14 | 高可用架构案例（二）：如何第一时间知道系统哪里有问题?

2020-03-23 王庆友

讲述：王庆友
时长 17:35 大小 16.11M

你好，我是王庆友。

在前面两讲中，我与你介绍了系统的高可用都有哪些设计原则和具体手段。其中我也特别提到，**要想保证系统的高可用，我们还需要对系统进行全面有效的监控。**

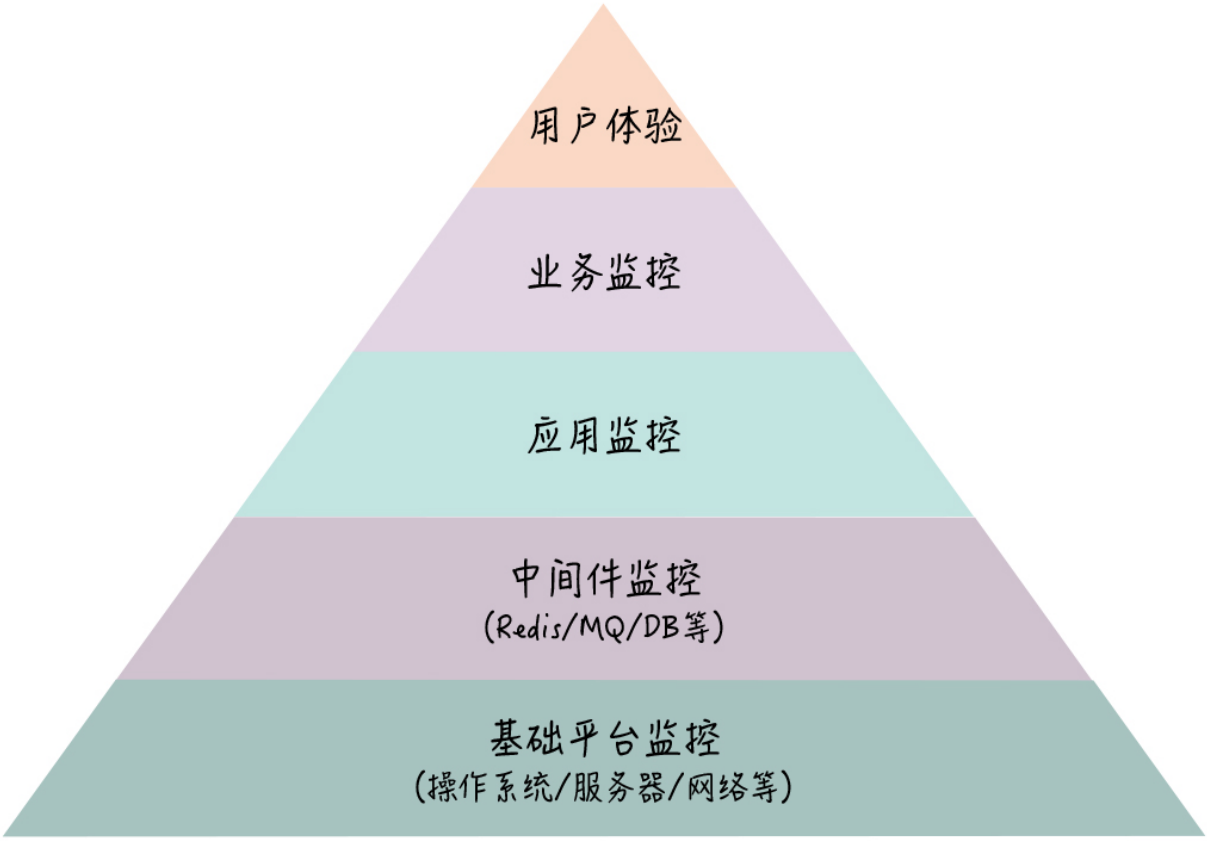
监控是系统的眼睛，无监控，不运维。今天我们就从监控的角度来聊聊如何保证系统的高可用。

在开发软件时，我们经常强调一个业务功能的可测性，甚至有一种说法是测试驱动开发。在开发之前，我们会先设计测试用例，再去考虑如何实现功能。同样，当我们对系统作了很多加固，也是希望能保证它的稳定可用。

