什么引起的。一张图就可以让你非常快速地感知。现在有了这张图，不再需要文字告警来通知我组件异常，也不用翻找过往经验做排查。只要看图就能结合已有的监控系统直接做分析、排查，最终得出结论。后者这种监控手段就是让故障背后的原因甚至根因更快触达到用户。

这是基于 grafana 的插件 flowcharting 基于 draw.io 的一个绘图，结合我们的数据填充和告警配置来实现的。

2）日志分析



3）故障中链路跟踪



4）预案执行

故障降级执行方案

程度	故障层次	故障波及范围	降级操作	降级对业务影响功能	恢复操作
严重 ↓ 轻微	区域 Region	区域整体不可用	1. 域名解析：移除 x 内外网域名对于 的解析 2. 后端资源： a. MySQL & Redis 写切 b. 停止 数据同步 3. 集群配置： 全部组件移除 集群配置（kubectl 操作） 4. 镜像库配置：将镜像信息获取源切至 （目前为局部有容灾需求业务直接同步）	1. 行为冻结，已有发布不会被修改 2. 独立部署 集群的容灾服务操作不受影响 3. 对于多区域部署的容灾服务 a. 配置修改会仅对 集群生效，两区域集群配置会不一致 b. 会将所有实例份额调配 集群，且不影响 集群	1. 后端资源： a. 开启 求数据同步 b. 同步完成后切 c. 开启 数据同步 2. 集群配置： Matrix 全部组件加回 集群配置 3. 镜像库配置：将镜像信息获取源 4. 域名解析：恢复 Matrix 内外网域名的 析（Gitlab CI 重新运行 release environment 的最后一次 job） 5. 确认故障期间业务配置差异恢复
		区域整体不可用	1. 集群配置： 部组件移 集群配置（kubectl 操作）	1. 群全部发布行为冻结，已有发布不会被修改 2. 独立部署 的容灾服务操作不受影响 3. 对于多区域部署的容灾服务 a. 配置修改会仅 群生效，两区域集群配置会不一致 b. 会将 实例份额调配 群，且不影响 集群	1. 后端资源：确认 数据同步情况 2. 集群配置： 全部组件加回 置（Gitlab CI 重新运行 release environment 的最后一次 job） 3. 确认故障期间业务配置差异恢复
	可用区 Available Zone	单个可用区故障 （ 单个 k8s 集群）	1. 移除 故障集群配置	1. 故障集群全部发布行为冻结 2. 对于多区域部署的容灾服务，涉及故障集群： a. 配置修改会仅对正常集群生效，区域配置会有差异 b. 会将故障集群实例份额调配到百分比最高集群，且不影响故障机群	1. 集群配置： 全部组件加回 群配置（Gitlab CI 重新运行 release environment 的最后一次 job） 2. 确认故障期间业务配置差异恢复
	节点池 Pool / 节点 Node	部分节点故障	1. 运维下线异常节点（与 ix 无关） 2. 根据实际故障情况限制全体或部分操作，修改 Gitlab CI/CD Variables 中维护相关的环境变量，运行对应环境 environment 的最后一次 Job，具体环境变量配置详见 维护模式 MR	1. 按照维护模式的配置限制全体或部分（按照接口或用户权限等规则设定）的操作	1. 将 Gitlab CI/CD Variables 中维护相关的环境变量置为 ""，运行对应环境 environment 的最后一次 Job 恢复

日志分析、链路跟踪、预案执行都是定位手段，在已经定位问题之后，并且匹配了相应的预案，要求去执行预案。当然前提是有相应的预案应对故障场景。然后依据操作手册，分别按不同的故障层次去执行处理。



恢复之后还要确认执行结果，看服务是否恢复正常。这是我们在故障处理中实际案例的截图。下面进入非常关键的故障后环节。

5）故障处理的一些原则/建议

- 统一目标：恢复优先，问题定界 > 根因定位
- 稳定心态：SRE 一定要冷静，不要慌
- 流程机制：故障升级、War Room
- 故障现场：组织协调约定、信息通报机制

4、故障后



大家要重视一点，在故障发生和解决以后，以为问题真的结束了吗？其实并没有。在故障后，我们应该做好以下环节：故障复

盘、故障改进、预案完善、容量压测、故障模拟、周边清查。我解释下其中几个概念：

- 预案完善：故障发生之前有做过预案，这些预案是不是完善的？在故障处理的过程中，有没有暴露出问题？是否要完善之前的预案中做得不完善的地方等等；
- 故障模拟：意思是用某种手段模拟某些故障，提前知道该如何解决这些问题；
- 周边清查：应该具备举一反三的能力，比如 A 服务出故障，那么 A 服务周边有一些相关联的服务有可能会受到影响。举个例子，比如集群里面有 N 台机器，收到机器 A 磁盘接近 100%的一条告警，只需处理机器 A 就可以了吗？当然不是，在处理完该异常之后，肯定要查看同集群的其他机器是否存在类似情况。

1）故障复盘

在故障复盘阶段，最核心的思想就是要回顾关键的时间点：故障是从什么时间发生的？从什么时间到什么时间结束的？在这个过程里面有哪些非常关键的操作？有哪些关键的信息？从什么时间点定位到问题？在什么时候执行怎样的操作？这些时间点都是非常重要的。

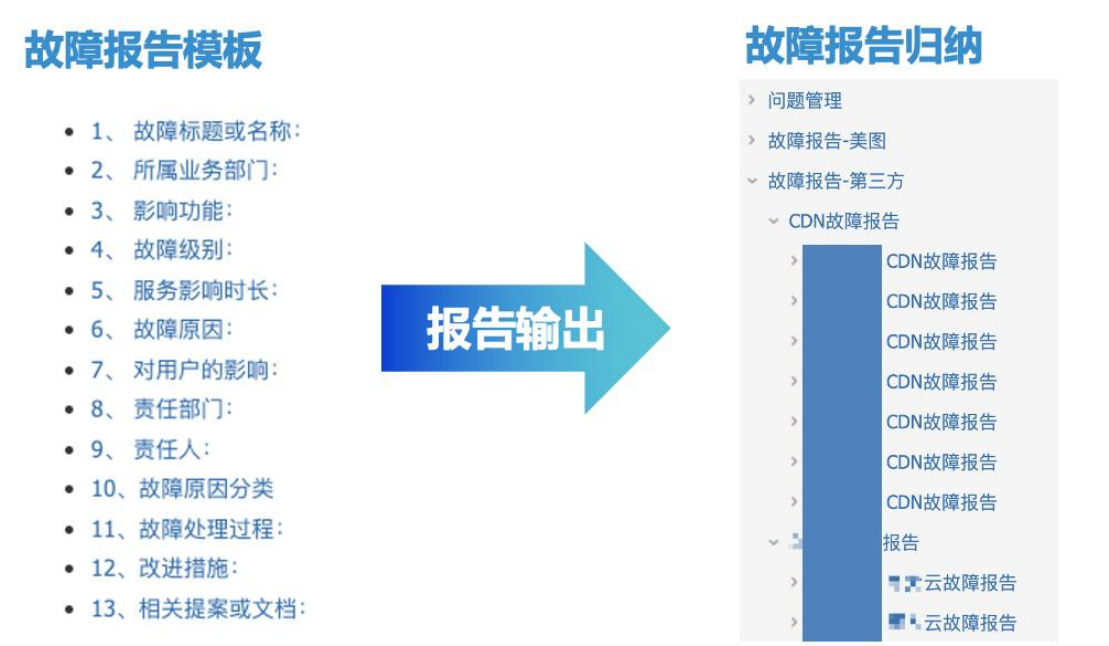
在故障复盘的过程中，要把这些操作的时间点一一还原回去，才能完整地回看操作是否合理有效。看在某个时间点执行了某个预

案，思考某个时刻做出这样的操作是否恰当，此时有没有其他更好的手段能够恢复服务，某个时间段为什么没有及时处理而导致故障时间变长等等。

做好故障复盘，我们有黄金三问法：

- 我们应该怎么做，才能更快地恢复业务？
- 我们应该怎么做，才能避免再次出现类似问题？
- 我们有哪些好的经验可以总结、提炼，并固化？

基于以上三个问题，通过自问自答的形式弄清故障复盘。



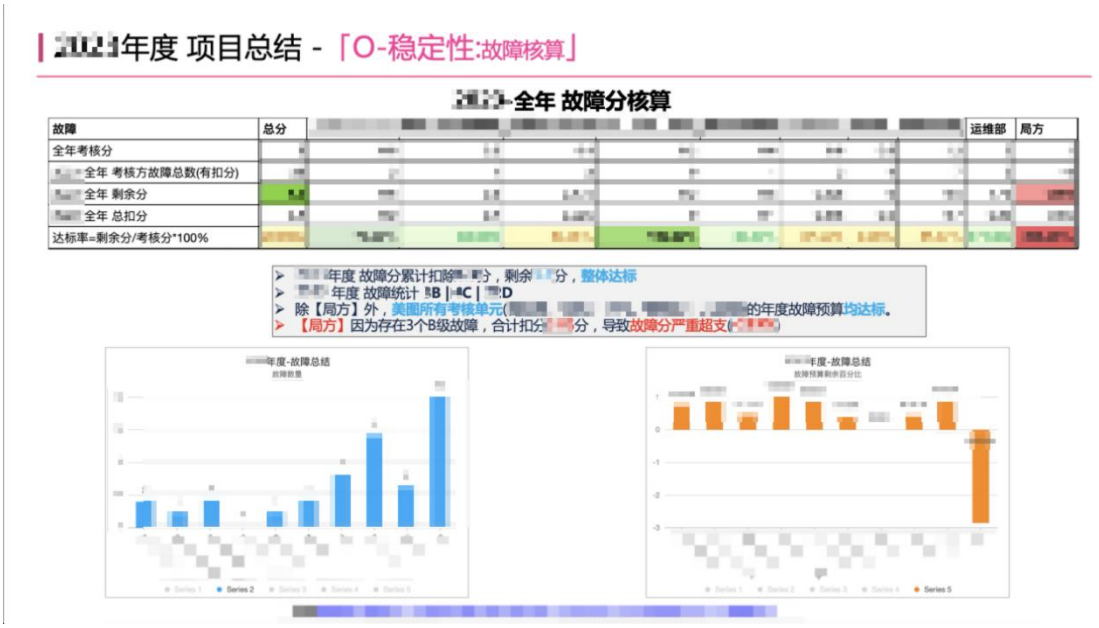
这是故障解决后绕不开的一点，在故障发生后都会做一个报告，上图就是我们内部的一个故障报告模板。在报告里面写清楚故

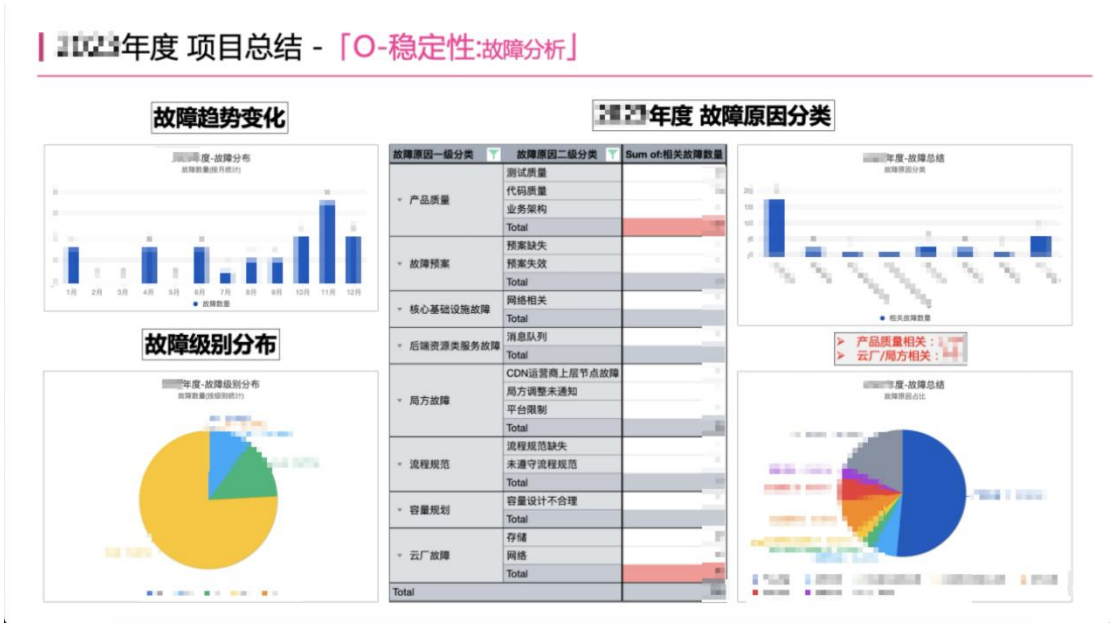
障级别、责任人、处理过程、改进措施，将这些关键点归纳成故障报告的文档。

2）周期回顾

如前文所述，故障管理的工作本质上是持续迭代的、稳定性保障能力也是螺旋式上升的。

除了做好故障前中后的各块工作，我们还需要周期性地对稳定性保障工作做汇总分析、复盘总结，下面就是我们做 OKR 复盘的一个实例。





(三)总结及展望

我所看到的几个技术趋势

根据当前行业的发展趋势，我们可以看到以下几个重要方向：

1. 云原生：云原生技术将会持续发展和深化。越来越多的企业在采用云原生架构，云原生的优势在于它的灵活性和可扩展性，这些特性将帮助企业更好地应对不断变化的市场需求和技术挑战。

2. 可观测性：可观测性工具和实践正在被更多企业认同和应用。通过提高系统的可观测性，运维团队能够更迅速地发现和解决问题，提升系统的可靠性和性能。

3. 混沌工程：混沌工程在越来越多的场景中落地实施。通过故意制造故障并观察系统的响应，企业可以提高系统的抗压能力和容错性，从而提升整体服务的稳定性。

4. Dev-X-Ops：各类 Ops（例如 GitOps、ChatOps、AIOps 等）融合类的实践越来越多，不论是什么 Ops，实用性才是最重要的。只要这些 Ops 实践能解决实际问题、提升工作效率，它们就值得被采用。

5. AIOps：AIOps 正在持续进化，并且其应用范围也在不断扩大。随着大模型和人工智能技术的快速发展，AIOps 可以帮助企业更好地处理复杂的运维任务，实现更智能化的运维管理。

综上所述，SRE 未来的展望主要集中在持续创新和应用实用性强的新技术和新方法上，以提升系统的可靠性、可维护性和智能化水平。

最后的话：如何面对汹涌的技术浪潮

面对汹涌而来的技术浪潮，作为 SRE 从业人员，如何应对这一变化迅速的环境？以下几点可以帮助我们在技术变革中保持镇静和从容：

1. 看清本质：理解每种技术的核心价值，不被表面的复杂性所迷惑，找到其中的本质，并以此为基础进行实践。

2. 拥抱变化：接纳并适应新技术和新趋势，例如 AI 技术的广泛应用。顺应技术发展的趋势，积极学习和应用新技术，以保持竞争力。

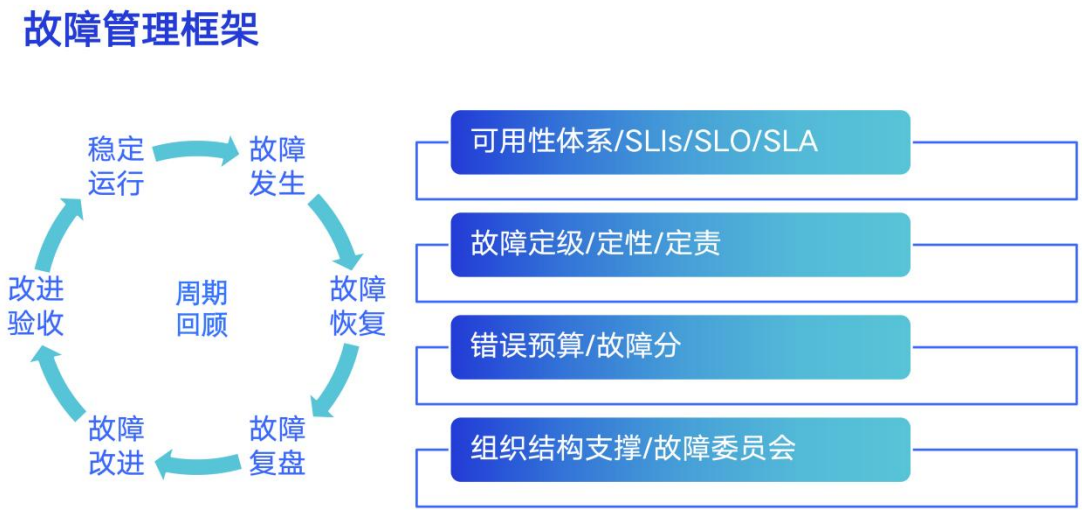
3. 顺势而为：在理解和接受变化的同时，合理运用这些新技术来提升工作效率和质量。在技术变革中找到适合自己的位置，并发挥最大的价值。

4. 做好定位：明确自身的角色和价值，知道自己能提供什么样的价值，应该提供什么样的价值。持续改进和提升自己的技能，以确保能够稳定地、高效地提供高质量的服务。

通过这四个方面的努力，我们可以在技术浪潮中泰然自若，迎接未来的挑战，而不用担心被新技术所取代。保持积极的态度，拥抱变化，不断提升自己的能力，这样才能在不断变化的技术环境中立于不败之地。

本章节根据 2025 年 04 年 10 日 Qcon 上演讲，补充如下；

补充总结及展望



故障管理框架可概括为一个闭环的持续改进流程：系统在稳定运行阶段进入常态监测；当故障发生后迅速启动应急处置并恢复故障恢复；随后开展故障复盘与周期回顾，梳理成因与改进措施；最后落实改进验证并重新回到稳定态，形成周而复始的演进路径。

该流程由四大治理支柱支撑：

① 可用性体系（SLIs/SLO/SLA）为目标设定与衡量提供量化基线；

② 故障分级、定性与责任划分确保响应优先级和问责清晰

③ 错误预算与故障分通过量化容错与激励约束平衡创新与稳定性

④ 组织结构支撑及故障委员会 为跨团队协同、资源调度与策略制定提供制度保障。

多年来的实践表明，利用这一循环与支柱相结合，可系统化提炼 SLI、SLO、SLA 等可用性标准，贯通预算管理与经济责任分配，实现从预防到复盘的全链路故障治理。

我所看到的几个技术趋势



趋势	关键词	关键价值	典型演进方向
1. 云原生	基石	持续解耦 & 弹性扩展	K8s 全生命周期管理、Serverless、无状态优先架构
2. 可观测性	一体化	度量+追踪+日志 融合，数据闭环驱动优化	OpenTelemetry 标准化、AIOps 异常检测、体验指标（UXI）
3. LLM Ops	智能辅助	SRE 引入 LLM 做根因分析、智能决策	· 事件语义解析 · 自然语言变更评审 · Prompt 版本管理
4. AI Agent	自愈闭环	SRE Agent 主动故障响应、自适应扩容	· 多 Agent 协作协议 · 经验知识图谱 · 终态自愈策略
5. MCP / ANP	协议化	“上下一致”运维控制协议 & 多租户纳管	· MCP：层次化闭环指挥 · ANP：跨租户策略编排
6. AI 可信系统	安全透明	可解释、合规、伦理可审计	· 责任链追溯 · 攻击面评估 · 数字水印与模型偏差监测

串联逻辑

云原生 → 可观测性：弹性基础奠定了统一数据采集与实时洞察的可能。

可观测性 → LLM Ops：高质量可观测数据为 LLM 提供上下文，驱动智能诊断。

LLM Ops → AI Agent：LLM 赋能的知识推理成为 Agent 自愈的“决策大脑”。

AI Agent → MCP/ANP：多 Agent 协同需要协议化编排，形成运维“控制面”。

MCP/ANP → AI 可信系统：协议化闭环与合规治理共同确保 AI 可靠、安全、可解释。

一句话总结：这一套路径以云原生基础设施为根，以可观测数据为血液，以 LLM 与多 Agent 为智能中枢，通过 MCP/ANP 协议实现跨环境协同，最终落到可解释、安全、合规的 AI 可信体系，实现运维与智能的深度融合与持续演进。



看清本质　拥抱变化　顺势而为

做好定位　葆有价值　泰然自若

认清浪潮：云原生、微服务、Serverless、服务网格、可观测性、低/零代码、平台工程、AI Agent、Git-Chat-Biz-Dev-Sec-Fin-Green-AI Ops……新名词层出不穷，本质仍是「用更高抽象的工程体系，把业务价值交付得更快、更稳。

价值优先：别被概念牵着走，始终围绕企业与客户的核心痛点评估 ROI——能提效、降本、控风险，才值得投入。

定位清晰：问自己“我的独特位置是什么？我能在这波技术中提供怎样不可替代的价值？”——找准专长，深耕垂类，而非全盘追逐。

小步试错：先做可验证的小实验（PoC / MVP），用数据说话；成功即滚动扩展，失败则快速止损，保持技术投入的弹性。

沉淀能力：把一次次试验中验证有效的方法论、工具链、最佳实践固化成平台化能力，让创新成为可复用、可迭代的“流水线”。

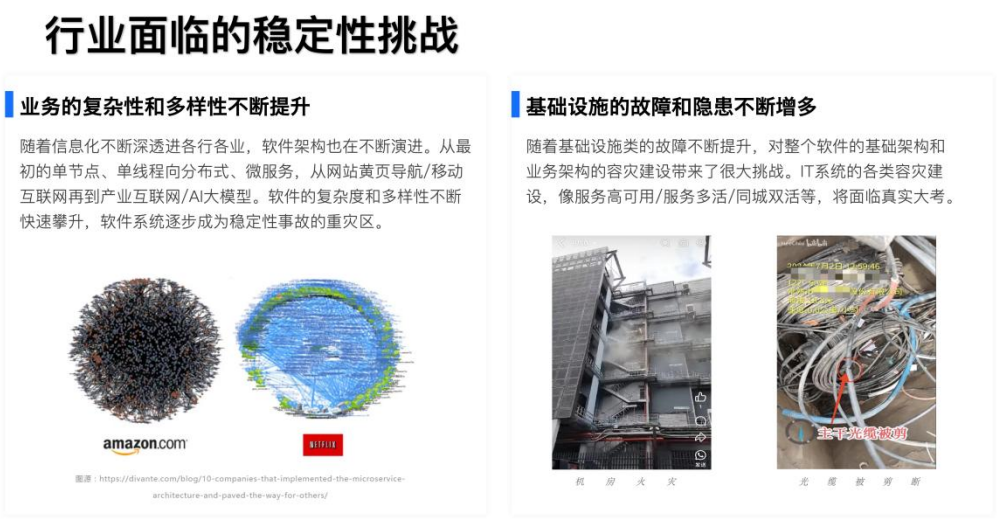
心态从容：技术迭代永无止境。保持学习曲线的同时，也要守住交付与运营的基本盘，既在浪尖上冲浪，也能随时回到稳健的航道。

5.3.6 B 站轻量级容灾演练体系构建

SRE Elite 精选原因

B 站的轻量级容灾演练体系构建聚焦于高效、低成本地确保系统稳定性。首先，通过建立原子级故障库和特化场景，演练对象和故障类型变得标准化，易于操作。演练体系涵盖技术支撑、跨领域赋能、组织文化及流程优化，以实现常态化、自动化的演练环境。其核心亮点包括：简化故障注入流程、丰富的观测能力、自动化参数计算、标准化的演练协议，以及跨部门的协同机制。演练的目标是通过自动化工具支持，降低操作复杂度，确保演练可以在没有 SRE 人员的情况下自主管理，最终推动全员参与的容灾演练文化，增强业务系统的抗压能力和恢复能力。此实践考虑充分，方案完整，工程化实践扎实，在业界中处于较为领先的地位，具备很强的参考意义。

（一）背景及设计原则



首先，当前新形势下的稳定性挑战日益严峻。随着 IT 的发展，信息化不断渗透到各个行业。从几十年前的单节点巨石服务，到如今的分布式、微服务和云原生架构，软件的复杂度和多样性都在几何倍数地攀升。这导致软件系统逐渐成为稳定性事故的重灾区。不单软件，硬件问题也变得更加复杂和严重，基础设施的故障和隐患也在不断增加，并且显露出更大规模的影响。

例如，前两个月新加坡某机房发生火灾，虽然仅一层受损，但对国内部分出海业务仍造成了重大影响。再如，7 月 2 日常熟光纤被剪事件，尽管光纤只被部分切断，但不巧阿里云和我们公司都在受影响的那一段，导致严重后果。**这些事件表明，当前基础设施故障带来的挑战已不再局限于单台设备的问题，而是可能导致整个机房或网络的瘫痪，进而引发大范围服务甚至整个站点的不可用。**



此外，行业内对高可用服务的建设要求越来越高。平时可以吹嘘高可用服务的能力，但当真正面临外部考验时，能否经受住考验才是关键。

对于我们 B 站来说，面临的核心挑战包括：

B 站热搜体质，小故障大流量

由于微博有热搜机制，而 B 站的主要用户群体是年轻人，导致很多时候我们虽然遇到的故障并不大，但很快就会被推上微博的热搜。这种情况给我们负责稳定性和技术的同事带来了巨大的压力。

以下是一些相关案例：

- 商业 CDN 故障引发的级联效应影响全链路的图片服务：

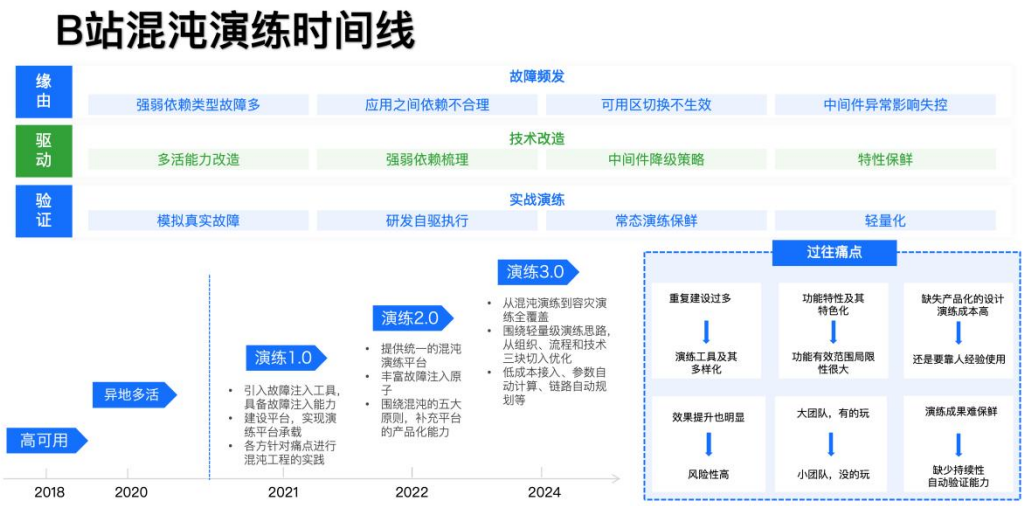
- IDC 网络设备故障导致专线异常，影响办公区网络；
- 业务代码变更导致数据库问题；
- 光缆被剪断导致可用区脱网等。

持续多活建设，有效性急需验收

在过去的几年中，众所周知，B 站遭遇了一次严重的系统故障——即所谓的“713 事件”。随后，我们内部启动了一系列专项工作，针对用户端、接入层、服务层、中间件层以及基础设施的各个层面，实施了多活技术支撑建设和业务架构的全面改造。

尽管改造工作已经进行了多年，但其成效如何，我们仍感到不确定。即便是负责此项改造的团队，可能也只是依据内部的多活白皮书和技术架构指导方案进行了部分改造工作，而这些改造是否真正达到了预期效果，尚需进一步验证。为了确保这些改造措施的有效性，我们迫切需要一个专门的平台来进行测试。这也是我们启动改造的初衷——频繁发生的故障迫使我们进行必要的改造，并在改造完成后，需要一个平台来验证其功能是否达标。

因此，我们需要对强弱依赖故障、应用依赖不当的故障、可用区切换的有效性、技术改造的推动效果、多活改造的推动效果，以及降级策略和兜底策略的实施效果进行严格的测试和验证。



为了评估这些改进措施的实际效果，我们有必要开展实战演练。在这些活动中，我们将在真实的工作环境中模拟实际问题，并鼓励研发团队自主驱动演练。鉴于保障团队的人力资源有限，若仅在故障发生后进行专项验收，可能无法实现持久的效果。因此，我们期望能够将演练常态化，以降低费用并维持效果。这便引出了轻量化演练的议题。

自 2018 年起，B 站便开始着手高可用性建设，而到了 2024 年，我们进一步推进了异地多活和双活架构的建设。

在 2021 年的演练 1.0 阶段，我们采纳了多种开源混沌工程工具，包括来自阿里和 Pingcap 的工具，并在此基础上构建了一个简易平台，用于故障注入和验证。

然而，由于各部门独立运作，我们面临了一系列问题。首先是重复建设，不同部门开发了多种功能相似的工具。其次，由于各业务部门的关注点不同，工具的设计和使用缺乏统一标准。此外，技

术团队开发的平台虽然比手动执行命令有所进步，但其效果并不突出。

尽管混沌演练的引入提升了效果，但同时也增加了风险。研发人员在功能验证时可能会使用这些工具进行测试，但工具的局限性有时会引发真实的线上故障。混沌工程并非随意注入故障，而是需要遵循逻辑和方法论。

大团队拥有充足的人力资源进行演练保障，而小团队则资源匮乏，难以获得同样的关注。通常，大家只在重大活动前进行专项演练，而日常演练缺乏持续性，难以实现常态化。

为了解决这些问题，**我们在 2022 年的演练 2.0 阶段**，内部发起了一项统一行动，整合了大型事业部的资源，建立了一个统一的平台。我们增强了故障注入的能力，简化了公共部分，并根据混沌工程的五大原则指导产品化设计。以下是关于 Netflix 提出的五大原则：

- 建立稳定状态的假设；
- 用多样的现实世界事件做验证；
- 在生产环境中进行实验；
- 自动化实验以持续运行；
- 控制最小化爆炸半径。

演练 3.0 的目标是希望在公司内部实现演练的常态化和真实有效的运行。为了达成这一目标，我们计划将演练从混沌演练逐步扩展到容灾演练，最终实现全面覆盖。



在前期工作的基础上，我们从去年开始推进演练 3.0，旨在使演练在公司内**实现常态化、真实有效地运行**。为此，我们提出了从混沌演练迈向容灾演练的目标，实现全覆盖。围绕常态化演练，我们引入了轻量级演练的思路，从**组织、流程、技术**三方面入手，全面优化。

我们不仅对平台进行改造，还关注以下关键点：

- **低成本覆盖与接入：**如何以更低的成本实现广泛的演练覆盖和系统接入。
- **自动化参数计算与链路选择：**在演练过程中，针对大量的输入参数，如何实现自动计算；如何自动圈选演练链路等功能。

这些需求引出了演练 3.0 阶段的目标——**轻量化演练**，这是我们持续一年多并仍在推进的工作。

为实现轻量化目标，我们在流程方面进行了优化。由于需要在全公司范围内推广，涉及的部门和角色众多，**如何高效协作成为关键**。这也呼应了之前提到的团队协作问题：如果团队协同不到位、职责不清，可能导致各自为政，影响整体进度。因此，我们需要**固化组织的演练流程和章程**，明确需要开展的重大演练，倒排计划，确定各部门在不同阶段的参与方式（后续会有详细的图示）。

其次，**技术流程**也需要标准化和固化，建立**标准化的演练过程**。在具体的技术支撑方面，我们需要强化产品能力。作为平台工程部门，我们思考如何减少人员的认知负担。只有将演练能力完全交付出去，**例如无需 SRE 人员陪同，业务方可以自行开展演练，这才是我们的最终目标。**

围绕演练过程，我们：

- **丰富观测指标能力**：拓展可监测的指标，增强对演练效果的评估。
- **扩展演练对象和范围**：增加可注入故障的演练对象，丰富演练场景。
- **实现自动化**：在脱离人工操作的情况下，对演练的预期结果进行指标量化定义，使演练过程能够自动化地收集结果。
- **建立护栏机制**：当出现问题时，自动防护，确保演练不会引入新的故障。

第三，在组织协同方面，我们建立了各职能、各业务的协同机制，提升内部对演练的认知。**演练并非“没事找事”，而是有实际收益的工作，需要大家重视。**此外，我们还需要进行培训，降低学习成本。即使产品体验再好，复杂的系统仍需一定的学习，因此相关的培训是必要的。

基于以上，我们提出了几个重要目标：

1. 平台角色定位：希望新版本的平台能够扮演演练专家的角色，极大降低用户的使用成本。

2. 规范演练行为：基于标准的流程，规范演练的行为和操作。

3. 强化安全性：严格控制影响半径，确保演练足够可靠、可控。

4. 推进常态化演练：这是我们的最终目标，如果无法实现常态化演练，前面的工作将失去意义。

（二）体系设计详情



轻量级容灾演练体系

接下来，我们引出了**轻量级容灾演练体系**，整个体系分为四大块。

一、技术支撑

1. 原子级故障库：在底层，我们建立了一系列原子级的故障库，包括模拟 CPU 异常、网络延迟、丢包等简单故障注入。

2. 场景封装

- 一般场景：封装常见的故障场景，如 CPU 高负载、容器异常、网络丢包等，方便用户直接选择和使用。

- 特化场景：针对公司内部业务特性，如双活、多活架构和自研中间件，定向封装故障能力，供内部团队使用。

3. 混沌实验构建：在上述场景之上，我们构建混沌实验，包括实验的创建、分发以及故障注入到具体载体上。

4. 观测能力：引入观测机制，评估故障注入效果，验证系统恢复情况，并有效监测业务稳态指标，及时发现业务异常。

5. 经验沉淀

- 故障组合与复杂场景模拟：通过组合多个故障，模拟复杂场景，如网关故障、SLB 故障、跨网机房级别的故障等。

- 演练模板：将上述经验沉淀为演练模板，服务团队在演练时只需选择对象和范围，配置业务稳态指标，无需关心故障设计细节，降低了演练门槛。

6. 产品层面的演练管理：在日常使用中，通过产品化的演练管理来编排故障，并生成演练报告。整体路线是从工具到平台，再到一站式服务，最终实现轻量化演练的目标。

二、技术落地

1. 研发过程融合：将故障注入能力融入 CI 流程和测试回归流程，特别是对于强弱依赖的演练，应在制品库出包前的准出阶段自动模拟故障，检测是否存在问题。

2. 传统混沌演练：继续开展系统故障、网络故障以及内部特定场景的演练。

3. 大型容灾演练：针对跨机房、多活架构的容灾演练，以及降级和兜底方案的验证。

三、跨领域赋能

1. 应急响应：在应急响应过程中，利用平台进行演练溯源和故障定位。

2.攻防演练：模拟故障链路，随机注入故障，提升系统的抗压能力。

3.大模型赋能：借助内部大模型团队的能力，对演练内容进行复盘和梳理，基于历史应急问题，识别薄弱环节，建议针对性演练。

四、组织文化

组织架构支撑：调整组织架构，建立虚拟项目团队，确保演练协同目标的实现。

制度建设与文化宣贯：制定演练制度，日常宣导，提高全体技术团队的安全容灾意识、演练防范意识和应急响应意识。

我们基于组织、流程、技术三个方面展开工作。认识到组织的重要性，如果组织协同不到位、责任不清，工作将难以推进。通过明确负责人与职责，确保每个人都积极参与并承担相应责任，从而提高演练的效果和效率。

组织支撑



在组织支撑方面，我们建立了**安全生产委员会**。该委员会由一线领导组成，在组织中担任重要角色，负责公司全局性的稳定性决策和重大稳定性演练路线的规划。他们还承担立法职责，制定惩罚规则，建立基线和红线。此外，委员会致力于提升公司整体的应急协同能力和文化意识水平，指导全局的管控、容灾演练系统的规划管理，以及各个子系统之间的协调。制度建设方面，则包括编写相关的白皮书。

在**制度建设部分**，我们规范了容灾架构设计，制定了容灾演练章程和故障演练规范。

在**文化宣导方面**，每年都会举办为期一个月的“安全生产月”活动。在每个楼层设置宣传展板，开展一天的公司专场活动。各部门，如机架部门和各业务线，会分享他们在稳定性改进方面的工

作。此外，我们通过学院培训，录制视频，方便大家随时学习。对于校招和新入职的同事，入职培训中也会特别关注这些内容。

“孤岛行动”这是我们近期开展并将持续推进的工作，即进行机房级别的断网测试。例如，本周二我们就进行了断网演练。通过这种真实的专项活动，推动公司各业务线主动检查并改进自身的问题，确保保障措施到位。这是一种基于目标牵引的方式，促进工作的开展。

此外，我们还通过**技术分享渠道**，如哔哩哔哩技术公众号，每周发布两篇文章，分享各技术线的实践内容。内部也有相应的平台发布技术实践，供大家学习和参考。

流程支撑



在组织支撑之外，我们还需要流程支撑，主要分为两部分：**演练执行流程和演练全生命周期流程**。梳理这些流程的目的是指导业

务更高效地开展日常实验演练，并帮助平台更有效地设计功能，避免因主观设计导致各部门使用不便。

演练执行流程

演练执行流程旨在指导我们如何开展一次实验演练，分为演练前、演练中和演练后。

演练前

- 业务梳理：确定业务场景，明确哪些场景需要演练，制定验证方案，评估演练可能带来的影响。
- 方案确认：决定演练方式，如采用染色流量演练、线上不切流演练，或先切流再注入故障进行验证。

演练中

- 制定演练范围和环境：确定涉及的集群和机器范围。
- 配置规则：模拟故障的业务场景，选择需要观测的指标，设计护栏机制，确保出现问题时能够及时拦截并恢复故障。
- 验证场景：通过指标和断言，验证演练是否符合预期。这部分主要支撑自动化演练。

演练恢复

- 确保故障恢复后，业务状态符合预期。关注故障撤销后业务是否仍存在异常，确保其真正有效恢复。

演练后

- 常态化演练计划：评估本次演练场景能否常态化，制定未来演练计划，包括突袭演练等。

- 待办事项推进：针对演练过程中产生的待办事项，安排时间推进，并通知相关负责人。

演练全生命周期流程

演练执行流程只是一次性的，但我们的目标是根据“五大原则”，在真实环境中模拟真实场景。因此，通常先在非生产环境进行演练，再逐步过渡到生产环境。

非生产环境演练

◆在非生产环境（如 UAT）中执行演练流程，发现并修复重大故障和隐患。

◆流量处理较为简单，可进行跨网环境的演练，可能不需要染色。

生产环境演练

◆需考虑是否进行染色，以及染色的粒度（实例级别或请求级别）。

◆是否需要提前预切流量，切流后在一侧注入故障进行验证。

◆验证与恢复阶段：需要联动应急场景，处理可能引发的故障，并按照相应流程进行恢复。

推进常态化与突袭演练

- 针对服务的各个重要业务场景，开展全生命周期的演练。

- 制定周期性的突袭演练计划，确保业务在各种情况下都能稳定运行。

技术支撑

技术支撑



在有了组织支撑和流程标准的支撑之后，我们需要落实技术手段，我们整体梳理了六个“化”，以支持组织协同和流程标准的有效落地，确保轻量化演练目标能够保质、保量、保效地开展。这六个“化”包括：

演练协议的标准化：使用统一的演练协议，保障所有的混沌操作统一控制

故障引擎的自研化：自行研发故障注入引擎，以满足特定需求，提高灵活性和可控性。

实验参数的精简化：简化实验参数，降低复杂度，使演练过程更高效、更易于执行。

演练覆盖的全面化：扩大演练的覆盖范围，确保所有关键业务场景都能被演练到，提升整体系统的可靠性。

演练过程和护栏的可视化：对演练过程和保护机制进行可视化展示，方便实时监控和管理，确保在演练过程中能够及时发现和处理问题。

演练流程的自动化：最终实现演练流程的自动化，减少人为干预，提高效率和准确性，保障演练工作的持续推进。

演练协议标准化



首先，我们强调了流程的标准化，进而提出演练协议也需要标准化。最初，我们的项目基于开源工具，主要使用了阿里巴巴的 ChaosBlade，这在前期为我们节省了大量的故障原子的开发成本。然而，在与行业内的交流中，我们发现一些公司声称其故障引擎完全自研，但实际上可能也是在他人代码的基础上修改而来。

与此同时，我们内部各部门也已经开发并封装了一些演练工具。这导致了一个问题：我们并非从零开始，而是需要整合许多内外部的演练工具。然而，由于团队不同，大家在技术设计上各有想法，导致工具的命令行参数、接口协议的抽象（如 Proto 文件）都不一致，尽管功能类似。

为了解决这一问题，我们需要将这些工具进行整合和统一，主要通过模型抽象。具体而言，我们提出了两个标准：

1. 标准的演练描述模型

演练对象：明确要演练的目标是什么。

演练范围：确定演练涉及的范围。

载体类型：演练范围内的载体类型，例如是否包含 IP、端口等信息。

演练参数：例如在网络故障演练中，需要哪些参数，这些参数的格式和内容需要统一抽象。

演练模式：包括是否有预检要求，场景是否符合故障注入的条件等。

2. 标准的演练交互模型

实验的注入和注册：统一实验的启动和登记过程。

实验的执行和恢复：标准化执行故障和恢复故障的流程。

审计和查询：建立统一的审计和查询机制，记录指令转换、执行结果等信息，方便监控和排查问题。

通过使用类似 Golang 的接口进行行为抽象，我们确保了系统的统一性和兼容性。这样，我们就能够在这套系统中快速兼容原有的多种演练工具，逐步引导它们过渡到我们新的标准化工具。这种标准化不仅有助于提高演练的效率和可靠性，还为未来的扩展和维护奠定了坚实的基础。

故障引擎自研化



在完成标准化之后，我们进一步推进了故障引擎的自研化，主要目在于分享我们在混沌实验过程中使用开源工具时遇到的问题，

以及自研所带来的改进。之前，我们大量使用了开源工具，但在实际应用中遇到了不少挑战，因此决定进行自研。

我们选择**自研的主要原因**包括：

1. 提高演练成功率：开源故障注入工具的可靠性不如预期，降低了我们的演练成功率。虽然代码实现质量一般可以接受，但成功率的下降对我们影响较大，因为团队需要对演练指标负责。

以下是一些具体的问题案例：

DNS 干扰演练的问题：

故障注入逻辑无效，导致故障逃逸：开源工具在进行 DNS 干扰演练时，通常直接在 `/etc/hosts` 文件的尾部添加一条新的域名解析记录来覆盖目标域名。然而，这种方式在现代操作系统和应用中并不可靠。

原因分析：许多应用程序在进行 DNS 解析时，会按照顺序查找 `/etc/hosts` 文件中的记录。如果前面已经有一条匹配的域名解析记录，后面新增的记录可能不会被应用程序使用。这导致故障注入逻辑失效，故障被逃避。

影响：演练过程中预期的 DNS 故障没有发生，无法验证系统在 DNS 解析异常情况下的稳健性。

网络不可用故障的问题：

注入命令过长，导致执行失败：当需要对大量（如数百个）实例注入网络不可用故障时，开源工具生成的命令行参数会过长，超过了系统对命令长度的限制，导致命令无法执行。

原因分析：开源工具在注入网络故障时，可能会为每个目标实例生成一条命令，当目标实例数量过多时，命令行参数累积，导致命令长度超出系统限制。

影响：无法对大规模的实例集群进行网络不可用故障的注入，限制了演练的覆盖范围和规模。

通过 TC（Traffic Control）实现的网络故障的问题：

使用 TC 工具来实现网络不可用故障时，故障恢复后网络状态可能仍不稳定，或者恢复过程出现异常。

原因分析：TC 工具在网络故障注入和恢复的过程中，可能未能完全清除或还原网络配置，导致残留的规则影响网络性能。

影响：即使故障注入结束，系统网络可能仍然受到影响，导致演练对生产环境造成不必要的干扰，增加了演练的风险。

CPU 负载注入的问题：

生效步长和节奏不匹配，导致负载未达预期：在模拟 CPU 高负载时，开源工具的故障注入逻辑在负载递增的步长和节奏上与我们的预期不一致。

原因分析：可能是开源工具在实现 CPU 负载递增时，使用了固定的线程或进程数量，未能根据实际 CPU 核心数和系统性能进行动态调整。

影响：无法准确模拟预期的 CPU 高负载情况，导致演练结果不准确，影响对系统性能和稳定性的评估。

2. **改进故障注入链路的可靠性：**开源混沌平台对故障注入的编排较为简单，类似于 HTTP 无状态请求。故障注入链路中每个节点之间缺乏状态管理，如果中间某个节点失败，整个链路会中断，难以回溯和排查问题。另外缺少状态上报和控制，在故障编排过程中，如果某个步骤出现问题，系统无法有效地上报状态，也无法进行控制和补救，导致演练过程断层。

3.加强服务/资源鉴权，增加演练安全性：开源项目无法满足我们对服务和资源鉴权的需求，注入对象过于裸露，无法与企业内部的服务对象、集群对象匹配，导致演练注入范围的权限控制存在问题。

安全风险：缺乏对演练注入范围的严格权限控制，可能导致对非预期目标的故障注入，带来安全和稳定性风险。

4.增强可观测性，方便问题排查：我们无法在未知情况下进行故障注入，因此需要对注入过程进行全面观测。

问题难以排查：故障注入的组件链路较长，包括用户请求、故障调度、故障注入代理，以及可能的 Kubernetes 调度。缺少对指令

转换、执行结果的记录和审计，一旦出现问题，难以及时定位和解决。

缺乏指标埋点和状态上报：在演练过程中，针对整体实验、单个故障、注入对象等，缺乏必要的指标埋点和状态上报，无法实时监控演练状态和效果。

基于开源 ChaosBlade 的二次开发和改造

为了解决上述问题，我们对开源的阿里 ChaosBlade 进行了二次开发和改造，主要目标包括：

提升演练注入和过程的控制力度与稳定性：

改进故障注入逻辑：针对 DNS 干扰演练，我们采用更可靠的方法来干扰 DNS 解析，例如修改 DNS 配置或拦截 DNS 请求，确保故障注入有效。

优化命令生成和执行方式：针对网络不可用故障的命令过长问题，我们重新设计了命令生成机制，支持批量注入，避免了系统命令长度限制。

完善故障恢复机制：深入研究 TC 工具的工作原理，改进网络故障的恢复逻辑，确保在故障恢复后，网络配置能够完全还原，不影响系统的正常运行。

动态调整 CPU 负载注入参数：根据实际 CPU 核心数和系统性能，动态调整负载递增的步长和节奏，确保模拟的 CPU 负载符合预期。

扩展演练故障的性能和注入范围的规模化：

支持大规模实例的故障注入：通过优化资源调度和故障注入策略，能够稳定、高效地对大规模实例集群进行故障注入，提升演练的覆盖范围。

提高故障注入的可靠性：减少故障逃逸和注入失败的情况，确保演练成功率。

提升代码可维护性，加速迭代效率：

符合内部技术标准：按照公司的技术框架和研发标准进行开发，并接入内部的可观测平台，提高了代码的维护性和开发效率。

建立标准的演练描述模型和交互模型：

演练描述模型：统一描述故障注入的对象、范围、载体类型、参数和模式，确保不同故障类型的定义和扩展具有一致性。

演练交互模型：规范演练过程中的交互方式，包括指令的发送、状态的上报、结果的收集等，确保各组件之间的通信和协作顺畅。

加强服务/资源鉴权：在故障注入过程中，严格控制注入对象和范围，确保只有授权的服务和资源才能被注入故障，提升演练的安全性。

增强可观测性和审计：在故障注入链路的各个环节增加日志记录和指标埋点，实时监控指令转换、执行结果等信息，方便问题的定位和排查。

通过自研，我们不仅解决了开源工具在实际应用中遇到的问题，还为演练体系的持续优化和扩展奠定了坚实基础。自研的故障引擎使我们能够更好地控制演练过程，提高演练成功率，确保演练的安全性和可靠性。

演练覆盖全面化

针对公司内的各种服务，包括小型部门和海外业务，我们致力于实现演练覆盖的全面化。为此，我们采取了以下措施：

1.对象类型的抽象与管理

抽象载体类型：尽管我们的服务架构复杂，拥有数十种类型的集群，但所有服务最终都运行在容器或主机上（无论是物理机还是虚拟机）。因此，我们将演练的载体类型抽象为容器和主机两类，满足大部分故障注入的需求。

依托 CMDB 系统：利用配置管理数据库（CMDB），我们可以方便地选择目标集群和组件。例如，要对某个大数据集群下的组件进行演练，只需在 CMDB 中勾选相应的组件，便可获取其实例范围。

2.自动化故障注入与回收

解决接入成本与安全隐患：面对数十万容器的规模（如 Kubernetes 集群），如果在所有集群中长期驻留故障注入组件，既增加了接入成本，又存在安全风险（如代码 Bug 导致全公司故障）。

动态打标与撤标机制：我们实现了自动化的打标和撤标机制。故障注入组件仅在演练时自动部署到目标集群（通过 DaemonSet），演练结束后自动回收并撤销标记。这样，即使有命令下发，未被打标的集群也不会执行，确保安全性。

3. 快速部署与性能优化

自动化部署工具：采用 Helm Chart，实现集群的快速自动化部署，简化了部署流程。

流程编排优化：基于前期的自研工作，对故障注入的性能进行了优化，重新设计了流程编排，解决了规模化下的广度问题，提高了演练效率。

4. 多环境支持与跨网络调度

支持多种环境：我们的业务涉及国内、国外、机房内部和外部等多种环境。对于机房内部（有专线连接）的演练相对容易，挑战主要在于机房外和海外业务的演练。

区域信息映射与调度：通过内部统一的 Zone 信息映射，方便地进行区域调度，支持在公有云环境和海外业务中开展演练。

Proxy 管理：建立了 Proxy 管理机制，方便实现跨网络的指令调度，确保演练指令能够安全、可靠地传递到目标环境。

通过以上措施，我们成功地扩大了演练的覆盖范围，确保各类服务和环境都能参与到容灾演练中，提高了整体系统的可靠性和稳定性。

演练过程可视化



第五部分讨论的是演练过程的可视化。这与之前提到的标准模型类似。当需要演练的对象较多时，除了接入我们主要的可观测团队（他们的标准化程度较高，基本都采用了 OpenTelemetry ）外，还需要考虑一些尚未采用标准的小型事业部的业务服务。如果不将这些服务接入标准，在演练时他们将无法查看相关指标。为此，我们制定了一个监控指标的标准，相较于前者，这个标准更为简化，指标也较为精炼。

在抽象层面，大家的思路基本一致，因此我们抽象出了一个“获取行为”。首先，需要明确你的对象是什么；围绕这个对象，确定它有哪些指标；对于每个指标，如何获取其实时数据。这意味着，在演练过程中，当选择了服务对象后，系统会将该服务对象的指标列出，供用户自行勾选。

勾选完指标后，在演练过程中就可以实时查看这些指标的情况。另一方面，我们的演练架构图也在持续迭代中。通过围绕演练对象，基于其 Trace 动态调用链和 CMDB 的静态拓扑关系，我们能够了解在某个点注入故障后，对其上下游以及自身会产生哪些影响。

演练流程自动化



第六部分讨论的是演练流程的自动化。为了实现演练流程的自动化，我们主要关注以下三个方面：

1. 演练场景能否自动验证：即我们的演练场景是否能够实现自动化的验证。

2. 演练能否自动执行：即演练过程是否能够实现自动化的执行。

3. 演练预期能否自动验证：即演练的预期结果是否能够实现自动化的验证。

对于场景的自动验证，我们主要借助了专业的测试平台。他们已经开发了许多功能强大、易用的平台工具，特别是在 UI 自动化测试方面。因此，我们直接与他们对接，复用了他们的 UI 自动化测试能力。由于他们的测试也关注场景，包括多场景结果测试和单应用结果测试，这使得我们在演练特定业务场景时，能够完全实现自动化验证。

在演练的自动化执行方面，主要包括以下三部分：

定时调度器：这一部分相对简单，类似于常见的代码定时执行机制。

流程编排的自动推进：我们能够自动推进演练流程的逐步滚动，确保演练过程的顺利进行。

护栏机制的支持：在演练过程中，设置护栏机制，确保在出现问题时能够及时预防和处理。这一点实际上与前面提到的可视化和可观测性密切相关。只要能够获取可观测指标，添加断言和规则逻辑引擎，就能实现这部分功能。

对于预期结果的自动验证，实际上与场景验证类似，但更偏重于指标层面。也就是说，我们需要判断在特定场景下，预期结果是正向的还是反向的。例如，在某些场景中，我们期望在演练后必须出现异常。然而，在实际演练中，我们发现有些预期的问题并未出

现，链路未按照预期运行。这种情况下，我们需要查明结果被代偿的原因，这是不应发生的。

业务场景演练实践



第三部分是关于业务场景的演练实践，我们将举几个实例进行说明。之前主要讨论了平台技术和组织方面的内容。通过将平台实验与演练协同一体化，我们希望将过去需要在小会议室进行线下协同的演练方式，转变为线上协同。借助演练的面板，我们可以将演练人员、相关系统、业务服务以及演练过程中涉及的信息集中在一起。

在演练开始的配置阶段，我们可以在同一页面上完成演练对象的设定范围圈选、流程编排、原子实验的具体配置，以及观测护栏

的配置。随后，在演练执行阶段，我们可以在同一页面推进执行流。最后，生成演练报告并进行验收。通过这一流程，实现了演练全过程的线上化和一体化管理。

可用区断网容灾演练



此前我们介绍了服务级别的演练，接下来讨论可用区（AZ）级别的断网演练。当前，这类演练并非通过我们的平台注入故障，而是采用其他方式实施。这里详细说明我们内部是如何开展这项工作的。

当服务级别的跨可用区演练基本就绪后，我们会进一步进行公司级别的大规模可用区演练。具体而言，我们直接在交换机上配置策略，切断机房之间的专线连接和公网入口。这样做的总体目的是确保当可用区级别的故障发生时，我们的业务系统和基础架构服务能够具备故障逃生和容灾的能力。

大型活动事件——S 赛演练



首先，给不太了解游戏的朋友科普一下，S 赛指的是英雄联盟全球总决赛。全球总决赛从 S1 到 S13（“S” 即 Season，意为赛季）已举办了十三届比赛，因此被称为 S 赛。由于 LOL 的玩家众多，且 S 赛在广大电竞爱好者心目中有非常高的地位，因此每次 S 赛都是对我们的一次考验，因此需要进行详细的规划，应对业务挑战。

总体规划与执行步骤：

1. 年度目标设定与规划：

制定大目标：每年设定一个主要目标，围绕该目标进行详细的方案规划。

关键节点拆解：将目标分解为全年需要完成的关键节点和任务。

组建团队（拉阵营）：确定在不同环节中需要哪些角色和事业部的参与，明确每个人的职责和推进事项。

2.现状摸排与演练执行：

摸排现状：在演练前，对当前的系统和环境进行全面的摸底排查。

处理白名单等特殊情况：在摸排过程中，处理需要排除的白名单等特殊配置。

开始演练：按照计划进行演练，模拟各种故障和异常情况。

复盘总结：演练结束后，对整个过程进行复盘，总结经验和发现的问题。

3.演练思路分为三大块：

断专线测试：

目的：先切断两个机房之间的专线，验证在实施双活或多活架构后，机房之间是否仍存在不合理的依赖。

断公网入口测试（南北向流量）：

目的：切断机房的公网出口流量，测试各个机房是否能独立承载进入的公网流量。

全断测试：

目的：同时断开所有连接，模拟一个机房完全脱离网络的情况，观察对整体系统的影响。

4.具体计划与执行细节：

治理多活风险：从最初开始，识别并处理多活架构可能带来的风险。

公网流量切换：尝试在两个机房之间切换公网流量，测试各自的承载能力。

逐步断开专线：

保留 1% 流量测试：在断开专线时，保留 1% 的流量进行测试，确保真实流量下的可靠性。

模拟断公网场景：

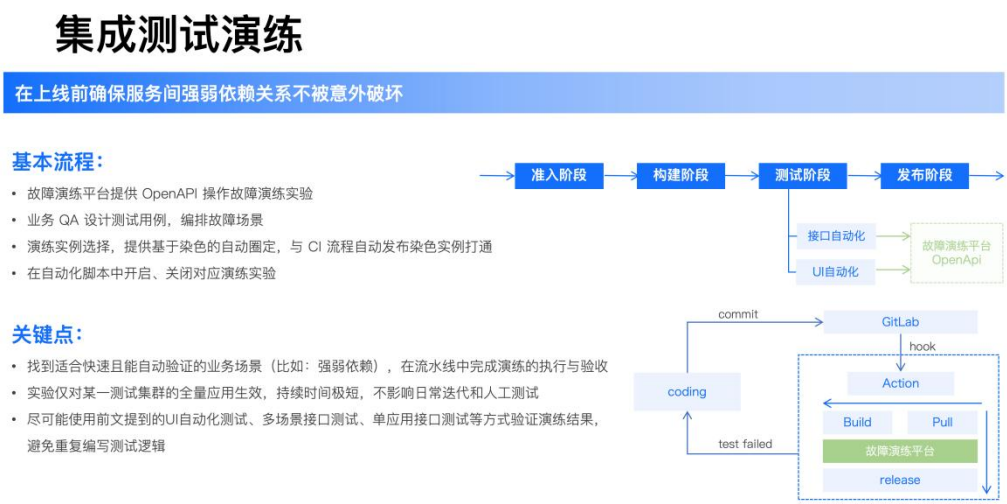
非主机房公网断开：断开非主机房的公网连接，观察系统反应。

再次保留 1% 流量测试：在全面断网前，保留少量流量进行最终测试。

完整脱网验证：最终实现机房的完全脱网，评估对业务的影响和系统的稳定性。

通过上述步骤，团队能够全面评估在大型活动（如 S 赛）期间，系统在面对各种故障场景时的应对能力

集成测试演练



1. 集成测试演练的核心场景：

故障注入能力的迁移：将故障注入能力移至测试环境，是日常测试团队经常进行的工作。

平台提供的 Open API：测试平台提供了开放的 API，供 QA 测试使用。

测试用例的编写与集成：QA 人员编写测试用例，并将其集成到 CI（持续集成）构建流程中。

自动化测试流程：在代码提交后，测试用例会自动运行，检测是否引入了不合理的依赖等问题。

2. 快速验证场景的重要性：

选择适合 CI 的测试场景：关键是寻找能够快速验证的场景，方便在流水线中运行。

不适合 CI 的场景：如模拟断网的测试不适合在 CI 流水线中执行。

充分利用测试功能：利用 UI 自动化测试、多场景接口测试等功能，进行有效的验证。

3.S 赛（英雄联盟全球总决赛）演练：

年度赛事准备：每年都会针对英雄联盟赛事进行专项演练。

核心流程：

场景分级：按照内部的分级规则对赛事场景进行分级（具体规则将在 PPT 中提供）。

形成场景地图：根据分级结果，构建具体的赛事保障场景地图。

能力应用与摸排：利用前述的各种测试能力，对场景地图进行全面的摸排和演练。

专项演练的效果：

问题发现：通过专项演练，发现了大量问题，主要集中在代码层面的强弱依赖和不合理的降级方案上。

提升系统可靠性：这些演练有效提高了系统的稳定性和对大型活动的保障能力。

（三）总结及展望

成效总结



效率提升：通过引入多种手段，演练从规划到落地的时间由周维度缩短为天维度，降低了成本，提高了效率。

全面覆盖：以前只有核心部门能充分利用演练工具（“吃得饱”），现在所有部门都能参与演练，使用这些工具，实现了全面覆盖。

演练频次增加：过去需要领导推动才能进行演练，频次可能是每季度一次。现在演练频次提升到了每周一次，业务同事因为演练不会引入过多新负载，变得更加积极，不再抗拒参与。

演练有效性验收成本：以前需要人工介入，成本较高，现在我们实现了自动化验收，并能够复用测试中的多个场景，从而实现了

技术建设的复利效应。这种自动化与复用机制不仅提升了工作效率，也大大减少了人为错误的发生。

实践反思

- 演练之前要有明确的目的，不然演练结束会觉得没有效果
- 大规模推广时候，需要结合业务团队特性来量身定做方案，主动推动
- 强弱依赖演练、预案演练、多活演练，具备高密度、高关注度、高收益的特征
- 演练自动化的关键还是在于过程护栏的有效性和演练效果验证的自动化
- 需要有面向演练场景的自动化用例，可能需要专门去补充
- 接口类的自动化用例，可以考虑用流量录制的流量集去转化
- 不同系统处于不同阶段，需要以不同的混沌实验帮助提升稳定性建设

5.3.7 蚂蚁故障应急全流程体系构建及应用实践

SRE Elite 精选原因

本案例来自蚂蚁集团的平台工程与技术风险部行业一线的实践总结构建了完善的故障应急全流程体系，主要亮点包括：明确故障

定义和分级，聚焦关键业务的 GOC 场景，专注保障最重要的服务接口和结果点，驱动业务稳定性建设；还建立极具特色的应急值班长体系，明确应急角色和职责，提升组织协同能力；开发技术风险管理平台（TRM），实现故障全生命周期的数字化运营，相关工作令人。

在展望未来时，此次分享向 SRE 领域的同仁们展示了人工智能与 SRE Agent 如何增强应急响应能力，并在智能化故障定位、决策和处理方面展现出巨大潜力，极具启发性。

（一）背景及设计原则

备注：

本文档内容基于 2024 年 Qcon 上海站，蚂蚁集团的 SRE 技术专家刘凯宁演讲实录。为提升可读性，本部分将采用第一人称进行叙述，并包含更多的口语表达。

我想与大家分享一句我非常喜欢的话："Fix the problem, not the blame." 在稳定性工作中，我们经常会遇到问题，可能会问："这是谁的责任？不是我的问题，该怎么办？" 但我们必须专注于解决问题，而非相互指责，这样的指责没有意义。如果我们以解决问题为导向，而不是追究责任，那么稳定性工作一定会做得更好。所以，我想先将这句话分享给大家。

在正式开始之前，我想提出几个问题，请大家思考一下：线上故障究竟是什么？谁应该为线上稳定性负责？以及如何评价应急处理的效果？接下来，我将为大家介绍蚂蚁故障应急的整体流程体系，并尝试回答这三个问题。

稳定性灵魂三问

- 线上故障到底是什么？
- 谁应该为线上稳定性负责？
- 如何评价应急到底做的好不好？

我希望此次分享能够给大家带来一些启发。虽然蚂蚁的实践并非最完美，但希望能够为大家提供一些借鉴和参考。

本次分享主要包括四个部分：

1. 蚂蚁的故障体系介绍：我们将从故障的定义开始，全面介绍我们在故障管理中所使用的平台。

2. 蚂蚁的应急体系：由于每个公司都在讨论故障，应急响应也有相应的标准和平台。我将介绍我们常用的平台能力，以及我们最重要的组织结构。这是本次分享的重点之一。

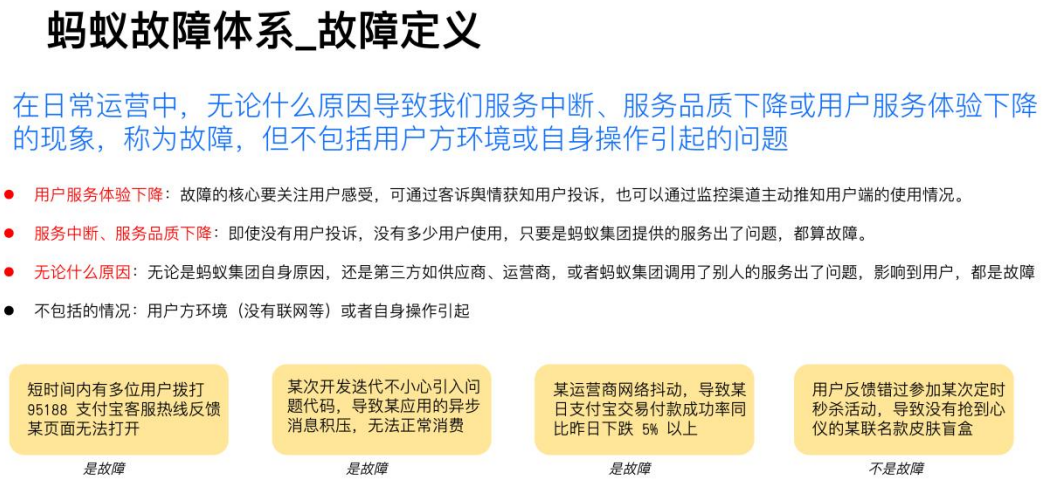
3. 故障生命周期：我将介绍一个完整的故障案例，详细说明其中包含的各类信息。

4. 未来展望：鉴于在本次 QCon 大会上，许多公司都在讨论 AI 技术。我将分享一些关于 AI 如何在我们的 SRE 领域中应用的初步想法。

（二）体系设计详情

蚂蚁故障体系介绍

故障定义



首先，我们进入蚂蚁故障体系的介绍。事实上，许多公司都在讨论他们遇到的各种故障，但究竟什么是故障？这一点必须明确定义。如果不加以清晰界定，可能会出现各种模糊的说法，比如“这

是一个线上问题”或“这是一个重大故障”，导致无法准确识别和处理。因此，我们首先需要明确什么是故障。

在蚂蚁集团，故障的定义如下：在日常运营中，无论任何原因导致我们的服务中断、服务品质下降或用户服务体验下降，都被视为故障。当然，这不包括用户自身的问题。

这里有几个关键点需要关注：

第一，用户服务体验下降：这指的是什么？每家公司都有不同的业务，对于蚂蚁而言，涉及支付交易、互联网业务、财富保险等。我们故障定义的核心是关注用户的感受，必须具备手段去感知用户是否遇到异常情况，并及时获取相关信息。因此，我们需要通过客户投诉、舆情监控等多种方式来感知用户的反馈。

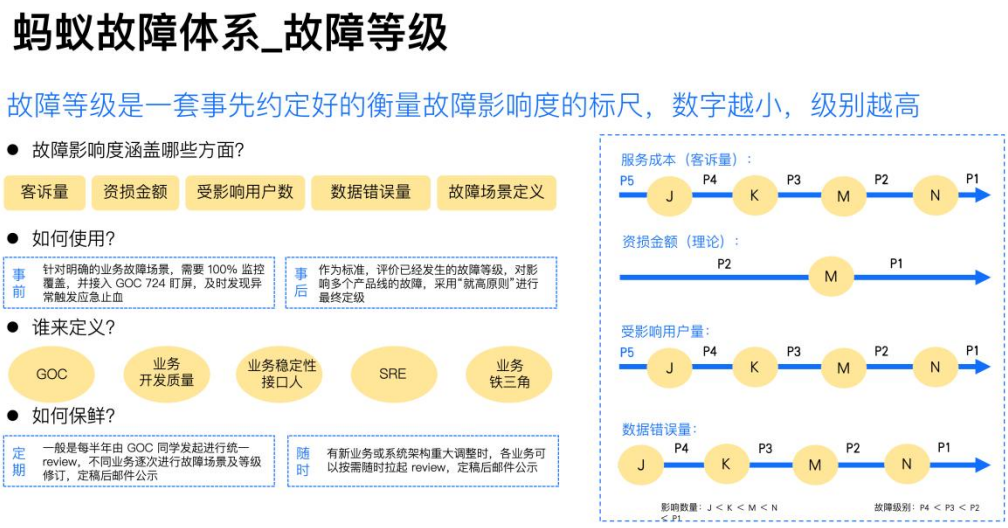
第二，服务中断和服务品质下降：正如之前提到的，可能存在一些用户无法感知的场景。如果用户感知不到，如何发现问题？为此，我们建立了可观测性系统，拥有大量服务层面的稳定性监控。即使用户没有直接感知，但蚂蚁集团提供的服务确实出现了问题，这也被视为故障。

第三，无论何种原因导致的故障：不论是我们自身的原因，还是依赖方的问题，例如运营商、第三方服务提供商，或我们调用的其他服务出现问题，都被纳入故障范围。由此可见，我们对故障的定义非常严格，正是由于这种严格性，我们更加追求在故障稳定性和应急领域的能力建设。

同时，也有一些情况不包括在故障定义中。例如，用户自身的操作问题。举个例子，如果客户在短时间内打电话反映某个页面打不开，这是典型的故障。另外，如果开发同事发布了代码，导致异步消息积压，这也是故障。再比如，运营商的网络抖动导致我们某些功能下跌，这同样被视为故障。

但是，像用户反馈自己错过了时间、没有抢到某商品等情况，就不算故障，因为这不属于我们的责任范围。明确了故障的定义后，接下来我们来看一下故障的等级划分。

故障等级



那么，什么是故障等级？故障等级是一套事先约定的衡量故障影响度的标准，数字越小，级别越高。通过故障影响度，我们才能评估故障的严重性。

故障影响度涵盖哪些方面？在蚂蚁集团，我们主要参考客户投诉量、资金损失金额、受影响用户数等指标。不同公司可能有不同的标准，以上仅供参考。如右图所示，我们可以先解释一下其中的其中ＪＫＭＮ，这些字母其实代表脱敏后的量级的阈值，如在服务成本里头（客诉量）J 可能代表 10 个投诉，K 是 100 个投诉（假设），具体是哪个数量不重要，这个和具体公司的情况相关.

