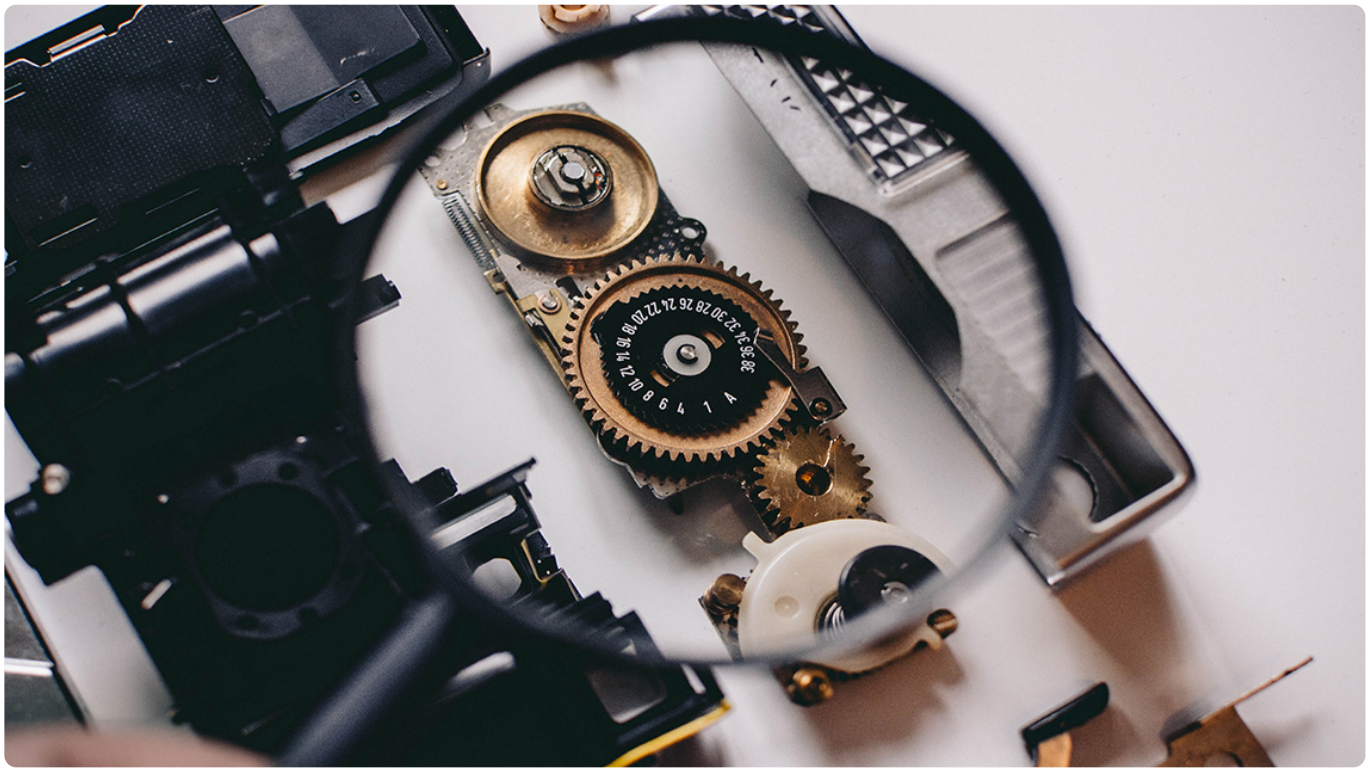
50 | 日志、监控与报警

2019-10-18 许式伟



讲述：姚迪迈
时长 17:14 大小 15.79M

你好，我是七牛云许式伟。

上一讲我们介绍了发布与升级，这是一项复杂的事务，有非常长的业务流程，包括：构建、测试、打包、部署以及配置变更。但总体上来说，发布与升级在 SRE 的工作范畴中，还并不是最难工程化的事务工作。我们简单分析就可以明白：发布与升级总体上来说，只和集群中服务之间的调用关系有关，而与具体服务的业务特性没有太大的相关性。

