22 | 事件管理（下）：如何保证事件的闭环处理？

2023-02-27 秦晓辉 来自北京

讲述：秦晓辉

时长 10:49　大小 9.88M

你好，我是秦晓辉。

上一讲我们介绍了事件降噪的几个典型手段，用大白话讲就是怎么让告警发得少点儿。这一讲我们介绍事件闭环管理，就是告警发出来得有人处理，所谓的闭环，就是指告警发出、认领、协作处理、问题恢复、复盘改进的整个过程。

虽然事件降噪的几个手段落实之后，事件数量确实变少了，但是处理告警事件显然不是一个让人愉快的事情，不愉快的事情就要团队共担，所以第一个手段就是排班，专人做专事。

排班，专人做专事

这个手段听起来并不高大上，但确实非常有效。值班期间虽然提心吊胆的，生怕背锅，但因为是轮班制，心里总有个盼头，挺过这个周期就好了。

轮班的人在值班期间是第一责任人，会拿出 120% 的精力来处理问题，责任到人显然更容易推进问题解决，其他不值班的人则可以心无旁骛地做一些长线的事情，不至于总是被告警打断。

排班系统通常不开源，通常是作为事件中心的一个功能，PagerDuty 就提供了排班能力，即使没有系统支持，也建议人为制定一个排班表，把这个制度落实下去，对告警闭环处理也会有很大帮助。

值班人员在值班期间，虽然已经高度重视了，但也难免疏漏，这就需要告警升级机制了。

告警升级机制

告警升级是指在第一责任人收到告警之后没有及时响应，然后系统自动通知二线、三线人员的一种机制。一线人员没有及时响应的原因可能有很多，比如手机静音了没有听到，晚上睡着了，或者临时出去有事忘带手机了等等。这个时候系统发现某个告警一直没有恢复，也没有被认领，一段时间之后，就应该通知值班人员的领导或者二线备份人员，如果二线人员也迟迟没有响应，就应该继续往上升级。

告警升级机制需要认领功能的配合，也就是一线人员收到告警之后要通过某种机制告诉系统：“我已知晓告警，现在我开始处理了，你不要升级了”。典型的认领功能一般是做在页面上的，告警后打开告警事件管理中心，选中相关告警一键认领，也可以通过上行短信或即时通讯工具中的上行回调机制来完成。

升级机制会给值班人员很大的压力，毕竟谁也不想稍不留神就把电话打到老板那里，所以一般只有严重的告警才会启用升级机制，警告或者通知性质的告警都不用启用升级机制。当然，这个规范怎么定，各个团队可以自行商定。

通过排班、认领、升级这些机制，可以确保告警递达指定的人，但要处理告警的话，只有值班人员自己就未必搞得定了，需要有协同机制把相关人都拉进来一起处理才可以。对于某个故障，可能同时有多个告警事件产生，大家基于一个统一的故障协同，而不是基于一堆事件分别协同，这就需要把这多个事件收敛成一个故障，下面我们来聊一下这个收敛逻辑。

告警收敛逻辑

一般收敛逻辑是三级收敛，event -> alert -> incident。举个例子，最原始的告警事件，比如 host1 在 timestamp1 产生了一条 cpu_usage_idle 的告警，我们称为一个 event。如果没有恢复，一段时间之后，比如 timestamp1 + 60min，一般会再发出一个告警，还是 host1，还是

cpu_usage_idle 这个指标。很明显，这两个告警事件是有关联关系的，指代的是一个问题，只是时间戳不同，这样的两个 event，就可以收敛为一个 alert。

从实现上来说，告警策略（也称告警规则）+ 指标标签集的哈希值，可以作为 alert 的唯一标识。比如刚才的例子，告警策略的 ID 假设为 32，标签集是：[“**name**=cpu_usage_idle”, “host=host1”]，这两个时间戳产生的告警事件，哈希值都是一样的。

