Google SRE

目录

[**概览** 1](#_Toc503449386)

[SRE方法论 1](#_Toc503449387)

[SRE视角 2](#_Toc503449388)

[**指导思想** 2](#_Toc503449389)

[拥抱风险 2](#_Toc503449390)

[服务质量目的 2](#_Toc503449391)

[减少琐碎事 2](#_Toc503449392)

[分布式系统的监控 2](#_Toc503449393)

[自动化系统的演进 3](#_Toc503449394)

[发布工程 3](#_Toc503449395)

[简单化 3](#_Toc503449396)

[**具体实践** 3](#_Toc503449397)

[基于视角序列数据进行有效报警 4](#_Toc503449398)

[On-Call轮值 4](#_Toc503449399)

[有效的故障排查手段 4](#_Toc503449400)

[紧急事件响应 5](#_Toc503449401)

[紧急事故管理 5](#_Toc503449402)

[事后总结:从失败中学习 6](#_Toc503449403)

[跟踪故障 6](#_Toc503449404)

[测试可靠性 6](#_Toc503449405)

[SRE部门中的软件工程实践 6](#_Toc503449406)

[前端服务器的负载均衡 6](#_Toc503449407)

[数据中心内部的负载均衡系统 7](#_Toc503449408)

[应对过载 7](#_Toc503449409)

[处理连锁故障 7](#_Toc503449410)

[管理关键状态:利用分布式共识来提高可靠性 7](#_Toc503449411)

[分布式周期性任务系统 7](#_Toc503449412)

[数据处理流水线 7](#_Toc503449413)

[数据完整性:读写一致 8](#_Toc503449414)

[可靠地进行产品的大规模发布 8](#_Toc503449415)

[**管理** 9](#_Toc503449416)

[迅速培养SRE加入on-call 9](#_Toc503449417)

[处理中断性任务 9](#_Toc503449418)

[通过迁入SRE的方式帮助团队从运维过载中恢复 9](#_Toc503449419)

[SRE与其他团队的沟通与协助 9](#_Toc503449420)

[SRE参与模式的演进历程 9](#_Toc503449421)

[**结束语** 9](#_Toc503449422)

[其他行业的实践经验 9](#_Toc503449423)

# 概览

## SRE方法论

**Google SRE**

传统软件工程专业花费很多精力讨论软件的开发过程，而不是其后的维护过程。有数据显示，一个软件系统40%~90%的花销其实是花在开发建设完成之后不断维护过程中的。业内流行一个错误的说法：一个系统如果已经开发完成，部署在生产缓存上，那么它就是“稳定的”，就不需要那么多工程师花费精力去优化、维护。从这个视角出发，我们认为如果软件工程职业主要专注设计和构建软件系统，那么应该有另外一种职业专注整个软件系统的生命周期管理。从其设计一直到部署，经历不断改进，最后顺利退役。这样一种职业必须具备非常广泛的技能，但是和其他职业的专注点都不同。Google将这个职位称为站点可靠工程师(SRE,Site Reliability Engineering)

传统的研发团队和运维团队分歧的焦点主要在软件新版本、新配置的变更的发布速度上。研发部门最关注的是如何能够快速地构建和发布新功能。运维部门更关注的是如何能在他们值班期间避免发生故障

SRE团队承担的职责：**可用性改进、延迟优化、性能优化、效率优化、变更管理、监控、紧急事务处理以及容量规划与管理**

**SRE方法论：**

**确保长期关注研发工作:**运维工作限制在50%以内，在每8-12小时的on-call轮值期间最多处理两个紧急事件。所有的产品事故都应该有对应的事后总结，无论有没有触发报警.

**在保障服务SLO的前提先最大化迭代速度:**“错误预算”，任何产品都是不是，也不应该做到100%可靠

**监控系统:**监控系统不应该依赖人来分析报警信息，而是应该由系统自动分析，仅当需要用户只需某个操作时，才需要通知用户

**应急事件处理:**通过将预案并且将最佳方法记录在”运维手则”上通常可使用MTTR(平均恢复时间)降低三倍以上

**变更管理:**采用渐进式发布机制、迅速而准确地检测到问题的发生、当出现问题时，安全迅速地回退改动

**需求预测和容量规划:**必须有一个准确的自然增长需要预测模型，需求预测的世界应该超过资源获取的时间。规划中必须有准确的非自然增长的需求来源的统计。必须有周期性压力测试，以便准确地将系统原始资源信息与业务容量对应起来

**资源部署:**资源的部署是变更管理和容量规划的结合物。新资源的部署和配置是一个相对比较危险的操作，必须要小心谨慎地执行

**效率与性能:**用户需求、可用容量和软件的资源使用效率。SRE通过模型预测用户需求，合理部署和配置可用容量，同时可以改进软件提升资源使用效率。通过这这三个因素能够大幅度推动一个服务的效率提升。SRE的目标是根据一个预设的延迟目标部署和维护足够的容量。SRE和产品研发团队应该共同监控和优化整个系统的性能。

