O'REILLY®

Compliments of Google Cloud

事故管理剖析

Anatomy of an Incident

Google's Approach to Incident Management for Production Services

谷歌对生产服务做事故管理的方式

Ayelet Sachto & Adrienne Walcer
with Jessie Yang

REPORT

译者：刘征

本文档非 Google 官方翻译，内容解释权归 Google 所有。

Anatomy of an Incident
by Ayelet Sachto and Adrienne Walcer, with Jessie Yang

Printed in the United States of America.
Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.
O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (http://oreilly.com). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.
Acquisition Editor: John Devins
Development Editor: Virginia Wilson
Production Editor: Beth Kelly
Copyeditor: Audrey Doyle
Proofreader: Piper Editorial Consulting, LLC
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea
January 2022: First Edition
Revision History for the First Edition
2022-01-24: First Release

978-1-098-11372-8
[LSI]

目录

第一章：概述

> 如果没有猜错的话 ——— 在接下来的几周里，我们将会在个人和工作上面临巨大的压力，我们需要快速应对各种不断变化的状况。但我们已经为应对危机准备了十多年，并且已经做好了准备。在全球比以往任何时候都更需要信息、沟通和计算的时候，我们会确保 Google 能够提供帮助。
>
> ——Benjamin Treynor Sloss，Google 站点可靠性工程团队工程副总裁，2020 年 3 月 3 日

中断是不可避免的（这确实让人沮丧）。作为科学家和工程师，你们需要从长远角度看待问题，设计系统以实现最佳的可持续性、可扩展性、可靠性和安全性。但是，你们只能基于现有的知识进行设计。在实施解决方案时，你们也无法完全预知未来。你们不能总是预见到下一个零日事件、头条热搜话题、天气灾害、配置管理错误或技术变革。因此，你们需要随时准备应对这些可能影响系统的事件。

谷歌在过去十年中最大的技术挑战之一是 COVID-19 新冠疫情爆发所带来的。新冠疫情引发了一系列快速出现的事故，我们需要迅速应对以继续为用户服务。我们必须大幅提升服务容量，让员工在家高效工作，并在供应链受限的情况下找到新的服务器修复方法。正如 Ben Treynor Sloss 所言，谷歌能够在这一系列重大变故中持续提供服务，因为我们已经为此做好了准备。十多年来，谷歌积极投资于事故管理，这种准备是提高事故响应能力最重要的事情。准备工作能增强恢复力。恢复力和处理中断的能力是衡量技术长期成功（以数十年为单位）的关键因素。除了做好工程设计，还需要时刻准备应对业务服务的中断。

恢复力是公司运营的关键支柱之一。因此，事故管理是公司必不可少的流程。事故不仅对客户有影响，也对操作人员造成了负担。事故带来压力，通常需要人工干预。因此，有效的事故管理应该优先的考虑：预防性和主动性的工作，而不是被动应对。

我们知道管理事故压力大，找到和培训响应人员也很困难；我们也知道有些事故不可避免，中断会发生。与其问“如果发生事故你会怎么做？”，不如问“事故发生时你会怎么做？”。通过减少这种模糊性，不仅能减轻操作人员的负担和压力，还能缩短解决时间，减少对用户的影响。

我们写这份报告（白皮书）是为了总结一份：技术事故响应实践的指南。我们首先构建一些讨论事故的常用语言，然后深入探讨如何鼓励工程师、工程领导者和高管在组织内部思考事故管理。我们旨在涵盖从准备事故、响应事故、恢复事故，到保持健康组织的所有内容，以便大规模地应对各种突发情况。让我们开始吧。

什么是事故?

事故(incident)是一个含义广泛的词。其含义可能因不同群体而异。例如，在 ITIL 中，事故是指任何计划外的中断，如工单、报错或告警。无论这个词如何使用，重要的是要在其特定的定义上达成一致，以减少信息孤岛，确保每个人都在说同一种语言。

在谷歌，事故是指：

- 被升级的问题（因为影响太大，而无法单独处理）
- 需要立即响应的问题
- 需要有组织的进行响应的问题

有时，事故可能由服务中断引起，即服务在一段时间内不可用。中断可以是计划内的，例如在维护窗口期间系统故意不可用以进行更新。如果中断是计划好的并且已通知用户，则就不算是事故——并不需要开展立即、有组织的响应的事情。但通常情况下，我们指的是由未预见的故障引起的意外中断。大多数的意外中断都是事故，或最终会发展成为事故。

事故可能对客户造成影响。它们还可能造成收入损失、数据损坏、安全漏洞等，这些都可能影响客户。当客户受到事故影响时，他们对你的信任可能会动摇。因此，你需要避免过多或过于严重的事故，以保持客户满意；否则，他们会选择离开。

频繁的事故也会影响事故响应人员，因为处理事故的压力很大。找到具备适当技能来处理事故的站点可靠性工程师 (SRE) 既具挑战性又昂贵，因此你不希望通过让他们只负责事故响应来使其疲惫不堪。相反，你应该通过主动预防事故来提供他们技能成长的机会。在这份报告的后面，我们将进一步讨论这一点，以及减少压力和改善值班健康的方法。

并非所有问题都是事故

区分事故和中断很重要，同样重要的是区分指标、告警和事故。如何区分指标和告警，告警和事故？并不是每个指标都会成为告警，也不是每个告警都是事故。为了帮助你理解这些术语的含义，我们将首先讨论监控和告警在维护系统健康中的作用。

监控

监控是保持系统健康的最常见方法。根据《SRE Google 运维解密》的定义，监控是指收集、处理、汇总和展示系统的实时定量数据，例如查询计数和类型、错误计数和类型、处理时间和服务器在线时间。监控是一种度量。

在度量方面，我们建议采取以客户为中心的方法来制定服务质量目标 (SLO；在第 26 页的"减少事故的影响"中有更详细的讨论) 和优化客户体验。这意味着收集能准确反映客户体验的指标，并尽可能收集多种度量，如黑盒、基础设施、客户端和应用程序指标。使用不同方法测量相同的值可以确保冗余和准确性，因为不同的测量方法各有优势。以客户为中心的仪表板也能很好地反映客户体验，对于故障排除和事故调试至关重要。

重要的是，要专注于度量可靠性和对用户的影响，而不是度量已确认的事故个数。如果专注于后者，员工可能会因为害怕被惩罚而犹豫声明事故。这可能导致事故声明延迟，不仅浪费时间和丢失数据，还因为事后处理效果不佳。因此，声明事故并及时关闭比事后补救要好。

在这方面，有时人们会将可靠性和可用性混用，但可靠性不仅仅是"服务可用性"，特别是在复杂的分布式系统中。可靠性是指在大规模下提供一致服务水平的能力，包括可用性、延迟和准确性等方面。这在不同服务中可能（也应该）有不同的体现。例如，YouTube 和 Google 搜索的可靠性是否相同？根据你的服务，不同用户的期望会有所不同，可靠性也可能有不同的定义。

一般来说，如果系统的中断更少、更短、更小，它就更可靠。因此，最终取决于用户能容忍的停机时间。采用以客户为中心的方法，用户定义了你的可靠性。因此，需要尽可能接近地度量用户体验。（我们在第 26 页的"减少事故的影响"中对此进行了更详细的讨论。）

告警

我们已经讨论了系统健康监控。现在让我们谈谈监控的关键组成部分：告警(Alerting)。当监控发现系统行为异常时，会发送一个信号，这个信号就是告警。告警可能意味着两件事：某些东西已经损坏，需要有人修复；或者某些东西可能即将损坏，需要有人检查。紧急程度——即何时需要采取行动——应指导你选择如何响应。如果需要立即采取（人工）行动，应发送紧

急通知。如果在接下来的几个小时内需要人工行动，应发送告警。如果不需要立即行动——例如信息是用于分析或故障排除——则信息保持为指标或日志的形式。

需要注意的是，告警的方式可能因组织偏好而异。例如，它可以在仪表板上显示，或以工单形式呈现。在谷歌，通常采用后者；监控系统在 Google 问题追踪器中创建一个具有不同优先级的"错误-bug"，这就是我们的工单形式。

现在你已经了解了基础知识，让我们深入探讨可操作的告警。

可操作告警的重要性

如前所述，当特定条件满足时，告警会触发。但你必须谨慎，只针对真正重要和可操作的事项发出告警。考虑以下场景：作为当班人员，你在凌晨 2 点被呼叫，因为过去 5 分钟内 QPS 增加了 300%。这可能是一个流量波动大的服务，有时流量稳定，但偶尔会有大客户发出大量查询。

这种情况下半夜叫醒你有何意义？实际上毫无意义。这个告警是不可操作的。只要服务没有崩溃的风险，就没有必要叫人起床。查看历史数据会显示服务需要应对这样的流量峰值，但这些峰值本身并不构成问题，不应生成告警。

再考虑一个更微妙但更常见的可操作告警问题。你的公司需要每晚备份生产数据库，因此设置了一个每四小时运行一次的 cronjob 进行备份。一次备份由于瞬时错误失败——用于备份的副本发生了硬件故障，并被负载均衡器自动移出了服务模式——但随后几次备份都成功了。结果还是创建了一个工单。

因为一次备份失败而创建工单是不必要的。这只会产生噪音，因为系统在无人干预的情况下自行恢复了。

这种情况经常发生。虽然最终只需简单地关闭工单并附上"处理时已经好了"的信息，但这种行为存在一些问题：

琐事 *(toil)*

有人不得不花时间查看工单、分析图表和报告，最终发现他们不需要采取任何行动。

告警疲劳 *(alert fatigue)*

如果 95% 的"数据库备份失败"告警只是被简单关闭，实际问题被忽视的风险会显著增加。

如前所述，事故是具有特定特征的问题。告警只是一个信号，表明可能有事故正在发生。你可能会遇到很多告警但没有实际事故。虽然这种情况不理想，但并不意味着你需要启动正式的事故管理技术；也许这是计划中的维护，你预期会收到这些告警。

同样，你也可能有事故但没有任何告警——例如，你从安全团队得知他们怀疑生产系统被入侵，但你的团队没有触发任何相关告警。

实际上，人们对告警和事故的感知有所不同：

- 正式的事故管理比简单处理告警要更有压力。
- 经验较少的响应者比经验丰富的响应者更不容易启动事故管理流程。
- 事故更可能需要额外的团队资源，因此其他团队成员可以更早判断是否需要介入。

这种情况不仅限于你的团队，事实上，它适用于整个组织。

告警通常比事故多。获取告警的基本指标（例如，每季度有多少告警）是有用的，但事故需要更详细的分析（例如，上季度的五个重大事故都是由于新功能在预生产环境中测试不足）。你不希望这些报告被所有收到的告警信息淹没。考虑到受众——告警指标主要对团队有用，而事故报告可能会被高层阅读，因此需要管理适用的范围。

希望这能澄清何时你可以更自信地说"这不是事故"。然而，这也带来了一个二分法：如果有些事情不是事故，那意味着有些事情是事故。你该如何处理这些事故？我们将在下一节探讨。

事故管理生命周期

最佳的事故管理不仅仅意味着尽可能快速地处理事故。良好的事故管理意味着关注事故的整个生命周期。在本节中，我们讨论一种系统化的事故管理方法。将事故视为系统中持续存在的风险。处理这些风险的过程称为事故管理生命周期。事故管理生命周期涵盖了准备、响应、恢复和缓解事故所需的所有活动。这是运营服务的持续成本。

所谓生命周期，我们指的是事故存在的每个阶段。这些阶段如图 1-1 所示，具体如下：

准备 Preparedness

> 包括公司或团队为应对事故发生而采取的所有措施。这可能包括工程上的安全措施（如代码审查或发布流程）、事故管理培训，以及识别错误的实验或测试演习。这还包括设置监控和告警。

响应 Response

> 当触发因素导致潜在风险变为实际问题时的应对措施。这包括响应告警、决定问题是否是事故，并与受影响的人员沟通。

缓解和恢复 Mitigation and recovery

> 使系统恢复到功能状态的一系列行动。这包括为了避免影响或防止影响扩大的紧急缓解措施。恢复阶段还包括进行事后分析和反思，撰写事后报告。事后报告是一份关于事故的书面记录，包含采取的措施、影响、根本原因和防止再次发生或减少未来影响的后续行动。

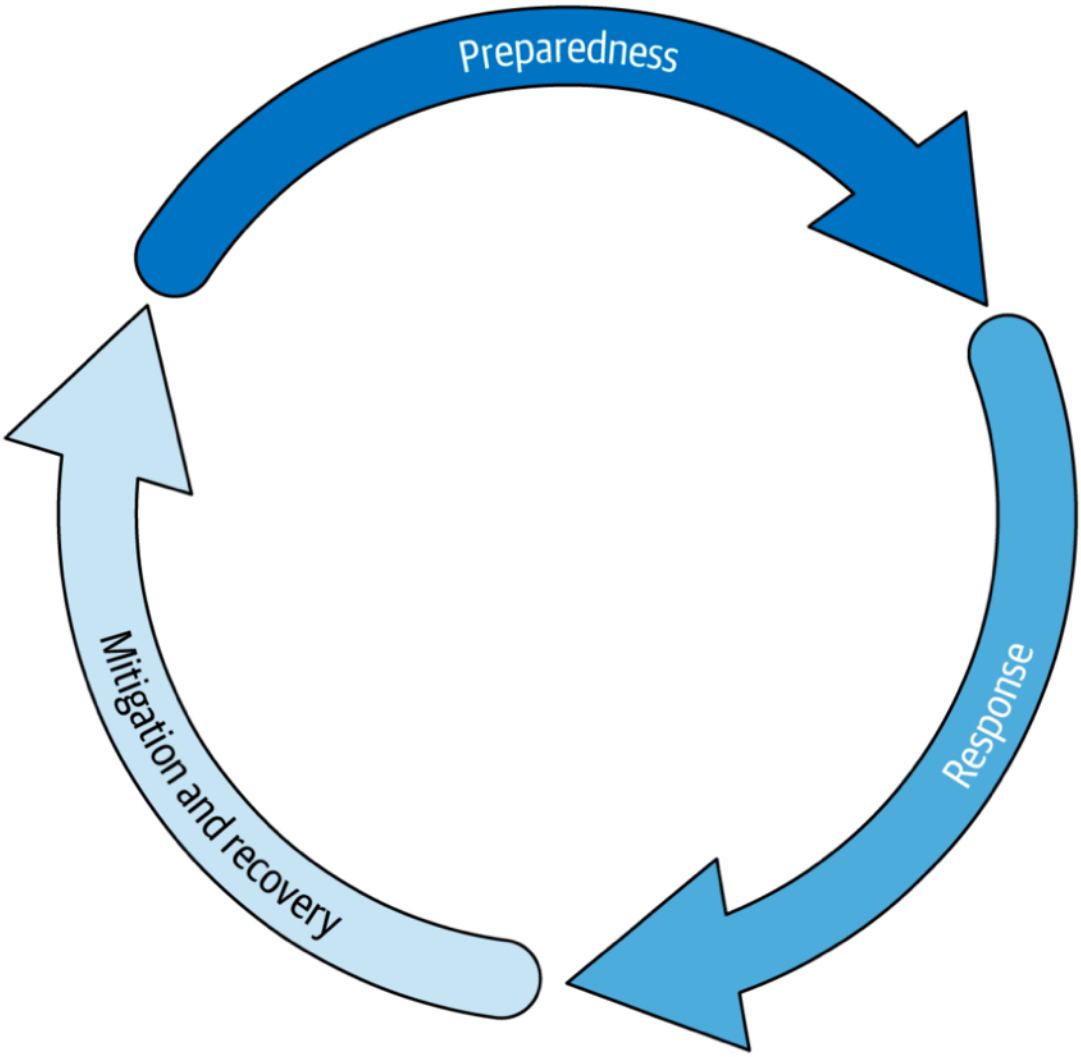

图 1-1. 事故管理生命周期

一旦恢复阶段结束，你将重新进入准备阶段。根据系统的复杂性，所有这些阶段可能同时进行——但可以确定的是，至少总有一个阶段在进行中。

第二章：实战演习事故响应准备

我们已经讨论了管理事故的三个阶段和事故管理生命周期。现在，让我们讨论如何实战演习事故管理，以便在实际事故发生时，确保我们已经做好了准备。

灾难角色扮演和事故响应演习

为了增强恢复力，测试和演练事故响应准备是非常有价值的。我们建议在团队中进行灾难角色扮演来训练事故响应的全过程。在谷歌，我们通常称之为“厄运之轮 (Wheel of Misfortune)”。这是一种重现过去在生产环境中曾经遇到的真实事故场景的一种方法。

