SRE 实践白皮书

v1.0.1

2024年2月

SRE-Elite.com

目 录

1 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，首批来自于互联网、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；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，深度的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，大会成为了 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，并逐步成为了最具影响力 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：

1. 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。

2. 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。这涉及到对系统的架构和资源进行优化，以便在不降低性能的情况下，适应实际工作负载持续不断的峰谷状态变化。

3. 性能： SRE 关注系统的性能，旨在确保系统能够在合理地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。

4. 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，并合理的开展系统的升级和维护工作。

5. 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。

6. 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。这包括制定和演练紧急情况的应急计划。

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的融合在一起，所有参与团队都在上述共享的 SRE 目标上做着各自的贡献。

2 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，组织可以更好地平衡软件开发速率和系统稳定性之间的需求，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：

1. 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。

2. 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。

3. 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。

4. 快速创新： 通过减少故障和提高系统的稳定性，SRE 团队为开发团队提供了更稳定的平台，使其能够更专注于业务创新和新功能的开发。

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：

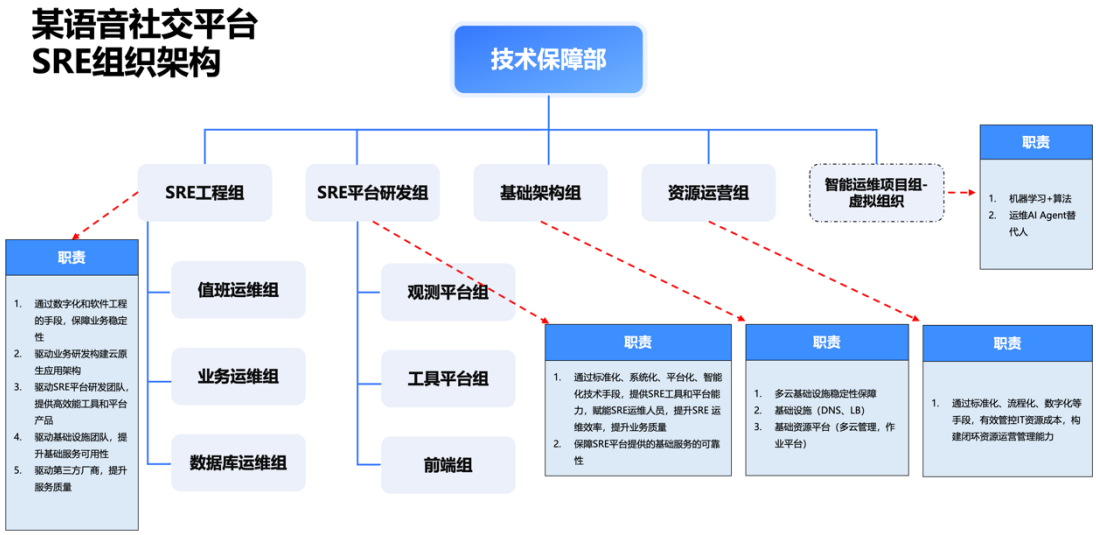
1. 中心化 SRE 团队： 一个专门的 SRE 团队负责支持整个组织的可靠性工作。这种模式有助于集中专业知识，确保在整个组织中实施一致的最佳实践。

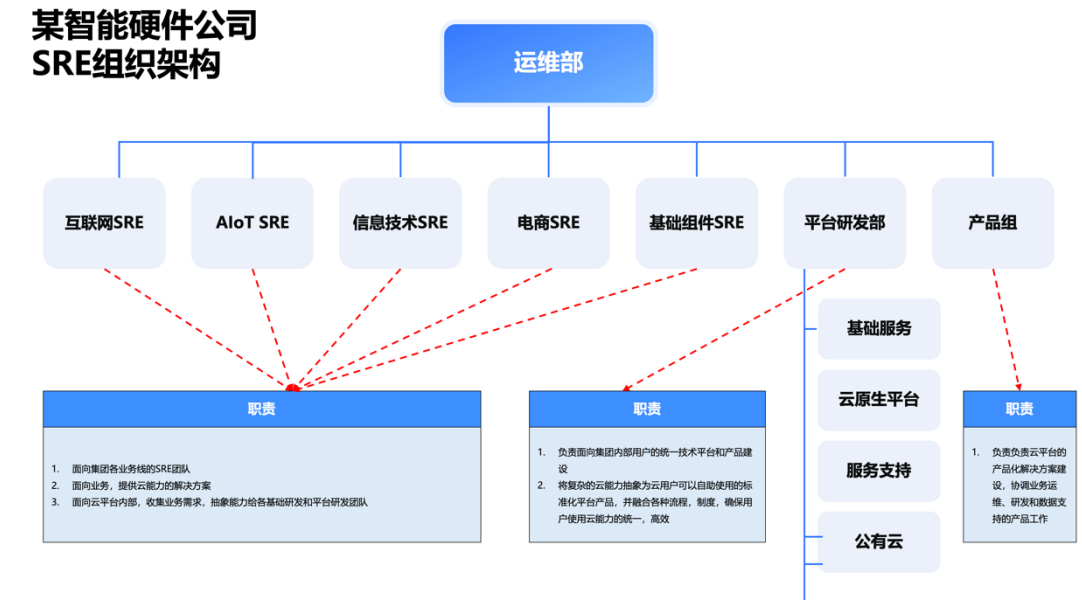
2. 嵌入式 SRE 团队： SRE 团队成员被嵌入到各个产品或服务团队中，与开发团队紧密合作。这种模式有助于更好地集成可靠性工作到产品开发的全过程中。

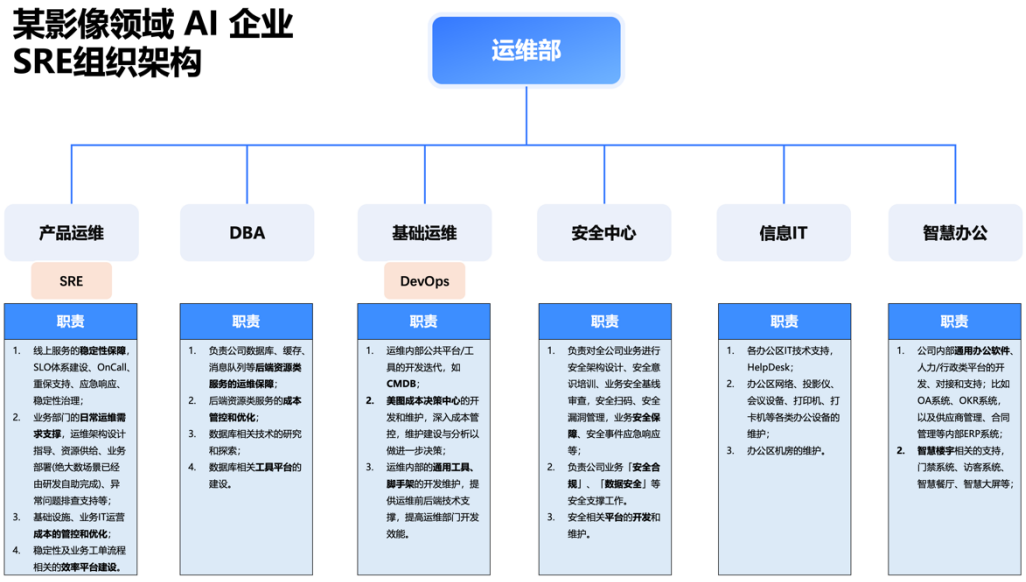
3. 混合模式： 一些组织采取混合模式，既有中心化的 SRE 团队，又在一些关键项目中嵌入 SRE 角色。这种方式能够兼顾专业化和贴近业务的优势。

