# 阿里技术专家对 SRE 的解读

原创

阿里云开发者 于 2021-01-20 10:55:28 发布 1161 收藏 5
文章标签： 存储 运维 监控 数据可视化 安全 API 项目管理 数据安全/隐私保护

本文链接：https://blog.csdn.net/alitech2017/article/details/112859956


> **简介：**产品/基础技术研发 和 SRE 这两类角色是相互协作、相互服务的关系，拥有共同的目标：满足业务需求，更好服务业务。

## 前言

在技术工作中，对于产品/基础技术研发和 SRE 两种角色，通常会有基于「是否侧重编码」的理解。对于产品研发转做 SRE，经常会产生是否要「脱离编码工作」的看法，或者认为是否要「偏离对产品/基础技术的推进」。

基于过往的技术研发和稳定性保障的经验，分享下个人对 SRE 的理解，探讨「面向产品/基础技术的研发」和「稳定性保障」两种角色之间的协作关系，更好地为业务服务。

## SRE 概述

最早讨论 SRE 来源于 Google 这本书《Site Reliability Engineering: How Google Runs Production Systems》（豆瓣链接）。由 Google SRE 关键成员分享他们是如何对软件进行生命周期的整体性关注，以及为什么这样做能够帮助 Google 成功地构建、部署、监控和运维世界上现存最大的软件系统。

从 wikipedia: Site reliability engineering中可了解到 SRE 的定义：

> Site reliability engineering (SRE) is a discipline that incorporates aspects of software engineering and applies them to infrastructure and operations problems. The main goals are to create scalable and highly reliable software systems.

其中有句形象描述 SRE 工作的描述：

> SRE is "what happens when a software engineer is tasked with what used to be called operations."

即 SRE 的目标是构建可扩展和高可用的软件系统，通过软件工程的方法解决基础设施和操作相关的问题。

在 Google SRE 书中，对 SRE 日常工作状态有个准确的描述：至多 50% 的时间精力处理操作相关事宜，50% 以上的精力通过软件工程保障基础设施的稳定性和可扩展性。

基于上述描述，我对 SRE 的理解是：

- 职责：保障基础设施的稳定性和可扩展性
- 核心：解决问题
- 方法：通过操作类事务积累问题经验，通过编码等方式提升问题的解决效率

## 软件生命周期

Google SRE 一书中，对软件工程从生命周期角度有一个很形象的描述：

> 软件工程有的时候和养孩子类似：虽然生育的过程是痛苦和困难的，但是养育孩子成人的过程才是真正需要花费绝大部分精力的地方。
> 一个软件系统的 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。

项目生命周期中，设计和构建软件系统的时间精力占比，通常是少于系统上线之后的维护管理。为了更好地维护系统可靠运行，需要考虑两种类型的角色：

- 专注于设计和构建软件系统
- 专注于整个软件系统生命周期管理，包括从其设计到部署，历经不断改进，最后顺利下线

第一类角色对应产品/基础技术研发，第二类角色对应 SRE，二者的共同目标均是为了达成项目目标，协同服务好业务。

## 稳定性保障价值

针对稳定性的影响，直接参与处理客户问题的同学会更有体感：

- 通过问题发生时客户直接反馈的影响程度、紧急程度，感受到稳定性给客户带来的焦虑
- 通过问题处理结束后客户的反馈，感受到客户对稳定性保障的感谢或愤怒
- 通过事后在营收状况、客户规模变化，感受到稳定性对业务营收的影响
- 通过产品规划的的延期，感受到稳定性对产品迭代的影响

