首先，我们需要一个全栈系统监控的东西。它就像是我们的眼睛，没有它，我们就不知道系统到底发生了什么，我们将无法管理或是运维整个分布式系统。所以，这个系统是非常非常关键的。

而在分布式或Cloud Native的情况下，系统分成多层，服务各种关联，需要监控的东西特别多。没有一个好的监控系统，我们将无法进行自动化运维和资源调度。

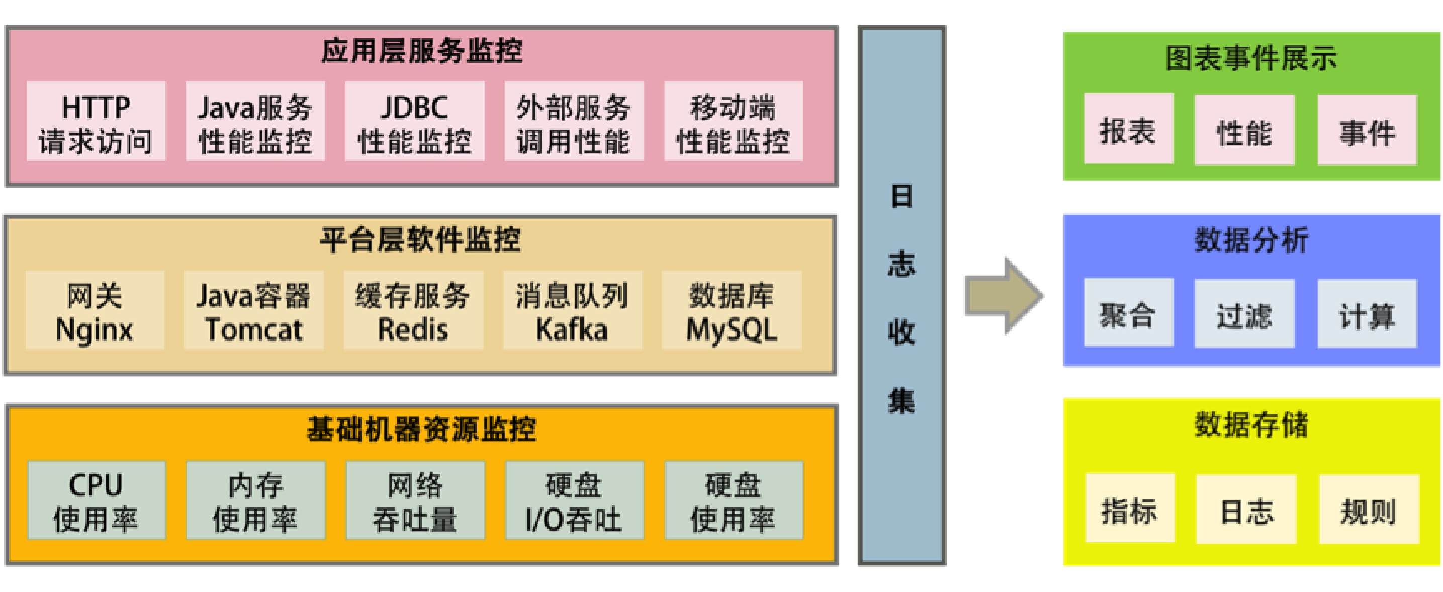
这个监控系统需要完成的功能为：

- 全栈监控；
- 关联分析；
- 跨系统调用的串联；
- 实时报警和自动处置；
- 系统性能分析。

多层体系的监控

所谓全栈监控，其实就是三层监控。

- **基础层**：监控主机和底层资源。比如：CPU、内存、网络吞吐、硬盘I/O、硬盘使用等。
- **中间层**：就是中间件层的监控。比如：Nginx、Redis、ActiveMQ、Kafka、MySQL、Tomcat等。
- **应用层**：监控应用层的使用。比如：HTTP访问的吞吐量、响应时间、返回码，调用链路分析，性能瓶颈，还包括用户端的监控。



这还需要一些监控的标准化。

- 日志数据结构化；
- 监控数据格式标准化；
- 统一的监控平台；
- 统一的日志分析。

什么才是好的监控系统

这里还要多说一句，现在我们的很多监控系统都做得很不好，它们主要有两个很大的问题。

1. **监控数据是隔离开来的**。因为公司分工的问题，开发、应用运维、系统运维，各管各的，所以很多公司的监控系统也是各是各的，完全串不起来。
2. **监控的数据项太多**。有些公司的运维团队把监控的数据项多做为一个亮点到处讲，比如监控指标达到5万多个。老实说，这太丢人了。因为信息太多等于没有信息，抓不住重点的监控才会做成这个样子，完全就是使蛮力的做法。