而其中 P 级别从左到右数字逐渐减小，但故障等级逐渐升高，对应的影响也越来越大。我们目前的等级划分从最低的 P5 到最高的 P1，大家可以参考。但这里存在一个问题：这些 P 级（P1、P2 等）具体意味着什么？稍后我会详细介绍。我们需要理解的是背后的思路。

定义故障等级的作用是什么？在故障发生之前，我们可以基于这些定义，开展可观测性建设和平台能力构建，目的是提前识别潜在问题，进行预防和保障。当故障发生后，我们需要对故障进行评估。我们采用“就高原则”，即如果一个案例符合多条故障定义，我们以最高等级为准。

那么，故障等级由谁来定义？一些部门可能有自己的质量部、技术风险部或运维部门。在蚂蚁集团，由 GOC（全球运营中心）、业务开发质量团队、业务 SRE 以及业务稳定性部门的同事共同制定故障等级。制定完成后，需要提交给业务负责人审批。由于业务随着时间发展，公司的战略也可能调整，如何保持故障等级的有效性？通常，GOC 同事每半年会统一进行一次审核，例如针对支付

宝、财富、保险等业务域，统一审核故障定义和等级，并予以公示，大家按照统一标准执行。

大家可以思考一下，自己的公司是否有一个明确的机制，能够清晰地了解自身业务可能出现的故障类型，以及可能达到的最高等级。

故障序列



当公司出现新业务或重大架构调整时，我们应及时启动故障等级的审查，确保必要的工作和保障措施到位。此前提到了等级和 P 级，那么在蚂蚁内部，所谓的“序列”是如何定义的？“序列”究竟代表着什么？

故障序列是根据故障的根本原因和实际影响对故障进行分类的方法。关键在于区分“故障根因”和“实际影响”。有时，高稳定性团队可能会感到委屈，因为故障并非源于自身的技术问题，可能是由于产品运营的配置失误、网络运营商的抖动，或其他外部原因所致。因此，我们有必要将故障进行分类，明确技术故障与外部原因导致的故障之间的区别，并对其进行清晰的定义。

在定义故障序列时，主要参考以下几点：

1. 技术主要责任：首先也是最重要的一点，判断故障是否源于技术上的主要责任。如果故障确实因技术原因导致，那么可能需要扣减我们的故障预算。因此，我们主要关注 P1 到 P5 的故障等级，它们通常涉及技术主要责任，并对业务产生实际影响。

2. 故障记录与配额：另一个关键点是是否记录故障，以及如何通过故障配额（Quota）来评价我们每一年的稳定性表现。许多公司都有故障配额的规定，例如，一年内最多只能出现多少次故障，超出后将受到处罚，绩效可能会受到影响。因此，故障配额非常重要。

然而，有些故障并非技术原因引起，可能是业务操作失误，因此不需要扣减故障预算。对于外部原因导致的故障，我们定义为“W”；而由业务产品操作引起的故障，我们定义为“S”。每个公司都可以参考这些分类，制定自身的序列定义。

为便于在同序列故障在日后发生时能力进行更有效的故障应急，我们需要将部分序列所对应的指标纳入能力考察范围，并逐步提升相关能力，具体有以下四个指标：

响应原则

例如，我们常提到的“1-5-10”原则，即“一分钟发现，五分钟响应，十分钟应急止血”，尽管许多公司都在倡导“1-5-10”，但真正能够实现的公司并不多。因此，我们将目标适当延长，提出“30 分钟恢复”，为应急处理留出一些缓冲时间。

监控发现率

关于监控发现率，大家可以回想一下线上故障的情况，有多少故障是通过监控发现的？有客户投诉或外部反馈，但真正通过监控发现的故障有多少？根据我们的实际数据，监控发现率可能只有 60% 到 70%，这已经相当不错了。因此，我们将监控发现数量和监控发现率视为非常重要的衡量指标。

根因定位能力

那么，什么是根因定位能力呢？这指的是我们具备一种应急的平台能力，当线上出现故障时，能够快速且准确地找到问题的根本原因。许多公司都在开发根因定位的产品，蚂蚁内部也有一个专门的研发团队，负责根因定位平台的建设，他们的主要任务就是确保根因定位的准确性。

如何评价他们的工作效果？关键在于当线上出现故障时，根因定位能力是否能一击即中，准确地定位问题的根源，从而及时解决故障。

Action 完结率

Action 完结率，指的是在故障发生后，如何快速推进问题的解决，并将相应的行动项彻底完成。当出现故障时，能够迅速推进并完成修复措施，对于恢复系统的正常运行至关重要。

GOC 故障点



在这里，大家可能会有疑惑：我们公司有如此众多的业务，究竟应该保障哪些？我们必须优先保障最重要的业务，而不是面面俱到地试图保障所有业务。因此，我们定义了最重要、最关键的服务接口和结果点，将其称为 GOC（Global Operation Center）场景。只

要保障好这些 GOC 场景，我们的公司就不会出现重大的问题。并不是所有业务都需要全面保障，我们只需确保 GOC 的稳定性即可。

那么，GOC 是什么？我们通过三个领域来定义 GOC 场景：

1. 场景描述：明确说明需要保障的是什么。我们需要对所要保护的业务场景进行清晰的定义，明确其功能和作用。

2. 场景影响评价：确定需要达到什么标准才能证明工作效果良好。我们需要设定衡量指标，例如成功量、成功率、响应时间等，以评估场景的稳定性和性能。通过这些指标，我们可以清晰地了解自己的工作如何被评价。

3. 可触达的故障等级：明确该场景在出现故障时可能涉及的最高故障等级。如果某个场景即使完全失效，最高只会触发 P4 级别的故障，那么我们可能不会投入大量精力去保障它。但如果某个业务非常重要，稍有不慎就可能触发 P1 级别的故障，那么我们就必须重点保障。

实例说明

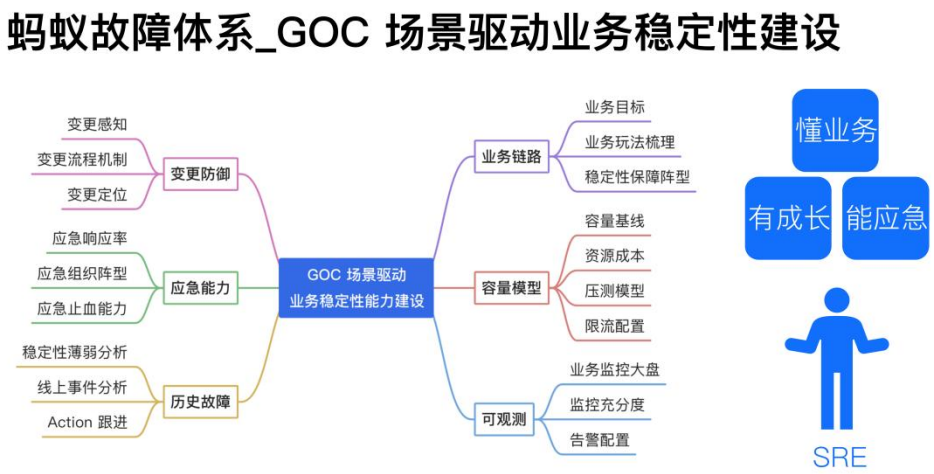
以登录功能为例，每个公司可能都有登录这一功能。我们需要思考在自己的业务中，是否对登录功能进行了明确的定义和保障。

免密登录：我们将成功量作为免密登录功能的稳定性衡量指标。将成功量视为评估登录功能稳定性的依据。如果成功量下降了某个百分比，就可能触发相应的故障等级。这些触发条件和对应的故障等级都可以被明确定义。

账号密码登录：对于账号密码登录场景，我们可能使用成功率来进行衡量。不同的场景采用不同的指标，这取决于其特性和重要性。

通过这些具体的数字和指标，我们可以清晰地定义每个业务场景的稳定性标准。尽管不同场景的衡量方式可能不同，但只要理解了核心理念，我们就能有效地保障关键业务的

GOC 业务场景驱动业务稳定性建设



GOC 场景定义的目的：

通过定义 GOC（Global Operation Center）场景，驱动业务稳定性能力的建设。

聚焦最重要、最关键的服务接口和结果点，确保公司整体稳定性。

明确业务链路和目标：

- 清晰了解所保障业务的核心目标和完整的业务链路。
- 明确稳定性保障团队，当问题发生时，能够迅速召集相关人员共同解决。

绘制容量模型：

- 从接入层到中间件，再到下游数据库，全面了解系统架构。
- 明确系统可承受的最大 QPS、日常流量和峰值流量。
- 描绘流量模型，识别不同时间段的流量特征（如早高峰、午高峰、晚高峰）。

全链路压测：

- 准备好压测所需的测试脚本和资源。
- 定期进行全链路压测，评估系统容量，确保在高负载下的稳定性。

配置限流措施：

- 根据容量模型，设置合理的限流策略。
- 防止突发流量超出系统承受范围，保障服务稳定。

完善监控体系：

- 明确业务的关键监控点和指标（成功量、成功率、响应时间等）。
- 建立实时监控和预警机制，及时发现和处理异常情况。

严格变更管理：

- 对可能触发 P1、P2 级别故障的核心业务，实施严格的变更管控。
- 建立完善的变更流程机制，具备变更感知和定位能力，确保变更安全可控。

加强应急响应能力：

- 制定应急预案，明确应急流程和责任人。
- 提升团队快速定位和解决问题的能力，确保故障发生时能够迅速恢复服务。

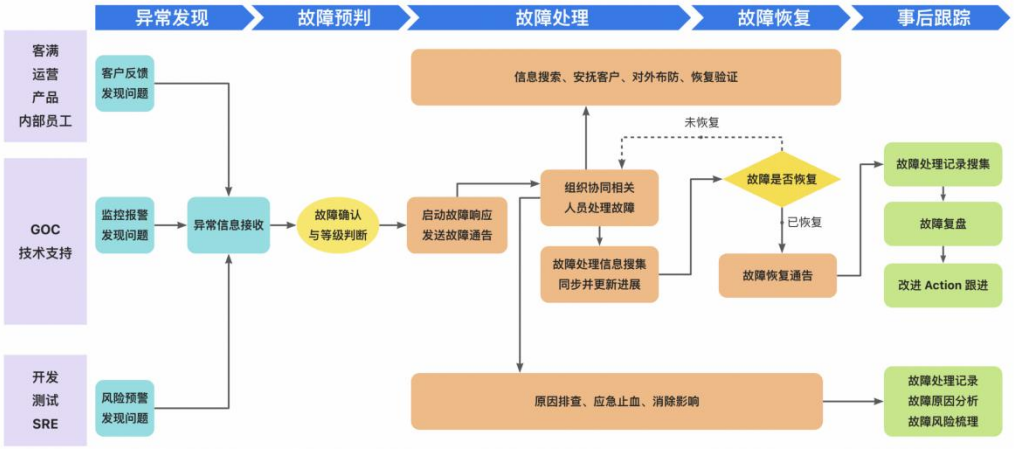
业务 SRE 的要求与成长：

- 深入理解业务：SRE 需要真正懂得所保障的业务，清晰其重要性和运行机制。
- 多领域能力提升：在压测、限流、容量管理、变更控制、防御措施、应急响应、监控等领域不断学习和成长。
- 具备全面的稳定性保障技能：能够统筹各项工作，全面提升业务的稳定性。

通过 GOC 场景实施稳定性保障工作：

将理论应用于实践，持续优化和完善稳定性保障体系。

蚂蚁故障体系_故障处理流程



当线上出现故障时，需要明确的处理流程和团队协作，SRE 无法单独解决所有问题。故障处理流程分为以下几个阶段：异常发现、故障预判、故障处理、故障恢复和后续改进。

故障处理的主要阶段和步骤

1. 异常发现：

- 通过内部客户投诉、监控告警异常、内部开发人员主动发现等方式，收集所有异常信息。
- GOC（Global Operation Center）设立异常信息接收机制，汇总并记录所有异常情况。

2. 故障预判：

- 基于预先定义的故障等级，GOC 对异常进行初步判断，确认是否属于故障。
- 确定故障的严重程度，决定发布何种级别的故障通告。
- 一旦故障确认并完成等级判定，正式启动故障应急流程。

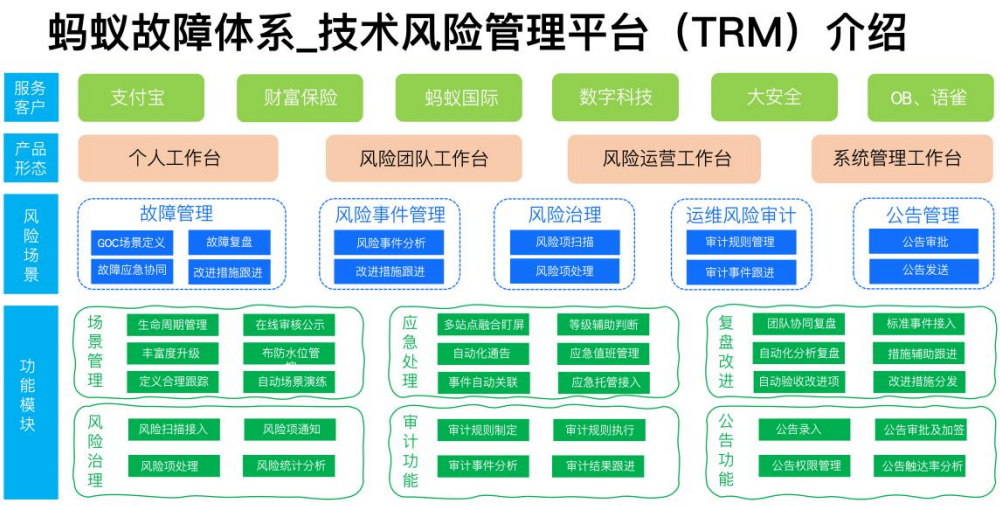
3. 故障处理：

- 产品运营：关注客户受损情况，评估故障对客户的影响。
- GOC：组织各方人员参与应急响应，协调故障处理工作。
- 开发、测试、SRE：快速恢复线上服务，查找问题根因，制定并实施应急预案或止血方案。

4. 故障恢复和后续工作：

- 故障验证：产品运营验证服务是否已恢复正常，客户体验是否恢复；GOC 确认各项指标恢复正常后，发布故障恢复通告。
- 问题根因分析：开发团队深入排查故障根因，总结问题产生的原因。
- 后续行动：制定改进措施和行动计划，跟踪落实，防止类似故障再次发生。
- 故障复盘：全面回顾故障处理过程，总结经验教训，完善故障处理流程。

平台能力支持



在完整的故障处理流程基础上，需要借助平台能力提升处理效率。平台工具和机制包括：

- 提供统一的异常信息接收和管理。
- 支持故障等级判断和通告的快速发布。
- 协助各团队高效协同，监控故障处理进展。
- 记录和跟踪后续行动项的落实情况。

我想向大家强调一点：在构建平台能力时，必须基于特定的场景。我们不是为了平台而做平台，而是为了解决具体的问题才去构建平台。当存在技术风险需要时，再去开发平台，而不是先搭建一个平台，然后再询问 SRE 或质量同学是否需要。因此，这一理念值得大家思考。

在蚂蚁这边，我们的技术风险管理平台（TRM）面向各个业务部门，客户基本涵盖了蚂蚁的所有 BU，如支付宝、国际大安全等，甚至包括已经独立的 OB、语雀等。他们都可以在这个平台上进行操作。产品形态主要服务于普通的 SRE 同学以及业务的接口人。稍后我会介绍我们的整体应急阵型。针对不同的角色，我们提供不同形式的工作台。一些 GOC 的同学可能会在系统中录入 GOC 故障定义的场景，因此我们有风险运营的机制。

这个平台具备哪些功能？最重要的功能包括：

故障管理： 主要是指前面所提及的故障场景、故障定义等信息用平台沉淀起来，同时通过故障复盘、Action 跟进之类的的管理能力整体形成功能闭环。

风险事件管理： 风险事件是指未被定义为故障、影响较小的业务事件，可以归类为风险事件。

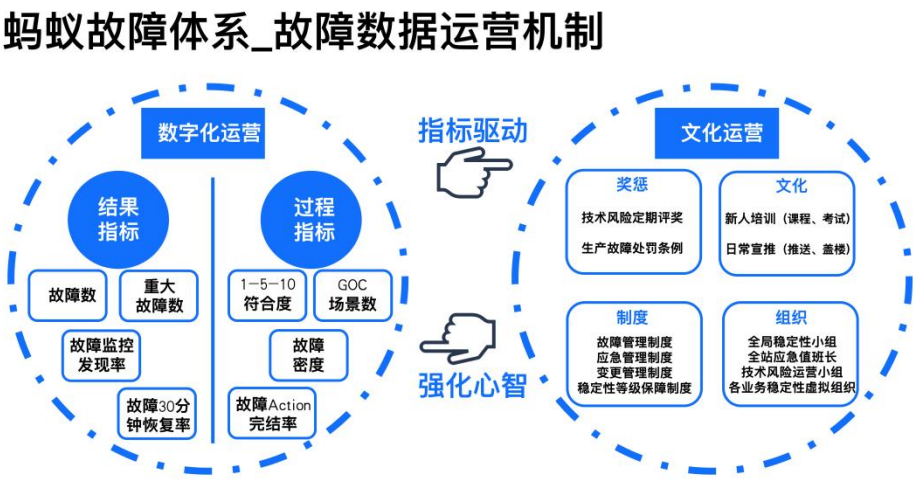
风险治理： 主要是集中处理一些风险巡检系统的风险产出物，业务稳定性同学可以基于风险项信息进行风险的跟踪与治理。

运维风险审计：蚂蚁历史上曾出现过由于运维操作不合规导致的线上问题，这在各家公司都可能发生。因此，我们需要收紧重要权限，如果出现违规操作，需要进行审计。

公告管理：平台提供公告功能，用于发布重要的稳定性信息。

其他功能模块，如场景管理、审计功能、复盘改进等，都是为整个故障生命周期和各种风险场景提供服务。

故障数据运营机制



那么，我们制定这些工作的目的究竟是什么？累积故障数据的意义何在？在此，我向大家介绍一下：我们希望通过数字化运营，将故障水平提升至一个相对较好的状态。为此，我们会定义一些结果性指标和过程性指标。

结果性指标方面，例如故障数量，这是非常直观的。今年，我们必须控制故障数量在一定范围内，若出现过多故障，就可能存在问题。对于重大故障而言，如从 P1 到 P5，我们更关注 P1、P2 级别的故障数量。由于 P1、P2 级别故障的影响面较大，我们需要设定指标，将其控制在合理范围内。

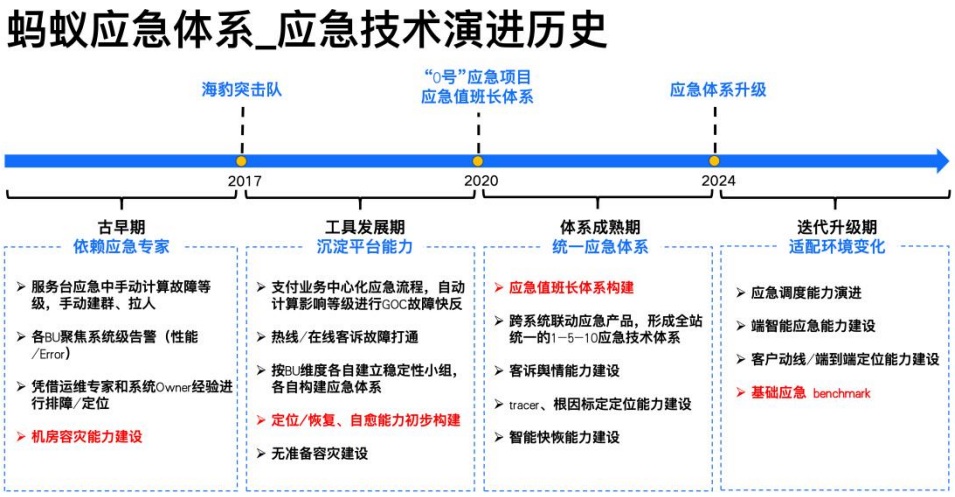
此外，还有故障监控率、监控发现率、30 分钟恢复率以及平均修复时间（MTTR）等指标。这些结果性指标直接影响我们对稳定性的评价和绩效考核，是整体评价体系的重要组成部分。

过程性指标方面，当我们定义好这些故障场景且线上出现问题时，我们的 MTTR 是否满足要求？故障处理措施的完结率达到了怎样的水准？例如，去年我们有许多故障处理措施超期，今年能否及时完成？通过数字化运营，结合过程指标和结果指标，我们可以生成一份全局稳定性的故障数据报表。利用这份报表，我们能够在月度或半年度开展各类评审，对我们的稳定性工作进行评价。

在文化运营方面，我们制定了相关的奖惩措施、文化理念和制度等。原本下一页是要介绍我们的惩罚制度的，但来之前老板提醒我不要提及，担心会影响招聘。因此，这一部分就省略了。总之，我们的奖励机制非常完善，但若触犯红线，必然会受到相应的惩罚。毕竟，从事稳定性工作的同仁，谁没经历过几次故障呢？

然而，我们通过数据与文化的相互促进，利用数字化运营拓展出各类指标，定义好指标后，让我们的文化起到驱动作用。同时，借助奖惩措施、文化制度等一系列手段，加强稳定性团队的意识，使他们真正认识到稳定性工作的重大意义，明确这是值得付出和投入的事业。

蚂蚁故障应急体系



在之前的内容中，我向大家介绍了故障的相关概念，接下来，我将讲解故障应急体系。

首先，让我们来看一下应急工作。蚂蚁集团的应急体系已经建立了很多年，从 2017 年开始至今，每个阶段的重点各不相同。

最初期，我们称之为“古早期”。在这个阶段，我们主要依靠一些非常优秀的应急专家。当线上出现问题时，他们会第一时间进行处理，没有明确的职责边界，也不存在推诿的情况，他们的目标就是迅速解决问题。然而，这种方式工作量巨大，压力也非常大，因此我们意识到需要建立一个梯队化的体系。

当时，这些应急专家最重要的任务是完成机房级别的容灾建设，之后线上出现的问题，基本都由他们负责处理。

从 2017 年到 2020 年，我们开始构建各种平台。我们意识到，有些问题不能仅靠人力解决，必须依托平台。因此，我们沉淀了多

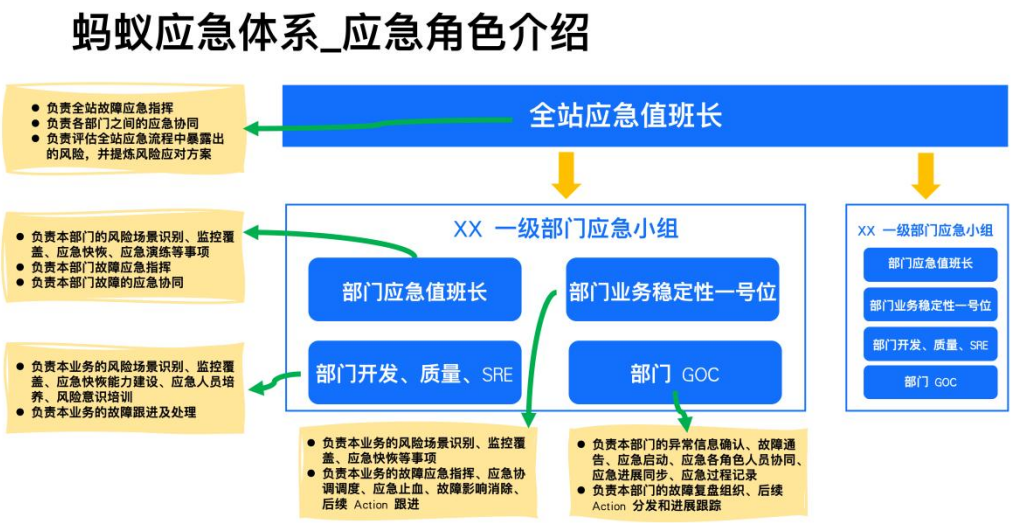
种平台能力，推出了第一版的定位恢复和自愈功能。这一时期被我们称为“工具发展期”。

在 2020 年，我们正式成立了应急值班长组织，稍后我会重点介绍。从那时起，随着应急值班长体系的构建，线上问题和故障出现了断崖式的下降。原因在于我们明确了谁应对线上稳定性负责，谁需要采取哪些措施以快速恢复问题，并设立了相应的评价指标。

因此，我们在那个阶段实现了跨系统的应急产品联动，形成了全栈的“1-5-10”标准。同时，我们也拓展了其他能力，例如客户投诉、舆情监控，以及智能会议功能。

到了 2024 年，也就是今年年初，我们的团队对 GOC 和稳定性组织进行了升级。为了适应整体的变化和发展需求，我们演进了一些精细化的应急能力，例如应急调度能力、终端智能的应急能力，以及基础应急的基准测试。这意味着，单有平台是不够的，我们还需要评估工作实际效果，确保应急体系的有效性。

蚂蚁应急体系应急角色



前面提到应急角色的重要性，那么具体来说，蚂蚁集团的应急体系有哪些角色？我们主要分为两个级别，五种角色。

第一个级别是全站的应急值班长，这是一个至关重要的角色，负责全站的整体应急指挥。全站应急值班长的职责是协调各部门在线上出现问题时的应急协同，及时揭示重大风险，保持全局掌控，并推动应急流程的执行。他位于应急指挥体系的最高层。

第二个级别是部门级，不同部门有各自的稳定性保障人员，包括部门的应急值班长、开发质量同事、以及各业务线的应急人员。具体而言，部门内会设立一个统一的应急小组，由部门应急值班长负责。在每个业务线下，配置专门的稳定性一号位。部门应急值班长的任务是识别并监控部门的风险场景，构建监控覆盖、告警触发、应急响应等能力。稳定性一号位则专注于其所负责的业务，与部门应急值班长形成上下游的合作关系，不是上下级，而是互为支持的关系。

在每个部门中，GOC 同事（定义部门级 GOC 场景的技术支持人员）负责在现场出现问题时快速响应并激活应急措施，确保部门的应急流程顺利进行。

这样，每个部门的应急小组与全站应急值班长紧密联动，形成了一个完整的应急体系。回答“谁为线上稳定性负责？”的问题，这些角色和他们的协作体系就是蚂蚁线上稳定性的保障核心。

应急值班长体系的建立经历了一段曲折的发展。早在 2019 年，蚂蚁集团就成立了全局稳定性小组，目标是避免重大线上故障。然而，当时的组织结构是虚拟的，没有正式的值班长角色。这种虚拟组织虽然能够提供初步的稳定性支持，但尚未达到体系化的水平。



过去提到，虚拟组织在明确“谁负责线上稳定性”方面存在一些弊端，因为各部门都有各自的能力，但职责不够清晰。为了解决这一问题，2020 年，蚂蚁集团正式向全公司公开了应急值班长制

度。我们的目标是培养蚂蚁的技术人员具备全面的应急能力，包括“应急止血”和“应急组织”两大核心能力。因此，我们建立了全局稳定性的应急值班长机制，但并不是每个人都可以担任此职，需具备以下几点：

- 组织力与决策力：应急值班长必须具备出色的组织和决策能力，这意味着新人通常难以承担这一角色。
- 实战经验：值班长需有丰富的线上应急实战经验，并在处理线上故障时展示出优异的效果。

在选拔上，我们有一套标准流程（具体细节略），但必须提到的是激励机制。被评选为应急值班长的员工将获得蚂蚁 CTO 的授权和象征性的荣誉牌，并在全公司进行宣传。获得这一称号不仅意味着表彰，也要求承担更大的责任。例如，值班长需对线上问题保持 24 小时的响应准备——责任越大，激励也相应更大。

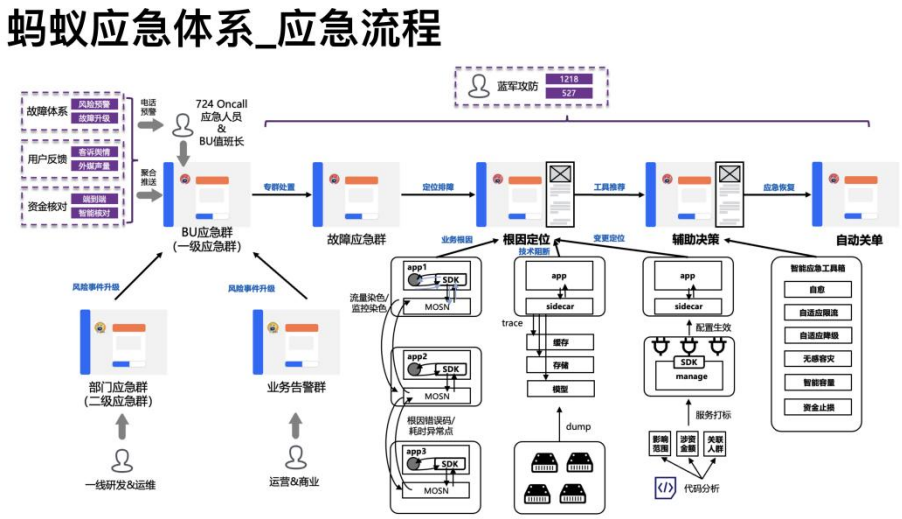
应急值班长的职责十分严峻，需要具备线上快速止血和组织应急的能力。线上问题通常涉及多个角色和部门，不仅是值班长的个人职责，而是一个全面的协同任务，涉及产品运营、GOC、开发质量、SRE，甚至公关、风控和安全等多个团队。因此，全局值班长的应急组织能力至关重要。

值班长需具备三项关键素质：

- 专业度：在稳定性工作中展示卓越的专业能力；
- 认同感：对稳定性工作有高度的认同，愿意为其付出努力；
- 责任心：始终如一地承担责任，不能因个人情绪忽视告警响应。

这些素质确保应急值班长能够高效推动线上问题的快速解决，并协调各方力量维护系统稳定。

应急流程

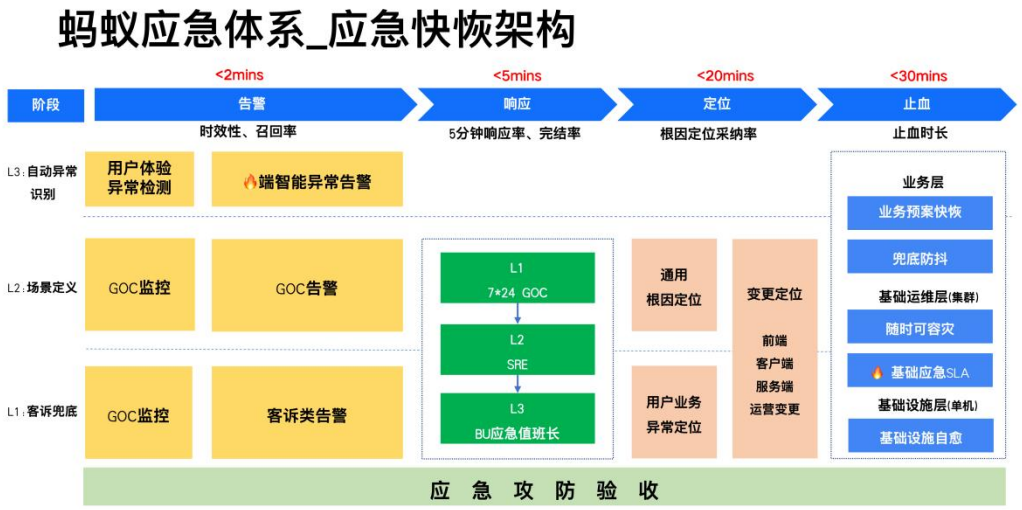


接下来具体讲解一下蚂蚁集团的应急流程。虽然前面已经介绍了故障处理流程，但这里特别提出应急流程，目的是展示我们在应急响应中的平台能力。

在应急启动之前，我们已经建立了多种感知渠道和能力。例如，线上会有电话预警、预案推送，或是各部门的应急群开始处理预警信息。这些预警阶段尚未被定义为故障，但我们会预先组建好业务单元的应急群，以便在真正的故障发生时迅速拉起全局的故障应急群，确保统一协调。

故障应急群建立后，我们依赖根因定位团队第一时间锁定故障根因，虽然定位可能不完全精准，但我们有明确的要求和标准来提升根因定位的准确度。同时，我们还提供辅助决策的能力，以确保在关键决策时有足够的数据支持。

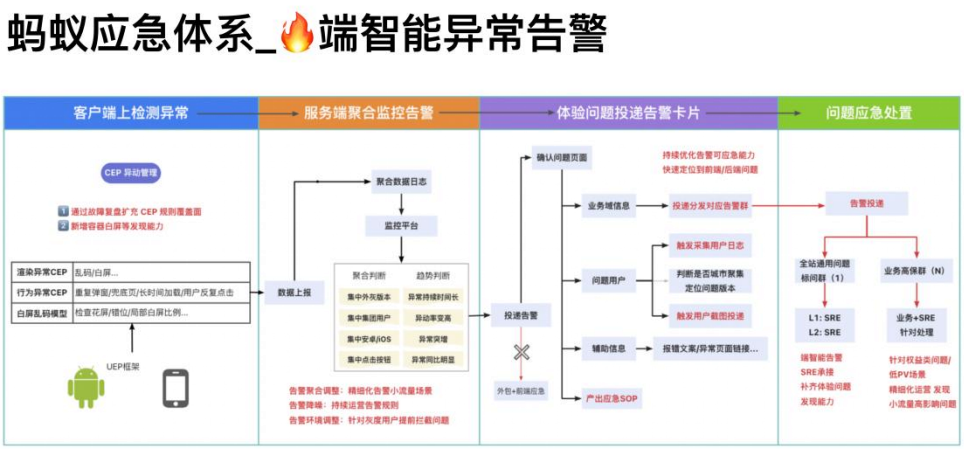
在一些情况下，系统会自动完成工单关闭，这样的自动化处理进一步优化了应急流程。通过这些平台能力和流程，应急处理流程变得更加高效和可控，实现了从预警到故障处理的全流程闭环管理。



关于应急快速恢复，蚂蚁集团的关键能力聚焦在不同阶段的核心指标。每家公司都会设立类似的应急时间节点：例如“一分钟响

应，五分钟监控响应，三十分钟止血恢复”。在这些阶段，我们需要明确具体的能力评价指标。

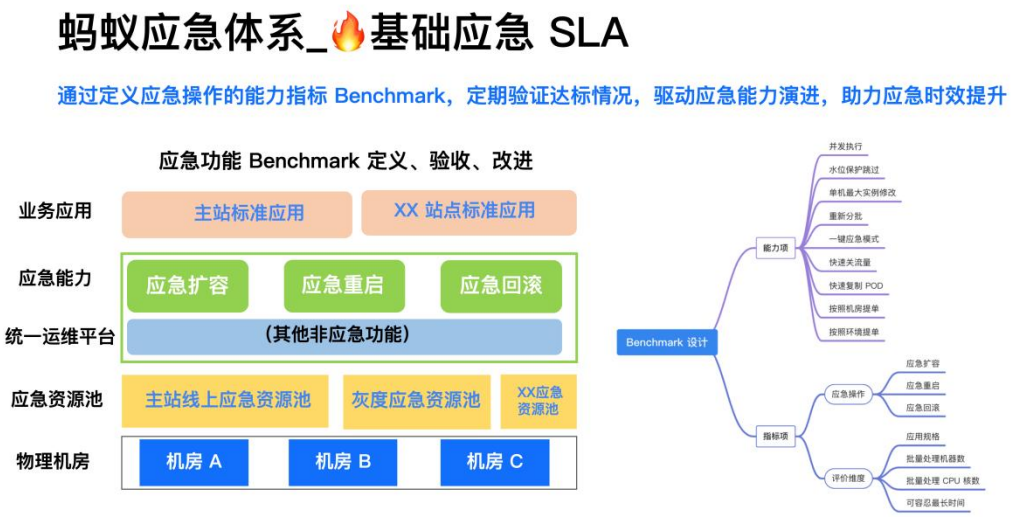
这些评价指标正是前面提到的各种运营指标，确保每一项能力都能被有效衡量。若没有清晰的指标，能力的执行效果将无法得到客观评价。换句话说，每一项应急能力必须有与之对应的指标，以便明确判断工作的完成质量和效果。这种指标体系不仅验证了应急能力的有效性，也为进一步的优化提供了依据。



我们最近新增了一项关键能力——**端智能异常告警和应急SLO**。这一端智能功能能够及时感知客户端的异常状况，例如白屏、黑屏、重复弹框等异常情况。客户端一旦出现这些异常，我们

的系统可以立即感知并迅速将异常聚合到服务端进行分析和决策，判断是否属于实际问题。

如果确认存在问题，系统会自动拉起故障处理群组，并将该事件接入故障处理和应急流程，形成关联。客户端上传异常后，服务端聚合告警并生成问题卡片，从而启动应急响应。目前，这项能力已经帮助我们发现并响应了许多线上问题，极大提升了故障的感知和快速响应能力。



在基础应急管理中，我们需要定义和构建多种应急能力，以确保在面对故障时能够快速响应和处理。例如，我们会设定应急扩容、应急重启和应急回滚等能力，确保在平台出现问题时能够有效应对。

为此，我们构建了不同的应急池，赋予这些应急能力具体的执行要求。这些要求包括：

- 并发能力：确保系统能够处理多个请求，以应对突发流量。
- 重新分批能力：能够灵活调整请求处理策略，确保服务不中断。
- 一键切换应急模式：在发现问题时，能够迅速切换到应急模式，降低潜在影响。

我们还会设定具体的指标来衡量这些能力的有效性，并进行定期验证，以确保这些应急能力在实际使用中能够发挥作用，确保系统的稳定性和可靠性。通过这样的措施，我们能够有效降低故障对业务的影响，提升整体应急响应能力。



在评估应急管理效果时，我们可以使用标准化的评价指标。这些指标包括：

- 发现时间：故障是在多长时间内被发现的，是否通过监控系统发现。
- 响应时间：从发现故障到采取行动的时间（5 分钟响应）。
- 止血时间：止血（初步解决问题）的时长。
- 定位信息：是否有定位卡片及相关信息，帮助识别故障根因。
- 告警情况：是否触发了端智能告警等。

我们要求 SRE 团队在故障发生后 48 小时内完成应急复盘，这个复盘专注于评估应急响应的有效性，而非故障的全面分析。通过复盘，我们可以明确在应急过程中做得好的地方和需要改进的环节。



应急止血的方式可以因公司而异，常见的手段包括：

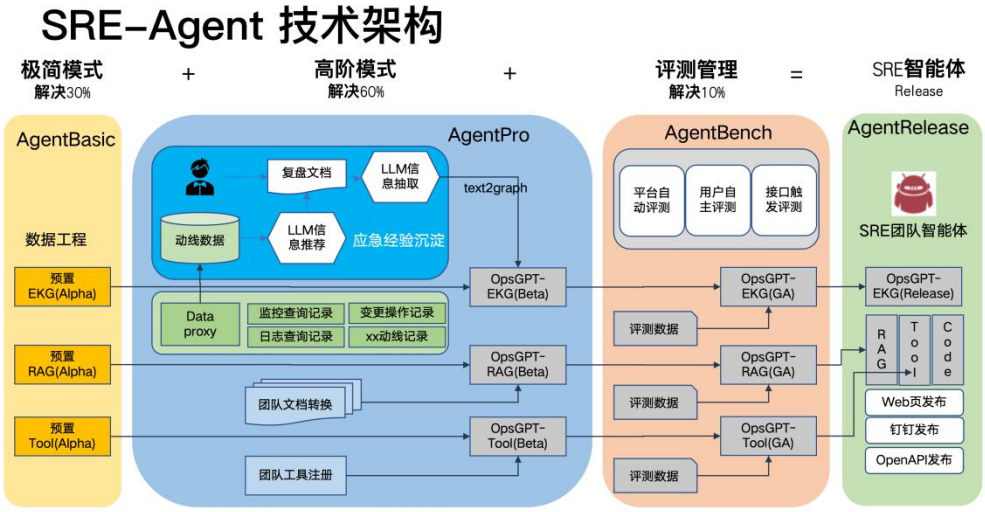
- 业务层面的操作：快速调整业务策略以减轻影响。
- 基础操作：如重启服务或调整资源配置。
- 容灾切流：在出现问题时快速切换到备份系统。
- 变更管理：快速回滚至稳定版本。

这些平台能力支持我们在出现故障时快速响应，并减少业务中断的影响。通过系统化的评估和复盘，我们能够不断优化应急管理流程，提高整体的稳定性。

（三）总结及展望

AI 助力，未来已来

04 AI 助力，未来已来



未来应急能力与 SRE Agent

1. 平台能力的演变

未来的应急体系将依赖于平台能力的提升。

SRE Agent 的协作：

- 提供预置知识，支持基本问题解答（如应用调用量、机房部署等）。

2. 高阶能力的融合

- 应急复盘和相关文档的整合：
    - 通过 OPS、GPT 等技术提升应急复盘能力。
    - 发展为 Agent Pro 模式，能够解决约 60%的问题。
- 剩余问题的处理：
    - 需要将 Agent 交付给 SRE 团队使用，收集反馈并持续优化，以解决剩余的约 10%问题。

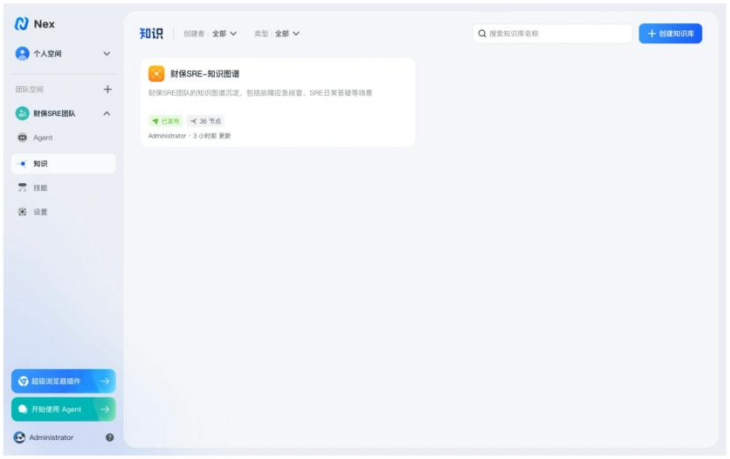
3. Agent 模块整合

- 包括 Agent Basic、Agent Pro 和 Agent Bench 三个模块。
- 最终形成 Agent Release，整合 SRE 团队的智能体能力。
- 当前版本已在内部使用，具备多种平台及应急能力。

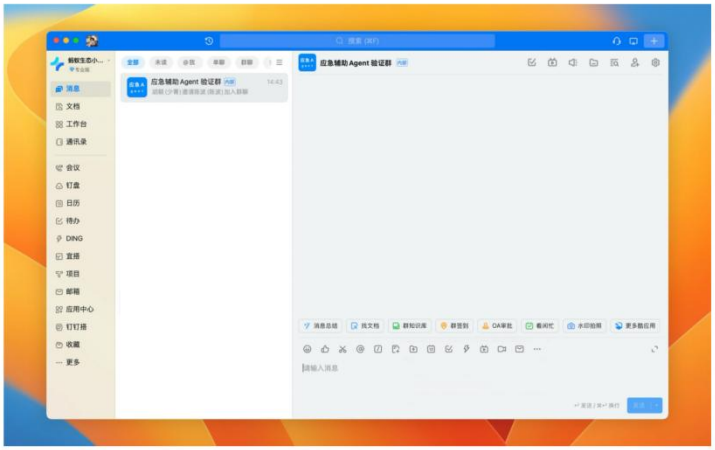
SRE-Agent 页面演示



SRE-Agent 构建演示



SRE-Agent 使用演示



1. 知识导入

利用知识图谱系统，导入相关的知识和信息。

将导入的知识整理为可视化数据，便于后续处理。

2. 知识可视化

生成可视化图形，展示导入知识的结构与关系。

知识经过编排后，形成逻辑关联，便于理解和应用。

3. 机器人关联

将编排后的知识信息与机器人系统进行关联。

机器人能够调用这些知识进行实际操作与决策。

4. 决策过程

机器人根据用户输入的信息进行初步决策：

若结论正确：

维持当前决策，执行相应的操作。

若结论错误：

触发学习机制，重新分析相关知识。

调整决策逻辑，更新知识库，以优化未来的决策。

5. 大模型能力体验

当前正在进行大模型能力的实际使用，收集反馈以改进功能。

鼓励团队成员关注和参与 SRE Agent 团队的进展与反馈。

6. 集成到钉钉

将 Agent 系统集成到钉钉平台：

生成应急卡片时，机器人自动分析潜在问题。

识别异常情况，并提供根因分析。

输出相关数据库和监控指标的实时反馈，便于团队快速响应和处理。

7. 应用反馈

在实际使用中不断收集反馈信息，优化机器人决策流程。

根据用户体验和问题解决情况，进行持续的功能迭代和更新。

5.3.8 xx 银行应急提升实践：深耕 1-5-10 工程

SRE Elite 精选原因：

该案例以“1510”工程为牵引，从组织、流程、工具三维度提升金融级故障应急韧性。通过横纵三层技术支持、一分钟监控感知、五分钟节点级定界、十分钟应急六板斧处置，实现端到端闭

环；配套早例会、复盘、治理、体验等机制沉淀知识并驱动持续改进；值班经理八大修养与跨部门协作文化保障指挥效率；最终形成可预见故障场景及一键应急平台，将专家隐性知识自动化，故障诊断秒级完成，为大中型股份制银行构建高效、可复制的稳定性保障范式。

（一）背景及设计原则

金融业的稳定性保障所面临的挑战正日益严峻，影响稳定性的核心挑战可以概括为三个相互关联的方面：

监管新常态：合规性和稳定性要求不断收紧，不仅要求严格、具体还有演练实操。例如，针对高级别故障需要向监管机构汇报并闭环落实改进措施落地情况、现场和非现场监管检查以及组织各金融机构开展的“不打招呼、令下即切”极限压测演练。

业务高要求：随着科技与业务的深度融合，不仅业务部门，科技部门也深刻感受到系统稳定性已成为保障一线业务展业和触达客户的基础，对系统停业时间（无论在计划内，还是计划外）的容忍度都不断降低。

技术（大变革）：以 XC、容器、大模型等为代表的新技术架构和应用层出不穷，叠加金融业固有的技术架构多样化痛点，运维规模、复杂度急剧增长，配套的管理要求也越来越高，这与现有运维人员的数量、能力增长速度是不匹配的，进一步增加了稳定性保障的难度。

在上述三重挑战下，任何内部微小的跑冒滴漏都可能迅速放大，导致稳定性天平失衡，引发严重问题。

为了积极应对这些挑战，并争取更具韧性的业务运营环境，我们集结了数据中心、研发中心和科技部门的力量，共同启动了“1510”工程。这项工程包括“主动预防”和“快速应急恢复”两个分支，核心目标是：1 分钟发现、5 分钟定界、10 分钟应急，并以此牵引数据中心整体运维能力的全面提升。接下来要介绍的是“1510 工程”关于快速应急恢复这个分支。

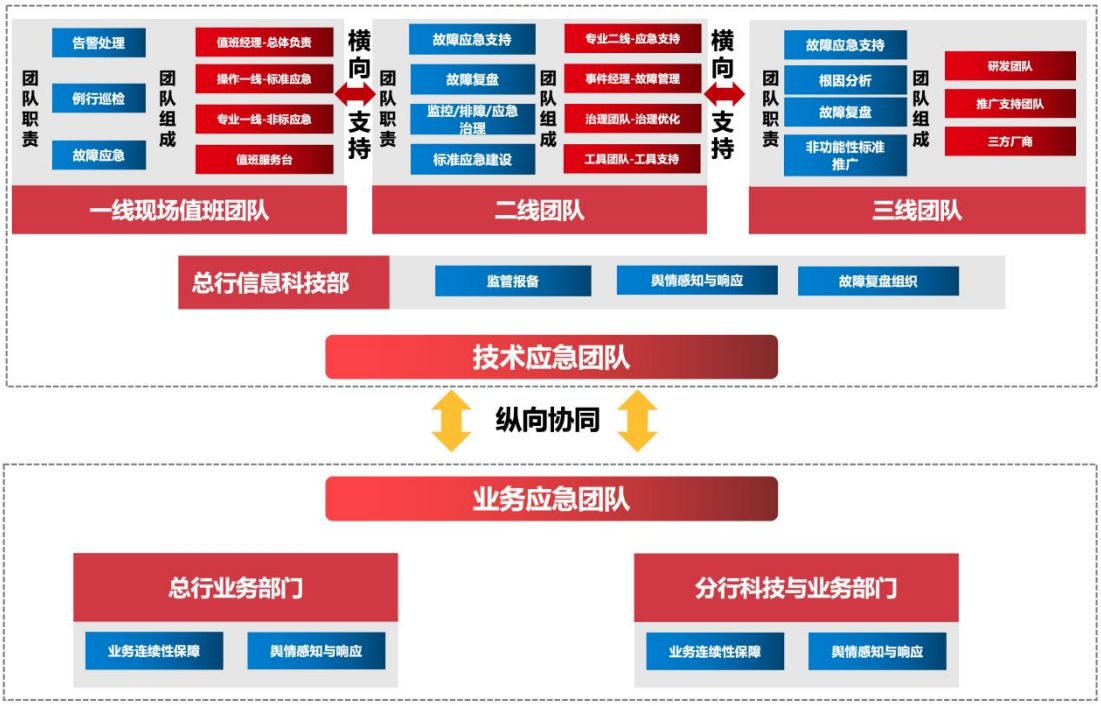
注：1510 代表 1 分钟故障感知能力、5 分钟故障定界能力（包含故障感知时间）、10 分钟完成应急处置，总体应急恢复目标为 15 分钟。

（二）解决方案

组织能力提升

“组织能力”的提升，这不仅是我们迈向持续优化运维体系的第一步，更是应对未知挑战、确保系统稳定运行的基石。

应急组织架构



核心目标：构建高效应急响应体系，力求“1 分钟发现、5 分钟定界、10 分钟恢复”

多层级横向技术支持：我们的组织架构采用“横纵”结构，构建了三级技术支持体系：

一线团队：作为应急处置的主力，负责告警处理、例行巡检及故障应急。其组成为：

值班经理：作为应急的总体负责人，全面负责应急指挥。

操作一线：负责标准应急操作的执行，例如，可预见故障场景的应急。

专业一线：针对非标准化故障，进行专业处置。

二线团队： 为一线团队提供支持与增援，协同处理一线团队无法独立解决的复杂故障。

三线团队： 提供更深层次的技术支持，通常包含研发、推广支持及三方厂商等资源，应对疑难杂症。二、三线团队除了应急支持，日常还承担标准应急流程建设、治理、应急演练落地以及非功能性需求的落地等职责。

背景说明：通常情况下，银行的生产环境和办公环境、研发测试环境之间是严格限制网络访问的，因此，所有生产环境的运维操作，都只能生产区终端执行。通常，银行会设置生产技术支持区、ECC（企业总控中心），其中，ECC 是现场值班应急的场所。

横纵应急协同：构建技术与业务的紧密协同机制，包括技术应急团队与业务应急团队的联动，总行数据中心、研发中心、信息科技部与总行业务部门、分行科技与业务部门的协同，确保在应急事件发生时，技术与业务能够快速对齐目标，形成处置合力。

应急四大原则

为了支撑高效的应急响应，我们遵循以下四大核心原则：

①第一时间通知：

对客服务的联机交易类故障必须第一时间口头通知值班经理和专业组长，然后再处理；重大故障值班经理电话通知处室经理、中心负责人。

②优先恢复业务：

故障应急应以第一时间恢复业务服务为首要目标，业务恢复后再处置账务调整、数据恢复等问题。

③并行现场应急：

值班经理决策启动应急程序，组织值班人员、相关人员现场应急；专业组长第一时间到 ECC 现场应急支援；相关专业条线并行现场应急，相互联动，必要时与业务并行应急。

④30 分钟恢复（15 分钟恢复为牵引目标）：

联机交易类事件必须力争在 30 分钟内完成应急处置，恢复业务服务，同时提出了 15 分钟的牵引目标，鼓励团队持续提升应急响应与恢复的速度。

应急管理运营机制



应急管理运营机制：重复、持续地通过早例会、故障复盘、治理、体验等机制落地应急管理要求，同时，统一每个人的故障应急认知、意识和行为。

对应急管理的持续改进与落地，我们构建了一套系统的运营机制，核心目标是让应急管理的要求能够渗透到日常工作中，并以此统一团队对于故障应急的认知、意识与行动。这里介绍其中 4 个机制：

早例会机制： 由数据中心、研发中心和科技部门共同参与，会议核心内容包括前一天的故障、变更问题、治理问题复盘，以及对已完成的故障整改任务进行评审。会议由数据中心的各级负责人轮流主持，谁主持、谁发问，例如，监控是否做到了比监管、银联更快发现，故障定界是否精准，应急是否使用了工具，有没有规范没落地的地方，每个团队在应急中都做了什么。这种跨部门协同与负责人轮动主持的方式，除了复盘和评审，还有 3 个作用：一是，通过主持的形式，可以强化各层级的负责人对应急要求的主动思考和反复训练；二是，有句话说的好，别人的失败，才是自己真正的成功之母，当各层级负责人对别人进行灵魂拷问的时候，他就会下意识地主动思考自己团队内是否也存在类似“灯下黑”的问题，有利于真正做到问题的举一反三；三是，早例会还是针对治理问题快速决策、拍板整改方案的高效平台，显著提升了治理工作的落地效率。

故障复盘机制： 针对五级以上（有业务影响的）故障以及其他具有典型意义的故障，由科技部门牵头组织进行深入复盘。通过标准化的复盘模板，系统性地分析故障发生的根本原因、影响范围，并制定详细的改进措施。这确保我们将每一次故障都转化为宝贵的经验，避免类似问题再次发生。在复盘章节将对此机制进行更详细的阐述。

治理机制： 我们组建了全职的治理团队，同时，每个专业团队配备了对口的治理整改负责人。团队围绕“不漏报、少误报、快处

理”的核心目标，持续优化工具、组织和流程。目前，我们的故障监控发现率稳定保持在 93%至 96%之间，日均告警量控制在 400 以内，平均每小时约 16 条，分摊到各专业条线后，每条线每小时约处理 3-5 条告警，基本实现了故障不漏、人能处理。治理日报和周报作为重要的过程总结和汇报形式，推动了治理工作的持续演进。

操作一线体验：这是一个颇具特色且深受欢迎的机制，我们要求工具平台团队和流程管理团队每周安排一人到操作一线体验一天，亲身体验监控告警的处理、应急操作的流程以及日常巡检等工作。通过这种方式，工具和流程团队能够直观感受一线同学在故障应急过程中的痛点与需求，收集最直接的反馈，为工具和流程的优化提供生动、真实的素材，从而更好地支撑一线应急工作的开展。这种接地气的体验，极大地促进了跨团队的理解与协作，确保了交付的工具和流程真正符合实际操作场景的需求，最终提升整体应急响应效率和体验。

一线现场值班机制



一线值班优化：标准化一线应急响应标准及流程

首先，在**值班协同**层面，我们着重对人员组织、工作职责及流程进行了精细化调整。我们对值班角色及其工作职责进行了重新定义，并配套建立了相应的绩效评估指标，例如，一分钟告警响应率、一线故障应急处理比例，以此牵引团队行为。同时，我们构建了一套面向生产环境的标准化应急响应流程，并特别优化了一线与二线、三线团队之间的协同工作流程，强调一线应急协同问题的闭环管理，确保问题得到彻底解决。为了支撑这些改进，我们从各专业团队挑选了专业人员，补充加入到一线团队。

其次，从**技术平台**的角度来看，我们构建一系列技术工具来辅助值班与应急响应工作。我们建设了应急协同平台，以加速故障发生时的指挥调度效率。此外，我们还提供了故障定界工具，使得团队能够快速准确地定位问题根源。更为重要的是，我们建立了通用的“应急六板斧”，为一线团队提供快速实现故障恢复的有效手段。

最后，**值班赋能**是确保一线团队具备足够应急能力的关键环节。我们建立了一套行之有效的应急移交机制，将经过验证的故障场景应急处理能力系统性地传递给一线值班团队。一方面，可以提升一线直接应急比例；另一方面，也解放二、三线节假日、夜间应急压力，二、三线在标准应急方面的投入，也转变成了收益。

值班经理能力培养-值班经理的八大修养



作为故障应急的核心人物，值班经理是应急指挥中的 C 位担当，是接棒第一人、坐镇指挥官，也是信息枢纽站、关键决策统筹人。我们根据应急指挥实战经验，**总结出了值班经理必须具备的八大修养**，这些修养同样重要，没有先后之分：

要有**有担当，有责任心**。在故障发生时，要主动响应，亲力敢为；

要具备**把控全局的能力**。能够在短时间内调度、协调多方，特别是场面混乱的时候，要能及时综合统筹，发现问题及时纠偏，要能控场、也要有多线程处置能力；

要具备良好的**沟通协调能力**。能够在紧急情况下有效沟通，能快速提炼和精准传达故障信息，确保各方理解一致；

拥有**果断决策能力**。以恢复业务为第一目标，盯紧应急时间线，关键时刻拍板决策应急处置方案。

要做到**知过去，通过程**。值班经理应熟悉过去几年的重大和经典故障的处理过程及经验教训；

具备一定的**技术判断力**。了解生产环境的技术架构，有各个技术栈的基础判断能力；

熟悉当前环境。熟悉当天值班人员、各领域的核心人员和关键角色，还要熟悉事、熟悉工具；

最后，要有强大的**压力管理能力**。在面对重大故障的时候，如果 CIO 就站在你的身后，值班经理是否能够依然有序、有效地组织故障应急。

值班经理像一位技艺精湛的杂技演员，需要同时掌控八个球，并确保它们都能平稳运行。

应急文化建设

应急文化塑造：让各团队精诚合作、聚焦目标，做难而正确的事。在前面已经详细探讨了各种应急机制与流程，正是这些具体的

规定与要求，逐步统一了我们对应急的认知、意识与行动方式，我们的应急文化也是通过这些机制去逐步塑造的。

我们的应急文化被构建为清晰的四个层级：



担当履责

- 要求每一位 SRE 工程师都拥有“守土有责，自扫门前雪”的主人翁精神。
- 具体而言，每个人对自己所负责保障的范围，要能做到监控、应急不跑冒滴漏，守好自己的“责任田”。

知行合一

- 在 SRE 领域，仅仅“知道”是远远不够的，真正难得的是将知识转化为行动，并做到极致。
- 不仅需要掌握广泛的技术知识和领域专家知识，更要对内部的规

范、标准了然于心，要有作为兜底把关人的思维，做了还要确保做到、做好，不要在复盘的时候拍大腿，例如，有监控，但监控是否完全落地了标准监控；架构是双活的，但业务是否对称双活；有应急场景，但应急是否演练验证过、并移交一线。

自我驱动

- 鼓励自我驱动的 SRE 文化。
- 每个团队的骨干，应该总有一两个痛点、难点挂在心头，日思、晚想、夜梦如何解决。
- 鼓励塑造个人的技术品牌与影响力，并运用这种影响力去推动和解决那些看似棘手的难题，形成一种自发的、持续的改进动力。

团队协作

- 在组织层面，消除了部门、团队之间的隐形墙，集中三个部门的力量，做成“应急”这一件事。
- 这是一种共同体意识，正如“成时举杯相庆，败时拼死相救”所描绘的那样，无论是在取得成绩时共同庆祝，还是在遭遇挑战时相互扶持，所有人都并肩作战。
- 在故障发生时，我们不仅要求各团队能够快速自证清白，排除自身嫌疑，更要拥有一种“拼死相救”的强烈主动意识和全局观，站

在更高的维度上思考如何协助其他团队，共同渡过难关。

借用一句话总结，对于稳定性保障这项系统性工程，**组织能力是放大技术杠杆的关键生产力。**

分钟级响应能力

我们在应对系统故障时所追求的核心目标：高效、迅速地发现、定位并处理问题。在 SRE 实践中，实现分钟级的响应并非易事，它需要一套完整的体系来支撑。



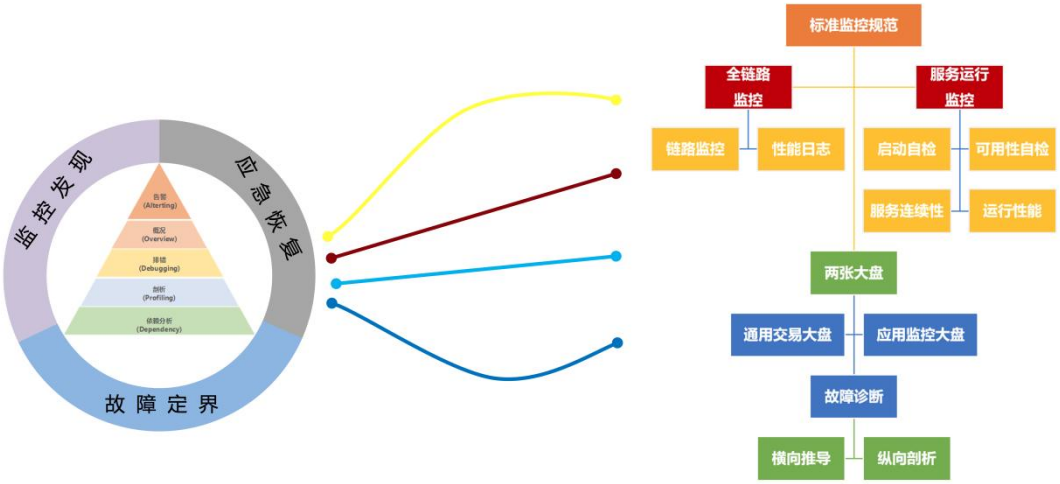
如何实现“1”

首先，是感知快。确保所有关键、重要的服务都能在一分钟内被准确感知到其状态或故障。

其次，是响应快与协同快。这两者相辅相成，工具平台和治理工作要确保能支撑一线快速聚焦重要告警、能为一线提供线上线下协同工具。

最后，是量可控。要在不误报、不漏报的前提下，确保告警的数量是人有能力响应、处理得过来的。如果告警泛滥、事件堆积，即使具备了快速感知和响应能力，也会因为工作量过载而无法实现“1 分钟”的目标。最后这一项是治理工作范畴，这次不展开讲。

发现更快-1 分钟感知



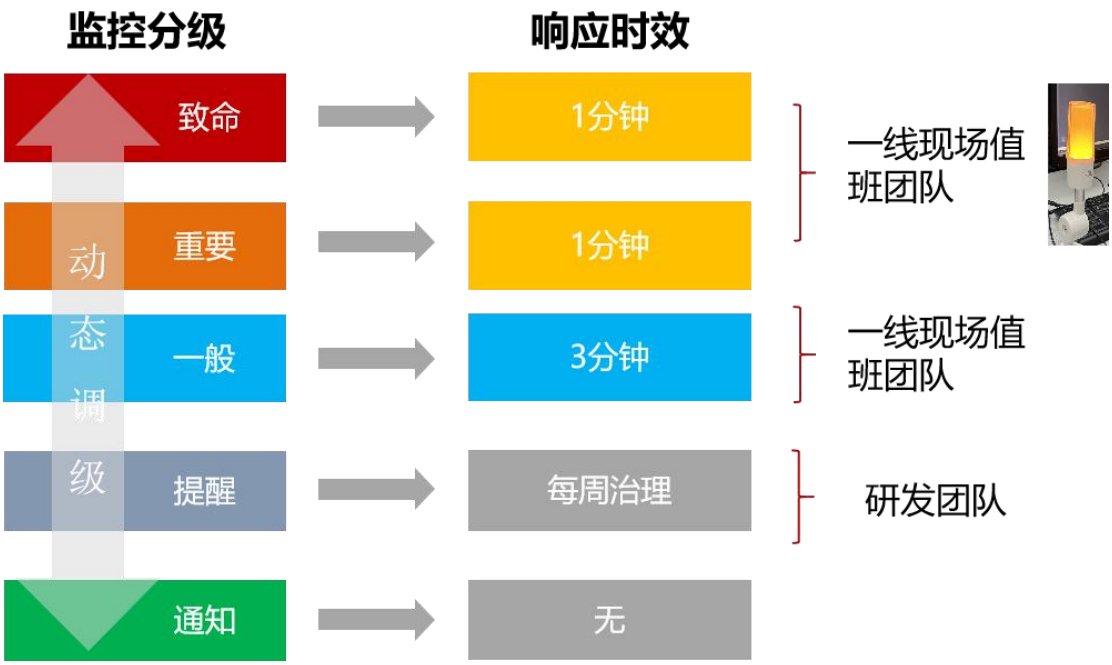
核心目标：在 1 分钟内感知故障，通过标准化的可观测性体系，实现快速发现与精准定位。

整体框架：围绕“全链路监控”和“服务运行监控”两大标准，构建统一、通用的监控策略。

基于自研的 RTP 开发框架，以日志+报文的形式打造应用的可观测性，通过全链路和服务运行两个标准监控确保关键、重要的交易、服务都有监控、且监控时效符合要求：

- 业务侧全链路监控-基于日志：
  - 实现交易量、技术成功率、耗时等业务交易黄金监控指标
  - 两个关键传播能力，实现技术失败、业务失败的快速定位：业务场景信息传播、技术失败信息传播
- 服务侧运行监控-基于应用主动上送的报文：
  - 服务启动自检：在服务启动时，主动上送一次启动自检报文，防止进程起来，服务不可用、假死的情况
  - 服务可用性：每分钟上送一次服务可用性自检报文，自检内容是可以自定义的，通用的包括到服务网关、缓存中心、消息中心等连通性自检，防止服务不可用情况
  - 服务连续性：类似心跳机制，是通过可用性报文计算得出
  - 服务运行性能：线程池、连接池等服务运行性能

响应更快-1 分钟响应



感知问题通过标准监控解决，但如果日常告警数量非常多，则会影响应急启动的判断，容易错失良机，也容易狼来了。为了能够快速聚焦重要的告警，做出准确的预判。我们细化了监控告警的级别、分工和处理时效要求，同时，通过告警的动态升级、置顶规则、合并规则，使一线可以快速聚焦重要告警，达到一分钟响应目标：

- **告警分级与响应时效：** 我们将告警按严重程度和影响范围细分为：
  - **致命 (Critical):** 对应 1 分钟响应时效，由一线现场值班团队负责。这类故障被定义为“生死告警”，实施严格的准入/准出控制，彻底杜绝过去“致命不致命、致命需要二次确认”的模糊状况，确保告警的真实性和紧迫性。在值班现场，我们甚至设置了**物理的告警灯**，专门用于强化致命和重要告警的提醒效果，确保即时关注。

- **重要 (Important)**：对应 1 分钟响应时效，由一线现场值班团队处理。
- **一般 (General)**：对应 3 分钟响应时效，由一线现场值班团队负责。
- **提醒 (Alert)**：对应轻微隐患，需进行每周治理，由研发团队负责。
- **通知 (Notification)**：对应信息知会，无需特定响应时效，由研发团队后续处理。

- **动态调级、置顶、合并机制：**支持按渠道、对象、监控策略、关键字、告警状态、持续时长等维度设置告警的动态调级、置顶、合并规则。
  - **动态升级/降级：** 告警等级可根据特定条件进行动态调整，确保告警的当前状态能准确反映问题的严重性。例如，某些“一般”告警在特定条件下可以升级为“重要”。
  - **自动置顶：** 致命告警、特定类型的重要告警，能够自动在告警列表中置顶，确保其优先级和可见性。
  - **告警合并：** 支持特定时间窗口内，相似、相关、同故障源（errSrc）的告警自动合并，帮助一线快速减少告警处理量，聚焦重要告警。

协同更快-线上线下应急协同



线上线下应急协同：通过有效的流程和工具，实现现场值班人员、线上二、三线支持人员以及业务人员的快速集结与协同作战。

- 我们根据告警级别、业务影响情况制定了 4 个级别的**应急响应模型：**一级和二级力求在 15 分钟内解决，三级故障要求在 30 分钟内解决，没有业务影响的四级故障则可以在一个小时内解决。应急响应模型可以帮助值班经理快速掌握应急调度人员要求、进展跟进要求、升级要求等。

- 当一线现场无法独立解决故障时，便会通过 **应急协同工具** 启动线上线下多方协同。该工具能够 **一键建群、自动拉人、启动会议以及自动对支持人员进入生产中心（ECC)进行来访授权、临时用户授权，**快速集结二、三线技术支持和业务人员。我们提供了 PC 端和