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。


某语音社交平台
SRE组织架构
技术保障部
SRE工程组
SRE平台研发组
基础架构组
资源运营组
智能运维项目组-虚拟组织
职责
1. 机器学习+算法
2. 运维AI Agent替代人
职责
1. 通过数字化和软件工程的手段，保障业务稳定性
2. 驱动业务研发构建云原生应用架构
3. 驱动SRE平台研发团队，提供高效能工具和平台产品
4. 驱动基础设施团队，提升基础服务可用性
5. 驱动第三方厂商，提升服务质量
值班运维组
业务运维组
数据库运维组
观测平台组
工具平台组
前端组
职责
1. 通过标准化、系统化、平台化、智能化技术手段，提供SRE工具和平台能力，赋能SRE运维人员，提升SRE 运维效率，提升业务质量
2. 保障SRE平台提供的基础服务的可靠性
职责
1. 多云基础设施稳定性保障
2. 基础设施（DNS、LB）
3. 基础资源平台（多云管理，作业平台）
职责
1. 通过标准化、流程化、数字化等手段，有效管控IT资源成本，构建闭环资源运营管理能力


某智能硬件公司
SRE组织架构
运维部
互联网SRE
AIoT SRE
信息技术SRE
电商SRE
基础组件SRE
平台研发部
产品组
基础服务
云原生平台
服务支持
公有云
职责
1. 面向集团各业务线的SRE团队
2. 面向业务，提供云能力的解决方案
3. 面向云平台内部，收集业务需求，抽象能力给各基础研发和平台研发团队
职责
1. 负责面向集团内部用户的统一技术平台和产品建设
2. 将复杂的云能力抽象为云用户可以自助使用的标准化平台产品，并融合各种流程，制度，确保用户使用云能力的统一，高效
职责
1. 负责负责云平台的产品化解决方案建设，协调业务运维、研发和数据支持的产品工作

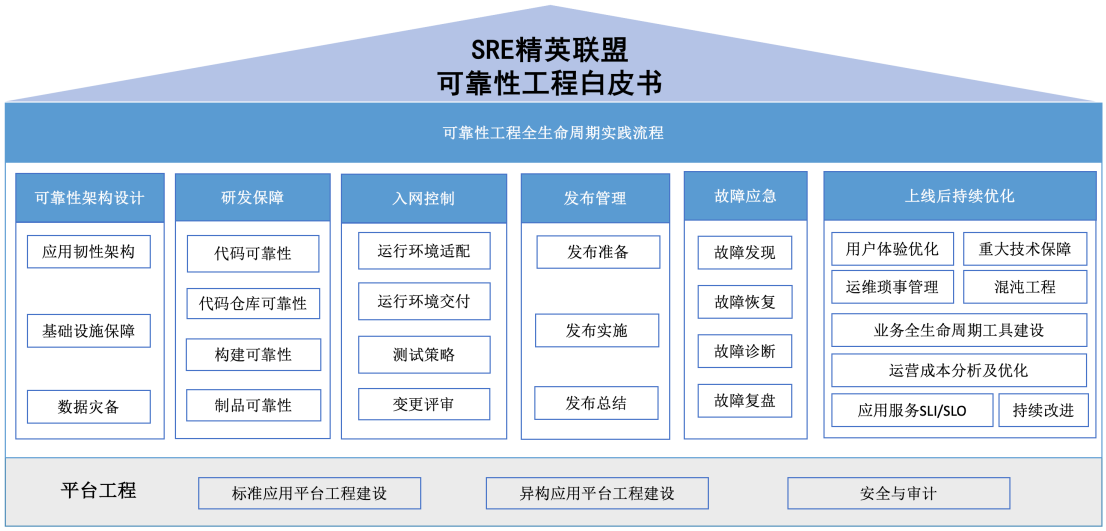


参考以上各个公司 SRE 团队在组织架构中的位置，通常 SRE 团队需要承担以下几类职责：监控、事故响应、事后回顾、测试与发布、容量规划、工具开发和可用性改进等。

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。

3 SRE 的流程

3.1 可靠性构架设计



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、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。

3.1.1 应用韧性架构

3.1.1.1 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、或根据用户与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。

3.1.1.2 解耦设计

在架构设计过程中，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的，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。

3.1.1.3 冗余设计

为了确保资源有足够的安全余量，每一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。

3.1.1.4 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。

3.1.1.5 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几种反应之一：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。

3.1.1.6 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。

3.1.1.7 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。

3.1.2 基础设施保障

3.1.2.1 机房多活

系统所部署的机器或所在地需具备一定的冗余性。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。

3.1.2.2 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。

3.1.3 数据灾备

3.1.3.1 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地验证。

3.1.3.2 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。

3.2 研发保障

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；

为了实现满足版本迭代周期要求这个总体的可用性目标，将研发阶段拆解为**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；

3.2.1 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；

3.2.1.1 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。

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。产生原因是由于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。

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

（W1/W2/W3 为权重系数）

3.2.1.2 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，缩紧，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，命名要能够直观的表达本身的意图，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）缩紧，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。适当的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）

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；

3.2.1.3 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，导致信息泄漏/非法侵入/系统崩溃等问题。

1．代码安全性提升措施

1）防止敏感信息泄漏

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。系统敏感信息泄漏会为攻击者提供更多的攻击方法，应用敏感信息泄漏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到代码，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的会引入依赖库，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，可以从敏感信息泄漏、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

3.2.1.4 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

降低函数圈复杂度的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

圈复杂度反应了代码的耦合度，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

3.2.1.5 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，会造成了代码冗余和浪费。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%

3.2.1.6 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

（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。

3.2.1.7 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。

3.2.2 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、安全提升等等工作，可由服务 SRE 和安全人员来承担相关能力的建设；

3.2.1.1 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，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

（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；

3.2.1.2 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；

3.2.1.3 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；

3.2.2.4 仓库可扩展性

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；

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。

3.2.3 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；

3.2.3.1 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。

（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了瓶颈，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

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；

3.2.3.2 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。

3.2.4 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；

3.2.4.1 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；

3.2.4.2 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；

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；

3.2.4.3 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，防止资源泄露和，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；

3.3 入网控制

3.3.1 运行环境适配

3.3.1.1 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；确定产品质量是否已经达到面向外部玩家的品质；通过相关评审来后项目组可根据项目实际情况开启正式运营。

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。

3.3.1.2 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型评估，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，这个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。

3.3.1.3 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。

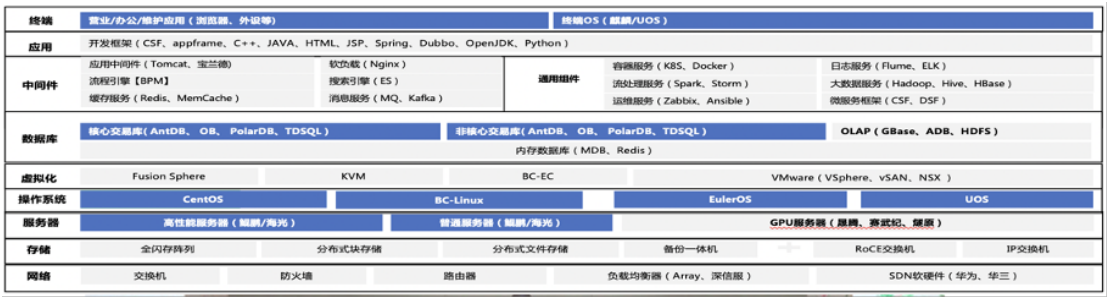
数据库性能优化。分析数据库性能瓶颈，优化查询语句、索引、表结构等，提高数据库性能。

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。

3.3.1.4 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，更多的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，搭建同比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；

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，在真实环境搭建的信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。

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。

3.3.2 运行环境交付

3.3.2.1 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)。

3.3.2.2 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。

3.3.2.3 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。

3.3.3 测试策略

3.3.3.1 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。

3.3.3.2 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

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。

3.3.3.3 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，可以分为查询类和办理类，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，为了更真实的拟合生产实际，考虑通过对单集群的压力测试来反应真实的生产场景。

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；处理缺陷即是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，不过在执行测试是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。

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、测试数据等及时的修复，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。

3.3.3.4 数据迁移（有些公司有独立 DBA 岗位）

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。

3.3.7 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。

3.3.7.1 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消息中间件等 PaaS 云服务等。为了更好的推进评估工作，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，将高可用要求更早的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，可恢复性的基础设施环境需能够有效地从自然灾害或人为灾难中恢复，即考虑到各种风险，包括自然灾害、技术灾难、人为错误或网强行攻击等，并采取预防措施以减