定期进行事故响应演习有许多现实的好处。在谷歌灾难恢复测试 (DiRT) 计划的早期，有些测试被认为风险过高而无法执行。然而，多年来，通过专注于这些由高风险测试所暴露出来的领域，许多暴露出来的风险已经得到了彻底解决，以至于这些测试现在是自动化执行的，并且被认为是无聊的测试。

如果想要实现这一点，效果既非立竿见影，过程也不会轻松愉快——这需要投入时间，需要多个团队付出巨大的协同努力；但我们已经能够极大的降低全球系统中的重大风险，以至于“风险只是另一个定期运行的自动化测试”。

定期测试

执行定期的测试具有很多显著的好处。多年来，谷歌一直在进行 DiRT 测试，以发现并修复生产系统中的问题。随着团队不断测试他们的服务，高风险测试的数量减少了。这实际上是个好迹象——团队已经使系统更加具有恢复力，以至于发现暴露系统的弱点，也变得越来越难。

测试失败——由于某种原因导致的测试失败——也变得更加罕见。即使发生，这些系统也往往能以预期的方式失败，并因此能得到迅速缓解。故障响应者现在更加自如地启动故障应急预案，能够在压力下保持头脑冷静，并且由于这些测试，撰写事后复盘报告的次数也减少了。谷歌多年的努力已经得到了回报——大家的心态已经从“灾难测试是对我个人的挑战”转变为“灾难测试是大家共同的任务”。

细化的测试和自动化

测试正在逐渐从解决纯粹的技术问题（例如"我们是否知道如何从完全损坏的数据库中恢复？"）转向更细化的"修复流程"的一系列挑战。

技术测试较易讨论和自动化：基本上就是编写一些代码来执行一系列命令，并检查预期的响应。而想要找到有问题的流程，则更加困难——例如"只有一个人有权批准这个，但他们并不回复电话/邮件"——尤其是对于那些那么不经常执行的陌生流程。

准备响应者

进行事故响应测试——即使只是理论测试——也可以帮助我们识别出有问题的流程，评估其概率和风险因素，并增强响应者的信心。即使测试未按计划进行，你也能更好地发现事故响应流程中的弱点。事故响应者也将会更好地做好在技术、心理和情感上的准备，以应对未来会发生的实际事故。

情感上的准备并不可低估。如前所述，事故管理会给响应者带来巨大压力，导致疏忽大意、反应变慢和判断力模糊。压力还可能引起焦虑、疲劳、高血压和睡眠质量差等健康问题。

进行事故响应测试不仅可以减少这些不良影响，更重要的是识别暴露出这些问题，以便采取纠正措施——如请求帮助、休息，甚至完全移交事故的处理。管理人员和领导也应该：时刻关注响应者的压力、疲劳和倦怠的迹象，并尽可能的提供必要的帮助。

编写事故响应测试

编写事故响应测试(Writing Incident Response Tests)的一个良好起点是：查看并回顾最近发生的事故。在谷歌，我们在每次事后复盘分析中都会问这些标准问题：

- 出了什么问题?
- 哪些方面做得好?
- 我们在哪里是凭运气?

首先查看出了什么问题，因为这是需要改进的地方。这些往往是容易解决的具体问题——例如，监控发现了问题但并没有通知任何人。一旦识别并修复了问题，就需要测试这个修复结果。这一点不能忽视：仅仅修复问题是不够的；修复可能不完整，或在其他地方又引起了回归。

在测试正确性时，从小而简单的测试案例开始。随着对响应过程信心逐步增加，就可以开始研究更复杂的问题，包括那些不完全是技术性的（如人员流程方面）。

当这些小范围测试进行了一段时间后，开始查看“我们在哪里是凭运气？”的那些方面。通常，这些问题更微妙，解决它们可能也不会容易。再次，从小处着手，将问题分解成更小、更易解决的部分。

这些测试应保持缓慢的进行，但要保证稳定的节奏——既不想让团队淹没在测试工作中，也不想失去进度。举例来说，每四周进行一次一小时的测试，要比花费 10% 的运营预算在这些测试上更容易被接受。随着这些程序的发展和测试价值的显现，你会找到适合的测试频率和深度。

第三章： 扩展事故管理响应

我们已经讨论了通过事故响应演习、角色扮演和定期测试来练习事故响应准备。这些策略能帮助你在真实事故发生时做好准备并开始管理（见第31页"建立有组织的事故响应程序"）。但是，当你的组织开始扩展时【译者注：扩展是指将事故管理的流程在更大范围的业务系统上逐步应用推广的过程。】，如何管理事故呢？在本节中，我们讨论如何在大量的系统之上扩展事故管理（流程/实践）。

在 Google，我们为所有系统提供了最佳的事故管理覆盖。Google 规模庞大，每年处理超过 2 万亿次搜索，需要大量的数据中心、至少一百万台计算机和超过 80000 名员工。所有这些活动都通过一个庞大且高度互联的系统系（ system-to-system 简称 SoS）进行，依赖其技术堆栈保持生产运行。支持这个技术堆栈意味着适当的人员随时待命，以便在问题出现时进行故障排除和修复。这些人员是我们站点可靠性工程团队中的响应人员，他们为系统提供事故管理覆盖，并在事故发生时进行响应。

组件响应者

在站点可靠性工程团队中，我们还拥有组件响应者，他们负责 Google 技术基础设施中某个组件或系统的响应（图 3-1）。

图 3-1. 组件响应者

组件响应者是某个单一系统的专家，精通该问题领域，是优秀的故障排除专家，并在危机期间实践缓解策略。他们可以持续访问执行紧急响应所需的工具和系统，能够很好地应对压力，并在危机期间保持清晰的思路。

单个组件响应者的责任范围有限，这使他们能够深入了解其领域及相关系统。这些响应者是防止故障从一个组件蔓延到整个堆栈的第一道防线。这些单独的组件比整体的系统体系级 SoS 堆栈要小，正如我们将在下面描述的"系统体系（SoS）响应者"一节中讨论的那样，通常

具有明确且独立的系统边界。因此，可以设置合理的监控和告警机制，使组件响应者始终了解其系统的故障模式。

当技术堆栈的范围超出一个人可理解和维护的能力时，我们将技术堆栈拆分，以便多个响应者可以分别对整个堆栈的单个组件提供覆盖。随着时间的推移，这些组件变得更加复杂，并进一步分解。通过保持有限的范围，主要响应者可以在任何给定时间解决小范围内的问题。然而，也存在风险，即忽视了跨多个组件的生产故障，或者如果问题超出其专业范围，则无法为组件响应者提供足够的支持。

例如，假设一个底层故障在技术堆栈的显著部分发生级联效应。这种级联效应的速度超过了人类自我组织的速度。在一次影响范围广泛的事故中，我们很快就会达到每个组件团队都被呼叫、分配了响应人员并管理自己的状态。这些组件团队并行工作，但这些响应人员可能彼此并不知情（图 3-2）。其中一个响应人员处理的事故可能是根本原因，而其他则是后果。但究竟是哪一个呢?

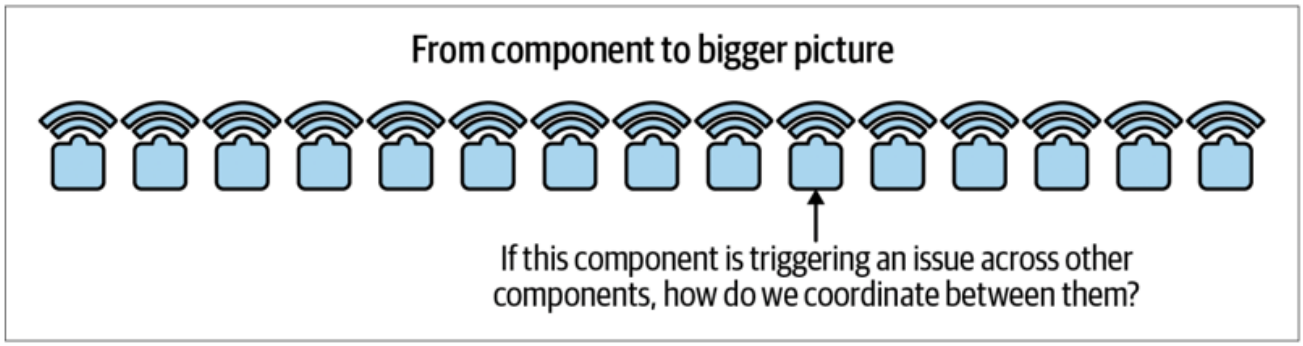

图 3-2. 从组件到更大的场景

在一个足够大且复杂的技术堆栈中，一个主要响应人员几乎不可能同时驱动缓解并维护所有依赖关系和被依赖关系的状态。为了缓解这种风险，除了出色的组件响应人员之外，我们还建立了一个二级响应人员的结构。我们在 Google 称这些二级响应人员为系统响应人员，接下来我们将讨论这一部分。

系统响应人员

系统响应人员（SoS 响应人员）负责处理跨多个组件系统、跨系统边界或复杂情况的事故。这些 SoS 响应人员经过专业培训，具有适当的权限和地位，并有权领导有组织的协调响应。他们是第二道防线，更全面地关注问题，并在应对分布式计算故障时提供关键优势支持（图 3-3）。

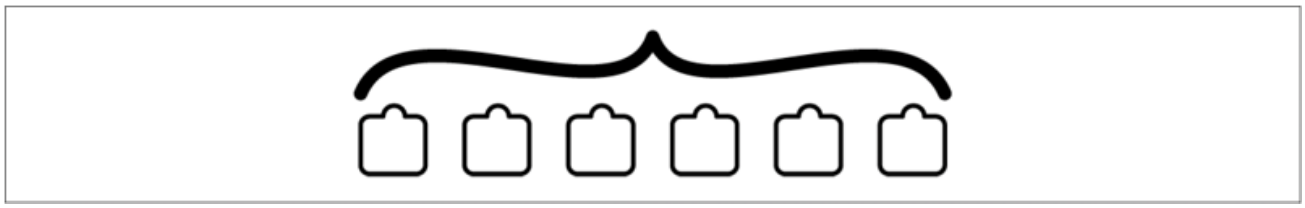

图 3-3. 系统响应人员

我们认为 SoS 响应人员是多系统事故管理者，技术全能，关注整体；他们在处理需要更广泛视角的事故方面有专业知识。通常，这些事故需要多个团队的参与；例如，一次重大系统级 SoS 故障会导致许多服务中断。这些事故可能会引发或已经引发下游故障，并可能扩展到服务边界之外。此外，这些事故可能已经持续了 30 分钟或更长的时间，且没有解决迹象，影响客户。

SoS 响应人员适合应对这些影响广泛的事故，因为他们知道如何组织他人并掌控复杂局面。他们还擅长诊断系统行为，找出根本原因，专注于扩展响应并广泛沟通事故情况。

在 Google，我们有两种类型的 SoS 响应人员。尽管每种类型都有其独特的功能，但它们经常协同工作：

产品专注的事故响应团队（IRTs）

> 这些团队保护特定产品领域的可靠性。例如，广告 IRT 和 YouTube IRT。并不是每个产品领域都需要事故响应团队，但随着产品不断推出新功能、变得更加复杂，并积累了技术债务，这些团队将非常有帮助。这些团队的成员不一定了解产品堆栈的每一个细节，但他们了解产品的整体运营和依赖关系。

技术事故响应团队（Tech IRT）

> 这是我们最广泛关注的事故响应团队。该团队专注于跨产品的事故、责任不明的事故或根本原因不清的普遍事故。Tech IRT 是我们的最后一道防线。成员是资深的 Google 员工，他们至少在两个不同的团队中担任过组件响应者，广泛了解系统运行，最重要的是，他们具备出色的事故管理技能。

Tech IRT 的成员继续为原团队工作，同时轮流进行全球 24/7 的值守/值班。他们能够在这些重大紧急情况下继续工作，因为他们经常练习这项专门技能。

Tech IRT 成员每年两次接受为期两周的生产培训，深入了解系统运行和故障的细节。他们还需要每季度展示有效使用紧急工具的能力。

图 3-4 描绘了 Google 的事故响应组织架构。随着架构级别的增加，产品日常功能的细节变得更加抽象。每个角色同样重要——金字塔的每个后续级别都承受较少的寻呼负载。如果组件响应人员无法解决问题且威胁到产品稳定性，他们可以将问题升级到产品专注的 IRT。