## SRE视角

**硬件**

不使用专门的物理服务运行专门的软件服务器

# 指导思想

## 拥抱风险

用户在一个有着99%可靠性的智能手机上是不能分辨出99.99%和99.999%的服务可靠性的区别。SRE旨在寻求快速创新和高效的服务运营业务之间的风险的平衡，而不是简单地将服务在线时间最大化。

**管理风险:**不可靠的系统会很快侵蚀用户的信心。所以我们要减少系统出故障的几率。

**度量服务的风险:**通过一个客观的指标来提现一个待优化的系统属性

**服务的风险容忍度:**根据基本产品或服务的定义建立，将一组商业目标转化为明确的可以实现的工程目标

**成本:**构建和运维可用性再多一个”9”的系统，收益会增加多少。额外的收人是否能够抵消了达到这一可靠性水平所付出的成本

## 服务质量目的

**指标:**SLI是指房屋质量指标—--该服务的某项服务质量的一个具体量化指标，大部分服务都将请求延迟—--处理请求所消耗的时间—--作为一个关键SLI。其他常见的SLI包括错误率(请求处理失败的百分比)、系统吞吐量(每秒请求数量)等

## 减少琐碎事

SRE要把更多的世界花费在长期项目研发上而非日常运维中。

琐事就是运维服务中手动性的，重复性的，可以被自动化的，战术性，没有持久性价值的工作。而且，琐事与服务呈线性关系的增长

## 分布式系统的监控

**监控:**收集、处理、汇总，并且显示关于某个系统的实时量化数据，例如请求的数量和类型，错误的数量和类型，以及处理用时，应用服务器的存活时间等

**白盒监控:**依靠内部暴露的一些性能指标进行监控。包括日志分析、Java虚拟机提供的监控接口，或者一个列出内部统计数据的HTTP接口进行监控

**黑盒监控:**通过测试某种外部用户可见的系统行为进行监控

**监控台页面:**提供某个服务核心指标一览服务的应用程序。该应用程序可能会提供过滤功能、选择功能等，但是主要的功能是用来显示系统最重要的指标

**四个黄金指标:**延迟、流量、错误、饱和度

## 自动化系统的演进

**自动化的价值:**一致性、平台性、修复速度更快、行动速度更快、节省时间

## 发布工程

**发布工程哲学:**自服务模型、追求速度、密闭性、强调策略和流程

**持续构建与部署:**构建、分支、测试、打包、部署、配置管理

## 简单化

可靠性只有靠最大程度的简化不断追求而得到

软件系统本质上是动态的和不稳定的，只有真空的软件系统才是永远稳定的，如果我们不在修改代码，就不会引入新的Bug。如果底层硬件或类库永远不变，这些组件要也就不会引入Bug。如果冻结当前用户群，我们将不必扩展系统。我们的工作最终是在系统的灵活性和稳定性上维持平衡