少潜在的损失，至少应包括备份、冗余、快速恢复服务和关键信息系统的保护。

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。包括：将系统拆分为多个独立的子系统或模块，每个拆分的部分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；支持横向与纵向的扩展能力，通过增加服务器数量与提高每个服务器的处理能力，来提高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;减少总线节点服务依赖，由多节点组成的逻辑交互改为端对端的访问方式，减少影响交易因素，少自身性能问题影响其他应用系；增加异步访问机制，同步的机制在性能出现问题时，会在短时间消耗完最大连接数，哪怕这个最大并发数是正常情况下的 10 倍，将同步连接改异步通讯，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。

3.3.7.2 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”，对日志的非结构数据进行加工处理，生成非结构化数据，进行运行数据分析，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。

3.3.7.3 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。

3.3.7.4 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。

3.4 发布管理

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。

3.4.1 发布准备

由于发布涉及到产品、开发、测试和运营等多个职能团队的参与，因此在正式实施发布操作之前，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“发布总协调”的角色。在发布前，SRE 需要进行以下关键任务：

- 审核发布产品功能和服务的可靠性。
- 评估发布的风险，并制定相应的应对策略。
- 选择合适的发布方式和方案，以最大程度降低影响。
- 协调各职能团队完成相应的准备工作，确保协同合作。
- 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。
- 预先制定针对各种可能的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。

3.4.1.1 发布风险评估

发布风险评估是指在进行软件或系统发布前，对可能存在的风险进行评估和分析，以便在发布过程中及时采取措施来降低风险，确保发布的稳定性和可靠性。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，发布带来改变，改变带来风险，【发布】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。

1．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．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．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．其它风险

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的追求越高越好，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。

3.4.1.2 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

如果时间很紧，可能需要选择更快速的发布方式，例如滚动发布或灰度发布。滚动发布可以逐步更新应用程序的不同部分，以确保整个应用程序始终可用。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。

在时间紧迫的情况下，有必要考虑采用更为迅速的发布方式，如滚动发布或灰度发布。滚动发布通过逐步更新应用程序的不同部分，确保整个应用程序持续可用。与此同时，灰度发布则通过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，从而在较短时间内获取关键反馈。

这种灵活的发布方式允许在时间敏感的背景下进行有效的测试和验证，确保新版本应用程序的性能和可靠性达到预期水平。在紧迫的时间压力下选择适当的发布方式，有助于在不牺牲质量的前提下推动项目的快速前进。

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。

3.4.1.3 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，使其成为最为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。

3.4.1.4 发布 checklist【制定发布执行清单】

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。清晰的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。

3.4.1.5 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。

3.4.2 发布实施

3.4.2.1 发布准备

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。

3.4.2.2 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。

3.4.2.3 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，持续关注关键的基础或业务指标，包括但不限于：

- 基础指标：系统负载、内存容量、网卡流量
- 业务指标：请求速率、请求时延、请求成功率

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。

3.4.2.3 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。

3.4.3 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。

3.5 故障应急

3.5.1 故障发现

3.5.1.1 监控发现

监控发现是指 SRE 通过监控主动发现系统中断的情况。为此，SRE 应考虑以下内容：

1．监控指标设计

SRE 优先考虑通过应用层指标监控当前用户的体验情况。常见的应用层指标有：延迟、流量、错误率、饱和度等黄金指标。当应用层指标缺失时，SRE 可以考虑通过进程层面、OS 层面 或 其他基础设备的指标，来推断用户体验情况。

2．指标标准化

为了提升指标的一致性和监控效率，SRE 团队在与开发团队密切合作时，可以提供一个统一的开发框架。这个框架旨在简化指标

收集和监控配置的过程，确保关键性能指标的规范化。同时，SRE 团队在深入理解系统架构的基础上，应当督促开发团队引入更多能够真实反映用户体验的衡量指标。

为了实现跨应用和 IT 基础设施的协调一致，建议制定明确的指标规范。这些规范不仅促进了不同团队在统一技术环境中的协作，也有助于从日志数据中衍生出必要的指标。

在跟踪日志的规范化方面，通常推荐采用 OpenTelemetry 协议，通过 SDK 的方式规范跟踪日志，并建议开发团队内部达成一致。

3．告警触达

SRE 配置的监控策略，应以“故障影响程度”进行行分级，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

3.5.1.2 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。监控关注的指标有限，如资源使用率（文件系统可用空间），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、系统启动时长等）

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。

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 应可方便的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，此时比例是越低越好。

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。

3.5.1.3　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，人工上报成为了监控系统的重要补充。

人工上报通常涉及以下几个方面：

1.舆情监听：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。

3.5.2 故障诊断

3.5.2.1 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

3.5.2.2 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。

2．辅助定界手段

1）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。

2）链路跟踪手段

链路跟踪通过对于符合 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。

3）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

4）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

3.5.2.3 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。

3.5.3 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。

3.5.3.1 架构自愈（啥也不用干）

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/VGA），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。

3.5.3.2 应急预案（已知的预案）

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。

3.5.3.3 应急维护（人工干预，未知预案）

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。

3.5.3.4 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。

3.5.4 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几率及爆炸半径。

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。

3.5.4.1 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 方法抽丝拨茧层层扒开迷雾，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。

3.5.4.2 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”）。通常可以通过大约五个“为什么”找到最根本的原因，但有时根据不同的情况只需要问 2-3 次，有时需要问 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，引发慢 SQL 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为了针对根因制定改进措施，避免同样的问题再发生。

3.5.4.3 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。

3.5.4.4 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。

3.6 上线后持续优化工作

3.6.1 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。

用户体验是指用户在使用产品或服务时所感受到的一切，包括感官、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。

3.6.1.1 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。

3.6.1.2 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。

3.6.2 重大技术保障

3.6.2.1 整体统筹保障

整体统筹保障是指针对公共事件重大技术保障，梳理好完善整体计划和资源统筹，有效协调各组织和部门有效合作，共同保障系统在公共事件支持时，持续稳定的运转。

整体统筹保障措施包括：

（1）建立重大技术保障指挥中心，整体统筹重大技术保障的各项工作。包括技术方案、设备资源、人力、物力、财力、宣传等全方面的统筹安排，以充分调动各个部门和岗位的力量，以满足重大保障需求。

（2）保证信息共享，包括信息的采集、传递和共享，以保证组织内部各个部门和岗位之间的信息互通。

（3）明确任务范围，这是为了落实公共事件支撑的目标，并进行任务的 checklist 文档整理，防止操作失误。

（4）时间确认，这是为了确定任务的精确执行时间，控制好执行步骤和执行流程批次，避免造成流程的干扰和任务的拥塞。

（5）角色及责任确认，这是为了在出现问题的时候，第一时间知道联系谁。每一款工作明确责任人，并负责跟进到底。

（6）任务执行确认，这是为了能够在保障任务执行的时候，有序开始，不会手忙脚乱。

（7）总结经验（参考 SRE 复盘机制）、确认任务范围、做好保障时间确认、责任确认、操作执行确认。总结经验为了给下次保障做经验积累，避免下次保障又全部从头开始筹备。

3.6.2.2 技术方案保障

技术方案保障是指在组织开展技术项目或实施技术方案时，对技术方案的可行性、可靠性、安全性、成本效益等进行评估和保障，以确保技术方案的顺利实施和达成预期目标。

技术方案保障措施包括：

（1）技术方案的可行性评估：对技术方案的技术可行性、经济可行性等进行评估，确保技术方案符合组织的实际需求和发展战略。

（2）技术方案的可靠性保障：对技术方案的可靠性进行评估和测试，确保技术方案能够稳定运行，达到预期效果。

（3）技术方案的安全性保障：对技术方案的安全性进行评估和保障，确保技术方案不会对组织和用户造成安全风险。

（4）技术方案的成本效益评估：对技术方案的成本效益进行评估，确保技术方案的实施成本和效益的比例合理。

（5）技术方案的实施和维护保障：对技术方案的实施和维护进行规划和保障，确保技术方案能够顺利实施和维护，达成预期目标。

技术方案保障是组织技术项目和技术方案实施的重要保障措施，需要在技术方案开发、实施和维护的各个环节中进行。

3.6.2.3 工具可靠性保障

工具可靠性保障是指针对公共事件的重大技术保障，通过保障工具的开发、测试和质量控制，以确保其在公共事件重大技术保障过程中的可靠性和稳定性。

对于公共事件重大保障的稳定性来说，要求体现在变更执行时间和变更执行顺序要非常精确，同时针对异常突发事件，需要及时预警。由此，SRE 需要通过工具可靠性保障，提高系统变更的效率和预警的准确性。