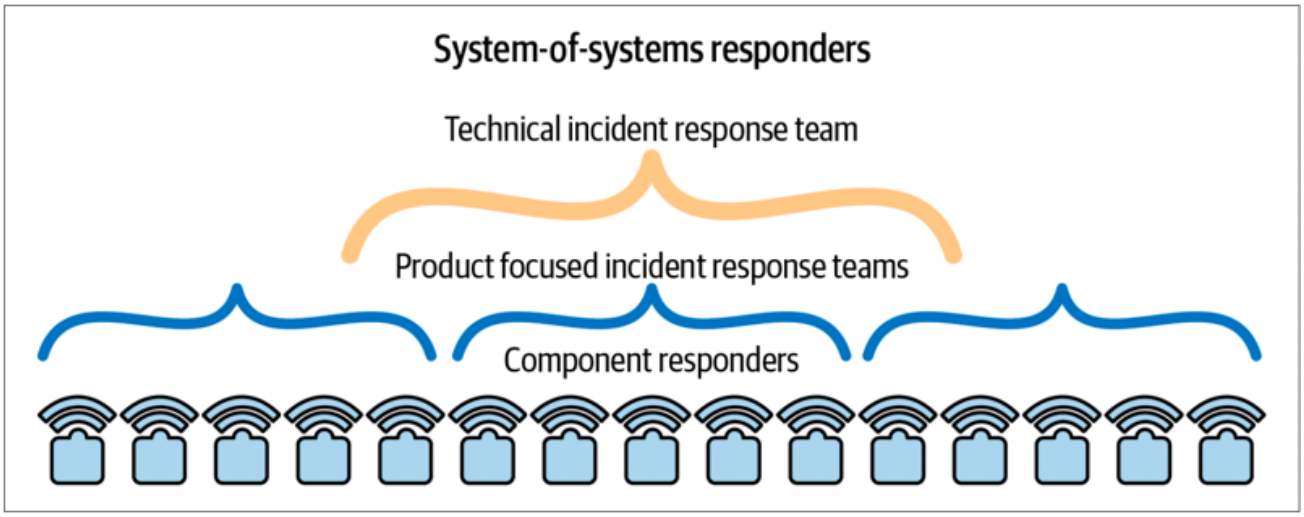

图 3-4. 事故响应组织架构

如果一个问题威胁到多个产品，或通过共享基础设施的解决方案可以更快缓解，Tech IRT 将被激活，作为所有下级问题的升级点，负责最广泛范围的操作。

那么，是什么使得这个组织架构能够无缝运行呢？答案是共同的协议、信任、尊重和透明度。接下来我们将详细探讨这些。

事故响应组织架构

成功的事故响应组织有四个特征：统一协议、信任、尊重和透明（见图 3-5）。

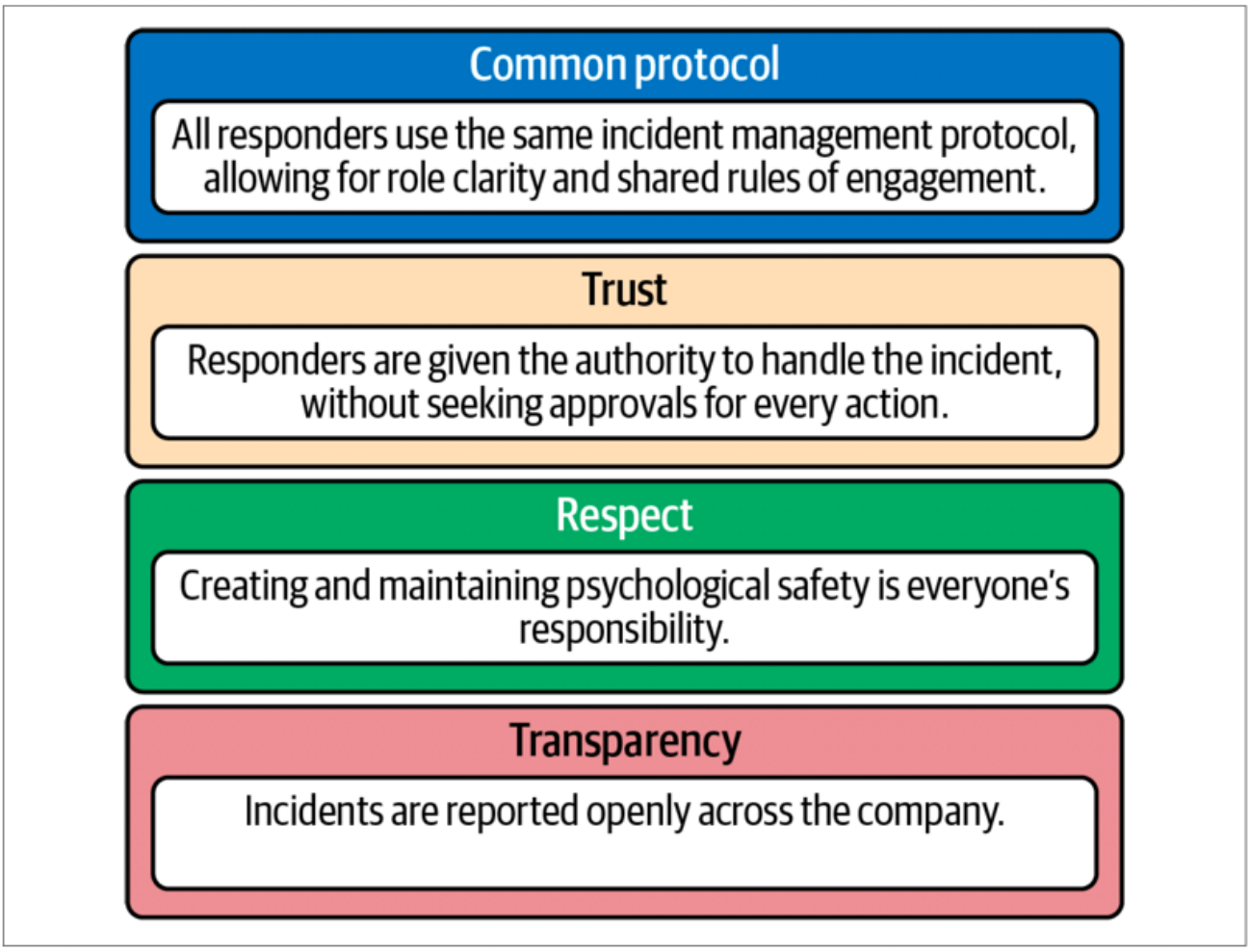

图 3-5. 成功事故响应组织的特征

统一协议

在 Google，我们广泛使用 FEMA 事故指挥系统（ICS）的内部变体，其中事故响应人员有明确的角色，如事故指挥官、记录员和通信员。通过使用共享且明确定义的流程，我们建立了有效的紧急响应习惯，包括保持活跃状态、明确的指挥链和减少整体压力。每个人都了解交接流程，知道应该交接给谁，以确保知识的有效传递。就像象棋不能在麻将桌上玩，在紧急情况下，所有人都必须按照同一个规则行事。

信任

在事故发生期间，事故指挥官需要行使权威。他们需要指挥他人、组织混乱的能量，并判断合适的行动方案。对于许多组织来说，将权威级别与操作职责对齐是一个挑战，但我们的标准操作程序避免了只有高层业务主管才有权做出服务变更决策的惯例：我们将这种权威赋予具有背景知识和实时状态信息的主题专家（SME）。

尊重

确保所有响应人员在认为有必要时能够放心地升级情况非常重要。如果响应人员因为升级事故而受到审查、批评或被认为无能，他们可能不会在适当的时候进行升级。除了基本的礼貌，我们必须相信每个人在现有信息的基础上做出最佳决定。如果出现问题，关键不是责备某人，而是找出如何提供更准确和可操作的信息，以确保未来不再出错。这部分工作在事后分析过程中进行，Google 坚持严格的无责政策（稍后会详细介绍）【译者注：对事不对人策略】。

透明度

我们不进行信息隔离。当事故发生时，所有细节对所有人开放。如果禁止访问事故信息，就无法进行升级和互操作性——我们在事故解决后撰写的事后分析会在公司范围内的每周通讯中分享。我们鼓励通过阅读其他团队和产品领域发生的事故来进行跨团队学习。

风险管理

除了事故响应组织结构的特征外，还需要考虑如何管理风险。从识别到解决事故的时间不应超过三天。正如之前所说，事故管理在时间和人力上都非常昂贵。长时间保持在事故管理的活跃状态会导致疲劳和倦怠，可能促使你开始考虑跳槽。事故是已经升级并需要立即有组织响应的问题。这种紧急状态并非自然状态——人类的大脑延髓不应该被长时间刺激，他们的身体也不应该长期分泌大量皮质醇。

如果史前人类不断狩猎或被剑齿虎追捕，无法感到安全或休息，我们的进化会截然不同。如果你预计长时间处于战斗或逃跑模式，最终会导致团队成员的持续流失。

事故管理与风险的职能

为了减少事故管理的时间，重要的是认识到事故管理和风险的功能。事故管理是一项短期任务，旨在迅速纠正危险情况。事故的严重程度可以分为几个简单的类别。在 Google，我们根据组织的产品适当地量化了这些类别（见表 3-1）。

表 3-1. 严重性定义

严重性	定义	试金石
重大	面向用户的重大故障，产生负面新闻或对 Google 或特定 Google 客户造成巨大的收入影响。内部生产力故障只有在产生可见外部后果（如负面新闻周期）时才视为重大。	可能或已对 Alphabet/Google 品牌和业务造成损害。
严重	对用户可见的故障，但不会对 Google 服务或特定客户造成持久损害，或对 Google 或其客户造成可观的收入损失，或 50% 或更多的 Google 员工受到显著影响。	此类故障如果持续发生且未得到缓解，可能或将对 Alphabet/Google 品牌和业务造成损害。
中等	从差一点到重大/严重故障。大量内部用户受到显著影响。存在已知的解决方法，减轻了影响。	此类故障如果持续发生且未得到缓解，可能会随着时间推移导致越来越多的不稳定性和更高的维护成本。
轻微	外部用户可能未注意到故障。内部用户受到不便。导致网络、数据中心、实例等之间的流量发生意外波动。	此类故障如果持续发生且未得到缓解，不太可能随着时间推移导致更多不稳定性，但代表正常操作条件。
微不足道	事故对用户没有任何可见影响，对生产几乎没有实质性影响，但从中学到了宝贵的教训，需要以低优先级跟踪一些后续行动项目。	此类事故如果持续发生且未得到缓解，不会被视为过程失效。
忽略测试	这甚至不是一次事故。去做其他事情吧。	虚惊一场。

事故的规模大致反映了情况的“风险性”（根本原因/触发/影响）。事故管理旨在缓解短期影响，为组织的决策者争取时间来决定下一步措施。事故管理并不意味着要持续到避免所有短期和长期影响。对于庞大的技术堆栈或积累的技术债务，可能需要数月甚至数年才能彻底解决根本原因/触发条件。事故应只在短期影响尚未缓解时保持“开放”状态，并进行积极的管理。

在医院中，这相当于评估出血患者的紧急风险，并为其止血。那么接下来呢？医院会确定出血的原因并防止其复发。可能需要为患者制定长期计划，如避免再次遇到剑齿虎，或治疗引起出血的皮肤病。无论是哪种方式，一旦立即的危险消除，就会制定长期计划，包括必要时的全天候支持，以确保患者安全并防止再次出血。同样地，在你的技术堆栈中，一旦立即的危险解除，就应转向制定长期行动计划。

在事故管理中，通常可以在几分钟内重现事故的时间线。如果处理的是紧急问题，每一分钟都可能影响用户或造成收入损失。因为每一分钟都很重要，事故管理对经理们造成了很大压力——正如本节前面提到的，这不是一种长期的积极体验。当处理事故的长期后果（解决根本原因或触发因素）时，理想情况下，不再有立即的用户伤害或重大利润损失。这很好。这些高优先级工作需要立即执行，但不需要像管理事故那样紧迫。这些工作的时间线可以按天或

周来衡量，而不必像之前提到的事故时间线那样不超过三天。在不需要的情况下，不要保持战斗或逃跑模式。关闭事故，转向恢复。

第四章：缓解和恢复

我们已经讨论了如何扩展事故管理，使用组件响应者和 SoS 系统级响应者来帮助公司扩展时管理事故。我们还介绍了成功的事故响应组织的特征，并讨论了管理风险和防止值守人员倦怠。现在，我们来谈谈事故发生后的恢复工作。我们将从紧急缓解措施开始。

紧急缓解措施

之前我们提到在服务事故期间"止血"。恢复工作中包括必要的紧急缓解措施(Urgent Mitigations)[^1]，以避免影响或防止影响加剧。现在我们来谈谈这意味着什么，以及如何在紧急情况下，更容易的实施缓解措施。

假设你的服务遇到了一个严重的问题。中断已经开始，并已经被检测到了，用户正在受到着影响，而你负责解决这个问题。你的首要任务应该是：立即停止或减轻对用户的影响，而不是立即找出问题的原因。想象一下，你在家里，屋顶开始漏水。你首先会放一个桶在漏水处，以防止进一步受到水的损害，然后再去拿出需要的工具，去修补屋顶（稍后我们会发现，如果屋顶问题是根本原因，雨水就是触发因素）。桶的作用是减小影响，直到修复好屋顶。为了在服务中断期间减轻对用户影响，你需要准备好一些应急措施。我们称这些应急措施为**通用缓解措施**。

通用缓解措施是指在你找出具体问题之前，可以采取的减小各种中断影响的行动。

适用于你服务的缓解措施可能会有所不同，取决于用户受影响的方式。基本的措施包括：回滚代码、重新分配流量，以及增加服务器容量。这些临时措施旨在：为你和你的服务争取更多时间，以便找到彻底解决问题的方法。换句话说，它们修复的是中断的症状，而不是根本原因。你不需要等到完全的理解了中断的原因，就可以使用通用缓解措施。

考虑进行研究并投资开发一些快速"一键修复"（即比喻中的桶）。记住，尽管桶是一个简单的工具，但它仍然可能被误用。因此，为了正确使用通用缓解措施，重要的是要在定期的恢复力测试中进行演练。

减小事故的影响

除了用于应对紧急情况或事故的通用缓解措施外，还需要考虑从长远角度减小事故的影响(Reducing the Impact of Incidents)。**事故**是一个内部术语。实际上，客户并不真正关心事故或事故的数量，他们关心的是可靠性。为了满足用户的期望，并实现用户所需的可靠性水平，需要设计和运行可靠的系统。

想要实现这一点，需要在事故管理生命周期的每个阶段中协调的行动：准备、响应和恢复。考虑在事故发生前、发生期间，和发生后可以做些什么，从而改进系统。

虽然很难直接衡量客户信任，但可以使用一些代理指标来评估提供可靠客户体验的效果。我们称客户体验的度量为服务质量指标 (SLI)。SLI 告诉你在任何时刻服务的表现如何，是否达到预期。

在这个范围内，客户可以是终端用户、人类或系统（如 API），或另一个内部服务。内部服务类似于为其他内部服务提供核心功能，而这些服务最终面向终端用户。你需要确保关键依赖项的可靠性（即硬性依赖或不可缓解的依赖——如果它失败，你也会失败）。这意味着如果面向客户的服务依赖于内部服务，这些服务需要提供更高水平的可靠性。

SLI 的可靠性目标称为服务质量目标 (SLO)。SLO 将目标汇总到一段时间内：它表示在某个管理时间窗口中，这是你要去实现的目标，体现了你的管理水平如何（通常以百分比衡量）。[^2]

可能大多数人都熟悉服务质量协议 (SLA)。SLA 定义了你向客户承诺的服务内容；即如果未能达到目标，你愿意采取的措施（如退款）。为了实现这一点，需要使 SLO（你的目标）设定的比 SLA 要更严格一些。[^3]

我们用来检查和衡量用户满意度的工具称为**用户旅程**。用户旅程是以文本形式编写的陈述，用来描述用户的视角。用户旅程探讨了：用户如何与服务互动的过程，以实现自己想要的目标。那些最重要的用户旅程被称为**关键用户旅程** (CUJ)。[^4]

一旦定义了对你和用户或客户重要的目标，就可以开始考虑：当未能达到这些目标的时候，都会发生什么。

计算事故的影响

事故会影响可靠性目标，影响的大小：取决于故障的数量、持续时间、影响范围和规模。因此，想要减小事故的影响，首先需要了解可以采取哪些措施。让我们先看看应该如何量化和衡量事故的影响。