计算方法是：

```
hash(32 + ["__name__=cpu_usage_idle", "host=host1"])
```

从 event 到 alert 的这个收敛逻辑，我们叫做一级收敛。只有这个收敛逻辑还不够，告警信息还是比较散，不好基于这些散乱的告警分别做协同，把多个 alert 收敛成一个 incident（故障），基于 incident 做协同才比较方便。但是，event 到 alert 是有一个固定的收敛逻辑的，可以通过程序自动收敛，而 alert 到 incident 却很难自动收敛。不过业界也会有一些常见的做法，下面我举几个例子。

1. 根据时间做收敛

把告警中心收到的所有告警，按照时间维度做收敛，比如按照分钟颗粒度，一分钟内所有告警收敛成一个故障，下一分钟所有告警收敛成另一个故障。显然，一个故障内的多个告警相互之间可能没有关联关系，所以这种收敛方法不是太好。

2. 根据时间 + 标签做收敛

除了时间维度，再加上某个标签作为收敛维度，比如机器标签，某个时间段内所有 A 机器的告警收敛成一个故障，所有 B 机器的告警收敛成另一个故障。或者按照服务维度，某个时间段内所有 A 服务的告警收敛成一个故障，所有 B 服务的告警收敛成另一个故障。看起来效果好多了，不过还是没办法和现实中的告警和故障建立完美的对应关系。

3. 根据时间 + 文本相似度做收敛

文本相似度需要引入算法，但是算法总得有个规律，我们很想把某个故障相关的告警聚拢到一起，但是显然，很难有个行之有效的规律，没有规律的算法效果自然好不到哪儿去。

既然没办法把告警自动收敛成故障，那就手工来做。一个故障关联的关键告警，还是相对容易区分的，只要把关键告警关联到故障，后续基于这个故障做协同就可以了。所谓协同，一个是信息同步、协同处理，一个是共同复盘、管理跟进项。

故障协同处理

首先，并不是所有的告警都需要升级成故障协同处理。一般来讲，如果告警可以被值班人员直接处理掉，对别的团队负责的服务没有影响，不需要通知别的团队，通常是不需要升级成故障的，在告警层面来协同就可以了，自己团队内部消化掉；如果值班人员和他所在的团队没办法独自处理告警，才需要升级成故障，拉其他团队的人进来一起处理。

多个团队共同处理一个故障，不同团队的人会发现一些不同的线索，需要及时同步给所有相关的人，这个时候就可以在故障下面添加评论，其他人就可以及时看到。等到止损之后，大家还要根据故障时间线复盘，产出一系列跟进项，这个时候就需要这个故障管理模块具备跟进项管理的功能，或者至少能够跟任务管理系统良好打通。

有了这样一个故障协同的机制之后，故障被处理掉的概率就大幅提升了，后续再配合一些运营统计手段，统计各个团队的平均故障止损时间，建立红黑榜，大家就会有更高的热情来处理故障。当然，人的热情再高，也不如机器来得快，如果有些告警能够直接关联自动化处理逻辑，无疑可以大大增加事件闭环率。

告警自动处理

很多监控系统都可以配置 Webhook，当告警触发之后自动回调某个 HTTP 接口，来串联一些自动化的逻辑，让告警事件无人值守自动处理。比如某个机房的某个服务挂掉了，Webhook 的逻辑是自动调用切流的接口，把服务流量切走，这样来达到止损的目的。

不过对于很多运维工程师来说，写一个 HTTP 接口还是有点儿麻烦，如果可以在告警之后直接调用一个脚本，在这个脚本里写自愈逻辑就好了。Nightingale 就可以这么干，配合 ibex 模块，可以在告警的时候自动去告警的机器上跑个脚本，比如磁盘满了自动跑个脚本清理一下无用的日志，只要在告警规则的回调地址里配置类似 `${ibex}/35` 的信息就可以了。