**系统的稳定性与灵活性:**可靠的流程会提高研发人员的灵活性，快速、可靠的产品发布会使得生产系统中的变化显而易见

**最小化以为复杂度:**在所负责的系统中引入意外复杂度时，及时提出抗议。不断地努力消除正在接受的和已经负责运维的系统的复杂度

**最小化API:**我们向API消费者提供的方法和参数越少，这些API就越容易理解，我们就能用更多的精力去尽可能地完善这些方法

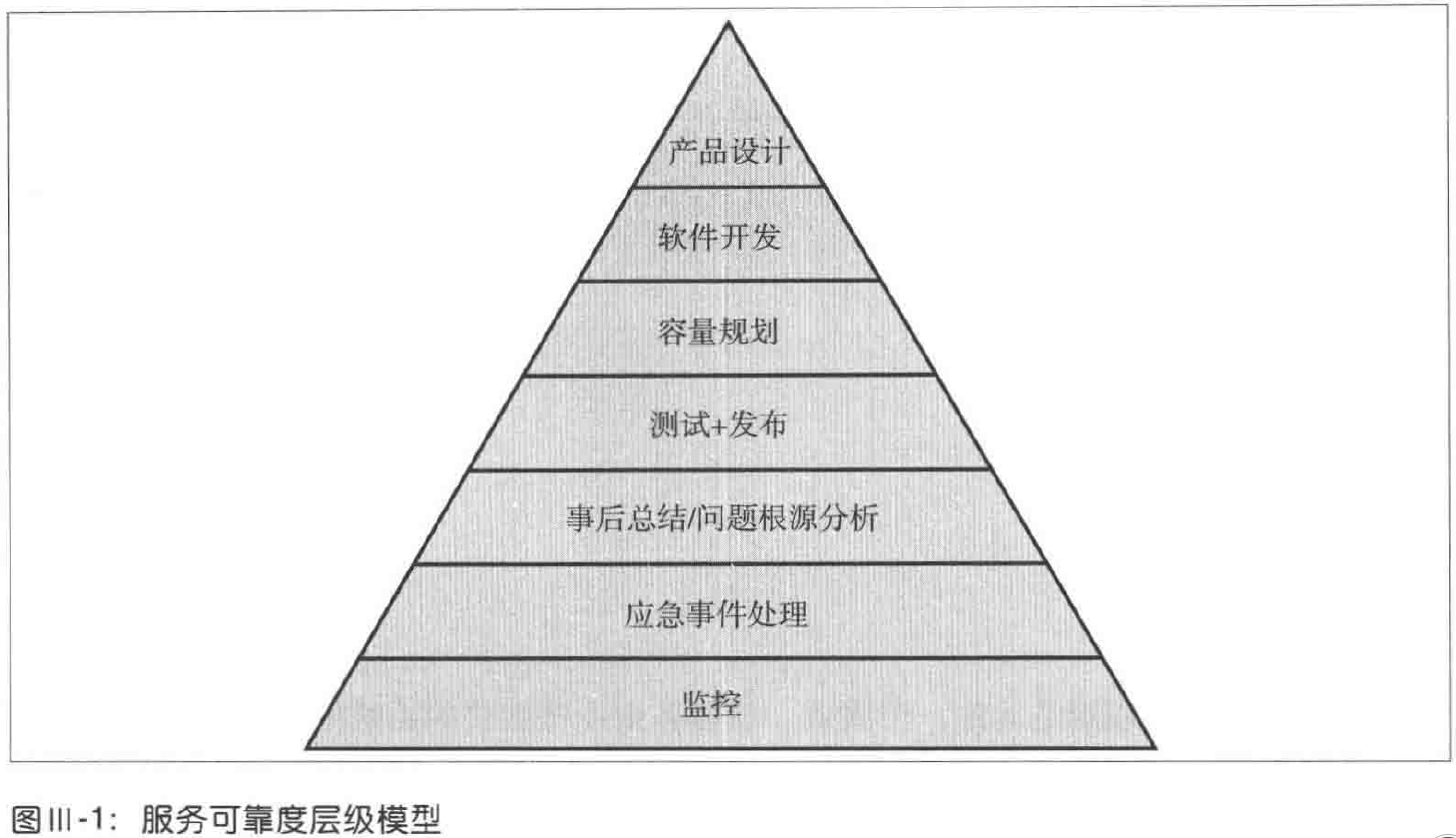
**模块化:**通过将API版本化，可以允许开发人员继续使用它们的系统所依赖的版本，以更安全和深思熟虑的方法升级到新的版本。这样不必要求整个系统的每一次功能增加或改进都需要全面的生产更新，整个系统中的更新节奏可以不同

**发布的简单化:**如果我们发布是按更小的批次进行的，我们可以更有信心地进行更快的发布，因为每个变更在系统中的影响可以独立理解。我们通过每次进展一点，同时考虑每次变更对系统的改善和退化来寻找最近方案

软件的简单性是可靠性的前提条件，明确实际上要完成的任务是什么，以及如何更容易地做到。

# 具体实践

SRE的职责是运维一个服务。终极职责是确保该服务可以正常运转



SRE一旦发现了系统中存在的问题，正确的解决方案不一定是当场把问题一次性修复好，而可以靠降低系统准确度、关闭一些不重要的功能，或者将用户流量导向其他没有问题的任务实例等手段暂时缓解问题。事后总结和问题根源分析