图 4-1 显示了衡量影响的方法：计算不可靠的时间，包括检测到问题的时间和修复（缓解）问题的时间，然后将其乘以事故的次数，即事故的频率。

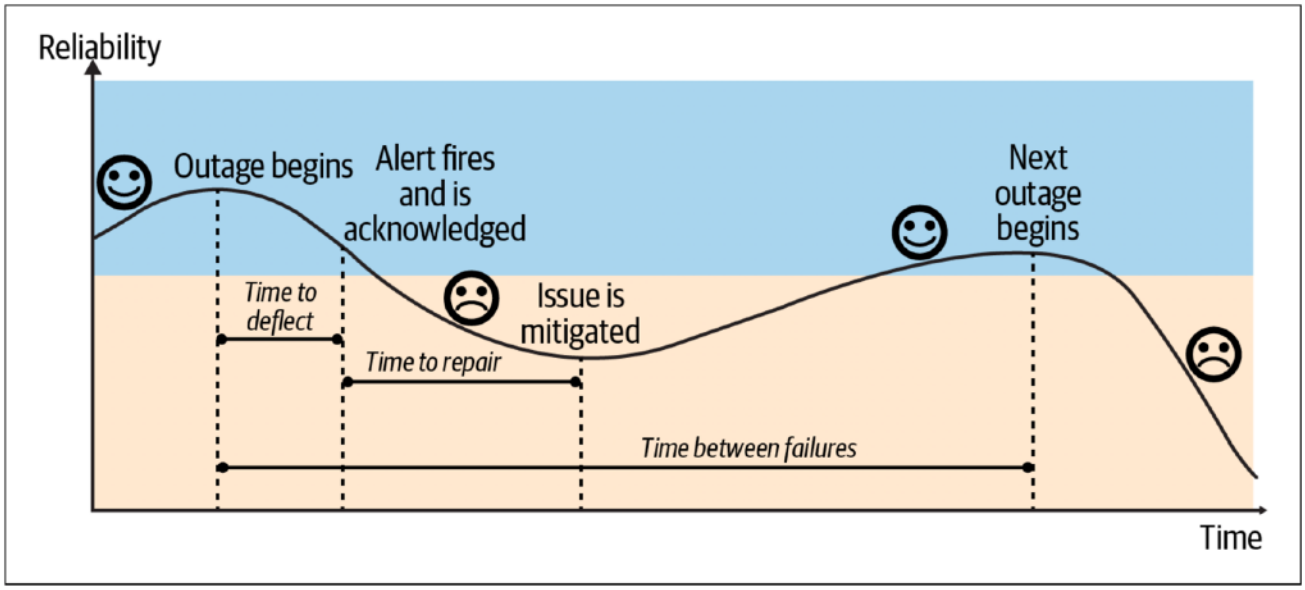

图 4-1. 中断生命周期

关键指标是检测时间、修复时间和故障间隔时间：

- 检测时间（TTD）是从中断发生到某个人被通知或告警的时间。
- 修复时间（TTR）是从某人被告警到问题缓解的时间。关键是缓解！这指的是响应者采取措施减轻客户影响的时间，例如通过将流量转移到其他区域。
- 故障间隔时间（TBF）是从一次事故开始到，同类型事故下一开始的时间。

想要减小事故的影响，并使系统恢复到已知正常的状态，需要技术和"人"的因素的结合，例如：流程和支持。在谷歌，我们发现一旦涉及人为干预，中断至少会持续 20 到 30 分钟。通常，自动化和自愈系统是很好的策略，因为它们会有助于缩短检测时间和修复时间。

$$Unreliability \quad \Sigma \frac{TTD+TTR}{TBF} \times Impact$$

需要注意的是，使用的方法也很重要。简单地降低告警阈值可能导致误报和噪音，过度依赖自动化快速修复可能减少修复时间，但会忽略根本问题。在下一节中，我们将分享几种策略，这些策略可以更有效地减少检测时间、修复时间和事故频率。

缩短检测时间

减小事故影响的一种方法是缩短检测事故的时间(Reducing the Time to Detect)（图 4-2）。在起草 SLO（可靠性目标）时，需要我们进行前置的风险分析，并确定出那些优先的待办事项，识别可能阻止实现 SLO 的因素，这也有助于缩短检测到事故的时间。此外，你可以采取以下措施来最小化检测时间：

- 将 SLI（客户满意度指标）尽可能与用户的期望对齐【译者注：考虑用户体验的好坏，以及关键用户旅程的可用性】，这些用户可以是实际用户或其他服务。将告警与 SLO（你的目标）对齐，并定期回顾评审，确保它们仍然能代表用户的满意度。

- 使用最新的信号数据。选择最佳的数据获取方式：流、日志或批处理。在这方面，在**告警速度**与**噪音数**之间找到合适的平衡度也很重要【译者注：SLI 告警的灵敏度高情况下，考虑到所选择 SLI 信号数据源的质量，如果假性告警越少，则噪音数量越低。】。噪音告警是 Ops 团队（无论是传统的 DevOps 团队还是 SREs）最常见的一个抱怨。

- 使用有效的告警以避免告警疲劳。在需要立即采取行动时使用呼叫【译者注：短信、电话外呼等任何快速触达的通知方式】。只有正确的响应者——特定团队和所有者——才应该收到告警。另一个常见的投诉是收到不可操作的告警【译者注：与事故无关的人员也经常会在半夜被值守的人喊醒，冤】。

随之而来的问题是："如果只对需要立即采取行动的事情进行呼叫，那其余的问题如何处理？"一个解决方案是：为不同情况使用不同的工具和平台。可能"正确的平台"是，一个工单系统或仪表板，或者仅需要根据该指标，用拉取的处理模式，进行相应的故障排除和调试。

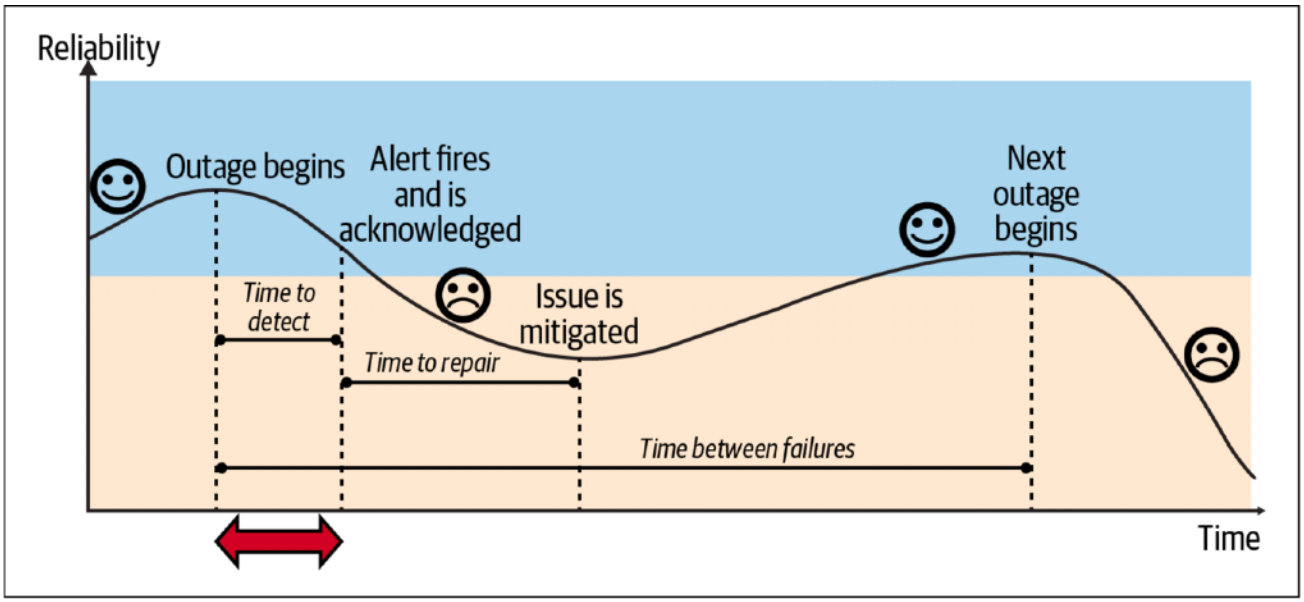
Reliability
Outage begins
Alert fires and is acknowledged
Issue is mitigated
Next outage begins
Time to detect
Time to repair
Time between failures
Time

图 4-2. 中断生命周期

缩短修复时间

我们已经讨论了：用缩短检测时间作为减小事故影响的一种方法。另一种方法是：缩短修复时间(Reducing the Time to Repair)（图 4-3）。缩短修复时间主要涉及“人”的方面。使用事故管理协议和组织事故管理响应可以减少事故管理的模糊性，缩短对用户影响的时间。除此之外，你还需要培训响应者，制定明确的程序和手册，并降低值守的压力。让我们详细探讨这些策略。

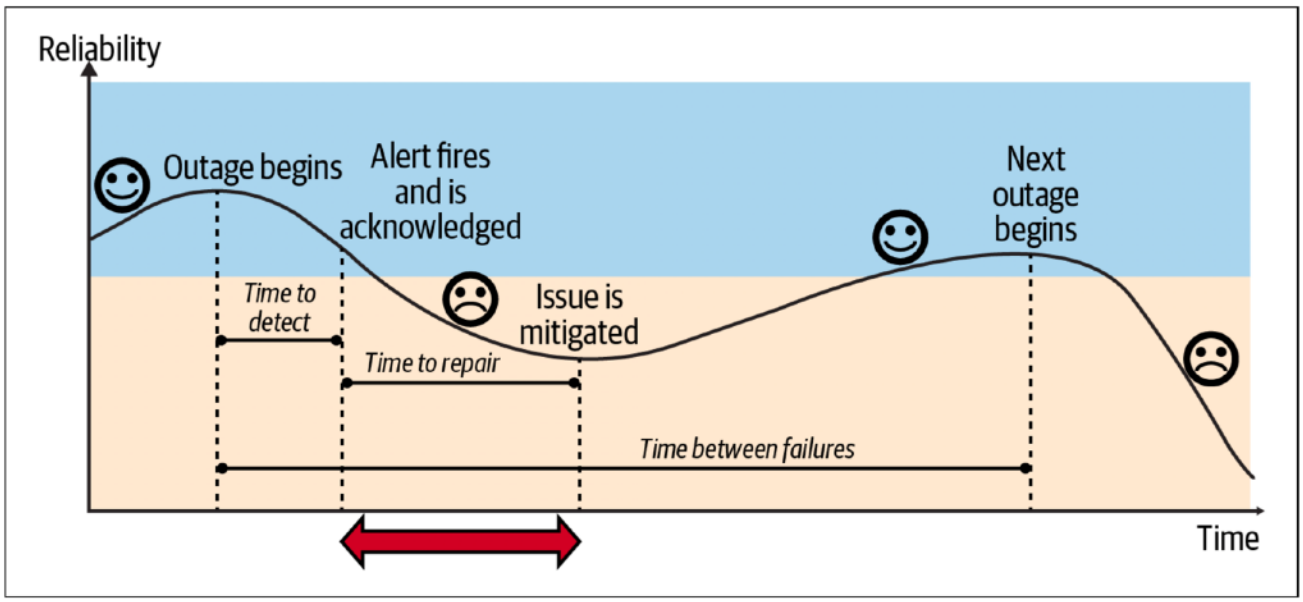

图 4-3. 中断生命周期：修复时间

培训响应者

未准备好的值守人员会导致更长的修复时间。考虑对值守人员进行灾难恢复测试培训，或者进行我们之前提到的“厄运之轮”演习。另一种方法是通过导师指导进行值守准备。让值守人员成对工作（“配对值守 pair on call”），或者让新人在他们的轮班期间与有经验的值守人员一起工作（“跟班 shadowing”），有助于增强新队员的信心。记住，值守可能是有压力的。制定明确的事故管理流程可以减少这种压力，因为它消除了任何模糊性，并明确了所需的行动。[^5]

建立有组织的事故响应程序

事故管理中存在一些常见问题。例如，缺乏责任感、沟通不畅、缺乏层次结构和自由发挥/英雄主义，可能导致更长的解决时间，也会增加值守人员和响应者的额外压力，并最终影响到客户。为了解决这个问题，我们建议通过建立一个层次结构明确的结构、任务和沟通渠道来组织响应。这有助于保持清晰的指挥链，并指定明确的角色。

在谷歌，我们使用 IMAG（谷歌事故管理），这是一个基于消防员和医护人员使用的事故指挥系统（ICS）的灵活框架。IMAG 教你如何通过建立层次结构明确的结构、任务和沟通渠道来组织紧急响应。它建立了一种标准、一致的方式来处理紧急情况和组织有效的响应。[^6]

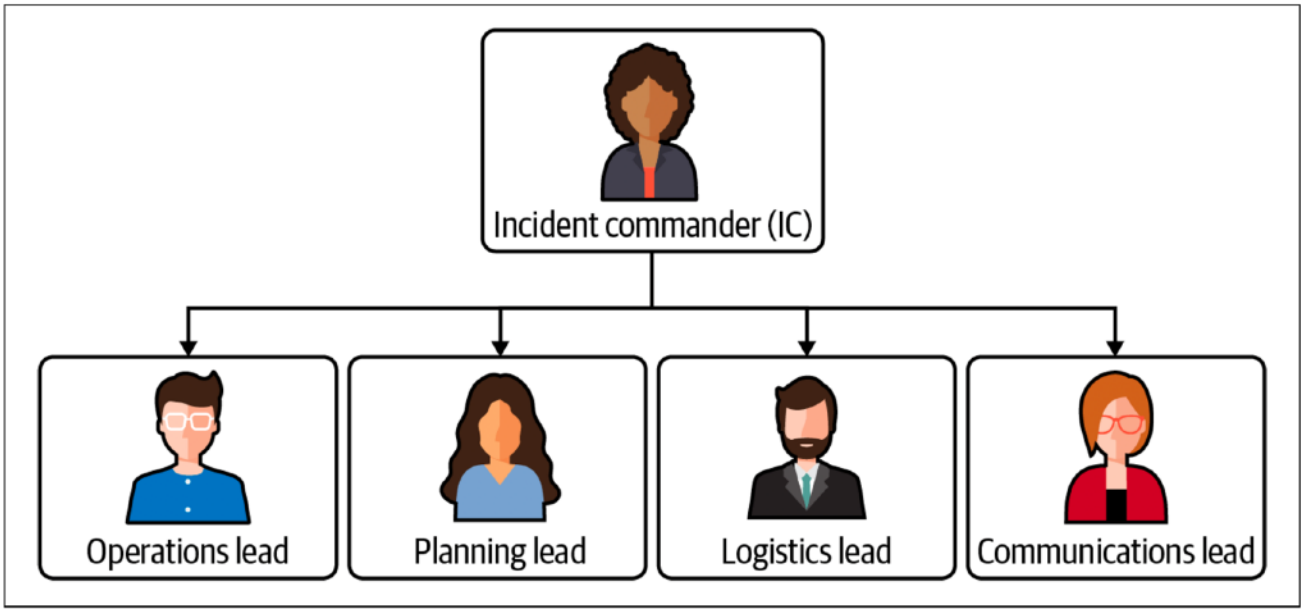

图 4-4. 一个 ICS 层次结构示例

IMAG 协议为解决事故的人提供了一个框架，使紧急响应团队能够自我组织和高效工作，通过确保响应者和相关利益相关者之间的沟通，控制事故响应，并帮助协调响应工作。它规定事故指挥官（IC）负责协调响应并分配职责，而其他人向 IC 报告。每个人都有一个具体的、明确的角色——例如，操作负责人负责解决问题，沟通负责人负责处理沟通。

通过使用这样的协议，你可以减少模糊性，明确团队合作，并减少修复时间。[^7]

建立明确的值守政策和流程

我们建议记录你的事故响应和值守政策，以及在中断期间和之后的应急响应流程。这包括明确的升级路径和责任分配，以减少处理中断时的模糊性和压力。

编写有用的运行手册/操作手册

文档很重要，它将工作经验转化为所有队员都能访问的知识，无论工作年限。通过优先记录和安排时间编写文档，并创建记录程序的操作手册和政策，队员们可以更容易识别事故的表现形式——这是一项宝贵的优势。操作手册一开始不必完备；从简单的开始，提供一个明确的起点，然后逐步改进。一个好的经验法则是谷歌的"看到问题，立即解决 see it fix it"的方法，并让新队员在入职时就来更新这些操作手册。