真正难工程化的是监控与报警。

好的监控与报警系统是怎样的？

监控一个复杂的业务系统本身就是一项极其复杂的事务。即使在具有大量现成的基础设施的情况下，设计监控项、收集数据指标、显示并提供报警支持这些工作，通常需在 10 人的 SRE 团队中，可能就需要有 1 ~ 2 人全职进行监控的构建和维护工作，这些工作都是非常定制化的，与具体业务密切相关。

如果我们把服务比作一个人的话，发布与升级更像是一个交通工具，尽管内部机制也许复杂，但是从功能上来说，它只是把我们载到某个目的地，人的个性在这里并不需要得到充分的重视。

但监控与报警不同，它更像是私人医生，需要因人而异，因地制宜，提供一套完整的健康保障方案。

监控与报警的目标是什么?

简单说，监控的核心目标就是要解决好两个问题：其一，什么东西出故障了。其二，为什么它出故障了，根因在哪里。

一个监控与报警的系统（简称监控系统），在于它可以让一个业务系统在发生故障时主动通知我们，或者告诉我们即将发生什么。当系统无法自动修复这个问题时，通过报警引导 SRE 人工来介入。SRE 首先判断目前是否真实存在某个故障，如果存在故障则采取手段来消除故障，解决故障后最终还要找出导致故障的根源问题，并推动各方去彻底解决掉它，避免未来出现类似的问题。

一个初级的监控系统是比较好做的。我们坚持不懈地往里面加各种监控项，系统看起来越来越完善。很多人会由此而感觉到了满足。

但是，一个有 “完善的监控项” 的监控系统，是不是就是一个好的监控系统了?

这当然不是。

要做好监控，我们一定要分清楚什么是现象，而什么是原因。

某个监控项不正常了，“某个东西出故障了”，这只是一种现象（Symptom）。但这个监控项 “为什么出问题” 了，则是原因。我们找到的原因，也可能只是中间原因，而不是根因（Root Cause）。

在一个复杂系统中，某一个服务的现象可能是另外一个服务的原因。例如，数据库性能问题。数据库读操作很缓慢是数据库 SRE 角度检测到的一个现象。然而，对前端业务的 SRE 来说，他们看到的是网站缓慢这个现象，而数据库读操作的缓慢则是原因。当然数据库慢只是一个中间原因，它还无法对应到一个明确的动作，但这对缩小问题定位范围已经产生了巨大作用。

一个完善的监控系统，并不是 “报警很多很完善” 的系统，而是 “信噪比高，有故障就报警，有报警就直指根因” 的监控系统。

“信噪比高”关注的是误报率问题。

我们不应该仅仅因为 “某东西看起来有点问题” 就发出报警。报警处理会占用员工的宝贵时间。如果该员工正在工作时间段，该报警的处理会打断他原本的工作。如果该员工正在家，该报警的处理则会影响他的个人生活，甚至是把他从睡眠中叫醒。

当报警出现得太频繁时，员工就会进入“狼来了”效应，怀疑报警的有效性甚至忽略报警。有的时候在报警过多的时候，甚至会忽略掉真实发生的故障。由于无效信息太多，分析和修复可能会变慢，故障时间也会相应延长。高效的报警系统应该提供足够的信息，并且误报率非常低。

“有故障就报警” 关注的是报警的覆盖率。如果我们通过客户报障或其他手段发现故障，对于监控系统来说，就应该认为是一次监控事故。

“有报警就直指根因” 关注的是报警的有效性和排障的效率。一个初级的监控系统，往往很容易产生的报障现象是，线上发生一个故障会同时会产生大量的报警，这些报警杂乱无章，接警人看到一堆报警后，并没有有效的信息指引他如何快速消除故障，并找到故障的根本原因。

日志：监控与报警的基础

一个现代化的监控与报警系统，最底层往往基于一个日志系统。什么是日志？它不局限于业务服务输出的程序日志，实际上更多的数据来源是各种系统指标的采集。简单说，凡是时序相关的、持续产生的数据，都可以称之为日志。