## 基于视角序列数据进行有效报警

**监控:**处于整个生产环境需求金字塔模型的最底层。监控是运营一个可靠的稳定服务不可缺少的步伐。服务运维人员依靠监控数据对服务的情况作出理性判断，用科学的方法应对紧急情况。同时，监控数据页可以用来确保服务质量与产品目标保持一致

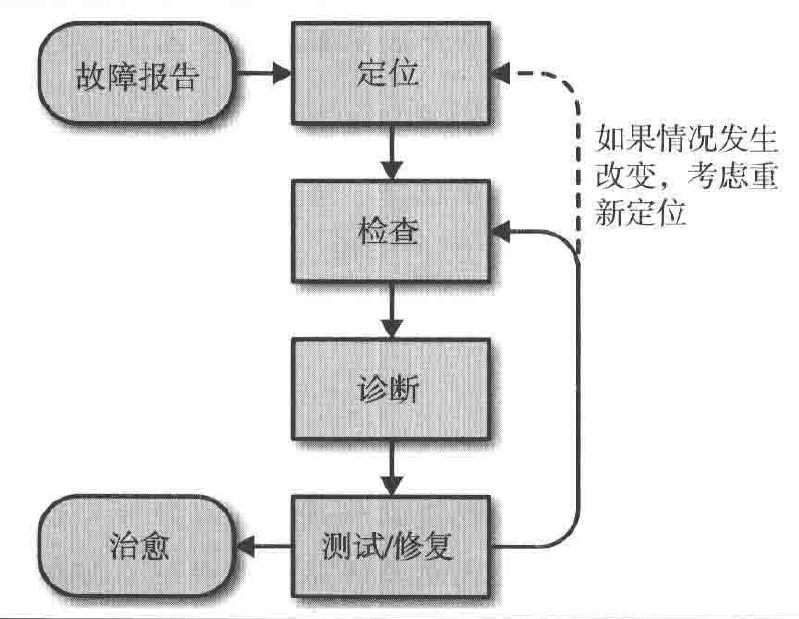
一个大型系统不应该要求运维人员持续关注其中使用的无数小组件，而是应该自动汇总所有的信息，自动抛弃其中的异常情况。监控系统应该主要从高级服务质量目标层面进行报警，但是也应该保持足够的粒度，可以追踪到某个具体组件