但我们怎么判断系统的各个节点当前是否正常呢？这个就对应了节点的可监控性，如果你事先想好了系统应该如何监控，如何判断每个节点是否正常，那你就会更清楚应该采取什么样的措施。很多时候，我们可以从监控的角度来倒推系统的可用性设计。

监控的分类

在第 11 讲中，我和你介绍了系统的组成，它包括接入层、应用系统、中间件、基础设施这几个部分，那我们的监控也是针对这些部分来实施的。一般来说，监控可以分为 5 个层次，如下图所示：



从上到下，分别为用户体验监控、业务监控、应用监控、中间件监控、基础平台监控。

1. **用户体验监控：**指的是从前端用户的访问速度出发，来监测系统的可用性，包括页面能否打开、关键接口的响应时间等等，用户体验监控一般结合前端的埋点来实现。
2. **业务监控：**它是从业务结果的角度来看，比如说订单数、交易金额等等，业务监控也是最直观的，我们知道，如果业务数据没问题，系统整体也就没有问题。对于业务监控，我们一般是从数据库里定时拉取业务数据，然后以曲线的方式展示业务指标随着时间的变化过程。除了当前的曲线，一般还有同比和环比曲线。同比是和前一天的数据进行比

较，环比是和一周前的数据进行比较，两方面结合起来，我们就能知道当前的业务指标有没有问题。

3. **应用监控：**指的是对自己开发的代码进行监控，比如接口在一段时间内的调用次数、响应时间、出错次数等等。更深入一点的应用监控还包含了调用链监控，我们知道，一个外部请求的处理过程包含了很多环节，比如说网关、应用、服务、缓存和数据库，我们可以通过调用链监控把这些环节串起来，当系统有问题时，我们可以一步步地排查。有很多 APM 工具可以实现调用链监控，如 CAT、SkyWalking 等等。
4. **中间件监控：**指的是对标准中间件进行监控，它是第三方开发的代码，比如数据库、缓存、Tomcat 等等，这些组件对应的是系统的 PaaS 层。这些中间件往往带有配套的监控系统，比如，RabbitMQ 就有自带的监控后台。
5. **基础平台监控：**指的是对系统底层资源进行监控，如操作系统、硬件设备等等，这个层次的监控对应的是系统的 IaaS 层。Zabbix 就是典型的基础设施监控工具，它可以监控 CPU、内存和磁盘的使用情况。

监控的痛点

我们知道，一个大型的互联网平台，背后对应的是大规模的分布式系统，有大量的软硬件节点一起协作，这里的任何节点都有可能出问题，所以我们需要通过监控，及时发现和解决问题，提升系统的可用性。

但想要实现高效的监控，这不是一件容易的事情。下面，我给你举一个线上事故处理的例子，你就能理解监控面临的挑战。

首先，Monitor 发现订单曲线突然跌停，当前的订单数量变为 0，于是，Monitor 快速拉起电话会议，或者在微信群里 @所有人进行排查。这时候，一大堆相关的或不相关的人，都开始排查自己负责的那部分系统，比如说，运维在 Zabbix 里检查网络和机器，开发在 ELK 系统（Elasticsearch+Logstash+Kibana）里检查错误日志，DBA 检查数据库。

过了一会儿，负责 App 服务端的开发人员，在 ELK 里发现有大量的调用下单服务超时，于是他去询问下单服务的开发人员这是怎么回事。下单服务的开发人员就去检索错误日志，结果发现调用会员服务有大量的超时情况，然后他就去问会员服务的开发人员这是怎么回事。会员服务的开发人员通过错误日志，发现会员数据库连接不上，于是他把问题反映给 DBA。DBA 先拉上负责网络的同事一起看，发现网络没啥问题，然后他再去检查会员数据库本身，这时，他发现有慢查询把 DB 给挂住了。

这样，通过一系列的接力式排查，问题终于找到了，最后 DBA 把慢查询杀掉，所有人都去检查自己的系统，发现没有新的错误情况，系统恢复了正常。而这个时候，距离问题的发生已经过去了很长时间，在这个期间，技术被老板催，老板被商户催，而商户也已经被用户投诉了 N 次。