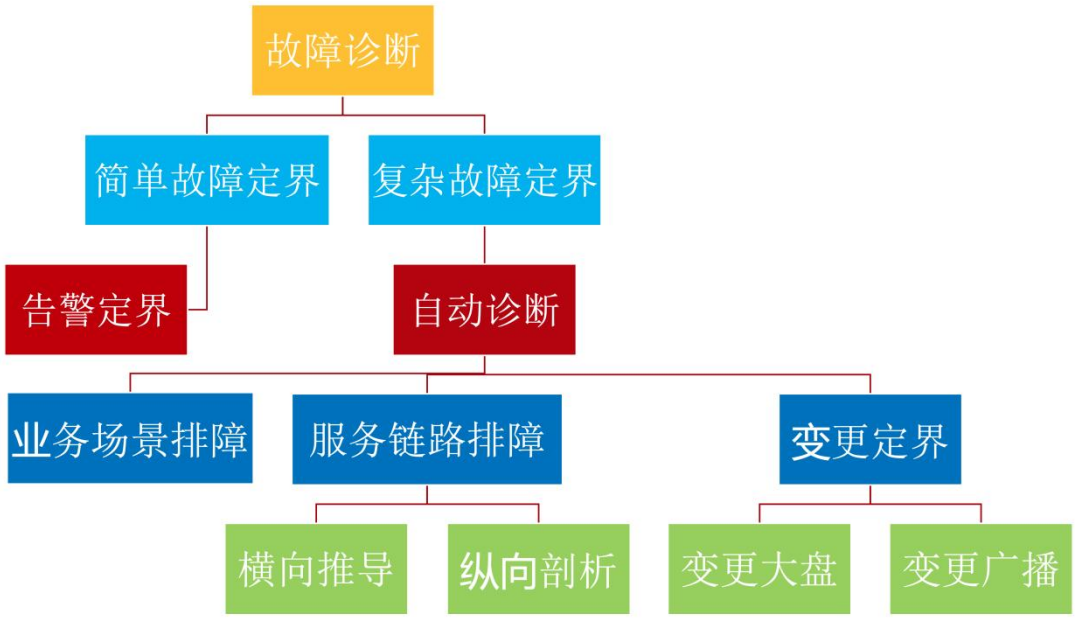
移动端 两种接入方式，方便不同场景下的协同。

- 在此过程中，应急协同工具还提供了重要的应急协同辅助能力，例如，应急过程中的 **信息共享、进度通报、协同任务分配以及决策指挥**。

节点级定界能力

如何实现 “5”

故障应急启动后，最关键的是精准的节点级定界能力。



如何实现 “5”，即一切围绕如何在最短时间内定界故障节点。

针对不同类型的故障有不同的定界方法。总体思路是：简单故障力求通过告警就能直接定界，复杂故障，我们先搭建了基础排障

能力，包括横向业务场景排障能力、服务调用链排障能力和变更定位能力，再实现基于专家规则的自动诊断。

1. 简单故障定界：依赖清晰的告警 – 对于那些简单、直接且有明确告警的故障，我们力求通过告警信息直接进行定界，这就像是故障在以预设的信号直接告诉我们问题所在。

2. 复杂故障定界：构建基础的排障能力 – 面临复杂的、表现不明确的故障时，我们打造了以下基础排障能力：

– **业务场景排障能力：** 通过业务场景排障大盘，按照"业务场景–渠道应用–渠道接口/服务–服务链路 –异常应用定界"的层级关系，逐层观测业务运行情况。

– **服务链路排障能力：** 基于服务调用链监控以及应用输出的性能日志，实现技术失败、交易耗时的定界以及秒级性能剖析。

– **变更定界能力：** 通过“变更大盘”快速筛选近期所有变更信息，并利用“变更广播”机制及时知会相关人员，从而快速定位到可能导致故障的变更。

3. 自动诊断：实现专家规则自动诊断 – 在构建了业务场景、服务链路和变更定界这些基础能力之上，我们进一步实现了基于专家规则的自动诊断能力，将过往的故障处理经验和知识转化为自动化规则，当新的故障发生时，系统能够依据这些规则自动进行诊断和定界，大大提升效率和准确性。

简单故障定界--告警定界



简单故障：告警直接定界

我们高度重视告警的直接定位能力，也是早会、故障复盘会、治理必问的问题。

其核心在于：

1. 告警对象信息丰富：

- 提供告警对象本身的属性，例如 CI 名称和关键属性，直接指向故障发生的具体组件。
- 向上关联其归属信息，包括业务模块、所属应用等，帮助理解故障在业务架构中的位置。
- 向下延伸至运行环境信息，如 OS/POD、宿主机、物理机，清晰地展示故障对象所处的底层基础设施，例如告警涉及的节点归属于哪个 POD、哪个宿主机、哪台物理机。

2. 告警内容信息丰富：

- 技术失败告警：基于全链路传播能力，告警内容包括了交易量、耗时、技术成功率三大黄金指标，还可直接展示错误源信息-errSrc (故障源应用)，快速识别技术根源。对于单一故障源引发的大量告警（即告警风暴），监控平台的告警合并规则能够自动将其聚合成一条告警，这极大地简化了告警列表，显著提升了告警定界的效率。
- 业务失败告警：自动关联告警影响的业务信息，阐明其影响的业务场景，方便从业务视角评估故障范围和影响用户的情况。

备注：errSrc（故障源）：一笔涉及多个服务调用的交易，在请求下一跳服务时，如果服务提供方报错，则由服务提供方在链路日志中将 errSrc 标记为自己，同时，输出具体错误信息（errMsg），并往回传播；如服务提供方超时，则由请求方将 errSrc 标记为服务方，并往回传播。

复杂故障定界--业务场景排障

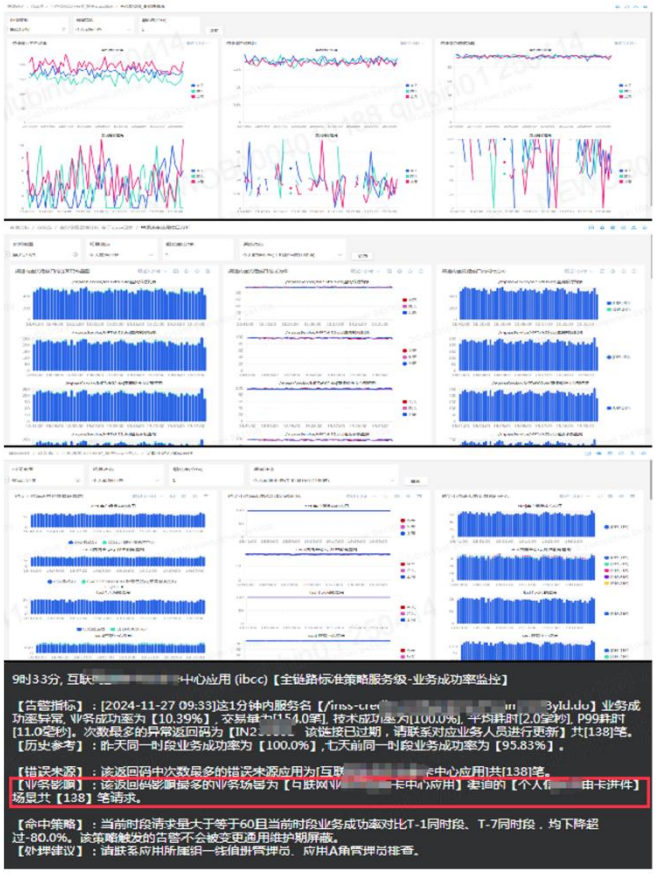


图1 基于业务场景查看各办理渠道的运行指标

图2 基于业务场景+渠道，查看该渠道接口运行情况

图3 基于业务场景+渠道，查看该渠道办理该业务场景所依赖的中后端应用运行情况

图4 全链路监控的告警信息增强，告警内容提示业务影响。

对于复杂故障的定界能力，我们是基于全链路监控平台去搭建的。通过将业务场景信息注入链路日志，利用全链路的信息传播能力，从前往后看，可以看到业务场景涉及的中后端关联应用有哪些，业务依赖的主流程接口有哪些，快速圈定排查范围，在一个页面内查看完所有相关接口的监控指标；从后往前看，可以快速知道

errSrc(错误源)可能影响了哪些业务场景、哪些上游的应用，从而快速评估故障的扩散范围。

基于业务场景的排障方法遵循一个清晰的层级关系进行逐层观测与定位：从最上层的业务场景，向下穿透到渠道应用，再到渠道接口/服务，进入服务链路，最终聚焦到具体的异常应用。

我们以一个具体的例子来阐述这一过程：考虑全行范围的“个人转账”业务场景。个人转账可在多个渠道进行，涉及用户登录状态检查、密码校验、扣款账户状态检查、余额判断、反洗钱等服务请求。假设某一天，个人转账的成功交易量突然下降，但不为零，而转账相关的接口技术成功率、耗时却表现正常。这可能是什么原因呢？一种可能性是某个渠道调用的余额查询接口出现了问题，导致转账业务到不了转账提交那一步。

那么，我们如何快速了解或判断个人转账业务会依赖于余额查询接口，并进而定位到该接口的具体问题呢？这就是业务场景排障发挥作用的地方。

复杂故障定界--服务链路排障



服务调用链排障能力，在全链路 trace 能力的基础上增加了传播能力，并基于“应用+业务模块”为核心的 CMDB 数据，将应用的链路日志和性能日志以及纵向技术栈指标建立关联关系，形成横到端、竖到底的可观测能力。

全链路可视与定界：

利用链路追踪技术，将服务间的调用关系可视化为链路聚合拓扑图，这使我们能够直观地识别出交易链路中的瓶颈或失败节点。

结合告警信息，例如技术失败或耗时异常的告警，可以直接导航到相关的链路视图，快速锁定可能存在问题的应用服务。

多维分析与精准定位：

在定位到问题应用后，通过应用多维分析大盘，可以从应用站点级、服务级、节点级分析本应用的交易情况，并支持按渠道、模块、交易类型、返回码等多维统计分析，帮助我们聚焦于有问题的交易或场景。

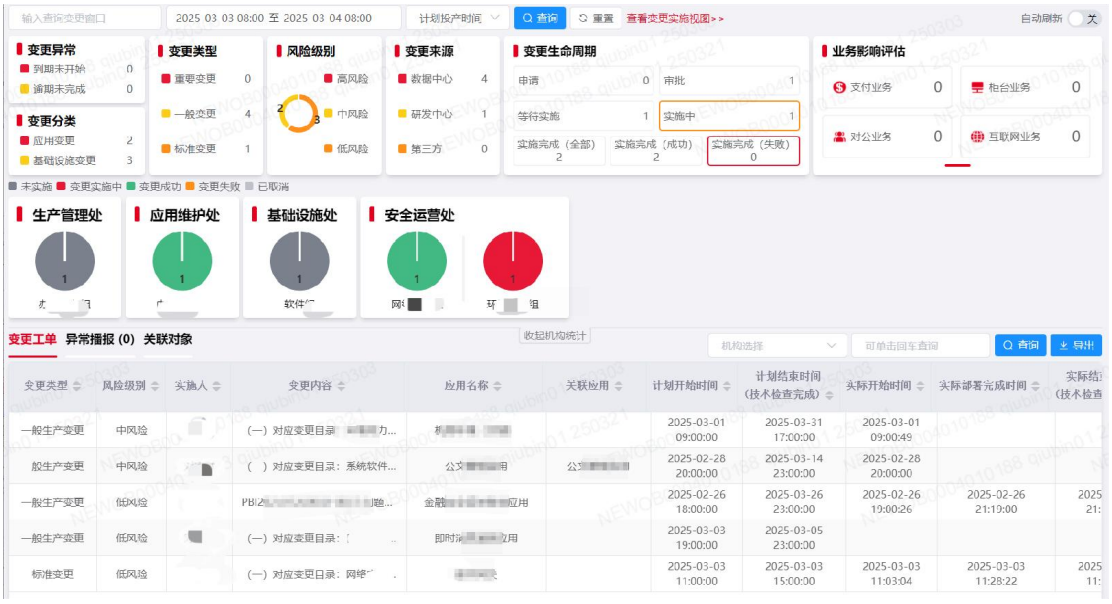
选择有问题的交易记录，我们可以进行单笔跟踪，精确定位到链路上的失败节点或耗时异常的节点，从而完成故障的初步定界。

性能剖析与根因分析：

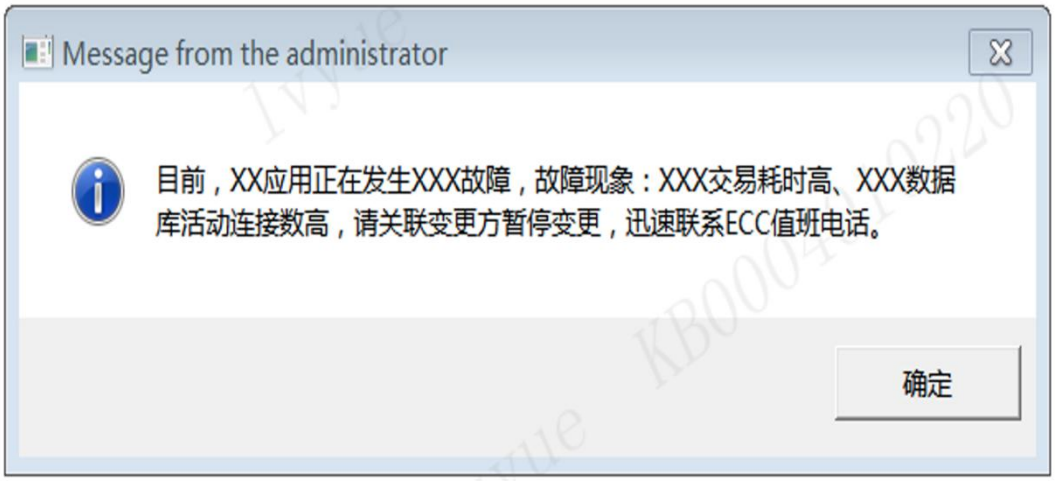
在完成定界后，如果需要深入分析性能问题或定位故障的根本原因，平台提供了秒级的应用性能分析能力。

应用性能分析大盘，支持对关键性能指标的剖析，例如，RPC 耗时、sql 耗时、druid 耗时、请求 TPS、堆栈日志分析、gc 日志分析等，帮助我们更细粒度地了解应用内部的运行状况。

变更定位



变更大盘：通过变更视图，快速查找应用本身、关联影响的变更信息，对于关键业务有影响的变更，提供快速筛选功能。这个视图支持多种维度的筛选，让我们可以根据变更的类型、风险等级、来源、当前的生命周期状态，以及对重要业务的影响程度，快速定位到特定应用相关的变更。



变更广播：终端变更寻播对于变更窗口人多、变更多的场景，可以非常便捷地将故障信息实时推送到变更实施人员的终端设备上，及时提醒他们关注当前发生的故障，并评估自己的变更是否可能是故障源，以便及时终止那些潜在的、可能加剧故障的变更。

故障自动诊断

前面几个是我们的基础故障定界能力，从上面，我们可以看到，故障定界时要检查的指标很多，个人完全掌握所有指标相关的排查方法，基本不可能，跨专业领域的协同分析，成本很高，很难满足 5 分钟故障定界的要求。因此，我们建立了基于专家经验的故障自动诊断模块，实现模拟人工的故障排查过程。



在全链路监控的基础上，基于 CMDB 数据，将应用的链路日志和性能日志以及纵向技术栈指标建立关联关系，形成横向推导和纵向剖析两大能力，对全栈数据进行自动分析：

交易链路层的横向推导：包括业务场景的故障源推导、服务级技术失败推导、耗时推导等，通过定界故障源系统的站点级、服务级、节点级情况，分别对应站点流量切换、服务降级/限流、进程重启/隔离等应急措施。

纵向技术栈剖析：包括应用资源、软件资源、系统资源、基础设施，重点检查生死指标和性能指标。

横纵诊断结果以列表的形式展示，查看诊断返回详情就能获取异常节点信息：



点击“详情”还可以进一步查看该诊断项的详细诊断过程，按模块失败总数从高到低排序或按时间从开始到最近检查结果。



落地价值

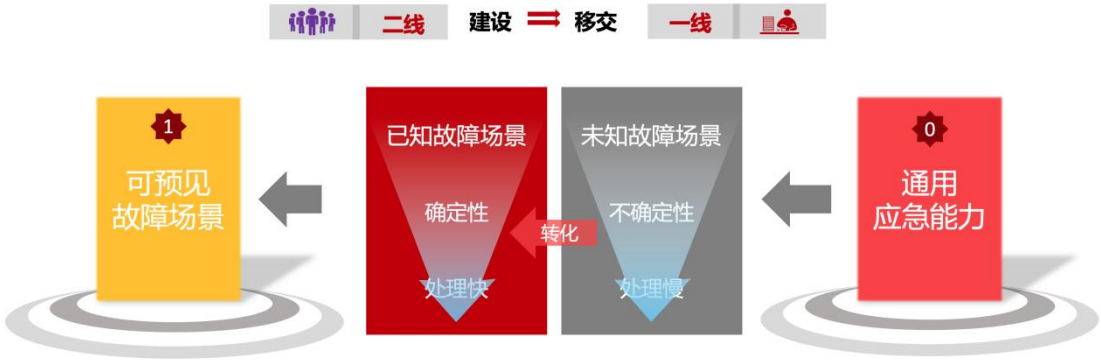
- 从“发现”到“定位”——将原本依赖少数资深专家的隐性知识，转化为可持续的自动化能力；
- 在典型故障场景中，诊断时间从数分钟缩短至数十秒；

- 解放团队在繁杂指标间的排障定界投入，将更多精力投入到修复与优化上。

打造可预见故障场景

如何实现“10”：从通用应急能力向可预见故障场景的持续转化

在应急处置方面，未知的故障场景不确定性高，应急慢，已知的故障场景确定性高，处理快，因此，应急恢复的建设思路是先建基础通用应急能力，再逐步打造可预见故障场景，实现快速应急。



打造通用应急能力-应急六板斧

我们构建了一套通用的应急能力体系，也就是我们常说的“应急六板斧”：切流、重启、扩容、隔离、降级限流以及回切。

这六板斧分别对应着不同的故障场景和应急需求，旨在帮助我们快速、有效地响应和处置各类系统异常。

- **切流：**主要应用于站点级或单元级的故障。当某个数据中心或区域发生问题时，我们可以迅速通过工具（比如“同城双活应用快速切流”、“防火墙一键逃生”）将流量切换到健康的单元，确保业务连续性。
- **重启：**适用于服务出现 Full GC、OOM 或服务状态异常等场景。通过“一键启停”工具，我们可以便捷地对应用服务进行快速现场保留、重启，重启操作往往能有效恢复系统功能。
- **扩容：**是解决资源紧张类故障的利器。当系统面临预期的或突发的流量洪峰、或者资源（CPU、内存、文件系统、数据库表空间等）成为瓶颈时，我们可以通过相应的工具快速增加资源供给，提升系统承载能力。
- **隔离：**适用于集群内部出现单个节点故障的情况。在这种场景下，通过负载均衡的“节点隔离”或 服务运营管理平台的“服务隔离”工具，我们可以将故障节点快速从集群中剔除，防止故障蔓延影响整体服务，同时还能保留故障现场，便于后续分析原因。
- **降级/限流：**主要适用于链路弱依赖、避免影响其他业务的故障。在出现上游依赖服务不可用或自身资源不足以应对当前流量

时，我们可以通过例如“互联网渠道交易限流”或控制“银联支付贷中开关”等手段，对非核心功能进行降级或限制流量，从而优先保障核心业务的可用性。

- **回切：**适用于解决由变更操作（如应用部署、配置更新）引入的问题。当发现变更导致系统异常时，我们可以快速通过自动化部署平台实现应用版本的自动回退。

总而言之，这“应急六板斧”，配以相应的自动化工具，构成了我们应对复杂分布式系统故障的通用应急能力基石，能够覆盖绝大多数常见的应急场景。

可预见故障场景建设



所谓的可预见故障场景，是告警出来就可以直接应急的，这中间无需耗费时间排查，直接进入应急处置流程。

可预见故障场景，主要通过"历史故障分析"、“故障复盘"、"经验总结"、“头脑风暴"等方式，将成熟的故障定界及处置经验，形成可预见故障应急场景清单，实现确定的故障场景有"确定的告警"、"确定的应急处置方案"、"确定的应急动作"：

确定的告警：清晰定义故障发生的监控告警，确保能第一时间被发现。

确定的应急处置方案：针对特定场景，有预先设计好的、标准化的应急方案。

确定的应急动作：方案中的具体操作步骤必须细化、可执行，避免临场决策与判断。

一键应急能力



为了应对复杂的系统挑战并有效处置突发事件，我们构建了一套集报警、知识和应急处置于一体的“一键应急能力”平台，旨在将应急流程的各个环节无缝衔接，从而实现端到端的快速响应。

复盘改进

进入故障复盘环节前，不妨先思考两个核心问题：关于故障复盘，我们最害怕和最期待的是什么？或许有人会说，最害怕的是复盘会变成问责会，哪里还敢有什么期待呢？

复盘会的重点是发现问题并改进，我们内部一直倡导这样一个信念：**能在内部复盘的故障，那都是上天给我们的一次改正机会，**如果你们的领导是抱着讨论的态度和你复盘的，相信能让大家放心把问题讲出来，把改进的要点讲清楚。



XXX 事件分析报告

一、事件现象

二、事件情况简述

北极星指标统计：应急处置时长 xx 分钟

1. 故障发现时长：xx 分钟
2. 故障定位时长：xx 分钟
3. 应急恢复时长：xx 分钟

三、事件处理情况

四、事件影响

【网络舆情情况】：

【客户投诉情况】：

【资金损失情况】：

【交易影响情况】：

【业务影响情况】：

【其他情况】：

五、系统、业务处理流程/场景

六、事件原因分析

直接原因：

根本原因：

管理原因：



七、 事件责任认定

责任系统：

主要责任人：

理由：

八、待改进点分析

一）故障为什么会发生

关键环节	具体分析点	改进点分析情况（具体是什么，如何规避问题再次发生）
设计环节	是否为架构设计问题，如何规避问题再次发生	
开发环节	是否为程序设计问题，是否需改进	
测试环节	测试是否已发现，为何未发现	
变更环节	是否为变更发布导致，如何规避问题再次发生	

二）如何更快后恢复

关键环节	具体分析点	改进点分析情况（具体是什么，如何规避问题再次发生）
监控环节	监控是否第一时间发现	
故障定位	故障定位工具是否可以发挥作用	
应急处置	应急方法是否做到一键操作或脚本化	
规则协同	规则协同是否有序，是否需要改进	

九、后续改进措施

十、举一反三（可选）

以上是一份标准化的“XXX 事件分析报告”模板，涵盖了从事件现象到举一反三等十个章节，力求全面、深入地分析故障。这份模板详细包含了以下关键要素：

事件现象： 对故障的初步观察和描述。

事件情况简述： 精确记录故障发生的关键时间点，包括故障发现、定位、恢复的时间，以及核心北极星指标。

事件处理情况： 详细记录故障处理过程中的每一步具体操作和决策。

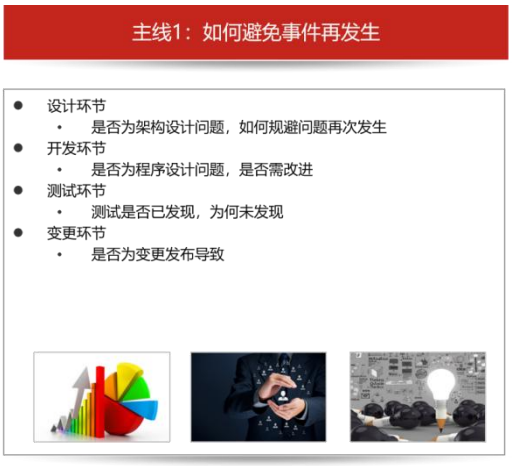
事件影响： 全方位评估故障的影响，包括但不限于网络舆情、客户投诉、直接或间接的资金损失、对交易和核心业务功能的影响，以及其他次生影响。

系统、业务处理流程/场景： 回顾并分析故障发生的具体系统、相关的业务处理流程以及涉及的场景，为后续分析提供上下文。

事件原因分析： 系统性地从直接原因、根本原因和管理原因三个维度深度挖掘故障发生的深层逻辑和诱因。

事件责任认定： 明确在故障中哪些系统和人员负有主要责任，并分析其理由。

待改进点分析：





- 故障发生的原因：从设计、开发、测试、变更等研发和发布流程的关键环节入手，深入剖析是否存在问题，例如架构设计是否

合理、程序是否存在缺陷、测试方案是否充分、变更流程是否规范等，并思考如何从源头避免同类问题再次发生。

- 应急恢复过程中存在的问题：聚焦故障应急响应、处置效率，分析监控系统是否能第一时间准确报警、故障定界工具是否有效、应急处置方法是否最优、现场多团队之间的协同是否顺畅等，识别响应流程中的不足之处。
- 后续改进措施：基于“待改进点分析”的结果，制定具体的、可执行的、有明确责任人和完成时间的改进计划，确保复盘成果真正落地。
- 举一反三 (可选)：将本次故障中获得的宝贵经验教训进行提炼和推广，识别在其他系统或领域可能存在的类似风险，并主动进行排查和预防。

（三）总结及展望

XX 银行通过“1510”工程牵引整体运维能力的提升，构建业务连续性保障能力。本案例聚焦应急组织流程、故障发现、故障响应、故障定界、快速恢复、复盘等方面的改进经验，从应急组织架构、应急管理机制、一线应急机制、组织文化营造、分钟级响应能力、节点级定界能力、可预见故障场景建设等方面阐述从组织到工

具的应用应急能力提升方法。为规模体量相似的中小商业银行、同业提供可借鉴的故障应急能力提升思路：

1、了解组织在故障应急提升方面的作用、应急组织机制如何实现应急能力提升的闭环优化、值班经理应当具备哪些修为

2、了解分钟级响应、节点级定界、可预见故障场景、一键应急等能力的建设思路

5.3.9 腾讯 IEG SRE 应急响应实践

SRE Elite 精选原因：

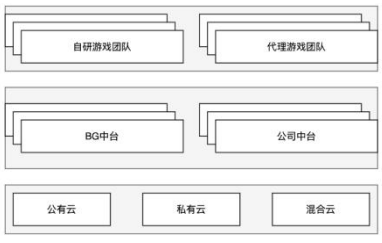
方案亮点在于依托蓝鲸基座,以平台工程的方式整合监控、CMDB、权限等能力，形成“告警→响应→诊断→恢复→复盘”全链闭环。标准化告警接入配合巡检确保观测确定性，自动升级流程打通跨团队协作；APM+eBPF 零侵入全栈观测结合 LLM Agent 智能根因定位，将诊断压缩至分钟级；混沌工程与每日过载验证保障过载保护有效，常态化 On-Call 与演练提升团队实战熟练度，使多数故障实现分钟级自愈，真正把应急从人治升级为体系化工程。

（一）背景及设计原则

游戏应急响应的目标和痛点

我将首先基于腾讯游戏的特点，与大家探讨在应急响应领域我们所遇到的困难和痛点。随后，我将针对这些痛点，详细分享我们如何基于“蓝鲸基座”构建一个通用的应急响应方案。最后会简要分享下我在执行平台业务应急响应上的一些思考和经验。

腾讯游戏的特点



现状：
- 研发团队：自研和代理
- 游戏架构：模块多差异大，发行区域：国内和海外，全球发行
- 支撑平台：BG内、BG外平台服务
- 规模大：单游戏体量和整体体量

特点：
- 上下游涉及服务、人员多（多组织，多角色）
- 异构性非常强
- 运营环境复杂

在讨论腾讯游戏的特点及其SRE 稳定性面临的挑战时，需要认识到其核心特性：

业务模式与产品多样性：

腾讯游戏涵盖自研与代理模式，同时进行国内外的全球发行，新生代游戏更是普遍面向全球市场。

多品类（如动作类）、多玩法、多运营需求以及不同工作室的存在，导致其架构差异巨大、模块较多。

中台体系的支撑：

- 为支持游戏的快速上线与高效运营，IEG（互动娱乐事业群）构建了 BG（业务群）级的中台，同时公司层面也有统一的中台能力作为补充。

庞大的体量规模：

- 腾讯游戏的整体体量及单个游戏的规模都相当庞大，这使得许多潜在问题（例如效能瓶颈）的影响会被显著放大。

复杂的协作与架构：

- 由于涉及多样的业务模式、产品及中台系统，整个生态系统的上下游参与者众多，形成了多组织、多角色的复杂协作关系。
- 核心架构具备显著的异构性特点。
- 运营环境复杂，不同游戏特性在环境部署需求上有明显差异。（比如延迟敏感的游戏会要求就近部署、多接入层动态调度等）

这些特点共同构成了腾讯游戏在 SRE 稳定性实践中必须面对和解决的独特挑战，尤其体现在大规模、多变的环境下保障服务的连续性和可靠性。

腾讯游戏应急响应的目标

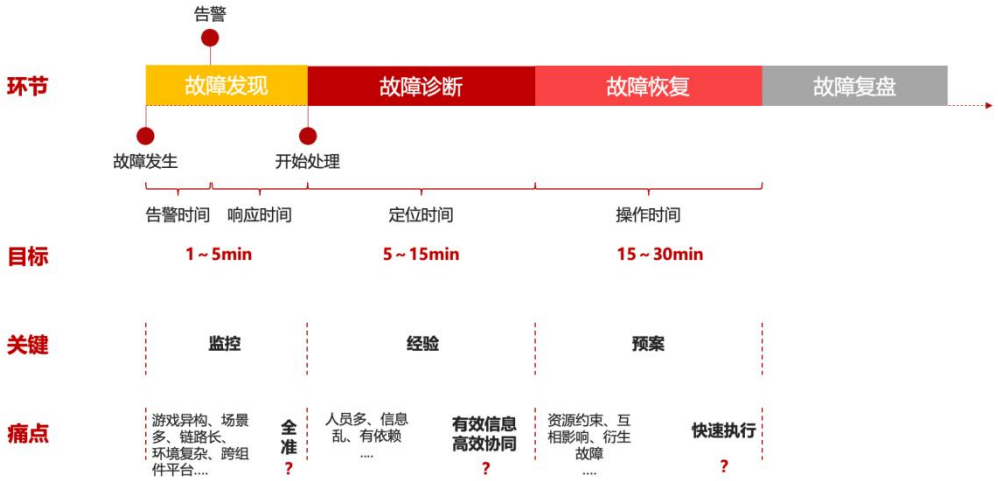


游戏作为一个对**用户体验要求极高**的产品，即使是微小的网络抖动或连接中断，都可能导致大量玩家投诉甚至流失。鉴于此，在**支付、登录、注册等关键游戏业务场景下**，我们的稳定性保障设定了严格的服务级别指标，旨在：

- 确保能够在故障发生后一分钟内完成发现；
- 实现故障在五分钟内完成诊断；
- 目标是在十五分钟内完成系统的恢复，从而最大限度地减少对玩家体验的影响。

腾讯游戏应急响应的痛点

腾讯游戏应急响应的痛点



1. 故障发现（1~5min）

在这个阶段，目标是尽快发现故障。关键在于实时监控，确保能够及时捕捉到系统中的异常。由于游戏架构异构、场景多且链路长（涉及多组件平台）、游戏部署环境复杂，能否做到监控的全和准是该阶段的主要痛点。

2. 故障诊断（5~15min）

此阶段的目标是精确诊断故障原因，依赖于过往经验和技术积累。有效的信息收集对故障的快速定位至关重要。由于游戏自身模块较多且单一功能场景会涉及多组件多平台，链路长，这会导致无法快速的收集信息，此外模块与模块之间、服务与服务之间会存在相互影响，一个故障的产生可能不是单一链路的问题，需要整体协同定位，如何高效协同找到根因也是当前环节的痛点。

3. 故障恢复（15~30min）

目标是在这个阶段快速恢复服务，确保系统尽快恢复正常运行。关键在于快速执行预案，并确保不会丢失重要数据。痛点集中在资源的有限性以及不同系统之间的相互依赖，可能导致恢复时间较长。而且，故障之间往往存在连锁反应，需要谨慎操作以避免扩大影响。

基本止损优化策略尝试，有用，但不够

实际上，在着手构建专业的应急响应系统之前，我们团队进行了多方面的探索与实践。这其中包括了：

1. 告警优化

基础优化：对 CPU、内存、磁盘、IO、网络等基础性能进行监控，确保系统的健康状态。

业务服务优化：关注业务服务的调用量、成功率等关键指标，确保服务的稳定性。

用户体验优化：监控用户操作过程中的登录、下载、支付、单品等环节，提升用户体验。

综合 SLI/SLO：综合考虑服务质量指标，通过 SLI/SLO 来衡量系统的可靠性。

2. 架构优化

冗余、容错：通过增加系统的冗余设计，提高系统容错能力，确保在故障发生时能够快速恢复。

负载均衡：通过负载均衡技术，分担系统的压力，避免单点故障。

在线更新与升级：系统能够在线更新与升级，确保在不停机的情况下进行功能扩展和优化。

3.部署优化

跨区域部署：通过跨区域部署服务，提高系统的可用性和容灾能力。

网络优化：确保数据的快速传输，优化网络架构，减少延迟。

服务接入与数据层优化：优化服务接入方式，简化数据传输流程，减少瓶颈。

4.预案执行

自动化预案：构建自动化的应急预案系统，实现故障的自动响应与修复，减少人为干预，提高恢复效率。

资源调度与混沌工程：通过资源调度和混沌工程来提升系统的弹性和抗压能力，确保在高负载时仍能保持稳定运行。

这些工作带来了显著的效果，它们有效解决了我们所面临的大部分稳定性问题，极大地改善了系统的整体可靠性。

然而，尽管取得了这些进展，我们仍然面对着一些棘手的问题，这些问题是上述单一或局部的优化手段无法彻底解决的。这正是促使我们进一步思考和构建一套更为全面、系统化的应急体系的根本原因。

我们意识到，需要一种更具穿透力的方法来应对那些复杂、突发的未知风险，而不仅仅依赖于对已知问题的改进。

建设应急系统之前的必要性阐述

在阐述建设的必要性之前，我们先来看两个案例。

案例一：游戏更新导致存储层代理模块 CPU 满载，无法及时触发告警引发故障

背景：

游戏 A 11/7 日发布了新版本，客户端版本新增了协议，会放大存储层压力，11/13 日 A 业务活动放量，导致存储层 proxy 模块 CPU 持续 100%，引发故障。

事故过程：

- 19:56 游戏 A 运维收到告警，显示“用户登录失败超过阀值”，遂联系负责登录后台的开发同事第一时间核查和解决。
- 19:57-20:30 游戏 A 运维、研发协同存储层负责同事一起排查，期间尝试重启游戏后台服务、对服务进行降级等措施，暂时恢复了登录，但是部分游戏功能依然受到影响。
- 20:30-20:35 存储层服务团队定位到一台 proxy cpu100%，遂立即进行扩容。
- 20:40 扩容完毕，业务 A 逐步恢复……..

事故原因：

……..

proxy 模块 CPU100%却无告警核查结果：告警策略配置了但未生效，导致没有触发告警。

……

故障影响大于 40 分钟

案例二：热更新引发登录失败问题，误判为版本问题导致故障排查延迟

背景：

游戏 A 6/11 日发布客户端资源热更新，更新后出现登录成功率下降。

事故过程：

- 6/10 15:00~15:30 登录组件运维按计划执行某一模块升级，灰度和全量期间观测无异常。
- 6/11 16:00 游戏 A 发布热更。
- 6/11 16:08~16:14 游戏 A 运维收到登录异常告警，期间登录组件运维也收到游戏 A 登录异常告警，经排查发现自身服务成功率无异常，同时周知游戏 A 时，了解到 A 刚发了热更，遂认为是版本问题，于是等待 A 业务修复。
- 6/11 16:15~16:40 游戏 A 运维和研发联合排查，期间排除了新功能的影响，定位到用户的异常均在登录模块中的 lua 脚本，而该 lua 脚本这次没有改动，且和本次版本更新内容没有关联，并且只是部分玩家出现异常。
- 6/11 16:41 游戏 A 寻求登录组件帮助，组件运维联系研发进行排查。
- 6/11 16:42~16:50 登录组件研发经排查，发现后台存在某些异常的 ID 有一直在发起登录请求，也成功返回了数据，但是却没有登录成功，确认是由于 A 游戏本次更新触发了 6/10 组件升级带来的 BUG 导致。
- 6/11 16:51~16:56 登录组件运维紧急回滚版本。
- 6/11 17:00 游戏 A 登录异常逐步恢复。

故障影响近 1 小时

案例分析

第一个案例揭示了告警覆盖的盲区与决策失误的代价，一个关键模块告警的缺失直接导致故障诊断出现偏差，进而将故障影响扩大到超过 40 分钟。这凸显了完善的可观测性、尤其是关键指标的全面监控与告警机制的重要性。

第二个案例则指出了跨部门或跨岗位协同的潜在漏洞，诊断过程中关键人员的遗漏参与，使得问题解决耗时近一个小时。这强调了在应急响应中建立清晰、敏捷的协同机制与信息共享渠道的必要性，确保所有关联方都能及时到位，共同参与故障处置。

这两个案例虽然简单，但都指向了稳定性保障中的核心痛点：一方面是信息获取的完整性和准确性（告警缺失、可观测性不足），另一方面是故障处置的人员协同效率。后续我们将探讨如何通过系统性的方法和工具来弥补这些不足，提升整体的 SRE 稳定性保障能力。

同时，我们深入分析了以往耗时较久的故障，通过复盘发现，问题主要集中在以下三个核心环节：告警、协同与处置。

首先来谈谈告警环节。这里面的痛点多种多样，可能表现为：

告警缺失： 本应触发的告警未能如期发出，导致问题发现不及时。

告警延迟： 告警未能实时送达，延误了响应的最佳时机。

告警泛滥： 系统产生过多无效或低优先级告警，使得真正关键的问题告警被淹没在信息的海洋中，难以被有效识别和处理。

接下来是协同环节。高效的协同是应急响应的关键，其挑战在于：

人员未及时到位： 故障发生时，未能迅速集结所有必需的响应人员，导致处置力量不足。

关键角色缺失： 在响应过程中，发现缺少对故障类型至关重要的专业人员，影响问题分析和决策效率。

最后，也是最考验 SRE 硬实力的处置环节，问题处置。

快速有效地解决故障极具挑战性，因为问题的根因复杂多样，而且对 SRE 的个人能力、经验积累，以及整个工程体系的成熟度提出了非常高的要求。故障处置的速度和成功率在很大程度上取决于

工程师的能力、自动化工具的支持程度和预案的完善度，是一个概率与能力综合影响的过程。

在这三个环节中，告警问题可能并非总是最直观的痛点，但它却是整个应急响应流程的起点。下面我们将重点展开讨论告警方面的问题。

告警痛点：游戏业务的监控告警配置痛点困难分析

游戏业务有其显著特点，尤其体现在监控指标的复杂性和多样性上。我们发现不同类型的游戏所需的监控指标截然不同。例如，对于 FPS（第一人称射击游戏）类游戏，其基础监控项就已数量庞大。

类别	监控项	说明
下载	下载量/成功率/下载速度	关键项
安装	安装量/成功率/耗时	关键项
登录	域名劫持	域名解析异常(劫持)/跨网
	登录成功率	各个步骤的转化率
	登录时长	玩家平均登录时长
	游戏内登录	辅助判断掉线以及网络异常
	游戏内登出	辅助判断掉线以及网络异常
在线	大厅在线/掉线/重连	判断大厅基础体验

	gamesvr 网络连接数	辅助判断大厅掉线
	gamesvr 当前房间数量	辅助判断大厅掉线
单局	网络延迟/loading 耗时/FPS	单局体验判断情况
	单局在线/房间数/掉线	单局基础运行情况判断
	流量监控	辅助判断游戏内异常/掉线异常/网络异常
	平均丢包率/平均延时	监控玩家网络情况/局内体验
注册	注册监控	渠道、平台注册
客户端崩溃	崩溃率和崩溃量	监控玩家游戏体验
FPS	FPS 明细	监控游戏流畅度
匹配	匹配时长/超时次数	监控玩家游戏体验/网络异常
角色创建	角色创建流水监控	关键项
金钱/道具	金钱消耗流水监控	关键项，道具分类监控
购买	购买失败量	监控玩家游戏体验
周边系统	周边系统的访问质量	关键项
后台逻辑异常	逻辑异常监控告警	常规项
服务器基础	CPU/内存/IO/流量/ping/…	常规项
后台基础	进程不存在/core/通道	常规项
容器平台	pod/node/apiserver/….	常规项
文件上传	录像上传量/成功率	监控玩家游戏体验

一个 FPS 类游戏的基础监控

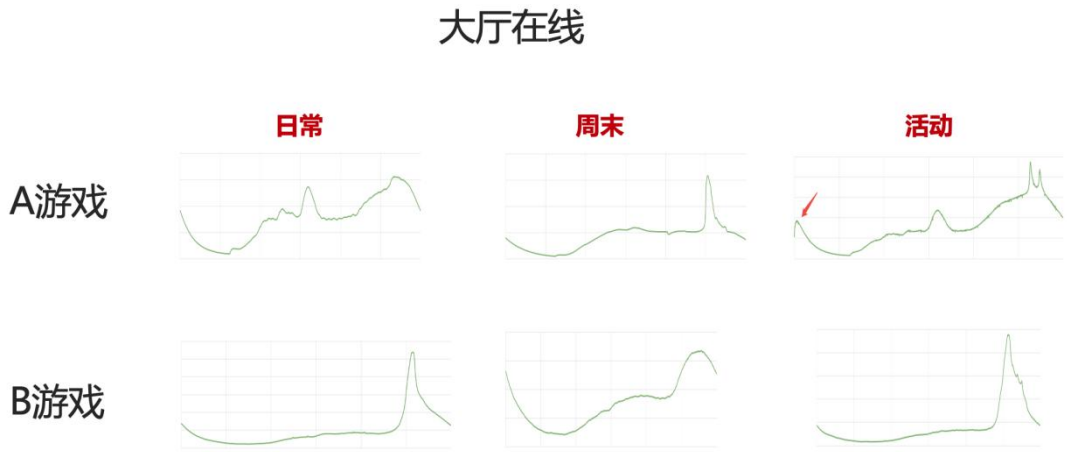
除了 FPS 外，还有 RPG, STG, MOBA, SLG，卡牌，开放世界沙盒等各种游戏类型。

更具挑战性的是，这些指标会受到多重因素的影响，包括：

游戏类型差异： 不同类型游戏的核心指标不同，例如射击类关注延迟和帧率，策略类关注对局完成率和匹配时长。

地域特性： 不同地域的网络环境、用户习惯会导致游戏监控指标表现出差异。

运营活动： 新版本上线、限时活动等会带来流量骤增和用户行为变化，直接影响游戏监控指标。

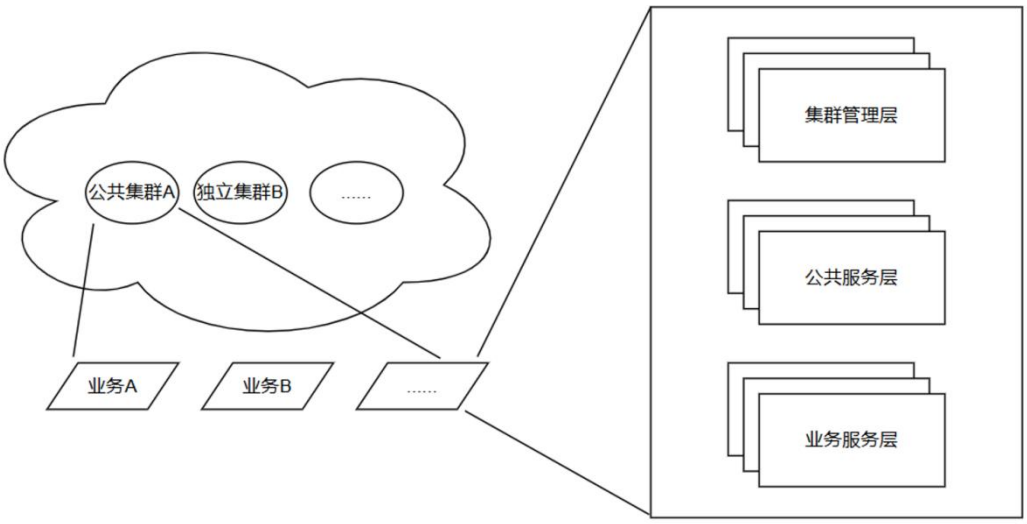


举例来说，即使是**同类型的两款游戏（上图 A 和 B 游戏）**，由于用户基数、活跃时间分布或运营策略的不同，其**指标曲线会展现出非常显著的差异**。此外，同一款游戏在不同时间段（如工作日与周末、白天与夜晚）的在线人数和行为模式变化，**也会导致监控指标波动巨大。**

这意味着我们无法简单地采用一套统一的标准或规则来对所有游戏进行有效的监控。如果试图为每个游戏的每个细分场景都配置详尽的监控规则并确保其精准性，监控项的数量将呈爆炸式增长。尽管我们可以尝试通过分级或降噪技术来管理海量告警，但这又极易导致重要异常被遗漏，从而影响应急响应的及时性。

这正是我们在游戏监控实践中遇到的一个现实的、迫切需要解决的问题。下一步，我们将探讨在面对平台业务时，情况又有何不同。

告警痛点：平台业务的独特挑战



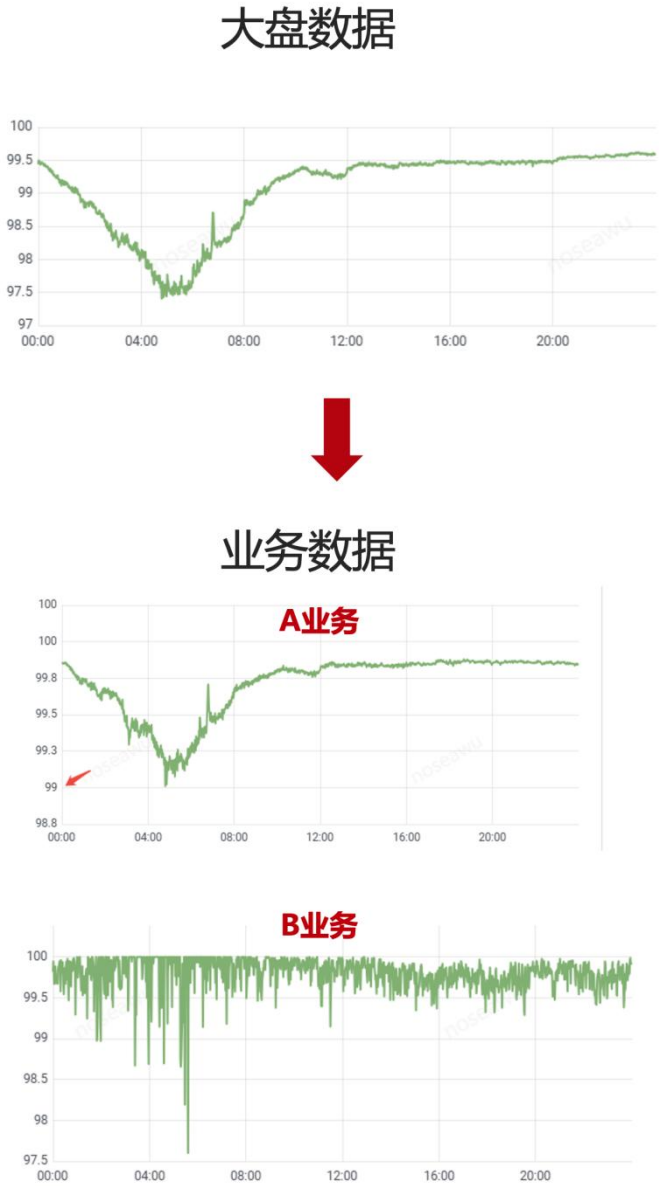
从上面的部署架构模式可以看到，对于平台业务，尤其是服务众多内部客户的平台业务，其特点是面向外部表现为一个统一服务，但在内部却支撑着可能上百个差异巨大的业务。正如云计算平台通常会遇到的情况，一旦核心平台出现问题，将可能波及大量下游客户。

在游戏业务体系中，我们的平台业务主要分为了面向 c 端游戏平台业务，面向 c 端组件平台业务，面向后台组件平台业务。

类别	监控项	说明
集群维度	A 服务大盘请求量	基础项，可以发现端侧异常
	A 服务大盘请求成功量	基础项，可以发现自身或者下游异常
	A 服务大盘请求失败量	基础项，稳定且敏感业务需要此项数据监控
	A 服务大盘请求成功率	基础项，可以发现自身或者下游异常
	A 服务大盘错误码总量	基础项，稳定且敏感业务需要此项数据监控
	A 服务大盘细分错误码总量	基础项，用于针对特定错误码监控
	A 服务大盘请求延迟	基础项，用于服务能力的及时感知
	A 服务大盘容量水位	基础项，用于容量异常的及时感知
	A 服务大盘配置检查	基础项，用于部署容灾、过载保护等配置监控
	B 服务……	
业务维度	A 业务请求量	基础项，作用同大盘，但是细分到 A 业务
	A 业务请求成功量	基础项，作用同大盘，但是细分到 A 业务
	A 业务请求失败量	基础项，作用同大盘，但是细分到 A 业务
	A 业务请求成功率	基础项，作用同大盘，但是细分到 A 业务
	A 业务错误码总量	基础项，作用同大盘，但是细分到 A 业务
	A 业务细分错误码总量	基础项，作用同大盘，但是细分到 A 业务
	A 业务请求延迟	基础项，作用同大盘，但是细分到 A 业务
	A 业务容量水位	基础项，作用同大盘，但是细分到 A 业务
其他	基本同游戏业务	
……		

网址：SRE-Elite.com 微信：SRE 精英联盟

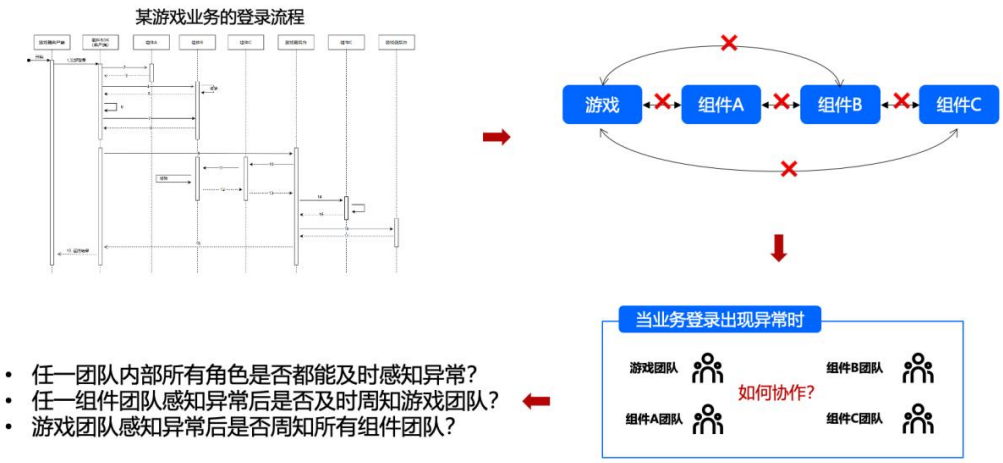
以上是一个面向后台组建平台业务的基础监控指标列表



虽然平台业务相比游戏业务，SLI 项不多，但是由于接入游戏业务众多，且游戏业务的异构性、服务接入方式、量级、运营周期、运营活动不同，**会导致业务维度监控差异较大。**

面对这样的场景，传统的整体监控模式显然不足以应对，必须下沉至每个具体业务层面构建精细化监控体系。然而，游戏业务的独特性在于其高度的多样性和快速迭代。这导致在游戏行业中构建普适且全面的监控体系面临巨大挑战，往往陷入覆盖面与精细度难以兼得的困境，即难以同时实现 0 死角覆盖与全面监控。这暴露了监控层面的固有难题。协同问题也因游戏场景的复杂性而进一步凸显。

协同痛点：信息孤岛导致无法协同。



ß

IEG 的业务场景极其复杂多样，与其说是一个简单的技术栈，不如说是一个由多个组件交织而成的庞大生态系统，例如，一个简

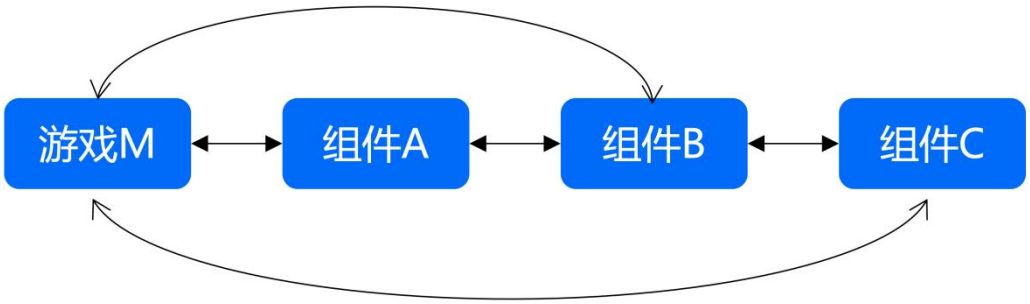
单的登录流程就可能牵涉到多个组件，而组件与组件之间、游戏与组件之间的交互关系盘根错节。在这种高度复杂的环境下，一个微小的异常就可能迅速蔓延。

在过往的实践中，我们发现一个普遍存在的、也是非常耽误我们处置问题的情况：**当故障发生时，研发或运维团队往往倾向于先排查自身负责的系统，而不是快速判断是哪一个环节出了问题。**

这种“先看自己”的习惯虽然体现了对自身系统的负责，但在跨组件、跨团队协作定位和解决问题的场景下，却白白浪费了黄金的应急响应时间。这正是我们在实践中深刻体会到的痛点，也是推动我们不断优化应急响应流程的重要驱动力。

处置痛点：诊断和恢复时间过长

案例：



游戏 M 登陆流程及组件

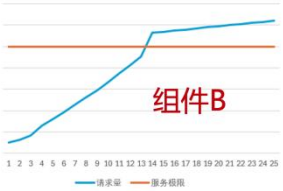
背景：

游戏 M 预计于周年庆当日 0 点举行活动，预计登录峰值增长 200%，且提前与组件 A、B、C 做了报备，组件 A、B、C 也充分评估容量满足报备要求。

组件B返回时延（单位：毫秒）	组件A单机器性能（单位：请求/秒）
100	42000
1000	24000
2000	18000

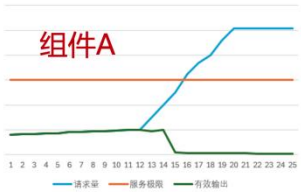






- A调用时延增大，性能下降
- A有效输出低于极限
- 游戏请求超时，重试

……



事故过程：

00:05 组件 A 运维收到告警电话，开始处理，发现从 00:02 开始游戏 M 登录请求成功率下降，并且有持续下降趋势，**此时登录服务开始产生大量告警，于是开始联系组件 A 研发紧急处理。**

00:08 组件 A 通过查看日志，发现大量组件 B 请求超时错误，于是紧急联系组件 B, 组件 B 回复服务 CPU 高，正在紧急扩容。此时组件 A 也发现，自身服务 CPU90%，通过查看请求量数据，发现在自身容量的 30%处出现异常，判断当前 CPU 90%应当是下游容量

不足导致自身性能受影响（处理耗时增加+业务不断重试），于是并未做进一步的处理，**等待组件 B 恢复。**

00:15 收到组件 B 同步，已经扩容完毕，但是组件 A 观测数据，发现登录请求成功率并未恢复，登录成功请求量恢复并逐步增长到自身极限性能，**自身 CPU 从 90%增长到 100%，此时请求量在持续上升。**

00:18 组件 A 开始紧急扩容。

00:32 组件 A 扩容完毕，游戏 M 登录请求成功率逐步恢复。

右上角两个图分别展示了不同组件 B 调用延迟对组件 A 处理性能的影响。下面左右两张图展示了整个过程中，组件 B 和组件 A 请求量和有效处理能力的趋势，可以明显看到，当组件 B 超过自身极限后，导致组件 A 调用延迟增大，组件 A 的有效输出断崖式下跌，远远达不到自身的处理极限。这也是组件 A 运维判断自身不需要扩容的决策依据。

案例分析

这个案例确实比较复杂，游戏的特殊性在于其受环境因素影响较大。本次故障并非单一事件，而是一个多米诺骨牌效应。最初是活动进行中，组件 B 出现问题，修复组件 B 后，却紧接着导致了

组件 A 的问题。值得注意的是，组件 A 的问题也受到游戏活动自身特性的影响。整个故障处置过程因此变得非常复杂。

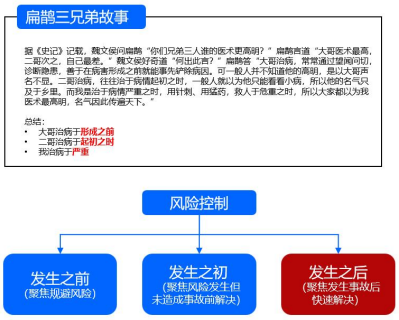
复杂性根源：游戏业务受外部环境因素影响大，故障往往不是独立出现而是层层传导。

故障链条：活动触发 → 组件 B 故障 → 修复组件 B → 导致组件 A 故障。

活动影响：组件 A 的故障也与游戏活动本身的特性相关，进一步增加了复杂性。

在这种复杂场景下，我们发现单凭人工判断和处置能力，虽然能快速解决某些局部问题，但整个故障解决链条可能反而被拉长。这种不确定性的场景，我们认为更应该依赖工具化的手段来解决。

总体思路



应急的可控，其实就是降低不确定性，所以标准上要往这里靠：
- 对于确定性高的事情，要极致的优化，消除潜在影响确定性因素。
- 对于确定性低的事情，要极致的推演，提升确定性。

其实应急响应就如同扁鹊治病，当我们在思考如何解决应急响应的问题时，其实应该站在整个风险控制生命周期去思考。

对于风险控制，需要有一个清醒的认知，事故是无法避免的，所以对于风险控制的关键不是消除故障，而是做到可控：时间和空间的可控。

而要做到可控，一种思路是定义规范、标准，然后通过工程化落地确保标准的落实。

环节	约束条件		
	人力资源	硬件资源	时间
发生之前	敏感	敏感	不敏感
发生之初			
发生之后	不敏感（确定性高）	不敏感（确定性高）	敏感

如何建设通用应急响应平台？

尽管本次分享的核心聚焦于“应急响应”，但我们必须强调一个根本理念：应急本身并非风险控制的全部，它更准确地说，是风险控制链条中的“最后一公里”。将整个风险控制体系视为一个流程，其核心环节理应涵盖故障或风险发生之前、发生之初以及最终的发生之后。我们的主要精力和资源，必须压倒性地投放在前两个阶段，而非仅仅局限于最后一个“发生之后”的应急处理。

通过真实数据能够最直观地印证这一理念的有效性：

- 在过去一个月中，某款游戏共发生 80 余起故障事件。
- 其中，有超过 70 起故障，得益于前置的故障自愈机制，得到了自动化解决。

这意味着，在风险预防和故障发生之初投入的资源与努力，能显著减少需要人工介入的情景。因为一旦进入应急响应阶段，本质上仍需依赖人员进行干预处理，而我们始终追求的目标是尽可能减少或消除对人工操作的依赖。前置的投入越多，自动化、自愈能力越强，最终需要团队进入“备战”状态去应对的故障就会越少，SRE 团队的核心价值也由此获得更深层次的体现。

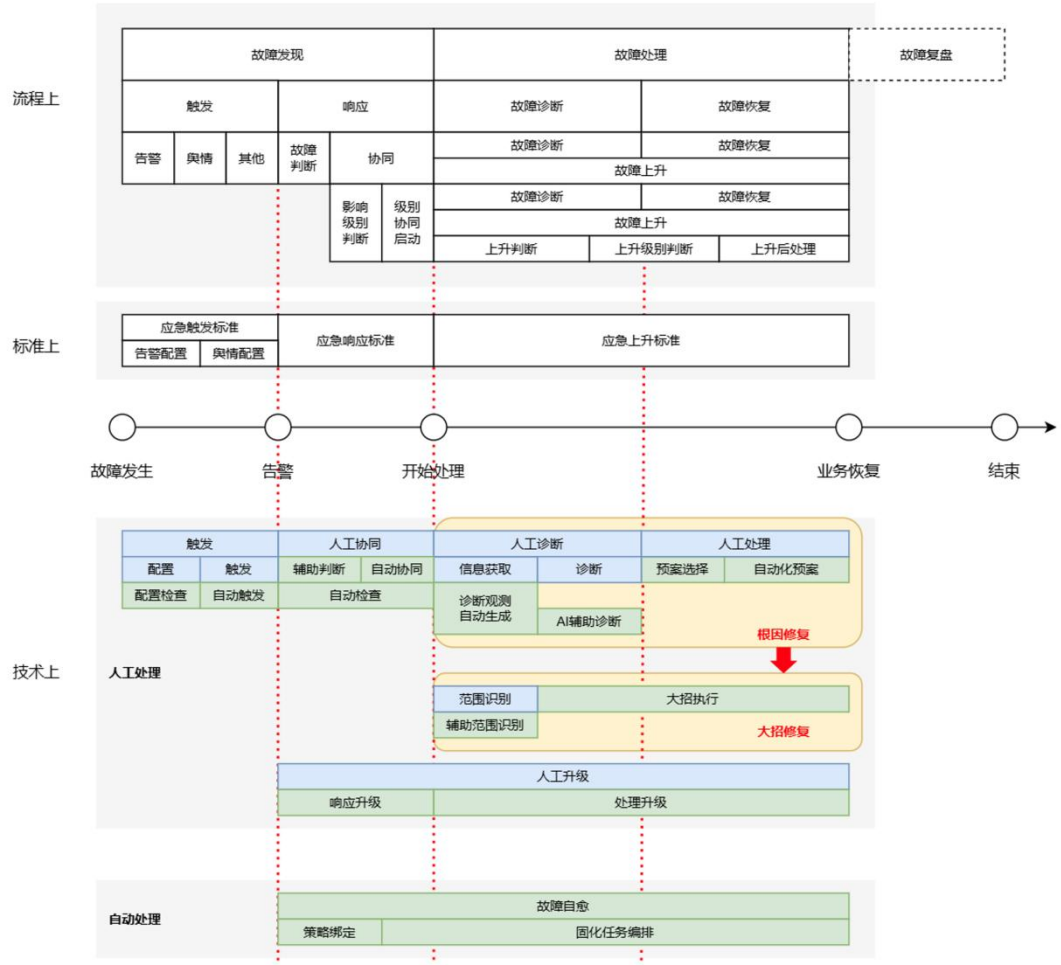
接下来，我们将深入探讨如何构建通用的应急体系，但这依然是对前述理念的延续和支撑，旨在即便在故障发生之后，也能通过结构化、体系化的建设，提高应急处理的效率和效果，最大化减少故障带来的影响。

（二）体系设计详情

基于蓝鲸底座的应急响应方案建设

方案思路

我们将从方案设计的视角切入，聚焦于如何利用统一的基座平台，构建一套通用的应急响应解决方案。



我们在构建这样一个通用的应急响应方案时，核心的理念在于追求极致的“确定性”保障。因为在应急场景下，快速有效地解决问题是首要目标。我们深信，在整个应急处置流程中，那些确定性高的环节应当避免任何人为失误的引入；而对于那些确定性较低的环节，则应充分 使用工具和辅助手段，以提升整体的确定性水平。正是基于此思考，我们构建了一个涵盖“流程-标准-救障（即故障处置）”的完整链条。正如你从图上看到的那部分被标注为绿色的区域所示，这些都是我们经过仔细拆解后，识别出可以通过工具和技术手段有效解决的环节。

我们整个应急响应平台和体系的建设目标主要聚焦于三大方向，旨在解决我们前面提到的诸多痛点：

首先，我们立志于消除信息孤岛，实现信息的实时共享。这意味着所有相关的监控数据、告警信息、人员状态和处置进展都应该在第一时间触达需要的人员。

其次，我们将标准化的应急工作流程通过平台工程化落地，以此来避免多团队协同过程中可能出现的混乱和效率低下问题。

最后，我们致力于通过平台能力辅助信息的采集、决策的制定以及执行的落地，从而显著提升整个应急响应环节的执行效率。

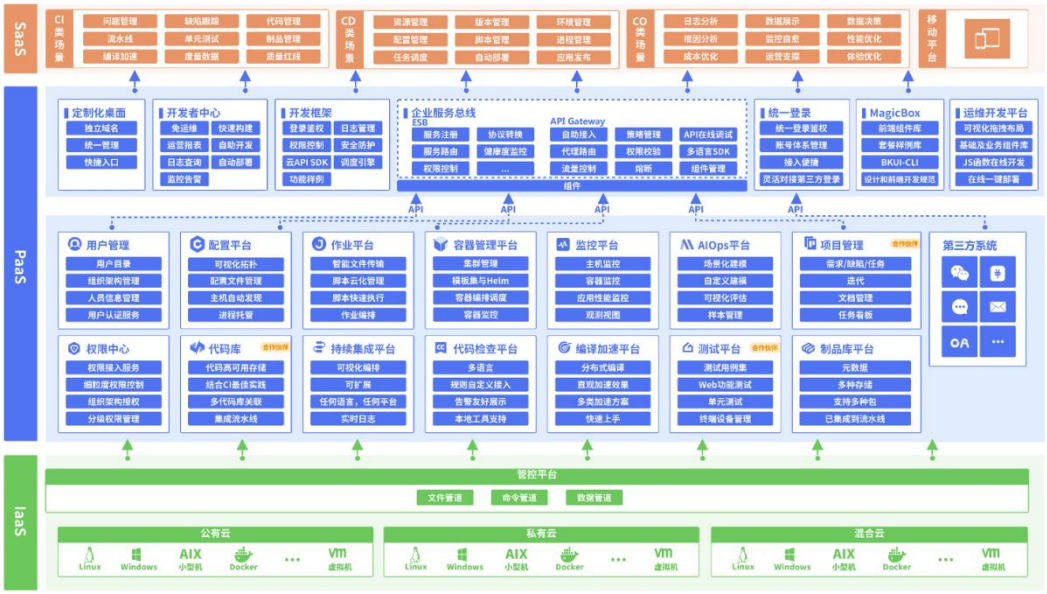
为实现这些目标，我们的平台需具备如下能力：

- 提供便捷的方式来获取监控信息、人员状态；

- 支持高效的观测手段；
- 并且能够执行策略驱动的任务。

其实蓝鲸平台天然满足这些需求。

通过蓝鲸基座便捷构建应急系统



在构建应急响应体系的基石层面，我们充分利用了公司内已有的成熟平台能力：

权限中心： 提供了便捷、集中的全线权限管控能力，这是应急响应过程中进行权限调整和隔离的基础。

配置平台： 精确、完整地存储了包括人员信息和 拓扑结构在内的游戏业务关键配置信息，在故障定位和人员拉起方面至关重要。

监控平台： 能够提供从设备层到上层应用层全链路的、完整的监控信息，是故障发现和根因分析的眼睛。

正是基于这些已然具备的平台能力，我们得以更加顺畅、高效地设计并搭建了我们的应急响应体系。

方案架构



这个应急系统架构展示了不同层次的组件和功能模块，它们协同工作以确保有效的应急管理。下面是对各个层次和功能模块的描述：

入口层

- **功能：** 作为系统的消费入口，可以快速启动应急响应。
- **组成：**
  - **蓝鲸监控：** 通过监控告警启动应急响应。

- BKChat：通过信息流启动应急响应。
- **接口和页面**：通过系统页面启动应急响应。

能力层

- **功能**：负责处理核心的应急管理和资源管理任务，包含多个重要模块，确保应急响应的能力。
- **组成**：
  - **预案管理**：用于管理应急预案，包括演练计划、预案流程等内容。
  - **灾备管理**：管理设备的使用和调度，保障在应急事件中的设备需求。
  - **应急演练**：执行和规划应急响应的演练，以提升团队响应和处理的能力。
  - **应急管理**：在应急事件发生后，进行统一的管理和响应。

数据层

- **功能**：数据存储和管理层，提供系统操作所需的数据支撑。
- **组成**：

- **CMDB 数据**：包括业务人员信息、业务拓扑信息、资产信息等数据，为应急管理提供必要的资源信息。
- **监控平台**：提供实时监控和告警功能，用于识别和响应潜在的系统故障。
- **告警事件**：实时处理告警信息和事件，并实施应急响应。

平台层

- **功能**：为系统提供支持和管理服务，保证各个模块之间的协同工作。
- **组成**：
  - **监控平台**：支持实时监控和数据流转。
  - **AI 开发平台（AIDev）**：集成 AI 相关功能， 包括 Agent 开发的配套能力
  - **标准运维平台**：为运维和操作提供标准化服务，确保应急操作的一致性。
  - BKVision：用于对业务数据进行大屏可视化
  - **蓝鲸配置平台**：负责系统配置和资源调度信息。
  - **权限中心**：管理系统的访问权限和安全控制。

在我们的应急系统架构中，底层依托的是蓝鲸平台所提供的各项核心能力，包括监控平台、标准运维平台、BKvison，信息流等。这些平台能力是整个架构能够正常运行的基础。