## On-Call轮值

On-call模式使Google可以用工程化的手段解决服务扩展性问题，在服务变得越来越来复杂、SRE团队负责的组件越来越多的情况下仍可保持相当高的可用性和可靠性

## 有效的故障排查手段

故障排查是运维分布式计算系统的一项关键技能

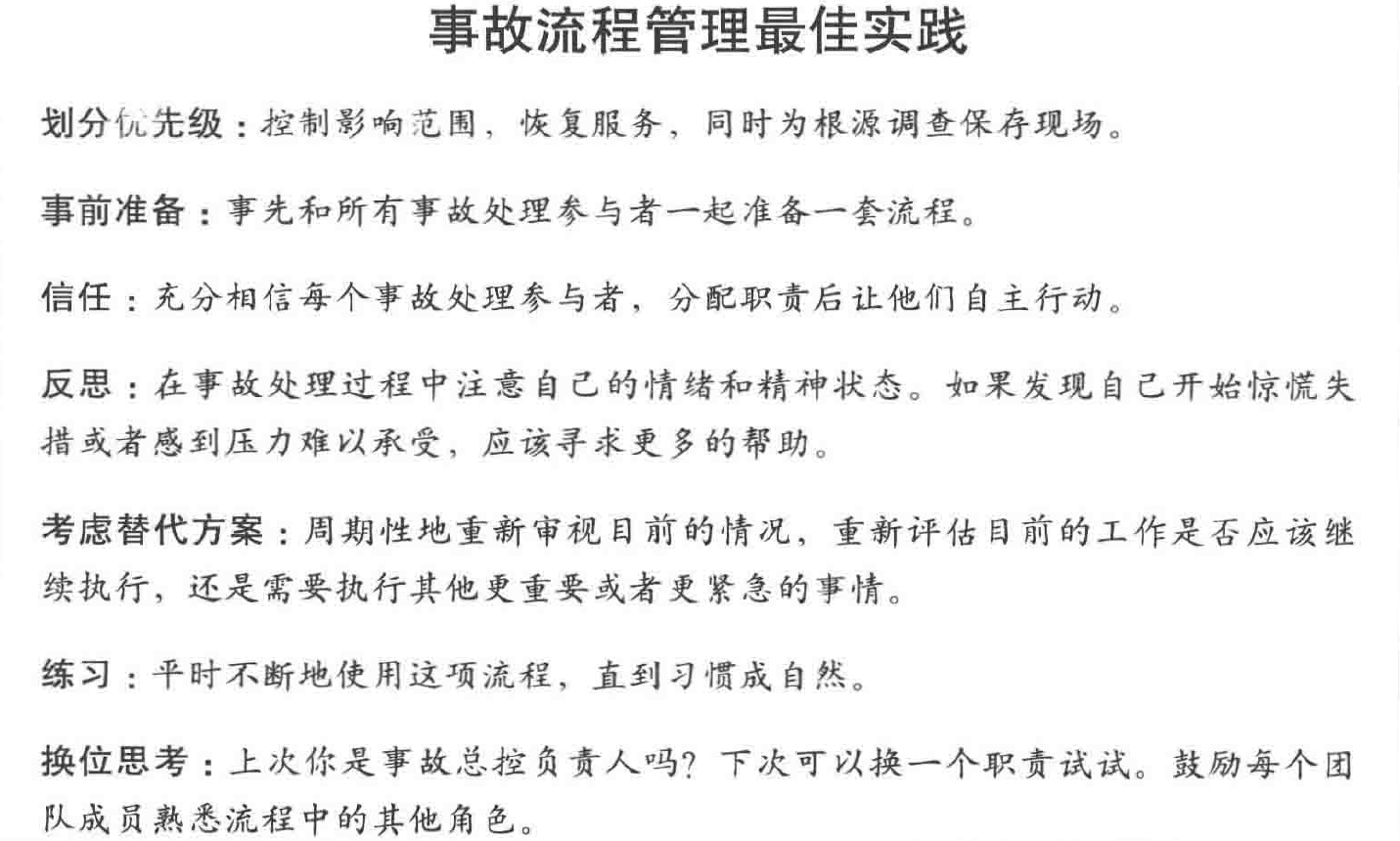


## 紧急事件响应

东西早晚要坏的，这就是生活。不管一个组织有多大，做的事情有多么重要，它最明显的特质就是:在紧急事件来临时人们如何应对。没有几个人天生就能很好地处理紧急情况。在紧急情况下恰当处理需要平时不断地进行实战训练。建立和维护一套完备的训练和演习流程

## 紧急事故管理

有效的紧急事故管理是控制事故影响和迅速恢复运营的关键因素。如果事先没有针对可能发生的紧急事故进行过演习，那么当事故发生时，一切管理理念都起不了作用



## 事后总结:从失败中学习

学习是避免失败的最好办法。在事故发生后，我们要修复根源性问题，同时将服务恢复到正常状态。如果没有一种方法从已发生的事故中学习经验，那么事故就可能循环反复地发生。

书写事后总结的主要目的是为了保证该事故被记录下来，理清所有的根源性问题，同时最关键的是，确保实施有效的措施使得未来重现的几率和影响得到降低，甚至避免重现

**基本事后总结条件:**

用户可见的宕机时间或者服务质量降低程度达到一定标准

任何类型的数据丢失

On-call工程师需要人工介入的事故(包括回滚、切换用户流量等)

问题解决耗时超过一定限制

监控问题(预示着问题是由人工发现的，而非报警系统)

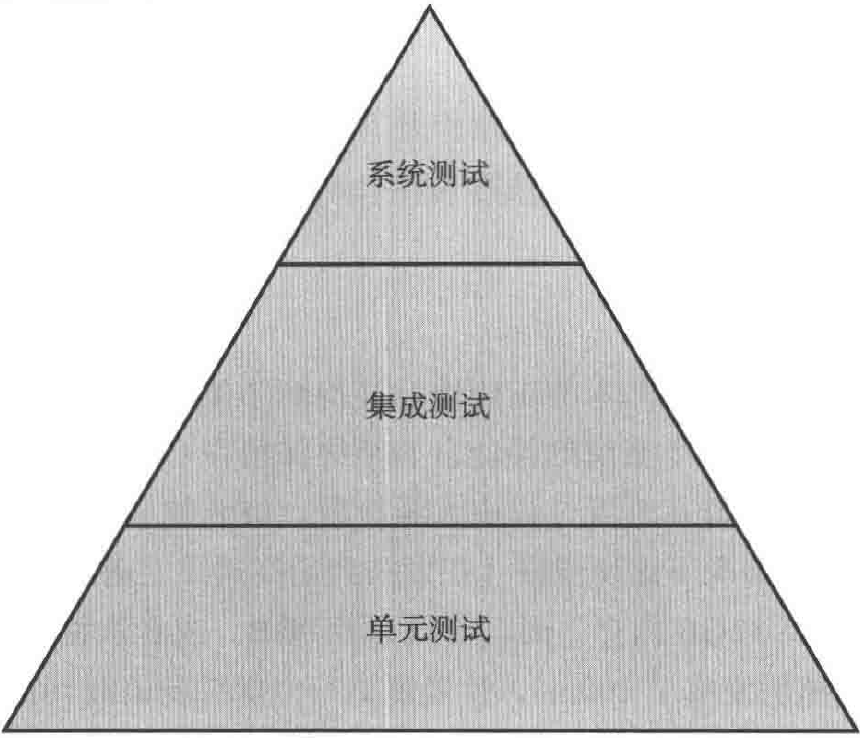
在事故发生前定义好事后总结的标准是很重要的，这样每个参与事故处理的人都知道是否应该书写书面报告