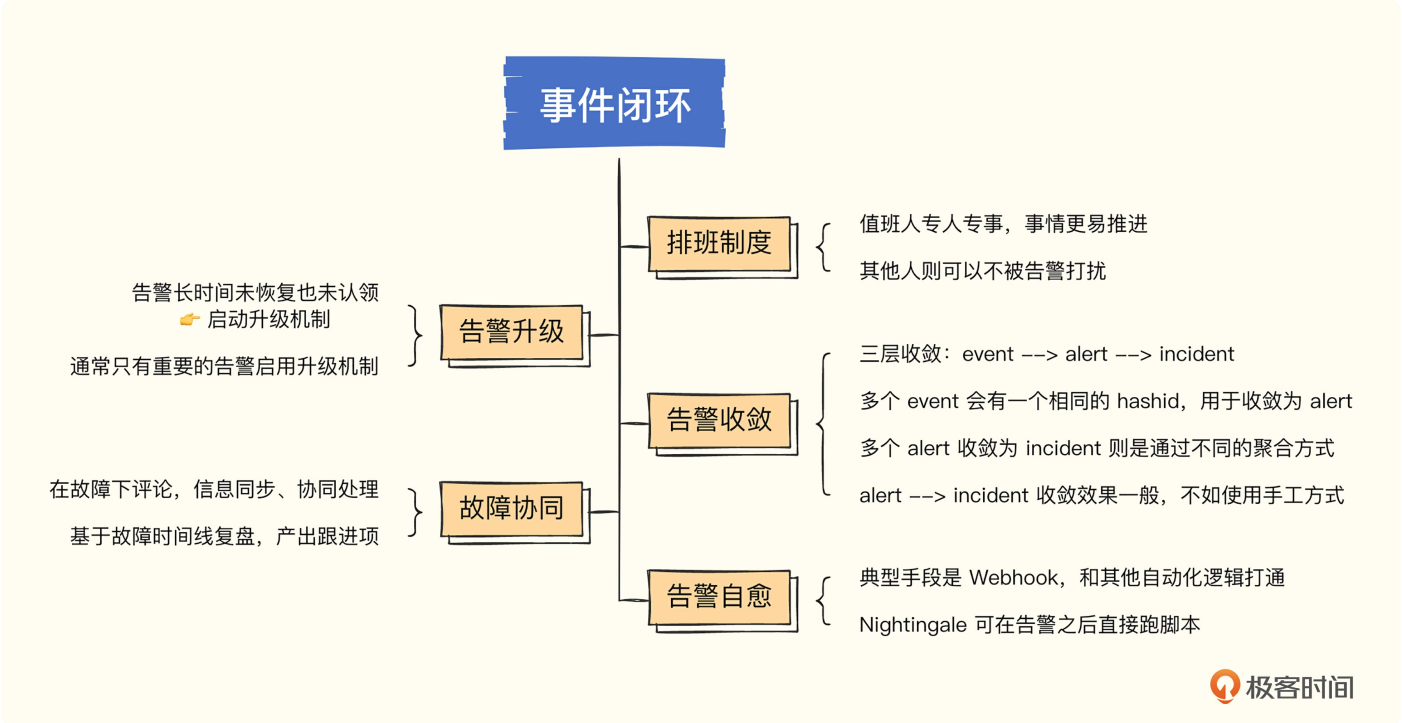
`${ibex}/35` 就表示告警触发之后自动执行 ID 为 35 的自愈脚本，默认在告警的机器上执行。如果想到固定的机器上执行，比如去某个中控机上执行，只需要把机器标识信息拼接到 URL 后面，比如 `${ibex}/35/cloud-center01.bj`。

新版本的 Categraf 已经内置了 ibex-agent，不用再单独安装任务执行的 agent 了，只要在服务端单独部署 ibex-server 模块，和 n9e-server 协同打通即可，非常简单。