一个好的监控系统应该有以下几个特征。

- **关注于整体应用的SLA**。主要从为用户服务的API来监控整个系统。
- **关联指标聚合**。 把有关联的系统及其指标聚合展示。主要是三层系统数据：基础层、平台中间件层和应用层。其中，最重要的是把服务和相关的中间件以及主机关联在一起，服务有可能运行在Docker中，也有可能运行在微服务平台上的多个JVM中，也有可能运行在Tomcat中。总之，无论运行在哪里，我们都需要把服务的具体实例和主机关联在一起，否则，对于一个分布式系统来说，定位问题犹如大海捞针。
- **快速故障定位**。 对于现有的系统来说，故障总是会发生的，而且还会频繁发生。故障发生不可怕，可怕的是故障的恢复时间过长。所以，快速地定位故障就相当关键。快速定位问题需要对整个分布式系统做一个用户请求跟踪的trace监控，我们需要监控到所有的请求在分布式系统中的调用链，这个事最好是做成没有侵入性的。

换句话说，一个好的监控系统主要是为以下两个场景所设计的。

“体检”

- **容量管理**。 提供一个全局的系统运行时数据的展示，可以让工程师团队知道是否需要增加机器或者其它资源。
- 性能管理。可以通过查看大盘，找到系统瓶颈，并有针对性地优化系统和相应代码。

“急诊”

- **定位问题**。可以快速地暴露并找到问题的发生点，帮助技术人员诊断问题。
- **性能分析**。当出现非预期的流量提升时，可以快速地找到系统的瓶颈，并可以帮助开发人员深入代码。