## 跟踪故障

提高可靠性的唯一可靠的方法论是建立一个基线，同时不断跟踪改变

## 测试可靠性

如果你还没有亲自试过某件东西，那么就假设它是坏的



## SRE部门中的软件工程实践

软件工程项目在Google SRE组织内部随着人数增多而增加，成功与失败的案例为以后的项目铺平了道路

## 前端服务器的负载均衡

通过负载均衡屏蔽后端故障

## 数据中心内部的负载均衡系统

在实践中，加权轮询策略效果非常好，极大降低了最高和最低负载任务的差距

## 应对过载

避免过载，是负载均衡策略的一个重要目标。应对过载的应该选项是服务降级

按照QPS来规划服务容量，或者是按照某种静态属性一般是错误的选择。更好的解决方案是直接可用资源来衡量可用容量

## 处理连锁故障

连锁故障是由正反馈循环导致的不断扩张规模的故障，连锁故障可能由于整个系统的一小部分出现故障而引发，进而导致系统其他部分也出现故障

当一个系统过载时，某些东西总是被牺牲掉。一旦一个服务越过了临界点，服务一些用户可见错误，或者低质量结果要比尝试继续服务所请求要好

## 管理关键状态:利用分布式共识来提高可靠性

SRE需要针对可能遇到的灾难做好事先准备，预先制定应对策略，以保障在灾难来临事系统仍能正常运行。这些应对策略基础需要将服务分布式运行。跨物理区域分布式运行系统相对来说是比较简单的，但是却带来维护系统一致状态视图的需求

## 分布式周期性任务系统

随着行业向大型分布式系统迁移，数据中心逐渐成为事实上最小的硬件单元，也就意味着技术栈中很多部分都要改变

## 数据处理流水线

周期性的数据流水线是非常脆弱易坏的。我们发现当第一次安装周期性流水线时，工作进程数量、运行周期、分块处理技术，以及其他参数仔细调校过后。整个流水线的性能可能跟稳定。但是随着数据量的自然增长等种种变化会对整个系统造成压力，导致各种各样的问题出现。这种问题包括任务运行超时、资源耗尽，某些分开处理卡住导致整体运维压力上升等

周期性的数据流水是很有价值的。但是如果一个数据处理问题本身是持续性的，或者会自然增长成为持续性的，那么就不要采用周期性的设计方式，而是采用一种类似Workflow的设计特点的系统

带有强一致性的持续数据处理，就行Workflow这样，在分布式集群环境中工作性能和扩展性都非常好。一个系统能周期性地提供用户可依赖的结果，并且是一个非常可靠且稳定的可运维系统