以上的事故处理过程还算比较顺利的，毕竟我们通过顺藤摸瓜，最后找到并解决了问题。**更多的时候，我们面对事故，就像是热锅上的蚂蚁，众说纷纭，谁也不能肯定问题出在哪里。结果呢，我们病急乱投医，胡乱干预系统，不但没能解决问题，而且往往引发了二次事故。**

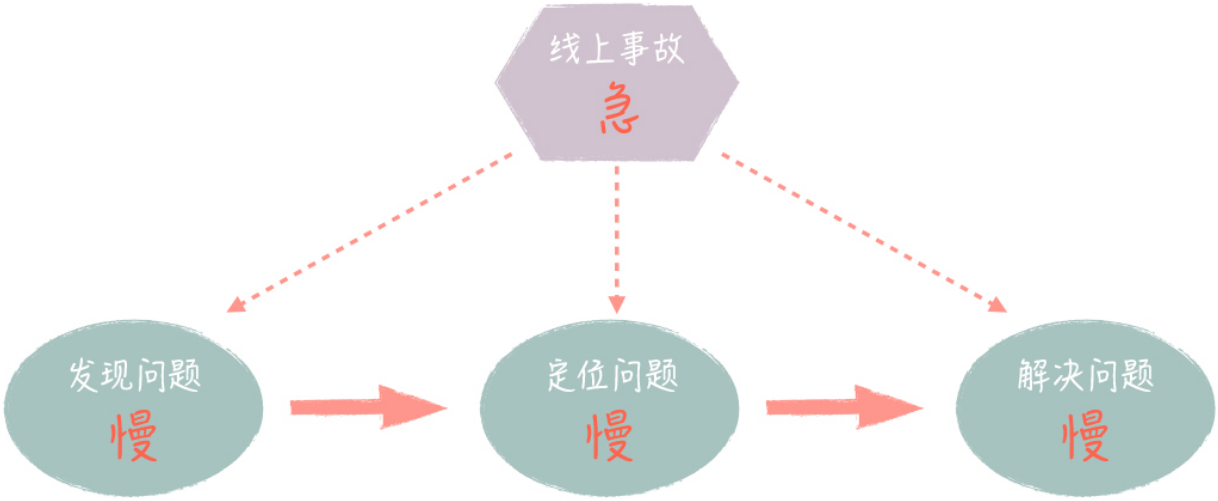
你可以发现，在这个例子中，虽然我们有应用日志监控，有 Zabbix 系统监控，有网络和数据库监控，但对于一个大规模的分布式系统来说，这种分散的监控方式在实践中有一系列的弊端。

首先，不同的节点，它的监控的方式是不一样的，相应地，监控的结果也在不同的系统里输出。

同时，系统不同部分的监控都是由不同的人负责的，比如说，运维负责的是基础平台监控，开发负责的是应用系统监控。而监控信息往往专门的人才能解读，比如应用监控，它需要对应的开发人员才能判断当前的接口访问是否有问题。

最后，系统作为一个整体，需要上下游各个环节的人一起协作，进行大量的沟通，才能最终找到问题。

你可以看到，这种监控方式是碎片化的，对于处理线上紧急事故，它无疑是低效的，这里有很多问题。



1. **发现问题慢：**业务监控的曲线一般 1 分钟更新一次，有时候因为正常的业务抖动，Monitor 还需要把这种情况排除掉。因此，他会倾向于多观察几分钟，这样就导致问题的确认有很大的滞后性。
2. **定位问题慢：**系统节点多，大量的人需要介入排查，而且由于节点依赖复杂，需要反复沟通才能把信息串起来，因此很多时候，这种排查方式是串行或者说无序的。一方面，无关的人会卷入进来，造成人员的浪费；另一方面排查效率低，定位问题的时间长。
3. **解决问题慢：**当定位到问题，对系统进行调整后，验证问题是否已经得到解决，也不是一件很直观的事情，需要各个研发到相应的监控系统里去进行观察，通过滞后的业务曲线观察业务是否恢复。

那么，我们怎么解决监控面临的这些困境，以高效的方式解决线上事故，保障系统的高可用呢？

解决思路

你可以看到，前面这种监控方式，它是碎片化和人工化的，它由不同的工具负责监控系统的不同部分，并且需要大量专业的人介入，并通过反复的沟通，才能把相关的信息拼接起来，最后定位到问题。