将编写操作手册作为事后复盘分析的重要行动项目之一，并将其视为个人对团队的积极贡献，这通常需要领导的优先支持和资源分配。

减轻响应者的疲劳

如第二章所述，响应者疲劳的心理成本是有据可查的。如果响应者疲惫，他们的解决问题能力会受到影响。确保班次平衡，如果不平衡，使用数据来找出原因，并减少琐事。[^8]

投资于数据收集和可观测性

做出基于数据的决策很重要，缺乏监控或可观测性是一种反模式。如果你无法看清路况，你就不知道前进方向。因此，鼓励组织内的度量文化，收集贴近客户体验的指标，并衡量你在目标和错误预算消耗率方面的表现，以便及时反应和调整优先级。还要衡量团队的琐事工作量，并定期审查你的 SLI 和 SLO。

尽可能收集高质量的数据，特别是更贴近客户体验的数据；这有助于排除故障和调试问题。收集应用程序和业务指标，以便拥有关注客户体验和关键用户旅程的仪表板和可视化。这意味着为特定受众和目标设计的仪表板。管理者对 SLO 的视角，将与用于在排查事故和故障过程中使用的仪表板非常不同。

如你所见，有许多方法可以缩短修复时间，并最大限度地减小事故的影响。现在让我们看看延长故障间隔时间来减少事故影响的另一种方法。

延长故障间隔时间

为了延长故障间隔时间并减少故障次数，可以重构架构，并解决在风险分析和流程改进中识别出的故障点（图 4-5）。此外，还有一些措施可以帮助延长 TBF（故障间隔时间）。

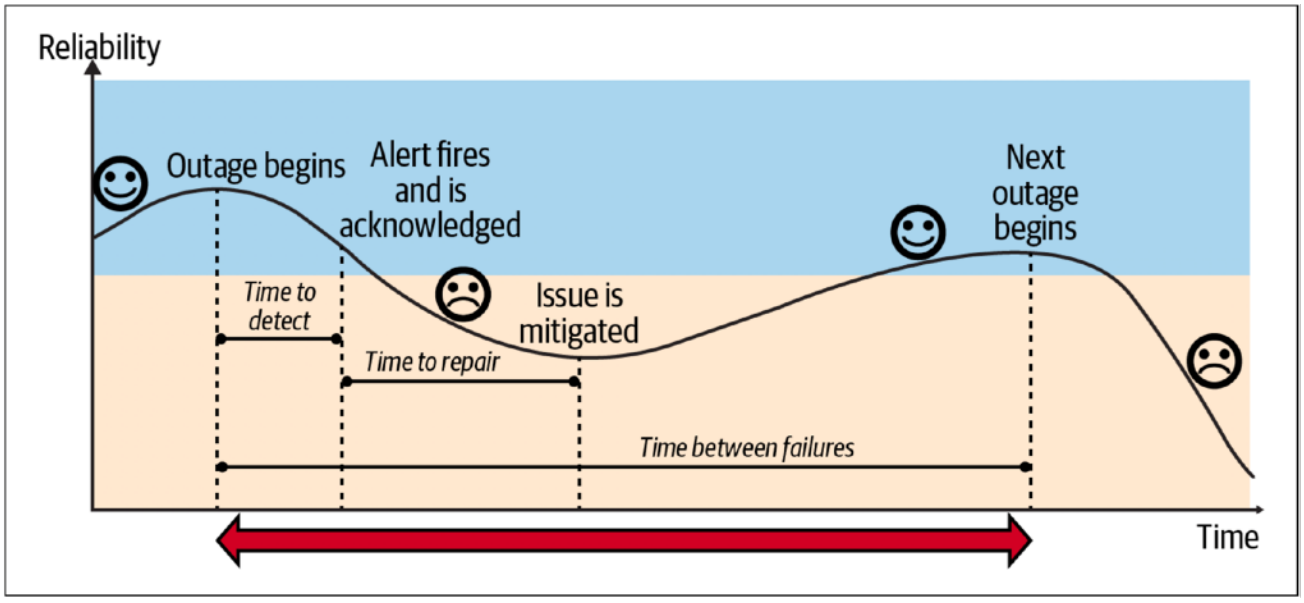

图 4-5. 中断生命周期：故障间隔时间

避免反模式

我们在本报告中提到了几种反模式，包括缺乏可观测性，缺乏正反馈回路，这会导致系统发生过载，并引发级联问题，如崩溃。这些反模式需要避免。

分散风险

通过冗余、解耦责任、避免单点故障，和用全局优化来分散风险，并采用高级部署策略。考虑渐进式的滚动和金丝雀发布，将更新工作分布在数小时、数天或数周内，这样可以在所有用户受到影响之前，减少风险并识别问题。同样，进行自动化测试、滚动发布和自动回滚，以便及早发现任何问题。主动发现问题，总要比让问题来骚扰你会更好；那就要通过实践混沌工程和引入故障注入，以及自动化灾难恢复测试（如 DiRT，见第二章）来实现这一点。

采用开发实践

采用促进质量文化的开发实践，并创建集成代码审查和健壮测试的过程，这些过程可以集成到持续集成/持续交付（CI/CD）流水线中。CI/CD 可以节省工程时间，并减小对客户的影响，使你能够自信地部署。

以可靠性为设计原则

在 SRE 中，我们有一句话：“碰运气不是一种策略。” 当谈到故障时，问题并不是会不会发生，而是什么时候发生。因此，从一开始就以遵循：可靠性为设计原则，构建能够应对故障的健壮架构。通过以下问题来了解你如何应对故障：

- 我的系统能够应对哪种类型的故障?
- 它能容忍意外的单实例故障或重启吗?
- 它如何应对区域性AZ或地区性Region故障?

意识到风险及其潜在影响范围后，进入风险缓解阶段（如在风险分析中所做的那样）。例如，为了缓解单实例问题，使用持久磁盘和配置自动化，并且备份数据。为了缓解区域和地区故障，可以在各个地区和区域分配资源并实施负载均衡。还可以进行横向扩展。例如，将单体架构解耦为微服务，更容易独立扩展它们（“做好一件事”）。横向扩展还可以意味着地理上（Region）的扩展，例如拥有多个数据中心以利用弹性。我们建议尽可能避免手工配置和特殊硬件。

优雅降级

在你的架构中实现优雅降级(Graceful degradation)方法非常重要。将降级视为一种策略，例如限流和负载分流。问自己，如果不能为所有用户提供所有功能，我能否可以最小功能为所有用户服务？能否限流用户流量并丢弃高成本的请求？当然，什么是可接受的降级程度，要依赖于服务和用户旅程。返回 x 个产品和返回未更新的账户余额之间存在差异。但作为经验法则，能提供降级的服务，总比停止服务的好。[^9]

深度防御

深度防御(Defense-in-depth)是构建系统以应对故障的一种方式，更准确地说，是容忍故障。如果依赖某个系统获取配置或其他运行时信息，确保有一个备用或缓存版本，当依赖项不可用时，而它们仍能继续工作。[^10]

N+2 资源

在分布式系统中，拥有 N+2 资源是实现可靠性的基本原则。N+2 意味着：你有 N 的容量来处理高峰期的请求，并有另外 2 的实例，其中一个可用于应对意外故障，另一个可用于计划升级。如前所述，你的可靠性取决于关键依赖项的可靠性，因此在架构中选择正确的构建块（Building block）。在云平台上构建时，确保使用服务的可靠性水平，并将它们与你的应用目标相关联。注意它们的范围（例如，在 Google Cloud Platform 中，范围可以是区域性的、区域间的或全球的[zonal, regional, global]）。记住，在设计时就主动解决可靠性问题，可以降低后期的成本。[^11] 并不存在一刀切的解决方案，应让需求指导你来做出因地制宜的设计决策。

非抽象大型系统设计 (NALSD)

在讨论可靠性和 SRE 的设计时，我们不能不提到非抽象的大型系统设计。在谷歌，我们发现，在设计阶段解决可靠性问题可以降低未来的成本。如果采用迭代式系统设计和实施风格，可以用更低的成本，开发出健壮且可扩展的系统。我们称这种方法为非抽象大型系统设计 (NALSD)，它描述了谷歌用于生产系统的迭代式设计过程。你可以在谷歌的 SRE 课堂栏目中了解更多相关内容。

从失败中学习

最后，你可以从失败中学习，使未来更好（更多内容请参见第40页的“心理安全”）。如前所述，事后分析是实现这一目标的工具。确保你有一致的事后分复盘析流程，能够产出错误修复（bug fix）、缓解措施和文档更新的后续跟踪落地行动项。像跟踪其他错误（bug）一样跟踪事后 复盘分析的行动项（如果还没有这样做），并应该优先考虑事后复盘分析工作，而不是“常规日常”的工作。[^12] 我们将在下一节中更详细地讨论事后复盘分析。

[^1]: 推荐阅读：Jennifer Mace 的《通用缓解措施》<https://www.oreilly.com/content/generic-mitigations/ 。>

[^2]: 参见《SRE Google 运维解密》（O'Reilly）中的第 4 章，“服务质量目标 (SLOs)”。<https://sre.google/sre-book/service-level-objectives/ >

[^3]: 参见 Adrian Hilton 2021 年 5 月 7 日的文章《SRE 基础 2021：SLIs vs SLAs vs SLOs》。<https://cloud.google.com/blog/products/devops-sre/sre-fundamentals-sli-vs-slo-vs-sla >

[^4]: 参见《Google SRE 工作手册》（O'Reilly）中的第 2 章，“实施 SLOs”。<https://sre.google/workbook/implementing-slos/ >

[^5]: 参见 Jesus Climent 2019 年 12 月 5 日的文章《缩短生产事故缓解时间—CRE 生活教训》。<https://cloud.google.com/blog/products/management-tools/shrinking-the-time-to-mitigate-production-incidents >

[^6]: 参见《Google SRE 工作手册》中的第 9 章，“事故响应”。<https://sre.google/workbook/incident-response/ >

[^7]: 参见《SRE Google 运维解密》中的第 14 章，“管理事故”。<https://sre.google/sre-book/managing-incidents/ >

[^8]: 参见 Eric Harvieux 2020 年 1 月 31 日的文章《使用 SRE 原则识别和跟踪琐事》。<https://cloud.google.com/blog/products/management-tools/identifying-and-tracking-toil-using-sre-principles >

[^9]: 关于负载分流和优雅降级的更多内容，参见《SRE Google 运维解密》中的第 22 章，“解决级联故障”。<https://sre.google/sre-book/addressing-cascading-failures/ >

[^10]: 参见 Ines Envid 和 Emil Kiner 在 Google 博客上的文章《深入了解 Google Cloud 网络：确保环境安全的三项深度防御原则》，2019 年 6 月 20 日。<https://cloud.google.com/blog/products/networking/google-cloud-networking-in-depth-three-defense-in-depth-principles-for-securing-your-environment >

[^11]: 参见《Google SRE 工作手册》中的第 12 章，“引入非抽象大型系统设计 (NALSD)”。<https://sre.google/workbook/non-abstract-design/ >

[^12]: 参见 Google Research 的 Betsy Beyer、John Lunney 和 Sue Lueder 的文章《事后分析行动项：计划工作并完成计划》。<https://research.google/pubs/postmortem-action-items-plan-the-work-and-work-the-plan/ >

第五章：事后复盘分析及其应用

在前一章中，我们介绍了几种减小客户影响的方法，包括技术和人员方面，因为两者都会影响检测时间、缓解/恢复时间和故障间隔时间。在本节中，我们将讨论事故结束后的工作：撰写事后复盘分析，并将其作为强大的工具来分析问题并从错误中学习。

在事故结束后，应该确保集中精力在如何减少未来的事故上？为了解决这个问题，我们建议采用数据驱动的方法（图 5-1）。这些数据可以来自风险分析过程，或者是我们之前提到的度量数据。依靠从事后复盘分析中收集的数据，以及对之前影响客户的事故的学习非常重要。

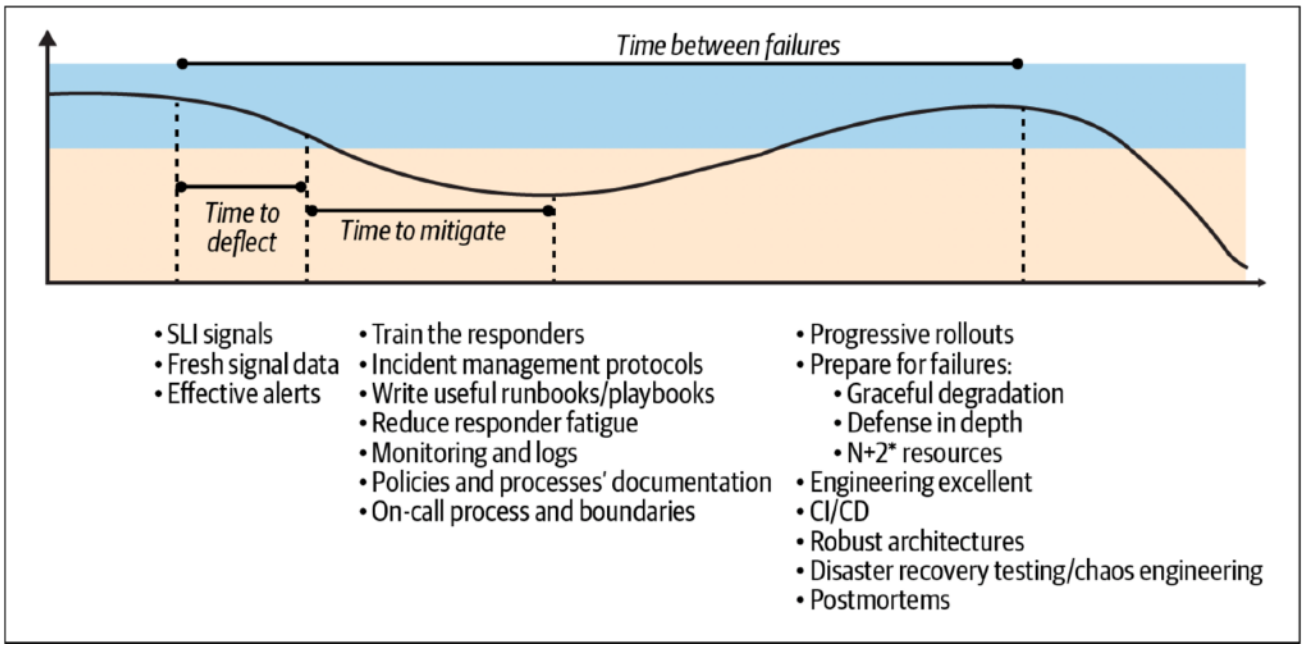

图 5-1. 你应集中精力的地方

一旦你积累了足够多的事后复盘分析，就可以识别出模式。让事后复盘分析成为你的指南非常重要；在失败分析上的投资可以引导你走向成功。为此，我们建议创建一个共享库，并在内部团队中广泛分享事后复盘分析。

心理安全

在谈论事后复盘分析时，不可避免地要讨论心理安全(Psychological Safety)。因此，在深入探讨撰写事后复盘分析的细节之前，我们先来谈谈事故管理文化中固有的心理安全，并讨论早期升级的价值。