稳定性保障的价值由此凸显：

- 保障客户的产品体验，满足客户对约定的可靠性诉求
- 加速业务迭代，满足业务对稳定性诉求，业务注意力集中在更快速推出满足客户需求的功能

## SRE 如何保障稳定性

稳定性问题通常会有这些特征：

- 人为导致，依赖专家经验
- 一系列因素综合导致
- 不可避免
- 100% 保障没有必要

线上稳定性问题，人为操作不当导致的比例很高，集中在 **发布** 和 **线上运维** 两个环节，均是高频操作。对于复杂系统，这两个环节对专家经验有较强的依赖。

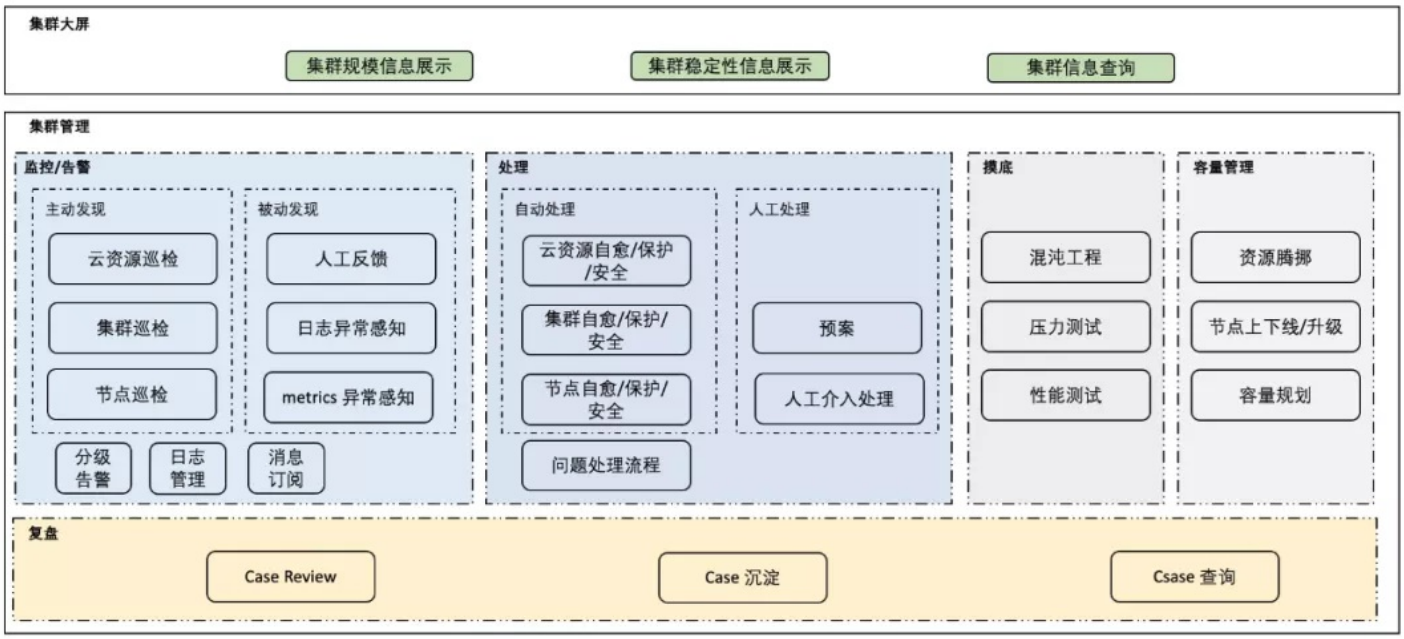
发生的稳定性问题通常具有系统性的特征，即非单个功能组件缺陷导致，而是由一系列因素综合作用导致，如缺少监控告警导致不能及时感知，缺少日志不能有助于快速定位问题，缺少良好的问题排查流程导致依赖个人能力，缺少良好的协调沟通极致导致问题处理时长增加、客户影响程度加剧等。

问题是不可避免的，流量的突增、服务器/网络/存储的损坏、未覆盖的输入等，均会诱发问题的出现。

业务对外有 SLA，向客户承诺一定程度的稳定性，未达到时按照协议进行赔付，同时问题又不可不免，在满足内部 SLO 标准的前提下继续提升稳定性，会带来更高的实现成本，对业务的收益增量也会更小。

SRE 需要对问题特征有深入理解，系统性设计和实施解决方案，并抓住一段时间内的主要问题进行解决。

一种可参考的整体解决方案如下：




**落地过程中，可先从如下三个抓手系统解决：**

- 可控性
- 可观测
- 稳定性保障最佳实践

**可控性方面，包括如下三个主要维度：**

发布管理

- 重点解决发布导致的人为稳定性问题
- 包括发布前重要变更评审和发布中变更动作管理等

操作管理

- 重点解决黑屏操作导致的人为稳定性问题
- 包括统一集群操作入口、集群操作权限管理、集群操作审计等

设计评审

- 重点解决软件系统设计阶段应用稳定性保障最佳实践
- 包括集群方案评审和重要功能设计评审等

**可观测方面，包括如下几个重要维度：**

监控

- 重点解决软件系统运行态的感知能力
- 包括监控收集/可视化系统的搭建和维护等

日志

- 重点解决软件系统的问题可排查能力
- 包括日志收集/存储/查询/分析系统的搭建和维护等

巡检

- 重点解决软件系统功能是否正常的主动探测能力
- 包括巡检服务的搭建、通用巡检逻辑的开发维护等

告警

- 重点解决异常的及时触达需求
- 包括告警系统的搭建、告警配置管理、告警途径管理、告警分析等

**稳定性保障最佳实践**，是从历史问题和业界实践方面抽象出意识、流程、规范、工具，在系统设计之初就融入其中，并在系统整个生命周期中加以使用，如通过模板固化最佳实践：

- 项目质量验收标准
- 项目安全生产标准
- 项目发布前 checklist
- 项目 TechReview 模板
- 项目 Kick-off 模板
- 项目管理规范
- etc.