那我们能不能把系统所有的监控信息自动关联起来，并且以一种直观的方式展示，让所有人一看就明白是哪里出了问题，以及出问题的原因是什么呢？

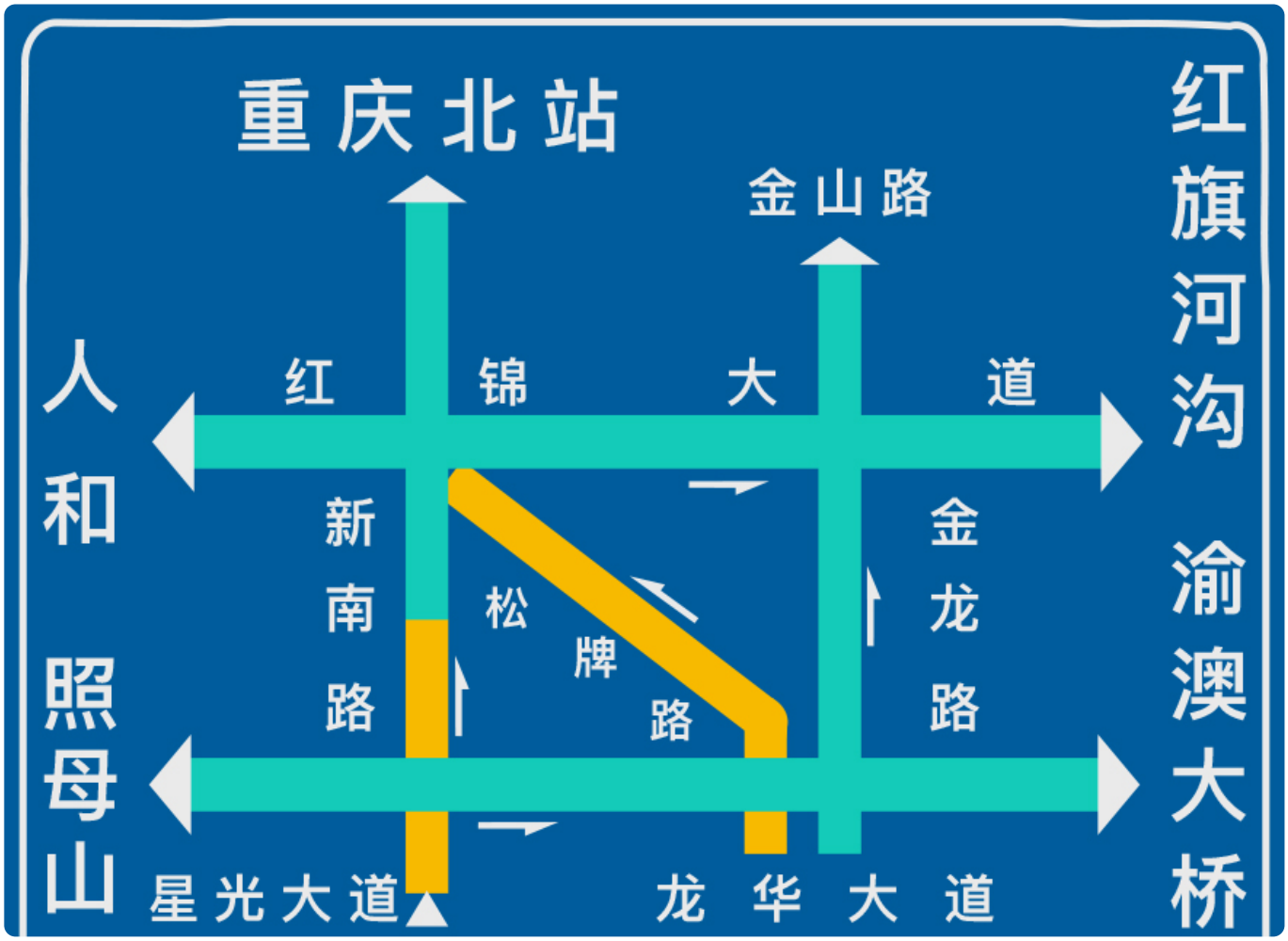
从这个思路出发，对系统的监控，我们需要做到两点：

1. 系统能够自动地判断每个节点是否正常，并直观地给出结果，不需要经过专业人员的分析。
2. 系统能够自动把各个节点的监控信息有机地串起来，从整体的角度对系统进行监控，不需要很多人反复地进行沟通。