如果客户受到影响，应该尽快解决问题。如果组织内的人们不觉得升级或扩大事故规模是安全的，那么问题就难以解决。如果公司环境阻止人们质疑，或因升级事故而受到惩罚，响应者可能会犹豫是否质疑。如果是这样，事故只会在改善之前变得更糟。

失败是正常的，需要接受这一点。这就是为什么实施 SRE 原则需要支持性和赋权文化的原因。关键在于理解，在不断通过新功能和系统改进服务的过程中，事故和中断是不可避免的。因此，如果不从事故中学习，就错失了改进的机会。正如合气道创始人植芝盛平所说："失败是成功之钥，每一个错误都教会我们一些东西。"

将运维问题视为软件工程问题，当事情出错时（而且确实会出错），要寻找的是系统中导致问题的缺陷。你要改进系统，以帮助避免人为错误。

人类永远不是事故的原因，而是"允许"事故发生的系统和流程。

如果发生了中断，那是系统的错误，而不是人类的错误，因为人为错误是不可避免的。目标不是消除人为错误。【译者注：带有缺陷的系统导致了人为错误，或者人类遭遇到了事故；这里我们要把人的原因摘除的干干净净，这一点需要依靠大家构建心理安全的企业团队文化。】

实施事故管理实践时的心理安全

实施事故管理实践是一项组织变革，需要一些文化前提条件才能让团队从错误中创新和学习。拥有心理安全和无责流程是至关重要的。(Psychological Safety When Implementing Incident Management Practices)

心理安全是一种信念，即谁不会因为提出想法、问题、质疑、担忧，或遭遇到了错误而受到惩罚或羞辱。——艾米·埃德蒙森，哈佛商学院诺华教授，领导力与管理学博士

心理安全促进了绩效导向型组织的一些主要特征，特别是将失败视为学习机会和接受新想法。例如，Westrum 的组织文化模型预测了基于心理安全的软件交付绩效：生机型组织比其他两种类型更有可能成为顶级绩效者。[^1]

具有较高心理安全的团队更能利用成员多样化的想法，销售目标超额完成17%（相比之下，不安全的团队错失目标19%），并且被高管评为有效的概率是其两倍。[^2]

处理事故时的心理安全

在风险管理中，每个人都知道自己可以表达意见和识别问题，而不会受到惩罚或嘲笑，这是至关重要的。当发生事故时，必须报告并宣告为事故【译者注：声明发生了事故，开始进入事故应急管理流程】。在事故期间，可能需要分享以前事故的信息，如果这样做可以揭示过去的错误（这与无责原则有关）。你还可能需要将事故移交给下一位值班工程师，并提出改进内部流程、工具和功能的建议。

没有心理安全和无责原则，人们会避免提出可能揭示事故根本原因的正确问题。因此，团队无法学习或创新，因为他们忙于管理形象，并且害怕承担个人后果。

为了在团队中培养心理安全和无责原则，关注学习机会：将每次事故视为学习机会，鼓励多样化的观点，邀请每个人（尤其是那些发表不同意的人）表达意见和想法。作为领导者，你还应该承认自己的不足【译者注：没有人是全知全能和权威的，大家都要从发问，和假设开始分析故障】，通过提问来展示好奇心。

不归咎于个人

无责原则和心理安全是相辅相成的，一个可能自然导致另一个。假设发生了一次中断。如果经理问的第一个问题是“是谁造成的？”，这会造成一种互相指责的文化，使团队害怕冒险，从而阻碍创新和改进。相反，你应该提倡无责原则：

无责原则是将责任从个人转移到系统和流程上。[^3]

指责文化会妨碍人们迅速解决事故和从错误中学习的能力，因为他们可能会隐藏信息，避免因害怕受罚而宣告事故。而无责文化允许你专注于改进。你要假设个人是出于善意行事，并根据现有的最佳信息做出决策。调查误导性信息的来源对组织比归咎于人更有益。因此，支持团队的设计和维护决策，鼓励创新和学习，当事情出错时，关注系统和流程，而不是个人。

从错误中学习

错误是宝贵的学习和改进机会，但前提是正确识别错误的程序性和系统性原因。例如，在谷歌，Ben Treynor Sloss 发送季度工程报告《谷歌的成与败》，以培养一种能够从错误中学习的赋权文化。[^4]

促进心理安全环境的其他提示

事故响应者需要一定的信心才能有效应对事故。尽管他们可能处于压力大的情况下，但在处理事故时，响应者必须感到心理安全。

这种心理安全涉及多个层面：

来自队友

- 响应者不应该担心他们的行为会被同伴评判，尤其是在犯错误时。
- 说“我需要帮助”应该得到奖励，而不是质疑或责备。

来自合作团队

- 有些团队可能会觉得 X 团队的成员有居高临下的坏名声，因此不愿与他们交流。更糟糕的是，有些团队接受这种文化，或者利用它来避免与其他团队互动。[^5]这种态度不应被容忍——它会增加紧张情绪，并延缓事故响应。

来自管理层

- 经理负责团队的心理安全。在事故期间，经理通常不做技术工作，而是专注于确保团队的福祉——观察压力和倦怠的迹象，也许在团队处理事故时订购披萨。有时经理可能只是简单地对事故响应者说，“休息五分钟，清理一下头脑。”
- 经理也可以在获取组织其他部分的额外帮助方面发挥重要作用。
- 经理为团队提供与组织其他部分之间的缓冲，并在发生冲突时介入解决。

来自组织

- 心理安全只有在组织文化中得到认可时才能蓬勃发展。应该有一种无责文化，重点是修复导致事故的流程。

- 业界有诸如“三振出局”政策，这种政策要求对涉及三次影响生产的错误的个人进行解雇或严厉的训诫。虽然这种政策旨在鼓励响应者在事故期间格外小心，但它往往导致响应质量降低（“我不想成为那个做出错误决定的人”）、推卸责任（“我们没有弄坏它，是另一个团队弄坏的”）或隐藏有价值的信息（“不要透露我们已经知道这个问题的事实”）。

- 如果领导者希望他们的团队——以及整个组织——蓬勃发展，他们必须培养一种尊重、信任和协作的文化。这必须从组织的高层开始。

如前所述，心理安全环境的一个明显好处是缩短了升级时间。如果一个组织接受协作文化，事故响应者更有可能寻求额外的帮助，无论是来自自己的团队还是公司中的其他团队。

在审查事故时，一个反复出现的主题总是“如果我们早些升级，就可以节省 $XXX 的收入损失”，即使是在拥有健康、心理安全环境的团队/组织中。对于事故响应者来说，请求帮助很难，因为这可能被视为软弱或准备不足的表现。我们被训练去隐藏不安全感（即使是感知到的不安全感），并且通常被教导要成为英雄，全力以赴为团队贡献。这些行为在事故响应中实际上是负担，一个不堪重负或疲惫的响应者更容易犯错。因此，升级应该是廉价且快速的，不应有任何附加条件。始终假设最好的意图。如果事实证明升级是不必要的，找出为什么会发生升级，可能是因为文档不完整或缺失，并修复有缺陷的流程。

事故响应者应该高度警惕尝试独自完成所有工作的倾向，而是应该尽早且频繁地升级。在谷歌的一个事故响应团队中，有一句格言：“我们告诉其他团队，我们不介意被频繁呼叫，但我们仍然没有被足够频繁地呼叫。”

撰写事后复盘分析

现在我们已经深入讨论了心理安全，让我们转向撰写事后复盘分析。当事情出错时，这是你从中学习并改进未来的机会。虽然“糟糕的工程师”可能会想“希望没人看到”，但优秀的工程师会注意到问题并想“太好了！告诉大家！”这就是撰写事后复盘分析的意义所在。

撰写事后复盘分析是一种系统分析形式：它是深入研究导致事故的故障，并识别改进工程和工程流程的过程。撰写事后复盘分析不仅仅是一种额外的实践，而是一种在服务中实践系统工程的核心方式，以推动改进。

在撰写事后复盘分析时，创建一个无责文化和假设事故会发生的流程是很重要的。如前所述，防止失败很重要，但要意识到日常失败是不可避免的，特别是在大规模系统中。事故为你和你的团队提供了共同学习的机会。事后复盘分析是你们集体从错误中学习的系统解决方案，并帮助分享这些知识，以及从他人的错误中学习——例如，通过阅读他人的事后复盘分析。

事后复盘分析提供了一种正式的从事故中学习的过程，以及一种防止和减少事故、其影响和复杂性的机制。例如，你可能会学到避免使用补丁作为永久解决方案。事后复盘分析突出趋势并优先考虑你的努力。它们应该是无责的——这可以防止关于问题、谁做了什么以及谁可能有错的侧面讨论。事后复盘分析不是为了归咎于谁，而是专注于从事故中学到了什么以及未来的改进。

每个事后复盘分析都应该包括一些信息。例如，好的事后复盘分析包括明确的行动项（AI action item），以及这些行动项的负责人和截止日期。记住，这不是为了归咎，而是为了增加责任感，消除模糊性，并确保行动得到跟进。此外，重要的是要有一个清晰的时间线，包括中断开始时间、问题检测时间、升级时间（如果适用）、缓解时间（如果适用）、影响和中断结束时间。如果发生了升级，说明为什么以及如何发生的。为了避免混淆，澄清事故和中断的术语，以及事故开始和事故检测的术语。我们建议在事故发生期间保持一个"实时文档"，作为调试和缓解的工作记录，稍后可以用于事后复盘分析。该文档有助于正确记录时间线，并确保不会遗漏重要的行动项。

在事后复盘分析中避免责备性语言，并实践心理安全。以身作则，问很多问题，但绝不要寻求归咎于谁。这是关于理解事件的现实、采取的行动以及未来如何防止重发。

谷歌的最佳实践是将事后复盘分析分享给可能受益于所传授教训的最大范围受众。透明的分享使他人能够找到事后复盘分析并从中学习。[^6]

我们发现，建立一种无责的事后复盘分析文化会带来更可靠的系统，并且对于创建和维护一个成功的 SRE 组织至关重要。

用于组织改进的系统分析

我们已经讨论了无责的事后复盘分析，并提到事后复盘分析是一种系统分析形式。然而，你是否真正深入了解你的系统，充分理解发生了什么以及为什么？事件应该被分析以得出结论，而不仅仅是叙述。事故之后或在事后复盘分析中，分析的深度在于是否对事件和系统各方面进行了深入分析，以揭示和解释结论。这很重要，因为它增加了团队在事故之后解决正确问题的概率。

在撰写事后复盘分析时，你应该力求对系统有最完整和准确的了解，以确保所做的修复是正确的。在图 5-2 中，标有“你认为的问题是什么”的圆圈反映了你在事故期间对系统的理解——这是你能控制的部分。标有“实际问题是什么”的圆圈反映了事故期间系统的实际状态。在复杂技术生态系统中，理解所有细微差别是极其困难的（实际上，我们曾有一位高级工程师花了整整一个月时间来理解一个20分钟的事故！）。然而，事故之后的分析越深入，圆圈的重叠部分越大，你越接近理解根本问题（图 5-3）。