工具可靠性保障主要包括：

（1）自动化工具可靠性保障。从 SRE 团队的角度看，对变更时间的精度要求，可以看成是重要业务活动特有的 SLI 可观测指标。涉及SLI指标，SRE团队就会想办法优化提升。所以在实际落地时，会努力减少人工操作，将所有系统时间同步，通过流程编排工具，优化执行步骤，将执行过程自动化。

（2）容量评估工具保障。针对公共事件支撑，SRE 团队会通过容量评估工具，结合业务的历史的SLI、SLO指标，以及业务当前水位和目标水位规划资源分配，提前做好业务、以及周边平台组件的容量规划评估、资源筹备和程序自动部署的工作。

（3）可观测工具可靠性保障。是指通过业务状态检测工具开发，实时获取业务状态信息和业务作业执行流程的返回结果，辅助 SRE 做公共事件的实时状态验证。SRE 需要建立监控和预警系统，及时发现和预警重大技术保障过程中的突发事件，提高应急响应的

效率和准确性。

比如哀悼日暂停游戏服务、双十一等大型公共事件保障支持中，SRE 可以通过开发全方位的可观测数据化视图工具，实时监测业务状态变化。

（4）AIOps 工具可靠性保障。是指利用 AIOps 技术，建立智能预警和决策支持系统，提高应急响应的智能化和自动化水平。针对业务平台系统关键场景曲线指标做实时数据异常检测和数据预测，针对 SLO 服务消耗错误预算燃烧率数据预测等，在尽可能早的时间内发现异常情况，并且提供有效措施干预止损，减少系统误告率；同时根据不同的决策需求和数据输入、系统反馈或服务状态等，智能匹配生成最优告警处理和故障自愈解决方案，并且进行实时的 AIOps 算法自适应调整和优化。

比如在公共事件重大保障时，可以通过 AIOps 工具进行业务可用性状态预测判别，容量预测与容量自动扩缩流程；同时通过 AIOps 智能告警和故障自愈，减少人为的故障处理时间。

（5）数据备份和恢复工具保障。在公共事件发生突发事件时，数据可能会丢失或损坏，需提前确认数据备份和恢复工具的可用性，确保在任何情况下数据不丢失。

3.6.2.4 突发事件保障

突发事件流程保障是指在重大事件保障过程中，面对突发事件，组织内部能够迅速、有效地响应和处理，以保障系统安全和业务的正常运转。

突发事件保障的措施主要包括：

（1）突发事件的预警和识别：组织需要建立预警机制，及时获取和识别突发事件的信息。

（2）突发事件的评估和分类：对突发事件进行评估和分类，确定其性质、影响范围和紧急程度。

（3）突发事件的应急响应：根据突发事件的性质和紧急程度，启动应急响应机制，组织相关人员进行及时故障处理。保障在出现紧急情况的时候场面不混乱，执行有章法（可参考 3.5.2.1）

（4）突发事件的信息发布和沟通：及时向内部和外部发布信息，保持沟通和协调，保证信息有效同步，让突发事件处理有条不紊。

（5）突发事件的处理和复盘：对突发事件进行处理和反思，总结经验教训，及时调整和改进工作方式，完善应急预案和流程，提高工作效率和质量。帮助业务全面挖掘问题的根源，总结成功的经验和不足之处，为未来的工作提供有力的指导和参考。（可参考 3.5.4）

3.6.2.5 示例 1：哀悼日停止游戏服务保障

以全国哀悼日所有游戏停止服务保障为示例，SRE 面临着技术和流程上的巨大挑战，主要体现在：

（1）支撑压力大：所有游戏停止服务是属于重大公共事件，备受社会的广泛关注，如果支撑失误，可能会造成企业口碑损坏或者企业经济损失等非常恶劣的后果。

（2）时间有限：停止游戏服务，保障重大公共事件的支撑流程确认时间有限，需要特殊启动故障和重大事件的应急预案，留给腾讯游戏 SRE 准备实施的时间非常有限

（3）技术复杂：停止游戏服务，保障重大公共事件的支撑流程的确认涉及比较复杂的流程实施，几百款业务同时操作停服和起服，需要重新评估平台和周边组件的性能影响；并且需要有完善的保障计划，在技术保障前，逐一确认保障手段的执行方法，确定最终保障的目标和落地效果。

由此，在整体统筹保障上：

（1）建立重大技术保障指挥中心，整体统筹重大技术保障的各项工作：腾讯游戏 SRE 分别组织了现场的技术保障会议室和线上腾讯会议，方便快速响应和快速沟通。现场通过提前预订会议室，保证所有负责重点业务的 SRE 保障人员都集中在会议室里面进行保障；而涉及全国各地的 SRE 人员，则可通过线上腾讯会议，保持实时沟通。

（2）保证信息共享：通过腾讯游戏 SRE 故障应急机制，保障信息的实时传递和共享。涉及到相关的角色可以有：Operations Lead，操作指挥，简称 OL。他需要带领团队制定保障计划，确认时间，确认步骤等执行细节。到达执行时间的时候，OL 下达开始执行的指令。 Incident Responders，简称 IR。他们按照计划开始操作，正常完成或者出现异常会第一时间进行信息同步，汇报给 OL 和 QA（质量跟进的人员，他们负责整体保障结果的确认）。 Incident

Commander，故障指挥官，简称 IC，即此次应急保障总指挥。在遇到问题、出现异常的时候，OL 会同步信息到 IC。如果是一般问题，OL 可以处理的，一般会直接解决。如果不能解决需要协调更多资源的时候，或者需要更高层决策的时候，都会汇报到 IC 这个角色。以 SRE 支撑全国哀悼日禁娱全游戏停服为例，在腾讯游戏，OL 即为 SRE 小组的组长，IR 即为各团队负责业务相关的 SRE 同学，IC 为整个公共事件技术保障项目的总负责人。

（3）明确任务范围：腾讯游戏 SRE 通过系统导出涉及到对外部玩家提供服务的所有游戏业务，并进行负责人二次确认，保证停止游戏服务的任务执行范围，符合国家部门的规范要求。通过对数百款业务停服和开服流程整理和确认，防止操作失误。

（4）时间确认：通过对数百款游戏业务的任务执行时间分类，适时协商部分低优先级业务提前停服和延后开服，避免在规定时间点批量执行，产生不可预知的任务拥塞和大范围故障

（5）责任确认：通过腾讯游戏 SRE 管理平台，明确业务的运维责任人和业务管理责任人，保证出现任何问题，及时联系相关同事快速处理并跟进到底。

（6）任务执行确认：业务在停服和开服操作期间，自动上报和人工验证登记业务执行操作时间点，保证整体任务的批量统筹、统一管理和事后复盘。

（7）总结经验：在完成全国哀悼日停止游戏服务保障后，以执行业务小组为单位，统一推进业务完成整体保障的流程梳理和回溯

复盘，并输出改进优化意见存档，在更大的组织单元方面进行经验分享，不断迭代和优化流程。

在技术方案保障上：

（1）技术方案的可行性评估：梳理整体的任务执行流程要求，在组织内进行讨论和可行性评估确认。其中技术方案执行要求包含：

- 所有业务要准时关服，准时开服
- 关服宁可早不能晚，开服宁可晚不能早
- 开关服后要验证，确保符合预期
- 游戏内玩家要移除，游戏在线要清零
- 登录要关闭，大区状态要显示关闭
- 后台服务可以不用关闭，零点有结算不能关闭
- 关服操作执行时间长的业务，提前执行关服操作，做好提前时间预估
- 零点是集中执行命令的高峰时间，提前和任务平台团队做好沟通，做好容量协同保障
- 开服后是登陆高峰期，提前和周边团队（登录、鉴权、支付）做好容量评估和协同保障
- 所有操作提前准备好操作流程，按步执行，莫慌张