这里，我们可以借鉴一下道路交通监控的例子。

我们经常可以在市内的高架上看到交通拥堵示意图。在下面的这张交通信息图上，你可以看到，每条道路都通过上下左右不同的方位，有机地关联在一起，形成一个整体的交通网络；

同时，在交通图上，通过红黄绿三种状态，实时地反映了每条道路的拥堵情况。这样，司机就可以非常直观地了解道路是否畅通，从而提前避开拥堵路段。



这里有几个关键词：实时、直观、整体。下面，我们就来对照下软件系统的监控，来看看要想实现类似的监控效果，我们应该怎么做。

首先要**实时**，我们需要第一时间知道系统当前是否有问题。

然后要**直观**，节点是否有问题，我们需要很直观地就能判断出来，就像交通图上的红黄绿颜色标识一样。我们知道，在发生紧急事故时，人脑很可能会处于错乱状态，这个时候，我们一定不能指望专业的头脑或者严密的分析来判断问题，这样不但慢，而且很容易出错。所以，系统哪些部分有问题，问题是否严重，以及出问题的大致原因是什么，这些信息，监控系统都必须能够直观地给出来。

最后是**整体**，我们需要针对系统做整体监控，就像交通图一样，它是针对周边整体的道路情况进行展示，我们也需要把系统的各个节点放在一起，清晰地给出节点依赖关系。系统真正