原始的日志有可能是结构化的，也可能是非结构化的。如果是非结构化的数据，那这就需要先经过文本解析过程进行结构化。结构化后的日志存储系统，本质上就是一个时序数据库。

日志通过收集、结构化、清洗等步骤后，就可以对外提供日志分析和检索服务。分析和检索的结果可以直接提供数据结果，也可以用报表形式呈现，或者在满足特定条件下触发报警。

采用时序数据库来做监控系统的好处是，不依赖特定的脚本来判断系统是否正常工作，而是依赖一种标准数据分析模型进行报警。这就使得批量、大规模、低成本的数据收集变得可能。

收集回来的数据可以同时用来渲染图表和报警。报警规则往往用简单的数学表达式形式表达。这样，数据收集过程就不再是一个短暂的一次性过程，所有收集回来的历史数据都可以被用来作为报警规则的计算因素。

不同监控规则产生的报警（alert）可能有不同的优先级，比如紧急状态和一般状态。紧急状态的报警通常意味着已经产生或即将产生用户可见的影响，需要立刻介入处理。而一般状态则可以延迟到第二天再处理。报警的目标对象不一定是某个人，而可能是某个系统，比如工单。

当然，监控一个指标并不一定是出于报警的考虑。它还可以有各种原因，如下：

- 分析长期趋势。例如每日活跃用户的数量，以及数量增长的速度。
- 跨时间范围的比较，或者是观察 AB 测试组之间的区别。例如，增加新节点后，memcache 的缓存命中率是否增加？网站是否比上周速度要慢？使用 A 方案和 B 方案哪个更有助于用户的活跃？
- 临时性的回溯分析，即在线调试。例如，我们网站的请求延迟刚刚大幅增加了，有没有其他的现象同时发生？

添加监控项

搭建好了监控系统，收集上来了监控数据，我们第一件事情就是添加监控项。不得不承认，它是监控与报警系统中最难的一件事情。我们都需要注意些什么呢？

4 个黄金指标

首先，让我们谈谈监控系统的 4 个黄金指标。它们分别是：延迟、流量、错误和饱和度。如果我们只允许监控某个系统的 4 个指标，那么就应该监控这 4 个指标。

延迟，也就是服务处理某个请求所需要的时间。延迟指标区分成功请求和失败请求很有必要。例如，某个由于数据库连接丢失或者其他后端问题造成的 HTTP 500 错误可能延迟很低。在计算总体延迟时，如果将 HTTP 500 回复的延迟也计算在内的话，可能会产生误导性的结果。但是，“慢” 错误要比 “快” 错误更糟！极少量的慢错误请求就可能导致系统吞吐能力的大幅降低。因此，监控错误回复的延迟是很重要的。

流量，是系统负载的度量方式。通常我们会使用某个高层次的指标来度量，比如 IOPS、每秒交易量等。不同的业务系统的流量指标有较大差别。例如，对于普通 Web 服务器来说，该指标通常是每秒 HTTP 请求量（IOPS），同时可能按请求类型分类（静态请求与动态请求）。对于音频流媒体系统来说，这个指标可能是网络 I/O 速率，或者并发会话数量。针对键值存储系统来说，指标可能是每秒交易数量，或每秒的读取操作数量。

错误，也就是请求失败的数量。请求失败的表现很多样。最简单的当然是显式失败，例如 HTTP 回复 500 状态码。还有的请求可能是隐式失败，例如 HTTP 回复虽然是 200，但回复内容中提示出现了错误。还有一种是策略原因导致的失败。例如，如果我们要求回复在 1s 内发出，任何超过 1s 的请求就都认为是失败请求。

饱和度（Saturation），它度量的是服务容量有多 “满”。通常是系统中目前最为受限的某种资源的某个具体指标的度量。比如，在内存受限的系统中，即为内存的使用率；在 I/O 受限的系统中，即为 I/O 的使用率。要注意，很多系统在达到 100% 利用率之前性能就会严重下降，增加一个利用率目标也是非常重要的。

饱和度是最需要进行预测的指标。比如，一个典型的预测是：“看起来数据库会在 5 个小时内填满硬盘”。

在复杂系统中，饱和度可以配合其他高层次的负载度量来使用。例如，该服务是否可以正常处理两倍的流量，是否可以应对 10% 的额外流量？这些是 SRE 面临的非常现实的容量规划上的问题。