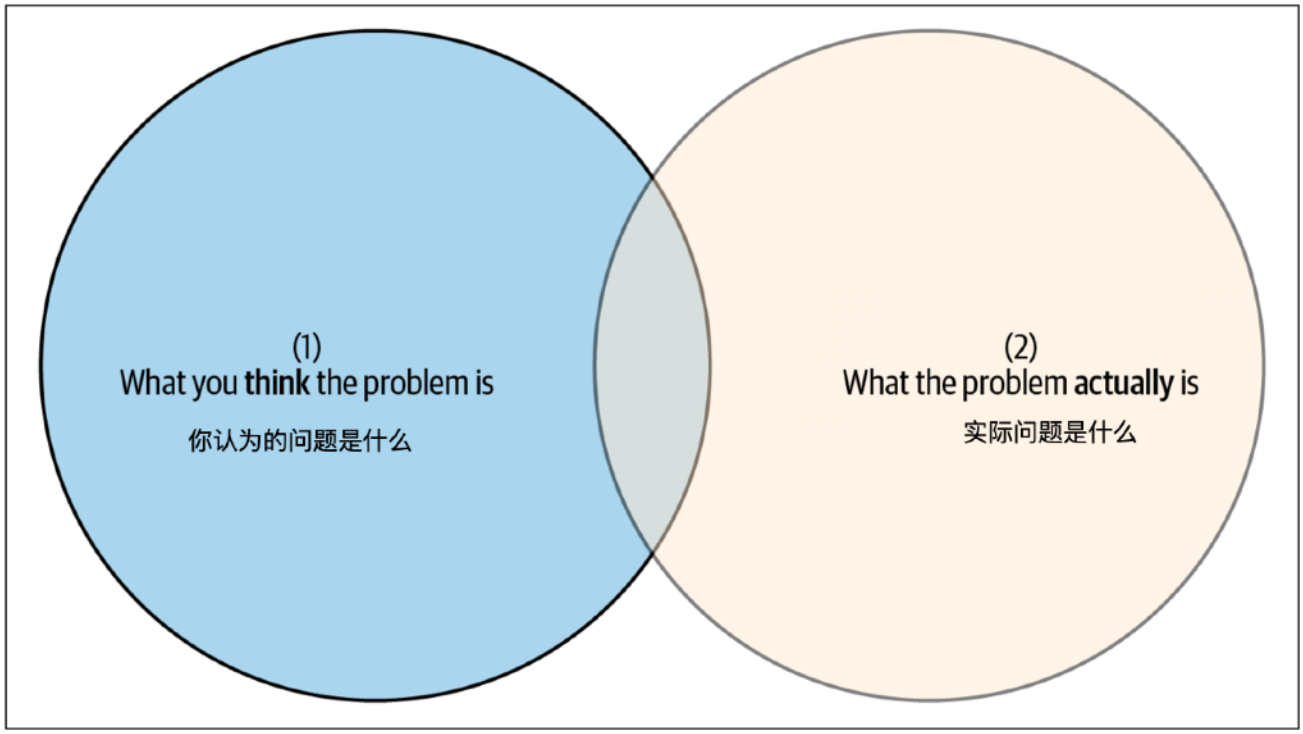

图 5-2. Venn 图显示理解与事实之间的差距

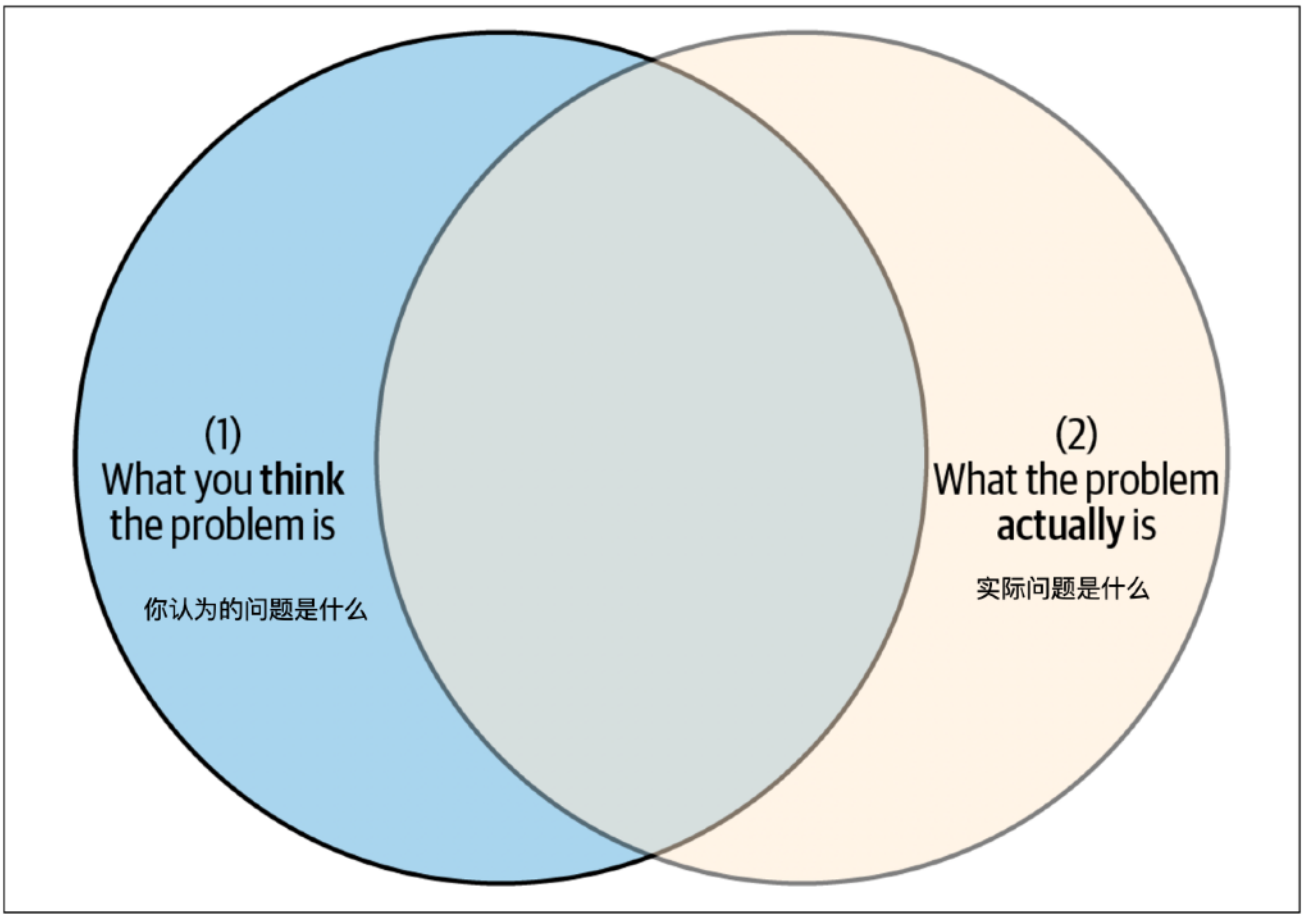

图 5-3. Venn 图显示系统分析的好处

即使事故已经缓解，系统再次稳定，理解真正的问题仍然重要。这涉及可操作性——即你在事故之后有能力修复或更改的内容。事故后的系统增量改进有助于随着时间的推移建立恢复力。这是第三个重要的圆圈，代表你可以控制并可以实施修复的系统中的事物（图 5-4）。这个圆圈无法移动，因为总有一些你无法控制的事情会影响系统的健康（例如天气、地球的大小、光速）。

在中心的那个小交集（在集合理论中表示为 1 ∩ 2 ∩ 3）是你团队在事故之后可以做的最好的工作。“你认为的问题是什么”和“你能修复什么”的重叠部分 [(1 ∩ 3) - 2] 是危险的：这些是你认为在长期内会有帮助但实际上不会解决真正问题的解决方案。你可能正在解决与主要问题相关的某些事物，或者你可能正在处理另一个隐藏问题的表现症状。假设你已经解决了一个实际上没有解决的问题是一个危险的境地——因为缺乏对这一风险的认识，这种情况变得更加严重。如果特定事故再次发生，你将面临客户信任的降低和本可以更有效利用的时间的浪费。

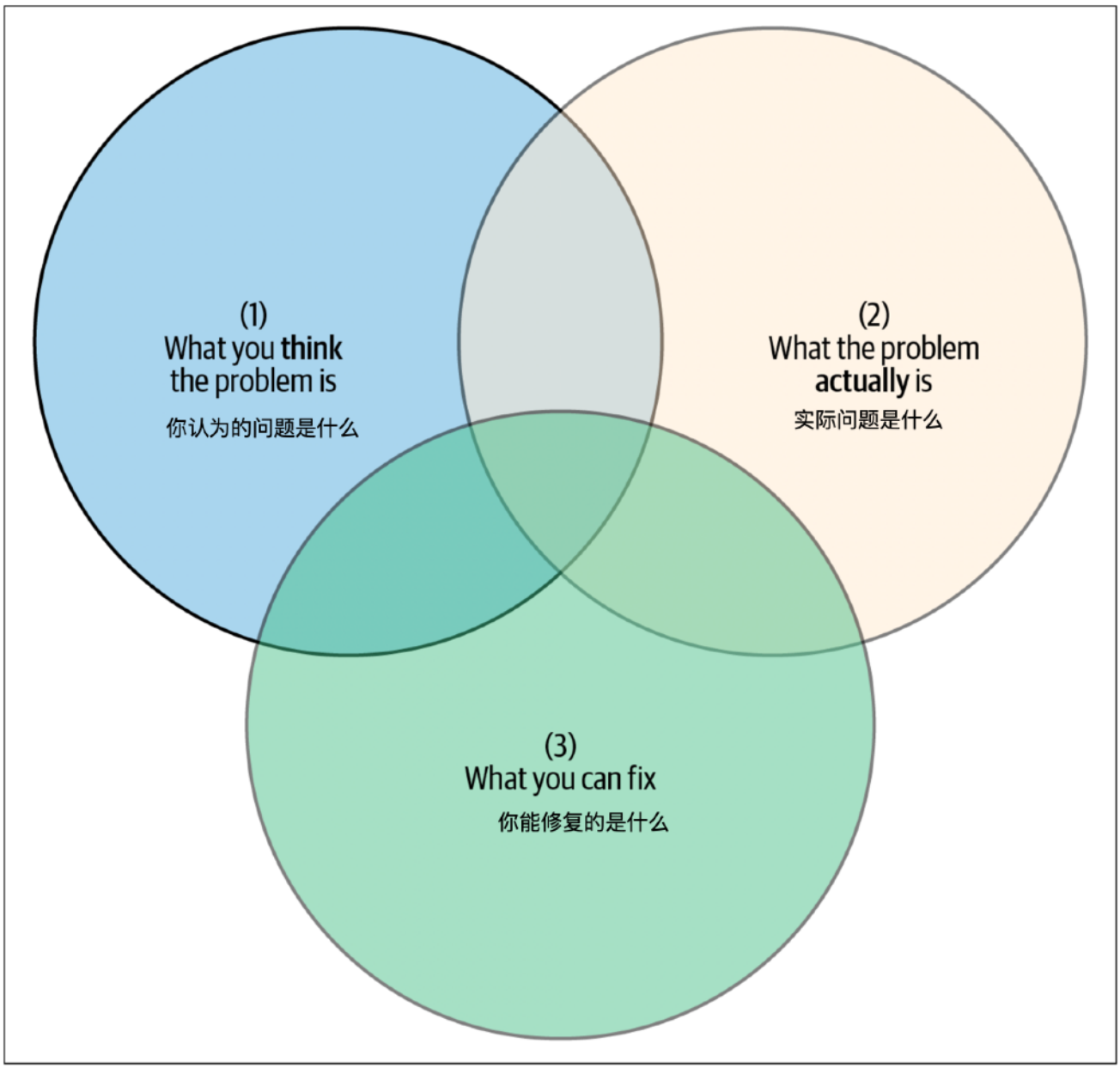

图 5-4. Venn 图显示工程工作与系统分析的相互作用

通过更深入的系统分析，中心的那个小片段（1 ∩ 2 ∩ 3）在两个不动的圆圈中被最大化（图 5-5）。换句话说，你正在最大化你优先考虑的修复措施的有效性。如果你想确保你正在针对正确的问题，移动圆圈是值得的。关键是要在系统分析上投入足够多的时间，以便你和你的团队能够以高概率选择最适合的工程项目来提高系统的恢复力。但要注意收益递减——例如，花一个月时间调查每一次中断并不是明智的资源使用。在以下部分中，我们提出了一些关键点，可能有助于思考如何移动圆圈。

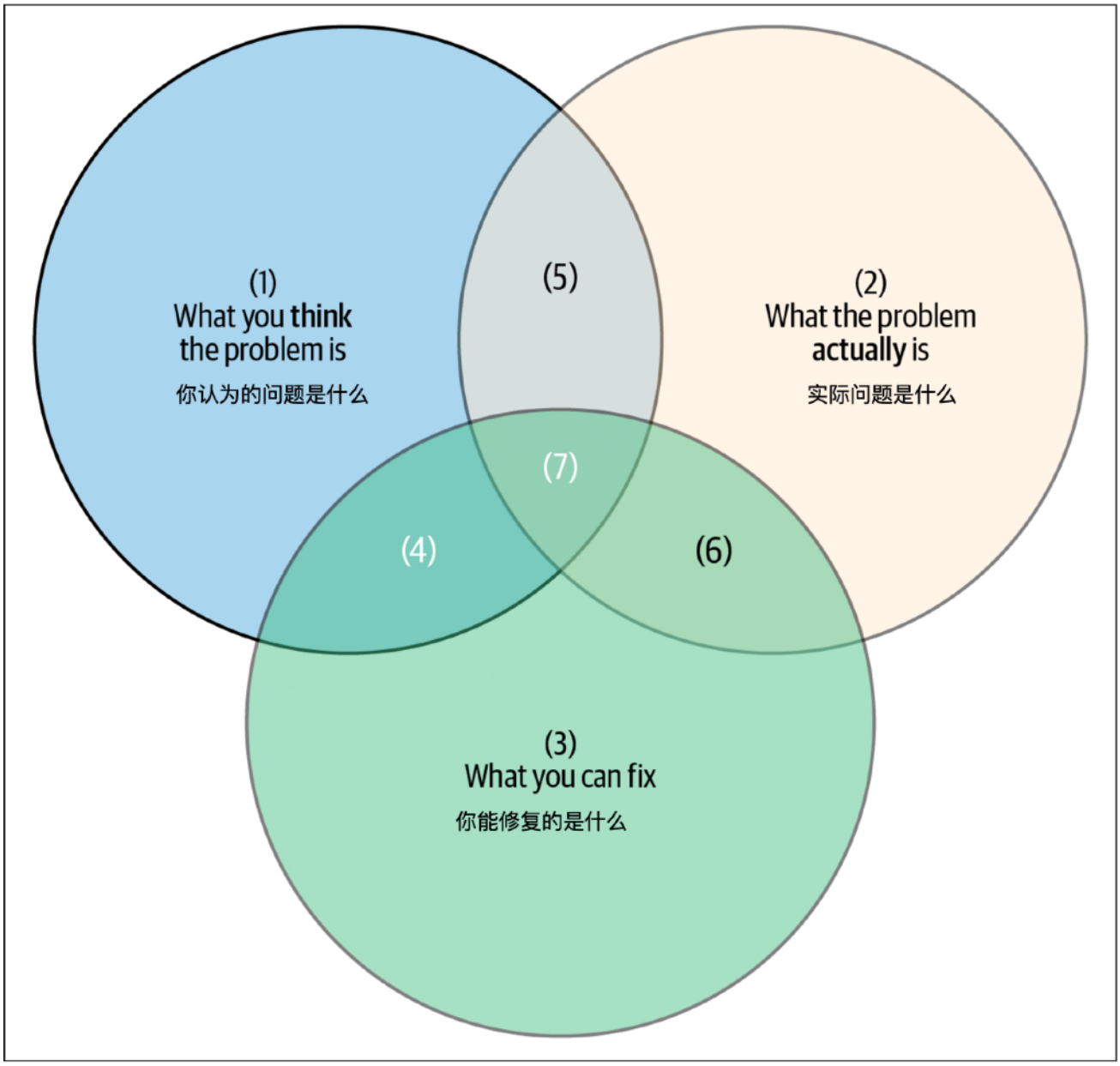

图 5-5. Venn 图突出工程工作与系统分析之间的相互作用

根本原因与触发因素

让我们从两个关键术语开始：根本原因和触发因素。

根本原因 (Root cause)

系统中的潜在危险，或者系统为什么变得脆弱。危险可能在系统中存在无限期——系统环境需要某种变化，才能将这种危险转化为中断。明确一点：在复杂系统中，事故很少只有一个根本原因。熟练的从业者认为，事故的根本原因是相互作用的一系列因果因素，导致危险状态。

触发因素 (Trigger)

使根本原因转变为事故的情况。这是相关但独立的概念！为了防止中断再次发生，有时重要的是解决根本原因。有时更合理的做法是围绕这些触发因素建立预防措施。

根本原因和触发因素共同作用造成了事故。当然，这是一种简化的说法。借用医学术语，根本原因与触发条件相互作用，产生了结果情景（即事故）。在复杂系统中，根本原因和触发因素与事故类型之间并不存在一一对应的关系，复杂性使得各种结果都有可能发生。让我们来看一些例子：

房屋火灾

- 根本原因：煤气泄漏
- 触发因素：靠近漏气炉子的电插头产生火花，引燃了泄漏的煤气并引发了房屋火灾
- 事故：房屋火灾（但这个根本原因可能导致其他事故）

蚂蚁入侵

- 根本原因：温暖的季节适合虫子和害虫在自然环境中繁衍生息
- 触发因素：随意吃东西，留下大量的碎屑
- 事故：蚂蚁入侵

内存不足 (OOM)

- 根本原因：配置文件更改引入了内存泄漏
- 触发因素：出人意料的大量请求
- 事故：OOM

在第三个 (OOM) 场景中，根本原因可能在触发条件存在之前的几年就已经存在了——这是技术债务最终比预期更昂贵的一种方式。而这个根本原因甚至可能不是一个错误，它可以是对系统行为的任何约束。约束本身并不是危险的，直到系统面临某种环境条件，将其转变为危险。需要澄清的是，触发因素可能不是二元的。触发条件可能存在于动态范围内，只有当系统的环境条件和根本原因相互作用时才会成为事故。这两者可以看作是创建事故生命周期的关键组成部分。

事后复盘分析中的根本原因部分应详细说明事故的根本原因和触发因素。为了防止中断再次发生，有时重要的是解决根本原因，有时更合理的是围绕触发因素建立预防措施。

然而，仅仅将根本原因和触发因素分开讨论并不会提高团队事后复盘分析的质量。所有部分都有适当的内容是最低要求，但事后复盘分析还应包括深入的分析，便于团队外的工程师理解，并且是可操作的。这是一个经常出现的问题吗？是否记录了缓解步骤，或者需要查找错误？事后复盘分析是否适当地解释或量化了系统的正常运行情况，以显示故障的对比和影响？如果你说产品 89% 的用户受到了影响，这具体意味着什么？

孤立的系统与整体堆栈

事故影响的系统不太可能在真空中存在（除非你来自 Hoover、Dyson 或 Roomba）。不幸的是，一个常见的反模式是将系统分析的范围限制在看似损坏的部分，而不考虑系统上下文（系统功能相关的环境部分）。以下是一些可以扩展系统分析深度的思考点：

- （如果适用）这个事故是作为单一事件进行审查，还是讨论了相关的/关联的/子事件？
- 你或任何主要的内部客户是否发现了以前未知的依赖关系？
- 端到端通信的效果如何？

虽然事故可能只发生在整体堆栈的一个子部分，但这并不意味着你的事故是孤立发生的。查看事故是否以及如何影响整体堆栈和公司的成员，可以揭示系统故障的见解。这可能包括你的事故是否引发了其他事故或级联故障，或者你的公司在事故期间是否能够有效沟通。

时间点与发展轨迹

在研究中，元分析技术是将多个研究汇总成更大的整体结论。如果你将每个事后复盘分析视为一个展示系统在某个时间点状态的研究，那么将这些分析综合起来可以帮助识别新兴的模

式和见解。我们建议利用每个事后复盘分析作为检查系统随时间变化的机会。考虑以下几点：

- 这个事故是否从系统的长期轨迹进行审查?
- 是否存在相同类型的故障重复出现?
- 是否存在任何长期的强化或平衡循环?