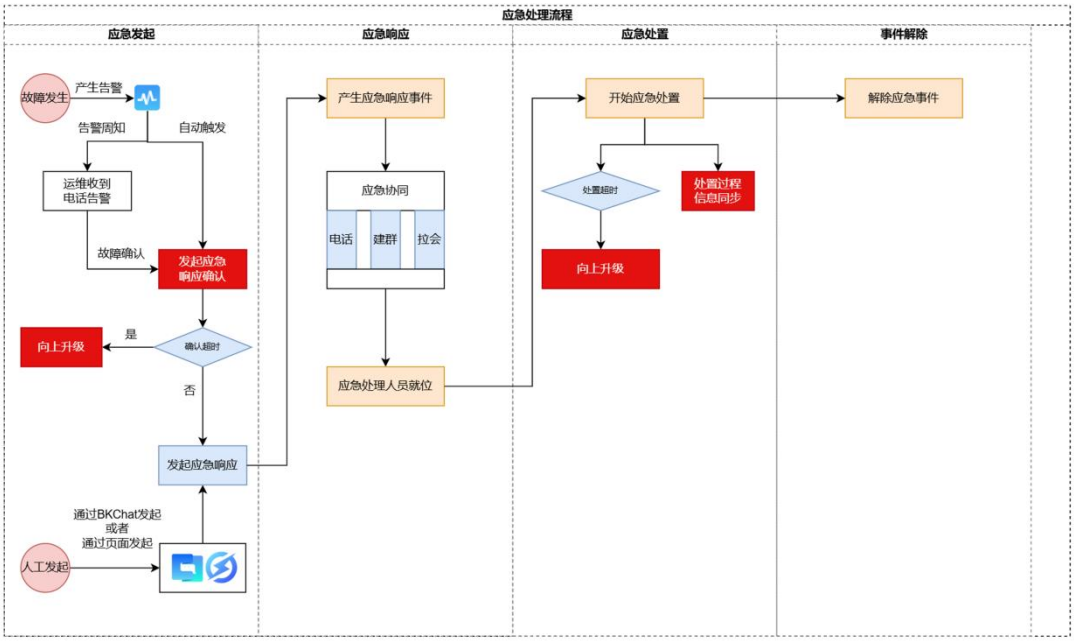
在数据层面，我们利用 CMDB 系统来集中管理和存储各种业务信息，包括业务人员信息、业务拓扑信息和资产信息等。CMDB 的数据层整合了各类业务相关数据，从而为应急响应提供了必要的信息支持。当然监控的数据更是关键。

我们的主要研发工作量来自能力层。我们开发了包括预案管理、灾备管理、应急管理、应急演练等功能模块。

- 预案管理主要负责应急预案的创建、更新和管理，确保在紧急情况下可以迅速执行。
- 灾备管理是针对灾备预案的建设，在面对容灾类故障时可以快速一键恢复。
- 而应急管理则聚焦于对发生的各类应急事件进行追踪和处理。
- 应急演练聚焦通过既定的计划实施故障演习，提升人员对于故障处理的熟悉程度。

此外，这一系统是与我们的合作伙伴嘉为共同协作共建的。我们将腾讯游戏的经验与嘉为在行业应用方面的经验相结合，形成了此应急系统。虽然这一系统最初应用于腾讯游戏领域，但其设计理念和架构是通用的，可以适用于其他行业的应急管理需求。

应急管理流程概览



应急管理流程的核心在于确保故障发生后，能够迅速响应并有效处理。具体来说，整个流程主要划分为发起、响应、处置与解除几个关键阶段，其中我们特别关注以下几个方面：

首先，确保故障发生后，运维能够**及时响应**。响应流程通过**监控触发告警**，发送确认通知启动，运维人员（如 SRE 或运维工程师）必须在规定时间内**及时处理告警**。如果未能响应，系统会自动将告警**向上升级**，通知部门经理。响应完成后，系统会自动启动**应急群聊**，促进**跨团队协同**，确保相关人员迅速加入并高效合作。

在故障的处置阶段，我们将**故障诊断与故障处置**进行了整合，因为在实际修复过程中，这两部分常常无法明确区分。整个过程中，系统自动收集并记录所有**关键信息**，并同步给所有相关人员，无需人工干预，直到故障被完全解决。

此外，为了确保**上级及时了解整个处理过程**，系统会实时同步故障处理进展。这种设计优化了**响应效率**、**跨团队协作**以及**故障升级机制**，使我们能够在故障发生后迅速启动应急响应，确保各项工作能够顺利进行并高效处理。

接下来，我将详细阐述如何通过这一流程提升系统的确定性，涵盖从告警的接入到故障复盘的整个过程。

告警配置

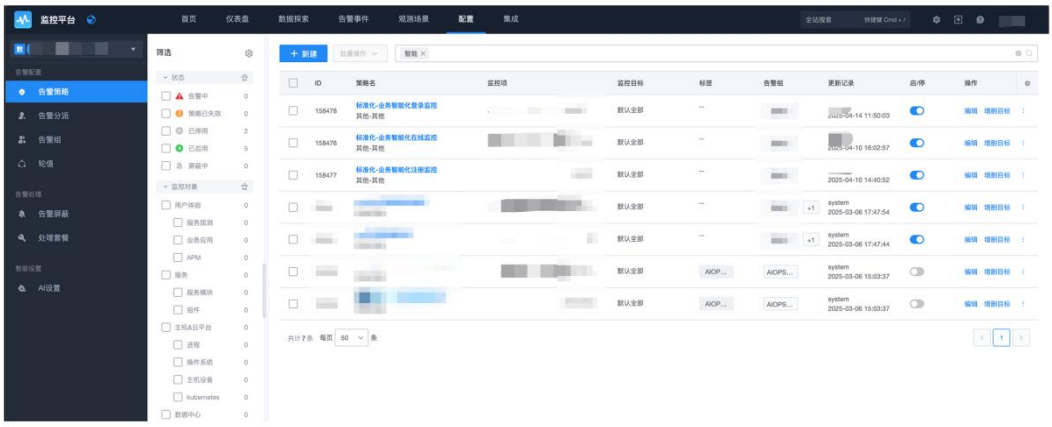
通过巡检确保告警正确接入



面对告警信息爆炸的现实困境，团队务实地选择聚焦核心，通过建立一套标准化的告警接入机制，旨在有效识别并及时处理真正关键的告警，从而提升整体应急响应效率。

核心思路概括为：

- 聚焦重点，而非全面接入：鉴于告警数量庞大且良莠不齐，团队明确无法也不应将所有告警接入，而是优先筛选并接入那些能够反映核心业务问题的通用性告警指标。



- 标准化告警接入规范：在蓝鲸监控平台上，制定了一套清晰的告警接入标准。业务团队遵循此标准进行告警配置，确保接入的告警具备有效性和代表性。（注意上图告警策略标准化字眼）

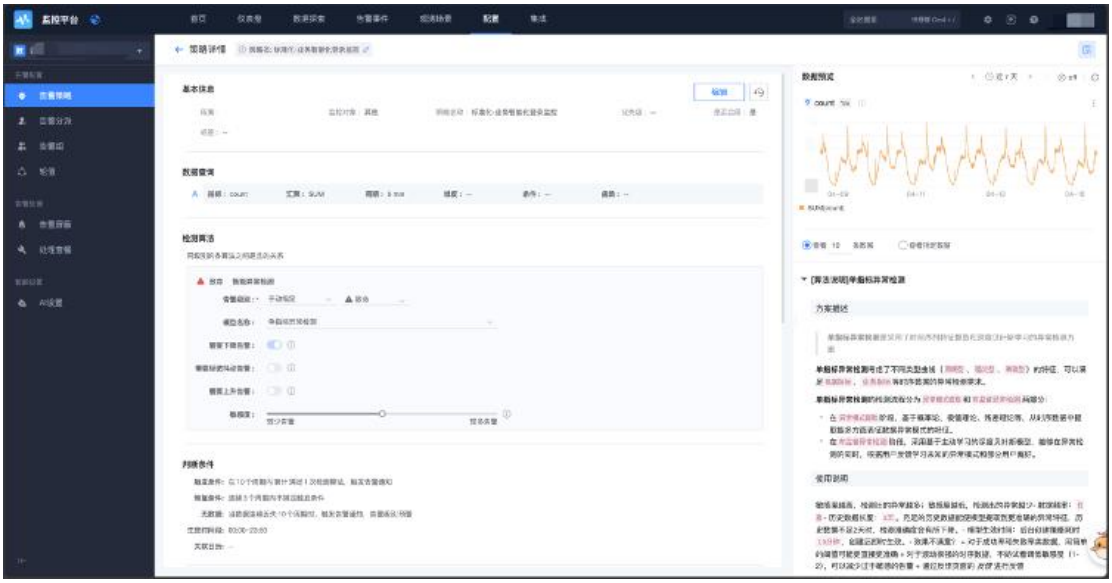
- 巡检确保告警正确接入：配套告警接入标准，构建了巡检系统。该系统定期扫描并检测业务团队的告警配置是否符合既定标准。

- 自动化跟进与升级机制：对于发现的不符合标准，例如未配置某项关键告警的情况，系统会自动生成工单提示业务团队限时整

改修复。若业务团队未能在规定时间内完成修复，系统将自动将问题升级并通知到其上级管理者，形成强制性的闭环管理，确保关键告警的覆盖率与准确性。

通过这种规范化的配置与自动化的巡检及提醒机制，腾讯 IEG 团队有效地解决了告警接入阶段人为失误的潜在风险，确保了关键业务指标能够被及时有效地监控，并在发生异常时能够快速触发响应流程。这套流程不仅提升了告警的有效性，也通过自动化的方式减轻了人工检查和跟进的工作量，保障了应急响应体系的基石——准确且及时的告警信息源。

智能异常检测（无需配置阈值、无需配置同环比）



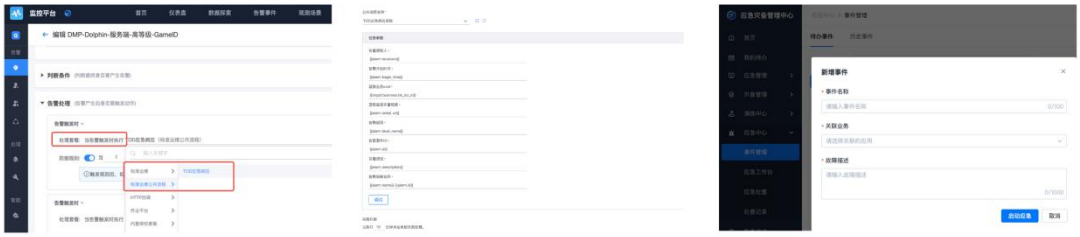
我们的监控平台支持单指标智能检测功能，这意味着一旦那些关键的核心指标接入监控系统后，系统将自动为这些指标配置智能

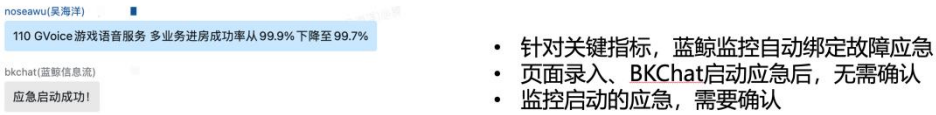
检测策略。通过这一手段，我们成功地杜绝了因人工配置失误而引发的问题的可能。

完成监控配置后，下一步便是将监控与应急响应紧密绑定，实现从发现问题到解决问题的快速联动。

应急机制

应急响应触发：监控平台自愈流程接入





- 针对关键指标，蓝鲸监控自动绑定故障应急
- 页面录入、BKChat启动应急后，无需确认
- 监控启动的应急，需要确认

应急响应的触发可以由人工报告，但更主要的手段是通过监控系统进行自动触发并介入处置。在腾讯 IEG 的实践中，我们广泛使用了蓝鲸监控的处理套餐作为触发机制。

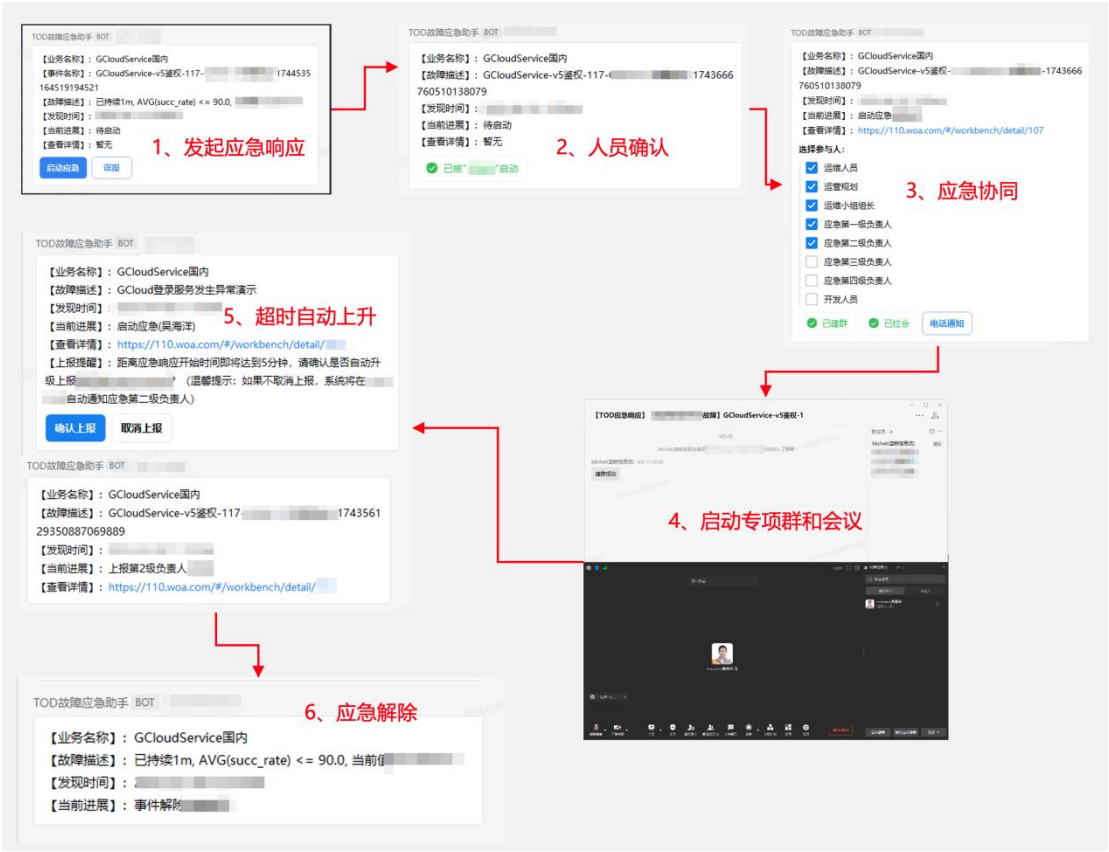
具体来说，该套餐提供了标准化的公共配置，并且设计为无需人工干预即可自动生效。只要系统中的关键指标被接入并，系统会自动绑定到这个套餐，当这些指标出现异常时，系统将自动完成以下步骤：

告警生成： 系统会自动根据预设规则生成告警。

套餐调用： 告警触发后会同时调用，例如自动执行自愈脚本、触发降级或限流策略等。

应急触发： 如果自愈失败，触发应急响应流程。

应急系统使用概览



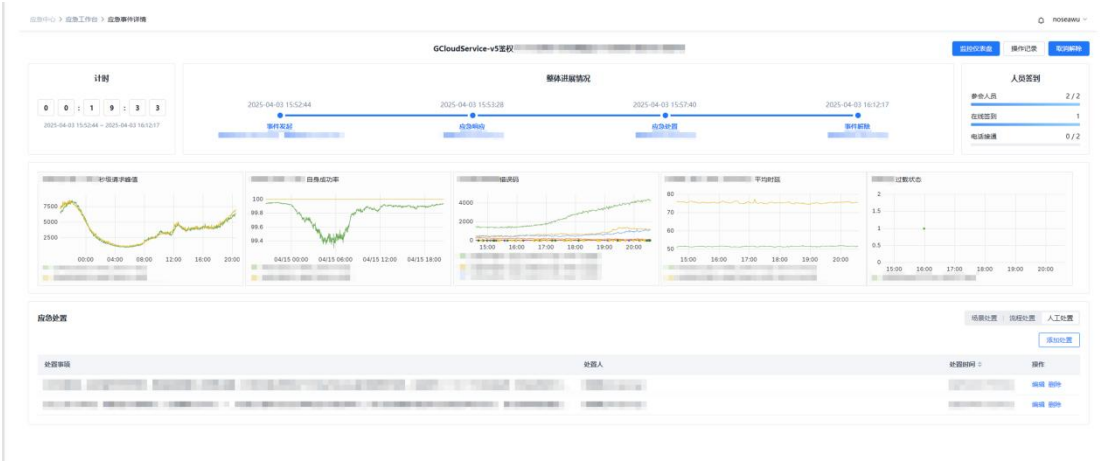
针对应急响应流程，我们首先从监控发起告警开始。

告警触发后，会启动一个确认环节，因为需要人工判断告警的真实性，排查误告的可能性。

确认告警有效后，应急响应人员随即开始协同，在这个阶段，系统支持根据组织结构配置和选择相关的协同人员。

应急响应人员确定后，1 分钟内启动紧急响应并触发相关人员拉起专用会议，当前主要以群聊和腾讯会议为载体，其中会议被认为是协同效率最高的方式。

在整个处置过程中，系统将全程收集处理信息，直至问题得以解决。此外，如果处置过程发生超时，系统会自动进行升级处理。



当前界面是整个应急响应流程的可视化载体，其设计重点在于突出关键信息，确保参与处置的各方能够快速、准确地掌握事态进展：

- 时间信息：界面清晰地标识应急事件的起始时间，这是判断事件持续时长和评估影响范围的基础。

- 处置人员信息：明确展示当前事件的处置负责人或团队，便于快速沟通和任务分配。
- 核心指标数据：实时呈现与事件相关的关键业务或系统指标，例如错误率、延迟、流量等，这些数据是判断异常状态、评估影响程度及验证恢复效果的重要依据。

这张幻灯片聚焦于应急协同的前置环节，即如何在事件发生初期通过界面聚合信息、提升协作效率。而后续的应急处置环节，其主要的耗时瓶颈往往集中在根因定位这个阶段。

上升机制：响应上升和处置上升

响应上升：

•响应上升指运维人员未及时对应急事件进行响应

•如果运维人员电话未接通，会逐级上升，直到其中一个层级电话打通

•如果运维人员电话接通，但是未能对应急事件进行确认，会在设定时间超时后进行上升

处置上升：

•处置上升指运维未在规定时间内对完成故障修复而进行上升

•按照启动应急开始计算，每到一个层级的处置超时时间，就会对该层级人员进行电话周知



这里，我们着重讲一下应急事件的“上升”机制。简单来说，如果一个事件被系统自动检测并触发告警后，在指定的时间窗口内未得到及时处理或确认，系统将自动启动逐级上升流程。上图看到，这个时间阈值是可配置的。例如：

- 如果在 3 分钟 内没有对告警进行确认响应，事件将自动上升到第一级处理人。
- 如果在接下来的 5 分钟 内仍未得到处理，事件将进一步上升到第二级，通常是你的直属主管（Leader）。

这种处置上升机制，核心在于确保任何异常事件都能在限定时间内被高效率地处理。如果在规定的处理时限内未能解决问题，则

会自动向上级传递，直至足以能够协同足够资源或具备决策权限的上级介入处置，从而避免事件被遗漏或延误。

故障发现和故障诊断

蓝鲸监控：辅助诊断（通过故障自愈套餐触发基础环境状态检查）

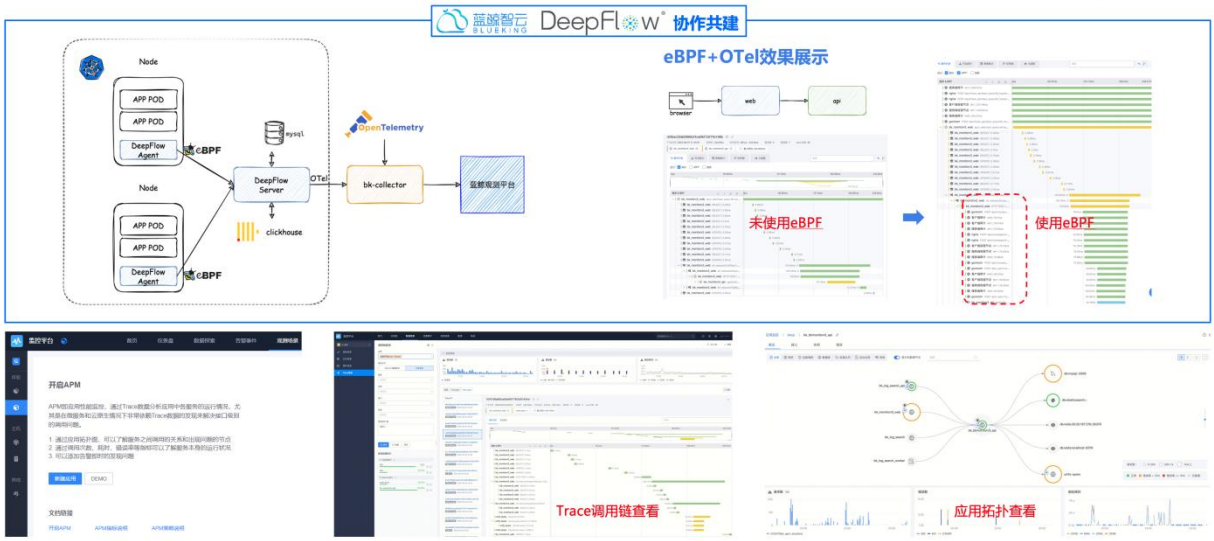


故障自愈除了可以修复明确处理流程的故障之外，也可以用于简单的信息提取来辅助故障诊断。

比如上述案例展示了通过故障自愈实现了对基础环境的健康检查，可以协助运维快速诊断是否故障为硬件、网络等问题导致。

简单的信息提取只能用于一些简单场景的故障诊断辅助，接下来会介绍更深层次的辅助诊断方案。

蓝鲸监控：辅助诊断（APM+eBPF）



以上我们是与 DeepFlow 开源社区协作共建的一个解决方案，整体方案的特性如下；

特性	说明
真全栈覆盖	从应用层 → 系统调用层 → 网络层，做到全链路观测、无盲区
双视角融合	融合 开发者视角（OTel/APM） 与 SRE 视角（eBPF） 数据
零侵入部署	利用 eBPF 技术无需修改业务代码或加载 SDK，大幅降低接入成本
高扩展性支持	支持自定义协议解析（如 TRPC）和业务定制字段（通过 WASM 插件）
全链路数据关联	实现 trace + metric + log + profiling 全数据关联与跳转
云原生一键接入	借助蓝鲸平台，支持容器/K8S/物理机的自动化部署与插件管理

这个 1+1 >2 的方案的协同优势如下：

维度	APM / OTel	eBPF
观测视角	应用逻辑调用链，开发者视角	系统调用、网络流，SRE 视角
接入方式	SDK 插码，易受限（如发行游戏/第三方服务难以改动）	零插码，运行时通过内核抓取，无需业务配合
数据完整性	无法覆盖未接入 SDK 的组件（如 NGINX、代理网关等）	可捕捉所有系统级和网络级调用，包括“盲区”组件
数据粒度与覆盖	高层抽象，但有时缺少系统底层视角	细粒度、涵盖系统层级、底层性能、网络交互
融合方式	使用 TraceID、TCP 序列号、SyscallTraceID 实现两类数据自动关联	能补齐 OTel 缺失组件链路，增强拓扑完整性

基于此解决方案，若业务场景并非极为复杂，基本上已经能够很好地进行根因的定位。

蓝鲸 AIDev：智能辅助分析和根因诊断



LLM 模型服务层

包含“LLM 网关服务”和“LLM 模型集成”两部分。

“LLM 网关服务”负责对外暴露统一的 API 接口，接收上层请求并转发到具体模型。

“LLM 模型集成”负责将不同的大语言模型（比如 OpenAI、Anthropic、内部自研模型等）接入到平台中，并统一管理模型调用、鉴权、限流等细节。

LLM 工具构建层

核心由两个 Agent 组件构成：

LLM Workflow Agent：负责定义和执行多步骤的业务流程（workflow），例如串联一系列模型调用、工具调用和决策逻辑。

LLM Agent Framework：提供通用的 agent 运行时框架，支持插件化地加载各种功能模块。

底部支撑能力包括：

角色管理：定义对话或任务中不同“角色”（system、assistant、user 等）的权限和行为规范。

提示词管理：集中存储、版本化和调优各类 prompt 模板。

工具管理：管理可被 agent 调用的外部工具或服务（如检索、计算、数据库查询等）。

知识库：持久化存储结构化或半结构化的领域知识，供 agent 在执行时检索与引用。

LLM 场景服务层

CI 智能场景（Continuous Integration）：在持续集成流程中提供代码审查、自动化测试建议等。

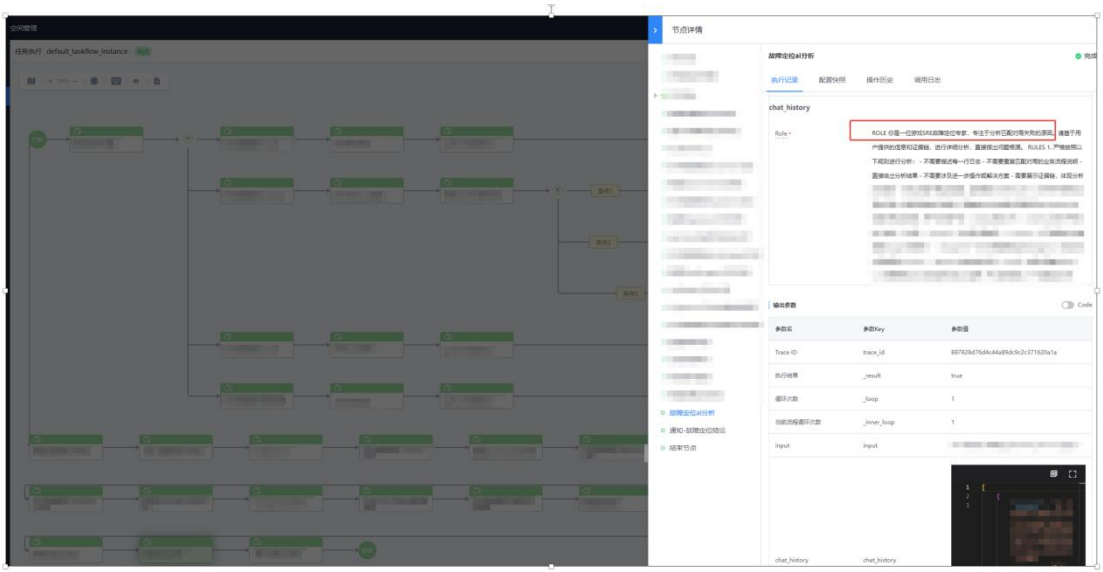
CD 智能场景（Continuous Deployment）：在持续交付/部署环节提供发布策略优化、回滚决策支持等。

CO 智能场景（Continuous Operations）：在持续运维环节提供故障诊断、应急响应辅助等。





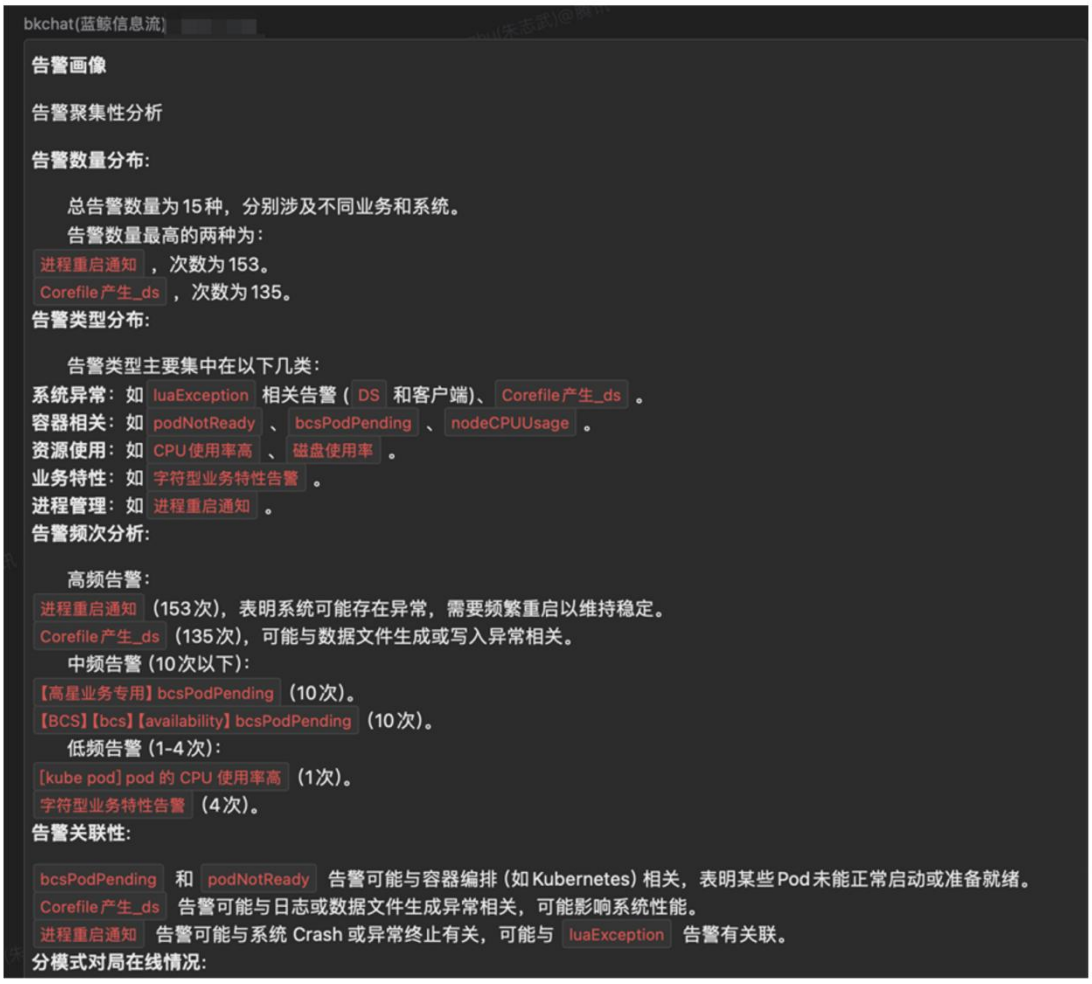
然后我们就分享一个 LLM Workflow 在 SRE 应急响应中的应用。Workflow 这种模式之所以被我们采用，很大程度上是因为它非常适合那些场景相对固定、变化不大的定位场景。



上图可以看到我们通过我们的 AIDev 平台编排了一个固定的 LLM workflow，其中 一个 LLM 节点， 我们使用了一个 “你是一个游戏 SRE 故障定位专家，专注于分析匹配对局失败的原因。请基于用

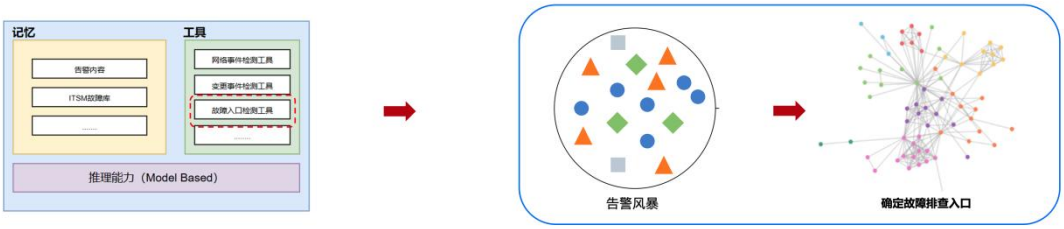
户提供的信息和证据链进行详细分析，直接指出问题根源.....” 去进行辅助故障定位

以下是定位的结果通过 bkchat 蓝鲸信息流反馈给用户。。



具体到生产环境，由于其节点众多且复杂，一个设计良好的 Workflow 能够帮助我们快速地进行一次完整、全面的事故分析。大家可以看到，它最终能够输出一个涵盖整体细节的分析结果，极大地提升了运维同学快速定位问题的效率。

基于 AgentFramework 实现根因诊断



基于“Agent 化”应急方案，整个流程由系统智能触发，无需针对每个场景手动编排 Workflow。首先，Agent 会利用蓝鲸基座提供的入口检测工具对告警进行降噪、聚合并构建实体关系图，从中迅速识别出故障的“入口”服务或组件。确定入口后，系统会在整体网络拓扑中截取与该入口关联的子图，再通过内置的根因定位算法对这段子图深度分析，直接输出故障根因节点。



在这个案例中，业务故障的上层表现最终追溯到 Redis 异常。Agent 方案首先在告警中识别出影响最广的入口节点，随着分析子图的不断收缩，系统自动锁定了 Redis 实例为故障根因。

故障发现：基于 Transformer 和对比学习的故障入口确定方案



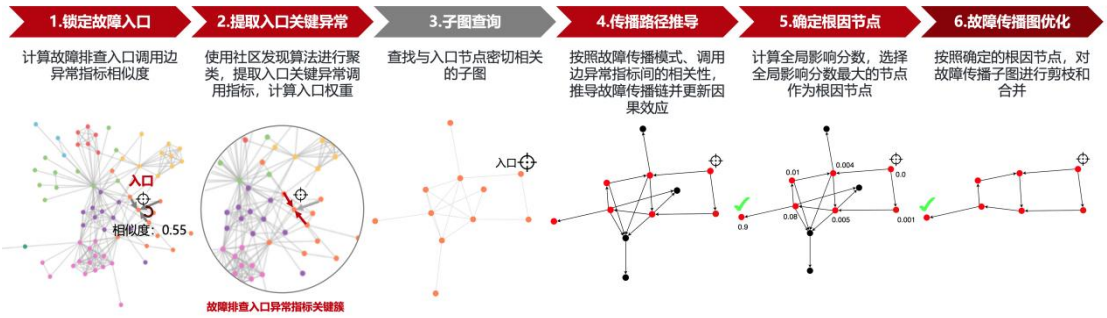
我们构建的应急体系中，关键一环在于提供智能化工具，快速定位问题根源。幻灯片展示的正是其中的一个核心组件，它专注于帮助运维人员高效识别故障的入口。

① 第一步是告警降噪，即去除噪声告警。告警降噪采用 Transformer 模型作为告警语义的理解模型，同时设计告警嵌入方案（指标嵌入+维度嵌入），将告警数据转化为告警嵌入。在训练阶段，以历史告警构建的告警三元组（<基准告警，相似告警，无关告警>）作为样本，通过对比学习的训练方式，让模型可以很好的区分相似告警和不同告警之间的差异。在预测阶段，训练好的降噪模型可以得到每个告警对应的嵌入，将告警嵌入和噪声嵌入计算相似度，相似度大于阈值就预测为噪声告警，就可以进行降噪去除。

② 第二步是告警聚合。首先会把降噪后的告警数量时序进行异常检测，拿到异常的时间点对应的告警。然后这些异常告警根据从第一步告警降噪模型输出的告警嵌入之间的相似度进行聚合。

③ 第三步是关联图谱。将聚合后的告警维度和实体关系图谱（CMDB+BCS）进行关联，从而确定告警实体对应的故障排查入口。

故障诊断：基于社区发现和图的动态规划的故障诊断方案



针对复杂业务系统，其网络拓扑结构往往是极度复杂的。我们依赖图谱数据（CMDB+BCS）及调用关系数据（APM / eBPF 调用指标），推导故障传播图，确定根因节点。 流程如上图所示。

① 第一步是锁定「故障发现」确定的故障排查入口。通过计算故障排查入口调用边上的异常指标相似度，确定异常指标关键簇。

② 第二步是提取故障排查入口关键异常。通过使用社区发现算法对异常指标进行聚类，提取入口关键异常调用指标，计算入口权重。

③ 第三步是子图查询。从故障入口出发，找到和入口节点密切相关的实体关系子图。

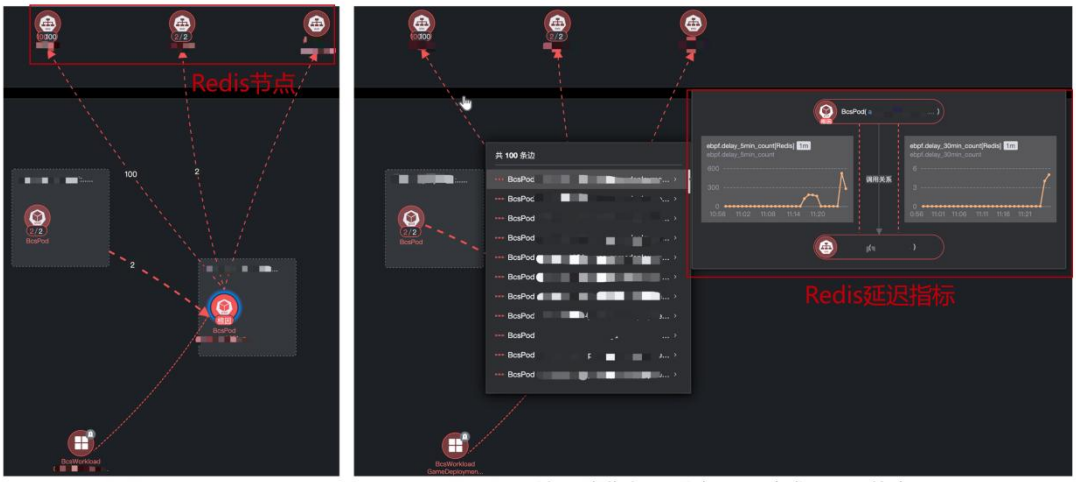
④ 第四步是传播路径推导。沿着调用边的方向去进行游走推理，如果满足预设的故障传播模式，就会将这些点标记为红色，代

表异常。同时计算调用边和被调用边之间异常指标的相似度，来更新故障传播图调用边之间的因果效应。

⑤ 第五步是确定根因节点。经过上面的游走推理，整个故障传播子图已经拿到了，计算故障传播子图所有节点的全局故障影响分数，选择全局故障影响分数最大的作为故障传播子图的根因节点。

⑥ 第六步是故障传播图的优化，即故障传播子图的剪枝和合并。剪枝是指去掉故障子图里面没有异常或者不符合异常传播模式的边和节点；同时每一个入口都能拿到对应的一个故障传播子图，取全局异常分数最大的根因节点作为最终的根因节点，同时合并根因节点和最终根因节点一致的多个故障传播子图。

故障诊断案例：Redis 延迟引发的故障：

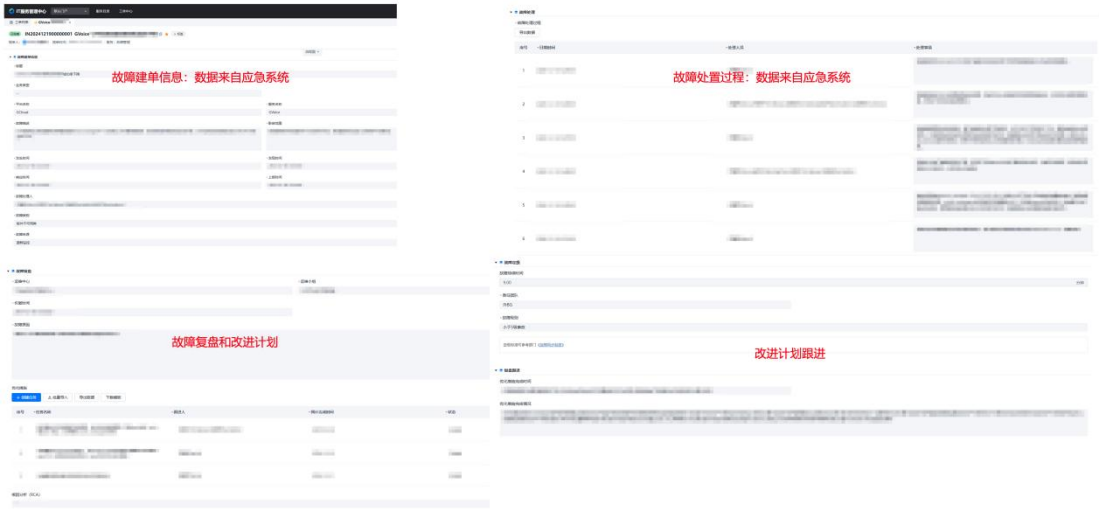


故障传播图　　调用边异常指标（业务Pod请求Redis节点）

上图通过一个生产环境中的实际案例，来具体剖析前面提到的应急响应机制是如何发挥作用的。

这个案例是由 Redis 延迟引发的业务级故障，左图是故障传播图，右图点击业务 Pod 到 Redis 节点的调用边，可以看到对应的 Redis 延迟指标。正是通过前述的故障排查和分析方法，才得以快速定位到其深层原因：Redis 出现了异常。

故障复盘：通过 ITSM 系统实现复盘和改进



接下来我将分享我们在故障复盘方面的实践，这页幻灯片展示了我们在 ITSM 系统在复盘流程中扮演的关键角色。

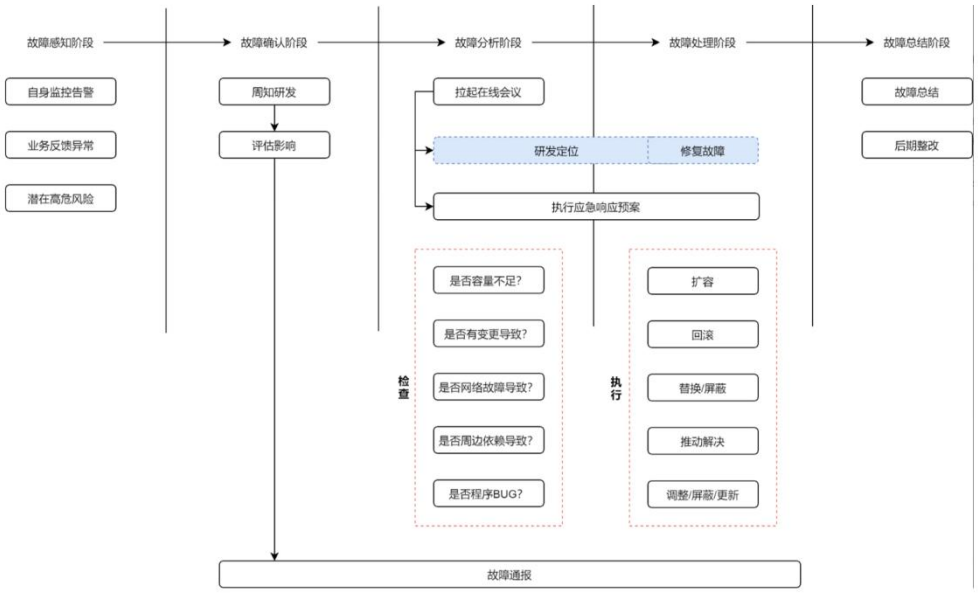
流程化优势：ITSM 系统最显著的优势在于，它能够方便地帮助我们固化并制定流程、能够让各业务团队在故障处理和复盘过程中遵循统一的标准。这确保了不同团队在应对故障时，拥有统一且规范的行为模式。

系统联动与人员协同：ITSM 强大的联动能力使其能够与其他周边系统无缝对接，实现信息互通和流程自动化。同时，它也能有效连接所有相关人员，无论是协调跨团队响应，还是推动后续改进措施，都可在同一平台上完成。

通过将故障复盘与跟进集中到应急系统中，我们能够更便捷、高效地完成整个复盘过程，并确保后续改进措施得到切实落实。

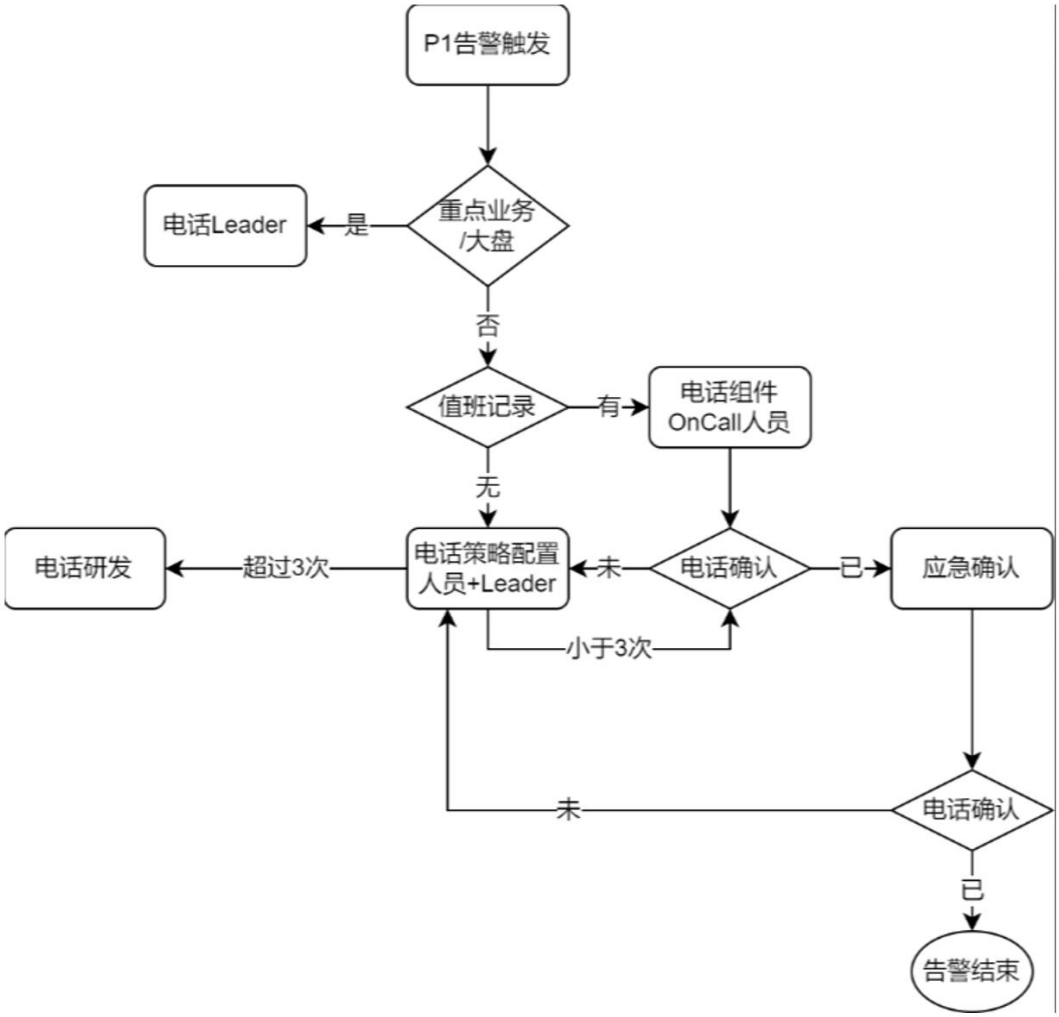
执行应急响应的思考和经验

关于应急响应，我们的经验是是，必须高度重视流程建设，以下是我们一些流程规范的示例



分类	术语	定义
告警级别	P1	•该级别告警触发时，需要立即进行处理。 •通知方式：电话+企业微信
	P2	•该级别告警触发时，无需立即进行处理。 •通知方式：企业微信
响应级别优先级：一级>二级>三级。	一级	定义：关键路径异常，且持续时间大于 10 分钟，需跨团队关注与协作处理 •该级别要求立即周知至总监以及部门重大故障群。 •该级别要求研发、运维立即共同响应。
	二级	定义：关键路径异常，且持续时间小于 10 分钟，需跨团队关注与协作处理该级别要求研发、运维共同响应。
	三级	定义：非关键路径服务异常，且持续小于 10 分钟，无需跨团队协作处理，该级别要求运维响应。
业务级别	重点业务	包含收入活跃>xxx 的游戏，以及自定义添加的游戏。

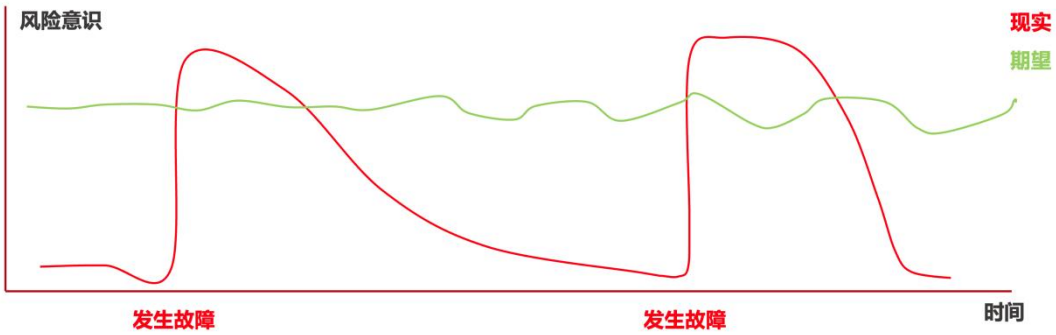
角色	责任	候选人员	说明
事故总控	负责组建事故处理团队、分配任务给团队成员，协调周边和资源以便更快速的处理故障。	运维负责人研发负责人	运维负责人负责运维侧协调安排 研发负责人负责研发侧协调安排
事故处理	处理故障	直属运维直属研发	运维作为现网唯一操作人员
发言人	本次事故对外发言人，职责包括： •向关注事故的人周期性的发送事故进展 •维护当前事故文档，保障其及时性和正确性	运维负责人服务经理	



正是因为在整个组织层面，应急流程可能相对宽泛，对于单个业务，尤其是核心业务，流程的建设至关重要。前两位讲者也都特别强调了流程建设的重要性，而流程建设的核心目标在于：

- 确立全链路确定性：通过预先规定的步骤和规范，确保在应急场景下各环节的确定性执行。
- 明确协作边界：精确划分各团队和角色的职责边界，避免出现应急响应时责任不清、忙乱无序的情况。

常态化应急



常态化应急：以小博大，培养高风险意识与实战能力

在经历多场实战后，我们深刻认识到，常态化的应急至关重要。我们曾满心期望在应急发生时能如预设般高效运转，遵循那条理想的绿色曲线。然而现实往往是，在故障发生之前，团队可能处于一种松弛的状态，一旦真正的故障来临，却可能手忙脚乱，走上那条充满混乱与失序的红色曲线。

因此，不能只在发生大规模、轰动性故障时才启动应急体系。即使是体量较小、影响范围有限的故障，也应当按照既定的应急流程进行处置。

推行常态化、覆盖大小故障的应急机制，其核心目的在于：

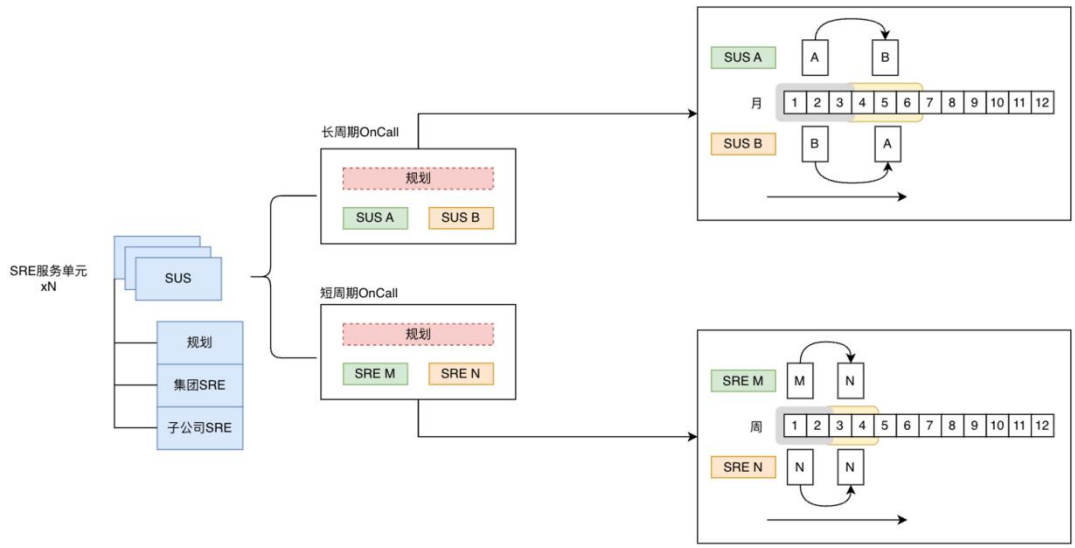
固化流程，强化操作熟练度：通过反复实践，使应急响应流程深入人心，团队成员能够在压力下保持清晰的头脑和标准化的操作。

培养高风险意识：持续的应急训练能够不断提醒团队风险的存在，促使每个人都对潜在的问题保持警惕，从而在日常工作中更加注重稳定性和安全性。

提升团队协同效率：在模拟或实际的小规模应急中磨合团队间的协作方式和沟通机制，确保在大故障发生时能够迅速进入“作战状态”。

通过将应急处置融入日常工作，即使是微小的故障也能成为学习和提升的契机，从而筑牢应对更大挑战的根基。

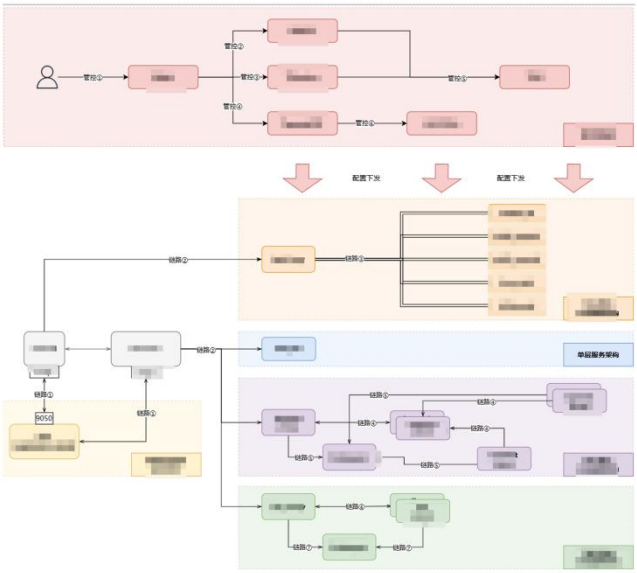
组织建设：OnCall



我们做“on-call”制度，其根本目的在于消除人员层面可能存在的单点依赖。毕竟，在实际工作中，我们确实遇到过这样的情况：当紧急事件发生 SRE 同学处理时，关键的负责人却因为各种原因（比如恰好不在服务区）而暂时失联，这并非个人责任问题。

而“on-call”机制的优势恰恰在于，一旦我们全面推行了这套体系，当某个服务的原负责人无法响应时，系统中会指定另一位熟悉此服务的 SRE 同学迅速介入，接手并解决问题，确保故障处理的及时性和有效性。

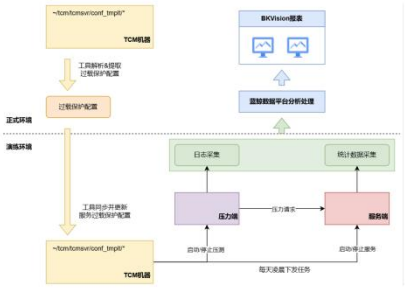
断路分析



- 列出所有管控、服务链路
- 周期实施
- 拆分成原子链路
  - 说明链路功能
  - 是否具备容灾能力
  - 是否具备自愈能力
  - 链路故障影响
  - 链路故障恢复预案
  - 恢复预案自动化程度

断路分析，我们认为，是实施稳定性保障手段中尤为有效的一种方法。它要求在管控系统以及整个服务链路的每一个关键节点上，都详尽地标记出可能存在的问题点，并明确如何进行修复，以及是否具备相应的自愈能力。

确保过载保护配置的有效





针对频繁面临流量突发现象的业务系统，过载保护配置的有效性至关重要。配置变更、程序修改或基础设施环境调整都可能导致该保护失效，一旦突发流量超出系统容量且自动扩容（如 Kubernetes 弹性伸缩）无法在 2－3 分钟内完成，系统就存在崩溃风险。

解决方案

我们引入混沌工程，构建每日自动化验证流程：

- 自动触发各类过载场景
- 验证过载保护配置在真实压力下的弹性与鲁棒性

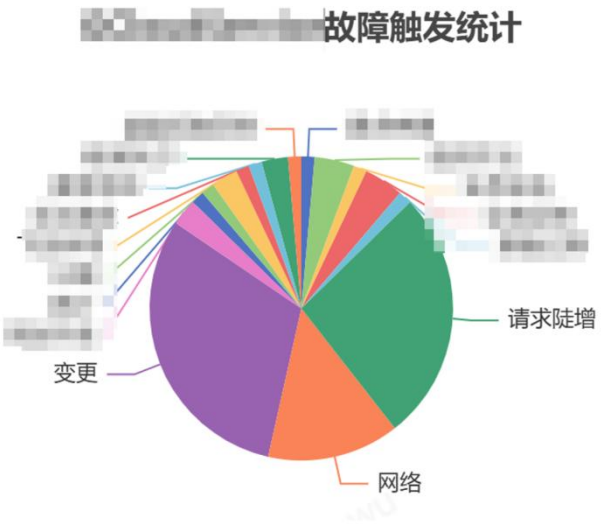
- 监控验证结果并生成报告，确保及时发现配置失效

价值与成果

此验证机制可在极端流量突发时保证系统保持最低可用状态，避免因过载导致雪崩。它为构建高韧性系统提供了关键保障。

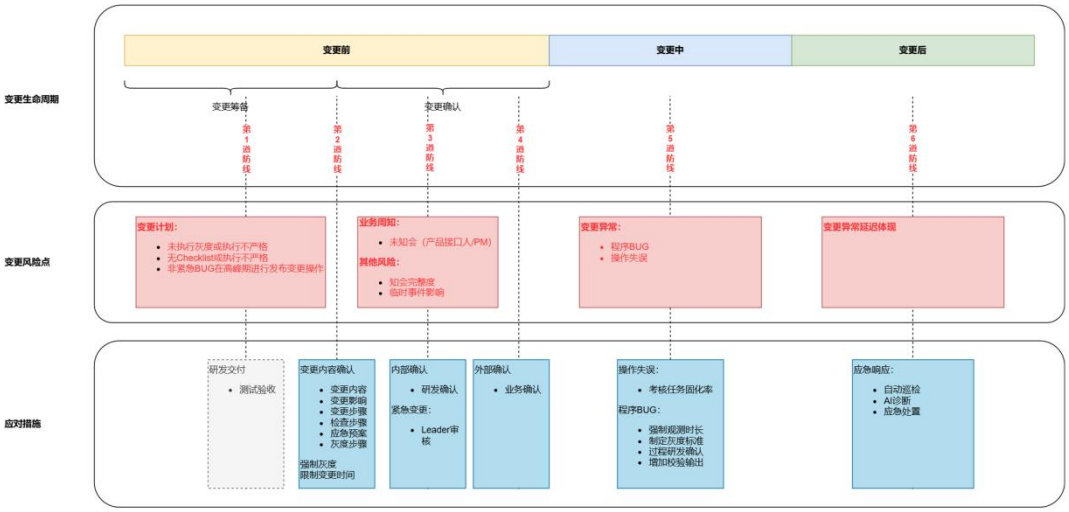
针对频繁面临流量突发现象的业务系统，确保过载保护配置的有效性至关重要。我们在实践中借助混沌工程，通过每日自动化的验证流程，持续检验系统过载配置的弹性与鲁棒性。

变更风险



某组件故障触发分析

在我们探讨如何构建高效的应急响应体系时，变更无疑是一个需要重点关注的关键点。正如之前的专家们所强调的，很大一部分生产环境故障都与变更操作紧密相关。根据我们某个组件的数据统计，大约有 32%的故障直接或间接源于变更。



面对这一挑战，我们在变更管理流程中构建了多重防护防线，旨在最大限度地降低变更引入故障的风险。

上图展示了“变更防控”流程在生命周期各阶段的风险防护与应对措施部署。

整体流程分为“变更前”“变更中”“变更后”三个阶段，并在六个关键节点设置六道防线，以拦截不同环节的潜在风险。

在“变更前”阶段，第 1、2 道防线通过变更策划和预实施核查拦截风险：包括未执行或不严格执行灰度、无或不严谨执行 Checklist、非紧急 BUG 在高峰期变更等。应对措施要求研发交付后

进行测试验收（灰度演练），并在变更内容确认环节对变更内容、影响范围、步骤、应急预案及灰度方案强制审批、限制时间窗口。

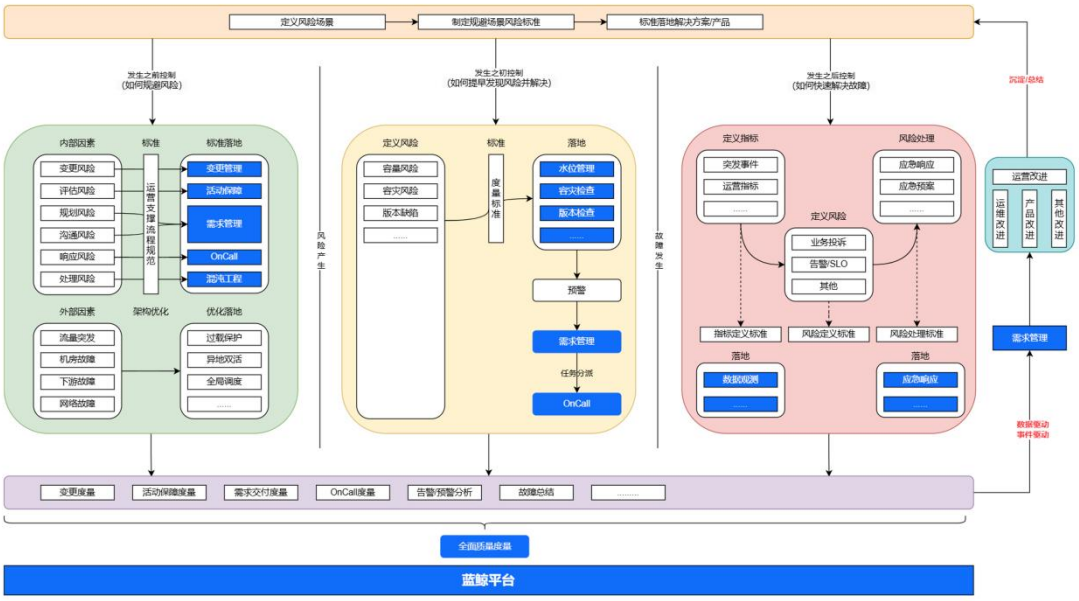
进入“变更确认”节点（第3、4道防线）时，内部（研发Leader审核）与外部（业务/产品PM确认）双重把关，紧急变更需额外复核，方可推进到“变更中”。

在“变更中”阶段，第5道防线重点防范操作失误与程序BUG。操作失误通过考核任务固化率触发人工干预；程序BUG则通过延长观测时长、制定容量/性能验收标准、增加回滚出口等措施防止故障扩散。

变更完成进入“变更后”阶段，第6道防线通过自动巡检、AI诊断与应急处置工具，及时发现并修复潜在异常，将延迟或残留问题反馈至变更管理台账，形成下一轮优化闭环。

更为重要的是，这些变更管控措施必须与整体的应急响应流程紧密集成。因为一旦不幸出现由变更引发的故障，最快速有效的应对手段通常就是立即执行回滚操作，或者进行流量切换，从而迅速恢复服务的正常运行。此时，迅速的行动远比深入分析故障原因更为紧迫和有效。

（三）总结和展望



应急只是风险控制链条中的一环。要想在事后应对时游刃有余，就必须把精力更多地投入到前端：基于业务特点设立可执行的标准和规范，依托平台能力打造相应工具，将这些规范真正落地。架构越健壮、平台能力越强大，后端发生故障时恢复就越轻松。采用“小步快跑”的方式，持续发现问题、持续改进，构建**故障的可控**的 SRE 运作体系，方能在现网中不断降低风险。

5.3.10 小米米家故障应急保障体系实战：稳中求胜的构建之道

SRE Elite 精选原因

米家承载 8.6 亿设备、1 亿月活、日 PV 200 亿，一次失误即波及千万用户。团队以六层技术架构配合“三句话”多活容灾（层层防护、多机房、多版本快照回滚），并建立“流程前置 - 平台自动化 - 组织兜底”三维稳定性体系。研运一体化平台打通工单 - 灰度 - 监控，AI 因果 RCA 和 SRERobot 实现秒级定位与自愈。5 - 30 分钟分级应急及告警小助手，自动拉群、匹配预案、闭环复盘。

2022 跨机房专线中断与 2024 机房火灾两次实战，通过 79 项改进验证体系成效，极具参考意义，体现“稳中求胜、持续迭代”的 SRE 文化。

（一）背景及设计原则

米家应用是小米推出的一款智能家居管理平台。用户能够借助该平台，对米家以及第三方生态链中的各类智能设备进行统一连接、控制，并可自定义联动这些设备。除了实现对智能设备的统一管理之外，米家应用还具备支持场景联动以及自动化规则的功能。当前，米家应用在全球范围内处于领先地位，是全球领先的智能家居管理平台。

核心指标

- 设备接入量：8.61 亿，年均增长率 30%
- 月活跃用户：1 亿

- 累计开发者：1.3 万
- 六大区域，六大核心机房，涵盖 中国、亚太、欧洲、美洲等

以上数据既让人惊喜，又让人惊吓，因为每一次我们米家的 SRE 工程师去变更的时候，我们的一个失误都有可能造成千万的用户和设备不可用，这是一个极大的风险。

一、米家产品架构体系



(一) 设备层

覆盖“人”（手机、可穿戴）、“家”（家电、智能终端）与“车”（SU7 ULTRA 等车载终端），构成完整的人—车—家感知网络。

(二) 系统层

由座舱子系统、IoT 子系统和 OT 系统组成，分别支撑车载、家居/穿戴以及传统运营技术场景的核心能力。

(三) 接入层

端侧通过域名+负载均衡引流，依托车接入网关、IoT 网关与 OT 网关完成协议转换，实现统一接入和初步隔离。

(四) 业务层

集中处理设备绑定、事件上报、控制指令和场景编排，支持 APP 中添加/查看/管理设备，以及开发者的自定义联动逻辑。

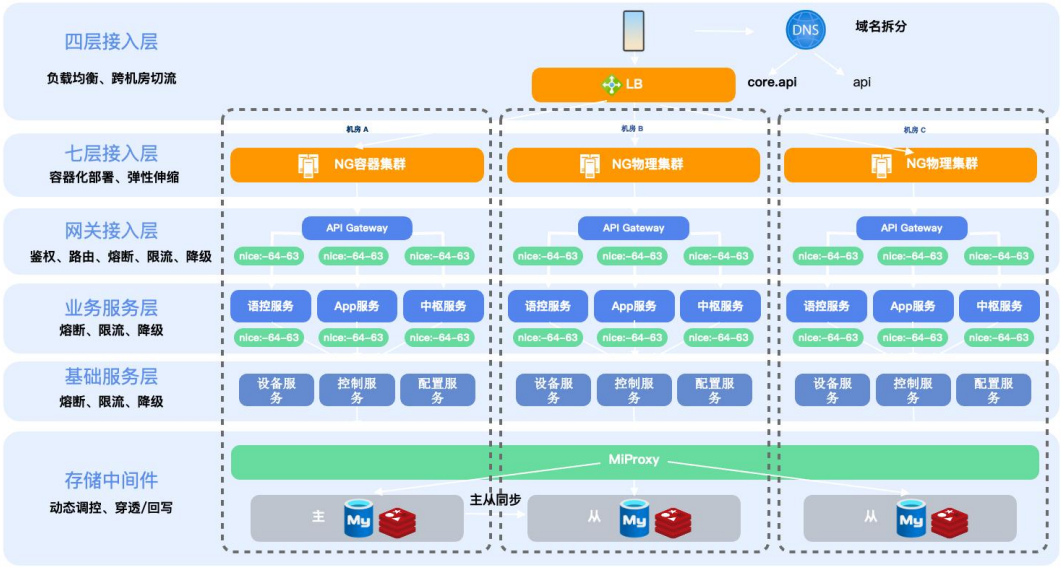
(五) 平台服务层

以“中台”形式抽象通用功能，提供设备管理、设备影子、设备控制，设备联动编排、SPEC 标准库及 SDK 标准接口，保证数据与业务协议一致。

(六) 基础设施层

数据层负责采集、清洗、分析、运营与配置；存储层采用 OLTP 架构、对象存储与分布式存储（MySQL/Redis/Pegasus），保障高可用与数据一致。

二、米家全链路服务治理多机房容灾架构



讲完了产品层的的架构体系，我们分享一下米家的容灾体系，可以归纳成三句话

第一句话：层层递进、层层保护

在 四层/七层接入层、网关层、业务服务层 和 基础服务层 均配置限流、熔断、降级与故障转移机制；

任一组件异常时，可快速隔离并启动降级或切流，保障整体链路稳定。

第二句话：多机房多活

至少在 机房 A/B/C（中国区示例为三机房）进行一致化部署；

结合 DNS 拆分 与 四层 LB，实现跨机房流量智能分发与自动切换。

第三句话：快照与事务回滚，保证数据一致性

在存储中间件层引入 MiProxy，动态路由至主 / 从 MySQL + Redis；

所有线上配置与数据操作均支持快照和回滚事务，确保一致性与完整性。

三、SRE 挑战

在如此复杂的情况下，米家的 SRE 团队面临压力过载、交付平衡双重困境

一、压力过载（Quality Pressure Overload）

- 业务规模大:核心域名 QPS 峰值达 200 万，日 PV 200 亿
- 业务增长快：年均业务增速约 30%，还在持续增长中
- 技术架构复杂：IoT 产品 SKU 数千，设备品类与通信协议多样，用户可自由组合冰箱、电视、洗衣机、空调、灯具打造复杂的个性化场景
- 客户舆情压力大：米家 App 影响用户生活，故障后大量投诉，5－6 分钟内可冲上微博热搜