（2）技术方案的可靠性保障：相关游戏业务的停服和开服流程，提前经过业务运维责任人和业务管理责任人、游戏平台组件责任人评估确认，保证任务执行成功可靠

（3）技术方案的安全性保障：将最终确认停止游戏服务保障技术方案执行细节，交给游戏安全和游戏合规部门评估确认。

（4）技术方案的成本效益评估：通过业务优先级梳理和业务责任人确认，适时协商部分低优先级业务提前停服和延后开服，做到平台任务执行可靠性和平台容量管理成本、运营成本的适度平衡。

（5）技术方案的实施和维护保障：对技术方案的实施执行，提前和任务平台团队做好沟通，保障容量和稳定性，同时协商要求游戏相关中台、系统平台组件负责人提供 On-Call 人员值守，保证整体流程万无一。

在工具可靠性保障上：

（1）自动化工具可靠性保障：腾讯游戏 SRE 一人会负责多款业务的运维工作。通过运维平台系统，设计并编排业务的停服和开服流程，提前准备好定时任务，到了设定时间，任务就自动开始执行，出现任何异常，平台会及时通过告警知会相关业务 SRE，保证整个操作不会手忙脚乱。

（2）容量评估工具保障：针对全业务游戏停服保障，涉及到大批量业务在同一具体时间点附近，通过运维平台执行停服和开服自动化作业，对平台和业务侧造成巨大的任务请求流量和系统调度压力。所以业务 SRE 团队会结合业务的历史的 SLI、SLO 指标，提前做好业务侧的容量评估确认；同时也会协同平台侧的 SRE 同学，重新评估任务执行平台的压力承载，和模块自动扩容部署的工作。

（3）可观测工具可靠性保障：针对全业务游戏停服保障，腾讯

游戏 SRE 快速开发了全方位的多业务可观测数据化视图工具，实时监测业务状态变化。把各业务关键 SLI、SLO 指标标准抽象到一个业务全局大屏视图，作为全局统一监控。通过一个大屏，可以汇总全业务全局查看业务状态。SRE 可以从全局看到每一款业务的是否还有玩家在游戏内，流量是否下降，游戏是否处于关闭状态，网络连接数是否为零等状态信息，辅助业务快速监测业务状态变化和服务稳定性。

（4）AIOps 工具可靠性保障：针对全业务游戏停服保障，腾讯游戏 SRE 通过 AIOps 工具进行业务可用性状态预测判别，AI 异常检测模型开发，完善对业务停服状态数据的预测和判定、完善数据分类、业务异常流量快速定位，辅助判断业务是否已经处于关闭和停服状态，同时通过 AIOps 算法，进行容量预测与容量自动扩缩流程；并且通过 AIOps 智能告警和故障自愈，大大减少了人为的故障处理时间。

（5）数据备份和恢复工具保障：针对全业务游戏停服保障，腾讯游戏 SRE 提前确认数据备份和数据恢复工具的可用性。保证游戏业务在执行停服版本更新前，对业务程序文件进行备份，以及对关键执行记录和执行流程存档。

在突发事件保障上：

（1）突发事件的预警和识别：针对全业务游戏停服保障，通过舆情分析和业务服务稳定性监控预警机制，及时获取和识别突发事件的信息。

（2）突发事件的应急响应和信息沟通：针对全业务游戏停服保障，提前梳理可能发生遇到的突发事件的性质和紧急程度（如业务未按时完成关服操作怎么处理；如平台自动化工具执行异常时，业务是否有其他备份工具可以完成流程操作等），启动腾讯游戏 SRE 应急响应机制，保证相关人员能进行及时故障处理，同时保障信息的实时传递和共享及时向内部和外部发布信息。保障在出现紧急情况的时候场面不混乱，执行有章法

（3）突发事件的处理和复盘：针对全业务游戏停服保障，以执行业务小组为单位，统一推进业务完成整体保障的流程梳理和回溯复盘，并输出改进优化意见存档，在更大的组织单元方面进行经验分享，不断迭代和优化流程。（可参考 3.5.4）

整体上，通过完备的整体统筹保障、完善的技术方案保障、灵活的 SRE 工具可靠性保障、和敏捷的突发事件应急保障，腾讯游戏在要求时间整点完成关服，零延迟；在要求时间内，有序开服，零超时；整个保障零故障，零失误。最终做到对公共事件重大技术保障的实时和全局掌控。

3.6.2.6 示例 2: 交易类大促核心保障流程和方案

1. 保障目标制定

一般会制定业务目标和技术目标，分别介绍如下：

- 业务目标：一般从故障数、稳定性保障体验、线上问题、客诉、舆情、资金安全等角度考虑，主要目的是阐述本次活动保障在稳定性角度的核心目标，例如：0 P1P2 故障、0 资

金安全故障、0 重大舆情事件 等

- 技术目标：一般从技术平台演进、效能提升、成本降低等角度考虑，主要目的是阐述本次活动保障过程中的技术能力带来的重要改变，例如：通过分时调度能力降低 40% 服务器资源、通过全链路压测平台升级带来压测效率提升 30% 等

2. 业务链路梳理

主要是对要保障的业务进行全面的整理，识别出来核心功能链路，使用 MECE 方式，实现不重复、不遗漏的业务链路梳理，并且基于业务重要性程度进行链路分级，挑选出优先级较高的链路进行高保。业务链路的范围为整体保障工作进行了圈定，后续一切工作都围绕业务链路展开

3. 容量评估

基于业务链路的梳理，需要对链路进行技术容量评估，包括入口流量来源、服务接口、上下游服务调用链、数据库依赖、缓存依赖等，重点是识别出来整个链路上的强弱依赖和可以降级的功能节点；合理的业务容量评估是后续所有工作的前提。

4. 资源提报与交付

基于已经评估好的容量模型，可以将容量模型转化成资源模型，包括服务器资源、数据库资源、缓存资源、非标服务资源等，一般也会根据机房部署结构进行分机房的资源模型拆解，最终转化为资源需求文档。SRE 基于资源需求文档准备资源，在固定的时间

内交付对应资源，为后续压测做好准备。

5. 全链路压测验证

资源交付以后，则需要按照既定的容量模型对业务接口进行全链路压测。一般要制定压测方案，阐述清楚压测前准备、压测过程中操作、压测问题跟进方式、压测报告产出等内容，有时也会进行压测前风险巡检，避免一些压测问题重复发生，提前规避风险。在实际压测过程中，可能会多次、多角度的对业务接口进行压测，通过不同的压测模型发压才能使压测更加精准，最终产出的压测报告一般会包括容量指标、服务器 CPU 负载、MEM 负载、RT、数据库负载等信息。

6. 限流、预案整理与演练

完成压测以后，需要对每个压测接口进行限流配置，以保护系统不被预期外的流量打爆。一般限流平台的实现是基于令牌桶方式，现在更多的方向是通过 Service Mesh 进行统一限流管理。不能配置限流的节点，一般会进行有一些业务预案来提前确保功能可用，因此需要对业务链路中的各种预案进行整理。预案必须要有演练的能力，否则在生产环境真实出现异常时无法快速的通过预案恢复，就有可能会造成更大的业务影响

7. 监控、告警整理

可观测性在大促保障中是非常重要的。核心的业务链路都要覆盖好监控，并且不同的业务视角可能会有不同的业务监控大盘，对于技术保障来讲，容量负载大盘是非常有必要关注的。监控到位以