整体系统思维的一部分是考虑系统随时间的变化。一般来说，避免同样的事故发生两次是件好事。

我们已经探讨了系统分析在组织改进中的应用及其对你和你的团队的好处。现在让我们来看一个实际的例子。

[^1]: 参见《DevOps 文化：Westrum 组织文化》。<https://cloud.google.com/architecture/devops?hl=zh-cn >

[^2]: 谷歌的 Project Aristotle 项目。<https://rework.withgoogle.com/ >

[^3]: 参见 Coursera 的"Developing a Google SRE Culture"课程。 <https://www.coursera.org/learn/developing-a-google-sre-culture >

[^4]: 要了解更多关于从错误中学习的信息，请参见《Site Reliability Engineering》第 15 章，"Postmortem Culture: Learning from Failure"。<https://sre.google/sre-book/postmortem-culture/ >

[^5]: 同上，参见《Site Reliability Engineering》第 15 章，"Postmortem Culture: Learning from Failure"。<https://sre.google/sre-book/postmortem-culture/ >

[^6]: 欲了解更多信息，请参见《Site Reliability Engineering》附录 D，"Example Postmortem" <https://sre.google/sre-book/example-postmortem/ ，以及关于> Google Compute Engine 事故的公开通信。<https://status.cloud.google.com/incident/compute/16007 >

第六章： 真实事故案例-玛雅末日事件

为了看到前文所讨论的一些原则在实践中的应用，我们将深入探讨一个谷歌重大宕机事故的现实例子。我们将回顾事件的经过，了解规模化组织结构的运作方式，并展示如何解决这个问题以及我们从中学习到的经验。

对于谷歌来说，玛雅末日并不是2012年的新纪元现象。相反，玛雅末日发生在2019年6月2日，与一个名为Maya的网络自动化工具有关。Maya负责标记管理和网络流量调度，一个微小的代码改动导致了实体类型持续被错误标记。

大约在中午，我们正在进行计划中的维护，确定了一系列将在多个服务器上运行的操作和配置变更（包括在Maya上）。当这个错误标记与我们的作业调度逻辑冲突时，我们"发现"了一种新的故障模式，与流量调度相关的作业被大规模取消调度。出入这些区域的网络流量试图将重新调度的任务塞入剩余的网络容量中，其中流量调度功能仍然有效，但最终未能成功。网络变得拥挤，我们的系统正确地对流量过载进行了分级，并自动排空了较大、对延迟不敏感的流量，以保留较小、对延迟敏感的流量。

流量拥塞开始了。结果，我们的监控系统启动了事故管理流程的第一步：告警。当组件响应者从监控系统收到告警时，这反映了其负责的系统中发生的变化。我们的监控系统注意到错误率阈值被突破了，并向负责该网络组件的值守人员发送了自动通知，值守人员开始评估情况。

与此同时，受影响区域的网络容量减少导致溢出，这种网络拥塞引发了我们网络和计算基础设施中的级联故障。总体来说，我们的网络优先考虑用户流量高于内部流量，包括员工的流量。这实际上是合理的，因为我们宁愿从无法解决问题的99.9%的员工那里重新分配容量，并尽最大努力为我们的用户服务。参与事故响应的0.1%的员工通常知道如何继续处理并绕过这个限制。但是，这次级联故障的一个影响是我们的内部工具出现了重大中断，扰乱导致了大量告警的发送，导致大量人收到了呼叫短信。当每个值守人员都切换到事故响应模式时，他们注意到了：由于网络问题导致的服务不可用。网络组件值守人员迅速确定了网络拥塞的原因，但同样的网络拥塞导致服务降级，也减缓了他们恢复正确配置的能力。

每个人都想尽最大努力支持他们的用户，并了解服务恢复的预期轨迹，因此原本负责网络组件的值守人员团队突然新加入了很多同事。

我们在谷歌将组件分为三类：

- 基础设施组件，如网络管道或存储服务。
- 产品服务组件，如 YouTube 流媒体或 Google 搜索的前端。
- 内部服务组件，如监控、零信任远程访问、Maya 和算力管理。这些内部服务组件都在经历困难。

由于网络具有广泛的依赖性，所以在网络组件值守人员解决完问题之前，其他人都无法继续工作。其他值守人员开始提供帮助，并询问他们的服务何时能开始恢复。很多不同响应者预期的并行性，并没有加速问题的解决。根本原因和次生效应开始变得模糊不清；一个团队的原因是另一个团队的结果，每个人都在尝试贡献他们的知识。虽然每个人都是其系统栈的专家，但大多数人都没有对整体系统全面的大局观，不知道哪些工具路径变得不可用。

为什么？未触及拥堵网络的路径是正常的。如果路径在那时看起来像外部用户，则拥堵网络的路径也是正常的，因为我们优先考虑了它们。因此，我们向外部用户提供的服务是可用的——例如视频通话或编辑文档。然而，如果路径是内部服务，如作业控制或 Maya 配置，它就被降级并卡住了。

我们都在观看此次火山爆发，然而，在 20 分钟后，我们得出了问题的结论 “可能与熔岩有关。”

宕机一小时后，一位组件响应者注意到，影响我们基础设施的系统级问题过于普遍，围绕事故的协调沟通变得混乱不堪。此时，已有超过40位队友加入了事故响应通信频道，试图提供帮助。监控数据影显示出：当前事故影响了半个地球。Google Cloud、Gmail、Google Calendar、Google Play等服务都受到了影响——导致企业都无法运作，大量员工无法高效工作，人们无法相互沟通。一些员工试图使用那些不依赖受损网络的零星服务，而另外的人们都已经放弃了。

近40人卷入了本次事故，网络英雄并没有足够的时间和精力，用来制定和协调实施适当的缓解措施，向所有利益相关者广泛沟通，并管理各方期望。因此，他们进行了升级。我们的网络组件值守人员向技术事故响应团队（Tech IRT）发出了求助请求；他们的请求触达到了许多处于合适工作时段时区里的Tech IRT成员，能够处理事故的成员表示了他们的可用性。由于事故影响如此广泛，许多人已经参与了事故。一些Tech IRT响应者没有担任事故指挥官的

角色，因为他们是处理网络问题的团队成员或经理，可以帮助解决主要根本原因，所以他们选择了协助操作的工作。

接受事故指挥官角色的 Tech IRT 的成员，以前没有处理过受故障影响的网络组件，但他们能够评估系统状态和响应人员的情况。利用他们的训练有素，这位指挥官运用一种机制访问到了生产系统，该机制立即将他们的行动标识为“事故响应”，并绕过了“内部流量降级”的标志。一旦内部流量得到了一些空间，他们就指挥网络值守人员开始介入并解决问题。

在此过程中，他们迅速对正在进行的沟通，以及所有试图“提供帮忙”的人进行了组织和结构化。一旦这种混乱的工程能量被组织起来之后，每个人都开始一起取得到了进展。他们能够更清楚地跟踪不同系统的当前状态，并看到缓解措施的实施速度。随着这些繁琐的管理工作不再让网络组件响应者们不堪重负，他们和他们的团队有了实施适当缓解计划的空间，包括丢弃大量负载，为健康重启和一些紧急强行配置变更腾出系统空间。

一旦开始步入了缓解事故的路径，Tech IRT 成员就专注于将事故推向结束。他们设定了一些退出标准，以便我们可以关闭事故，确保其他系统在任何需要执行的恢复操作中得到支持，然后确保被卷入的响应团队都能够顺利交接并离场。

事故结束后，服务都恢复正常，我们进行了深入的事后分析复盘，以分析事故，并理解根本原因的细微之处，以及这些故障模式所揭示的新兴属性。自那以后，参与的网络团队已经开展了一些非常酷的工作计划，重新构建了 Maya，来防止这种故障模式，以及类似的，但以前未考虑到的故障模式，预防它们再次困扰我们的系统。

最后，我们用内部的个人档案徽章、荣誉性的表情包和奖金等方式奖励了相关参与的人员。对大多数人来说，这次非常严重的事故，是他们职业生涯中最艰苦的一天。也为每个参与事后分析复盘的人提供一些小奖励，是他们帮助我们从中得到学习，让我们持续的增长韧性。

第七章 总结与展望

我们探讨了事故的基础知识，并详细了解了事故管理生命周期的三个阶段：准备、响应和恢复。这涵盖了很多内容，但你现在可能会想，“接下来该怎么做？”

首先，要学会在适当的时候使用事故管理。事故响应需要大量人力资源。通常需要一个或多个人参与其中，从最初的告警，到问题解决的整个过程中。事故响应的目的是在问题发生时实施缓解措施，以争取时间来做出优先级决策。这意味着常规的产品修复可能会被推迟，长期计划和改进可能不会被优先考虑。事故响应可能导致服务质量目标 (SLO) 被违反或客户承诺无法履行，并且参与事故响应的员工都会感受到较大压力。

有研究表明，现实世界中的第一事故响应者更容易出现倦怠和疲劳；同样的趋势也适用于处理非现实事故的人——即那些工作与生活不平衡、活动极端或可能缺乏控制的员工。这些因素在技术事故管理工作中很常见，意味着员工可能会感受到倦怠的影响和职业后果。这里的风险包括，最好的情况下是工作表现不佳，最坏的情况下是员工流失。由于这种倦怠产生的相关风险，公司必须尽量做好事故管理，并尽可能减少事故管理的频率。

你的下一个行动是将**事故管理**视为一项关键运维学科，并努力在这方面取得出色的表现。那么，什么是“擅长”事故管理呢？这意味着你的团队（而不仅仅是个别人员）需要积极改进这一循环的所有部分。虽然这听起来不像是：有几个超级英雄消防员冲了进来，他们拯救世界的场景那么戏剧化，但英雄主义心态是有害的。缓慢而仔细地改进事故准备，开发响应事故的工具、技术和通信渠道，并优先考虑可持续和可扩展的工程工作，才是强大事故管理实践的核心。

通过将所有内容视为一个连续且相互关联的循环，每个人都变得重要，并且可以避免将责任归咎于任何一个人或系统组件。无责文化的实践营造了一个心理安全的工作环境，让员工能够在其中蓬勃发展，并创造出色的产品。这些方法帮助谷歌度过了最近全球历史上的巨大不确定时期，也可以帮助提高贵公司的韧性。

总体而言，不要将事故管理应用于每一个潜在问题或类型问题。谨慎而合理地使用事故管理，以避免让团队成员感到倦怠。当你完成事故管理时，停止管理事故，开始进行解决长期问题或风险所需的工程工作。识别并使用其他可能有用的工具。

进一步阅读

- 来自《Google SRE 工作手册》的监控 <https://sre.google/workbook/monitoring/ >
- 来自《Google SRE 工作手册》的事故响应 <https://sre.google/workbook/incident-response/ >
- 来自《Google SRE 工作手册》的事后分析文化：从失败中学习 <https://sre.google/workbook/postmortem-culture/ >
- 事后分析行动项目：计划工作并执行计划 <https://research.google/pubs/postmortem-action-items-plan-the-work-and-work-the-plan/ >
- 使用 SRE 原则减少生产事故影响——CRE 实战经验 <https://cloud.google.com/blog/products/devops-sre/shrinking-the-impact-of-production-incidents-using-sre-principles-cre-life-lessons >
- 缩短生产事故缓解时间——CRE 实战经验 <https://cloud.google.com/blog/products/management-tools/shrinking-the-time-to-mitigate-production-incidents >

参考书目

“Google Data Center FAQ”。《Data Center Knowledge》，2017 年 3 月 19 日。<https://www.datacenterknowledge.com/hyperscalers/google-data-center-faq >

Aleksandra. “63 Fascinating Google Search Statistics”。《SEOtribunal》，2018 年 9 月 26 日。<https://seotribunal.com/blog/google-stats-and-facts/ >

“Incident Command System Resources”。美国联邦紧急事务管理局，美国国土安全部，2018 年 6 月 26 日。

Beyer, Betsy, Chris Jones, Niall Richard Murphy 和 Jennifer Petoff 编辑。《Site Reliability Engineering: How Google Runs Production Systems》。O’Reilly Media，2016 年。

"Data Access and Restrictions"。《Google Workspace Security Whitepaper》，2021 年 10 月。 <https://workspace.google.com/learn-more/security/security-whitepaper/page-7.html >

Treynor Sloss, Benjamin. "An Update on Sunday's Service Disruption"。《Inside Google Cloud (博客)》，Google Cloud，2019 年 6 月 3 日。 <https://cloud.google.com/blog/topics/inside-google-cloud/an-update-on-sundays-service-disruption >

致谢

作者感谢 Jennifer Mace, Hazael Sanchez, Alexander Perry, Cindy Quach 和 Myk Taylor 对本报告的贡献。

作者简介

Ayelet Sachto是 GKE SRE 的站点可靠性工程师，曾在 Google UK 担任战略云工程师，并领导 EMEA 地区的 PSO-SRE 项目。在她 17 年的职业生涯中，她开发和设计了大规模应用程序和数据流，同时实施了 DevOps 和 SRE 方法。她是众多技术文章、演讲和培训的作者，包括 O'Reilly 课程 "SRE Fundamentals in 3 Weeks"，并在数十个会议上发言和领导了数百个工作坊。Ayelet 还是技术社区的积极成员和导师。在空闲时间，她喜欢创造各种东西，无论是厨房中的一道菜、一段代码，还是有影响力的内容。

Adrienne Walcer是谷歌 SRE 的技术项目经理，专注于提高弹性，减少大规模事故对谷歌服务、基础设施和运营的影响。Adrienne 曾为谷歌的 O'Reilly 出版物 《A Practical Guide to Cloud Migration》作出贡献，并在最后一次 USENIX LISA 会议 (LISA21) 上就规模化事故管理发表演讲。在加入谷歌之前，Adrienne 曾在 IBM Watson Health (前身为 Explorys Inc.) 担任数据科学家，并在 Strong Memorial Hospital 和 Cleveland Clinic 从事生物统计工作。她拥有乔治华盛顿大学的系统工程硕士学位和罗切斯特大学的学士学位。在空闲时间，Adrienne 喜欢玩龙与地下城游戏，并在 Second Harvest 食品银行做志愿者。

关于译者

刘征：中国DevOps社区核心组织者，前Elastic资深开发者布道师，《DevOps Handbook》《The Site Reliability Workbook》译者；精通DevOps/SRE/ITSM等理论体系和相关实践等落地实现。致力于在全国范围内通过社区来推广DevOps的理念、技术和实践。推动开源技术堆栈的应用，包括运维大数据分析平台、云原生服务治理、APM全链路监控和AIOps等使用场景。

博客：https://martinliu.cn

反馈： liuzh66@gmail.com

微信公众号：