出问题的地方往往只有一个，其他地方都是连带的，如果监控系统能够给出节点的上下游依赖关系，对于定位真正的问题源是非常有用的。

所以，对照道路交通监控的思路，我们可以采取这样的监控方式：

首先，系统中的每个节点对应交通图的一条道路；

然后，节点的健康状况对应道路的拥堵情况，节点同样也有红黄绿三种不同的颜色，来展示该节点是否正常；

最后，节点之间的调用关系对应道路的方位关系。

这样我们就能构建一个实时的、直观的、一体化的监控系统，类似交通图一样，可以一眼就看出系统的问题所在。

好，回到刚才事故处理的例子，如果我们的监控系统按照这种方式来设计，它的监控效果会是什么样的呢？

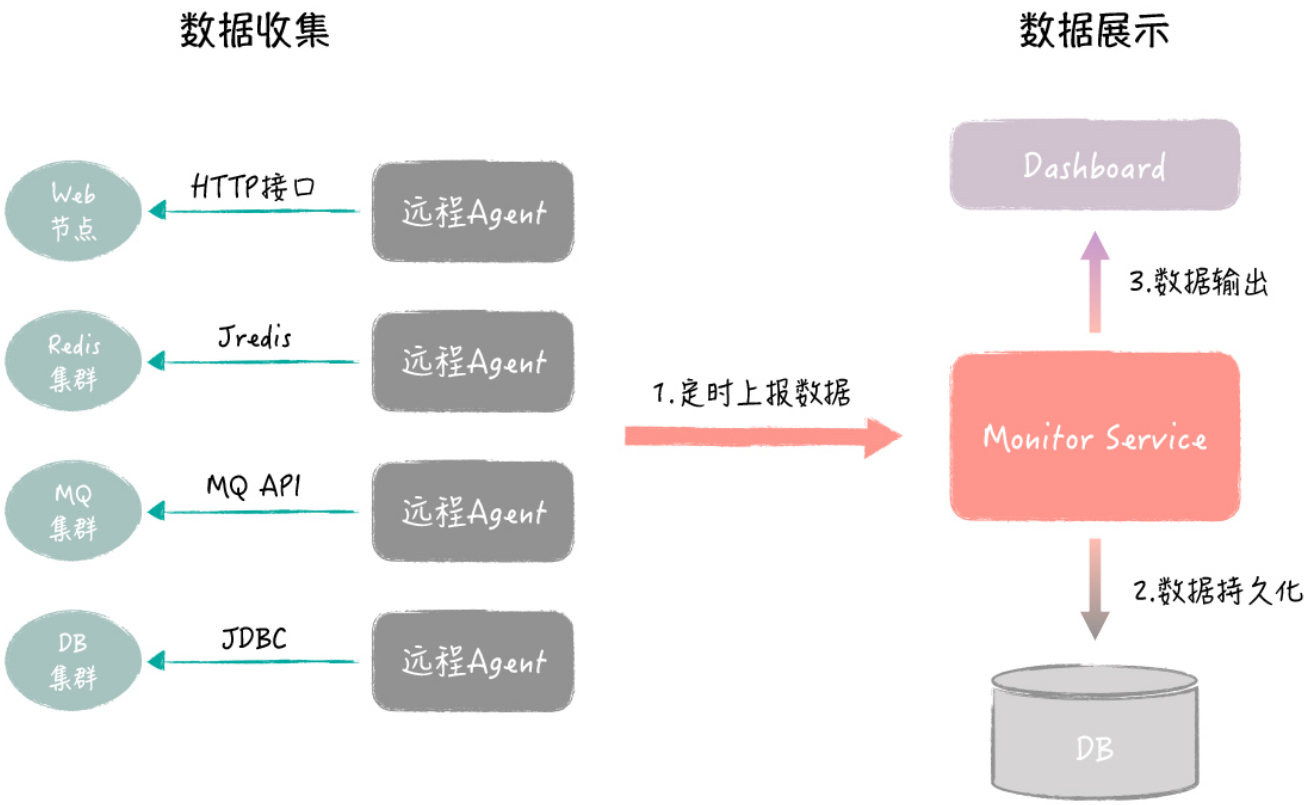
首先所有的节点，包括服务端应用、下单服务、会员服务还有其他服务，以及它们各自用到的缓存、消息队列和数据库，这些节点的健康状态我们在一个页面里就可以看到，包括它们的依赖关系。

如果会员数据库出了问题，我们根据依赖关系倒推，会员数据库 -> 会员服务 -> 下单服务 -> 服务端应用这 4 个节点都会爆红，而其他节点不受影响，保持绿色。服务端应用和下单服务节点会有错误消息提示接口调用超时，而会员服务和会员数据库节点的错误消息提示的是数据库连接超时。

这样其他绿色的节点，我们就不用排查了，然后我们观察爆红的节点，通过上下游依赖关系，就知道最终的问题很可能出在会员数据库上，DBA 重点检查会员数据库就可以了。当数据库问题解决以后，我们可以看到所有爆红的节点马上变绿，立即就能确认系统恢复了正常。

架构方案和效果

根据前面的思路，我们设计了监控系统的整体架构，如下图所示：



每个被监控的节点，均有对应的 Agent 负责采集健康数据，不同的节点类型，数据采集的方式也不一样：Web 节点通过 HTTP 接口调用，Redis 通过 Jredis，MQ 也通过对应的 API 接口，DB 则采用 JDBC。

Agent 每隔 3s 采集节点数据，然后上报数据给 Monitor Service；Monitor Service 负责确定节点当前的状态并保存到数据库，这样就完成了节点健康状态的检测；最后，前端 Dashboard 每隔 3s，拉取所有节点的状态，以红黄绿三种颜色在同一页面展示，同时还会显示具体的出错信息。

那我们是根据什么规则来判断节点的健康状态呢?

这里，我以 DB 为例简单说明一下。Agent 每隔 3 秒会去尝试连接数据库，并进行简单的表读写操作，如果连接和读写都能够成功，那就说明该 DB 当前的运行是正常的，相应的，在 Dashboard 里面，这个 DB 节点会显示为绿色。

Redis 和 MQ 类似，我们主要也是检测组件的可用性；Web 应用的健康规则会相对复杂一些，我们会结合 Web 应用接口的功能和性能来做综合判断。这个监控系统的设计，我还会在下一讲里具体介绍，你到时候可以深入理解其中的细节。