为什么我们需要做负载测试，也是为了评判服务的饱和度，究竟受何种度量指标的影响。大部分服务都习惯使用某种间接指标，例如 CPU 利用率，或者网络带宽等来评判饱和度，因为这些指标通常有一个固定的已知的上限。

延迟增加是饱和度的前导现象。所以 99% 的请求延迟（在某一个小的时间范围内，例如五分钟）可以作为一个饱和度早期预警的指标。

如果我们度量所有这 4 个黄金指标，同时在某个指标出现故障时发出报警，或者对于饱和度来说，快要发生某类故障时进行预警。只要能做到这些，服务的监控就基本差不多了。

关于长尾问题

构建监控系统时，很多人往往会采用某种量化指标的平均值。比如，延迟的平均值，节点的平均 CPU 使用率等。这些例子中，后者存在的问题是很明显的，因为 CPU 的利用率的波动可能很大。

但是其实同样的道理也适用于延迟。如果某个 Web 服务每秒处理 1000 个请求，平均请求延迟为 100ms。但是 1% 的请求可能会占用 5s 时间。如果用户依赖好几个这样的服务来渲染页面，那么某个后端请求的延迟的 99% 值很可能就会成为前端延迟的中位数。

区分平均值的 “慢” 和长尾值的 “慢” 的一个最简单办法是将请求按延迟分组计数。比如，延迟为 0～10ms 之间的请求数量有多少，30～100ms 之间，100～300ms 之间等。可以按分组制作成直方图。将直方图的边界定义为指数型增长（这个例子中倍数约为 3）是直观展现请求分布的最好方式。

采用合适的精度

应该仔细设计度量指标的精确度，这涉及到监控的成本问题。例如，每秒收集 CPU 负载信息可能会产生一些有意思的数据，但是这种高频率收集、存储、分析可能成本很高。如果我们的监控目标需要高精度数据，但是却不需要极低的延迟，可以通过采样 + 汇总的方式降低成本。例如：

将当前 CPU 利用率按秒记录。

按 5% 粒度分组，将对应的 CPU 利用率计数 +1。

将这些值每分钟汇总一次。

这种方式使我们可以观测到短暂的 CPU 热点，但是又不需要为此付出高额成本进行收集和保留高精度数据。

怎么添加监控项?

为什么我前面说，添加监控项是最难的事情？添加监控项看起来像是一个很繁琐的事务工作，但实际上它非常依赖你的架构能力。

一个很牛的监控 SRE，他要干的绝对不是不停地添加新的监控项以 “完善某个业务的监控” 。

少就是指数级的多！

就和软件开发工程师需要经常需要重构，去删减掉自己历史的无效代码一样，负责业务监控的 SRE 也需要经常重构自己的监控指标，以期用最少的监控项，来全面覆盖和掌控系统的健康状况。

我们需要遵循的监控与报警的设计哲学。记住这些哲学将有助于鼓励团队在解决问题时向正确的方向进行。

当我们打算为监控系统增加新规则时，在心中回答以下问题：

> 该规则是否能够检测到一个目前检测不到的、紧急的、有操作性的，即将发生或者已经发生的用户可见故障？
>
> 是否可以忽略这条报警？是否还有其他人也会收到这条报警？
>
> 这条报警是否确实显示了用户正在受到影响？是否存在用户没有受到影响也可以触发这条规则的情况？例如，系统维护状态下发出的报警是否应该被过滤掉。
>
> 收到报警后，是否要进行某个操作？是否需要立即执行该操作，还是可以等到第二天早上再进行？该操作是否可以被安全地自动化？
>
> 该操作的效果是长期的，还是短期的？

以上这些问题其实反映了在报警上的深层次的理念：

> 每当收到紧急状态的报警时，应该立即需要我进行某种操作。每天只能进入紧急状态几次，太多就会导致“狼来了”效应。
>
> 每个紧急状态的报警都应该是可以具体操作的。
>
> 每个紧急状态的报警的处理都应该需要某种智力分析过程。如果某个报警只是需要一个固定的机械化动作，那么它就应该被自动化。
>
> 每个紧急状态的报警都应该是关于某个新问题的，不应该彼此重叠。

接警：故障响应

接到报警我们应该怎么做?