二、交付平衡困境（Delivery vs. Stability）

技术层面挑战

- 故障定位与快速恢复：小流量 SKU 故障难以监测，如特定某款型号的门锁出故障，Root Cause Analysis 及快速恢复压力大
- 设备长尾效应：数千型号兼容性难题，证书更新、固件兼容需长期维护

流程层面挑战

- 测试覆盖不足：内部 IoT 测试平台难以实现对海量边界场景的 100% 自动化验证
- 变更管理压力：为了保持竞争力，需不断迭代新设备与功能，变更节奏与上线回滚能力需与时间赛跑

综上所述，米家 SRE 团队在应对日益攀升的核心域名 QPS、海量 IoT 设备与用户舆情压力的同时，又要保持快速迭代与严格的变更管控，不仅要在技术层面解决小流量故障定位与设备兼容性地狱，还要在流程层面填补测试与回滚的盲区。因此，小米的 SRE 团队从三个个层面的体系进行了建设。

四、稳定性体系设计原则

三层体系建设



一、流程层：抓“两头”保障可靠性

- 研发入口：
  - 技术方案评审、稳定性准入机制、架构韧性评估，从一开始就嵌入质量保障；
- 运维尾端：
  - 上线前进行风险演练与准入审核
  - 上线中实时监控、限流降级与故障转移
  - 上线后快速响应、根因定位与复盘改进

二、工具层：让流程“跑到平台上”

- 可观测性标准化：统一指标、日志与 Trace 采集格式，打通全链路视图；
- 自动化应急响应：基于规则引擎自动触发告警、自愈脚本，并将执行结果反馈回规则中心；

- 变更安全防控：灰度发布、限流熔断与一键回滚，减少人工操作盲区；
- 核心目标：以一条可复用的工具链承载流程，提升协作效率，杜绝人为失误。

三、组织层：超预案场景的“兜底”

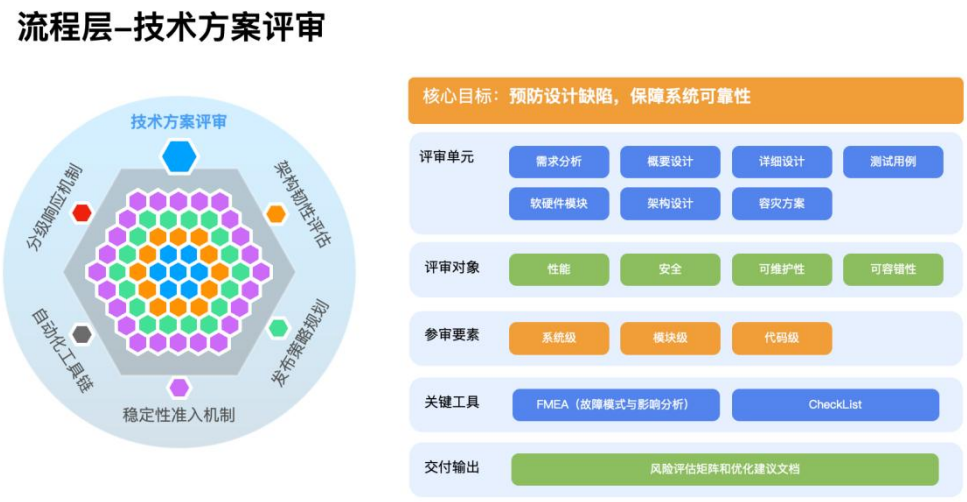
- **AI 知识库联动：**故障经验、SOP、监控规则等在知识库中沉淀，供应急时快速检索；
- **作战室机制：**事故或演练触发后，跨团队迅速集结，统一指挥、协同处置；
- **使命：**在流程与工具之外，用人力与组织协同保障未知场景的快速决策与落地。

正是在这样的双重困境下，团队构建了“前置防护－平台化工具－分级组织”三层运维体系，下一步我们将详细介绍这些策略如何帮助米家在亿级设备与千万用户规模下，实现从预防到响应、从发布到复盘的全链路高可用保障。

（二）体系设计详情

上面分享了我们总体的设计原则和分层逻辑，往下就针对每一层进行展开。

流程层 · 技术方案评审



核心目标：预防设计缺陷，保障系统可靠性

我们贯彻“DevOps 前置介入、尾端闭环保障”的理念，具体做法如下：

前置介入 · 需求与架构评审

- **早期沟通：**从软件研发生命周期初期，即与产品及开发团队共同研讨产品需求与设计，及早发现设计缺陷与可靠性风险。
- **评审内容：**
  - **需求分析：**功能边界与使用场景是否完备？

- **概要/详细设计**：核心模块与软硬件规划是否合理？
- **高可用考量**：是否内置限流、熔断、故障转移等机制？
- **测试策略**：测试用例覆盖度与自动化程度是否充分？

分层深入·系统→模块→代码

- **系统级评估**：对整体架构进行稳定性与容灾方案审核；
- **模块级评估**：验证子系统接口、性能与运维可持续性；
- **代码级审查**：细查关键依赖与安全漏洞（如发现 FastJSON 旧版本漏洞即时要求升级）。

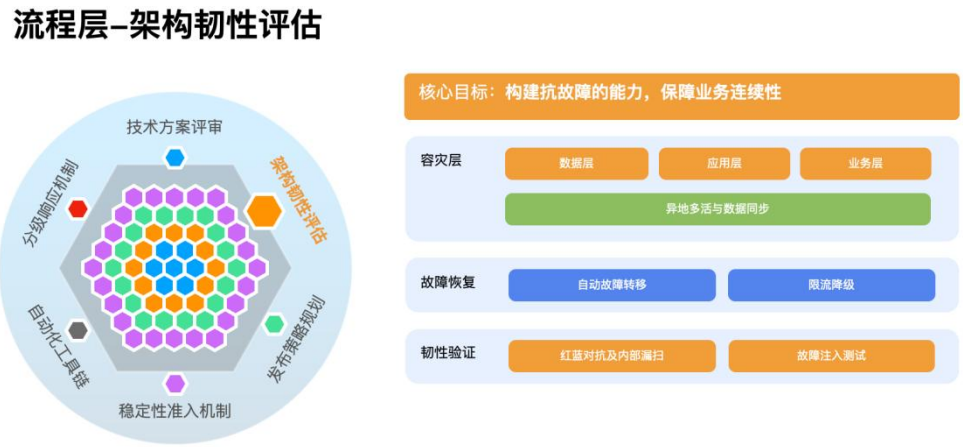
聚焦四大评审维度

- **性能**：资源利用率、最优算力申请、峰值 QPS 承载能力；
- **安全**：漏洞隐患、权限隔离与数据保护；
- **可维护性**：运维自动化、配置管理与监控可视化；
- **可容错性**：熔断策略、回滚流程与多机房多活支持。

评审工具与交付物

- FMEA（**故障模式与影响分析**） 与 Checklist 表单：
  - 将两者合并，形成**风险评估矩阵**与**优化建议文档**；
  - 输出给架构师与团队 Leader，确保每轮评审都有“可落地”的改进方案。

流程层 · 架构韧性评估



核心目标：构建抗故障的能力，保障业务的连续性

容灾层面

- **多活部署：**验证数据层、应用层和业务层是否在异地多活机房中并行运行；
- **数据一致性：**检查跨机房的数据备份与回滚机制，确保出现故障后可快速回溯到安全状态；

故障恢复能力

- **自主健康检查：**每条业务链路都要具备实时自检能力，能及时发现异常；

- **自动故障转移**：在发现服务不可用时，能够无缝切换到备用实例或机房；

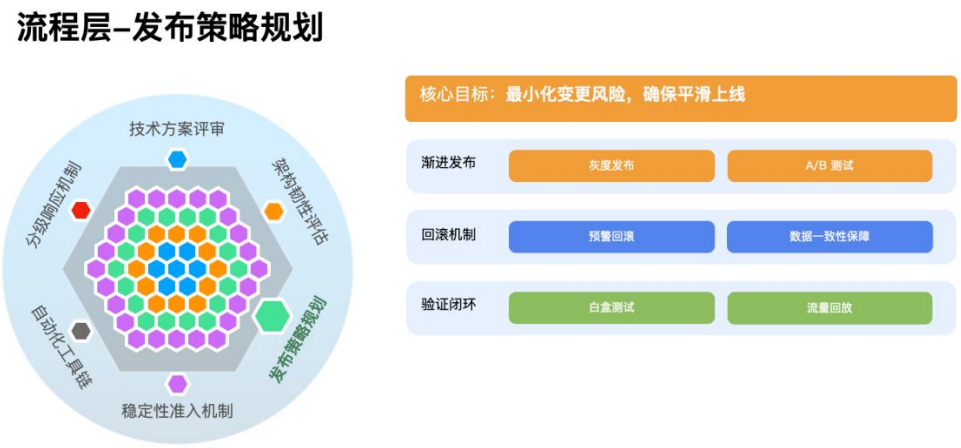
- **限流降级**：遇到大规模故障或流量洪峰，立即触发限流与降级策略，优先保留核心功能；

韧性验证

- **故障注入与混沌演练**：通过自动化脚本或混沌工程，定期模拟网络抖动、节点宕机等场景；

- **红蓝对抗与安全攻防**：邀请专业团队渗透测试，识别潜在漏洞与配置缺陷；

流程层 · 发布策略规划



在六大能力环中，“发布策略规划”是米家 SRE 在变更中投入最多精力的环节。其**核心目标是：最小化变更风险，确保平滑上线**。我们围绕这一目标，精炼出“三大法宝”：

渐进式发布

- 按用户灰度、按区域灰度逐步放量；
- 启用 A/B 测试与蓝绿发布，验收新功能后再全量下发；
- **目的**：将故障或回归风险限制在小范围内，避免大面积故障。

智能回滚机制

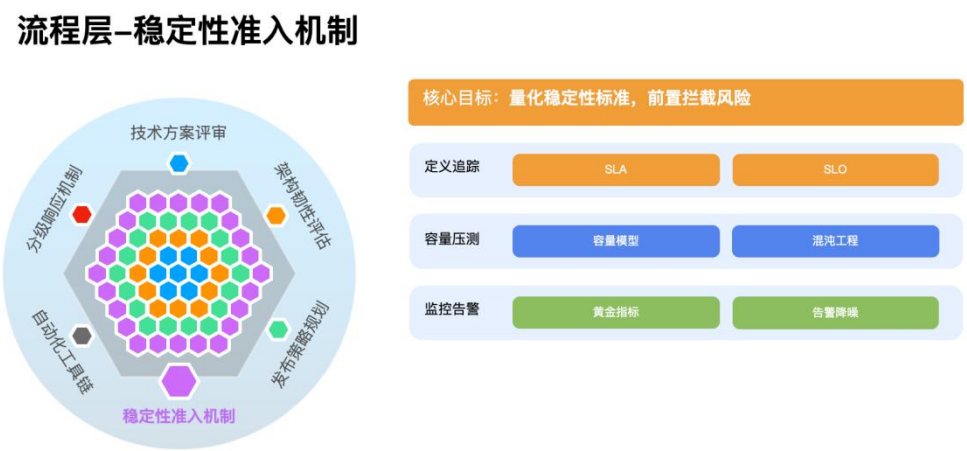
- 每个关键服务上线前绑定 SLO，发布过程监控 SLO 指标；
- 一旦发现异常（如错误率或延迟超阈），立即触发预警并自动回滚；
- 配置和数据变更均采用快照、备份与事务性回滚，确保回退后数据一致。

验证闭环

- 内部 BVT 自动化框架覆盖单元测试和接口测试；
- 构建 POC “影子环境”——线上全量流量回放与高危动作沙箱，验证隔离与权限策略；

◦ **收敛成果：**每次发布都在与真实场景高度一致的环境中演练，缺陷提前暴露。

流程层 · 稳定性准入机制



核心目标：量化稳定性标准前置拦截风险

稳定性准入三大关键要素：

定义与追踪（SLA & SLO）

◦ 对外 SLA 设为 **99.99%**，内部通过 SMDB 系统将 500+ 微服务分为 P0/P1/P2 等级；

- **P0 生死告警**：重点关注可用性、容量承载及抖动频次——即使微量抖动也要触发告警。

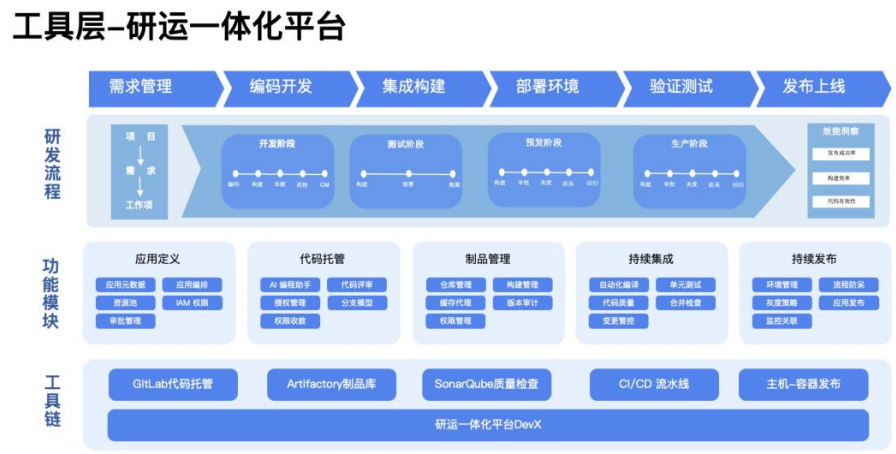
容量评估与优化

- **容量模型**：结合年均 30% 增长率，构建资源—业务增长的线性回归模型；
- **弹性伸缩**：利用容器化技术按月监控 CPU、内存、QPS、IO 等指标，动态调整各可用区（AZ）配比，达到资源与成本的最优平衡；
- **混沌工程**：在准入阶段注入故障，验证系统极限承载与自动恢复能力。

监控与告警

- **黄金指标**：精炼少量关键指标进行全链路监控，实现链路级分析与快速根因定位；
- **告警降噪**：基于异常检测引擎过滤无效告警，并按时间窗口 / Endpoint 聚合，防止告警风暴、提升敏感度。

工具层 · 研运一体化平台



端到端研发流程

从**需求管理→编码开发→集成构建→部署环境→验证测试→发布上线**，全链路覆盖开发、测试、预发布与生产四个阶段，并配备**效能洞察**（发布成功率、构建效率、代码有效性）指标。

核心功能模块

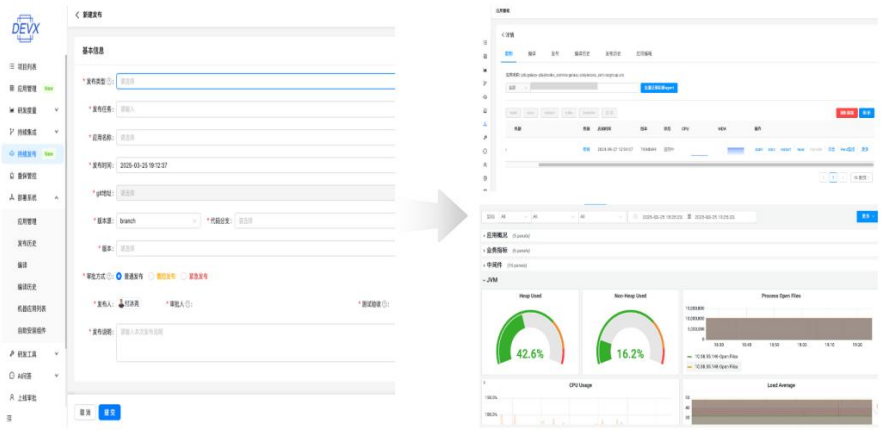
- **应用定义：**应用元数据、资源池分配、审批管理、应用编排与 IAM 权限
- **代码托管：**GitLab 托管、AI 编程助手、分支模型、代码评审与授权/权限收敛
- **制品管理：**二进制仓库、缓存代理、版本审计、构建管理与权限控制

- **持续集成：**自动化编译、单元测试、SonarQube 质量检查、合并检查、变更管控
- **持续发布：**多环境管理、灰度策略、流程防呆、监控关联与一键应用发布

下游工具链

底层集成 GitLab、Artifactory、SonarQube、CI/CD 流水线，以及主机/容器发布能力，共同支撑 平台的研运一体化。

工具层-研运一体化平台



以从“发布工单”到“实时监控”的一体化闭环为例子

发布前配置（左图）

- **侧边导航**：项目列表→应用管理→持续发布；
- **填写“新建发布”表单**：
  - **需求工单**：关联功能需求的工单编号与测试报告（接口 / 集成 / 压力测试）；
  - **代码合入**：选择 GitLab “release” 分支、分支授权、SonarQube 静态扫描通过；
  - **发布策略**：首次上线需从灰度模板中新建策略，老版本则直接提工单；
  - **多级审批**：依次由功能负责人（Leader）、SRE（稳定性评审）、QA（用例准备）三方在线审批。

灰度上线与实例管理（右图上半部）

- **一键触发**：审批完成后自动构建并按策略推送第一个灰度实例；
- **实例面板**：可对容器 / 虚拟机实例执行 start/stop/restart/nuke/transfer 操作，查看版本、状态、CPU & MEM 指标；

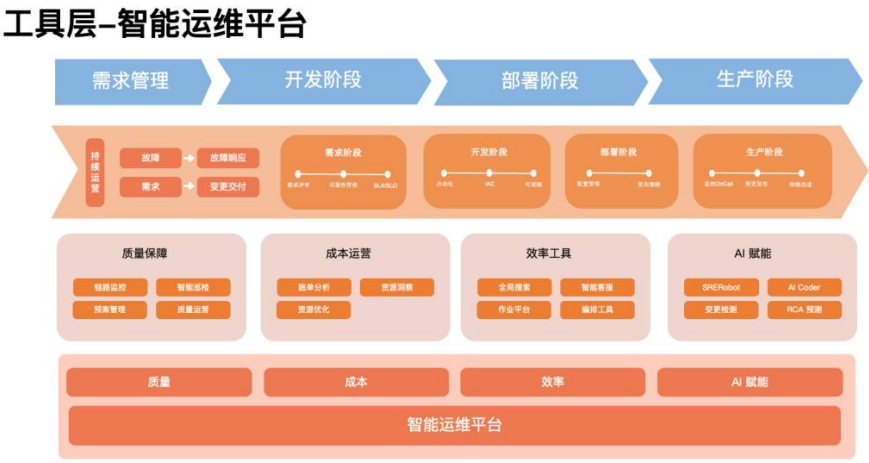
全景监控与验收（右图下半部）

- **多维监控面板**：Heap Used、Non - Heap Used、Open Files、CPU Usage、Load Average 等黄金指标可视化；

- **观察窗口**：在 30 分钟无异常后，研发工程师与 QA 共同确认回归通过，继续推进下一个灰度环或全量上线；

- **闭环反馈**：上线结果自动归档至效能洞察模块，用于发布成功率与流程优化。

通过左侧的“发布工单→多级审批”流程与右侧的“灰度实例→实时监控”视图，研发一体化平台将发布操作与运维监控无缝衔接，实现了从需求到交付、从审批到验收的端到端可控发布场景。



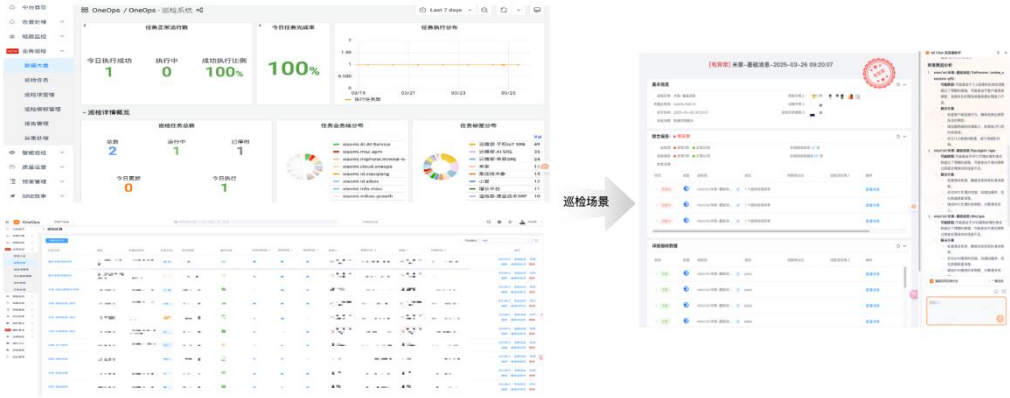
小米的智能运维平台，按照 SRE 角色场景设计规划了四个板块，质量、成本、效率、AI 赋能，为什么要这样设计呢？小米融合云平台具备完备的 CMDB、LVS、主机&容器管理平台、监控平台、预算成本平台等等，而 SRE 的工作职责主要面向质量、成本、效

率，我们按照 SRE 高频的工作场景规划了相应的平台能力和功能，下面进行展开

- **质量保障**
  - 链路监控、预案管理、智能巡检、质量运营
  - 场景：故障告警触发后，运维工程师可在“链路监控”面板一键查看端到端调用，借助“预案管理”快速执行标准化故障脚本。
- **成本运营**
  - 账单分析、资源洞察、资源优化
  - 场景：每月 SRE 团队对比增长率与云资源账单报告，通过“资源洞察”界面拉取各 AZ 用量，驱动弹性伸缩与容器编排，动态调优资源配比。
- **效率工具**
  - 全局搜索、作业平台、智能客服、编排工具
  - 场景：当多个团队协同排查问题时，可通过“全局搜索”快速定位服务元数据，利用“作业平台”批量下发运维脚本，无需频繁切换终端。
- **AI 赋能**
  - SRERobot、AI Coder、变更检测、RCA 预测

- 场景：SRERobot 在夜间自动分析最近的告警与日志，生成根因预测报告；AI Coder 辅助生成故障处理脚本，节省手动撰写时间。

工具层-智能运维平台-巡检



场景：日常业务巡检的端到端闭环

全局健康看板（左图）

每天早晨，运维工程师打开“业务巡检”基础看板，一键扫全：

- **SaaS 服务 & 自定义业务**：展示今日执行成功数、执行中、成功率及趋势折线；
- **SLA/SLO 指标**：汇总各服务的可用性与性能得分；
- **任务分布**：饼图展示所有巡检任务在业务线与监控点的分布；

- **日/周清规则**：高危异常（日清）、低危异常（周清）均在看板上标红提醒，确保无遗漏。

自定义巡检任务管理（中图）

基于 IAM 账号体系，运维工程师可：

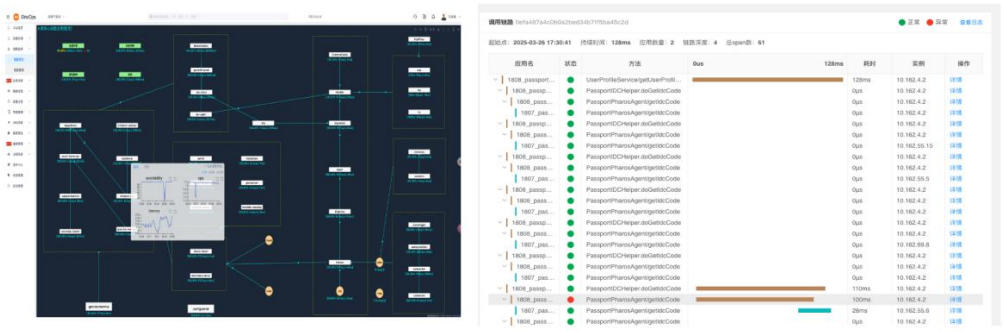
- **按组织→产品线→资源**层层映射，精准绑定每条巡检任务；
- **批量调度**：针对千余垂直业务线（如手机 App、系统工具、家电中台等），灵活配置执行频率与负责人；
- **资产关联**：账号、主机、容器与监控指标均可一键查阅，保障巡检覆盖面与可追溯性。

智能诊断与闭环（右图）

巡检任务发现异常后，系统自动生成 AI **助手报告**：

- **摘要分析**：归纳本次巡检的主要异常项（如响应延迟、错误率上升）；
- **优化建议**：结合历史 RCA 模型，提供脚本示例与操作步骤；
- **一键落地**：可直接触发工单、下发自动化修复脚本或邀请 SRE 讨论。

工具层-智能运维平台-故障分析



场景：故障分析

1. **简化核心链路图（左图）**

- **手绘精华**：并非展示米家后台数万微服务，而是由资深 SRE 与架构师携手“手绘”出的 **500+ 核心功能节点**及其调用关系；

- **清晰可读**：图中每个方框代表一条关键业务线或功能服务，虚线区域则归并为上/下游整体；

- **故障闪烁**：当某功能触发 SLO 异常，对应节点与连线自动闪烁高亮，瞬间显现“爆炸半径”——哪些上下游也将受影响，帮助快速锁定问题区域。

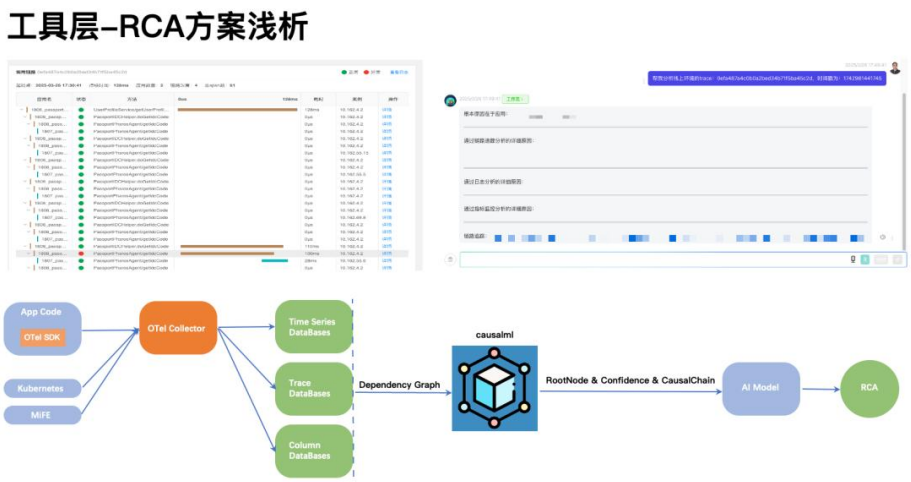
- **下转定位**：点击异常点，可以透视到 黄金指标的关键信息，再次点击可以跳转到关键链路的指标监控，快速定位根因

2. **调用链详情列表（右图）**

- **Trace 概览**：顶部展示 Trace ID、起始时间、持续时长、链路深度与总 Span 数；

- **方法级条目**：按调用顺序罗列各服务方法（应用名、接口名），以绿色/红色标识正常或异常调用，并突出高耗时行；

- **实例定位**：展示每次调用对应的实例 IP 与耗时，支持“一键查看日志”深入排查。



场景：RCA 方案， 告警触发 → 定位 → 预案匹配 → 执行闭环

告警触发 & Trace 列表（左上图）

- 当某条调用链超出 SLO 阈值，监控平台自动在“故障分析”面板中将高耗时或异常调用行高亮；

- 列表展示每个方法调用的应用名、接口、耗时、实例 IP，并支持“一键查看日志”快速锁定问题链路。

AI 小助手 & 预案匹配（右上图）

- SRE 输入：“帮我分析该 Trace 的 root cause”；
- AI 助手基于通过滑动时间窗口拟合的方式，推动给开源的因果推断工具链，基于因果推断工具的故障权重打分，推送给大模型
- 平台自动从知识库与预案库检索对应 Runbook，呈现建议给 SRE。

人工确认 & 执行

- **SRE 对 AI 建议进行确定性复核；**
- 一键触发对应自动化预案或修复脚本完成故障修复；
- 修复过程与结果同步回知识库，持续优化 RCA 模型与预案体系。

实现流程：端到端的 RCA 数据管道

以下流程对应下图左半部，从“告警触发”开始，到“Root Cause 输出”结束：

埋点采集

- 在 App 代码（OTel SDK）、Kubernetes Pod 与 MiFE 前端注入 OpenTelemetry 探针，自动收集调用链 Trace、关键性能指标与日志。

数据汇聚

- 各节点的采集数据统一发送至 OTel Collector，进行格式化与批量转发。

多路存储

- 根据数据类型分别写入：
  - **时序数据库**（指标数据）
  - **Trace 数据库**（调用链）
  - **列存数据库**（日志与上下文）

依赖图构建

- 从 Trace DB 中提取所有服务调用关系，生成全局依赖图（Dependency Graph），作为因果推断的骨架。

因果评分

- 将依赖图结点、Trace 统计权重（调用时延、错误率、调用频度）输入 CausalML 引擎；

- 挑选得分最高的节点作为 RootNote ，并将其连同置信度（Confidence）， 因果链（Causal Chain）传递给 AI 助手与 SRE，完成 RCA 闭环。

组织层 · 应急响应机制



在突发故障场景下，米家应急响应机制分为“重大”、“严重”、“一般”三层，结合**决策层级**与**升级阈值**，确保高效集结与闭环处置。

一、分级与决策（见上图“分级响应体系”）

故障级别	触发条件	决策层级	汇报 & 集结阈值
重大级别	收入受损或核心功能不可用	部门经理 + 专家组	5 分钟内全部成员集结
严重级别	关键功能异常影响业务体验	部门总监	15 分钟内完成流程闭环
一般级别	局部功能抖动或非核心服务降级	值班 OnCall 与团队决策	30 分钟内内部排查完毕

- **高 T 决策**：重大问题需高层快速拍板；
- **阈值触发**：触及“收入+核心功能”即升级至重大级；

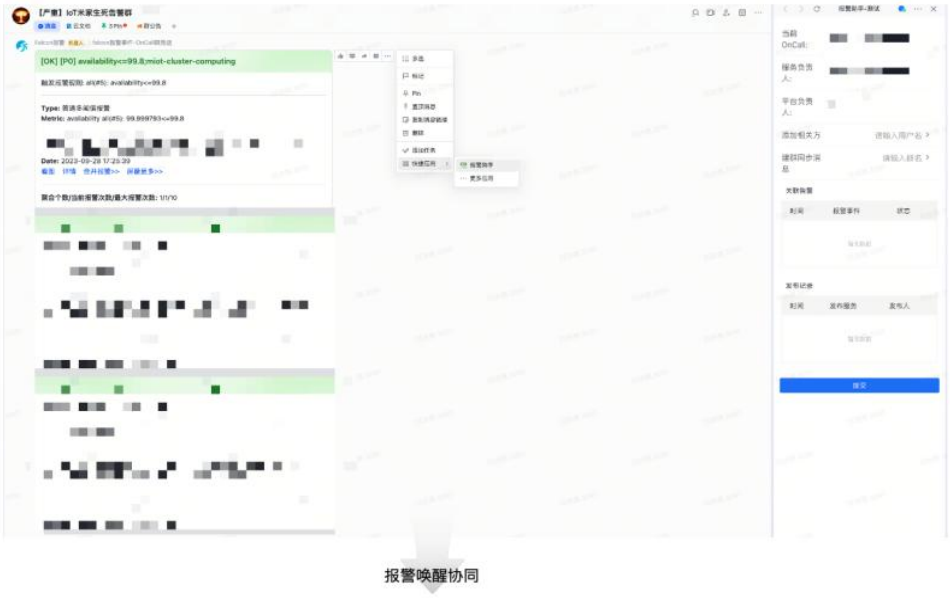
二、协同处置流程（见上图“协同流程”）

- **触发告警**：监控系统检测到异常并发出告警；
- **自动定级**：系统依据 SLA/SLO 与业务规则自动判定故障级别；
- **启动作战室**：对应级别的角色在线集结（部门经理/总监/OnCall）；
- **角色赋能**：自动派发工单与权限，赋予各方所需工具与数据访问；
- **信息同步**：共享链路拓扑、监控大盘和 AI RCA 报告；

- **预案执行**：运维、开发、QA 按 Runbook 协同处置或触发自动化脚本；
- **状态反馈**：实时更新故障进度，30 分钟—5 分钟内完成闭环，并同步复盘。

通过“**三层分级＋标准协同流程**”，米家运维组织在 5－30 分钟内完成从**告警**到**闭环复盘**的全流程响应，确保千万级用户和亿级设备规模下的系统高可用与业务连续。

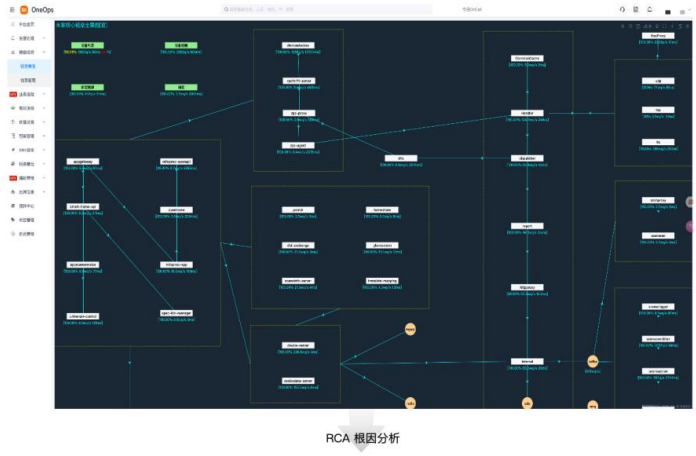
组织层·应急响应实例：P0 生死告警群 ＋ 自动化小程序



当可用性跌破 P0 阈值时，监控系统会在专门的 “IoT 米家生死告警群”（左图）中推送红色高危告警，并自动弹出右键菜单中的“告警小助手”。SRE 工程师仅需以下几步，即可完成从“告警”到“信息聚合”再到“协同处置”的全流程：

告警小助手一键建群：点击“告警小助手”后，自动抓取本周 OnCall 值班人、研发负责人 与 平台负责人，新建临时作战群，同步该业务的 变更记录 与 相关告警历史到群里头

组织层–应急响应机制–案例



信息自动聚合：所有被拉入的响应角色即可在群内一览全貌——包括调用链拓扑、告警语句与历史趋势。

组织层–应急响应机制–案例



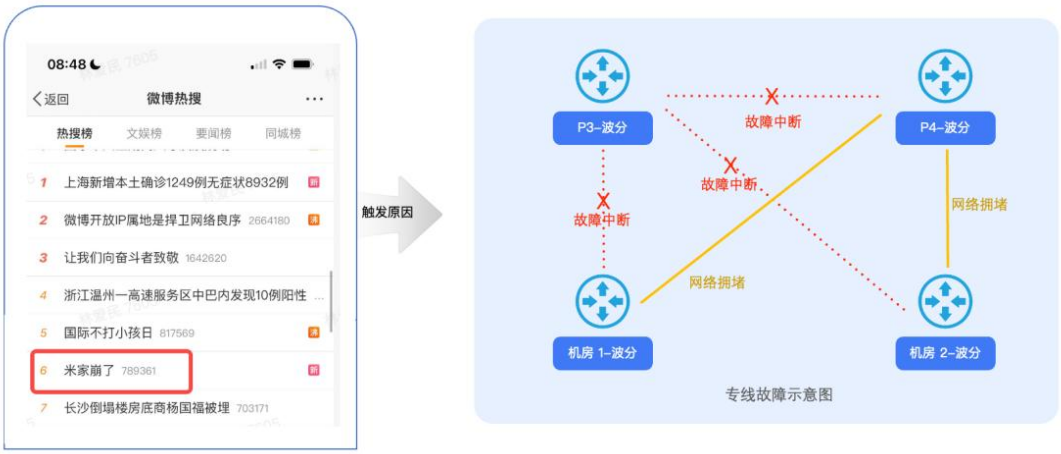
然后进入预案平台，快速切流或执行业务层的降级预案，最后预案平台会联动 质量平台，生成复盘报告入口，将协同过程的飞书

信息生成一个摘要文档，便于 SRE 和研发工程师进行信息补全最终形成改进 TODO。

总体案例

案例一：2022 年跨机房专线中断

事件回顾



第一个案例发生在 2022 年的 4 月 30 号，由于网络的故障，第一次“喜提”微博热搜。

背景回顾

当日正值业务高峰，团队在多机房架构下对 POP 点专线的拨分设备进行替换升级评估；由于对兼容性风险评级不足，工程师误判为“低风险”后直接实施，导致主线路断连而备线路因带宽不足发生严重丢包。

触发与演进

- 17:30 SRE OnCall 工程师接到 C3 和 C4 两个核心机房出现丢包的监控告警，随即开始排查。
- 随后告警持续升级，至 19:15 故障已连续恶化超过 115 分钟，常规降级与切流未能恢复服务。
- 在超过 115 分钟后，SRE 这个时候已经没有其他任何办法了，在评估全量流量转移至单一机房的风险与可能性后（单机房扛不住，所有业务会雪崩，；多机房流量互串），SRE 团队果断决定“多机房切换至单机房模式”，但由于单机房带宽仍无法承载全部业务，初次切换依旧未能成功恢复。
- 20:05 硬件服务商定位问题根源波分设备兼容性缺陷，并立即推送新的固件版本；
- 20:35 固件升级完成后，跨机房专线恢复正常，业务可用性重回稳定。

复盘流程

事实还原

撰写多篇详实文档，按时间轴细化从 17:30 到 20:35 的每一次决策与系统状态，量化收入损失、用户投诉量与流量丢失数据。

原因分析

- 触发原因：在未完成兼容性验证的前提下对波分设备进行升级。
- 根本原因：设备替换流程缺乏严格的风险评级与可行的回滚预案。
- 复盘原则：所有参与者严格自我反思，仅剖析自身决策与执行偏差，杜绝相互指责。

改进建议

- 在设备替换与升级预案中新增“兼容性压力测试”与“回滚演练”步骤。
- 优化多机房流量切换脚本，提升备线带宽分配与自动切流能力。

执行与跟踪

- 在质量平台自动生成 Postmortem 事件，列出 79 项改进任务并分派给各责任团队；
- 持续跟踪任务完成情况，确保所有改进在一年内闭环。

案例二：云厂商火灾触发数据锁定

事件回顾



在 2024 年 9 月 10 日，小米位于新加坡的 IDC 机房发生火灾。该机房四楼所承载的米家摄像机服务 OSS 对象存储服务受到影响，数据被锁定，从相关图示可见其状况。



复盘流程

案例二：云厂商火灾触发数据锁定 — 应急处置全故事

下图以时间线形式展示 9 月 10 日新加坡机房火灾导致对象存储锁定时，米家 SRE 团队的从慌乱到沉着全过程。

一、响应定位（09 - 10 10:20 – 11:22）

- 10:20 可用区 C 首次 1 分钟级波动，SRE 接警，初步判断是云厂商底层网络抖动，在联系云厂商给出问题原因。
- 11:20 同一区域再度抖动，服务提供商依旧“毫无头绪”，并没有给出第一次抖动的反馈。
- 11:22 SRE 当机立断，立即“启动作战室”预案——缩容 C 区、扩容 A/B 区全量流量

二、预案处置（09 - 10 11:22 – 20:23）

初步切换（11:22 – 12:10）在小米预案平台，一键触发 “缩容 C 区 + A/B 区灾切”

12:10 但是在操作完毕之后，业务基本恢复，但 OSS 读写异常，此时联系原厂商定位故障原因，云厂商答复：“对不起，现在对象存储没法用了，能不能 new create 一个 Bucket？”，当时 OnCall 工

程师的心“咯噔”一下，因为旧数据可能永远丢失；但目前有限还是先处理新业务的写入问题，SRE 团队立刻创建“New Bucket”任务，把所有流量都导入到新的 bucket 之中，业务写数据逻辑恢复

13:15 业务可用性恢复到 99.9%

业务恢复后，SRE 启动利用内部 HBase 源数据与备份进行详细数据恢复的方案，但是需要一天的时间，幸运的是 20:23 的时候云厂商告诉我们数据恢复了。

三、09-12 15:00 在质量平台完成第一轮复盘，输出六大模块：**还原事实，评估影响，原因剖析，问题反思，改进建议，跟踪反馈：**

可以看到，与我们 22 年那次 4 月 30 号相比，形成的鲜明的对比。这次我们就很果断有信心该执行相关的预案，有一定的进步，但是还是有问题：

第一个就是为什么 多 AZ 的 OSS bucket 会被锁定呢？为什么监控发现不了

第二个问题是为什么数据恢复需要一天时间，预案还要商量

在此，我们仍有诸多提升空间，因此我们需持续进行复盘与改进。

（三）总结及展望

回顾米家产品这几年的发展历程，业务发展飞速，质量要求越来越高，SRE 围绕质量工作虽然有了一些积累和规划，但仍处于追赶业务的亡羊补牢阶段，如何通过体系化能力的建设及 AIOps 能力弯道超车是米家全体 SRE 同仁短期工作的重中之重，我们会持续带着敬畏心和专注力，持续打磨做好米家的质量保障工作。

展望未来，首先要做好当下，持续打磨好质量体系的工具平台能力，让运维保障类工作更加高效落地；其次，展望未来，在 AI 时代，运维体系和技术的变革不仅仅是挑战也是机遇，我们要拥抱 AI 技术能力带来的效能提升的同时，认清楚技术与体系运营的边界，用 AI 赋能的同时，深挖运维保障体系建设的灰色地带，和业务一起做好技术体系和服务体系的建设工作，确保业务的稳定、健康、高质量的发展。

5.3.11 广发证券数智化重构故障管理：构建主动防御新体系

SRE Elite 精选原因：

广发证券设计了通过应急实现全生命周期稳定:运维左移通过架构韧性模型前置可靠性；变更管控用数字化平台与感知因子，达成事前防御、事中阻断、事后追溯；应急指挥借 ECC 与 ChatOps 机器人秒级集结，数字化预案保障快速恢复；大模型运维整合算法中心、知识库与智能体，智能定位根因、降噪、脚本合规；机器人把监控、变更、巡检等能力原子化嵌入 IM，释放专家价值，并探索多模态看板异常识别。

整体方案完整，落地，并具备一定的前瞻性及探索性，是证券行业的 SRE 数智化标杆案例。

（一）背景及总体解决思路

本次分享将围绕几个核心方面展开：

- 全生命周期故障管理：探讨如何在应用程序和系统的整个生命周期中融入故障管理的理念和实践。
- 运维左移：阐述如何将运维职责和活动更早地融入到软件开发和测试阶段，从而在早期发现和解决潜在问题。
- 变更管控：详细说明如何对系统变更进行严格管理和控制，以降低因变更引入故障的风险。
- 应急指挥：介绍在故障发生时如何建立高效的应急响应和处理机制，确保能够快速恢复服务。

- 智能化与场景融合：探讨如何将智能化技术（如大模型、基于算法的异常识别等）与具体的故障管理场景相结合，实现人机协同的运维模型。

一、全生命周期故障管理

证券公司交易系统的运行具有每天**“秒杀”（开盘业务量巨大）、应急时效性要求高、客户权益受损零容忍等特点。**与此同时，行情变幻莫测导致系统负载难以预测，而客户对系统可用性的要求更是到了极致，因为股票交易遵循时间优先、价格优先原则，哪怕一笔交易出现问题都可能导致客户投诉甚至产生资损。另外近年来，证券行业数字化转型、重要交易系统架构升级、容器云/多云/AI 等技术发展创新，都给证券公司运维带来较大挑战，行业内重大故障频发，公司对系统稳定性建设愈发重视。

以往，故障管理主要关注系统故障应急发现、定位、处置、恢复等能力建设。但仅靠出现故障后被动应急的方式很难满足证券行业对客户体验的高要求，广发证券信息技术部运维团队的工作不局限于传统运维部门快速处理故障和发布变更等被动性工作，而是积极推动转变角色定位，要求运维人员需要参与到软件设计开发阶段，包括从架构设计到关键功能逻辑实现的环节，协同构建具有架构韧性的系统；在推进变更自动化同时，构建变更管控解决方案，实现变更风险可感知、变更因子可观测及变更风险可防御，防范因

变更引入的故障风险；同时结合 chatops 和人工智能技术，构建在故障发生时应急处置总线，提升应急恢复能力，形成一个闭环。

通过将故障管理贯穿于整个系统生命周期，我们才能真正实现从“救火”到“防火”的转变，系统地提升系统的可靠性与韧性。

二、全生命周期管理—稳定性保障全景



为保障复杂系统稳定性，广发证券将稳定性保障前移至系统设计与测试阶段，构建了围绕 **六大核心能力** 的稳定性保障全景体系，强化软件生命周期各环节的紧密协作。在系统上线前将非功能性需求左移至研发设计阶段，上线后持续挖掘并防范线上风险，推进优化解决，形成稳定性保障闭环。

故障可恢复：关注“系统关键业务功能”的容错能力设计，使其具备限流、熔断、降级、隔离等容错能力，确保系统面对故障时能快速恢复正常运行。

性能可扩展：在业务增长、用户流量突增或面临生产性能故障时，系统能够通过灵活调整资源和服务配置来满足不同的性能需求，展现出架构的弹性。除在架构层面确保性能的可扩展性，还应制定性能扩展预案，并定期进行压力测试和演练，以确保系统在实际业务增长或高流量事件中能够平稳过渡。

变更可管控：实现”变更计划、变更申请、变更评审、变更发布、变更验证、首日保障、故障应急、复盘优化“全变更周期管理闭环，推动应用软件在制品、应用配置、数据库脚本的持续交付(CD)自动化发布，构建变更影响评估、变更风险防御、变更状态感知、异常变更定位能力解决方案，降低变更风险。

业务可监控：构建应从基础设施层向上延伸至客户体验层，涵盖基础设施监控、硬件服务器监控、平台软件监控、应用可用性监控、应用功能监控以及性能和客户体验监控。。

问题可观测：制定可观测数据埋点的技术标准，在多团队达成共识，围绕问题剖析，设计面向运维、研发、测试、供应商等多种视角的数据分析场景，通过指标、日志、链路追踪等手段快速定位问题。

部署可感知：将构成系统所有软硬件及其依赖资源数字化，实现对资源对象状态及变化的实时感知。部署可感知能力使得从整体系统资产到微观的环境配置或业务参数，都能被随时观测和掌握。。

上图中间蓝色的区域是 SRE 稳定性保障的关键过程和主要工作内容，涵盖了从容量管理、变更管理、监控管理，到架构管理、应急管理、风险治理和运行评估等多个方面。

上图下面红色部分是我们梳理的推进稳定性保障进行左移的切入点。以容量管理为例，去年底的天量行情大家一定记忆犹新，不少同业都出了稳定性事故，面对天量行情不是说加满 CPU/内存/带宽，多部署几个实例副本就能提升系统容量，我们需要评估哪些组件能 scale up，哪些能 scale out，是否有组件是瓶颈，在系统设计阶段就应该定义系统的设计容量，考虑垂直和水平扩容方案，消除容量瓶颈，在测试阶段就应该进行压力测试，绘制系统容量曲线，这样才能在稳定性保障过程中做到心中有数。

三、全生命周期故障管理—故障可恢复

故障的可恢复性指，当系统因内部或外部因素导致业务中断或面临故障风险时，系统应具备及时发现故障的能力，并能够迅速采取降级、限流、切换、重置、隔离等韧性措施以恢复业务连续性

在去年的天量行情应对过程中，我们也再次感受到良好的架构韧性在稳定性保障中发挥的重要作用，除了系统扩容，我们还需要依赖降级/限流/熔断/隔离等手段，比如行情降低刷新频率，就能在不太影响体验的情况下，降低系统和带宽压力，对不重要的交易功能做限流，保障关键交易，或者通过调整业务特性开关的方式，改

变用户与系统交互逻辑或者短路一些有问题的业务逻辑，大大降低系统压力，提升系统稳定性等，这些能力都依赖研发阶段在架构韧性上的考量，使系统内建架构韧性。



为了实现故障快速恢复，我们大概在 22 年开始投入资源做故障发现与处置，在运维平台上打造了应急管理场景工具，实现一站式应急指挥和从发现、定界、处置、定位、复盘的应急过程管理，持续完善业务监控、智能日志监控，系统全景感知提升故障发现能力，开发了故障自愈、数字化应急预案等场景工具，提升处置效率。但是出故障后被动应急的方式很难满足证券行业对客户体验的高要求。因此，我司运维团队逐渐将工作重心左移，以更积极的姿态参与到软件设计开发阶段，从架构设计到关键功能逻辑，构建具备良好可运维性的软件交付机制。

四、全生命周期故障管理



在全生命周期故障管理实践中，我们构建了一套涵盖系统内建稳定性、主动变更防御与管控分析，以及数智化重塑故障管理全过程的体系，以期在故障发生前、发生时及发生后都能具备更强的应对能力。

运维左移，内建系统稳定性： 目标是使系统从设计之初就具备更高的稳定性。通过架构评审、上线评审、非功能规范的制定与执行，以及定期的架构韧性评审站会，我们将稳定性要求前置到研发和设计阶段，避免运维在应急时陷入只会采取有限手段、过度依赖研发和厂商的被动局面。

构建主动变更防御和管控分析： 鉴于变更常常是引发故障的主要因素，我们对变更实施了全过程的严格管控。这包括变更评审、时序管控、同步对齐、技术验证等环节，确保变更的规范性和安全性。通过变更事件感知、自动化变更、变更复核以及功能验证、压力测试等手段，我们致力于提前发现和解决变更引入的潜在故障风

险。标准化的操作流程、首日保障、首笔验证、关键巡检等措施构成了变更生效后的防御机制。强大的变更防御能力结合实时的变化感知，使我们能够快速识别变更带来的异常，从而将风险扼杀在萌芽阶段。

持续提升应急管理能力应急发现与定界：利用智能化手段提升预警和告警的及时性和准确性，快速确定故障的影响范围。建立高效的应急指挥体系，基于 chatops 打造应急总线，高效建立应急连接，联动人、工具、数据、系统，持续打造应急预案和应急工具，提升处置效率。

数智化重塑故障管理全过程：有效的故障管理离不开工程化、数字化和智能化的支撑。我们持续在混沌工程、可观测性、变更管控和应急管理等领域进行建设。通过线上化、数字化和智能化的手段，我们不断地重塑故障管理的各个环节。建立高效的指挥体系，联动应急预案和应急工具，组织人员快速应急止损，控制故障扩散。

（二）体系设计详情

一、 事前防范：以运维左移提前做好风险防范

近年来 Devops 理念已然深入人心，它极力倡导产研、测试、运维各职能紧密协作，通过自动化手段，显著提升业务价值交付的效率与质量，核心聚焦于业务价值的快速流转。然而，DevOps 的推动常常以研发团队为主导，其核心关注点在于业务价值的交付速度，这使得它在满足 SRE 对系统极端稳定性、高可靠性等方面深层要求时，有点力不从心，难以完全契合 SRE 的核心使命。

从软件完整的生命周期视角审视，问题的发现和解决越在早期阶段，其所需的修复成本便越低廉。这无疑对我们 SRE 团队提出了更高要求，需要我们主动出击，**积极拥抱并推进“运维左移”（Left Shift）策略**。这意味要将诸如可靠性、可维护性、可观测性等一系列非功能性需求，前置到系统设计与研发的早期阶段，从源头嵌入稳定性基因。

然而，“运维左移”的实际落地挑战巨大，不像 Devops，它拥有成熟的文化理念、丰富的工具链以及大量的成功实践可供借鉴，“运维左移”在落地过程中，往往需要从零开始构建共识、设计新的协作机制，**并在团队内部逐步培育相应的文化氛围**，更离不开扎实的工程实践作为支撑。

接下来，我将尝试从如何构建共识、建立有效机制、推动文化变革以及夯实工程性支撑等多个层面，为大家详细剖析落地“运维左移”的具体策略和实践路径。

1、围绕系统稳定性制定“ 架构韧性模型”



运维左移的关键路径 达成共识：首先需要在各团队间就非功能性需求及标准达成明确共识。

运维左移的核心工作：架构韧性建设

广发证券运维团队基于实践经验，将系统架构韧性细致划分为 L1 到 L5 五个等级，以便更好地指导和评估系统的鲁棒性。

架构韧性 L1～L5 模型，根据这个架构韧性的模型我们定期会组织一些技术评审

- L1 基础保障级
- L2 具备逃生通道（可恢复）
- L3 可热操作止损（无单点风险）
- L4 可止损隔离（自愈 / 半自愈）
- L5 跨系统间止损免疫

通过系统梳理和明确要求，团队从以下四个关键维度，支撑和优化系统的架构韧性，并作为我们工作的牵引：

- 故障可恢复性：涵盖了冗余设计、故障隔离策略、自动化故障转移、熔断与降级机制等，确保系统能在发生故障后迅速恢复服务。
- 性能可扩展性：关注通过自动化扩容、数据库性能调优、多级缓存应用、负载均衡配置以及异步任务处理等技术手段，保障系统在面对流量增长时的稳定与高效。
- 运行可观测性：注重构建全面的监控指标体系、统一日志规范、Tracing 全链路追踪能力、以及拨测支持等，从而实时掌握系统运行状态，提升问题定位效率。
- 系统可运维性：强调实现自动化部署、建立可靠的回滚机制、完善配置管理、提供清晰的维护文档及细致的访问控制，以提高系统的日常管理与维护效率。

2、通过技术架构评审检验系统准入



运维左移的落地，需要建立工作机制，形成闭环，在系统立项和上线以及架构重大改造等环节，我们设立了技术委员会评审机制，技术委员会的委员包括了各运维条线的专家，将非功能性需求和稳定性保障要求纳入评审范围，同时这些委员也有推动技术规范的落地和培训。另外在故障应急复盘、应急演练、三线风险挖掘等运维过程中发现的风险，我们也建立了风险揭示和问题单等流程，闭环反馈给研发/测试，从而形成一个持续改进、不断强化的闭环。

3、由一线专家牵头挖掘线上系统架构韧性

运维团队要实现从被动响应到主动出击、推动“运维左移”的核心转变，其前提是一个深刻的观念革新。为了系统性地在整个组织内促进和强化这种“运维左移”的文化，我们定期组织和举办系统架构韧性评审站会，在一个开放包容的讨论氛围下，专注架构韧性探讨，定期开展持续提升运维能力。



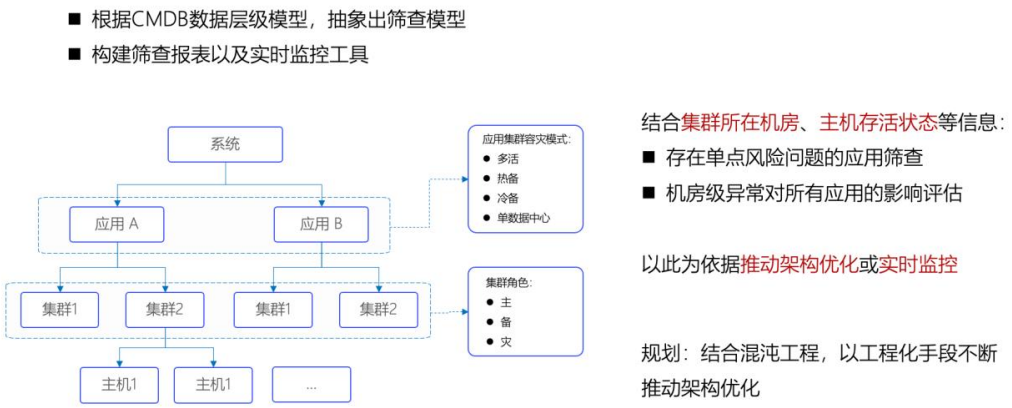
评审站会现场照片



通过这样的评审机制，我们不仅能对现有系统架构的鲁棒性、弹性进行评估，更重要的是，它建立了一个跨团队的学习交流平台。在评审会上，来自不同团队的运维专家可以互相分享经验、最佳实践以及遇到的挑战及其解决方案，从而在开放、协作的环境中

相互启发、共同成长，最终目标是整体性地提升各个团队的运维能力水平。

4、推动应用消除单点风险达成架构韧性 L3 工程实践



最后，运维左移落地，离不开工程性平台/工具的支撑，比如可观测行落地，需要建立可观测平台和最大程度方便研发埋点，以最小成本实现系统可观测，变更发布也需要交付平台的支撑，落地运维对自动化发布标准和要求。

上图是我们在做业务连续性治理时，通过工程性手段推动系统及应用架构单点风险优化的例子，结合集群所在机房以及主机存活状态等信息，筛查单点风险应用，建立机房级异常对应用影响的感知。

未来，我们也会通过混沌工程等工程手段，持续挖掘架构风险，建立对系统故障处置的信心。

二、变更管控：持续强化准入交付门槛

1、建立一下数字化的变更管控闭环

在 SRE 的实践当中啊，这个变更管控是属于非常重要的一个环节，我们将会从变更生命周期管控，变更操作执行痛点，主动变更防御，变更感知与风险分析四个方面进行阐述。



“变更”无疑是系统稳定性的头号杀手。尤其对于我们金融证券行业而言，更要如履薄冰，因为任何一次变更完成后，都可能直接面临业务流量的上限井喷高峰。在我们的金融交易系统中，变更通常按照周进行迭代发布。每次变更上线后，严格的测试验证必不可少，而上线首日更是需要我们倾注全部精力进行重点保障。

备注：监管层如证监会、央行、发改委 等部门常在节假日前后发布重磅政策如降准、IPO 调整、行业扶持等，是为了利用假期期间消化影响，节后开盘市场剧烈波动，节后首日常被称为“政策窗口期”或“情绪释放日”，股民集中交易，券商 APP、交易系统、行情数据平台等压力剧增，假期前后往往也是 IT 部门推进系统升

级、数据库调优、发布新版本的时间窗口（因业务量相对低），**但节后立刻遭遇市场交易井喷，就会放大变更引入的隐患。**

为了最大限度地降低变更风险，我们构建了一套严密的变更管控流程，涵盖了从最初的评审到最终的保障各个环节：

- **评审阶段：**这是变更的第一道关口。我们会在这里进行深入的沟通变更内容，从全局视角把控所需资源，进行严格的系统内评审，并通过必要的 CAB 评审（即变更咨询委员会评审），从源头确保变更的合理性和可行性。
- **同步阶段：**在评审通过后，至关重要的是确保相关方高度同步。这包括运维团队间的对齐，明确操作时间对齐，进行细致的重要保障评估，并对整个变更的操作时序进行评审，避免因信息差或误解导致的执行偏差。
- **实施阶段：**进入实际操作层面，我们严格遵循见单操作、操作复核规范，确保每一步变更实施都可追溯、可审计。同时，引入双岗复核机制，通过二人相互校验，进一步降低人为失误的概率。
- **验证阶段：**这是检验变更效果的关键环节。我们会进行全面的测试验证并记录问题，基于验证结果最终确认是否上线。这其中包含了必不可少的功能测试，以及在高流量场景下的压力测试，确保系统在真实负载下表现稳定。

- **重点保障阶段：**正如前面所说，变更后首日业务重点保障至关重要。为了应对潜在的突发情况，我们会进行关键时刻巡检，甚至启动保障集结机制，快速调动相关资源。更会进行首笔验证，确认核心交易流程畅通无阻。在一天交易结束后，还会进行盘后复盘，总结经验教训，为后续变更提供指导。

2、以痛点驱动设计变更管控



即使我们已经采取了多种变更管控措施，但在实践中仍面临一些棘手的痛点。

首先，**事前变更影响难以评估**是一个普遍存在的挑战。这主要体现在审批过程过度依赖个人的经验和判断，往往难以进行全面客观的影响分析；同时，审批所依据的信息可能与实际的执行操作存在脱节，导致评估结果不够准确。尤其是在跨团队和跨系统复杂交互的场景下，准确预判变更的连锁反应变得异常困难。

其次，**操作不标准引发操作风险**也是常见的症结所在。许多常规操作缺乏统一的执行范式，难以形成可沉淀、可复用的标准流程。这不仅容易导致关键操作步骤的遗漏，还可能使得技术检查和验证工作不到位，无形中增加了非预期故障的概率。

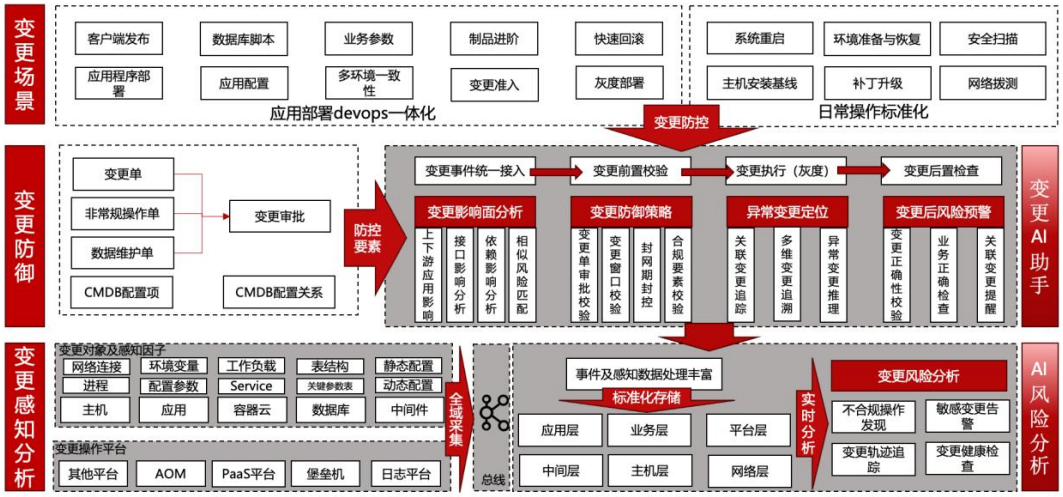
再次，对于**高危变更缺乏事中管控**也是一个亟待解决的问题。在系统处于高危时段，或者变更本身就具有高风险特性时，往往缺乏有效的实时管控手段。一旦出现异常，无法快速启动阻断机制来停止或回滚问题变更，这可能导致故障范围的快速扩大，造成更严重的损失。

此外，我们还发现**变更对象关键变化无感知**是导致不稳定性的一大诱因。许多故障的根源在于变更触发了变更对象（例如系统配置、关键参数等）的稳定性因子发生了异常变化，而我们通常对此缺乏有效的感知能力，也未能建立相应的预警和事后分析机制。

同时，**变更操作行为无感知**也带来了不少困扰。我们常常缺乏对全局变更操作事件的整体视图，难以获取跨团队之间关联的变更操作信息。这使得一些非预期的操作行为难以被及时发现并预警，同时也给事后的审计分析带来了困难。

最后，当异常或故障发生时，**快速定位引发故障的变更**往往变得非常困难。这主要是由于缺乏全局的变更信息视野，难以迅速锁定是哪一个具体的变更操作导致了问题。现有的变更流程未能有效地将变更审批、变更对象状态变化与实际操作事件关联起来，导致故障溯源效率低下。

3、将变更管控场景工程化



为了应对这一挑战，我们设计了一套全面的变更管控解决方案，其核心理念在于将日常标准化的变更操作都纳入统一的变更管控平台进行管理。

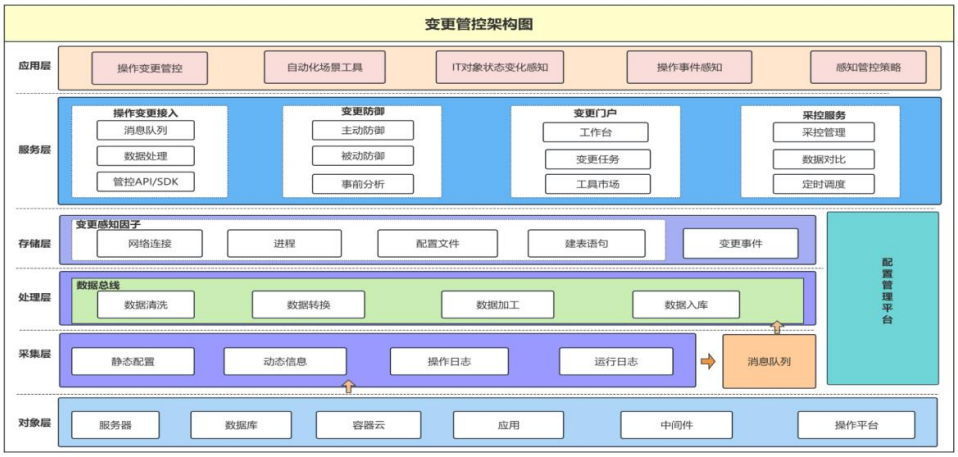
首先，通过构建强大的变更防御能力，我们对这些纳入平台的标准变更操作进行管控，减少那些容易出错的“黑屏”操作和繁杂的手工执行，确保每一步变更都清晰可追溯。

然而，并非所有变更都能完全纳入平台进行自动化管控，例如在堡垒机或某些 PaaS 平台上的操作。对于这些游离于标准化流程之外的变更，我们建立了一套强有力的兜底机制，着重于构建变更感知和分析能力。这套机制包括两个关键组成部分：

- **操作事件的全局采集：** 我们会把那些难以直接管控的操作平台，例如堡垒机以及各类 PaaS 平台产生的操作事件，全面地采集回来，汇聚到统一的数据库中。

- **变更感知因子的深度采集：** 针对主机、应用、容器云、数据库、中间件等各种核心变更对象，我们对其关键的变更感知因子进行细致的采集，例如配置变化、网络连接状态、进程启动停止等。

通过对这两类数据的全面采集，我们便为后续深入的变更感知分析奠定了坚实的基础，从而能够更好地识别、理解并应对系统中发生的各类非标准化变更。



变更管控的系统架构如上图

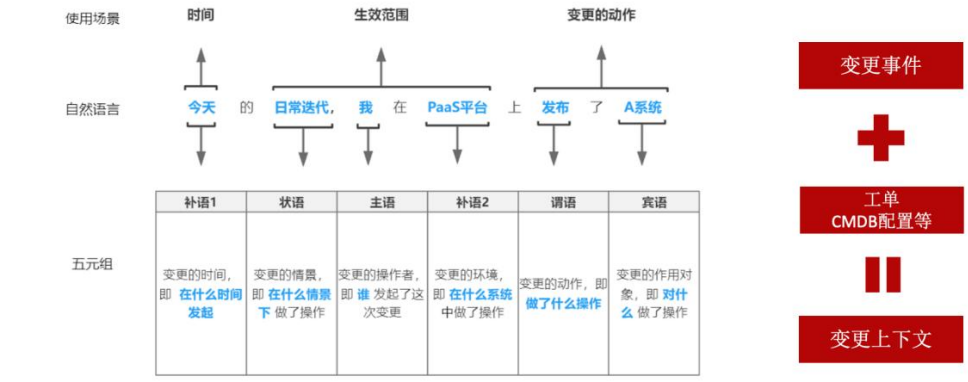
- 对象层（底部）：最底层是我们的变更对象，包括服务器、数据库、容器云、应用、中间件以及各种操作平台。这些是整个系统要管控的核心资产。

- 采集层与处理层（中部）：再往上是数据的采集和处理。采集层负责获取各种数据，例如静态配置、动态信息、操作日志和运行日志等。这些采集到的数据会通过数据总线（处理层）进行清洗、转换、加工，最终入库。

- 服务层（中部）：在数据处理之上，我们根据上层应用的需求，对服务层进行了细致的微服务划分。这包括：
  - ◆ 操作变更接入：提供统一的变更入口，确保所有变更操作都经过规范的流程。
  - ◆ 变更防御：这是非常核心的部分，提供了主动防御、被动防御和事前分析等多种机制，在变更执行前、执行中、执行后进行风险识别和拦截。
  - ◆ 变更门户：为用户提供操作工作台、变更任务管理以及工具市场等功能，是用户与系统交互的主要界面。
  - ◆ 采控服务：负责采控管理的执行、数据的对比以及定时调度任务的管理，实现对变更对象的精细化控制。
- 应用层（顶部）：最上层是应用层，包含了多种应用能力，例如操作变更管控、自动化场景工具、以及对 IT 对象状态变化、操作事件的感知能力，并基于感知到的信息进行感知管控策略的执行。

整个架构通过清晰的分层和模块化的设计，实现了变更管控的全流程覆盖，从底层的对象到上层的应用，形成一个完整的、可观测和可防御的体系结构。

4、数字化变更管控的核心模型：变更事件



注：参考蚂蚁变更管控开源项目AlterShield（https://altershield.io）

作为变更管控的基础，我们对变更事件进行了标准化建模，这借鉴了蚂蚁开源的变更管控项目 AlterShield 的理念。一个变更事件通常包含以下核心要素：

- 变更时间：何时发起变更。
- 变更操作者：由谁执行变更。
- 变更情景：在何种场景或情景下进行操作。
- 变更环境：在哪个平台或系统中执行操作。
- 变更动作：具体的修改或操作是什么。
- 变更作用对象：变更影响的是什么实体。