## 数据完整性:读写一致

数据完整性是指数据存储为了一个合理的服务质量，在可访问性和准确性方面必须达到的一个度量标准

备份不重要，重要的是恢复

数据可用性必须作为任何以数据为中心的系统的首要重点

## 可靠地进行产品的大规模发布

**流程与自动化:**使用标准工具来自动化一些常见流程

将所有需要手动执行的流程文档化

将迁移到另外一个数据中心的流程文档化

将后就和发布新版本的流程自动化

**功能开关框架**

可以同时发布多个改动，每个改动仅仅针对一部分服务器、用户、是他，或者数据中心起作用

灰度式发布到一定数量的用户，一般在1%~10%之间

将流量根据用户、对话、对象和位置等信息发送到不同的服务器上

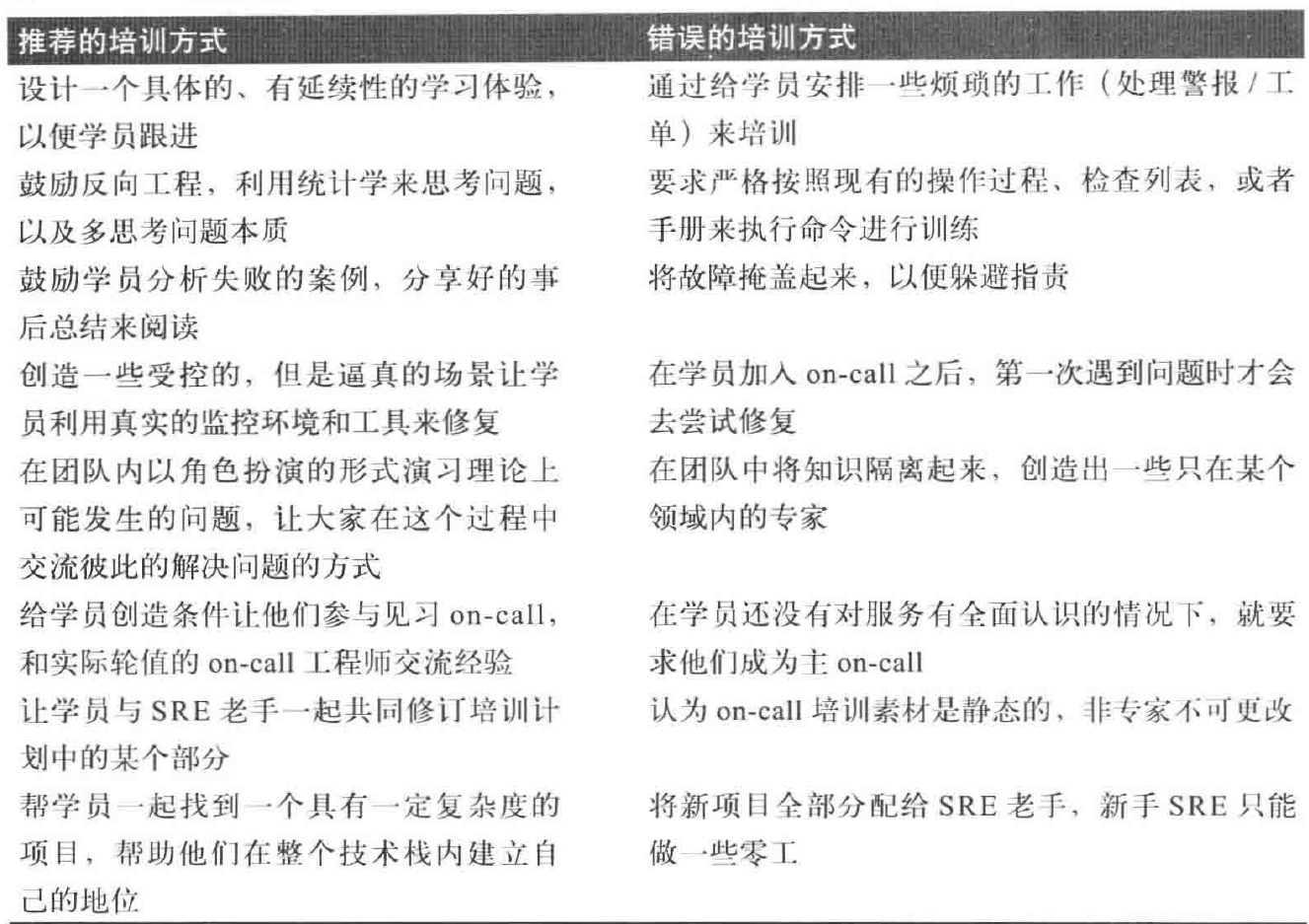
设计中可以自动应对新代码出现的问题，不会影响到用户

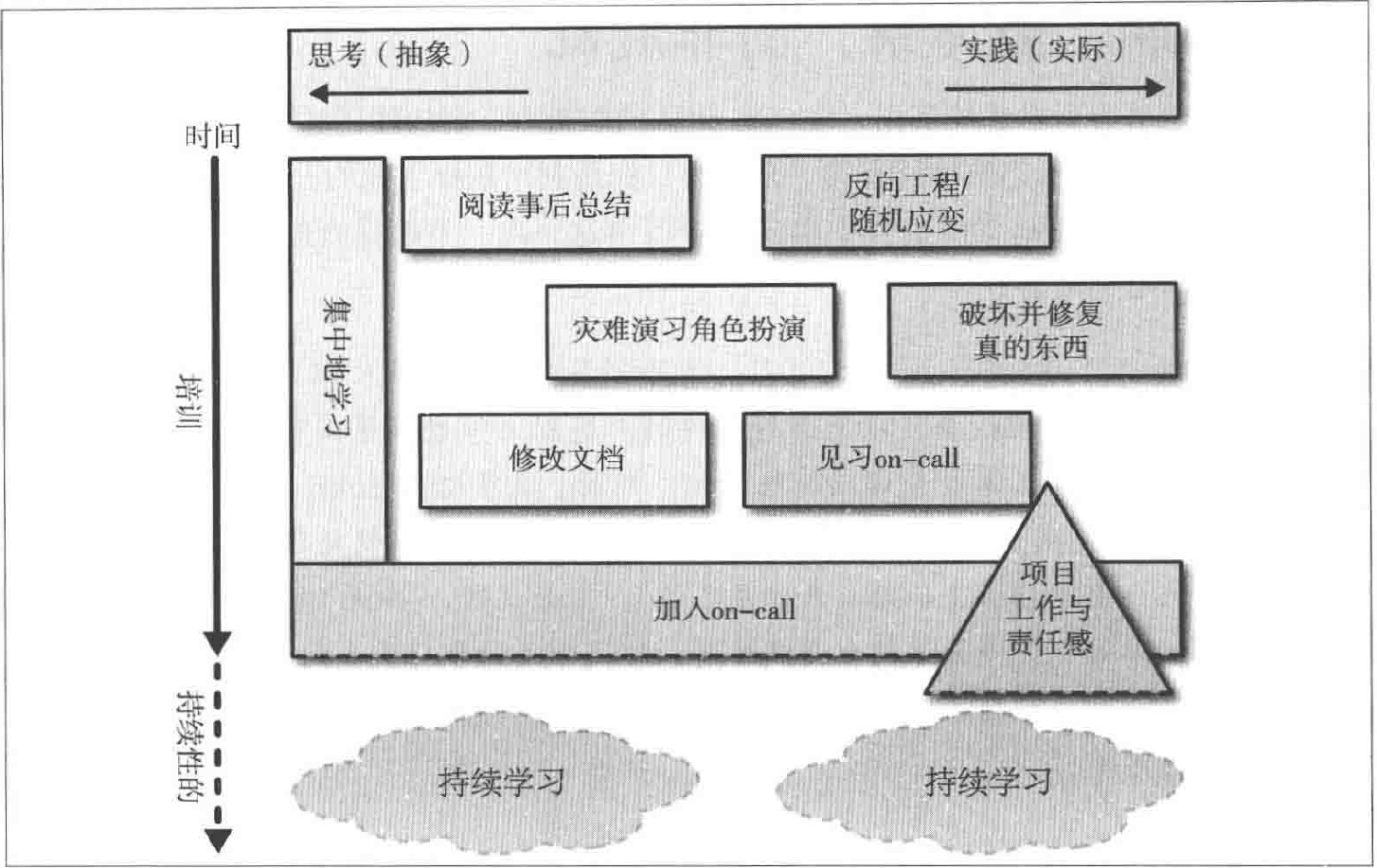
在严重Bug发送，或者其他副作用场景下可以迅速单独屏蔽某个改变

度量每个改动对用户体验的提升

# 管理

## 迅速培养SRE加入on-call





作为SRE，我们必须要能够以比扩张机器更快的速度扩张我们的团队

## 处理中断性任务

如何决策对中断性任务的处理策略：

中断任务的SLO，即预期的响应时间

排队的中断性任务有多少

中断性任务的严重程度

中断性任务的发生频率

可以处理某种中断性任务的人有多少

## 通过嵌入SRE的方式帮助团队从运维过载中恢复

从技术角度，最好是量化的角度指出团队需要改变的原因

提供一个详细、具体的”改变”作为例子