接警后的第一哲学，是尽快消除故障。找根因不是第一位的。如果故障原因未知，我们可以尽量地保留现场，方便故障消除后进行事故的根因分析。

每一个监控项的报警应该尽可能代表一个清晰的故障场景。这会极大改善监控的有效性，直指根因，消除故障自然也就更快速。

虽然越少越好，但是不清楚故障原因的报警是难以避免的，否则我们的报警就难以完整覆盖所有的故障。比如，对于业务服务的入口级的故障，我们怎么也得报出来。每发生一次新的入口级的故障场景，我们就有必要把这个故障场景独立出来，这样下一次出现同类故障时我们就能够直接定位到根因了。

一般来说，有清晰的故障场景的监控报警，都应该有故障恢复的预案。而在那些故障原因不清晰的情况下，消除故障的最简方法是基于流量调度，它可以迅速把用户请求从故障域切走，同时保留了故障现场。

解决了线上的故障，我们就要开始做故障的根因分析，找到问题发生的源头。

这主要仰仗两种分析方法。

一种是看看同时间段下，除了我们的故障现象外，还有那些异常现象同时发生了。如果我们的监控数据足够全面，这种分析方法可以很快地定位到 “怀疑对象”。

另一种方式是分析故障请求的调用链。这方面的技术已经非常成熟。很多公司的业务实现都会把请求从前端入口到后端的整个调用过程通过一个 request id 串起来。

通过随机抽样一些故障请求的日志，然后在日志系统中搜索 request id 找到整个调用链，分析调用链找到问题的根源。

七牛云发布的日志系统 Pandora，提供了完整的监控、报警和故障的根因分析模块。除了传统的 SRE 方法论外，我们也探索基于 AI 的智能化监控与根因分析能力。

结语

监控与报警是一项非常复杂的事务，这种难度不是因为业务流程复杂导致的，而是因为与业务的高耦合导致。

监控系统需要跟随不断演变的软件一起变化。软件的全局或某个局部发生重构，负载特性和性能目标就会变化。

某个不常见的、自动化比较困难的报警，很可能随着用户增长很快变成一个经常触发、需要一个临时的脚本来应对的问题。这时，就需要有人去寻找和消除背后的根源问题，而不是在持续被故障牵着鼻子走。

解决故障的方案有可能是需要进行业务的架构调整。这也是监控与报警的复杂性所在：你需要和业务开发工程师一起配合去完善系统。

添加监控看起来像是一个很繁琐的事务工作，但实际上它非常依赖你的架构能力。

一个很牛的监控 SRE，他要干的绝对不是不停地添加新的监控项以 “完善某个业务的监控”。

少就是指数级的多！

就和软件开发工程师经常需要重构，去删减掉自己历史的无效代码一样，负责业务监控的 SRE 也需要经常重构自己的监控指标，以期用最少的监控项，来全面覆盖和掌控系统的健康状况。

如果你对今天的内容有什么思考与解读，欢迎给我留言，我们一起讨论。下一讲我们聊聊 “故障域与故障预案”。

如果你觉得有所收获，也欢迎把文章分享给你的朋友。感谢你的收听，我们下期再见。

上一篇　49 | 发布、升级与版本管理

精选留言 (4)

写留言

Aaron Cheung
2019-10-18

本篇对 devops sre 都受益良多

展开

3

leslie
2019-10-19

老师今天的讲的这个其实是生产中蛮关键的：和一些研发或架构沟通过，甚至现实中很多中小企业会无视监控；合理的监控能精准的定位问题，而不是依赖人力去排除去沟通-尤其是软件上线后。

监控不是狼也不是摆设：合理的监控确实能方便定位问题；其实现在很典型的问题是有些大厂的云服务监控确实没有体现特性吧尤其是数据库这块，集成度越高的问题定位越...

展开

张裕

2019-10-18

对于客户端的监控，老师有什么建议？在某些数据如延时的监控上是否可以和服务端监控相互印证？

作者回复: 客户端监控是指什么？


#^_^#

2019-10-18

好多干货

展开