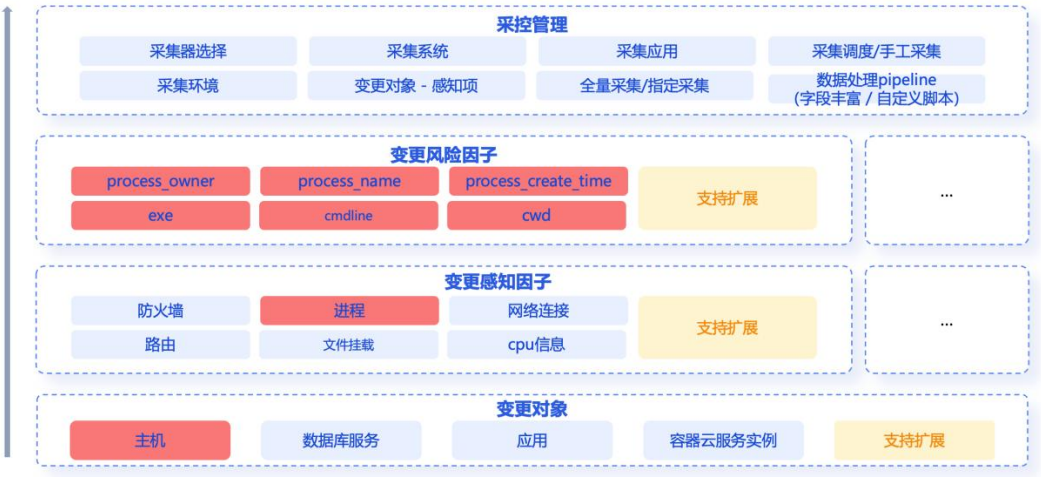
我们将这些要素结构化，形成一个五元组来描述变更事件，例如，“今天的日常迭代，我在 PaaS 平台上发布了 A 系统”这句话，就可以拆解为具体的五元组信息：

- 补语 1 (变更的时间)：今天（在什么时间发起）
- 状语 (变更的情景)：日常迭代（在什么情景下做了操作）
- 主语 (变更的操作者)：我（谁发起了这次变更）

- 补语 2 (变更的环境)：PaaS 平台（在什么系统中做了操作）
- 谓语 (变更的动作)：发布（做了什么操作）
- 宾语 (变更的作用对象)：A 系统（对什么做了操作）

然而，单纯的变更事件信息不足以支撑全面的管控。我们将结构化的变更事件信息与工单、CMDB 配置等其他相关数据相结合，构建出一个完整的“变更上下文”。后续的变更管控、主动防御措施的应用，以及对变更事件的感知和分析，都将依赖于这个丰富的变更上下文。

5、风险来自于变更因子的变化



变更管控体系中的一个核心概念，是我们构建的“变更感知因子”模型。这里所谓“变更感知因子”，是指在执行变更操作后，能够反映系统关联 IT 对象部署和运行状态变化的那些关键因素。请

注意，它不等同于我们日常监控的可观测指标，而是更侧重于捕捉系统特定状态的快照信息。

我们的目标是，将这些能够描述系统状态的“变更感知因子”与最终的可观测指标紧密关联起来。

以主机为例，我们会针对其上的多种感知因子进行精细化采集，例如：

- 防火墙规则
- 网络路由配置
- 进程运行状态信息
- 文件挂载点状态（NFS/NAS 盘挂载等）
- 网络连接状态
- 内核参数
- CPU 等硬件信息快照

当主机发生重启或经历了其他类型的变更后，这些因素的状态变化（比如动态路由的意外丢失或关键防火墙规则的缺失）都可能直接导致系统出现异常行为或潜在风险。

6、标准化变更操作工具



以下重点阐述如何通过标准化的工具实现变更流程的有效管理。

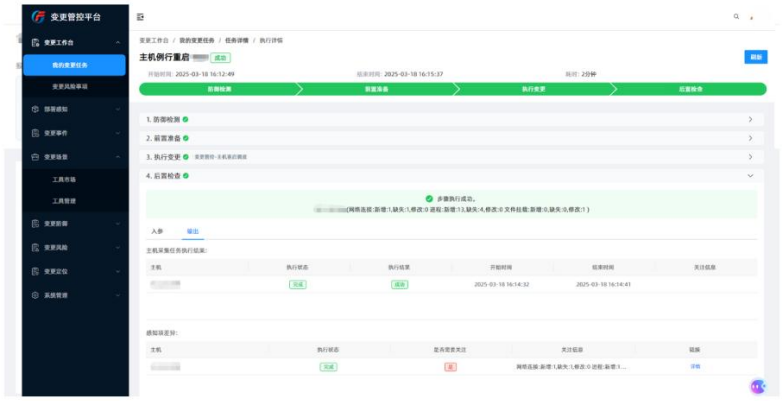
核心在于构建“能力原子化，标准化编排”的变更操作工具集，以应对日益复杂的系统变更。整个标准化变更流程被细致地划分为五个紧密衔接的关键阶段：

- **主动防御：** 这是变更流程的第一道防线，旨在前置识别和规避潜在风险。具体措施包括严格执行变更窗口检查、深度校验变更工单信息的准确性和完整性，并针对潜在的高危操作进行全面检查，确保变更在安全、可控的范围内进行。
- **前置准备：** 在正式执行变更前，进行充分的准备工作。这涵盖了向相关方进行变更通知、合理设置监控维护期以避免误报，以及采集变更对象的关键感知因子。同时，对变更前置条件进行细致确认，确保变更具备执行的基础和环境。
- **变更执行：** 此阶段是变更实施的核心，依赖于一系列标准化的变更工具和自动化流程，例如“变更工具 1”、“变更工具 2”或“自动化流程 3”等，通过这些工具高效、准确地完成既定的变更操作。

- **后置检查：** 变更执行完毕后，必须进行全面的验证。这包括及时关闭监控维护期，发送变更完成通知，并对变更后的感知因子进行采集和与变更前的状态进行比对，以确认变更是否达到了预期效果，并进行必要的技术检查。
- **变更复核：** 最后一步是进行最终的复核和确认。系统会发送复核待办通知，并由指定人员执行复核操作，确保变更的合规性和有效性得到最终确认。

通过上述标准化流程和工具的应用，我们旨在实现以下核心目标：

- 能力原子化：将复杂的变更操作分解为更小、更易于管理和组合的基本操作单元。
- 标准化编排：基于原子化的能力，通过标准化的流程引擎进行灵活、可靠的编排和组合。
- 主动防御：在变更发生前就构建起强大的风险识别和规避机制。
- 落实合规要求：严格遵循“双人复核”、“线上留痕”等合规性要求，确保变更操作的可追溯性和安全性。
- 工具共建共享：促进和鼓励各团队之间在变更工具上的协同构建和共享使用，提升整体效率和经验沉淀。



上图是主机标准化重启工具，其设计核心在于通过引入精细化的“变更感知因子”采集与比对机制，显著提升运维操作的安全性与可控性。

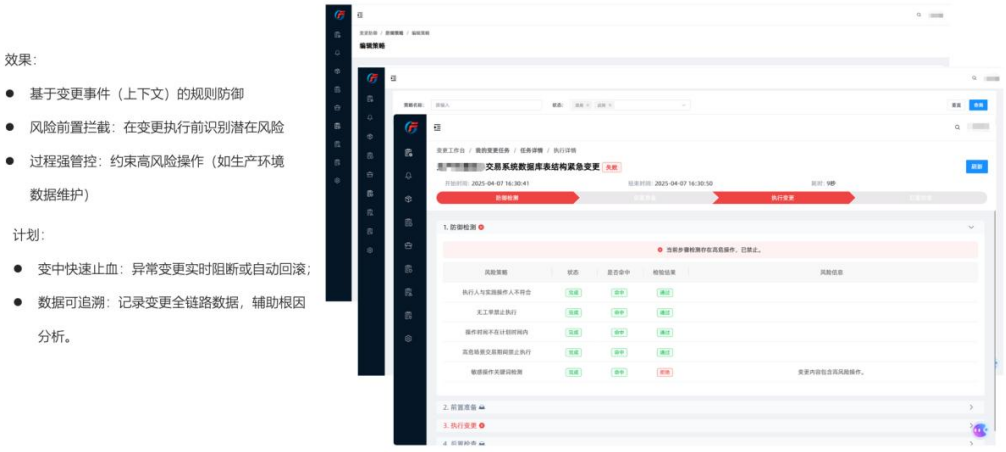
具体流程如下：

- 执行主机重启任务时，平台会在操作 前后 对目标主机的运行状态进行全面的“变更感知因子”自动化采集。
- 采集完成后，系统会智能生成一份详细的 比对报告。
- 这份报告能够清晰地展现主机在重启操作前后所发生的所有细微变化，使得操作人员能够一目了然地掌握系统的变更详情。
- 通过这种方式，我们成功地构建了一道坚实的防线，能够有效地 防范 那些潜在的、由于主机重启本身或相关操作可能引入的各种 风险。

本质上，这个工具是将复杂且容易出错的手动比对过程自动化、标准化，赋予操作人员更强的洞察力，从而规避因信息盲区或

遗漏导致的故障隐患，是我们在数智化故障管理体系中构建主动防御能力的一个具体实践。

7、构建主动防御的防控能力



作为 SRE 专家，我们深知变更操作是引发系统故障的最常见原因之一，因此，如何有效地管理和防御变更风险，是保障系统稳定性的重中之重。这张幻灯片的核心思想就是构建一套基于变更事件上下文的主动防御体系，将风险拦截前置，并对高风险操作进行强力管控。

核心目标：

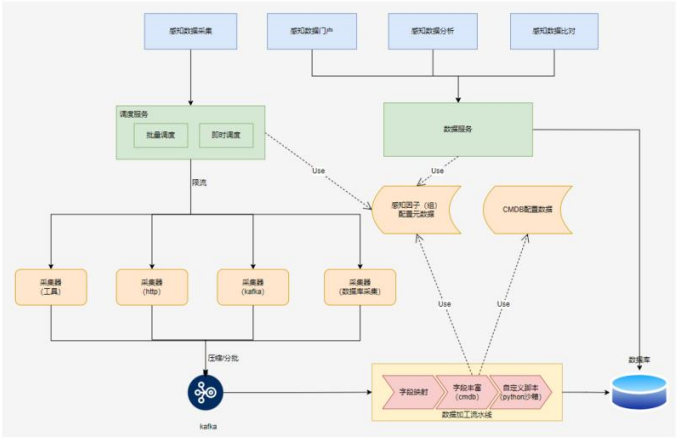
- **基于变更事件（上下文）的规则防御：** 这意味着我们的防御策略不再是僵化的规则，而是能够根据具体的变更任务、执行环境以及系统上下文，动态地评估风险并触发相应的防御措施。

- **风险前置拦截：**在变更执行前识别潜在风险,我们不再寄希望于问题出现后再去补救，而是在变更执行之前，通过自动化的风险检测机制，提前识别出可能导致故障的潜在因素，例如不符合规范的操作、与当前系统状态冲突的变更等，从而在源头阻止风险的扩散。
- **过程强管控：**约束高风险操作（如生产环境数据维护）： 特别是针对那些对生产环境有潜在巨大影响的高危操作，例如数据库的修改、关键配置的调整等，我们会施加更强的管控，确保只有经过充分评估和授权的变更才能执行，并且在执行过程中全程监控，杜绝人为失误。

除此之外，这套主动防御体系还能带来以下实际效果：

- **变中快速止血：**异常变更实时阻断或自动回滚： 在变更执行过程中，如果系统检测到任何异常情况，如服务性能急剧下降、错误率飙升等，防御体系能够立刻触发停止变更进程（止血），或者自动将系统回滚到变更前的稳定状态，最大限度地减少故障对业务的影响范围和时间。
- **数据可追溯：**记录变更全链路数据，辅助根因分析： 系统会详尽地记录每一次变更从发起、审批、执行到最终结果的全过程数据，包括执行人、执行时间、具体操作、系统状态等。这些全面的数据链对于后续进行故障的根因分析（RCA）和复盘改进至关重要，帮助我们不断优化变更流程和防御策略。

8、实现变更因子的采控





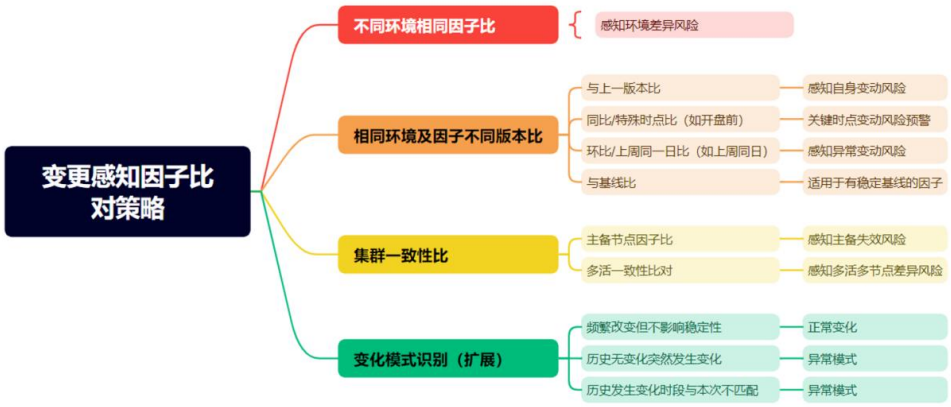
好的，我们这里聚焦的是“变更管控：变更感知采集逻辑”，核心在于构建一套能够高效、灵活采集变更数据的体系，以便我们能及时洞察系统中的细微变化，为后续的分析和决策提供坚实基础。

为了实现这一目标，我们专门开发了一套强大的采集框架：

- 通过插件的方式扩展采集能力，插件只需要关注核心采集逻辑，通过继承采集插件基类，并覆写采集方法，即可实现感知因子采集能力。
- 采集所涉及的调度、限流、压缩、分批上报等通用且复杂的逻辑，由采集框架统一处理。

- 为更好的支持感知因子的扩展和标准化，我们采取了统一命名方式，借鉴 opentelemetry 的命名规范，进行层次化的命名
- 未来，我们还将增加自动发现能力，发现并采集主机上常见的应用、中间件、数据库感知因子。

9、比不同：变更感知对比策略



变更感知的核心在于感知因子的比对策略通过构建灵活、多维度、精细化的比对方式，主动感知并应对变更可能引入的风险。

这项策略主要包含以下几个关键的比对维度：

- 不同环境相同因子比对：这是一个基础但非常重要的维度，通过比较相同配置或状态因子在不同环境（例如测试环境与生产环

境）中的差异，能够有效感知环境之间可能存在的配置差异带来的风险，及时发现那些部署或配置不一致的地方。

- 相同环境及因子不同版本比对：这是在同一个环境中进行更加细致的比较，用于感知因子自身的变动风险。我们采取了几种不同的比对方式：

- 与上一版本比对：这是最直接的方式，通过与最新的稳定版本进行对比，能够快速识别当前变更引入的变动风险。

- 同比/特殊时点比对：针对业务高峰或关键时点（例如交易系统的开盘前），通过与历史同期数据或特定时点数据进行比对，可以及时预警潜在的关键时点变动风险，确保系统在重要时刻的稳定性。

- 环比/上周同一日比对：这种周期性比对策略，例如与上周同一天的数据进行对比，能够帮助我们感知和识别异常的周期性变动模式，及时发现不符合常规规律的变化。

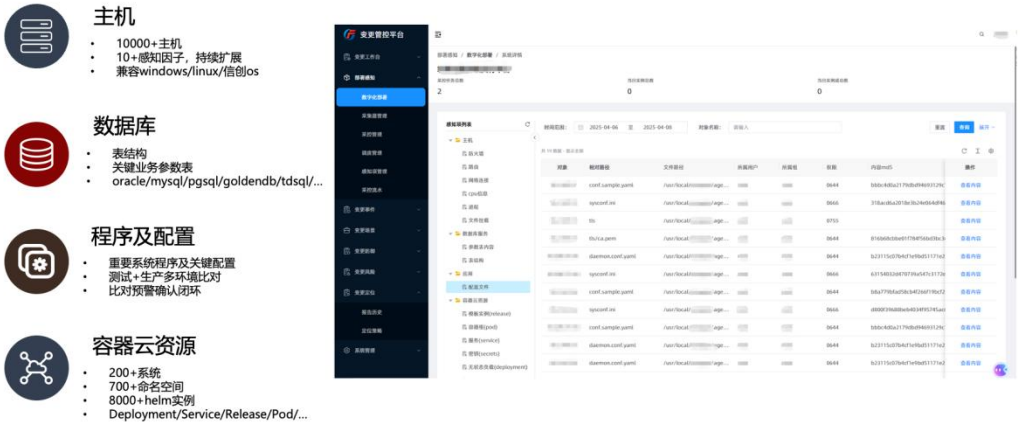
- 与基线比对：对于那些具有相对稳定基线的因子，通过与预设的稳定基线进行比对，能够有效识别偏离正常状态的变动。

- 集群一致性比对：针对我们广泛应用的集群化部署架构，确保集群内部的一致性至关重要，因此我们引入了集群一致性的比对：

- 主备节点因子比对：通过比较主备节点之间的关键因子，能够及时感知主备节点可能存在的配置不一致或状态差异，从而预警主备失效的风险。

- 多活一致性比对：在多活部署的复杂场景下，确保不同多活节点之间的数据和服务状态一致性非常关键，这项比对能够帮助我们感知多活节点之间的差异风险，保障多活架构的可靠性。
- 变化模式识别（扩展）：除了静态的比对，我们还引入了更高级的变化模式识别，通过分析历史数据，对当前的变化进行智能化判断：
  - ◆ 如果某个因子历史表现为频繁变动但从未对系统稳定性产生影响，当前的变动可能被视为“正常变化”。
  - ◆ 反之，如果一个因子历史从未发生变化，却突然发生了变动，这很可能是一个“异常模式”，需要重点关注。
  - ◆ 同样，如果当前的变化时段与历史上的变化时段不匹配，也可能预示着“异常模式”的出现。

完成这些多维度的比对后，系统会自动生成详细的比对报告，并可以根据用户的订阅或配置，主动发送给相关的技术人员或团队。



基于变更因子采集框架，我们构建了数字化变更感知，针对主机、数据库、程序及配置以及容器云资源等变更对象，实现了感知因子多环境采集比对：

- **主机感知因子采集：** 覆盖超过 1 万台 主机，持续扩展感知因子种类。全面兼容 Windows、Linux 以及 信创操作系统 等多样化环境，为跨平台统一监控奠定基础。
- **数据库关键信息采集：** 深入到数据库层面，采集包括 表结构 和一些 关键业务参数表 在内的重要信息，以便及时发现与数据库相关的潜在问题。
- **程序与配置的采集及比对：** 对重要系统的 程序 和 关键配置 进行采集与比对。**不仅支持 测试 和 生产 等多环境间的差异比对，而且比对结果会进行 闭环确认，确保配置变更的合规性与安全性。**
- **容器云资源采集与比对：** 针对容器云环境，**实现了对 200 多个系统、700 多个命名空间** 的各类资源进行全面采集与比对，确保容器化应用的资源配置健康、一致且可控。



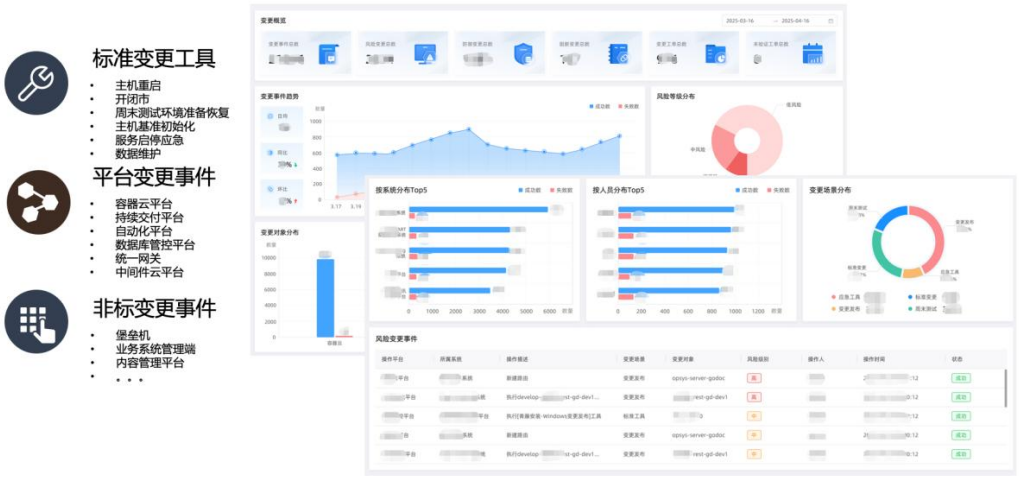
上图是数字化感知比对的案例截图：

关键配置文件比对：当发生变更操作，修改关键配置文件时，会将变动推送给负责人，由负责人二次确认，防止预期外变动。

主机感知因子比对：当主机层面发生路由、防火墙规则、进程状态等变化时，推送给负责人二次确认，防止因未持久化的变动失效导致的风险。

IP 访问关系构建：通过采集主机网络连接，构建主机间访问关系，为上层各应用场景（系统上下游关系构建、告警关联性分析等）提供数据支撑。

10、变更实践态势感知



作为数智化重塑证券公司故障管理体系的关键一环，我们针对变更事件构建了全面的态势感知能力，这对于主动预防和快速响应故障至关重要。这项能力主要通过统一采集和汇聚不同来源的变更事件来实现。

具体而言，我们对以下三类主要的变更事件进行了整合：

- 标准变更工具产生的事件：指的是变更管控平台上统一管理的标准化的变更流程和工具产生的事件，例如自动化的部署、配置更新等。这些事件通常结构化程度高，易于解析和处理。
- 变更操作平台的事件：这是指我司各类变更操作平台中记录的各种操作行为和状态变化。这类操作通常通过平台间对接即可实现采集，包括容器云平台、持续交付平台、数据库管理平台等。
- 非标准变更事件：指的通过堡垒机或者直接登录主机或者业务系统自带管理后台进行操作的变更事件，通常这类事件采集和接入的难度、成本比较高，我们将这些事件的采集优先级放在最后，

完成前面 2 类操作事件采集覆盖后逐步采集接入。通过推广标准变更工具，逐步减少这类变更方式的占比。

通过将这些多样化的、来源于不同系统的变更事件进行统一的采集和汇聚，我们得以建立起对整个变更生命周期的全面视图。这种整合能力是我们能够更深入地理解变更与潜在故障之间的关联，从而实现对变更事件态势的实时感知和分析，为构建主动防御体系奠定坚实基础。

11、场景驱动的变更管控



变更验证待办



集结通知



关键时刻重要巡检推送

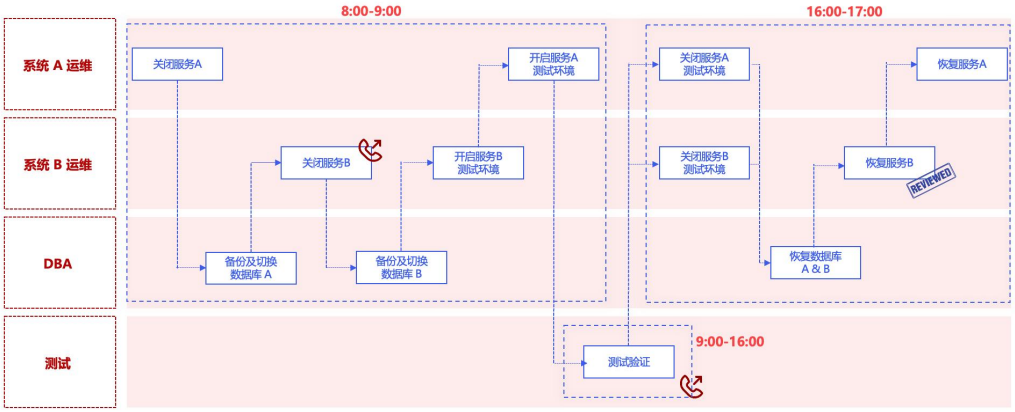
为了确保变更能够安全、高效地进行并顺利落地，我们依托运维平台打造了一系列场景化的工具，它们如同一个个前沿阵地，在变更过程的不同阶段提供坚实支撑。

- **变更验证表：** 变更实施完毕后，系统将自动编制一份详尽的变更验证任务清单，并向所有相关利益相关者进行推送。此机制确保了每项验证工作均有人专门负责，并且有明确的记录可查，从而

形成一个严密的验证循环，显著减少了因验证不足可能引发的风险。

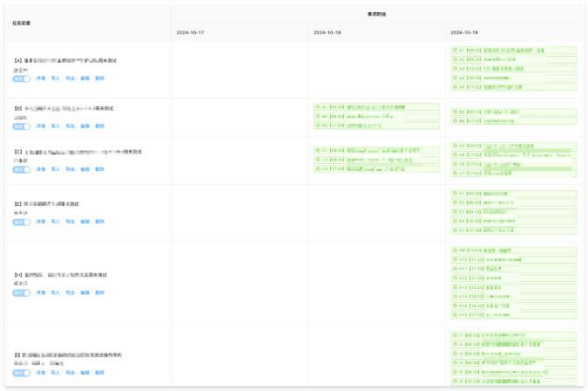
- **集结通知：** 变更部署完成后，平台会自动触发拉群动作，并将所有变更相关的干系人拉入一个群聊，发送集结通知。这确保了在变更后的关键时期，所有相关人员都能迅速集结，确保在第一时间对可能出现的问题进行响应和处置。
- **巡检推送：** 在变更完成后的关键业务时点或周期性检查点，系统会自动执行预设的巡检任务，并将巡检结果以消息推送的形式发送给运维人员。这些推送内容高度聚焦于系统和数据的关键状态，让运维团队能够实时、清晰地掌握变更后的系统运行状况。

12、大规模团队变更时序管控



券商行业的全网测试，通常是一个涉及几百人的大型变更场景，包括了运维/测试/研发/业务等多个团队的协作，我们针对大规

模变更场景，开发了时序工具，确保任务之间的时序依赖，高效推进整个变更过程，任务节点支持对接自动化平台，自动执行一些像环境准备和恢复的例行性的操作，减少操作风险。



时序配置汇总



个人任务列表



逾期升级督办

三、最后一道防线：应急指挥

1、应急指挥总体思路

即使我们在研发流程前端左移运维责任，并辅以严格的变更管理机制，系统异常的发生仍然是一个无法完全避免的现实。**因此，应急指挥体系成为我们保障系统稳定性的最后一道也是至关重要的防线。**应急指挥目前已经在公司各个条线形成了一个应急的协同文化，出现故障时不仅系统相关的运维、研发、测试能够快速参与到应急，上下游系统也能够按需订阅故障实时处置的信息。

为了最大限度地降低异常事件的影响，我们持续投入资源，不断完善应急保障体系，其构建核心围绕以下关键要素展开：

- **应急保障体系：**构建坚实可靠的框架，确保在突发状况下，能够迅速、有序地调动资源、明确职责。
- **一站式应急总线：**打造统一的平台，整合各类告警信号、监控数据和处理工具，实现信息的高效汇聚与流转，避免信息孤岛和人工传达的延迟与错误。
- **事件处理流程：**建立清晰、可执行的标准化流程，从事件发现、定级、处理、恢复到复盘，每一步都有章可循，减少混乱和不确定性。
- **机器人应急协同：**应急指挥是由一系列机器人（监控响应升级、故障应急、值班经理、智能影响分析等机器人）辅助建立一个跨团队的应急调度，确保所有的故障都能高效的运作。
- **数字化应急预案：**将预案电子化、结构化，方便快速检索、更新和执行。这些预案是团队在面对常见或复杂故障时的行动指南，能够显著提升应急响应的速度和有效性。
- **智能化应急：**将应急指挥与大模型、智能体、知识库关联，实现智能化的应急。
- **信息共享：**让不同的人能够在一个平台上共享数据，支持按需获取应急信息。

2、ECC 运营指挥中心

IT运营指挥中心（ECC）

- 主要包括故障监测、应急指挥、数据运营、重大演练等核心职能
- OnCall 人员的值守、应急作战指挥室的线下场所





IT 运营指挥中心（ECC）作为应急指挥体系的关键构成要素，同时承担着线下场所与应急核心职能，在应急指挥体系中扮演着至关重要的角色。

如左侧星型图所示，IT 运营指挥中心（ECC）是整个应急保障体系的枢纽，围绕着它，辐射出多个关键职能和协作方：

- **OnCall 值班经理 & 一线：** 作为值班体系的前沿，负责初步响应与处理。
- **故障发现：** 负责通过各类监控手段及时感知系统异常。
- **应急响应 作战室：** 应急事件发生时，快速集结相关人员进行协同处置。
- **故障定界 & 定位：** 通过分析与排查，明确故障影响范围及根本原因。
- **应急处置 决策：** 在获取故障信息后，迅速制定并执行恢复策略。

- **重大保障 集结：** 在关键时期或发生重大事件时，集中力量进行重点保障。

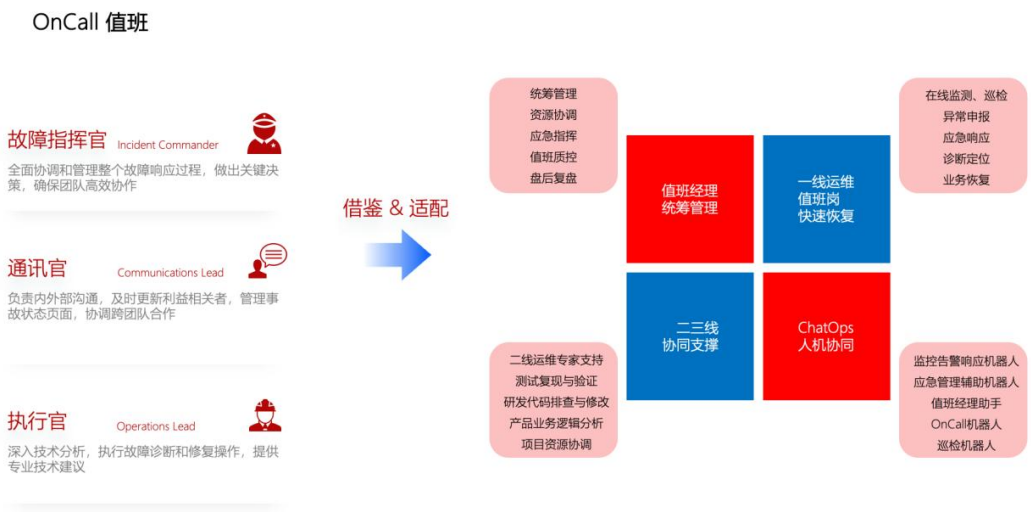
ECC 其自身具备多项核心职能，是我们在故障、数据运营、重大演练以及应急指挥中的关键抓手：

- 值班保障：每日组织一线人员值班保障。
- 故障监测： 负责对各类系统、服务进行不间断的监控。
- 应急指挥： 作为事件处置的统一指挥中枢，统筹协调。
- 数据运营： 负责收集、分析与展示各类运营数据，为决策提供支持。
- 重大演练： 组织和执行各类应急演练，提升团队的应急响应能力。

ECC 不仅仅是一个虚拟的概念，更是一个具有线下场所的实体，它是我们 OnCall 人员值守以及应急作战指挥的基地，正如右侧图片直观呈现的监控指挥中心场景。

3、一二三线协同的 onCall

我们参考 Google SRE oncall 机制，在运维保障体系中设定了分工明确的保障角色，例如值班经理，其职责涵盖统筹管理、资源协调和应急指挥，是保障高效响应的关键角色之一。



我们将整个应急保障体系划分为清晰的职责和协作层级：

- **一线运维值班岗：**这是我们快速响应的第一道防线，肩负着异常申报、通过在线监测和巡检及时发现问题，并进行应急响应、诊断定位和快速恢复业务的关键使命。
- **二三线协同支撑：**当问题需要更深入的技术分析和根治时，会激活二三线的协同体系。这包括二线运维专家的技术支援，还有测试团队对问题的复现和验证，以及研发团队进行代码排查和必要的 Bug 修复与代码修改。此外，还可能涉及到产品团队进行业务逻辑分析以及跨项目的资源协调。

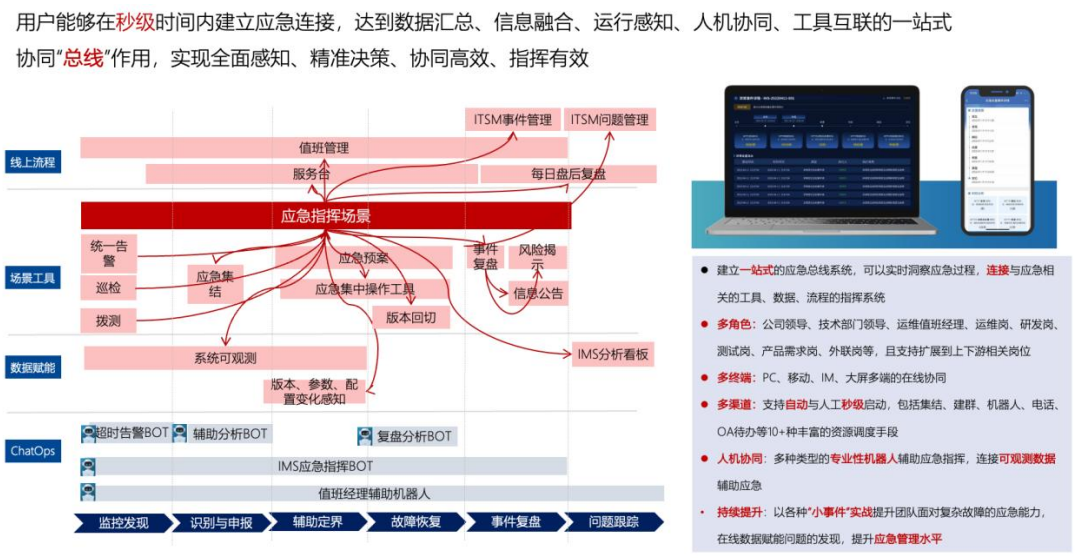
为了大幅提升应急响应的效率和自动化能力，我们在平台层面实现了 **ChatOps 人机协同 的能力**。这意味着在故障应急过程的各个关键环节，都会有智能化机器人提供强大的辅助支撑。比如：

- **监控告警响应机器人：**自动化处理告警信息并触发初步响应动作。

- **应急管理辅助机器人：**协助管理事故状态、流程推进和信息同步。
- **值班经理助手：**为值班经理提供数据统计、状态汇总和任务提醒等功能。
- **OnCall 机器人：**协助 OnCall 值班人员进行故障诊断和操作执行。
- **巡检机器人：**自动化执行例行检查任务，提前发现潜在隐患。

整个体系的设计，既有清晰的人员职责划分，又有强大自动化工具的支撑，旨在确保在面临生产环境问题时，能够实现快速准确的发现、高效 的协同以及迅速有效的恢复，最大化地保障业务的连续性和稳定性。

4、人机协同的应急的总线



应急总线的核心目标在于：打破信息孤岛，将与应急相关的各类工具、数据和流程无缝连接起来，形成一个能够实时洞察应急全过程的统一平台，确保在故障或突发事件发生时，能够以秒级的速度建立应急连接。

为了达成这一目标，我们构建的应急总线具备以下关键特性：

- **一站式接入与实时感知：** 系统汇集了分散在各处的应急工具、数据和业务流程，构建了一个统一的入口，从而能够实时、全面地感知应急事件的进展，消除信息不对称带来的沟通和协作障碍。
- **多角色广泛协同：** 系统天然支持公司领导、技术领导、运维值班经理、各级运维、研发、测试、产品需求及外联等广泛角色的在线协同，并具备向下游相关岗位扩展的能力，确保应急事件处理能够囊括所有相关方。
- **多终端无缝接入：** 通过提供 PC、移动、IM、大屏等多种终端支持，系统保证了无论值班人员或决策者身在何处，都能第一时间通过熟悉的设备接入应急流程，确保信息的秒级同步。
- **多样化秒级启动：** 无论是通过自动化规则触发，还是由人工发起，系统都能在秒级内启动应急流程，并提供十余种资源调度方式，包括自动集结人员、创建 IM 群组、调用专用机器人、进行电话通知、触发 OA 待办等，确保应急响应的快速启动和人员到位。

- **人机协同智能赋能：** 系统集成了多种专业化机器人，这些机器人能够与可观测数据深度集成，在应急指挥过程中提供辅助分析、智能判断和自动化执行能力，显著减轻人工负担，提升决策的精准性和效率。

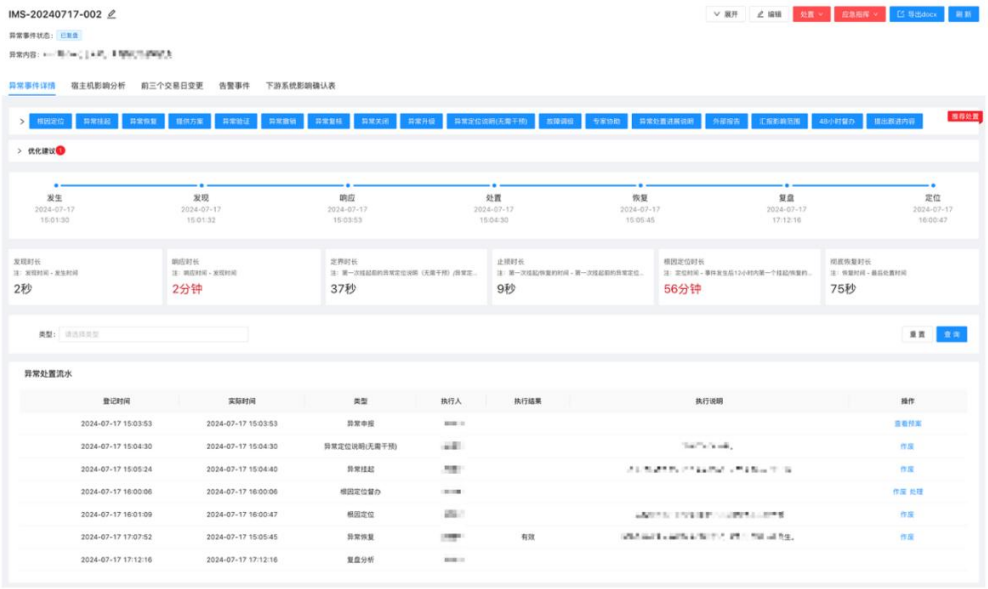
同时，我们强调通过日常的“小事件”实战演练来持续提升团队处理复杂故障的能力，并利用系统沉淀的在线数据赋能问题发现与根因分析，从而实现应急管理水平的不断优化与提升。

结合幻灯片左侧的流程图，我们可以更清晰地看到这个应急总线是如何将各个环节串联起来的：

- 以应急指挥场景为中心枢纽。
- 线上流程中的值班管理和服务台与 ITSM 流程紧密集成，保证事件处理的规范性。
- 统一告警、巡检、拨测等场景工具直接为应急指挥提供信息和执行能力。
- 系统可观测、版本、参数、配置变化感知等数据源为决策和根因分析提供关键数据支持。
- ChatOps 能力通过各类机器人辅助应急流程的各个阶段，从告警发现到复盘分析，实现基于 IM 的高效协同和自动化操作。

正是通过这样的体系构建，我们旨在打造一个能够快速响应、智能辅助、协同高效的一站式应急总线平台，为业务的稳定运行提供坚实保障。

5、实时在线感知应急过程



最终，所有的故障处理实践与方法都将汇集并落地于我们标准化的事件处理流程之中。这一流程体系的设计，旨在确保在面对任何突发状况时，我们都能遵循预设的、高效的步骤，从而最大程度地减少故障影响，保障业务的连续性与系统的稳定性。

6、数字化应急预案



为了有效地提升故障应急处置的效率，我们打造了数字化应急预案工具。

该工具围绕两个核心理念构建：

- **原子化**：将复杂的应急流程拆解为独立、可复用的“原子预案”。这种模块化的设计极大简化了预案的编排、更新以及与自动化的关联，提升了整体的灵活性和效率。
- **数字化**：通过平台化实现预案的可编排、可演练、可统计、可遍历、可自愈能力，为预案的持续优化和扩展奠定基础。数字化平台不仅能够轻松组合原子预案形成完整的应急流程，还能支持模拟演练以验证其有效性，对历史事件和执行情况进行统计分析以指导改进，方便追溯整体流程，并最终目标是实现系统异常时的自动触发和恢复。

如图所示，一个典型的数字化应急处置流程被清晰地呈现。流程按照阶段划分，从左向右对不同服务/组件的应急处置进行细化描述。

具体而言，流程自上而下推进：

- **判断阶段：**在此阶段，针对服务 A、组件 B、服务 C，首先进行快速判断是否能执行处置动作，并辅以链接和确认表来获取信息和进行决策。
- **处置阶段：**识别出可行的处置方向后，便进入具体的处置步骤。该阶段包含处置动作编排，然后执行针对不同服务/组件的具体预案：
  - 服务 A 执行应急卡片 A01 和应急卡片 A02。
  - 组件 B 执行自动化原子预案。
  - 服务 C 执行主机异常策略。
- 验证阶段：处置完成后，流程进入整体的验证阶段，以确认修复效果。

通过将应急预案进行数字化和原子化，我们旨在构建一个更高效、更具操作性且可持续改进的应急响应体系。



针对突发事件的应急响应，我们的核心策略是构建可高效执行的数字化应急预案，并强调其“原子化”与“数字化”特性。

原子化的理念在于将复杂的应急处置流程拆解成具备复用能力的最小单元。这样一来，无论是针对特定故障场景进行预案编排、后续的更新迭代，还是与自动化系统进行无缝对接，都能变得异常灵活和便捷。

数字化则赋予了预案一系列关键能力：

- **可编排：** 能够根据具体事件、受影响的服务或组件，像是服务 A、组件 B、服务 C 等，灵活地组合和调整“处置动作编排”，比如调用“应急卡片 A01”“自动化原子预案”或“主机异常策略”。*
- **可演练：** 允许我们在非生产环境中模拟执行预案流程，验证其有效性，从而在真正面临故障时游刃有余。

- **可统计：** 能够记录应急处理的全过程数据，包括每个步骤的耗时、执行结果等，为后续的复盘分析和持续优化提供数据支撑。
- **可遍历：** 整个应急处置流程，从“判断”“处置”到“验证”三个阶段，以及在每个阶段中针对不同服务的详细步骤和验证项（如“黄金指标”“日志输出”“业务验证”“下游确认”），都可以清晰地被追踪和理解，毫无盲区。
- **可自愈：** 通过与自动化能力的结合，理想状态下，数字化预案能够触发一系列自动化操作，帮助系统在一定程度上实现自我修复。
- **可持续扩展：** 随着业务和系统的演进，应急预案可以轻松地进行调整和改进，以适应新的需求和场景。

如幻灯片所示，数字化应急预案的流程清晰地展示了从判断故障是否符合处置条件（通过“链接”、“确认表”等原子预案或工具快速辅助）到执行处置动作，最终再对处置效果进行全面“验证”的过程。其核心价值在于，在故障发生时，我们可以基于预先编排好的、由各类原子化能力（如“链接”、“确认表”、“应用异常策略”、“主机异常策略”等）组成的数字化流程，一键启动应急响应，快速完成故障定界、处置，并最终验证处置的有效性，大幅提升应急效率和准确性。

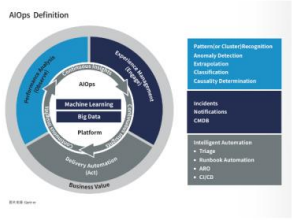
四、智能化与场景融合

智能化如何与运维场景深度融合，我们将围绕以下三个核心点展开：

- **广发证券对智能运维的理解：**我们将深入探讨广发证券在智能运维领域的认知框架，包括企业对智能运维的定义、期望达成的目标、目前所处的阶段以及对智能运维价值的深刻洞察。
- **运维平台智能化升级思路：**详细剖析我们将如何对现有运维平台进行智能化改造与升级，这涉及架构层面的优化、引入智能化组件的设计思路以及技术选型的考量，旨在构建一个更智能、更高效的运维支撑体系。
- **案例分享：**通过具体的实践案例，生动展示智能运维在实际运维场景中的应用效果，用真实世界的经验来阐释前述理念和升级思路。

1、大模型给智能运维带来新的突破

Gartner提出了AIOps的理念，定义为：AIOps平台的人工智能将大数据，AI 机器学习和其他技术相结合，通过主动，个性化和动态的洞察力支持所有主要IT运营功能。AIOps平台支持同时使用多个数据源，数据收集方法，分析技术和可视化等技术。
算法式的智能运维主要应用在指标与日志数据分析，仅适用于部分应用场景。随着大模型的兴起，新增了**生成式智能运维发展路线**，形成“**传统智能算法+专家经验+平台智能体+生成式算法**”相结合的人机协同的智能运维发展。





Gartner 早在多年前就提出了 AIOps 的概念，即人工智能驱动的 IT 运营，其核心是通过大数据、机器学习和 AI 技术，为 IT 运营提

供持续、个性化和动态的洞察，从而支持性能分析、体验管理以及自动化操作。AIOps 平台通常具备多数据源整合、数据采集、分析和可视化等能力，旨在实现 IT 运营的智能化和自动化。

然而，传统的基于算法的智能运维并非没有局限。它主要依赖于预设算法处理指标和日志数据，应用场景相对有限。

但随着大模型的异军突起，我们惊喜地看到，智能运维领域出现了一条全新的发展路径——生成式智能运维。

未来的智能运维发展模式，将是**“传统智能算法 + 专家经验 + 平台智能体 + 生成式算法”深度融合的人机协同模式**。这种模式将集各家所长，极大地提升运维效率与准确性。

左侧的图表清晰地展示了 AIOps 的核心构成及其主要功能领域。它是一个多层结构的环，中心是 AIOps 本身，内层是支撑技术——机器学习、大数据和平台。中间层强调持续洞察（Continuous Insights）是贯穿始终的流程。而最外层则勾勒出了 AIOps 所覆盖的主要 IT 运营职能：

- 性能分析 (Observe): 利用模式识别、异常检测、分类、因果确定等技术进行深入洞察。
- 体验管理 (Engage): 涉及事件处理、通知、CMDB 等，关注用户和业务体验。
- 交付自动化 (Act): 通过智能自动化实现故障分类分级、运行手册自动化、自动修复、CI/CD 等，提升运营效率。

右侧的表格则对传统的基于半监督算法式智能运维与新兴的生成式 AI 智能运维进行了直观对比，突出了生成式 AI 带来的范式转变：

对比项目	传统基于半监督的算法式智能运维	生成式 AI 智能运维
工作原理	用少量标注和大量未标注数据建模型	大模型先学再调，找数据隐藏规律
学习数据	主要用指标数据，少量人工标注	融合各种数据，啥都能学
学习能力	用未标注数据，适应新场景弱	大数据学习，适应新场景强
优势	标注少也能训练，算法不复杂	诊断疑难故障，给出周全方案
局限	依赖人工标注，新问题难处理	训练费资源，不知咋做的决策
主要应用场景	数据稳定、故障常见的情况	复杂系统和要求高的运维场景
对于人机协同模式的帮助	帮着核对处理，完善标注数据	给分析建议，和人配合超

总结来看，生成式 AI 展现出显著的优势，但传统模式依然有用武之地。未来的智能运维将是两者的协同发展，并通过将专家经验和平台能力融于一体，最终构建起图中间模拟对话界面所展示的，一个能够与人流畅交互、提供智能协助的新型运维体系。正因如此，随着大模型的兴起，智能运维正朝着一个以传统智能算法、专家经验、平台智能体以及生成式算法深度结合的人机协同方向加速演进。

2、我们对智能运维的理解

那我们对智能运维的理解是打造智能化的人机协同运维体系，其核心在于：

- 通过智能技术与专家经验的沉淀，构建具有专业技能的机器人。
- 辅助运维专家进行系统的感知、决策与执行。

这也就意味着，智能运维需要持续提升以下六大核心能力：

- **实时获得“发生了什么？”：** 快速感知系统当前状态。
- **关联分析“为什么会发生？”：** 深入探究问题根本原因。
- **智能预测“将会发生什么？”：** 预判未来潜在风险。
- **决策判断“采取什么措施？”：** 根据分析结果制定应对策略。
- **自动执行“如何快速执行？”：** 实现自动化操作与响应。
- **实时感知“工作执行的效果？”：** 对执行结果进行反馈与评估。

而实现“人机协同”的关键，在于有效结合运维专家与机器人的特点，并通过“数据+平台”进行管理。这包括：

- 沉淀专家经验，赋能机器人：将运维专家的宝贵知识和经验转化为可复用的、自动化的处理逻辑。
- 以“数据+算法”赋能运维专家：利用数据和算法能力增强运维专家的实时感知与辅助决策能力。
- 增加运维机器人岗位：通过机器人承担或重塑操作性、规律性、大计算、7x24 小时运行以及提升人机体验等工作类型。
- 建立平台化管理：通过统一平台实现对整个智能运维流程的协同与自动化执行，确保决策的有效落地。

总之，智能运维旨在通过人机协同，利用智能技术提升运维体系的感知、分析、预测、决策和执行能力，从而更高效、更可靠地保障系统的稳定运行。

3、智能运维能力建设方案



智能运维建设的整体思路，概括来讲，我们构建了“四个中心”的建设体系，旨在整合现有资源，实现运维能力和服务智能化升级。

首先，我们强调“以场景驱动”，将现有运维平台的能力转化为“运维 Agent 智能体能力中心”。这就像是把运维平台的各种“服务”打包成一个个智能化的工具箱，能够快速响应不同的运维场景需求，并最终落地到公司统一的智能体编排平台。

其次，我们将现有的智能算法能力，例如异常检测、根因分析、风险预测等，进行统一封装和整合，形成“智能算法中心”。这个中心将作为智能运维的大脑，为各种智能应用场景提供技术支撑。

再者，我们对现有庞大的机器人和聊天群生态进行智能化升级，希望为这些工具“插上翅膀”。通过关联知识库，实现智能化的问答（比如使用 RAG 问答技术），并为它们绑定智能体，让用户可以直接通过机器人与智能体进行交互。这不仅提升了现有工具的智能化水平，也鼓励并赋能了员工自主构建更多智能机器人。

最后，也是至关重要的，我们致力于激发全体运维专家沉淀宝贵的运维经验，以此构建一个“智能专家知识中心”。这个知识中心是智能化的基石，它不仅仅是文档和数据的集合，更是经过训练和提炼的、可供智能化应用调用的运维智慧库。例如，我们将各类运维领域的原始数据，包括厂商文档、API 文档、技术文档以及系统运行数据，都纳入训练范畴，确保知识的全面性和时效性。

智能运维建设



为了实现运维的智能化转型，我们在既有的运维平台基础上进行了核心能力的增强与重塑。

首先，我们构建了一个全局智能助手，它的强大之处在于：

- 能够动态加载和启用各类智能体。
- 具备强大的上下文感知能力，能理解当前页面的状态与用户的操作意图。
- 能够集成我们的运维知识库，提供智能问答服务。
- 支持将多个智能体的工作流程进行编排，实现复杂任务的自动化执行。

其次，我们落地了“无代码 AI + ChatOps” 的新范式，让运维人员能够更便捷地利用智能化能力，便捷地维护与更新运维知识库，为一线团队提供最新信息赋能。

- 在日常的企业微信群聊中，可以直接基于知识库进行智能问答，将知识转化为即时支持。

- 通过采集分析企业微信的聊天记录，系统能够感知到聊天的上下文，提供更加精准和智能的帮助。

最后，我们将所有智能体进行了统一集成和管控，这不仅能最大化工具的复用效率，也使得上层敏捷场景能够灵活地复用这些智能体能力，加速业务创新与问题的解决。

4、大模型应用案例：AI 辅助故障定位案例





AI 辅助故障定位系统，旨在利用大型模型（AI）的能力，在系统故障发生时，为 SRE 和运维人员提供智能化的支持，从而加快故障的定位和解决速度。

图示界面清晰地划分了两大区域：左侧是智能定位辅助信息模块，右侧则是故障详情与智能分析的核心交互区。

在左侧的智能定位辅助信息区域，系统会根据发生的具体事件（如 IMS-2025*-006）和受影响的系统，推送一系列有价值的辅助信息，包括：

- 前三个交易日变更情况：列出近期可能与故障相关的系统变更（如 CAB-20250307-001, CAB-20250313-017），帮助快速锁定导致故障的潜在变更源。
- 应急中心可观测感知：展示应急中心层面汇集的系统异常感知，例如系统健康度（当前为 97%），并提供链接供进一步查看详细的可观测数据。
- 统一告警事件：汇总与当前故障关联的统一告警信息，如近 2 小时的告警数量（当前显示 6 条），便于了解故障范围和影响。
- 历史相似事件：智能推荐与当前故障现象或类型相似的历史事件（如 IMS-20220418-004, IMS-20220916-006），为排查提供宝贵的历史经验参考。
- 推荐应急策略：基于大模型的分析，系统会提供针对性的应急处理策略建议（例如针对“对账单发送失败”、“中登报盘服务器故障”等具体场景的建议）。特别需要指出的是，这些建议由大模型筛选生成，使用时需要人工核对确认其有效性与适用性。
- 右侧区域是故障详情和核心的 AI 排障智能体交互区。这里详细展示了故障的原始描述（如“上周发版的服务连接出现‘Unknown prepared statement handler (5) given to mysql_stmt_precheck’”），并允许用户或系统向 AI 发起排查请求（例如询问“这个故障怎么排查？”）。
- 排障智能体收到请求后，会基于对故障描述和相关信息的综合分析，提供初步的故障原因分析和定位建议。以“Unknown

prepared statement handler (5) given to mysql_stmt_precheck”错误为例，AI 给出了以下四种可能的深层原因分析和解释：

1. SQL 语句预处理问题：详细阐述该报错通常指向预处理语句的状态异常，可能因其被提前关闭或初始化不当所致。

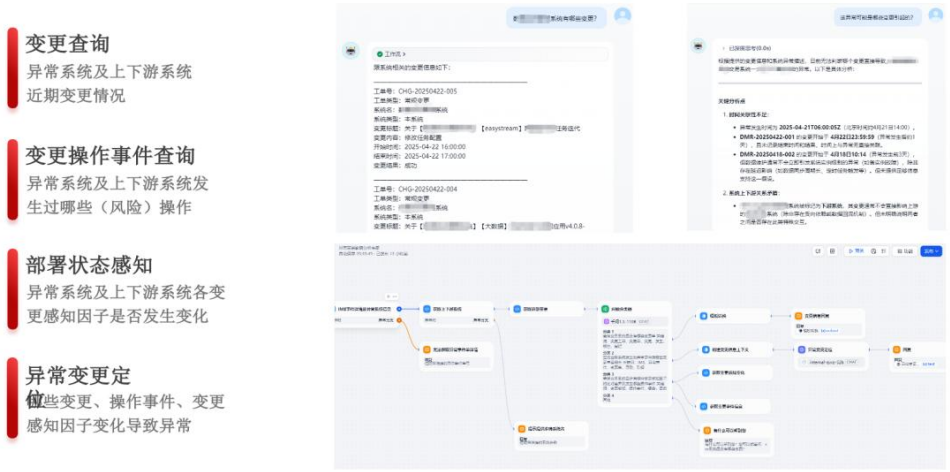
2. 数据库连接问题：分析数据库连接层面的问题也可能导致预处理语句执行失败，诸如连接池配置错误、连接超时或非正常连接关闭等都可能是原因。

3. 代码逻辑问题：指出应用层代码中可能存在的逻辑缺陷，例如预处理语句的创建和执行流程不符合规范或顺序错误，会影响其正常处理。

4. 数据库版本兼容性问题：考虑到近期可能的数据库版本变动，AI 提出了版本不兼容的可能性，认为新版本与现有预处理语句机制可能存在冲突，导致工作异常。

5. 通过这样一个 AI 辅助系统，我们将故障发生后的被动响应，提升至了 AI 驱动的智能分析与主动建议模式，极大地增强了故障定位的效率和准确性，让运维团队能够更快速、更精准地解决问题，最大程度地降低故障对业务的影响。

5、大模型应用案例：异常变更定位助手



借助之前变更管控系统沉淀的变更信息、变更对象变动数据以及操作事件记录，我们构建了一个异常变更定位助手。

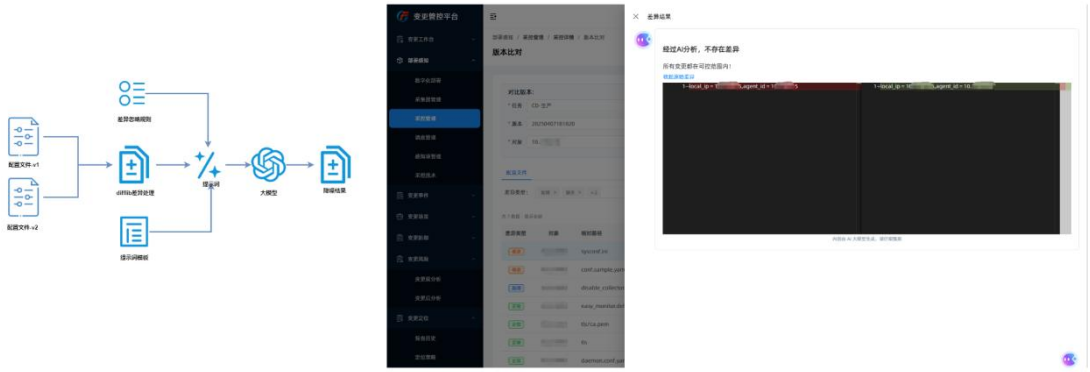
通过这个助手，我们可以便捷地查询异常系统及其上下游系统近期的变更情况。

更进一步，我们能够结合这些数据，利用大模型的推理能力，协助快速定位是哪些具体的变更或变化导致了系统的异常。

在实践中，我们采用的是千问的 QwQ 推理模型，它的推理能力令人印象深刻，能够有效地识别不同变更之间的时间关联性，以及它们与上下游系统之间的复杂关联关系，从而提升异常变更定位的效率和准确性。

6、大模型应用案例：感知采集对比一键降噪

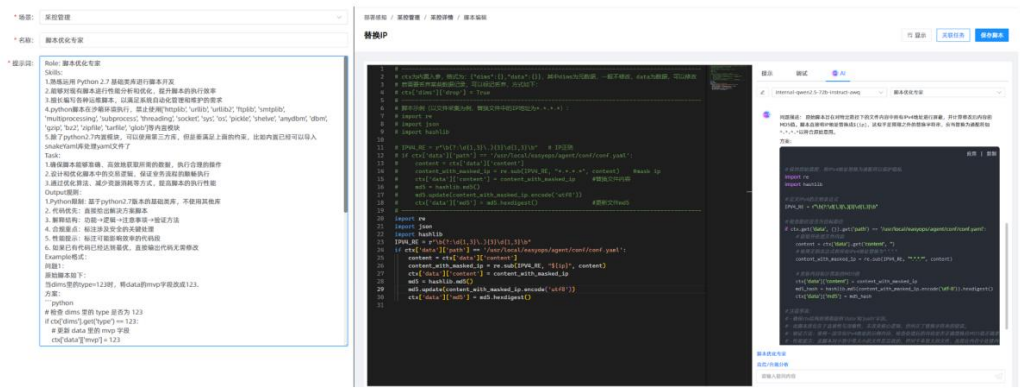
大模型应用案例：感知采集比对一键降噪



在构建感知采集体系时，我们面临着一个关键挑战：如何在不同环境（例如生产环境与测试环境）之间进行比对时，有效处理那些确定已知、但因环境差异而必然不同的信息，比如 IP 地址或密码。这些差异信息会产生“噪音”，干扰我们对真正问题的判断和定位。

为了解决这个问题，我们专门利用大模型的强大能力，开发了一个降噪功能。其核心思想在于识别并忽略掉这些已知差异，从而使我们在比对不同环境的数据时，能够专注于那些更能反映潜在问题的关键信息。这极大地提升了我们数据分析的效率和准确性，确保我们能更快、更精准地发现系统异常。

7、大模型应用案例：脚本优化与合规检查

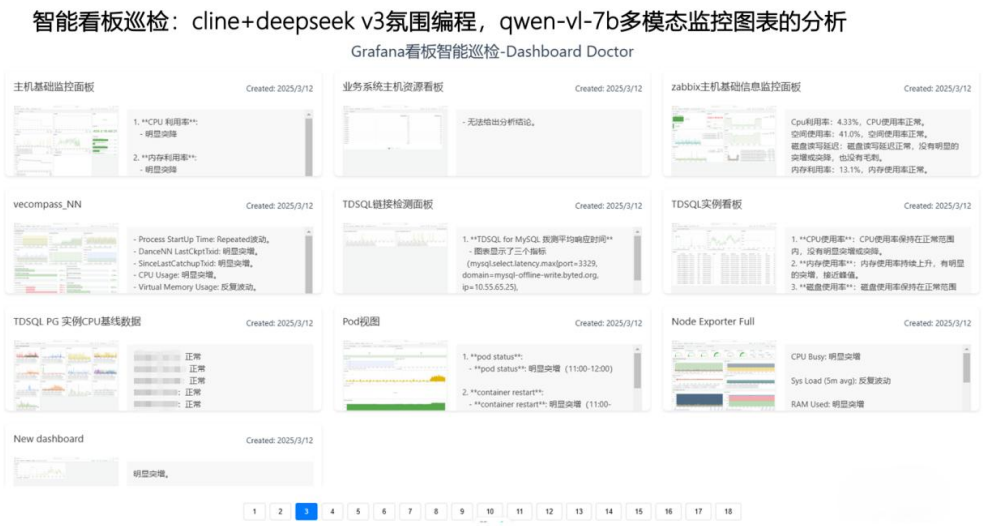


大模型目前展现出的一个非常突出的能力，在于它强大的代码生成与优化能力，基于此，我们在当前运维平台中深度融入了多项智能化功能，旨在通过集成大模型技术，实现：

- **代码优化能力：** 大模型能够对存量代码进行深入分析，识别潜在的性能瓶颈及冗余逻辑，并提出更高效、更简洁的实现方案，显著提升代码质量与运行效率。
- **合规检查功能：** 自动化地对照预设的技术规范、安全标准以及内部最佳实践，对代码进行全方位的合规性检查，确保所有提交的代码都能满足严苛的要求，从源头减少潜在的技术风险。

通过将这些智能化能力嵌入到日常运维流程中，我们正积极探索大模型技术在提升运维效率、保障系统稳定性方面的巨大潜力，这不仅是运维体系从数字化迈向智能化的关键一步，更是应对复杂系统挑战的必然选择。

8、大模型应用案例：多模态场景探索



在对智能化的持续探索中，我们也尝试了多模态场景的可能性，旨在探索大模型在更广阔领域的应用边界。

我们首先尝试将 Zabbix 的监控看板输入给多模态大模型进行识别分析，初步结果显示其具备一定的识别能力。

紧随其后，我们考虑到运维场景下监控看板作为最普遍的数据呈现形式，决定进一步利用无头浏览器对 Grafana 看板进行截图，并将这些截图输入给大模型进行分析。实验结果令人振奋，模型确实能够识别出图表中存在的异常模式，例如 **尖峰** 或 **突变** 等情况。然而，需要清醒地认识到，目前这项功能尚处于起步阶段，距离在生产环境中真正可用仍有 不小的差距。

尽管当前成果有限，但这方面的想象空间是巨大的，预示着大模型在辅助运维智能化决策方面拥有广阔的应用前景。

9、大模型探索与展望



好的，基于大模型在运维领域的探索与展望，我们目前聚焦放在三个核心方向。

“当下”我们正在进行的努力：

- 专家知识库的建设：这块我们主要通过 RAG (Retrieval-Augmented Generation) 技术来构建运维专家知识库，以期实现更精准的知识检索和应用。
- 沉淀专家经验的智能体编排：我们致力于将优秀的运维专家经验转化为可复用的智能体，并通过智能体编排的方式进行应用，让专家的智慧能够规模化赋能。
- 智能体与运维场景的深度融合：将这些沉淀下来的智能体与具体的运维场景紧密结合，让智能体在实际生产环境中发挥作用。

展望“下一步”的发展：

- 进一步探索更高级的智能体策略，充分利用大模型的推理能力去探索更复杂的运维的问题，尤其是在以下几个方向：

- 智能体策略探索：这包括对 FunctionCall (调用外部工具与函数)、ReAct (Reason + Act，推理与行动结合)、Plan&Execute (规划与执行) 以及可能更 High-level 的 MCP ReAct (结合 MCP 的 ReAct 框架) 等多种智能体策略进行深入的研究和实践。
- MCP 与多智能体协作：我们也希望通过探索 MCP 和多智能体协作的能力，让不同的智能体能够协同工作，解决更复杂的运维挑战。
- 运维领域的 Deep Research：我们计划在运维领域开展更深入的研究，重点探索智能排障和深度健康检查等核心痛点。

（三）总结及展望

至于“未来”，随着大模型技术的飞速发展，我们预见将涌现出能力更强大的模型，拥有更卓越的推理能力和多模态处理能力，并且将具备更强的运维通用知识。同时，行业会逐步构建 更加专业的知识库，不仅包括通用的运维知识，更会细化到银行、证券等金融行业的 专业运维知识库。我们对此的态度是积极观望，一旦这些基础设施成熟，将其应用到我们的实践中。

6 上线后持续优化工作

6.1 用户体验优化

解释什么是用户体验优化之前，我们先来解释一下什么是用户体验。

感官、情感、认知、行为等各个方面，一个好的用户体验应该是用户感到舒适、方便、快捷、高效和满意的。

而用户体验优化则是指通过维护系统的高可用性、减少系统的错误率、加快用户的响应速度等方面来提升用户在系统上的使用体验。

在 SRE 中，用户体验优化是一个重要的目标，它能够提高用户的满意度和信任度，增大用户对业务的黏性和忠诚度，从而增加用户的留存率和转化率，提高产品或服务的竞争力和市场占有率。

6.1.1 基于用户端的直接用户体验优化

基于用户端的直接用户体验优化，主要是针对用户直接接触到的产品或服务的方面进行优化，包括用户卡顿（页面加载速度、响应时间）、用户网络等方面，以提高用户的使用体验和忠诚度。

1. 面向用户端的优化措施

1）用户端卡顿优化：通过提高客户端的性能（优化代码、减少资源消耗、改进算法等）、压缩和合并前端资源以及使用缓存和预加载等方式来提高页面加载速度，减少用户等待时间和卡顿现象；

2）用户端网络优化：通过使用网络传输加速技术（包括使用 TCP 拥塞控制算法、调整 TCP 窗口大小、使用 TCP 连接池等）和双通道加速技术来提高数据传输的效率、减少用户网络的延迟以及增强通信的稳定性；如当用户处在弱网情况下，能利用双通道加速技术（如 Wi-Fi 和 LTE 同时发包的方式）来降低用户的网络时延和增强用户通信的稳定性，就可以减少用户流失。

3）用户端其他优化：除卡顿和网络优化这种较为通用的优化之外，还有一部分是基于业务逻辑，对用户请求的反馈提示的优化。例如：触发了排队机制的提示信息，服务器满员的提示信息等。这些信息有效提示，可以引导用户等待或者引导用户到其他有资源的途径，让用户感受到系统仍然在工作，避免流失。

6.1.2 基于系统端的间接用户体验优化

基于系统端的间接用户体验优化，主要是针对用户无法直接接触到的产品或服务的方面进行优化，包括系统的性能、架构、容灾等方面，以保证系统的鲁棒性、健壮性和可用性，提高用户的信任度和满意度。

1. 面向系统端的优化措施

1）基于鲁棒性的优化，提升系统健壮性：通过优化系统的容量规划、容错机制、架构设计（如弹性和可伸缩性）和负载策略（如负载均衡、优雅降级和限流策略），确保系统在高负载和故障情况下仍能正常运行，防止因系统故障而对用户造成影响；例如：系统突发大量请求增加时，系统能够自动扩展和收缩资源以适应变化的

负载，且系统能够实施限流和排队机制，限制并发请求的数量，并将请求放入队列中进行有序处理，避免系统过载和崩溃。

2）基于幂等性的优化，提升系统可用性：通过将系统关键操作设计为幂等操作，并且在关键操作中，实施幂等性检查机制，如果关键操作失败或中断，可以使用事务和回滚机制来确保操作的幂等性，避免数据不一致或重复操作；如系统接收到重复请求时，系统可以检查请求 ID 或其他标识符，并判断是否已经处理过该请求；如果已经处理过，则可以忽略重复请求或返回相同的结果；或者如果系统操作失败或中断时，系统可以自动回滚到之前的状态，避免数据不一致或重复操作，并且通过系统自动重试后，能够被正确的处理。

在 SRE 中，用户体验优化措施实施之后，我们还需要一系列的度量指标来衡量和评估优化的效果：

1）用户满意度（User Satisfaction）：通过用户反馈、调查或评级来衡量用户对系统或服务的满意程度，这可以通过定期的用户调查、NPS（Net Promoter Score）或用户反馈收集来评估。

2）用户可用性（User Availability）：通过监测系统的正常运行时间、故障时间和恢复时间来衡量系统或服务在用户可访问性方面的表现，关键指标包括平均故障间隔时间（MTTF，Mean Time Between Failures）和平均故障恢复时间（MTTR，Mean Time To Recover）。

3）系统错误率（Error Rate）：通过监测错误请求的百分比、错误码的数量或错误率来衡量系统或服务在处理用户请求时发生错误的频率，较低的错误率通常表示更可靠的系统。

4）系统或者页面响应时间（Response Time）：通过监测平均响应时间、最大响应时间和百分位数（如 P99）来衡量系统或服务对用户请求的响应速度，较低的响应时间通常意味着更好的用户体验。

5）用户反馈响应时间（User Feedback Response Time）：通过监测用户反馈的处理时间、解决问题的时间和用户满意度来衡量团队对用户反馈的响应速度。

6）客户端性能状态指标（Client Perf Status Metrics）：通过监测客户端关键路径上各动作的状态和发生时间，了解和衡量用户端操作和运行是否符合设计预期，主动确认是否存在优化的可能性。