后，需要配置好合理的告警阈值，以便在线上出问题时候能够第一时间通知到负责人进行及时响应处理

8. 作战手册整理

作战手册是指一份在活动上线前，所有保障工作的最终 review 文档，一般会安排作战手册宣讲会议，各方相关负责人都会逐条分析作战手册文档的内容，为活动上线最好最后的准备。作战手册一般会写明监控、应急流程、重要预案等信息。

9. 线上值班与问题跟进

当线上出现异常，需要第一时间拉起应急组织，可以使用作战手册整理的信息，快速评估是否能够通过作战手册的应急预案进行业务恢复，如果不能则需要快速拉起应急会议，相关责任人要第一时间感知到线上问题并作出恢复决策，有必要的情况下需要及时上升，让更高级别的人来评估影响面和做决策。

10. 事后复盘

这是技术保障的最后一部分，需要能够平和、客观的整理出活动保障复盘总结文档，不要遮掩做的优秀的地方，也不要避讳做的不好的地方。复盘是一种回顾工作的好手段，目的是指引下次活动保障做的更好。一般复盘需要各种不同角色的同学都参加，多种角度对同一件保障工作进行复盘才有效果。

3.6.2.7 示例 3：银行类通用重大保障活动

对于银行类通用重大保障活动，包括年终决算、人行关机压测、世博会等重大活动，需要周密部署，落实运维保障职责，避免

发生生产事件造成影响重大活动，引发舆情等情况。SRE 应以最高标准、要求、周密的措施保障系统稳定

1. 保障目标

目标是在重大活动期间内重要信息系统的稳定运行，不发生影响业务的正常运行，使得保障活动平稳度过。

2. 保障范围阶段

依据保障活动不同，保障的范围和阶段会有差异。总体来说，保障系统范围需要覆盖关键基础信息设施、数据中心基础设施、重要信息系统等，保障地域范围为公司集团全辖。其中总分行因系统差异需要单独整理要保障的范围。

对于一项重大保障活动，做到事前准备。按照重大活动时间段划分，SRE 通常划分如部署阶段、准备阶段、正式保障阶段、事后总结报告。

3. 保障组织

依据保障活动重要程度，在期间内成立由科技部门中心领导的组织，或行领导牵头、业务中心和科技中心领导作为小组成员等组织形式。该小组负责监督保障工作的开展、对重要事项决策，听取保障工作方案和进展报告。

4. 保障措施

1）前期准备措施，成立保障组织，并以行发文形式向各部门，子公司做出部署，明确保障目标和要求，部署保障任务和计划。

2）变更封板措施，原则上重要保障期间内全行系统不安排信息

系统生产变更，保障系统稳定。若为监管、第三方要求，必须紧急修复的生产问题或安全加固可特殊处理。

3）加强值守措施

加强 7*24 值守，重保期间内，安排技术总值班 ECC 值守，并增加 SRE 各运维岗位现场值守，包括系统、网络、应用、设备、基础环境等，开发中心需做好远程技术支持。协调重要合作厂商驻守。

4）监控巡检措施

SRE 加强对机房、网络、系统、应用程序的运行状态和资源使用情况的实时监测，提升检查频率和巡检频率，及时预警异常信息解决故障隐患。

5）应急准备措施

总行相关部门落实重要时期保障联络人员和应急响应资源，留守本地待命，确保在岗。重保前 SRE 完善应急手册预案并开展应急演练，对应用、系统、网络、存储、数据库等专项应急预案回顾，开展隔离、重启、分流、双机切换、自愈等快速处置回顾，开展重要系统灾难恢复预案和手册回顾，为应急处置和快速切换做好准备。协调必要外部应急技术资源保障，做好重要备品备件储备。

6）事件处置措施

SRE 贯彻“优先恢复系统服务”原则，当发生故障首先定界恢复，事后再确认根因，按照服务中断时间最短要求选择处置方案，优化压缩应急处置流程，并严格执行突发事件报告机制。

7）厂商支持措施

建立与第三方合作厂商的沟通机制，梳理确认合作厂商应急资源，督促第三方建立事件和问题处置流程以及应急方案，协调重要合作厂商驻守现场保障。建立供应商安全能力评估机制，签订安全责任协议，落实保障资源关键岗位配备足够人员。

3.6.2.8 示例 4：发布会直播通用重大保障活动

电商平台每隔一段时间，都将面临一场大促质量保障的“考试”，大促前也会进行直播类型的发布会，如“苹果新产品发布”。每一场重大促销活动对公司而言是提升业绩的关键，但也意味着业务系统将在短时间内面临流量瞬间暴涨带来的冲击，承受它的“生命之重”，同样的也在考验背后技术团队的应急保障能力。

1. 保障目标

目标是在发布会和发布会后预售期间内重要信息系统的稳定运行，不发生影响业务的正常运行，使得保障活动平稳度过。

2. 保障范围阶段

依据保障活动级别不同区分重要程度（P1-P3），以确定不同的保障级别和关注度。P1 为最高级别，P3 为最低级别，相关划分标准依据不同公司具体需求可以动态调整。

3. 保障组织

SRE：整体统筹，协调跨团队、跨部门工作，参与运营评估，容量规划，识别引流风险，推拉流及导播台可用性保障，确保直播系统稳定

DBA：活动数据库健康度巡检和风险识别，保障直播数据库的

稳定运行

质量团队：发布变更管控，组织拉通风险评审会，安排重保作战室

基础服务：保障相关资源按时交付，保障基础服务和平台运行稳定

市场运营：直播策划、推广，后台配置，新品上架

业务研发：链路压测、业务系统降级

4. 保障措施

1）活动运维保障准备阶段：

（1）流量分析预估

业务相关部门负责根据活动方案规划，对活动期间的流量及订单量进行研判，为活动运维保障资源需求提供基础数据。研判时应依据当前活动策略、促销目标等因素，同时充分参照历史活动数据，进行准确预估。

（2）资源报备及扩容

业务及支付相关部门依据活动预估数据提出基础资源使用和扩容需求，并向 SRE 发起资源需求报备。

资源报备原则：为保证重大活动需要的基础资源能够按需求交付到位，资源需求方应以“提前预估，尽早报备”为原则，提前进行资源需求报备。

基础资源按以下要求提出报备需求时，资源管理方应确保报备资源的充足供给，如有资源紧张的情况应优先保证已报备资源的供

给。

物理机，网络带宽、专线资源、LVS、DNS、SNAT 等需求，资源报备时间不应低于需求使用时间前 30 天；CDN 资源报备时间不应低于需求使用时间前 14 天，线上直播活动 CDN 资源报备时间可缩短至正式开播前 7 天。

特殊情况：资源报备时间不满足最低时间要求时，资源管理方应通过储备资源、资源调配、弹性上云等方式尽力满足需求，确实无法满足时应第一时间与资源需求方进行反馈。

（3）压测

为保证活动顺利进行，确保基础资源满足活动峰值需要，业务及支付相关部门应在活动开始前进行相应的压力测试或直播流测试。

（4）梳理预案

活动运维保障相关各方均应参考日常运维预案和历史重保问题预案，对核心服务及活动相关的应急预案进行梳理，确保活动中可以快速查询、调用和执行。

（5）监控有效性检查

在活动前准备阶段，活动运维保障相关各方均应对核心服务及活动相关业务指标的监控报警覆盖情况、配置情况进行检查，同时对接收人列表进行及时更新，确保监控报警的及时性和有效性。

（6）巡检及隐患排查

活动运维保障相关各方均应加强服务日常巡检。在活动前对核

心服务线上稳定性进行排查，消除单点、安全漏洞及其它风险隐患。对暂时无法消除的风险隐患进行及时通报，并制定应急预案。

（7）变更管控

重大活动（国内）及直播活动期间，运维保障相关部门均应采取必要的变更管控措施。

（8）跨部门拉通

重大活动正式开始前，由质量委负责拉通活动运维保障相关各方，对活动运维保障计划、方案、风险评估、值守安排等关键信息进行同步。

2）活动运维保障执行阶段

（1）活动值守保障

重大活动期间，运维保障相关部门应加强日常巡检，严格执行 Oncall 制度，及时处理监控报警，响应异常反馈；重要直播及重大活动中预估可能产生流量峰值的重要时段，参与保障部门应安排相关人员在公司进行值守，适时安排“作战室”方式集中值守保障。