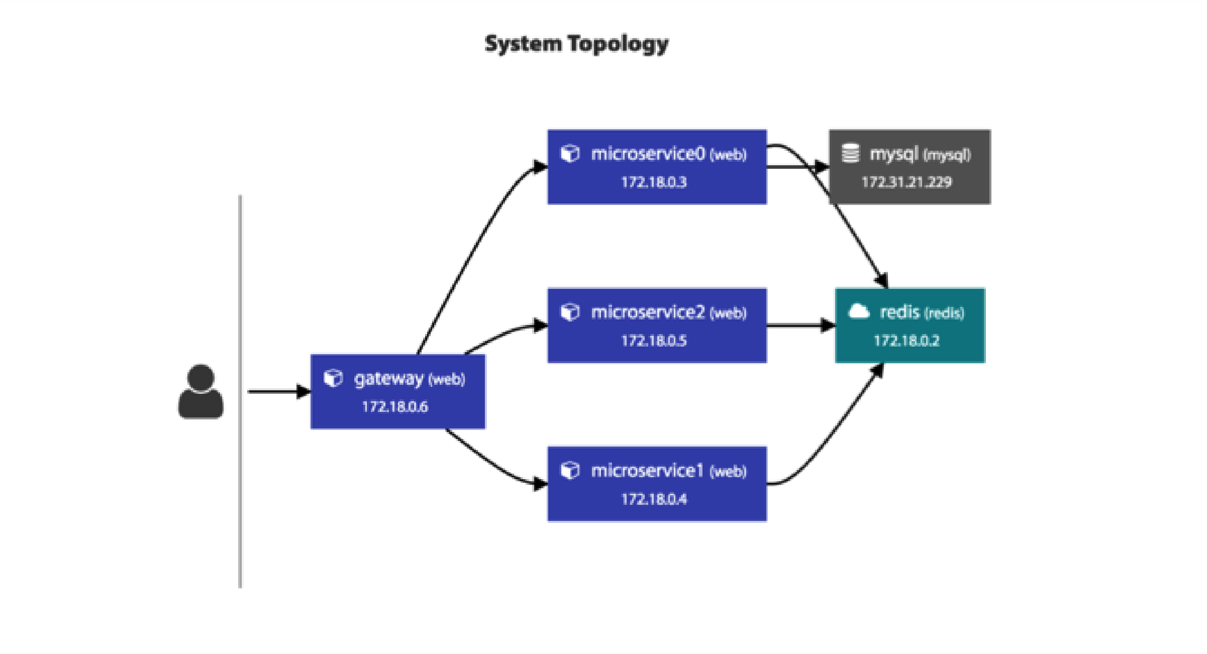
只有做到了上述的这些才能是一个好的监控系统。

如何做出一个好的监控系统

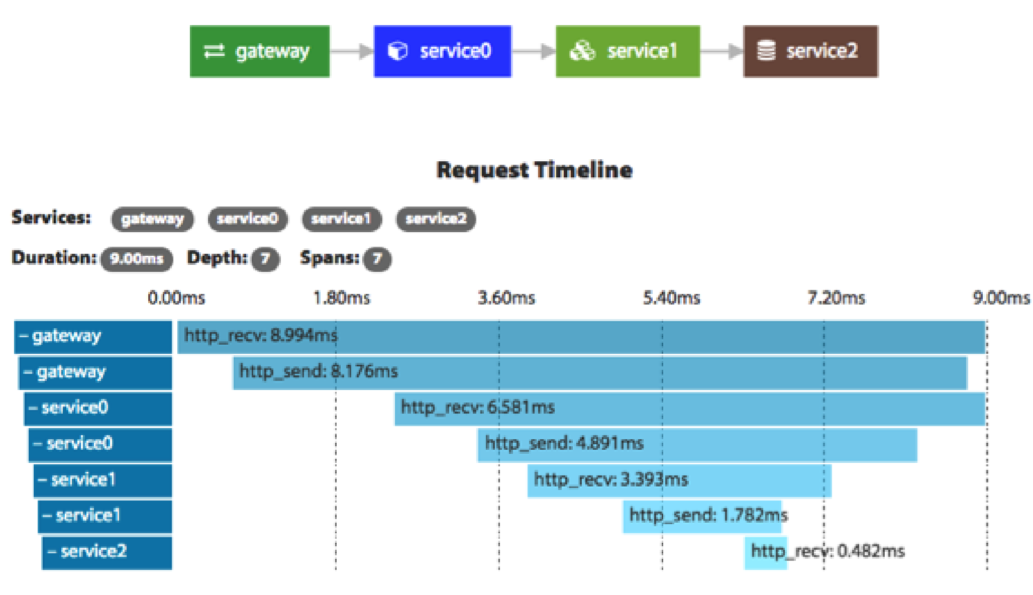
下面是我认为一个好的监控系统应该实现的东西。

- **服务调用链跟踪**。这个监控系统应该从对外的API开始，然后将后台的实际服务给关联起来，再将这个服务的依赖服务给关联起来，直到最后一个服务（如MySQL或Redis），这样就可以把整个系统的服务全部都串连起来了。这个事情的最佳实践是Google Dapper系统，其对应于开源的实现是Zipkin。对于Java类的服务，我们可以使用字节码技术进行字节码注入，做到代码无侵入式。