我们最后来看下监控的效果。

下图是某个业务系统的实际监控效果图，左边是系统的部署架构，最上面是两个 Web 应用，这两个应用分别有自己的 Web 服务器、MQ 和 Redis 节点。

> 提示：这里，我对细节做了模糊化处理，不过没关系，我主要的目的是让你能了解监控的效果，尽管图片模糊，但它不会影响你理解内容。

以左上角的应用为例，它的 Web 应用部署在 Docker 里面，所以这里只显示一个节点（虚拟机部署可以看到每个实例的 IP，但 Docker 容器无法看到，对外表现为一个地址）；对于 Redis，我们是购买公有云的服务，所以也是一个实例；但 MQ 是集群的方式，它有三个实例。

然后，这两个 Web 应用同时依赖后端的 3 个基础服务，这 3 个服务是并列的关系，每个服务又分别有自己的应用、MQ 和 Redis。所以，你可以看到，在这个监控页面里，节点的部署情况和依赖关系都是一目了然的。

在这个例子中，有一个节点显示为黄色，黄色说明它有问题，但并不严重。你可以在右边的异常消息列表里看到具体的原因（在最近 3s 内，这个 Web 应用的接口响应时间超过了正常值的 5 倍），每条异常消息包括了出错的节点、具体出错的接口、该接口的正常响应时

间，以及当前的响应时间。这样，你就可以很方便地把左边的出错节点和右边的异常消息对应起来，知道哪些节点有错误，还有出错的原因是什么。

另外，如果你在左边的图里点击某个节点，会弹出新页面，显示该节点的历史出错信息，并且新页面里有链接可以直接跳到 Zabbix、CAT 和 ELK 系统，这样你可以在这些专门的系统里做进一步的排查。

所以说，这里的监控系统提供的是整体的监控信息，可以帮助你快速定位问题的根源，在很多情况下，我们通过这里给出的错误信息，就可以知道出错的原因。当然，如果碰到特别复杂的情况，你还是可以在这里快速关联到各个专业的监控系统去收集更深入的信息。

总结

今天，我与你介绍了一下监控的分类，你现在应该对监控有了比较深入的了解，知道一个完整的监控体系都包含了哪些内容。

此外，我也结合线上事故处理的例子，和你说明了碎片化的监控带来的一些问题，并给出了整体化的解决思路以及具体的落地方案。在实践中，这套监控系统也确实发挥了巨大的价值，让我们可以高效地应对线上事故，提升系统的可用性，希望你能够深入地领悟和掌握。

在下一讲中，我还会和你介绍这个方案的实现细节，这样，你也可以尝试着去落地类似的监控系统。

最后，给你留一道思考题： 你的公司都有哪些监控手段，当处理线上事故时，你遇到的最大的挑战是什么？

欢迎你在留言区与大家分享你的答案，如果你在学习和实践的过程中，有什么问题或者思考，也欢迎给我留言，我们一起讨论。感谢阅读，我们下期再见。

上一篇　13 | 高可用架构案例（一）：如何实现O2O平台日订单500万?

精选留言 (5)

写留言

每天晒白牙
2020-03-23

服务器节点的监控由公司运维平台监控，当cpu，内存使用过高都会发报警短信，带上ip信息，收到后可以登录服务器去排查，但业务报警主要通过 Prometheus 埋点实现的，有的时候流量抖动也会触发报警，因为业务流程比较长，业务相对复杂些，加上节点多，日志少，在高度紧张的情况下，排查问题还是挺难的

...

展开

3

滕建兴
2020-03-24

老师问一下，这个是针对某个业务场景的整体化监控，但是一个系统很多场景，那不是要绘制很多这样的监控图

展开

作者回复: 前端一套代码就可以，下一讲会具体介绍

GEEKBANG_6638780
2020-03-23

你好，老师，问个问题。jdbc监控那块，是不是通过简单的表查询作用不是很大。就拿文中的列子来说，是因为慢查询导致的问题。假如我的数据库连接还没被慢查询占满，这个时候jdbc监控就不会有问题。如果想实现，慢查询监控，是不是要对mysql本身做更细致化的监控了，而不是等连接被打满以后才去解决问题。

展开

tt
2020-03-23

我们的监控难点在于，有很多服务是面向强势对公客户的，经常需要满足它们提出的所有要求，做好服务，不论是谁的问题，都是我们来解决。

展开

Jeff.Smile
2020-03-23

之前公司做的比较简单，无非是针对重点业务做email或者短信告警，系统层面用zabbix但一般是事后观察。