（2）事故应急处理

重大活动期间，各部门应严格执行故障应急响应流程；发生质量异常事件时，应通过各部门质量接口人对故障情况进行跨部门通报，做到高效协同、快速处理；故障处理应以优先最快恢复服务为第一原则。采用降级、回滚、切量、限流等预案快速恢复服务。

（3）问题追踪

重大活动期间，质量异常事件应及时上报、复盘，进行集中管

理，闭环跟进。

3）活动收尾阶段

（1）资源回收

活动结束后，SRE 依据业务及支付相关部门的资源使用需求，对相关资源进行下线、缩容或释放，保证资源的合理利用。

（2）数据统计

活动结束后，由业务及支付相关部门负责对活动期间的如：“参与人数”/“订单量”/“在线人数”等运营数据进行统计输出；SRE 负责对活动期间如“大秒服务 QPS”/“CDN 峰值”/“带宽峰值”等运维数据进行统计，并输出活动期间整体基础资源使用情况进行归档。

（3）活动复盘

活动结束后，参与运维保障的相关部门应以部门为单位，在 15 个工作日内各自对活动运维保障过程进行复盘，汇总活动数据、跟进质量异常、总结经验教训，更新完善相关预案。

3.6.3 运维琐事的日常管理及优化

3.6.3.1 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。

3.6.3.2 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。

3.6.3.3 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。

3.6.4 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。

3.6.4.1 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。

3.6.4.2 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。

3.6.4.3 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。

3.6.4.4 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。

3.6.5 运营成本分析及优化

3.6.5.1 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。

3.6.5.2 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

3.6.5.2 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。

3.6.5.3 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

率的应该有不同的标准，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

用户只要涉及到资源的下载和更新就会涉及到 CDN 成本。CDN 成本通常能占到运营成本的 TOP3 以内。不同的云厂商 CDN 的计算收费规则不尽相同。国内的厂商大都以 CDN 带宽计费模式为主，海外的厂商以 CDN 流量计费模式居多。

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，既然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“消峰填谷”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。

3.6.5.4 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。适合进行混部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，避免了资源的争强。

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和评估和规划时，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。

3.6.5.5 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。

（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。

3.6.6 混沌工程

3.6.6.1 正常行为定义

混沌工程是一种实验方法，用于测试分布式系统的弹性和容错能力。它通过在生产环境中有意地制造故障，来检验系统是否能够在故障发生时维持正常运行。在混沌工程中，正常定义行为指的是复杂系统在运行过程中表现出的稳定、可预测和可控的行为。在进行混沌工程实验时，需要先定义系统的正常行为，然后在实验过程中不断观察和监测系统的表现，以确保系统在故障发生时能够恢复到正常定义行为。这有助于提高系统的弹性和容错能力，从而增强系统的稳定性和可靠性。衡量系统正常是否的关键指标为稳态指标。

稳态指标是系统在故障发生时是否受到影响以及影响的程度的体现，作为混沌工程判断是否回滚混沌操作的重要依据。举例，在游戏登录模块发生网络分区异常的时候是否会影响到游戏功能，如果模块集群本身是高可用的，能容忍少量节点不可用的情况整体服务可用，那么登录功能的可用性理论上就不会受到影响，反映登录可用性的指标就是我们需要的稳态指标。

3.6.6.2 设计和实施混沌实验

需要明确实验的目标，是为了验证系统的哪些方面。高可用也可以分为多场景的高可用，是系统本身的多实例异常切换，还是自身的限流降级是否生效，又或者是依赖第三方发生故障时自身系统是否能正常熔断等等。然后是控制最小爆炸半径，在一个受控的环

境中开始实验，如开发或测试环境。这可以帮助你理解实验可能产生的影响，而不会影响到生产环境。在受控环境中运行实验，并监控系统的响应。收集数据，以便在实验结束后进行分析。注意！在设计混沌实验时，应确保实验的安全性和可控性，避免对生产环境造成不可预期的影响。

其次是手段，根据系统的架构和潜在的故障点，设计实验方案，模拟服务器宕机/网络延迟/数据库故障等。

1．网络类故障注入

根据网卡、IP、端口等信息注入网络丢包率或网络延迟。

2．硬件类故障注入

模拟硬盘、CPU、内存等硬件设备故障导致的服务器异常或重启。

3．性能类故障注入

CPU 占用：指定核提升 CPU 使用率到特定值；

内存占用：通过额外占用系统的内存，模拟系统无法分配新内存的状况；

磁盘占用：模拟大量磁盘 IO 操作及磁盘写满。

4．数据库/中间件故障注入（MySQL 为例）

模拟 MySQL 主从故障切换时，观察程序是否会自动重连到正确的实例；

模拟 MySQL 短暂网络抖动后，观察程序是否会重连，以及业务逻辑是否受影响；

模拟集群内部分节点异常，数据库是否能正常工作；

模拟MySQL实例内存/磁盘IO繁忙/CPU抢占/磁盘打满等情况，观察业务的稳态指标表现。

3.6.6.3 监控和分析实验结果

在实验过程中，收集和分析系统性能数据，以评估系统在故障情况下的表现。根据实验结果评估系统在面对故障时的弹性。如果系统在实验中表现良好，说明它具有较好的弹性。如果系统在实验中表现不佳，可能需要进一步调查原因，并考虑采取措施来提高系统弹性。比较实验结果和基线，看看系统的行为是否符合预期。如果系统没有按照预期恢复，那么你可能需要调查原因，并修改系统以提高其弹性。

数据的来源可以是定义的稳态指标，也可以是人工测试获取到的直观用户体验，客户端日志，服务端日志，服务端链路跟踪（trace）等综合分析。需要关注的是混沌的持续时间，以便于精准定位到对应时间窗口的相关信息。

结合监控图表展示，可以实现混沌过程及效果可视化。例如模拟CPU升高会拉到对应爆炸半径机器的CPU曲线图。可以并行配置稳态指标，一边模拟故障一边观察业务的指标变化，相较于过去的繁琐沉重的日志分析，混沌工程将故障对程序的影响可视化，程序对我们不再是黑盒，而是白盒，是否有影响，影响范围有多广，一目了然。

3.6.6.4 优化和修复问题

根据实验结果，找出系统薄弱环节，优化和修复问题，提高系统的弹性。在实施改进计划后，持续跟踪和监控系统的表现，确保改进措施取得了预期的效果。如果需要，可以进行进一步的混沌实验来验证改进效果。

通常情况下，系统在上线前会经过功能测试，所以程序在大多数情况下正常情况下的功能是会正常工作，我们需要主动模拟少数异常的环境，观察程序在异常发生时候的表现是否符合我们预期。程序在设计之初就会在架构层考虑冗余和高可用，但是是否真如我们所预期的表现需要模拟验证。

故障无法避免，特别是分布式应用，在设计之初就要考虑到网络分区带来的影响，不同程序的场景会根据 CAP 原则做出可用性（CA）和数据强一致性（CP）的取舍，从而表现出不同现象。通过模拟故障，发现程序不符合预期的表现，从而优化程序，提升系统的高可用性。

3.6.6.5 持续迭代和改进

持续监控系统的性能和可靠性，以便在问题再次出现时能够及时发现和解决。同时，定期运行混沌实验可以帮助你提前发现并解决潜在问题。定期进行实验和优化，以应对系统不断变化的需求和环境。

将混沌演练的场景固化为模版，集成到业务的 CI 流水线，每次正式版本发布自动进行混沌测试，混沌工程自动进行稳态指标判

断，从而自动化检测当前版本是否带来高可用隐患。相较过去手动模拟故障的演练方式，混沌工程具有可复用，自动化执行，智能判断的特点，和 CI 流程集成后变得更加轻量级，使得故障模拟，高可用检验，告警巡检不再是重成本的工作，可以伴随着每次的版本编译后部署自动检测，自动生成版本高可用打分报告，使得故障演练可以脱离运维和测试，由研发人员一键触发闭环。