除了以上通用型指标，还建议对用户使用的关键路径进行监控，关键路径监控是指关注那些对用户体验有决定性影响的系统组件或交互流程。这些路径是用户完成主要任务所必须的，比如电商网站的结账流程或者社交网络的信息发布流程。

关键路径监控通常包括：

（1）端到端事务跟踪：追踪用户请求从开始到结束的全过程，确保每个步骤都按预期工作。

（2）性能瓶颈识别：识别并优化那些延迟用户操作的关键组件。

（3）可靠性保障：确保关键路径上的服务具备高可用性和容错能力。

（4）实时警报：当关键路径上的性能下降或出现错误时，实时通知团队进行干预。

（5）影响评估：分析关键路径性能问题对用户体验和业务目标的影响。

通过关键路径监控，SRE 团队可以更全面地理解用户体验，并采取针对性的优化措施。这些措施不仅仅关注技术性能指标，而且着眼于用户行为和感知，同时可以更为集中火力，获得更好的优化效果。

6.2 运维琐事的日常管理及优化

6.2.1 运维琐事的介绍

运维琐事是指运维处理一些手动性、重复性、可以被自动化的、被动响应的、没有持久价值的工作，而且琐事与服务呈线性关系的增长。运维琐事包括系统监控、故障排查、配置管理、容量规划、数据备份等。当然并不是每件琐事都有以上全部特性，但是每件琐事都满足其中的一个或多个特点。下面对运维琐事的特点进行介绍。

1. 手动性

例如对某个业务的服务进行停服处理，需要登录服务器，然后执行停服命令。如果是通过在作业平台上执行停服指令脚本，可以

减少登录服务器的限制；如果是通过流水线作业来串联起该脚本指令，那么效率会更高。那这里手动登录服务器执行停服命令就被认为是琐事。

2. 重复性

如果某件事是第一次做，甚至是第二次做，都不应该算作琐事。琐事就是不停反复做的工作，例如业务的停服更新。如果你正在解决一个新出现的问题或者寻求一种新的解决办法，不算做琐事。

3. 可以被自动化

如果某个需求，除了业务运维可以手动处理外，也可以通过作业流程工具进行自动触发某个任务来解决该需求，就说明该需求可以被自动化。例如业务停服更新完起服后，业务运维需要检查服务是否都正常启动好了，这个需求就可以通过增添流程节点，检测起服日志中是否含有 success 等关键字来自动化判断服务进程是否都启动正常。

所以如果一个需求可以通过流程自动解决，或者是流程优化消灭了该类需求，那么这类需求就算作琐事。

4. 被动响应

处理琐事是处理那些突然出现的、被动去响应的工作，而不是主动安排的工作。处理紧急告警是琐事，我们可能永远无法完全消灭这类工作，但是我们也得必须努力减少这类需求，减少对运维的打扰。

5. 没有持久价值

如果在你完成某项工作后，这类工作的状态没有发生改变，这类工作就很可能是琐事。如果我们做了能给这类工作带来永久性改进效果的优化（例如操作步骤减少了），那么你做的事情就不是琐事。例如我们梳理了业务的核心配置文件，然后将其接入了进程配置管理平台，来达到平台化管理业务配置文件的目的，就不是琐事。

6. 与服务呈线性增长

如果在工作中所涉及的任务与服务的大小、数量呈线性增长关系，那这项任务就可能属于琐事，例如申请服务器这个操作，每个业务都有搭建新服的需求，都需要申请服务器，那么申请服务器的工作会随着服务业务的数量呈线性增长，那么我们可以通过在每个业务的流程里，自动嵌入申请服务器的节点，自动申请服务器来达到优化这个琐事的目的。

综上所述运维琐事的 6 个特点，可以发现如果对琐事不加以管理，运维很可能会被这些运维琐事淹没，影响工作执行质量和执行效率，也造成运维人才能力的提升空间受限，长期下去消耗工作热情。

6.2.2 运维琐事的质量管理

运维琐事的质量管理是指通过一些管理措施和方法来确保运维琐事工作的高效性和稳定性，来降低系统故障率、减少人为故障、提高系统的可用性和稳定性。通过建立规范的操作流程、使用自动

化工具、培训与管理人员、持续改进等手段，可以提高运维琐事的质量管理水平，确保系统正常运行和业务的连续性。

运维琐事的质量管理包括以下方面：

（1）流程管理：通过规范的运维流程，包括工作分工、操作步骤、审批机制等。确保每个运维操作都有明确的责任人和标准化的执行流程。例如我们每一次发布变更都需要有完整的操作 checklist 一样。

（2）自动化工具：引入自动化工具和脚本来处理重复性和繁琐的运维任务，避免手工操作。通过将操作内容自动化，来屏蔽操作人员能力的差异性，可以减少运营故障的发生。

（3）技能培训与人员管理：为运维人员提供必要的培训和技能提升机会，使其具备应对不同琐事的能力。同时，合理分配人员资源，确保每个运维任务都有专人负责。例如定期对团队内成员进行平台工具的标准化操作方案的培训。

（4）持续改进：定期回顾运维工作的过程，例如对操作任务的成功率的统计，对每一项任务进行分析，发现问题并制定改进措施，因为任何一个问题都会传递到业务运维，需要业务运维分析并解决，通过持续改进来提高运维琐事的质量。

6.3.3 运维琐事的效率管理

运维琐事的效率管理是指通过标准化、工具自动化、OnCall 需求分层、优先级管理和持续改进等手段，来实现高效的运维琐事管理，为组织带来更高的价值和效益。

运维琐事的效率管理包括以下方面：

（1）标准化和流程优化：建立规范的运维流程和操作步骤，确保每个操作都按照统一的标准执行，通过流程优化，消除冗余的步骤和无效的等待时间，提高工作效率。

（2）工具自动化：通过自动化工具平台的能力，减少人工干预和手动操作，提高操作速度、降低人为错误，达到释放运维精力用于更复杂和有价值的工作。例如通过 CMDB 平台（运维配置管理数据库）来管理业务的硬件等资源、通过进程配置管理平台来管理业务的核心配置文件和进程启停信息、通过监控平台来管理业务的服务器资源使用情况、通过作业管理平台来批量管理业务的任务模板等。

（3）oncall 需求分层：OnCall 是指业务运维团队将工作时间划分为若干个固定的时段，每个时段都会有专门的运维人员负责值班处理需求，从而达到可以随时响应业务需求，确保服务的连续性和稳定性，oncall 在进行响应需求时，需要对需求进行分层处理，例如根据需求的紧急程度进行分层、根据需求的工作范围进行分层、根据需求的所属平台进行分层等。通过分层后的需求，就可以分配到不同工种来解决，实现问题的高效处理。

（4）优先级管理：通过 OnCall 团队一线的需求梳理后，对每一项需求进行优先级管理，让团队一线通过标准化文档操作指引优先去处理紧急且重要的事情，避免运维人员工作精力分散，提高工单的流转率。

（5）持续改进：定期回顾运维工作的执行情况，例如对工单的类型比例分析、工单的耗时比例分析，这种数据驱动的策略来发现问题，例如发现哪一类需求比重比较高，耗时比较长的情况。持续优化，提高运维琐事的效率。

（6）文档沉淀：由于相关琐事较多，需要通过文档进行知识管理及文档的标准化，便于团队成功共享、查询和复用，

（7）AI 提效：利用人工智能技术来提高运维工作的效率，减少重复性和低价值的工作。具体措施非常广泛，现阶段，部署聊天机器人作为一线支持，对接文档支持库，处理常见的查询和简单问题，减轻人工客服的负担，是在提效上面非常值得探索，且可落地性强的方向。

6.3 业务全生命周期工具建设

业务全生命周期主要包含研发期，上线期，运营期以及长尾期阶段，在每个阶段都需要有能提升运营效率的通用化工具。建设的工具包含版本控制，监控和告警，日志管理，CI/CD，数据备份，自动化测试，容器编排，业务巡检等工具。

在 SRE 中，业务全生命周期工具建设是非常重要的一部分，通过使用这些工具和技术，SRE 团队可以更加高效地管理和优化业务的整个生命周期，提高业务的各个阶段的可靠性和稳定性，降低成本和风险。

6.3.1 研发期工具建设

业务研发期是指产品研发阶段，这个阶段需要使用一些工具来辅助研发和管理。在业务研发期，工具建设可以提高系统的可靠性、可用性、可扩展性以及研发效率，以支持产品的研发和快速迭代。

主要包括如下几类：

（1）版本控制工具：例如 Git，用于管理代码的版本、分支、合并和协作等。研发期版本控制工具是必要的，帮忙研发人员更高效的进行项目的开发。

（2）自动化构建工具：例如 Jenkins，用于自动化构建、测试和部署代码，提高开发效率和质量。研发期 SRE 人员参与较少，自动化构建功能实现研发人员版本自动化部署能力。

（3）代码质量工具：用于分析代码质量、安全性和可维护性等，帮助开发团队提高代码质量和可维护性，确保代码不会有安全的漏洞。

（4）代码审查工具：用于进行代码审查和评审，提高代码质量和可维护性，可提高代码的健壮性，减少程序可能出现的 BUG。

（5）测试工具：用于进行单元测试、集成测试和 UI 测试等，提升测试的效率。

效果评估：

（1）提高开发效率：使用自动化构建、测试和部署工具可以减少手动操作，提高开发效率和质量。

（2）提高代码质量：使用代码质量工具和代码审查工具可以发现和修复代码中的问题，提高代码质量和可维护性。

（3）提高团队协作效率：使用项目协作工具可以提高团队协作效率和沟通效果，减少沟通成本和误解。

（4）提高测试效率：使用测试工具可以更多的覆盖测试用例和场景，并且提高测试效率。

6.3.2 上线期工具建设

业务上线期是指产品上线阶段，这个阶段需要使用一些工具来辅助上线。上线期工具可以提高系统的可靠性、可用性和可扩展性。以支持产品稳定上线和运营。

主要包括如下几类：

（1）监控和告警工具：用于监控业务的运行状态和性能指标，并在出现问题时发出告警。

（2）日志管理工具：用于收集、存储和分析业务的日志数据，业务出现问题能及时发现。

（3）自动化部署工具：用于自动化部署业务代码到生产环境。

（4）故障排除工具：用于快速定位和解决业务故障。

（5）容器编排工具：用于管理和编排容器化的业务应用。

（6）上线风险检查工具：用于保障业务的健康状态工具集，工具可以定时巡检，保证业务实时的稳定性。

效果评估：

（1）提高上线效率：使用自动化部署工具可以减少手动操作，提高上线效率和质量。

（2）提高系统可靠性：使用监控工具和日志管理工具可以及时发现和解决系统中的问题，提高系统的可靠性和可用性。

（3）降低业务风险：使用上线检查工具可以保障业务上线环境的稳定性，降低上线风险问题的发生。

6.4.3 运营期工具建设

业务稳定运营期是指产品上线后的运营阶段，这个阶段需要使用一些工具来辅助运营和管理。在业务稳定运营期，建设工具来提高系统的可靠性、可用性，以支持产品的稳定运营和业务增长。

主要包括一下几类：

（1）监控和告警工具：用于监控业务的运行状态和性能指标，并在出现问题时发出告警。

（2）日志管理工具：用于收集、存储和分析业务的日志数据，业务出现问题能及时发现。

（3）自动化部署工具：用于自动化部署业务代码到生产环境。

（4）故障排除工具：用于快速定位和解决业务故障。

（5）容器编排工具：用于管理和编排容器化的业务应用。

（6）健康巡检工具：用于保障业务的健康状态工具集，工具可以定时巡检，保证业务实时的稳定性。

（7）成本分析与优化工具：用于运营期业务的成本分析和成本优化。

（8）安全巡检工具：用于运营期安全风险的检查，预防安全事件。

效果评估：

（1）提高系统可靠性：使用监控工具，日志管理工具，健康巡检工具，安全巡检工具，可以及时发现和解决系统中的问题，提高系统的可靠性和可用性。

（2）提高系统性能：使用负载均衡和容器编排相关工具可以提高系统的性能和可扩展性，支持高并发和大流量的访问。

（3）提高运维效率：使用自动化运维工具可以减少手动操作，提高运维效率和一致性。

6.4.4 下线期工具建设

业务下线期是指产品不再提供服务，准备关停服务。这个阶段需要使用一些工具来操作下线的流程。在业务下线期间，使用工具长久保存用户数据并对相关资源进行回收。

主要包括以下几类：

（1）数据备份工具：用于重要数据永久备份，业务退市后数据库一般是需要长久备份，用于后续数据的查询需求。

（2）资源回收工具：业务下线后，为了节省成本，需要尽快对相关资源进行回收。包括但不限于服务器资源，CDN 资源，COS 资源。需对业务相关的资源全部清理回收处理。

（3）遗留成本检查工具：为了确保业务相关所有资源都及时回收，需要工具对涉及到的相关资源使用情况进行检查。确保所有资源都得到回收和释放。

（4）权限检查工具：业务下线后，对应业务的相关权限，都需要工具来检查，例如 AKSK，周边平台的权限都因彻底回收。

效果评估：

（1）数据可追溯：业务下线后，相关的重要数据都能查询。如果外网用户需要查询数据核对情况，均能查询到用户数据。

（2）下线业务无运营成本：业务相关资源全部回收和释放，下线后业务成本为 0。

6.4 运营成本分析及优化

6.4.1 运营成本分析及优化的必要性

运营成本是任何产品都会涉及到的问题，其重要性是不言而喻的。通过运营成本分析，我们便可知一个产品的成本有哪些部分构成，各部分是否都是合理的，是否需要做优化。持续的成本优化是保障一个产品健康发展的有效手段之一，需要贯穿到整个产品的全生命周期来执行。

6.4.2 运营成本实时监控

由于误操作，程序应用代码错误，黑客攻击大量，而导致使用使用资源飙升，成本失控的情况屡见不鲜，因此，一定要对运营成本进行实时监控，避免无谓浪费。

1. 建立成本监控系统：

利用现有的监控工具或开发专门的成本监控系统，实时追踪云服务使用情况、服务器资源占用、带宽消耗等关键成本指标。

将运营成本数据与业务指标（如用户活跃度、交易量等）关联，以便从业务角度理解成本变化的影响。

2. 设置预警机制：

为各项成本指标设定阈值，当成本超出正常波动范围时，系统自动触发预警通知相关负责人。

预警机制应包括多级告警，以应对不同程度的成本波动情况。

3. 细化成本分类：

将成本分解到更细的粒度，如按服务模块、按功能点、甚至按代码提交进行成本追踪，以便更精确地定位成本波动的来源。

4. 每日账单监控：

账单自动化获取，配置云服务提供商或财务系统的 API，实现每日账单数据的自动获取，并与监控系统对接。如实现难度较大，SRE 起码养成每日登陆系统查看账单习惯，或者订阅邮件账单

6.4.3 运营成本分析及优化的指标

1. 财务指标

财务指标是运营成本分析及优化的重要指标，只有在财务的成本数据体现了成本降低，才是有效的成本优化。

财务指标包含：

1）单用户成本

单 PCU 成本：业务总运营成本除以业务最高在线人数（PCU）

单 DAU 成本：业务总运营成本除以业务日活跃人数（DAU）

单 MAU 成本：业务总运营成本除以业务月活跃人数（MAU）

2）单用户成本增长率

在业务的不同运行周期，单 PCU 成本、单 DAU 成本、单 MAU 成本中，若某个维度与历史数据比较，增长率为正，则说明该维度的运营成本在上涨，需要做运营成本的优化。相反，若增长率为负，则说明该维度的运营成本在下降。

2. 技术指标

SRE 对于成本优化动作主要提现在技术方案上，而技术指标则是体现技术方案是否有效的主要指标。

常见技术指标包含：

CPU 利用率：衡量服务器资源的 CPU 使用情况，通常使用 CPU 周均峰值，根据模块功能不同考核标准也不同，接入层、逻辑层、存储层的 CPU 利用率的考核标准会有差异。

内存利用率：衡量服务器资源的内存使用情况，业务进程类型不同，内存使用率也会有差异，消耗内存型的业务通常内存利用率较高。

存储使用量：衡量服务器的磁盘使用情况，如磁盘空间使用率，磁盘 I/O 吞吐量等，数据型业务的服务器较多用到这个指标。

网络吞吐量：衡量服务器网络使用情况，例如网络流量使用量、数据包收发吞吐量等，网关型服务器较多用到这个指标。

CDN 带宽使用量：涉及到用户侧客户端下载的业务多用到这个指标。

SRE 在做成本优化时，需要结合财务指标和技术指标综合评估，既要保障成本在合理的使用区间，又不能影响业务的稳定性和可靠性。

6.4.4 运营成本的统计及分析方法

1. 运营成本的统计

运营成本分类统计主要分为以下维度：

IaaS 层成本：包含 AWS、谷歌云、腾讯云、阿里云、华为云、自研云等提供底层基础设施的服务提供商，公共的 CDN 服务、计算服务、存储服务、带宽服务等的成本统计。

PaaS 层成本：公共登录组件、数据平台、安全平台等提供公共服务的平台成本统计。

SaaS 层成本：运维工具、流水线、日志管理、版本发布工具、运营工具等的成本统计。

2. 运营成本的分析

1）单用户成本分析

业务整体单用户成本，从业务整体运营成本评估单用户成本是否处于合理使用区间。

IaaS 层单用户成本，SRE 分别评估 AWS、谷歌云、腾讯云、阿里云、华为云、自研云等不同基础设施层的单用户成本，除了在每一个独立的云服务范围内降低单用户成本外，在各种云厂商之间

可以横向对比单用户成本，并可以迁移至单用户成本较低的云厂商。

PaaS 层单用户成本，SRE 需要评估业务在各 PaaS 平台的单用户成本占比，对比 PaaS 平台的总体单用户成本与单业务的单用户成本，同类型业务的 PaaS 层单用户成本等，根据综合评估的数据，进行成本优化。

SaaS 层单用户成本，SRE 需要评估业务使用各 SaaS 产品的单用户成本占比，在单 SaaS 产品的单用户成本与其他业务的差异。

2）单用户成本增长率分析

业务会经历研发期，上线期，稳定运营期以及长尾期等阶段，同一维度在不同阶段都会有对应的单用户成本，SRE 可以对比同一业务在不同阶段的单用户成本增长率变化，对于单用户成本上涨的阶段，SRE 需结合财务指标、技术指标等综合方法降低单用户成本。

3）多业务对比分析

针对同一类型的业务，SRE 可以综合对比同类型业务的整体 CPU 利用率，单业务模块 CPU 利用率等，与同类型业务横向对比，评估所属业务的 CPU 利用率是否在合理使用区间。

4）资源利用率分析（以 CPU 为例说明，内存和存储类似）

针对同一款业务不同模块，可以分析 CPU 利用率是否在合理使用区间。对于消耗 CPU 的业务模块，和消耗内存的模块，CPU 利用率应该有不同的标准，SRE 需要给出合理的评估意见。

一款业务会经历研发期，上线期，稳定运营期以及长尾期等阶段。

在研发期，SRE 需要提供少量测试和开发服务器即可，对 CPU 利用率没有考核。

在业务上线期，SRE 需要保障业务充足的容量，优先满足业务用量需求。

在业务稳定运营期，SRE 需要评估业务整体 CPU 利用率、各模块 CPU 利用率等，并给出 CPU 利用率的优化建议，使业务 CPU 利用率处在合理使用区间。

在业务长尾期，SRE 除了评估单业务的 CPU 利用率，还可以采用多业务混布的方式，提高多业务的综合 CPU 利用率，达到成本优化的目标。

5）CDN 使用量分析

用户一旦涉及资源的下载和更新，便会产生 CDN 成本。CDN 成本通常位列运营成本的前三位。不同云厂商的 CDN 计费规则各有差异。国内厂商普遍采用 CDN 带宽计费模式，而海外厂商则多以 CDN 流量计费模式为主。

CDN 成本优化可围绕以下几个思路：

CDN 流量计费模式：该模式通常使用 CDN 累计使用流量来计费。尽量降低用户下载的资源量。比如可做差量更新的资源，优先使用差量更新，既提升用户体验也能降低 CDN 资源。

CDN 带宽计费模式：该模式通常使用 CDN 使用峰值带宽来计费。可通过各种技术策略降低 CDN 带宽。通过“削峰填谷”的方式，让用户下载更新资源时更分散到各个时间区间，避免所有用户集中在高峰期下载从而达到降低 CDN 带宽的目的，比如提前下载，预下载等策略。

6.4.5 运营成本的优化方法

1. 单业务优化方案

缩减业务容量：

（1）业务周期。业务会经历上线期，稳定运营期以及长尾期等阶段。在业务上线期，SRE 需要充分满足业务容量需求。在业务进入稳定运营期之后，需要根据业务实际运营情况，结合在线规模、服务器 CPU 利用率、内存使用率、存储使用量等综合情况，对资源容量进行缩容，在不影响业务稳定运营的前提下，降低资源容量，达到成本优化的目标。在业务长尾期。

（2）业务混部。除了降低资源容量外，在业务进入稳定期后，业务正常负载无法重复使用服务器资源，在一定程度上造成了资源浪费。此时，可以考虑多业务混部的方式降低业务运营成本，即，多个业务或服务部署在相同的服务器上，从而起到合理利用服务器资源的目的。

业务混部的方式，有一定的风险，需要做好全面的架构评估和技术方案评估，需要评估的方面有：

a. 业务等级评估

进行业务混部首先就是要进行业务等级的评估，如果业务等级很高，不容许有失败率，那么趁早放弃这个方案。适合进行混合部署的业务可能有如下特点：

失败不敏感，重试成功后不影响

不直接服务用户

无状态

b. 至少 3 个月稳定状态的性能评估

比如，连续 3 个月 CPU 利用率低于 5%，连续 3 个月内存利用率低于 10%等

c. 资源消耗互补评估

计划混部的业务在资源消耗偏好方面具有互补性，例如：A 业务属于 CPU 消耗型，B 业务属于内存消耗型，这两款业务可以考虑进行混部，即使某些特殊情况发生资源消耗增加也不至于两个业务相互争夺资源。

d. 峰值波动互补评估

计划混部的业务的业务峰值具有互补性，例如：A 业务的峰值发生在上午 10 点，B 业务峰值发生在凌晨 2 点，这两个业务考虑进行混部，避免了资源的争抢。

e. 应急方案评估

任何一种方案都不能保证万无一失，一定要准备完整的应急方案，例如：紧急扩容方案、故障隔离方案、资源隔离方案等。

（3）动态扩缩容。业务每天或者每个阶段会有在线的波峰和波谷，不同在线规模需要的资源容量不同，可以根据业务在线规模动态扩缩容，通过容器的 HPA 等技术动态调整资源容量，对业务运营成本进行优化。

2. 平台化优化方案

空闲资源调度：对于单业务的空闲资源，SRE 平台部门可以资源整合，组成联邦集群，在业务负载低峰期，通过统一调度，处理分布式的离线任务，提高资源利用率。空闲资源被平台部门使用的同时，可以返还部分运营成本。

容量规划与评估平台：SRE 可以根据业务历史数据，准确评估未来容量需求（例如未来一年周期），通过对未来周期业务容量的准确评估和规划，可以通过批量集中采购的方式获取低价资源。同时，SRE 可以将不同类型业务的资源评估方案沉淀为容量规划与评估平台，当其他业务需要进行容量评估和规划时，可以借助容量规划与评估平台的能力，得出未来周期合理的容量，并合理地批量采购低价资源。

内部资源交易平台：不同业务体量不同，使用的资源类型不同，所获得的折扣券种类、折扣力度、数量都会不同。为充分盘活内部折扣资源，一个组织（例如同一家公司）内的 SRE 资源管理团队，可以开发一个内部折扣资源交易平台。不同业务间可以互相交易资源折扣券，使得一个组织内的折扣券，可以最大程度被利用。

6.5.5 运营成本优化持续运营

1. 运营成本的统计与分析工具

SRE 基于成本运营的统计和分析方法，可以建设运营成本统计和分析的可视化工具。功能涵盖但不局限于以下几个方面：

（1）运营成本的组成，SRE 可以直接看到 IAAS 层、PAAS 层、SAAS 层的运营成本构成及占比等信息，并可以对比不同周期的成本变化。

（2）运营成本的分析，针对财务指标、技术指标等运营成本数据，可以与同业务不同周期，不同运营阶段纵向对比；也可以与同类型不同业务横向对比。

（3）运营成本的优化方案推演，通过调整不同维度、不同指标的运营成本的数据，可以直观观察成本的变化情况，预估成本优化的 KPI 目标。

2. 资源调度工具

SRE 基于空闲资源调度的方法，可以建设资源调度工具。功能涵盖但不局限于以下几个方面：

（1）空闲资源分析，从联邦集群的角度，分析每个小集群（业务）的资源空闲情况，根据空闲 CPU、内存等资源情况，分析可被公共平台调用的空闲资源。

（2）空闲资源调度，根据空闲资源分析得出可以调度的资源，并合理分配可被调度的分布式离线任务。

（3）空闲资源监控，实时监控集群的空闲程度，优先满足业务的正常运行，当离线任务资源与业务进场资源冲突时，优先保障业务资源的使用。

3. 容量评估工具

SRE 基于容量评估与规划的方法，可以建设容量评估工具。功能涵盖但不局限于以下几个方面：

（1）容量的展示，实时展示当前与历史的业务资源容量使用情况。

（2）容量的分析，不同业务类型的容量，不同资源类型的容量，可以与同业务纵向对比；也可以与同类型不同业务横向对比。

（3）容量的预测，基于单业务历史的容量使用数据，和同类型多业务的容量使用数据，智能推荐未来周期的容量数据。为运营成本优化提供合理的建议。

4. 运营成本返点售卖工具

SRE 基于运营成本返点售卖的方法，可以建设成本返点售卖工具。功能涵盖但不局限于以下几个方面：

（1）可售折扣券的展示，实时展示不同业务当前可出售的折扣券的数量与类型等。

（2）折扣券的使用推荐，SRE 输入业务的资源类型和资源容量数据，运营成本返点售卖工具可以自动推荐合理的折扣券使用方案。

网址：SRE-Elite.com 微信：SRE 精英联盟

（3）折扣券的使用数据分析，基于业务使用的折扣券的历史数据，分析得出折扣券的覆盖率和使用率等数据，辅助 SRE 做出运营成本分析的决策。

6.5 持续改进

6.5.1 效率持续改进

效率的改进可以分为两类：

1. SRE 个体工作效率的改进与衡量

一般情况下来说，某项具体工作效率的改进得益于某个具体的人的技术水平或者某项新技术的诞生对工作产生了效率上积极的影响。从持续改进的角度来说，我们需要关注的是：

1）员工个人能力的培养

具有某方面技术优势的员工重点培养，给予更多空间和尝试机会。

2）新技术的关注与应用

鼓励用新技术解决老问题，在安全可靠的范围内测试新技术的稳定性，尝试新技术落地。

对于某项具体 SRE 工作效率改进效果的衡量主要是从时间角度，当然需要考虑“整体”时间这个概念。例如：

某项工作原来的执行时间是 10 分钟，新方法改进后为 8 分钟，效率提升 20%。但是新方法在执行前需要用 3 分钟时间来做准

备工作，整体时间为 11 分钟，实际效率提升为-10%。所以衡量需要考虑整个改进对于原有工作所有的改变和影响。

2. SRE 团队整体工作效率的改进与衡量

团队整体效率的提升需要用更宏观的视角和更长的时间跨度来看待，主要以半年或者一年为周期看待某一类工作是否产生了积极的影响。从持续改进的角度来说，我们需要关注的是：

1）某一类工作的人力投入是否持续降低

对所有的发布变更类工作进行至少半年以上的人力投入统计，评估人力成本投入是上升趋势还是下降趋势，并分析原因。呈上升趋势的，找到人力投入持续上升的原因，并及时解决。呈下降趋势的，找到是因为做了什么优化而导致的人力投入下降，继续强化去做。

2）在团队整体配备保持不变的情况下，是否能够承接更多或更大规模的服务

评估整个团队解决目前所有的工作所使用的解决方案有哪些，是否具有通用性。团队输出通用性方案的能力越强，这个团队保持配备不变的情况下，接手更多更大服务的能力越强，团队整体服务能力越强。

6.5.2 质量持续改进

质量的持续改进，一般是指 SRE 团队所提供的服务质量，这部分工作的持续改进大致可以分成两部分：

1. 服务质量的衡量

衡量服务质量的方法很多，例如：故障分级制度、满意度调查、平均无故障时间、SLO 等。

故障分级和满意度调查，可以了解当前服务质量状况。

平均无故障时间或平均故障间隔时间（MTBF），可用于持续地改进衡量标准。因为我们可以希望这个时间尽可能地长，说明我们的服务一直处于无故障状态，即服务质量一直达标。

在平均无故障时间（MTBF）的概念下，还可以进一步细计算：平均故障恢复时间（MTTR）。

平均故障恢复时间（MTTR）又可以继续被细分为：

MTTI，平均故障发现时间

MTTK，平均故障定位时间

MTTF，平均故障解决时间

MTTV，平均故障修复验证时间

以上都可以作为非常细致的服务质量持续改进的衡量指标。

这种衡量方法，不足的地方在于，他可以衡量那种引发了故障的情况非常好用，清晰且可衡量。但是，对于没有引发故障，但是用户的实际体验却实实在在受到影响的情况就不好衡量了，这时候我们可以用到 SLO 这个指标。

SLO 可以根据服务不同等级相对稳定在某个值，我们需要关注的是 SLO 有没有下降趋势。如果有下降趋势，那么具体是哪些微小的因素影响了用户体验？对于提升用户体验方面可以参考 3.6.1。

2. 服务质量的复盘

服务质量的复盘，包含对优秀案例的复盘和故障案例的复盘。

优秀案例的复盘，重点关注：

我们具体做了哪些工作导致服务质量提升？

为什么这样做可以提升服务质量？

我们如何把这种方法复制到其他服务质量的提升？

故障案例的复盘，请参考 3.5.4。

6.5.3 安全持续改进

一次安全事件的发生很可能将其他的所有工作都归零，安全的持续改进，也可以理解为是安全预防工作的持续改进，如何持续地预防安全事件的发生。

安全的持续改进可以从如下几个方面着手：

1. 默认安全规则

默认安全规则，即要求 SRE 团队在所有服务上需要做到的最低安全规范。

例如：禁止 SSH 服务对公网开放、在服务器上开放类似 VPN 的功能、安装未经安全评估的开源软件等。

2. 安全意识

所有人员应当定期参加安全培训，提升安全意识。不仅仅是 SRE 团队，更应该包含产品运营团队和开发团队，落实是“人人都是安全责任人”的概念。

3. 研发期安全预防

践行 DevSecOps 理念，把安全能力无缝且柔和地嵌入到研发过程，在产品研发过程中加强安全能力的建设，预防安全事件发生。例如：定时向研发团队强调默认安全规则，介绍经过评审的开源软件，推送具有安全风险的开源软件，介绍安全的系统架构设计原则等等。

4. 运营期安全预防

运营期的安全预防机制，是常态化进行巡检，输出安全报告或安全工单。例如：高危端口扫描、高危服务扫描、入侵检测系统接入扫描、具有安全漏洞的开源软件扫描、密码泄露扫描、内外部代码扫描、安全补丁未更新扫描等。

5. 评审机制

定期组织安全能力评审，对新上线系统以及在运营系统进行安全评估，制定整改计划。安全评审，要确保评审团队与 SRE 团队之间的轮换，避免长期固定的评审关系。

6.5.4 人员能力持续提升

人员能力持续提升，其实就是为了让团队的技术能力能够持续健康发展，这需要有一个可持续运转的机制来保障。类似这样的方法有很多，没有什么标准做法，下面介绍一下我们内部的方式。

1. 海量运维能力机考

每年进行一次海量运维能力上机考试，题目包含：Linux 基础/Window 基础，海量运维脚本开发、故障排除、云原生基础、安全

常识等，可以一定程度上帮助大家看到自己的水平变化和技能的熟练度。

2.技术水平能力积分

SRE 岗位所需要具备的海量运维、运维开发、DevOps、AIOps 等各项能力所对应的系统、工具的使用和操作记录、贡献记录、开发记录、代码质量、维护记录、创建记录等进行积分规则制定，根据这些规则为每一个 SRE 个人和 SRE 团队计算积分。这个积分从一个侧面体现其 SRE 个人和 SRE 团队的能力，用于衡量个人和团队能力提升的参考。

这个积分规则的合理性决定了这个制度是否能够执行下去，所以，这个规则需要能够客观地展示个人和团队能力，并且起到带动 SRE 团队发展作用，这是这个规则制定的关键。

例如：

某个 SRE 开发的一款 SaaS，不仅能够帮助其他的 SRE 完成日常工作，而且产品开发、产品运营都经常使用这个 SaaS，并且允许稳定，代码质量高，BUG 少。那么这个 SaaS 的开发、维护、代码质量等各项工作的积分就算在这个 SRE 的运维开发的能力项下，可以从一个侧面反映这个 SRE 的运维开发能力。

再例如：

当我们衡量云原生能力时，那么所有在 K8S、容器平台上的贡献纪录将被用于衡量每一个 SRE 在云原生方面的能力。如果我们要

鼓励大家更多地学习和应用云原生技术时，相对应的这部分积分的权重可以在一定的周期内调高，以鼓励现有团队进行转型。

有了这样的体系，整个团队的人员能力有了一个大致的数据衡量方法，可以客观地看到各个团队在不同维度上的能力强弱，从而人员能力提升有了动力。

6.5.5 流程持续改进

SRE 团队各个环节的工作其实都有专门的持续改进措施，以保证能够持续优化。例如：故障复盘、混沌工程等等。作为 SRE 团队的整体发展，其实还需要一个整体的流程以保证整个 SRE 工作是可以持续改进的。接下来就介绍一下这个整体能够保持持续改进的方法。

这套流程方法大致分为如下几个步骤：

1. 建立上线评估标准

SRE 团队需要出具一个上线评估标准，此标准用于评估一个新的服务是否满足交付给 SRE 团队持续的维护。

这个标准可能需要保护如下内容：

架构评估

合规评估

安全评估

其他评估

通过以上各项评估初步了解即将接手的服务特点，以便 SRE 团队制定维护目标和计划。

2. 确定 SLI/SLO

现在，可以根据现有的信息来初步制定维护目标即 SLI/SLO。

3. 运行分析

这个阶段对即将要接手的服务进行运行分析，记录系统运行的关键数据和指标，验证上面几个步骤是否还需要重新评估或修改目标，也需要验证 SRE 团队的维护方案是否可行。研发团队必须在这个阶段给予 SRE 后备支援与各项建议。这种关系成为团队未来工作的基础。

4. 改进与重构

上一个步骤的运行分析，在这个步骤需要输出改进和重构的建议，提交研发团队修改，SRE 团队也改善当前的维护方案。

5. 综合培训

这个步骤是确保整个团队都有所准备，不仅仅是 SRE 团队，也包含开发团队、运营团队、产品团队等，需要全面地让所有人都知道上线后，我们所需要遵守的规范和长期运行的目标。

这个培训应该包含：

项目整体概况

安全规范

该项目的 CI/CD/CO 的标准流程与协作方式（版本交付、测试、发布等）

线上系统的部署模式

故障处理与应急流程

其他需要整个项目团队知晓的内容

6. 正式服务

此时，SRE 正式开始接手这项服务，从这个时候开始各项运维职责、权限都转交 SRE 团队，包括运维操作、发布变更、访问控制等等。研发团队持续给予 SRE 后备支援与各项建议，并与 SRE 团队长期保持良好的协作关系。

7. 持续改进

活跃的服务根据运营需求的变化，不断变化系统依赖，技术升级和其他变化。例如：新增用户需求等。SRE 团队在面对这种持续改动的同时维护服务的可靠性。当面对每一个新的变化的时候，需要根据当时的具体情况，重复执行上述的 7 个步骤以保证 SRE 团队的工作能够在良性的循环中持续改进。

7 平台工程

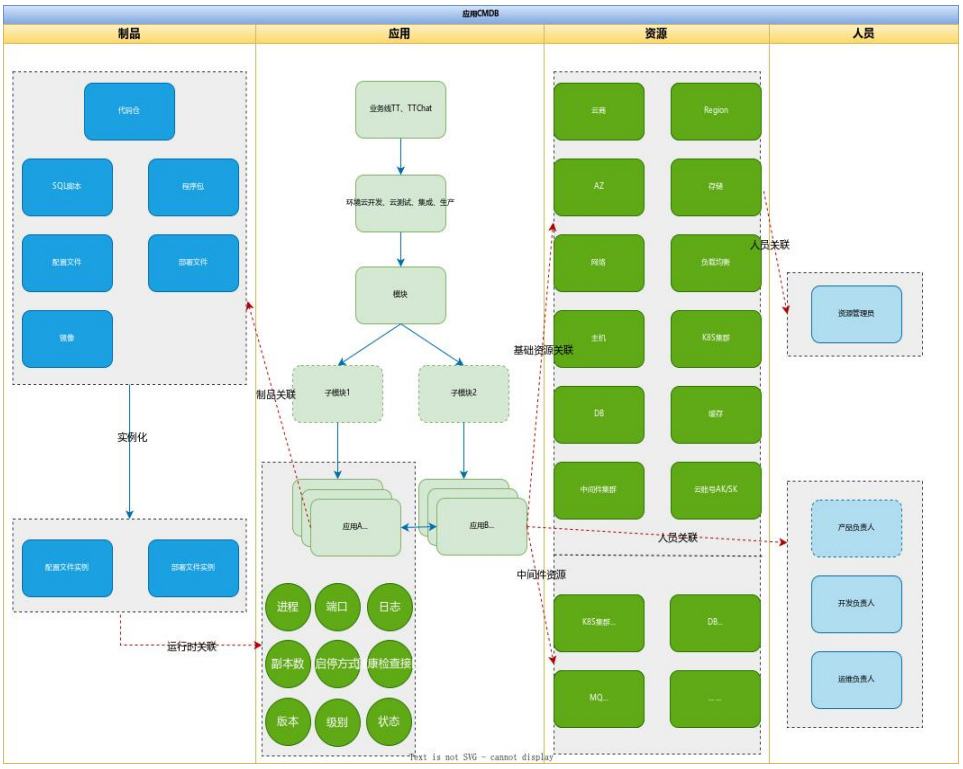
7.1 标准应用平台工程建设

在当今多变的技术环境中，构建一个既能满足当前需求又能预见未来扩展的应用平台，对于任何规模的组织来说都是一个挑战。标准应用平台工程的建设不仅需要考虑到技术的先进性和可靠性，还要兼顾操作的简便性和整体成本的经济性。因此，我们的目标是通过标准化、模块化和自动化的方法，构建一个能够支持快速迭

代、持续集成和高效运维的标准应用平台，以适应不断变化的业务需求和技术创新。

在本章节中，我们将详细探讨构建标准应用平台工程的各个关键组成部分，包括应用元信息平台的设计与实现，统一资源供给的策略和方法，以及扩展模块的集成和管理。下面将会讲述如何将这些组件融合成一个协同工作的有机体，确保整个平台的稳定性和可扩展性，同时提供清晰的管理界面和后端支持。

7.1.1 应用元信息平台



应用元信息平台旨在为整个标准应用平台工程的建设提供数据底座基础。一切的技术建设最后的核心关注对象都是应用本身，因此应用本身的各类信息、属性都需要被统一管理，诸如应用的唯一标识、应用归属（组织/业务/财务等）、应用属性（名称/应用类型/

等级/保障机制等）、运行时信息（端口/健康检查等）、人员角色信息（负责人/研发/SRE/测试等）、应用关联资源（容器/主机/数据库/缓存等）。

1.应用概念描述

（1）APPID：APPID 组成结构：<业务域>.<业务>.<应用>。各组成属性的命名，存在约束条件，仅可使用小写字母和减号-

（2）业务域：业务域是指由一系列联系较为紧密的业务的集合，是业务对象高度概括的概念层次归类，目的是便于业务的管理和应用。

（3）业务：一些能够为公司（或组织）达成某个目标或产生价值而进行的一系列活动。特定业务的业务能力取决于这个业务的类型。例如，保险公司业务能力通常包括承保、理赔管理、账务和合规等。在线商店的业务能力包括：订单管理、库存管理和发货

（4）应用（服务）：在既有业务能力范围内，可以为每个能力或相关能力组定义应用。应用是企业业务组成的最小单元。

（5）组织：一个组织整体的结构，是在企业管理要求、管控定位、管理模式及业务特征等多因素影响下，在企业内部组织资源、搭建流程、开展业务、落实管理的基本要素。

（6）产品：财务提供的拆账粒度，大于业务，小于组织

（7）应用场景：内部各个应用、运营系统通过 APPID 来进行应用的唯一定位，通过 APPID 进行相关资源的关联。

（8）环境：资源所处的环境，如 PROD、PRE、UAT、FAT

（9）资源（资源类型）：维持一个应用在各生命周期所需的资源，如容器、数据库、缓存、负载均衡、物理机等

2.业务价值

1）平台方

（1）资源提供方作为租户接入，并注册需要关联到应用的资源信息；资源挂树/回收时，服务树会校验该租户是否有资源的管理权限，保证租户间资源隔离。

（2）商品化资源可以定时向资源运营平台推送账单，资源运营平台也会检查该租户的身份（非该租户的资源不能接受此租户推送的账单）；最终会按照财务层级聚合账单推送给业务方。

（3）租户可以基于平台来构建资源管理模型，使得自己的资源管理更贴合业务，租户可以将 Quota、预算、资源利用率、SLO 等基于平台来构建，通过这些措施，租户可以更了解业务方的资源使用情况， 实现更合理的资源调配

2）业务方

（1）贴合业务线：相比通过变动频繁的组织架构，贴合业务线的服务树更适合用于各团队在产品/项目工作中进行资源归属的管理。（业务资源的盘点可在应用元信息平台进行）

（2）统一元信息：对于业务团队、计费中心、安全合规，有统一的资源 meta 信息，避免信息不对称并提升协同效率。

（3）批量管理：资源挂树后，针对层级的单/多类资源的批量管理，包括新建、删除、配置、迁移等，就有了发挥的空间。

（4）统计资源利用率：以帮助业务方提升资源效率，优化成本。

（5）统计 SLO 信息：以面向用户落实稳定性保障的成果。

（6）面向业务的权限管控：有了层级资源模型，结合服务树的鉴权模型，就可基于节点的人员信息进行权限管控，服务树基于 IAM 实现了一套灵活的认证鉴权机制。

7.1.2 统一资源供给

1. 应用概念描述

资源：资源是一个抽象的概念，由各资源平台决定资源的最小粒度。例如：容器平台中的一个应用、数据库中的数据库实例、对象存储中的 Bucket、一台物理机、负载均衡中的一个负载实例、一个业务域名，视频直播中的一路接入等，这些都是各云计算资源平台提供的一种资源。

2. 统一资源供给的价值

面向用户侧：平台工程的目标是通过对云资源进行抽象，以产品化思维，构建面向研发人员的自动化平台能力。通过向上屏蔽底层的复杂度，让研发人员能够不用太关注底层的复杂能力。进阶的平台工程期望能够实现底层资源的联动，比如操作负载均衡，向上联动域名，向下联动 RS 等能力，为开发者提供全套的云计算解决方案。

面向云平台内部：平台工程是很好的落地标准化的手段，通过间接约束研发人员的使用，逐步摒弃个性化的配置，使研发团队使

用云资源更趋同和标准，大大减少云平台内部 SRE 团队的维护复杂度。

因为云资源的业务复杂性，甚至维护管理云资源的组织复杂性问题，很容易导致面向体系内用户使用的时候各产品/服务相互独立，交互体验、操作流程和概念标准不一致。

平台工程可以通过资源供给侧前端 UI 的标准化和流程的统一，确保用户在各个产品页面都能获得一致的用户使用体验，减少使用云能力的复杂度，可以拉通各个云资源标准化区域、可用区、环境和项目分级等，拉齐用户对多个产品的认知与共识，同时为未来的上层建设提供基础。

3. 统一资源供给的前提

（1）具备一个公司通用的资源树（服务树/IAM），通过业务/项目/服务将资源进行挂载。

（2）在业务/项目/服务这些维度维护关键角色，诸如业务负责人，成本接口人，SRE 负责人，并与云资源提供方和用户方尽量拉齐一套申请资源和审批资源的流程。

（3）具备灵活的 BPM/任务流/工单体系

（4）与各类资源提供方达成一致，用户对资源的使用统一收口在平台中，资源提供方尽量提供视角一致的资源增删改查接口，理想情况下平台工程团队对所有资源接口再做一轮标准化封装。

7.1.3 持续集成

CI 平台帮助开发者自动化构建-测试-发布工作流，持续、快速、高质量地交付产品，通过屏蔽掉所有研发流程中的繁琐环节，让工程师聚焦于编码。提供流水线、代码检查、代码库、凭证管理、制品库等核心服务，用于满足企业不同场景的需求。

1. 应用概念描述

流水线：CI 最核心的服务，将团队现有的研发流程以可视化方式呈现出来，编译、测试、部署，一条流水线搞定；

代码检查：提供专业的代码检查解决方案，检查缺陷、安全漏洞、规范等多种维度代码问题，为产品质量保驾护航。

代码库：支持如 GitLab 与 Gerrit 仓库代码源。每个应用下的流水线都会有默认的代码仓库，流水线运行时会拉取指定分支代码运行。开启子仓库、设置关联应用、多仓库构建时，流水线中运行的代码将会是多个仓库打包后的代码。

凭证管理：为代码库、流水线等服务提供不同类型的凭据、证书管理功能

制品库：基于分布式存储，可无限扩展，数据持久化使用对象存储。

触发机制：手工：在流水线列表后方点击执行；定时：设定时间自动执行流水线；自动：监听代码 PUSH 或者 MERGE 操作，执行流水线；代码评审触发：在代码管理评审设置中配置评审时触发。主要用于预合并验证代码。

流转方式：

未完成流转：执行了就会开始到下一任务（不关注任务结束时间、不关注任务结果成功或失败）

完成流转：任务执行完成开始下一接任务（需等待任务执行结束，不关注任务结果成功或失败）

完成且成功流转：执行完成且任务执行成功才开始下一任务（需等待任务执行结束，需要任务结果为成功），如果任务不支持配置流转方式时，则默认使用完成且成功流转

2. 核心功能

流水线主要包含【基本信息】、【流程配置】、【变量配置】、【安全管控】四个部分，本文主要介绍基本信息和流程配置部分：

3. 基本信息

一条流水线的基本信息包含以下配置项：

流水线名称：必填，可以重复。

代码仓库：根据应用设置中的代码仓库自动填充。

默认关联分支：用户手动执行流水线时，默认运行的分支。只能关联一个分支

触发方式：手工、定时、自动。详情参考 触发方式

开启集成模式：详情参考 分支集成

默认流水线：一个应用下只能标记一条默认流水线，标记后，生成版本时默认会选择该流水线执行。

4. 流程配置

按照各个项目的研发模式和业务诉求配置流水线阶段与任务。一个流水线中至少配置一个任务才允许保存。

阶段：通常我们按照开发流程将流水线配置为不同阶段。如构建阶段、并行测试阶段、集成测试阶段、发布上线阶段。阶段可以配置串行或并行。

任务：一个阶段中可以添加多个需要在这个阶段中执行的任务，一个阶段内的任务可以全部选择【串行】或者【并行】运行。目前支持的任务参考流水线任务与仓库支持

（1）添加串行阶段。点击流水线【添加阶段】，或者流水线之间的【+】，可以添加一个串行阶段。

（2）添加并行阶段。将鼠标移动到一个阶段下方的空白区域，会自动出现【添加阶段】按钮，这个时候可以在这个阶段下添加并行阶段。

（3）添加阶段时，可以设置阶段内任务的串行或者并行。设置后，属于这个阶段的任务将按照串行/并行执行。

（4）添加任务。可以点击阶段中的【+】来添加任务。

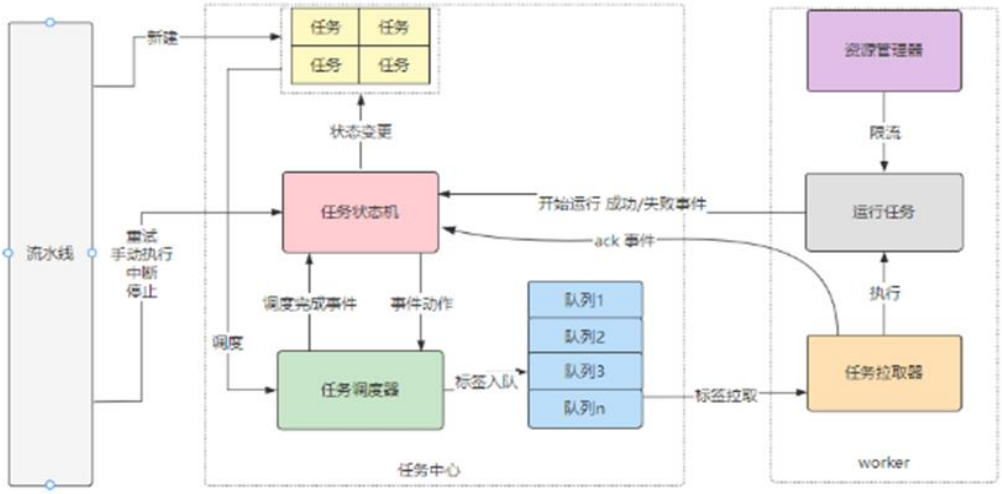
（5）任务添加默认都是阶段内最后一个，你可以拖动任务来改变任务顺序。

5. 流水线设计简介

流水线作为持续集成的核心组件，是否健壮、易用直接影响着持续集成的建设水平。在交付高峰期服务不可用现象频出，严重影

响着业务交付的顺畅度。同时业务方的构建资源没有做限制，部分业务方大量占用资源，影响平台的其他用户正常使用。

随着用户量、构建场景的增多上述的问题也会愈发严重，对于流水线的场景，资源最优解并不是强诉求，可通过任务拉取的方式，控制任务执行的频率，从而降低了资源的竞争和系统的可用性



（1）增加任务状态机，管理任务状态流转

（2）增加任务调度器，调度任务进入队列等待

（3）增加任务队列，从原来的主动选择 node 节点修改为现在的 node 节点主动拉取

（4）增加资源管理器，管理当前 worker 节点的资源信息

7.1.4 持续部署

持续部署将用户的业务实例组织起来，打通下游持续集成，并对接容器、虚拟机、物理机等物理资源，并为上游服务提供部署底座，为用户的业务提供部署的基础服务。

1. 应用概念描述

1）实例

用户用于部署的最小部署单元，可以是单个独立运行的 K8s pod，一个 statefulsets 生成的 pod，也可以是云平台虚拟机、物理机等。

2）分组

将多个实例组织起来的逻辑概念，用于支撑业务应用的并发或多活，同时负责为这一批实例提供统一的通用化配置选项等功能。

3）包部署

包部署是分组的一种类型，其实例可以是单独的 K8s pod、云平台虚拟机或物理机，在使用方面通常用于提供有状态应用服务。

4）镜像部署

镜像部署是分组的另一种类型，其实例由自研的 K8s statefulsets crd 生成的 pod 组成，通常用于提供无状态应用服务。

2. 核心功能

1）部署设置

针对实例的定制化配置选项集合，定义了实例对外提供服务的基础能力。

基础信息，例如应用根目录、运行用户、日志目录等。

实例启停信息，例如资源池、JDK 环境（针对 Java 应用）、初始化及启停脚本等。

实例高级配置，例如实例对外服务端口、配置下载目录、分散策略等。

实例健康检查信息，例如 HTTP/TCP/脚本检查方式、实例健康检查端口/周期/延迟/失败阈值等，以及健康检查时的告警策略及自动恢复策略等。

2）分组管理

与部署设置管理，对同样功能的实例进行统一化配置，主要包含：

基础信息，例如分组的类型（包部署/镜像部署）、关联的部署设置、是否灰度分组、注册中心及网关流量标签等。

规格信息，例如分组内的实例数，规格（CPU 核数、内存大小、GPU 型号及规格），磁盘大小等。

高级设置，例如实例是否保留固定 IP，接入配置中心的环境及版本，容器标签、注解、宿主机亲和性等。

3）部署变更

从持续集成制品库获取应用版本后，对实例进行部署变更，可以定义当次变更的以下内容：

单个实例部署的超时时间。

分组间并发度及分组内并发度。

部署维度，定义是按分组维度来部署还是部署分组内的部分实例。

3. 业务价值

1）部署能力

通过自研基于 dumb-int 的基础镜像，结合镜像仓库及持续集成等制品存储，持续部署实现了业务应用部署的全生命周期管理。

2）自定义部署设置

基于大量丰富的部署设置实现了无状态及有状态两套业务容器方案，充分满足不同的业务需求。其中无状态容器通过自研 K8s statefulsets crd 实现，有状态容器通过 K8s 原生 pod 及自研 agent 实现。

3）GPU 资源利用

除了基础资源，持续部署还与 GPU 资源深度结合，允许用户创建带 GPU 资源的容器。

4）健康检查及告警

通过 K8s 探针及 agent 建立容器的生命周期管理机制，同时实现了自动告警、自动拉起等预警保活机制。

7.1.5 部署编排

在持续部署的基础上，部署编排将跨应用的发布进一步组织起来，允许用户通过一个编排计划发布多个业务应用、降低发布复杂度，以及提升发布效率。

1. 应用概念描述

1）编排计划

用于将跨应用的多个分组组织到一起的逻辑概念，由一个个编排任务组成，同时可以控制这些任务的顺序及串并行。

2）编排任务

组成编排计划的最小单元，涵盖多个领域的不同任务类型，在编排计划中用于不同的分工。

3）监控模板

通过打通云监控，实现了多个指标的对接，同时允许用户设置监控超阈时的动作用以及时止损。可直接与编排计划关联，用于监控整个编排计划的发布过程，也可在部署计划中自由添加并指定监控时长，用于单点验证。

4）通知模板

通知模板用于在编排计划的生命周期内，对不同的时机发送不同的用户通知。

2. 核心功能

1）编排实例部署

部署编排对实例部署的编排主要通过以下几个方面来实现：

发布顺序及串并行控制：用户可以将若干应用的多个分组编排进一个编排计划内，将其以用户自定义的串并行方式进行部署。

精细化暂停点管理：为用户提供丰富的实例暂停点控制设置：允许用户按实例、实例比例、全部实例等规则选择暂停点；允许用户自定义暂停点的手自动通过方式及卡点时间。

实例分批次发布：提供了实例分批次发布的能力，允许用户将单个分组内的实例划分为任意个数的批次，并拆分成不同的编排部署任务；允许用户在同一个分组的不同实例批次之间任意插入需要的其他任务。

2）编排任务

编排任务主要分为三类：

核心任务：部署、卡点、监控。

其他任务：覆盖了诸如自动化测试、扩缩容、实例批量操作、静态部署、自定义脚本等与发布相关的各个方面。

定制化任务：用户可以自己定义任务的内容，以插件的形式接入编排。

3）智能监控

通过与云监控的紧密结合，我们为用户提供了不同的指标、告警及使用方式：在指标及告警方面，当前已经对接了业务黄金指标、自定义指标监控（日志监控、自定义监控、指定时序数据库等）告警、调用链指标及量化发布指标（自定义指标及调用链接口）。

在这些监控能力的使用方面，我们提供了部署计划级监控及监控任务，既允许用户在整个发布过程中进行监控，同时也允许用户在某个指定时间段内进行监控。

4）通知

在部署编排执行的过程中，虽然自动化程度较高，但有些情况仍需用户关注，因此我们允许编排计划及编排任务关联通知模板。在通知模板中，用户可设置以下内容：

通知时机：分为计划级别的时机和任务级别的时机，涵盖了计划及任务的生命周期。，时机包含编排计划的开始、成功、取消、

超时，编排任务的开始、成功、跳过、超时、失败等；接收人可以指定角色，也可以指定具体的员工工号。

接收人：允许用户灵活配置通知接收人的服务树角色或指定的接收人工号。

通知渠道：除直接的 TT 消息外，允许用户添加告警机器人，在通知时自动发送 TT 群内消息。

3. 业务价值

1）发布提质提效

通过编排计划将流程固化，以及将多个应用的发布流程串联，最大化地降低了发布过程中人为操作的出错以及跨应用之间的沟通延时及误差，从而降低了故障率及提升了发布质量。

同时，由于部署编排对接了测试平台及云监控，实现了发布自动验收及发布故障自动止损，进一步提升了发布质量及发布效率。

2）精细化实例管理

除了允许用户为实例逐个设置暂停点外，部署编排还允许用户以下几个方式来管理暂停点：

按一定规则为整个分组的实例设定暂停点，例如全部实例、实例百分比、实例下标等。

为不同的实例定制暂停点手自动通过方式及卡点时间，从而便于用户更精细化地管理。

而对于实例本身的管理，部署编排还允许用户对单个分组的实例实行分批次管理，在批次之间可以任意插入编排任务，从而更好地管控实例发布。

7.1.6 可观测

为了确保我们的应用平台的高效运行，我们需要构建一个全面的可观测性框架。这一框架应包括以下几个关键组件：

（1）日志管理：日志是可观测性的基础。我们将实现一个集中式日志管理系统，该系统能够收集、存储和查询来自平台上所有服务和应用的日志数据。此外，我们将采用标准化的日志格式和丰富的元数据，以便于搜索和分析。

（2）监控与警报：监控系统将实时收集关于系统性能的指标，如响应时间、吞吐量和错误率。当这些指标超出预定的阈值时，警报系统将通知运维团队，以便他们能够迅速采取行动。

（3）分布式跟踪：在微服务架构中，一个用户请求可能跨多个服务。分布式跟踪系统能够追踪请求的整个生命周期，帮助我们识别性能瓶颈和故障点。

（4）服务健康检查：健康检查机制将定期检查服务状态，确保服务的可用性和可靠性。这包括从简单的 HTTP 检查到复杂的业务逻辑验证。

（5）性能分析：我们将部署性能分析工具来收集关键应用和服务的性能数据。通过分析这些数据，我们可以优化系统配置，提升资源利用率和响应速度。

（6）可视化仪表盘：所有的监控数据和分析结果都将通过一个可视化仪表盘展示，这使得任何非技术人员也能轻松理解系统的当前状态和性能。

（7）事件管理和自动化处理：通过事件管理系统，我们可以跟踪和响应系统中发生的事件。结合自动化工具，我们可以实现对某些类型事件的自动化处理，减少对人工干预的依赖。

通过上述组件的整合，我们将能够实现对整个应用平台的深入洞察。这不仅仅是为了解决问题，更重要的是预防问题的发生。我们将能够实时监控应用的性能，预测潜在的系统瓶颈，并在问题影响用户之前主动地解决它们。

7.1.7 成本（定价、用量、出账）

1. 资源的计量计费：

资源的计量计费是企业内部平台管理中至关重要的一环。随着资源使用频率和范围的提升，业务部门渴望了解资源的使用情况，而成本作为一个统一的度量，能够为业务方提供量化的使用感知。因此，资源的计量计费成为必不可少的手段。为实现企业内部平台与公有云一样的计量、计价和出账能力，需要设计可监测且可计量的平台能力项，并确定其计价策略，监控统计各业务的服务用量，最终通过量价输出各业务的成本账单。

2. 计量

每个平台能力项对业务方提供的服务不一样，因此需要制定特有的计量指标，可参考以下通用计量模型设计流程：

（1）定义平台能力项：在各公共平台部门中，根据服务功能的不同，制定面向使用方角度的平台能力项，作为计量的承载主体。

（2）业务对象标签：定义平台所服务的业务方结算主体，主要目的是建立财务与技术之间统一的成本主体概念，可以直接引用应用元信息的业务概念或者以应用元信息为基础制定归集映射策略，与财务结算单元对齐。

（3）成本驱动因素：针对平台服务功能的不同，每个平台定义相应的计量指标（根据平台的实际情况，可设计一个或多个计量指标）。计量指标应该是面向业务方角度，对业务方有感知的指标，例如任务调用量、审核量、数据存储量等。

数据采集与统计：平台能力项需搭建成本驱动因素的日常数据采集能力，为计量提供基础数据信息，至少实现 T+1 按天维度的数据采集粒度，能细化实现小时或者分钟采集粒度更优，计量应该以业务对象标签维度进行统计。

3. 定价

平台能力项的服务定价，也就是成本驱动因素的平均单价，参照以下公式：

单价=平台能力项资源 TCO/成本驱动因素最大月取值

- 平台能力项资源 TCO：包括该平台能力项所直接使用的资源成本总和；

- 成本驱动因素最大月取值：按照一定月份周期内，平台能力项统计到的每月成本驱动因素数量，取最大月份的数量；

4．出账

确定单价之后，业务使用方对成本就有了稳定的预期。账单可以根据计量的颗粒度进行输出，为便于技术方和业务方及时了解成本变动情况，至少实现 T+1 按天维度的成本明细账单，能细化实现小时或者分钟成本明细粒度更优。

通过公式可以输出费用账单：成本=单价*用量

1）技术方：对单价的负责，是平台能力项的成本责任方，通过对底层资源的合理选型、技术架构的优化、资源利用率的提升和资源管控的强化，推动平台能力项单价的优化，将优化成果转化为成本收益，提升平台的成本竞争力。

2）业务方：对用量的负责，通过管理和优化业务应用实例的数量、存储量、生命周期、资源占用方式（共享/独占）、调用策略，降低对平台能力项的使用量，从而节约业务成本。

示例说明：

以 K8S 容器计算平台作为例子，按照每个集群进行定价，建立每个集群的成本驱动因素监控和统计，定期输出 K8S 算力成本账单。

平台能力项：K8S 容器计算服务，不同集群由于机型和地区不一样，采用不同单价计费

业务对象标签：业务 A，业务 B，业务 C ...（通过应用元数据一级模块归集映射，与财务结算单位对应）

成本驱动因素：考虑人力及时间投入，建立单一的综合指标-算力核时，包含 CPU 和内存用量按照一定比例系数折算的用量之和，比例系统可以参考行业或按照云厂商主机的 CPU 和内存成本比例统计。

监控和统计：T+1 天输出

平台能力项资源 TCO：以 K 集群为范围计算 TCO，K 集群 TCO=云主机成本+磁盘成本+日志服务成本+网络成本

成本驱动因素最大月取值：以近半年为周期统计，K 集群最大月取值为 5000 核时

K 集群单价=K 集群 TCO/5000 核时=xx 元/核时

账单：T+1 天输出，如业务 A 一天使用了 1000 核时，则业务 A 算力成本=1000*K 集群单价；如业务 B 一天使用了 1500 核时，则业务 B 算力成本=1500*K 集群单价

7.2 异构应用平台工程建设

异构应用（Heterogeneous Application）是指由多种不同技术、架构、编程语言或平台构建的应用程序。在异构应用中，各个组件可能运行在不同的操作系统上，使用不同的数据库系统，或者由不同

的编程语言编写，由此而带来了一系列的运维复杂度，如集成和管理的复杂性、技术栈的多样性导致的维护成本等。

面向异构应用，平台工程建设需要考虑不同应用系统和平台之间的异构性，包括不同的数据格式、协议、接口等，通过技术手段实现数据的转换、集成和共享，通过管理手段，包括对异构系统的统一管理、安全管理、维护管理等。

7.2.1 总体设计



如图所示，面向异构应用的平台工程的总体结构可划分为 3 层，自下而上依次是：原子平台层、PaaS 层和 SaaS 层，其中 PaaS 层包括 aPaaS 和 iPaaS 两大核心能力，SaaS 根据场景分为一级 SaaS 和二级 SaaS。除此之外，平台工程本身也需要提供平台服务管理和安全审计的能力。

7.2.2 aPaaS 结构设计

aPaaS，全称是 Application Platform as a Service，即应用程序平台即服务。与传统的 PaaS（Platform as a Service）平台不同，aPaaS 更加注重应用程序的开发和部署，而不是基础设施和资源的管理。

aPaaS 平台提供了一系列的工具和服务，包括应用程序开发、测试、部署、托管、监控和管理等。开发人员可以使用这些工具和服务来快速构建、测试和部署应用程序，而无需关注底层的基础设施和资源管理。

aPaaS 主要提供 2 个能力，面向开发者的工具和面向应用的运行环境托管服务。

1. 开发者工具

1）统一的后台开发框架

一套完善的前后台开发框架可以帮助开发人员更快地开发应用程序，并且可以简化代码的编写和维护。

（1）分析并确定框架功能：需要提供统一登录、身份认证、安全防护等基础的功能，提供消息通知、日志记录、访问统计等通用的组件服务，提供多种数据库访问方案；提供基本的预处理函数或公共组件使用的代码编写样例等。

（2）选择框架的技术语言：针对不同的编程语言，需提供相应的开发框架。目前，主流编程语言包括 Python、Golang、Java 等。应依据平台工程开发者的技能特点，提供所需之开发框架。

（3）研发前制定编码规范：为了保证代码质量和一致性，除了最基础的命名规范、注释规范、代码格式外，还需要包括平台级的公共变量规范、系统变量规范、错误码规范、日志输出规范等，并通过代码审查等手段确保开发者遵循规范。

2）通用的前端组件库

一组根据业务场景和产品特性预先定义好的前端组件和代码库，在进行原子平台开发/SaaS 开发的过程中，可以帮助开发人员快速地构建出高效、可维护的产品页面。

（1）制定设计规范：由产品经理和设计师主导，根据行业标准和业务场景，制定前端组件共同遵守的设计规范，确定最基本的设计元素，包括颜色、字体、图标、网格和布局等。

（2）交互设计：基于基本元素，设计一套常用的 UI 组件和交互模式，如按钮、表单、导航、对话框、搜索等。保证各产品间一致的视觉交互体验，延续相同的用户习惯，减少认知上的理解差距，减少无意义的探索和重复设计。

（3）统一组件实现：组件包含了通用的场景和产品的经典交互逻辑有前端组件，如横向导航必要字段与交互方式、页脚 footer 的通用内容与交互、项目空间选择器、无权限通用页面、版本日志展示框等，更能提升研发效率及平台的一致性。

（4）持续更新前端技术语言：一些著名的通用的前端组件库包括 React、Vue 和 Angular 等。这些组件的技术语言都在不断发展，选择了一个语言后，对应的前端组件库要不断更新。

3）可视化开发（LessCode or LowCode）

可视化开发（LessCode or LowCode）：是一种可视化编程语言，用于创建可拖放、可配置的界面和交互式应用。

（1）分析并确定产品功能：需要提供的基础功能有可视化代码编辑器、拖拽式的前端页面生成方案，还需要提供在线调试和预览能力，与 PaaS 的应用托管能力打通，支持一键部署并发布应用。

（2）前后台协作模式：这种模式需要预先定义好前台和后台的数据和协议规范，开发者先以前端的身份在可视化开发平台生成需要的产品 Web 页面，再以后台开发者的身份编写业务逻辑、处理数据存储等。

（3）可扩展的模板市场：提供常用的前端交互逻辑函数，这些函数可以统一使用 JS 语法编写代码，通过发起 Ajax 请求，获取接口数据，转发后台配置，绑定组件事件，配合页面生命周期等，实现不同的功能。

（4）可扩展的模板市场：提供官网类、后台管理类、公告类等应用级模板，包含应用完整的功能，如：函数，变量，数据库等，可直接基于应用模板创建新应用，无需从 0 到 1 进行组装，只需要将模板对应内容进行修改即可快速完成应用的开发。

4）增强服务（如 AIDev）

在提供了基础的开发者工具后，可以结合当下先进的技术，降低工具的使用门槛，提高工具效率。

（1）AIDev：通常指的是人工智能（AI）开发（Dev），通过这个技术可以与现有产品的资料查询、技术支持、代码生成等场景结合，进一步降低工具的使用门槛。

（2）微前端：一种前端和后端协作开发的模式，可以在大型的、复杂的 SaaS 研发中使用该技术方案，将组件拆分成多个小型的、独立的服务，通过 API 进行通信和协作。只要提前预先定义每个组件都要遵循的一套标准和规范，就可以将这种模式应用到 Web 应用、移动应用、桌面应用等。

2. 运行时环境托管服务

1）K8s 容器托管

Kubernetes（通常简称为 K8s）是一种开源的容器编排和管理平台，它本身特性有自动化容器部署、升级、回滚、监控、扩展和负载均衡等。SRE 更侧重于建设产品化 Web 页面，让开发者无需掌握专业的 K8s 技能，无需关注基础设施细节，也能使用容器托管能力。

（1）规范用户操作路径：将代码库、开发框架、可视化开发等开发者需要的资源通过产品化的 Web 页面呈现出来，开发者就可以按照“我要开发--创建应用--填写应用基本信息--选择开发框架--选择代码库”的路径完成 SaaS 开发前的简单准备工作；当开发者完成业务逻辑代码后，再继续按照“应用部署--部署到预发布环境--部署到生产环境”的路径，完成应用的发布上线。