另外，告警自动处理的这段逻辑，未必一定能够做到告警自愈，有的时候只是使用这个机制来抓现场，也是非常有价值的。比如某个进程挂掉了，在挂掉的时候我想知道当时机器的一些运行情况，比如各项资源的占用情况、系统日志的信息等等，我们就可以借助告警自动处理的这个方式，来自动跑个脚本抓取当时机器上的一些现场信息，相比收到告警之后手工登录机器查看要高效得多。

提高事件处理闭环率的方法，基本就是上面这些，下面我们对这一讲的内容做一个小结。

小结



这一讲我们主要想解决一个问题，就是告警事件发出之后，怎么保证一定有人跟进处理，而且是很快地跟进。首先是通过排班这个偏管理的手段让大家共同承担一个不那么让人愉快的事情，为了保证值班人员能及时响应，引入了**告警升级通知机制**。确保事件有人认领之后，就相当于确立了事件的跟进负责人，包产到户了，那后面的事情推进起来就快得多了。

告警事件的跟进负责人如果可以处理消化掉相关的告警，那是再好不过。如果他个人和团队没办法独自处理，就需要拉更多的人进来了。大家基于一个统一的故障协同机制一起排查，故障是一个包含了多条告警的实体，大家在这个故障下通过评论的方式发表看法，提供线索，止损之后还要复盘，产出故障对应的未来改进项。

此外，还有一个有效的策略——**告警自动处理机制**，一般通过 Webhook 来实现，Nightingale 则更进一步，内置告警自愈脚本的管理，可以和 ibex 打通做到告警的时候自动去机器上跑个脚本。

图片来自https://flashcat.cloud/product/flashcat-duty/

互动时刻

对于告警自愈这个功能，听起来很酷，但是具体有哪些场景可以应用告警自愈这个机制来自动化处理非常值得梳理，欢迎你在留言区分享你想到的场景，我们集思广益，也欢迎你把今天的内容分享给你身边的朋友，邀他一起学习。我们下一讲再见！

上一篇 21丨事件管理（上）：事件降噪的几个典型手段

下一篇 结束语丨弱水三千，只取一瓢饮

精选留言 (3)

写留言

晴空万里

2023-03-02 来自广东

但是我没有高屋建瓴分析汇聚脚本的能力 只能见到啥就是啥？

晴空万里

2023-03-02 来自广东

我们是研发工程部门 会负责整个公有云机器业务运维 告警自愈脚本确实需要梳理 例如 执行数据库SQL 机器卡住了 使用脚本删除该进程 执行一个定时任务 失败了 然后自动重试

peter

2023-02-27 来自北京

请教老师几个问题：

Q1：Prometheus支持webhook吗？

Q2：实际的告警处理过程中，是否容易造成冲突？比如运维内部人员之间相互抱怨，运维和开发人员之间相互指责等。

Q3：老师的公司为什么可以没有运维？

上一课请教老师问题，老师说自己的公司没有运维，为什么不需要啊？是因为不是互联网公司而是单纯提供方案吗？ 我问这个问题主要是想有个参考：比如我创建一个网站，注册用户五十万，这种情况是否需要运维人员？

Q4：能否以加餐形式讲一下移动端监控，安卓或iOS。

作者回复: 1，Prometheus生态的Webhook一般放在alertmanager里

2，我经历的公司，遇到故障的时候，大家首先想到的是如何快速止损，而非指责，如果一上来就想着指责的，要么是管理问题，要么是人员本身的职场素养太差了

3，我们是ToB公司，提供商业化监控和故障定位的解决方案，我们有交付人员负责落地产品，和ToC的公司是不同的。是否需要运维人员，不能简单的根据注册用户数量来，通常来讲，研发搞不定下载的稳定性、成本问题的时候，研发团队很大、服务模块很多的时候，通常才需要运维，当然我这也只是举例，实际考量的因素很多

4，没有计划

4，

共 2 条评论 >