一个例子：

| 维度 | 评估项 |
| --- | --- |
| 可观测 | 1. 是否提供了标准的 metrics API?<br>2. 是否可以将日志持久化到日志系统？<br>3. 是否配置了监控？<br>- a. 是否有对跌零因子的监控？<br>- b. 是否有服务降级的监控？<br>- c. 是否有限流的监控？<br>- d. 是否配置了对关键依赖方的可用性监控？<br>- e. 监控是否分级？<br>4. 是否配置了告警？<br>- a. 是否有对跌零因子的告警？<br>- b. 是否有服务降级的告警？<br>- c. 是否有限流的告警？<br>- d. 是否配置了对关键依赖方的可用性告警？<br>- e. 告警是否分级？<br>5. 是否可以配置巡检？<br>6. 使用了 structured log 便于进行 log 的查询、分析？ |

| 维度 | 评估项 |
| --- | --- |
| 可灰度 | 是否使用了具有灰度能力的 workload？ |
| 可回滚 | 1. 是否使用了均有回滚能力的 workload？<br>2. 组件是否进行了版本控制？<br>3. 配置是否进行了版本控制？ |
| 可保护 | 1. 是否有限流措施？<br>2. 是否有降级措施？<br>3. 是否配置了探针？<br>- a. 是否配置了 livenessProbe？<br>- b. 可被访问时，是否配置了 readinessProbe？<br>- c. 初始化耗时时，是否配置了 startupProbe？<br>4. 是否有快速失败的能力？<br>- a. 是否有超时结束能力？<br>5. 依赖方不可用时:<br>- a. 是否会持续对依赖方带来日常或更高压力？<br>- b. 是否会对上游带来反向压力？(如连接数、处理延时)<br>6. 己方不可用时：<br>- a. 对上游的影响是否可控？<br>- b. 恢复时是否可以控制请求压力？<br>7. 是否可以无损重建？<br>8. 是否多副本部署？<br>9. 是否配置了非亲和性？<br>10. 是否跨 AZ 部署？<br>11. 是否有处理预案<br>12. 是否均有访问管理？<br>13. 服务是否稳定性运行，是否会影响数据资产？<br>14. 服务是否稳定性运行，是否会泄露敏感信息？<br>15. 是否区分组件处于关键链路还是旁路？<br>16. 是否有「爆炸半径」的整理？<br>17. 是否有「跌零因子」的整理？ |
| 可控成本 | 1. 是否配置了 limit resources？<br>2. 变更是增加还是降低用户成本？<br>3. 变更是增加还是降低平台成本？ |
| 易于运维 | 1. 是否可以做到变更配置时无需重建实例？<br>2. 是否有白屏化运维途径？<br>3. 是否有「端到端管控链路」流程图<br>4. 是否有「端到端数据链路」流程图 |

为了便于理解，可以再针对 check 项形成分级，便于交流和进行项目稳定性评估：

| 级别 | 标准 |
| --- | --- |
| L0 | 1. 可观测、可灰度、可回滚 均不满足 |
| L1 | 1. 可观测、可灰度、可回滚 满足 50% 以上要求 |
| L2 | 1. 可观测、可灰度、可回滚 满足 90% 以上要求 |
| L3 | 1. 可观测、可灰度、可回滚 满足 90% 以上要求<br>2. 可保护满足 50% 以上要求 |
| L4 | 1. 可观测、可灰度、可回滚 满足 90% 以上要求<br>2. 可保护满足 90% 以上要求<br>3. 可控成本满足 50% 以上要求 |

| 级别 | 标准 |
| --- | --- |
| L5 | 1. 可观测、可灰度、可回滚 满足 90% 以上要求<br>2. 可保护满足 90% 以上要求<br>3. 可控成本满足 90% 以上要求 |

当最佳实践可以通过文档进行规范化，接下来就可以提供工具或服务将其低成本应用，使得稳定性保障最佳实践成为基础设施。

SRE 需要在稳定性相关的方法论和实践方面不断迭代，自上而下设计，自下而上反馈，合理、可靠保障稳定性。

## 共赢，携手服务业务

再回顾下软件系统生命周期中的两类角色：

- 产品/基础技术研发：专注于设计和构建软件系统
- SRE：专注于整个软件系统生命周期管理，包括从其设计到部署，历经不断改进，最后顺利下线

这两类角色是相互协作、相互服务的关系，拥有共同的目标：**满足业务需求，更好服务业务。**

SRE 通常会横向支撑多个项目，对线上问题的类型、解决实践有更为全面的理解和思考，基于此会形成最佳实践的理论、工具或服务，为研发提供理论、工具的支持，也可以在此基础上产品化稳定性保障解决方案，为更多的客户服务，创造更大的价值。

产品/基础技术研发对业务需求、功能/技术细节有更深入的理解，一方面直接带来业务价值，一方面可通过实践为稳定性保障带来切合实际的需求，进一步和 SRE 共同保障稳定性。

**两种类型的角色，需要朝着共同的目标并肩协作，与业务共同发展，实现共赢。**

## 小结

SRE 由于工作的性质，在横向方面会服务大量的业务，以实践积累对稳定性保障问题域的深入理解和稳定性保障重要性的深刻认知，在纵向方面会通过技术手段将稳定性保障最佳实践进行沉淀和应用；同时眼光又是与研发、业务一齐向前看，综合技术和管理创造价值。

以上是从个人角度对 SRE 及稳定性保障的理解，重点在于 **解决问题** 和 **创造更大的价值**。

## References


豆瓣 SRE
wikipedia: Site reliability engineering
wikipedia: Controllability
wikipedia: Observability
site: google sre


---

**扫码了解更多技术内容与客户案例：**