（2）提供最佳运行资源：每一个应用运行时，提供基础设施的基本要求，如运行时 Pod 的副本数、CPU、内存、存储要求等。

2）应用部署流水线建设

制定开发的代码分支管理规范和制品晋级规范，可以进一步减少产品-研发-运维-测试各岗位间沟通成本，快速建设应用部署的流水线。

（1）规范代码分支管理规范：用语义化版本号“主版本号.次版本号.修订号（Major.Minior.Patch）”，严格遵守版本号递增的规则。

a. 主版本号 Major：

产品侧：大功能，影响现有用户的使用习惯，新增亮点服务

研发侧：不兼容的变更，影响软件的升级与维护，影响基于该软件二次开发的服务，如架构升级，研发语言变更，数据结构重组，API 下架或 API 不兼容

运维侧：停机升级 / 数据迁移等，其他产品关联产品也需要配合升级

b. 次版本号 Minior：

产品侧：新增功能

研发侧：接口新增/接口调整（向下兼容）/性能优化等;原则上不允许 API 下架

运维侧：升级前需要查看下升级文档，允许做一些数据迁移相关的操作

c. 修订号 Patch：

产品侧：展示/交互/描述等小调整，例如文案的修复等

研发侧：bugfix，不会新增功能

运维侧：为了解决一些 bug 或安全问题，不会新增功能

（1）规范应用制品晋级标准：选择并使用几个关键节点作为制品晋级的必经阶段，定义制品的生命周期，并约定制品晋级的准则。如 1.0.0-alpha 表示当前应用的版本号是 1.0.0，该制品处于 alpha 阶段，仅完成了开发自测，可以发布到测试环境，提交给测试发起测试验收，禁止发布到预发布环境等其他环境。若需要晋级为 beta，则需要通过测试验收，才可以发布到预发布环境。

（1）提供把应用发布到某个环境的 CD 流水线插件：通过提供必要的应用基本信息，密钥信息，版本号，制品等级，环境信息等，使用 YAML 或者 Json 配置的方式，就能实现应用发布。

3）扩展服务（扩缩容、CLI 等）

（1）可扩展的存储服务，如为应用开发者提供存储服务，默认提供关系型数据库 MySQL、非关系型数据库 Redis，可扩展支持非关系型数据库 MongoDB、时间序列数据库 InfluxDB、搜索引擎数据库 Elasticsearch 等；

（2）扩缩容能力，在 K8s 能力的基础上，针对应用运行时需要的资源进行实时监控，确保对计算、存储、网络等资源要求是合理的，并能弹性扩展，以满足不同的业务需求；

（3）命令行界面 CLI（Command Line Interface）功能，不需要鼠标和图形化界面，而是通过键盘输入命令来完成各种操作。需要提供应用信息查询，应用部署，部署结果/历史查询等功能，支持 Linux、Windows 等主流的操作系统。

7.2.3 iPaaS 结构设计

iPaaS 是 Integration Platform as a Service 的缩写，翻译为集成平台即服务。在整个工程平台的设计中，iPaaS 连接上层的 aPaaS 和下层的原子平台，需要对已有的原子平台进行自动化的管理，自动化地接入外部第三方系统，实现对上提供统一的标准协议 API，可以让使用 aPaaS 的开发者们，快速拥有 SaaS 的组装能力。

1. 接入端设计

（1）原子平台提供 API：所有原子平台都通过 API 的方式，向上提供平台本身的核心能力。

设计网关的接入流程：让原子平台可以最低成本注册 API 资源。提供可视化的 Web 页面，首先需要“创建网关-完善网关基本信息”即是原子平台的基本信息，然后逐个“新建资源”，填写注册 API 的前端“请求方法、请求路径”，后端“Method、Path、Hosts、超时时间、Header 转换”，完善安全设置（如 API 使用的认证方式），实现一个 API 的有效注册。

提供 API 的管理能力：一个原子平台可能会提供数十个具体的 API，在 API 创建过程时，要提供可视化的 API 配置和文档编写页面，提供将 API 发布到不同的环境进行测试的方案，可以指定 API 进行特殊操作，如限制频率、授权使用、下架等。

（2）开发者使用 API：用户能看到的 API 要确保文档内容准确，并可以快速验证 API 可用性。

API 文档：要提供可供搜索的 API 文档，且采用统一的格式编写 API 文档，文档内容应该包括 API 基本信息（如更新时间、是否要申请权限）、API 地址（如请求环境、请求方法、请求地址）、公共请求参数、输入参数说明、调用示例、返回结果、返回结果说明等。

通用 SDK：要提供自助发布 SDK 的能力，每个原子平台可以将自己的 API 发布成 SDK，开发者通过下载并安装对应的 SDK，即可使用原子平台的全部 API。

在线调试：提供在线的测试环境，让开发者可以通过 mock 的方式，输入必要的参数，得到文档描述的预期效果。

2. 管理端设计

（1）确定网关服务模式：

选择流量网关：流量网关在 iPaaS 架构中的作用是全局性的，与后端服务完全无关的策略网关。不支持代码扩展。如一般使用 IAS，底层一般使用 Nginx，也可以选择 ngx_lua 。

设计业务网关：业务网关，即 API 网关，它与后端服务（如原子平台，业务系统）有一定的策略逻辑的网关。支持代码扩展，提供公共的能力（如限流、鉴权、监控等），不涉及后端服务具体的逻辑，不涉及 API 具体功能。可以采用插件的模式提供这些公共能力，需要设计一套插件的加载流程。

插件扩展功能：插件的内部要采用统一的结构，包括初始化 init、检查外包依赖 check_schema、rewrite、access、log 等，这样可

以对插件进行统一管理。依照这样的结构化模式，可以提供协议转换、路由寻址、负载均衡、服务发现、流量管理、安全防护、服务治理等插件，增强 API 网关的能力。

（2）建设插件生态：

插件文档：文档内容遵守统一的规范，包括插件名、插件版本、功能描述、属性字段、启用插件步骤、测试插件的效果、删除插件方法。

插件市场：提供“插件模板”、“插件商城”等功能，使插件开发者和插件使用者快速分享并启用感兴趣的插件。

7.2.4 通用原子设计

平台工程需要建设多个方面的能力，而原子平台位于整个平台工程的最底层，它的能力就组成了整个平台工程的最基础的核心能力。以下是在平台工程建设中，SRE 推荐的首要的、基础的原子平台。

1. 配置管理平台

（1）主机管理能力：提供主机信息的管理，支持主机信息的编辑，如导入导出，修改，复制等基础操作，可以扩展主机相关的字段，主机所属的运维人员，业务信息，SLA、城市等；主机信息不仅包括企业自建机房，私有云的主机，还应该可以同步各类云主机，并无差别对主机进行统一管理；主机信息变更可以通过订阅等方式，与周边系统进行实时同步与修订。

（2）空间管理能力：可以按照“业务”“项目”等空间管理的概念，划分主机的管理空间，并在这种空间维度上，对主机进行分类，如空闲机、故障机、待回收等；建设主机的默认拓扑关系，如“业务-集群-大区-模块”，可以通过“服务模板”“集群模板”的方式，快速创建同类型的主机拓扑关系。

（3）模型管理能力：“主机”的模型是最基础的配置管理信息，支持抽象，自定义创建更多的模型，如交换机、路由器、负载均衡、防火墙等。按照设备类型，组织架构，职能等多维度自定义模型，并使用对应的模型数据，进行配置信息的管理。

（4）API 能力：作为基础的配置管理信息，需要提供全面的 API，供消费方使用，并要封装部分场景类 API，提高 API 的性能。

（5）查询能力：围绕主机的全文检索能力，要有基本的 IP 信息查询，还要建设动态分组查询的功能，以适应不断变化的配置基础信息。

（6）云资源管理能力：对于高频使用的云产品，如 IaaS 层的产品，必须进行纳管，并提供针对全生命周期的操作管理功能（申请、购买、新建，操作，配置调整，回收，销毁）。对于云上 PaaS 层的产品，不同的云，有不同的形态，尽量按照统一管理模型纳管，如 CLB、COS 等。

2. 作业管理平台

（1）脚本执行：支持通过手动编写、本地上传的方式写脚本，脚本支持 SRE 常用语言，包括 Shell、Bat、Perl、Python、

Powershell、SQL 等；需要有“脚本管理”的功能，常用的脚本可以通过“脚本引用”的方式被其他脚本再次使用；选择目标执行的主机后，通过“滚动执行-滚动策略-滚动机制”的方式，能分批快速完成脚本执行。

（2）文件分发：支持从本地或从服务器上选择源文件上传后分发的模式，满足“一对多”“多对多”“多对一”等多种分发文件场景，且能自由控制上传/下载的速率，并设置文件分发超时时长；可以通过“滚动执行-滚动策略-滚动机制”的方式设置批量文件上传的滚动方案。

（3）任务编排：支持将脚本执行、文件上传、定时策略等多线任务进行编排，组装成一个作业，可以提前预置作业“执行方案”，在执行的时候，输入必要的参数即可。

3. 容器管理平台

（1）集群管理：提供 K8S 原生集群创建的能力，支持自定义设置 Master 和 Node 节点，一键自动安装集群组件，在用户独占集群时，保证安全隔离性；提供集群导入的能力，支持通过集群 kubeconfig 文件导入外部集群，对外部导入集群和已有的集群统一在容器管理平台进行管理；支持通过公有云的云凭证导入云上 K8s 集群，支持 Worker 节点自动扩缩容；集群管理支持节点添加和删除，集群删除，支持节点标签、污点与资源调度等节点管理功能，支持集群和节点级别的监控告警及主要数据的视图展示等。

（2）模板管理：提供在集群中部署资源的管理方案，支持设置容器编排的 Helm Chart。可以将同一套 Helm Chart，实例化到不同的命名空间，通过不同的 values，完成差异化的资源编排。

（3）应用管理：提供容器视图功能，可以通过应用视图或者命名空间视图管理容器，查看应用、POD、容器等的在线状态，启停容器，重新调度容器，对应用做更新，例如扩缩容、滚动升级等。

（4）镜像管理：提供公共镜像管理，包含了一些实用程度比较高，且开源共有的镜像资源。公共镜像对所有用户可见；提供项目私有镜像管理，是项目成员主动添加的镜像，或者是通过 CI 流程归档的私有镜像。项目私有镜像只有项目中指定的权限所有者才能访问。

（5）网络管理：提供服务管理能力，可以查看服务的列表，以及每个服务的详细信息，对服务进行操作，例如更新服务或者停止服务；提供负载均衡器管理能力，查看线上负载均衡器列表，以及每个负载均衡器的详细信息，启动、删除或者更新负载均衡器，并提供多云负载均衡器管理的能力。

（6）Workload 场景能力：针对业务场景定制面向无状态服务的 Deployment 和有状态服务的 StatefulSet 功能，如 Deployment 支持 Operator 高可用部署、支持滚动更新 / 原地更新、支持设置 partition 灰度发布、支持分步骤自动化灰度发布、支持 HPA、指定 pod 删除、支持 pod 注入唯一序号、支持防误删功能、支持 Readiness

Gates 可选功能等；StatefulSet 还支持 Node 失联时，Pod 的自动漂移、支持并行更新、支持 maxSurge / maxUnavailable 字段等。

4. DevOps 平台

（1）流水线：提供基本的流水线管理功能，如创建，查看、删除、设置标签等；提供可视化的 UI 界面来编排流水线，通过 Task（插件）-Job（作业）-Stage（阶段）的结构组成一个可视化 Pipeline（流水线），提供手动触发、定时触发、Commit 触发、制品库变化时触发等方式，最终可以让流水线的产出物，如 Artifact（构建），归档到指定的仓库中。

（2）代码分析：提供静态代码分析的自动接入功能，通过插件的模式接入到流水线中，就能检查源程序的语法、结构、过程、接口等，找出代码隐藏的错误和缺陷，如内存泄漏，空指针引用，死代码，变量未初始化，复制粘贴错误，重复代码，函数复杂度过高等，并将检查结果汇总，输出数据报表，给出修复的指导建议。提供自定义规则，可以在特定的场景对屏蔽代码分析。

（3）PreCI 功能：通过 JetBrains IDE 插件、VSCode IDE 插件和 PreCI 命令行工具等方式，提供基于开发者常用编辑器的本地代码检查功能，在打开文件/保存文件时，秒级返回检查结果；提供本地的规则集配置与代码分析插件的线上规则集同步功能，保持规则集一致；支持在 IDE 中查看检查结果，双击某个问题即可定位到其所在编辑器代码中的位置；支持在 IDE 中忽略问题和标记问题，支持注

释忽略；提供 pre-commit 和 pre-push 功能，实现不符合质量要求的不能提交到代码库。

（4）质量红线：提供一个“制品是否准出”的质量红线检查插件，插件可以综合项目团队内的要求（如单元测试，自动化测试通过率）和代码分析检查的指标（如代码缺陷、代码安全、代码规范、重复代码、复杂度等）设置阈值来决策代码质量是否符合预期，是否需要被红线拦截。

（5）编译加速：使用分布式编译、编译缓存等技术方案，支持 C/C++ 编译、UE4 代码编译、UE4 Shader 编译等，提供加速效果总览、任务注册、任务列表、加速记录等功能；编译构建机可以选择公共构建资源，也可以支持用户自行导入并注册构建机。

（6）环境管理：提供构建环境管理，用于在流水线中提供编译构建环境；提供服务器环境管理，开发人员可以将流水线中构建好的包（或者是自定义仓库中包）发布到服务器环境中，完成软件的功能验证或性能测试。

（7）凭证管理：管理的凭证类型要有密码、用户名+密码、SSH 私钥等，不同的凭证类型可以应用在不同的第三方服务当中。比如，通过 http 方式拉取代码库时，需要用到用户名密码+私有 token 凭证类型；而推送镜像到某个镜像仓库，则可能会用到用户名+密码凭证类型。

（8）研发商城：提供一个 DevOps 内容发布和聚合的中心，插件开发者、IDE 插件贡献者、流水线模板贡献者、容器镜像开发者都

可以将自己的作品发布到研发商城，对应的用户可以在研发商店快速检索、选择并安装到项目下使用，以此将 DevOps 最佳实践得到借鉴和推广。

5. 其他各类平台

包含但不限于：

（1）制品库平台：提供将 maven、rpm、npm、docker image、helm chart、二进制包等产物归档并分享的功能；结合流水线，提供制品库插件功能，直接归档 DevOps 阶段产品的制品；提供制品查询功能，可以根据制品名字、更新人等查询；提供制品的操作管理功能，包括重命名、移动、复制、删除、共享、下载等。

（2）代码库：在关联 GitLab、GitHub 等代码库时，提供源代码地址和访问凭证授权关联代码库；可以对代码库进行重命名，删除等操作。

（3）AI 平台：提供贴合企业需求的“场景方案”，如高危命令识别、单指标异常检测、多指标异常检测等；提供用户可以自定义创建的“算法模型”，可以接入样本集，经过特征工程、模型训练、评估，将训练好的模型发布，并应用到企业中；提供“样本管理”功能，可以进行查看、复制、删除等操作。

7.2.5 SaaS 分级

在异构应用平台工程建设过程中，针对 SaaS（Software as a Service）生态建设，可以根据其功能和服务的复杂程度进行 SaaS 分级，通过把各种共性场景的 SaaS 抽象出来成基础 SaaS，并提供原

子能力，再基于之上组合各种特定场景 SaaS，以便用户能够更低成本地快速构建覆盖不同应用场景的 SaaS，减少重复的开发工作量。通常可以分为以下两大类

1. 基础 SaaS（一级 SaaS）

基础 SaaS 是指通用的、可复用、可抽象原子能力的 SaaS，包含通用的功能模块和功能点，例如通用发布、通用监控、通用配置管理、通用流程管理、大屏表单编排管理等。基础 SaaS 可以为场景 SaaS 提供原子调用能力。示例如下

基础 SaaS	功能描述
通用发布	提供自动化作业的编排能力，支持对接制品库、负载均衡等系统，支持复杂逻辑如分支、并行等编排方式，实现发布执行作业全流程自动化
通用监控	提供流程引擎、表单引擎、值班管理、移动端等能力，支持根据运维工作需求自定义编排 IT 服务流程，并在流程中集成监控数据、配置数据、自动化执行作业等能力，可以将工作流和执行流无缝衔接
通用配置管理	提供可扩展的配置自动采集能力，支持云资源、数据库、中间件等 IT 组件的配置自动采集；提供数据质量运营服务，通过度量促进数据准确性工作的开展；满足资产管理、安全管理、故障排查等场景的数据查询需求，提供各运维角色的配置管理视图
通用流程管理	提供可观测数据（指标、日志、调用链、事件、性能、告警）的采集、处理、存储能力，基于不同的数据类型有不同的数据处理能力，每个服务都可以独立配置和复用，满足上层可观测场景的消费需求
大屏表单编排管理	提供拖拉拽方式组装的大屏和报表开发能力，支持从关系数据库、非关系数据库、API 等数据源获取数据，支持自定义的指标计算，用于运维各场景的度量统计和可视化展示

2. 场景 SaaS（二级 SaaS）：

场景 SaaS 是指针对特定垂直领域或特定业务场景的 SaaS，包含特定的功能模块和功能点，特点是给用户提供符合其用户习惯的定制化界面，用户体验大大提升。如面向单业务的发布、变更、配置管理等 SaaS。一些较轻量化的场景 SaaS 还可以基于 LessCode 或自动生成等方式来快速实现，高效满足不同业务特定场景的需求。

SRE 流程中可靠性架构设计、研发保障、入网控制、发布管理、故障应急、上线后持续优化工作等环节均可根据工作需求进行场景 SaaS 拼装，实现线上化操作。示例如下：

SRE 流程	场景 SaaS	场景说明
发布管理	统一发布中心	在通用发布的能力基础上，增加制品管理、参数管理、流程管控等能力，对接云上云下资源，对主机类应用和容器类应用的发布进行统一管理，通过发布方案和回滚方案的管理，有效控制发布质量
故障应急	统一告警中心	建设全链路可观测能力，整合 CMDB、trace、log、metric 数据、自动排查作业、自动恢复作业形成排障决策树，缩短故障恢复时长
	故障诊断中心	建设全链路可观测能力，整合 CMDB、trace、log、metric 数据、自动排查作业、自动恢复作业形成排障决策树，并通过告警快照等方式协助故障排查，缩短故障恢复时长
	应急管理中心	对应急预案、应急组织、应急流程进行统一管理，将自动化恢复作业关联到应急预案中的故障场景上，识别到对应故障即可进行应急恢复。支持应急演练线上化管理
上线后持续优化	运营成本分析	通过资源负载数据的统计分析和趋势预测，对低效无效资产及业务高峰带来扩容需求进行识别，并基于规则给出资源优化建议，提高资源运营精细化程度

除以上示例场景外，通过 iPaaS 的原子能力和一级 SaaS 的通用能力进行更多场景 SaaS 拼装，可有效减少运维琐事，如应用健康度

评分、容量管理、错误预算统计等场景均可通过场景 SaaS 结合不同技术实现线上化、自动化、智能化

7.2.6 服务管理

在异构应用的平台工程建设中，涉及一些公共的、需持续投入的服务管理功能建设，包含但不限于：平台可观测性、计算资源调度，服务计费等能力。

1. 平台可观测（日志、监控、APM）

在平台工程建设中，从 aPaaS、iPaaS，到原子平台，各类 SaaS，都需要一个完整的可观测服务。

（1）数据采集：提供监控采集插件，通过下发采集插件的方式，获取目标服务的数据，这些数据按照一定的采集格式，可以支持自定义数据上报；数据类型支持指标数据（Http 简易上报，Prometheus SDK 进行 PUSH），Trace 数据（符合 OpenTelemetry 协议），事件数据（HTTP JSON 数据，命令行工具，SNMP trap 上报）等，还可以接入告警源（如 Zabbix、Open-Falcon、Prometheus、腾讯云监控）。所有的数据都支持容器内的数据采集。

（2）数据检查：每个采集任务都要提供一个可视化的检查视图，可以查看当前主机/实例采集的数据情况；

（3）数据探索：提供数据检索功能，仪表盘功能，场景视图，视图报表等功能，将采集的各类数据（Metrics、Events、Logs、Traces、Alerts）可视化展示出来；

（4）策略配置：提供针对不同数据的检查或者告警策略，并通过 iPaaS 调用原子平台的能力，如作业管理平台，触发告警的产生后的运维操作。

（5）APM：提供 APM（Application Performance Monitoring）即应用性能监控功能，通用原子平台或者 SaaS 通过 SDK 上报数据，就可以展示应用内的 Trace 调用链，对应用进行排查故障，常规巡检，设置主动告知应用问题的策略等。

2. 服务治理

（1）服务可用性发现：提供统一的服务可用性 API，如 healthz 接口，上报服务的可用性状态；通过“观测服务”的能力，将服务注册到监控的仪表盘中；

（2）资源调度：使用容器管理平台，通过可视化的 web 页面统一管理平台工程的所有容器资源使用情况和在线状态，根据服务的负载、可用性、性能等因素，对服务进行调度和优化。

（3）故障自愈：这是服务优化的一个重要目标，SRE 根据平台工程架构和系统维护经验，对服务的监控和调度结果进行分析，提供当故障发生时的一个确定的恢复路径，并复用各类原子平台能力，实现工具的自动化。

3. 服务计费

（1）确定计费模式：根据各产品服务场景，提供按时间，请求量，数据量，次数，带宽，套餐等计费方式。

（2）采集使用数据：在产品内按照用户维度或者业务维度埋点并采集使用数据，可以通过日志记录、计时器、硬件时钟、第三方系统（如 Google Calendar、Time.is 等）、API 等方式采集数据。

（3）建立服务账单：账单需要考虑成本、用户体验、收益等各方面的因素，按照用户或者业务的维度，统计所有平台工程内的使用数据，给出按照天/月的账单，或者打包整体账单。

7.2.7 安全与审计

异构应用的平台工程由于涉及的模块多，涵盖多种技术栈，不同原子平台之间存在相互的数据通信，访问的用户角色千差万别，必须有统一通用的安全策略和审计标准，实现一致性和标准化的管理。

按照用户的操作路径，安全与审计主要体现在如何保护平台的安全、防止数据泄露、规范资产合规，以及如何全方位地跟踪和记录平台的使用情况。需要建设以下几个方面的能力。

1. 用户身份认证

（1）确定使用哪种认证方式：一般要提供最基础的用户名和密码认证方式，对于高敏感的平台，要提供 token，短信验证码等辅助认证方式。当认证通过后，用户可以获取平台的访问许可；当认证不通过，要给出明确的提示反馈，如密码错误，用户名不存在等。必要情况下，提供双认证方式，当用户名和密码认证方式生效后，可以启动短信验证码的认证方式。

（2）开放认证接口：让原子平台、SaaS 开发者可以通过接口集成统一的用户身份认证，提供 OAuth 2.0、OpenID Connect 等身份验证协议，方便让第三系统对接平台工程。

2. 权限管理

根据平台的功能和用途，设计授权策略，比如明确的授权范围（如某功能点的使用、某实例的数据访问等）、授权级别（如普通用户、管理员、高级用户等）以及授权期限等。

需提供集中统一的权限管理服务，支持基于 aPaaS 开发框架和第三方平台的权限控制接入，做到细粒度的权限管理。

（1）权限接入：提供不区分技术栈的权限接入方案，进行统一的权限管理；给通用原子平台和一级 SaaS 提供权限接入的 SDK（Python、Go、Java、PHP 等），使用 SDK 进行鉴权逻辑校验；给二级 SaaS 提供直接鉴权的方案；提供统一的无权限页面、申请权限页面、申请返回信息页等通用的逻辑界面。

（2）权限申请：提供 3 种主动申请权限的方式，申请加入用户组、申请自定义组合权限、从接入系统侧无权限跳转到统一的管理端申请权限；权限申请时支持拓扑实例选择、属性条件两种实例选择方式，申请人也可以组合选择所需要的权限；支持多个系统批量提交申请。

（3）权限模板：在接入的系统超过一定数量，单个系统内，系统之间的权限组合关系较多的情况下，提供一个可复用的权限集

合，通过权限模板的方式，把权限授权给用户或者用户组，实现权限模板更新后可以同步给已关联授权的用户组。

3. 数据保护与加密

（1）确定需进行保护的数据范围：最基本的敏感信息是用户个人信息，如用户的手机、电话、邮箱；根据国家政策、行业属性、企业要求也会有敏感数据的规定，如在互联网领域敏感信息有主机账号和密码，公有云的账号、密码、token，资产数据等。这个范围可以参考《中华人民共和国国家标准-信息安全技术网络安全等级保护基本要求》；根据平台工程的特性，系统初始化或运行中的配置文件，各系统间的数据传输，也需要加密。

（2）选择加密算法：一般的产品是默认要支持国际加密算法的，建议要支持国家商用密码，简称商密，拼音缩写是 SM。其中 SM1 和 SM4 是对称算法，对标 AES；SM2 是非对称算法，对标 RSA、ECDSA；SM3 是摘要算法，对标 MD5。可以按照具体产品的场景选择使用合适的加密算法。

（3）加密方案实施;存储中的数据，采用对称或非对称加密技术，确保储存在数据库或文件系统中的敏感数据得到保护。传输中的数据，使用 SSL/TLS 协议加密客户端和服务器之间的通信。产品页面上提供明文、密文统一的交互和表现形式，如密码统一显示 6 个*。每个技术栈要提供统一的加密 SDK 等。

4. 操作行为审计

（1）确定用户操作的日志格式：梳理各平台/系统的操作审计日志协议/标准，规范各平台/系统的操作行为输出内容要求，提供丰富的日志接入方式，如：日志采集、API 推送、各种技术栈的 SDK 等。

（2）建设统一审计中心：提供数据上报的 Web 端页面，系统管理员可以把操作审计日志接入到统一的审计中心；提供数据检索的功能，支持多维度检索权限范围内的系统操作日志；提供数据存储功能，支持平台管理员切换设置审计中心的存储，提供设置默认存储功能；提供审计策略的管理功能，支持用户新建、编辑、删除、克隆、启用/停用常规策略，当有用户行为触发审计策略后，支持查看命中审计策略的审计异常事件，支持对异常事件的检索；当审计数据量变多，各系统之间的操作日志进行关联审计的时候，可以通过 AI 策略发现异常数据。

5. 风险发现、分析与处理

从最上层的 SaaS、到 aPaaS、iPaaS、原子平台，各系统本身在设计的时候，就通过日志的方式，上报风险，并借助监控等可观测的能力来发现风险。建设面向异构应用的平台工程，除了保证平台工程本身服务可用性，更重要的是在风险发生后，有明确的处理流程，快捷的处理工具，及时恢复服务。

（1）提供风险发现能力：参考上述“服务管理-观测服务”的能力要求，各系统接入并使用观测服务的日志和监控能力，及时发现风险。

（2）工单跟踪风险处理流程：提供统一的风险工单处理流程，制定标准的规范化的风险工单，通过工单必要字段，建立基于工单响应和处理的 SLA 机制，全链路跟踪风险工单，并在风险工单处理完后，主动分析并输出风险报告。

（3）实现自动化响应和修复：基于已有的 SRE 风险工单处理经验，梳理不同审计场景的处理方法，沉淀出标准化的处理工具/套餐；提供自动化处理规则配置能力，满足一些明确处理方法的匹配规则，利用自动化工具实现自动处理、修复问题，可以更及时和快速地解决风险问题。

（4）持续改进风险分析能力：提供基于安全审计风险事件角度出发的分析数据视图和工具，帮助审计人员定期总结分析审计运营数据，主动发现风险隐患和可优化点，生成定制的审计报告，帮助业务方提供安全可靠的服务。

第四章 SRE 的进阶

整理第四章内容的时候已经是 2025 年了，SRE-elite 已经并入信通院云大所稳定性保障实验室，叫 SRE 工作组，成员超过了 40 人，每周二晚上我们延续着例会，分享新技术和各公司的 SRE 趣事，也偶尔在 Qcon 和 GOPS 等行业会上办专场。

回顾前边的几个章节，最长的是第三章，详细介绍了国内各企业 SRE 团队当前的职能，对于资深的 SRE 从业者还是有所帮助的，至少看到了与其他公司的差距；第二章计划更改为面向刚入行的初学者或者在校学生，主要讲述 SRE 的基础理论和基础概念。

第四章，面向带团队的 SRE 管理者，计划整理四个模块，探讨一下 SRE 团队短期如何生存，长期怎样发展；SRE 团队手中的重资产，也就是资源与算力的变现；以及 SRE 在新技术浪潮下如何跟上甚至驾驭数据与 AI 能力；最后是实现从止损到增收，从维护到运营。

1 SRE 团队的生存与持续发展

前几年，有些 DevOps 组织宣称的“谁开发，谁运维”“去运维，去测试”等理念影响了部分企业，裁减了运维团队，这其中有些做得好的，转向或者正在转向平台工程，希望以内部一体化研运

平台的方案替代传统运维职能，持续解决质量效率成本及安全问题；还有些做得不理想的，将裁减的运维职能直接闭环给了研发团队，但没有补充统一的工程化能力，本质上是放大了各业务开发团队的成本，放弃了统一的质量效率管控；其他的，介于两者之间。

本白皮书第三章中所描述的 SRE 职能，有些来源于转型成 SRE 的运维团队，还有一些来源于把自己称作 SRE 的研运工具开发团队或者中台团队，这两个群体融合在一起，构成了与 google 相似的 SRE 组织。

这样的 SRE 组织看似职能宽泛，以保障 SLA 为首要职责，但能否经得住各种新生理念、AI 新技术，甚至是经济波动的冲击，会不会重蹈传统运维组织的覆辙。

1.1 SRE 团队的生存法则

SRE 组织的生存法则，取决于设立这个组织的初衷，目前行业中有两种思路，一种思路是将 SRE 作为质量控制的决策者，组织架构上独立于业务产研团队之外，类似于监察机构，地位上与业务团队平齐甚至略高，比如可以驳回业务团队的发布变更等决策；另一种思路是组建一支辅助业务成功的 SRE 团队，业务的产研团队作为营收主体，对业务的财务结果负责，同时拥有决策权，SRE 团队可以提供建议及解决方案，可以将 SRE 人员分散于业务研发团队内，也可以统一组建 SRE 部门；

两种思路的生存法则都跟预算来源和权利来源相关，前一种思路来自于 SRE 与业务产研共同的上级领导，后一种来自于业务团队及共同的上级领导。

1.1.1 没有 SRE 岗位行不行

国内各行业对工程师岗位的命名规则不统一，按最大化原则，本章节中提到的 SRE 泛指从事第三章 1-7 节范围内职能的，对应用的可靠性及连续性有影响的工程师群体。在不同企业内可能被称为基础运维、应用运维、运维架构师、运维开发、工具平台开发、运营开发、平台工程开发、后台开发等。

任何岗位的生存与持续发展，都需要有谦虚的心态和适度的危机感，如三体中描述的，“弱小和无知不是生存的障碍，傲慢才是”。SRE 团队首先应该正视自己的存在性价值，说的直白一些，就是没有这个岗位，公司还能否运转，答案是，可以。

事实上，只有重视应用可靠性，而营收尚可的组织，才会设置 SRE 岗位，并寄予厚望；蜜月期过后（大致一年左右），如果可靠性未有明显改善，同时又出现负面评价，例如成本过高、引发矛盾、拖慢业务等，SRE 岗位将被缩减；如果可靠性有明显改善，而 SRE 岗位没有扩展出例行服务，SRE 岗位也将被缩减。

在组建初期，SRE 团队的重要性完全取决于业务研发及运维组织共同的上级管理层态度；一至两年后，SRE 团队的重要性取决于

业务可靠性提升的情况，以及 SRE 团队所占据的业务核心“研发运维运营”过程中，例行服务的范围和深度。

因此，SRE 虽然重要，但并不是不可或缺，SRE 团队要明白，自己的职能是可以被业务研发等岗位替代的，只是隐性成本会升高，业务质量会下降，管理层会权衡这些损失与上述 SRE 问题的利弊。而且管理层的要求也是会变化的，可能受公司经营情况、高层轮替等因素影响，难以预测。所以不论建立 SRE 团队的初衷是质量控制决策者，还是辅助业务成功，SRE 团队都应该具备谦虚服务于业务营收组织的心态，成为通过技术杠杆放大业务收益的工程力量。

1.1.2 SRE 团队的预算从哪里来

SRE 团队能否生存，最关键的因素是有没有固定预算，固定预算才代表这个事情要长期做，根据组织架构的不同，SRE 的预算有三种来源。

1，SRE 的预算及职能由各业务团队闭环

各业务研发团队自己承担 SRE 职能，招聘 SRE 人员，或者复用业务研发自己做，这种方式的弊端是不同业务之间的 SRE 职能难以统一，质量标准也难以统一，总成本大概率会超出妥善管理的集中式 SRE 团队，而且工具及平台建设能力弱，不利于业务长期发展。规模较小或业务线单一的企业大多选择这种管理方式

2，SRE 集中管理，统一划拨预算

规模较大，业务线较多的企业往往选择将 SRE 整合为集中的一个或多个团队管理，通常命名为“SRE”“平台工程”“技术风险”“技术运营”等，统一质量标准，集中资源建设工具平台，以中立的身份服务不同业务线，跨业务线全局统筹技术专家、算力等资源，提高 SRE 专业度，减少重复建设。

为了兼得分散管理的优势，有些 SRE 团队又在内部细分了 SRE 平台团队与 SRE 服务团队，前者集中发展 SRE 核心能力，后者紧密绑定服务各业务线，甚至可以定制个性化 SLA、SLO。

这种统一的 SRE 团队，预算来自上一级的统一拨款。

3，SRE 集中管理，预算来自各业务的货币化支付费用

对于实施了货币化改革的 SRE 团队，其人员、机器、软件等预算均来自各业务团队的实际采买，公司内部采用甲乙方结算模式控制供需平衡，由财务及 HR 团队参与管理，这种方式解决了中台团队投入资金的合理性及价值问题。

1.1.3 SRE 上一级领导的视角

SRE 团队可能误判工作内容的重要性，站在 SRE 团队+1 或+2 管理者的视角，也就是站在决定 SRE 团队预算的管理层视角，产品线的业务方向、市场营收，甚至合规安全等问题更为重要，对 SRE

团队的预期主要集中在上线后的可靠性方面，也就是说，作为幕后团队的 SRE，其主要工作内容大概率是不被理解的，甚至少有了解的意愿，更多的是看结果，而不是消耗精力参与分析过程；基于这个现实情况，SRE 团队做好可靠性只是基础工作，而且难以持续，难以发展，因此应该逐步将工作重心左移，以保障代码或研发过程可靠性的名义，参与到业务研发过程中（详见第三章第 2 节研发保障），在研发提效获得认同之后，可以逐步扩展线上监控体系，将以止损为目标的运维监控和以提效为目标的研发观测，逐步扩展为以拉升业务指标为目的的“业务监控”（详见第四章第 4 节 SRE 的四维监控），以确立 SRE 团队在代码生命周期中的固定价值和定位。

在此之前，SRE 团队的成本应该尽可能透明，并且有明确的参照标尺，例如与业务营收考核挂钩，可以通过成本分摊或费用结算等方式来实现，并且要给予承担业务营收 KPI 责任的团队一定的自主选择权。在猥琐发育的阶段，只有以算得清的成本自证价值，才能经历风浪，抵御组织架构动荡和管理层认知偏差的影响。

1.1.4 SRE 团队的生存法则总结

总结一句话，以高于同行或业务自建的 ROI，保障基础可靠性，不断降本增效创新，伴随及辅助业务发展

1，保障业务可靠性

对 SRE 来说，不论是采用分散管理还是集中管理，对业务线上可靠性的保障，都是基础工作，也是底线，这一项做不好，就会丧失存在价值。

1）贴近业务优化 SRE 流程

可靠性方面的 SRE 流程，主要包括上线前的风险评估及入网控制（第三章第 3 节）、变更管理（第三章第 4 节）、上线后的监控观测（第四章第 4 节），故障自愈、故障应急、灾备演练（第三章第 6 节）等，SRE 应该伴随业务各版本不断升级优化可靠性流程。

2）建设和升级自动化智能化平台

在多变的业务发展和技术环境中，构建一套既能满足当前需求又能支持未来扩展的 SRE 平台，对于任何规模的 SRE 组织来说都是必要的，SRE 作为以工程化能力立足的支持团队，应该面向代码全生命周期设计一体化的工程平台，在这一领域，入选 Gartner2023 和 2024 十大技术趋势的平台工程体系是不错的参考（第三章第 7 节）

3）左移参与业务研发服务

线上可靠性损失的根因主要集中在三个阶段，上线前研发阶段引入的代码 bug，上线中的变更事故，上线后的故障。前边第一点“贴近业务优化 SRE 流程”主要针对的是后两者，根据不同企业 Dev 和 Ops 水平的差异，代码 bug 导致的可靠性损失大致在 20%~40%之间，极端的有可能超过一半，因此如果 SRE 团队完全不参与

业务研发阶段的服务（包括了代码分析和扫描，详见第三章第 2 节研发保障 2.1.1 代码可靠性），对可靠性的管理是不全面的。

SRE 参与研发服务还可以提升研发连续性，为业务版本提效，SRE 处理研发环境及测试环境问题的效率远高于业务研发团队，从 ROI 视角更加合理，还能拉近团队间关系，增加 SRE 服务粘性，便于创新服务的推广。

基于跟业务开发良好的合作，还可以推动跨职能协作，例如在设计和编码阶段提前识别性能瓶颈、单点故障等隐患，避免后期重构的高成本（详见第三章第 1 节可靠性架构设计）；共同约定监控埋点、依赖项检查等，确保代码上线前符合生产标准；共享指标、日志和依赖关系视图，避免故障应急过程中的信息盲区；与业务研发配合实现限流、降级等容错设计，实现有损服务；提前嵌入非功能性需求（如安全合规），减少上线后的合规风险等等

4）定期反馈 SRE 质量指标

SRE 团队定期向业务反馈可靠性指标的核心价值在于建立透明化协作机制，通过数据驱动实现业务目标与系统稳定性的动态平衡。

首先要对齐业务目标，量化业务需求，共担风险，通过 SLO、错误预算等指标，将业务方对可用性、性能的模糊需求转化为可衡量的工程目标（如 99.9%可用性），避免资源过度、投入不足或过度损害用户体验。例如，电商大促前 SRE 与业务方共享错误预算使

用情况，若剩余预算充足，业务可加速新功能上线；反之则需优先修复稳定性问题。

其次，驱动业务主动优化与资源配置，例如 性能瓶颈可视化，定期反馈延迟、错误率等黄金信号（如支付接口延迟>500ms 占比），帮助业务团队定位高优先级优化点；再比如平衡成本与效益的决策 SRE 通过容量指标反馈资源利用率，指导业务方在扩容需求与预算间权衡。

再次，定期反馈 SRE 质量指标可以增强跨职能协作信任，打破“黑盒”认知，SRE 团队通过周报展示故障 MTTR（平均恢复时间）下降趋势，减少业务方的质疑； 同步指标，推动研发自主优化代码健壮性

最后，存档的 SRE 质量指标可以用于合规 审计，例如金融行业需要向监管证明系统符合 SLA（如年故障时间<52 分钟），定期指标报告成为合规依据； 银行 SRE 与业务连续性委员会共享 RTO（恢复时间目标）达成率，联合演练提升应急响应能力。

2，可靠性保障 ROI 高于平均

SRE 团队保障业务可靠性的同时，还要少花钱，才能体现专业度，获得更多业务的支持。尤其是对比同类团队，或者对比业务研发团队自己保障同样的可靠性指标所预估的成本，这就要求 SRE 团队可以量化自己的投资回报。值得注意的是，业务团队自己预估的保障预算，大多过于自信，趋于乐观，低于实际费用。这对于身处

同一家公司内，薪酬结构相似的 SRE 团队来说是不利的，因此 SRE 团队可以利用资源杠杆，通过自身技术手段，加大对业务资源优化的投入，以冲抵 SRE 团队自身成本，优化 ROI。

1）单业务资源优化

SRE 的单业务资源优化，最直观的就是容量管理，包括资源预算、实时监控、预测分析、自动化调度等（详见第三章第 6 节 04），也包括了对商务团队提供技术支持，在与各家资源供应商的谈判中通过议价及组合策略实现费用节约，SRE 需要建设可视化成本工具支持该场景。

2）跨业务算力调度

跨业务的算力调度，指的是 SRE 在按各业务预算，从各供应商采购回异构资源之后，统一转换为算力，通过调度复用，使得各业务的碎片算力可以被用于业务及中台团队的临时性需求，或预算外需求，例如运行虚拟化、微服务、训练推理等（详见第四章第 2 节）。SRE 需要建设综合算力调度及算力交易系统支持该场景。

3）工具工程化效率

SRE 是工程化团队，相对于业务研发团队，SRE 更聚焦于自动化及效率工具领域，因此在业务的全生命周期内，成熟的 SRE 团队能以远高于业务研发团队的效率建设及维护工具（详见第三章第 6 节 03），并以此冲抵 SRE 团队的成本。

4）故障损失规避预估

减少故障时长，就是间接减少业务损失，SRE 可以将此类间接价值转化为可量化的指标，例如根据历史故障的业务平均损失情况，推算 SLA 提升或错误预算的节约，与减少损失之间的数学关系，但站在某些类型的业务团队视角，用户在故障时段的收入损失，其中一部分是会在故障恢复之后补充支付的，这取决于业务对用户的粘性，例如电商；相反，有些业务故障对用户造成的间接损失，是无法预估的，例如金融交易，可能导致用户对业务线索赔，因此在估算 SRE 的挽回损失金额时，需要设定双方认同的系数。

除此之外，避免业务线品牌声誉损失、通过工具为业务提升版本效率等，都可以估算间接收益，前提是算法获得业务团队的认同。

3，降本增效创新

在以相对于业务研发及同类团队，以更低的成本保障业务可靠性之后，SRE 团队需要考虑的是后续的持续发展，也就是保持甚至扩大团队的价值，表现为 SRE 团队预算的增长。SRE 应该首先扩展业务团队有预算的服务项，因为这类服务项的业务价值相对真实。同时，还应该逐年降低已有服务的成本，以体现增效。

1）扩展有预算的新服务项

2）升级现有服务项

3）降低稳定服务的单价

4）增加客户数量

4，伴随业务发展

SRE 是辅助性的职能，其价值最终要叠加到业务上，SRE 不是花钱的运维团队，是通过技术杠杆放大业务收益的工程力量，因此 SRE 团队应具备经营意识，尝试通过 ROI 管理自身团队，并将救火止损定为短期目标，而将拉动业务指标，伴随业务增长作为长期使命。

1）SRE 货币化转型

2）始终紧跟公司战略

3）转型为业务增长伙伴

4）持续提升内外部产能

1.2 SRE 团队的阵地与产能

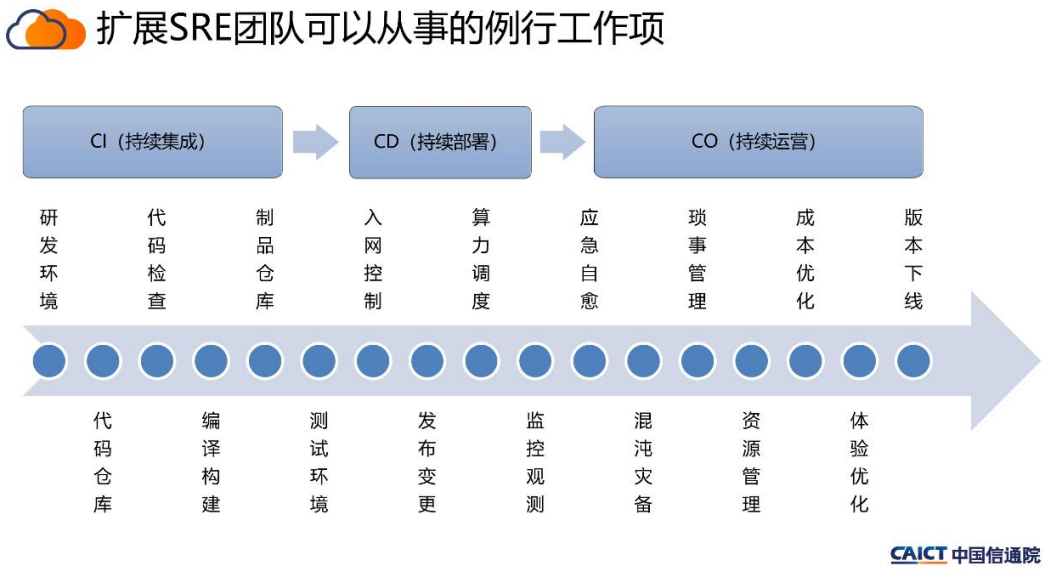
应用可靠性的提升是有上限的，在管理层的认知中，在可靠性被拉高之后，维持当前值的预算至少不应该再持续增长，甚至还可以降低，这就给 SRE 团队带来了困扰，需要时不时的解释和汇报；

而效率的提升是没有上限的，自第一行代码问世至今，软件带来的社会效率提升不知道已经是多少倍了，因此很多 SRE 团队会将对业务的可靠性保障作为主阵地，对业务的效率提升作为阵地的扩展，占据更多的例行预算。

1.2.1 SRE 团队的阵地

可靠性和效率，在代码的全生命周期中都有价值，SRE 团队可以根据自身情况，选择适合自己的阵地；从技术规划层面说，拥有平台工程的 SRE 团队，选择面更宽，扩展速度更快（详见第三章第 7 节 01）。

下图概述了在代码的全生命周期中，SRE 团队可以尝试进入的有价值服务领域。



从服务模式的角度看，有三种可以选择

1，技术服务，指的是 SRE 按业务要求，提供专业咨询、技术方案和操作运营等工作，使用业务方或第三方提供的工具平台，简单来说就是卖人头，专业话术叫人力云。

2，技术产品，指 SRE 团队根据业务需求，提供定制或者已有通用的工具软件，及相关维护，简单来说就是卖产品和维保，专业话术叫 SaaS 服务或私有化软件定制。

3，技术托管，指业务团队将 CI-CD-CO 全领域工作托管给 SRE，只提出技术及运营指标。简单来说，业务开发人员写完代码，提交之后的环节都托管给了 SRE，能做到这个程度的 SRE 团队已经充分获得了业务信任，实现了平台工程。

1.2.2 SRE 团队的产能提升

SRE 团队有了阵地，获得了某一部分的例行工作项，怎么长期坐稳并且越做越好成了下一个问题。所谓越做越好，简单理解就是要么服务的质量越来越高，要么就是服务的价格越来越低，这就需要 SRE 团队不断提升自身产能。

1，SRE 团队的内部产能提升

所谓内部产能提升，就是根据预算投入的方向，逐个优化 ROI，SRE 团队的成本投入大致有三个主要方向，雇佣工程师、租用设备、购买软件。在近些年自动化和 AI 快速发展的背景下，有些工作变成无人值守了，有些工作可以由对业务熟悉度和技能都不高的人员完成，这就给灵活用工带来了可能性，例如将工作分配给集团员工、子公司员工、外包员工的组合，以优化 ROI；租用设备部分可以通过算力优化提升（详见第四章第 2 节）；软件部分可以

采用低代码组装（详见第三章第 7 节 02）及 AI 辅助建设（详见第四章第 3 节）。

2，SRE 团队的外部产能提升

所谓外部产能提升，就是根据 SRE 团队之外的部门分工情况，甚至是公司外的行业现状，通过资源置换、开源协同等方式，在没有预算的情况下，获得 SRE 组织之外产能的补充，例如在公司内，SRE 团队算法资源的数量和质量相对不足，而算力及维护能力充足，公司内算法资源集中的团队，算力及维护能力大概率紧缺，这就形成了资源互换的条件，双方可以进行资源互换；在公司外的同行之间，可以通过开源项目补充 SRE 内部产能，不论是使用开源组件的团队，还是自主开源的团队，只要保持长期贡献的能力和意愿，都可以获得外部代码以提升内部产能；在不同行业之间，也有像信通院云大所稳定性保障实验室之类的组织，针对 SRE 建立了各领域关键技术的工作组，共享 SRE 成功经验，减少试错成本。

1.2.3 SRE 团队的工程经济学思维

工程经济学，是一种将技术决策与商业价值量化结合的方法论，用在 SRE，其核心就是：“将 SRE 团队有限的预算（人力、算力、软件）投向产生最大业务价值的服务”。

工程经济学的四个核心原则是，量化一切，平衡取舍，动态优化，长期视角。

SRE 团队需要自证“成本与价值”的合理性，可以通过定期的价值汇报，也可以通过货币化改革，一次性解决。

1.3 SRE 团队的货币化改革

可靠性不是所有企业各个时期的刚需，SRE 团队也不是可靠性的唯一路径。运维和 SRE 都可以被业务研发替代，只是这种替代丧失了集约效应，违背了木桶原理，使得可靠性与研运效率的上限降低、长期服务创新能力降低，而管理层有可能意识不到或者并不在意，因此 SRE 团队的生存与发展不能完全寄托在管理层的认知上，SRE 团队需要从商业模式上具备独立生存和发展的能力。

在企业中符合财务模型（预算-收入）的经营模式才是稳健的，SRE 团队需要将价值具象化到业务愿意付出的成本上，也就是即使业务团队自建，也不得不付出的成本上，成功的 SRE 团队通过集约效应与木桶原理，可以使得自身成本远低于业务自建的成本，同时在可靠性上保持较高的业务满意度，从而获得业务团队的价值认可，货币化的同时也意味着业务对 SRE 的服务拥有了自主选择权

SRE 团队货币化的目的 —— 获得无需自证的价值，长期发展的条件。

1.3.1 SRE 团队的货币化方案

SRE 团队货币化方案有四个参与方，SRE、业务、财务、HR

首先 SRE 团队需要梳理服务项并定价，之后与业务方确认，双方达成一致后 SRE 每月发出账单，业务确认账单，财务将账单计入业务当月成本，并承担价格监管及调整公示，HR 团队根据财务计算的 ROI 数据协助 SRE 团队调整成本。

1，SRE 团队动态成本确认

首先需要明确 SRE 团队的年度/月度成本，比如人力成本、资源消耗成本、软件采购成本，办公场地、差旅、外部合作（会议、认证、培训）、企业分摊等其他成本。

在虚拟结算阶段、试运行阶段，或者因部分团队配合度不够无法获得准确成本数据的情况下，可以与上级管理层商定一个总包费用，作为货币化起点。进入货币化正式运行期之后，再由财务团队按固定周期（月），给出 SRE 团队的精确成本数据，计算 ROI。

2，货币化服务项的设计

SRE 服务项的设计，以业务团队能接受为准，服务粒度可大可小，小到制品库单工具卖产品收费，大到按 OS 或核心数分阶梯设定单价，也就是研运全托管的服务模式

服务项应该是业务能感受到的服务，最好对照行业已有的服务收费模式，便于业务团队理解，例如 CI 工具的软件费用、变更、

灾备、工具定制等，服务项最好不是SRE 团队内部优化的工作，例如应急管理、故障自愈等，业务团队更愿意为结果付费。

初阶SRE 团队可以以人力成本为基数，按月收费或分次结算，也就是卖人做技术服务，高阶SRE 团队可以尝试以SRE 服务的价值定价，这要求SRE 团队在维持同等质量安全指标的前提下，效率及成本远优于同行。

3，货币化服务项的单价

核算SRE 团队投入的每个服务项的单月总成本，例如人力、资源、外采等，确定每个服务项的计量单位，例如按核心数、OS 数、调用量、并发量、计件等；总成本除以用量，算出初始单价，并根据考核方案设计冗余；也可以对标同类型商业化方案设定折扣。

4，与业务方确定服务项价格

用业务方的历史用量数据，推算过往几个月的SRE 服务费用，与不接受某服务项账单价格的业务商谈，确定折扣或是否继续使用，再根据协商情况调整单价及各业务折扣，计算差额成本，联合HR 团队制定降本增效计划，或与财务团队制定利润返还策略。

5，不适合货币化的服务

公司强制业务团队使用，不允许自由选择的服务，不适合货币化，例如监管审计类工作。管理层出于公司战略考虑，要求SRE 提

供难以收回成本的服务，例如对投后公司、战略伙伴的支持，应该给予单独的预算补贴。前瞻性的技术研究类投入，短期难以在生产中大规模体现价值，需要管理层拨款投资。这些不适合货币化的项目，需要定期审核，控制开销。

1.3.2 SRE 团队货币化的管理

货币化改革有可能是SRE 团队主动发起的，也有可能是被动的。有些是管理层看不清中台预算的合理性，希望借助业务部门的评判，筛选出SRE 团队的哪些服务是必要且成本合理的，哪些是必要但成本偏高的，哪些是SRE 在自嗨，其实没什么用的。相反，还有些是公司对SRE 团队的管理自由度较大，SRE 团队希望有更大的发展空间，甚至未来独立为科技公司，因此需要先行货币化来评估过渡。因此，SRE 团队货币化的管理严苛程度是不一样的，根据经营属性深度的差异，可以大致分为三类。

1，成为一支算得清成本的SRE 团队

以算得清成本为目的，还是相对容易的，这要求SRE 团队每月核算出准确的团队总ROI，管理层根据情况设定KPI，例如要求SRE 团队总体ROI 达到70%即可。对总体ROI 做要求，也意味着不同服务项之间可以相互补贴，以这种方式实施货币化改革，可以减少管理层对SRE 团队的精力投入，明确SRE 团队自身的价值成本，以及需要补贴给SRE 团队的金额。

2，培养一支有经营意识的 SRE 团队

经营导向的货币化改革，需要对货币化项做精细化管理，即 SRE 团队的每个服务项 ROI 均要达到不同的固定比例，总 ROI 再设定一个比例。这种方式下，管理层可以看到 SRE 在各个方向的投入，以及业务实际认同度，给出资源调配建议，并对前瞻性方向给予宽松的 ROI 进行补贴。

以上两种方式，都是将 SRE 团队作为中台支持团队进行管理的，因此本着服务业务，不给业务增加额外负担的原则，服务项 ROI 和总 ROI 的 KPI 都不应该超过 100%，实际 ROI 如果超过，应该以调降单价的方式减少业务费用。

3，打磨一支具备独立生存能力的 SRE 团队

独立生存，意味着需要 SRE 团队经得起业务团队的波动和同行团队的竞争，在没有外部补贴的情况下持续发展。SRE 团队需要具备造血能力，也就是可以在扣除试错成本之后，不断成功地创新服务项，发掘出业务团队愿意持续付费的卖点。这需要 SRE 团队内部管理模式既要维持稳定的质量控制，又能保持较高的效率以补偿试错成本，并且可以通过技术迭代控制成本低于业务自建及同行，最后还得建设基本的安全防护能力以防止团队品牌因漏洞、合规等突发隐患猝死。

年初服务项 ROI 的目标应设置于 100%以上，年底达标后，以调降单价的方式修正回 100%，并向业务团队发布降价公示，次年再设定新的 ROI 目标，以此模拟逐年的团队盈利。

1.3.3 SRE 团队的货币化工具

货币化自身是有管理成本的，SRE 团队应以工程化能力降低长期的管理成本。需要建设四种工具能力，HR 视角的单人成本和单服务项成本核算能力；SRE 视角的单业务资源费用和单服务项资源费用核算能力；业务团队视角的单业务账单系统以及明细（每个服务项的账单）；财务视角的单服务项 ROI 和 SRE 总 ROI 考核。这四个视角的工具能力，可以分开建设，相互调用，也可以做到一起，取决于企业内原有系统的建设情况，以及对货币化改革的支持程度。

1，**人力成本系统：**主要面向 HR 团队的工作界面。HR 团队有单人成本数据，但这类数据管控非常严格，一般不会做系统级的对接。SRE 可以建设针对货币化服务项的人员填报系统，将人员与货币化服务项挂钩，再由财务人员按月给出 SRE 每个服务项的准确人力成本数据，甚至 excel 交付都可以，应包括集团员工、子公司员工、外包员工等各类用工模式的总成本。

2，**资源成本系统：**SRE 给业务提供服务，必然会搭建各类工具，占用一定的算力资源，这部分费用也包含在 SRE 团队的货币化

费用中，需要以某种形式分摊到各业务账单里，例如按不同业务的操作量，或 OS 数、核数比例等。需要注意的是，这部分费用与业务模块占用的算力费用是不同的，这部分算力并不在业务的 CMDB 归属中。SRE 货币化算力成本的管理，可以建设在账单系统中，也可以统一归属到管理所有资源及算力的 FinOps 系统中。

3，**SRE 账单系统：**业务团队需要一套查看账单，确认账单的系统，要能清晰地看到每个月的，每个货币化服务项的 SRE 人力和算力费用、账单明细、历史趋势等，可以点击确认，也可以拒付。

4，SRE 货币化考核系统：根据业务确认账单的情况，财务人员可以核算出 SRE 每个货币化项的 ROI 和总的 ROI，用于对照目标对 SRE 团队进行考核，为 HR 和管理层提供数据。

1.4 SRE 货币化后的持续发展

货币化改革前 SRE 是公共资源，业务团队可以免费使用 SRE 团队的人力和部分算力，因此除了故障之外，对 SRE 还是比较宽容的，也不关心 SRE 搞了哪些靠不靠谱，用不用得上的创新和副业，必要的时候甚至还会配合点赞；而货币化改革之后，SRE 团队的成本需要通过各种货币化项结算给业务，且业务团队还有自己的 ROI 压力，这时候 SRE 会感觉到业务团队开始精打细算了。因此相对应

的，SRE 团队的思路也应该跟着调整，转变为面向业务真实需求和满意度的发展思路，追求业务方对 SRE 团队的服务质量和服务价格双满意。

1.4.1 优化服务项的“商业”模式

作为企业内部的 SRE 团队，优势是理解业务及客户关系，劣势在于业务方和其他中台团队也同样了解 SRE 的能力，人员沟通和转岗的便利，使得技术优势的有效期更短。

在这种情况下，要想达到让业务对服务质量和价格双满意是不容易的，需要借助 SRE 团队既能深入业务，又有工程化能力的优势，设计对业务团队粘性更大的“内部商业模式”。

1，服务项的设计要避免技术服务，至少是技术产品，推荐技术托管

在服务项的设计上要避免单纯的技术服务卖人头，这很容易与其他竞争对手陷入卷人均成本的恶性竞争，最终伤害到 SRE 团队的工程化能力，没有 SRE 平台的粘性加持，竞争对手或者业务研发团队也更加容易替代。

以技术产品模式设计服务项会稍好一些，可以隐含一部分价值定价，团队需要持续提升技术产品的迭代和运营能力，但服务质量会受限于业务用户能力和第三方技术服务团队；最理想的服务项模式是技术托管，即技术平台和操作使用均由 SRE 团队提供，达到最

佳质量效果。在有驻场 SRE 服务的前提下，能够更好地感知到业务团队的变化，获取一手真实业务需求，帮助业务解决一些突发的，没有预算支持的场景，这种付出可以一定程度上冲抵业务方对 SRE 货币化服务费合理性的质疑。

2，低成本延展服务范围以冲抵货币化费用

在货币化之前，以中台身份支持业务的 SRE 团队原则上应该是对业务有求必应的，但实际上 SRE 并没有动力扩大支持范围；在货币化之后，虽然有了明确的服务项和服务范围，但由于业务有权利质疑单价及费用的合理性，反而使得 SRE 团队愿意以某些额外支持，来换取维持自身货币化项的价格体系。关键在于为了减轻负担，SER 会优先扩展那些事半功倍的服务，也就是在技术栈和服务资源上与原有货币化项可以重叠复用的服务。

业务生产环境的自动化方案都是 SRE 负责的，因此从技术能力和服务人员的角度看，很容易扩展到其他环境，例如预发布环境、各类测试环境、研发环境等，甚至可以为业务研发团队提供全套的研发环境托管服务，降低业务开发除了写代码之外的维护负担。

成本优化是 SRE 的职能之一，货币化之前 SRE 的主责是可靠性，在保障可靠性的前提下，会根据业务要求，或出于责任心在力所能及的范围内优化成本，一般是采取与业务约定资源利用率，定期优化；货币化之后，SRE 会意识到资源杠杆的作用，在业务挑战货币化服务费的情况下，SRE 会与业务商谈，用计划外的成本优化

资金冲抵货币化费用，对于业务团队来说，这种计划外的现金收益必然是多多益善，因此双方很容易达成一致。因此 SRE 主观上会非常愿意通过自身经验，或调动 SRE 团队内的工程化技术能力进行大规模的资源成本优化，甚至不惜在错误预算范围内平衡成本优化的资金收益与技术风险。此类尝试也没有超出 SRE 的技术范围和服务人力，除了风险外，没有给 SRE 增加负担。

3，以平台工程视角加大 SRE 团队的技术投入

SRE 团队可以通过增强粘性的服务模式和主观上积极努力的服务扩展提升业务团队的满意度，但这些都是有瓶颈的，而且门槛较低。作为工程化团队，持续投资技术平台才是 SRE 不断提升竞争力的关键策略，投资技术平台对 SRE 团队来说不仅是工具升级，更是能力转型，使 SRE 团队在标准化、自动化、数据化、智能化的每一轮技术迭代中占据效率高点。

从当前来看，基于 PaaS 理念的平台工程，是最适合货币化后的 SRE 团队持续经营的技术底座（详见第三章第 7 节），其 iPaaS 层的原子化切割，保证了每一轮技术迭代的敏捷性和低维护成本，aPaaS 层的快组装能力，适合新服务项的低成本试错，PaaS 的分层结构与 SRE 团队内部的平台及服务两大职能划分相似，这也为团队内部的精细化管理探索铺平了道路。

1.4.2 SRE 内部的精细化管理

货币化后的 SRE 团队管理者，需要不断主动提升内部管理水平，尤其是以经营意识和独立生存为目的的两类货币化团队，都有持续降本增效和创新增收的动力。在预算有限的情况下，除了服务模式和商业规划的技巧，还可以通过 SRE 团队内部的精细化管理，持续提高产能，让每一分技术投资都精准驱动货币化收入和业务的增长。

1，了解基层 SRE 组织的状态和产能

SRE 团队的服务范围宽，覆盖了代码全生命周期，涉及到的技术栈多，在不同阶段各技术栈需要掌握的深度也有差异。SRE 管理者需要设计一套数据指标，绘制从 SRE 工程师到各级 SRE 团队的各类技术度量指标，用来有针对性地培训提升和调整重组各级 SRE 组织的状态和产能，以适应不断发展的业务需求。

以腾讯 IEG 对 SRE 工程师的技术度量指标为例，分为海量自动化、运维开发、DevOps、AIOps 四个大类，各类能力由所对应的工具平台的操作记录数据经计算得出数字指标，计算方法由各团队 SRE 代表共同商定，每半年商定更新一次，该指标仅用于 SRE 工程师的职级晋升。SRE 管理层通过技术度量指标，可以了解 SRE 个人及团队的技能状态和产能，也可以通过该指标直接调整 SRE 团队的技术重心，每次调整在三至六个月内就会在基层看到明显的转型效果，以适应业务团队、SRE 行业、IT 技术的变化。

2，算得清各服务团队和服务项的精确 ROI

SRE 管理层有了员工和各级团队的状态与产能之后，就可以将其转换为具体的业务产出，业务团队根据 SRE 的产出，按月支付货币化费用，作为 SRE 团队的收入，并结合 SRE 团队的成本，计算团队和具体服务项的 ROI。SRE 管理层可以根据 ROI 来调整各团队及服务项的成本投入，决策预研新的服务能力，通过技术度量指标调整 SRE 团队的技术方向，形成新的闭环。

这个闭环中除了技术度量指标外，还有一个精细化管理重点，就是 SRE 的业务产出度量，不同于业务线的全职研发工程师，他们往往是通过 DevOps 研效指标来度量产出，SRE 的服务项覆盖代码全生命周期，且技术栈宽泛。可以通过 AI 化的 ITSM 体系做 SRE 业务产出的半自动化，甚至全自动化度量。

以腾讯 IEG 的 SRE 团队为例，首先将 SRE 的时间消耗分类，例如操作类、应急类、质量优化、成本优化、自主建设、会议、学习等，再通过自动化平台、应急灾备平台、SLA 平台、Finops 平台、运维 PaaS 平台等自动计算出 SRE 工程师在各服务项消耗的工时，暂时不能准确自动统计的，通过 chatops 辅助 SRE 工程师创建工单并自动审计。以各 SRE 工时的消耗，作为业务产出，作为“单核服务费”货币化项的明细依据；以每个货币化周期单场景 SRE 工时的下降，作为 SRE 提效的依据，作为“单核服务费”货币化项降低单价的依据；以业务各个操作场景的工时为源数据，绘制各业务

的画像，作为后续各类型新业务接入 SRE 服务的定价依据，通过同类业务的 SRE 方案及经验分享，对新客户体现 SRE 团队的专业度。

3，感知到客户们的真实态度

在每个月业务团队手动确认账单的系统页面，设计方便易用的满意度调研功能，例如对本月货币化价格“满意”或“不满意”两个大尺寸按钮，作为用户表达情绪的渠道，和 SRE 管理层的告警通道，以及时联系业务用户了解基层 SRE 团队是否出现失职或误判，以适当的措施挽回，及时避免负面情绪在客户群体中积累扩散。保障 SRE 团队的服务效果，保障 SRE 货币化收入的稳健增长。

1.4.3 持续扩展货币化收入

基于现有的预算，或者当前货币化收入，结合精细化的 SRE 团队管理，就可以不断扩展服务范围，提升货币化收入了。可以尝试以下四个方向。

1，将 SRE 的服务扩展到代码全生命周期

在本节 1.2.1“SRE 团队的阵地”中，概述了 SRE 在业务代码全生命周期内，可以尝试进入的有价值服务领域，获得扩展的收入。其中各部分更详细的介绍都在本白皮书第三章。

2，尝试扩展公司外客户

在完成内部扩展，或内部扩展受阻的情况下，SRE 可以尝试将成熟服务延伸到公司外，以 toB 服务的形式扩展收入，例如同集团下的其他子公司，或有投资关系的企业。不建议 SRE 跨行业拓展服务。

3，公司内效率软件定制

工程化能力较强的 SRE 团队可以扩展业务团队的效率工具定制服务，SRE 在工具领域的能力积累相对于业务研发团队更强，因此在低于业务研发团队自评估成本的前提下，SRE 团队依然有很高的利润，而且在项目样本越来越大的情况下，更容易沉淀出通用工具，进一步提高货币化利润。对 SRE 来说，可靠性的提升是有上限的，而效率的提升没有，也就意味着内部 SRE 团队定制化服务的扩展空间是无限的。

在商机洞察方面，由于 SRE 团队可以提供驻场服务，与业务研发运营人员协同工作，因此非常容易总结出业务团队所需要的提效需求，如果以快速试错的方式先开工再谈费用，对业务团队来说感受更好。在商业模式上，既可以采取项目制收费，也可以采取内部订阅的 SaaS 模式收费，取决于业务客户初期可以腾挪的预算。

4，综合算力调度

SRE 团队管控的资源成本，是 SRE 自身人力成本的 5 至 10 倍，SRE 投入技术资源在成本优化方面，可以形成杠杆效应，收回成倍的货币化收益。下一节，我们详细探讨 SRE 团队的综合算力调度。

2 SRE 团队的综合算力调度 （校对中）

3 SRE 团队的数据与 AI 场景（编写中）

4 SRE 团队的基础监控与应用观测（编写中）

附录

1 参考文献

出版书籍：《SRE Google 运维揭秘》ISBN：9787121297267

出版书籍：《Google SRE 工作手册》ISBN：9787519845858

2 术语

此术语表描述了我们在白皮书行文中使用到的一些术语。

术语	定义
SRE	Site Reliability Engineering，即网站可靠性工程，是一套原则和实践，旨在创建和维护可扩展、高可靠和高效的软件系统。
SLI	Service Level Indicator，即服务水平指标，是衡量服务性能的具体指标。
SLO	Service Level Objective，即服务水平目标，是服务期望达到的性能水平。
SLA	Service Level Agreement，即服务水平协议，是服务提供者和客户之间关于服务水平的正式协议。
DevOps	Development and Operations，即开发与运维，是一套实践，旨在使软件开发（Dev）和软件运维（Ops）更加协同高效。
CI/CD	Continuous Integration/Continuous Delivery，即持续集成/持续交付，是一种软件开发实践，通过自动化的构建、测试和部署来加快软件交付过程。
AIOps	Artificial Intelligence for IT Operations，即 IT 运维的人工智能，是使用人工智能技术来改善 IT 运维的实践。
MTTR	Mean Time To Recover，即平均恢复时间，是从服务中断到恢复正常所需的平均时间。
MTBF	Mean Time Between Failures，即平均故障间隔时间，是系统在两次故障之间正常运行的平均时间。
PaaS	Platform as a Service，即平台即服务，是一种云计算服务，提供开发、运行和管理应用程序所需的平台和环境。
IaaS	Infrastructure as a Service，即基础设施即服务，是一种云计算服务，提供虚拟化的计算资源。
SaaS	Software as a Service，即软件即服务，是一种软件分发模型，用户通过互联网访问应用程序，无需安装和维护软件。
CDN	Content Delivery Network，即内容分发网络，是一种分布式网络，用于更有效地向用户分发内容。

API	Application Programming Interface，即应用程序编程接口，是一组规则和定义，允许不同的软件应用程序相互通信。
SDK	Software Development Kit，即软件开发工具包，是一组软件开发工具，用于帮助开发者创建应用程序。
BPM	Business Process Management，即业务流程管理，是一种系统化的方法，用于优化和管理组织的业务流程。
ECC	Enterprise Command Center，即企业指挥中心，是组织内部用于监控和管理 IT 运营的中心。
GOC	Global Operations Center，即全球运营中心，是组织内部用于全球范围内监控和管理 IT 运营的中心。