介绍SRE经常采用的”常识”背后的逻辑推理过程

提供以可伸缩的方式来解决崭新情况所必需的核心理念

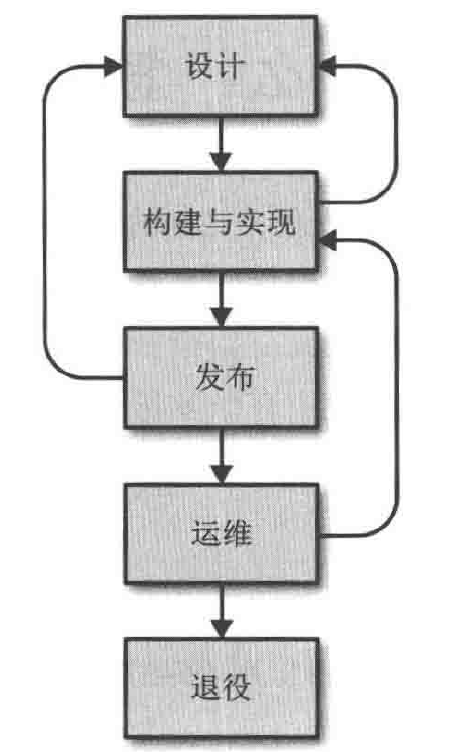
## SRE与其他团队的沟通与协助

SRE团队之间的协作存在一定的困难，但是也存在巨大回报。这种回报包括建立一套解决问题的通用方法，已经可以集合更多力量解决更困难的问题

## SRE参与模式的演进历程

错误越早被发现，就越容易修复它。SRE参与的时机越早，该服务的质量提升越快，最终质量也越好

生产就绪程度评审(PRR)



**SRE参与模型:**SRE承担重要服务的生产运维的责任，是为了提高该服务的可靠性

系统的体系结构和跨服务依赖

指标的选择、度量和监控

紧急事件处理

容量规划

变更管理

性能:可用性、延迟和资源效率

服务的可靠性可以通过SRE的参与来改建，该流程包括系统化的评审与生产运维方面的不断改进。Google SRE最初的系统化流程---简单PRR模型，在规范SRE参与模式中取得了长足的金币，但仅适合于已经进入发布阶段的服务

# 结束语

## 其他行业的实践经验

**指导思想：**

灾难预案与演习

书写时候总结的文化

自动化与降低日常运维负载

结构化的、理智的决策

**灾难预案与演习：**

确保系统按我们预想的方式应对故障

寻找系统中未预料到的弱点

寻找其他提高系统鲁棒性的方式来避免事故发生

**事后总结的文化:**

究竟发生了什么

响应的有效程度

下次是否可用采用其他方案解决问题

如何确保这次故障不会再次发生

**将重复性工作自动化，消除运维负载:**

**系统可用性**