3.6.7 应用服务 SLI/SLO

3.6.7.1 什么是 SLI/SLO

（1）SLI：由业务相关程序、客户端、中间件产生的监控指标，每个指标有明确的定义、计算方法，一个功能会包含多个 SLI。

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：用于在时间维度上衡量服务稳定性的上层指标，一般分功能来进行建设，下层由具体功能用户体验层的监控指标（SLI）构成，SLO 的最终值是由各个 SLI 的实际情况所共同决定的。

在特定情况下，可以考虑将 SLO/SLI 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。

3.6.7.2 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：VEALT，从 V（容量）、E（错误）、L（延迟）、A（可用

率）、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，6 个维度，并基于监控系统计算出相关指标的值。

维度信息

维度	含义	枚举值
velat	VELAT 方法论，分别代容量、错误、延迟、可用率、工单	Volume Error Latency Availability 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。

3.6.7.3 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，推动运维进行布署优化，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

3.6.8 持续改进

3.6.8.1 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，给于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）整个团队维持在同等配备的情况下是否可以接手更多活更大的服务

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。

3.6.8.2 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），可用于持续的改进衡量标准。因为我们可以希望这个时间尽可能的长，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。

3.6.8.3 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，如何持续的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的嵌入到研发过程，在产品研发过程就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。

3.6.8.4 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的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面，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、容器平台上的贡献记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的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，可以客观的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。

3.6.8.5 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 工作时可以持续改进的。接下来就介绍一下这个整体的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，不仅仅时 SRE 团队，也包含开发团队、运营团队、产品团队等，需要全面的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。

3.7 平台工程

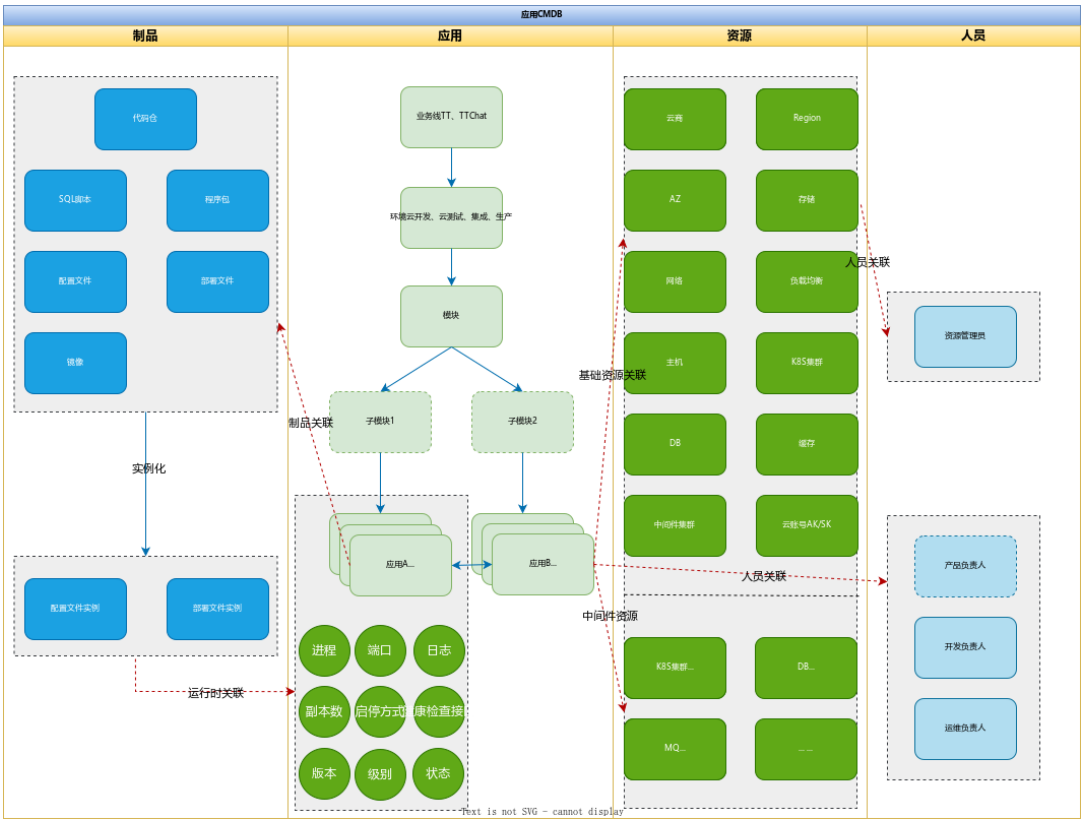
3.7.1 标准应用平台工程建设

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。

3.7.1.1 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。

3.7.1.2 统一资源供给 哲哥

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。

3.7.1.3 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：执行了就都会开始到下一任务（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。

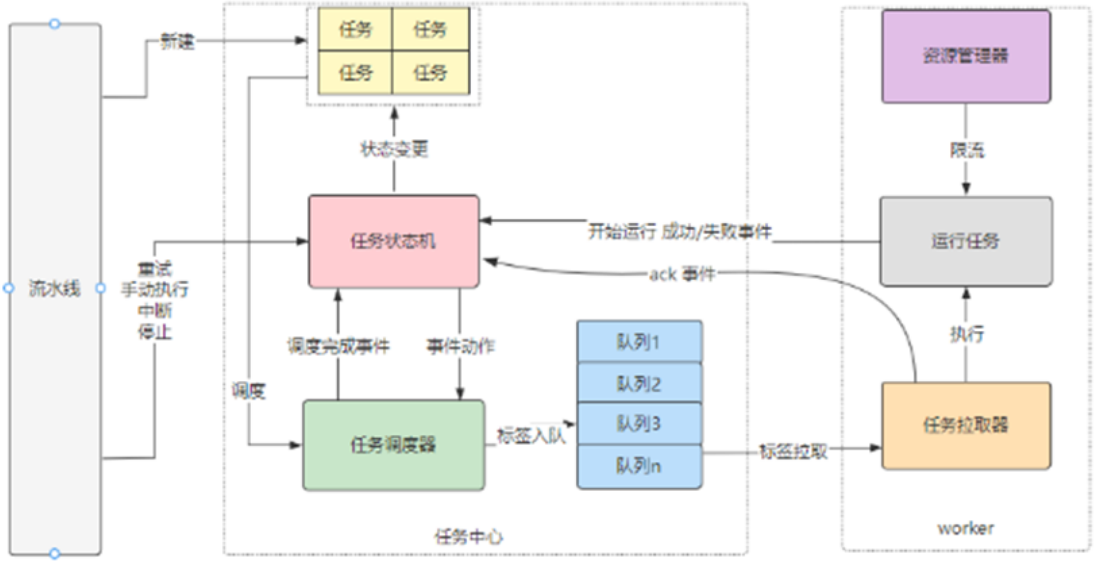
（5）任务添加默认都是阶段内最后一个，你可以拖动任务来改变任务顺序。

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

3.7.1.4 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、是否是灰度分组、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。

3.7.1.5 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，也可用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。

3.7.1.6 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。

3.7.1.7 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

3.7.2 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。

3.7.2.1 总体设计

如图所示，面向异构应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。

3.7.2.2 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：不同的编程语言，需要提供不同的开发框架，当前主流的编程预言师 Python、Golang、Java 等。可以根据平台工程开发者的技术特性，按需提供必要的开发框架。

（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。只要提前预定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，秘钥信息，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。

3.7.2.3 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”即就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。

3.7.2.4 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，与周边系统进行实施同步与修订。

（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，并将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，尽量按照同一管理模型纳管，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，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、流水线模版贡献者、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．其它各类平台

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。

3.7.2.5 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的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

基础 SaaS	功能描述
	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订制化界面，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	建设全链路可观测能力，整合 CMDB、trace、log、

	中心	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

3.7.2.6 服务管理

在异构应用的平台工程建设中，涉及到一些公共的、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。

3.7.2.7 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，必需要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，以及如何全方位的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，如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、秘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。

4 附录

4.1 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

4.2 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。