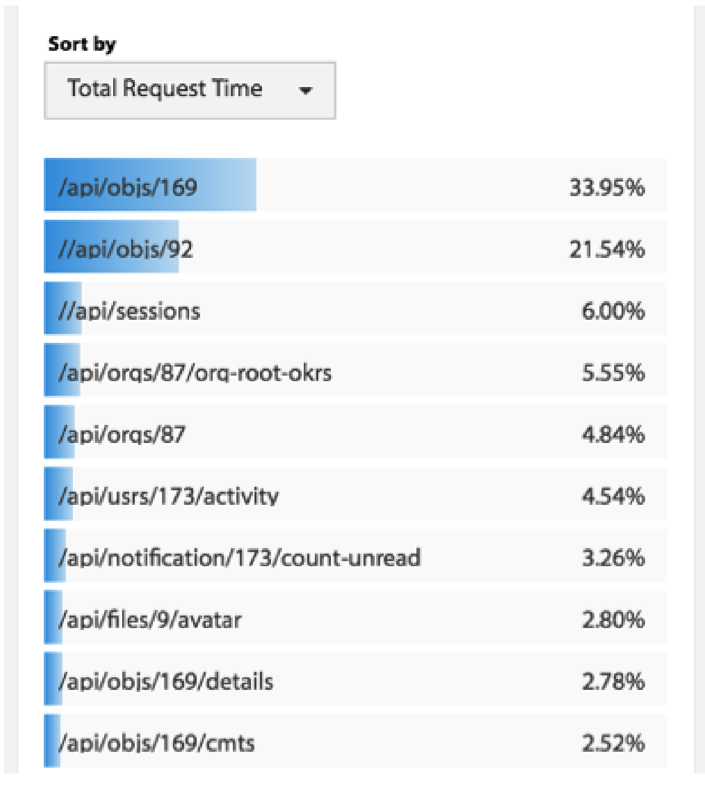
如下图所示（截图来自我做的一个APM的监控系统）。



- **服务调用时长分布**。使用Zipkin，可以看到一个服务调用链上的时间分布，这样有助于我们知道最耗时的服务是什么。下图是Zipkin的服务调用时间分布。



- **服务的TOP N视图**。所谓TOP N视图就是一个系统请求的排名情况。一般来说，这个排名会有三种排名的方法：a）按调用量排名，b）按请求最耗时排名，c）按热点排名（一个时间段内的请求次数的响应时间和）。

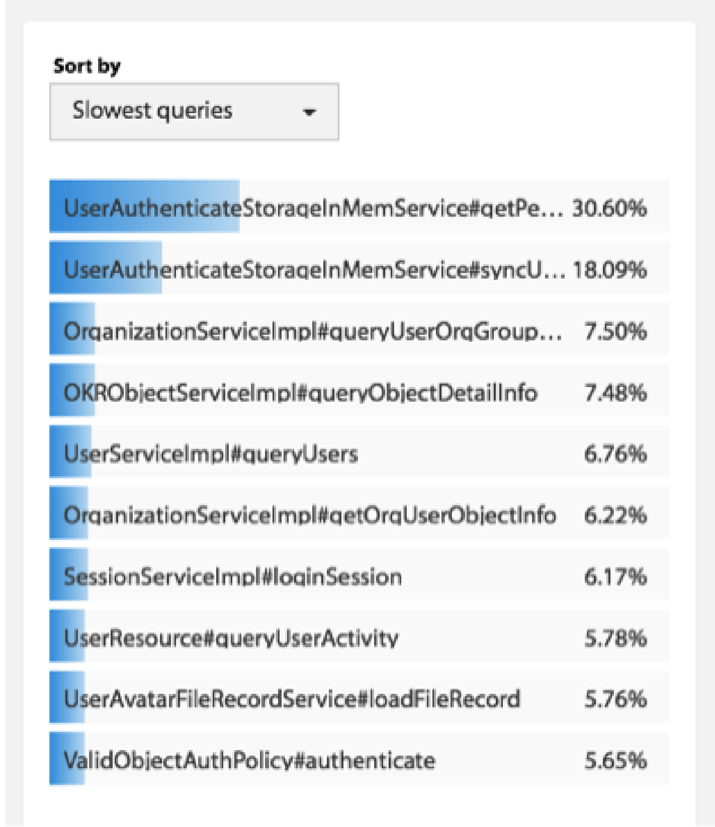


Sort by

Total Request Time

/api/objs/169	33.95%
//api/objs/92	21.54%
//api/sessions	6.00%
/api/orgs/87/org-root-okrs	5.55%
/api/orgs/87	4.84%
/api/usrs/173/activity	4.54%
/api/notification/173/count-unread	3.26%
/api/files/9/avatar	2.80%
/api/objs/169/details	2.78%
/api/objs/169/cmts	2.52%

- **数据库操作关联**。对于Java应用，我们可以很方便地通过JavaAgent字节码注入技术拿到JDBC执行数据库操作的执行时间。对此，我们可以和相关的请求对应起来。



Sort by

Slowest queries

UserAuthenticateStorageInMemService#getPe…	30.60%
UserAuthenticateStorageInMemService#syncU…	18.09%
OrganizationServiceImpl#queryUserOrgGroup…	7.50%
OKRObjectServiceImpl#queryObjectDetailInfo	7.48%
UserServiceImpl#queryUsers	6.76%
OrganizationServiceImpl#getOrgUserObjectInfo	6.22%
SessionServiceImpl#loginSession	6.17%
UserResource#queryUserActivity	5.78%
UserAvatarFileRecordService#loadFileRecord	5.76%
ValidObjectAuthPolicy#authenticate	5.65%

- **服务资源跟踪**。我们的服务可能运行在物理机上，也可能运行在虚拟机里，还可能运行在一个Docker的容器里，Docker容器又运行在物理机或是虚拟机上。我们需要把服务运行的机器节点上的数据（如CPU、MEM、I/O、DISK、NETWORK）关联起来。

这样一来，我们就可以知道服务和基础层资源的关系。如果是Java应用，我们还要和JVM里的东西进行关联，这样我们才能知道服务所运行的JVM中的情况（比如GC的情况）。

有了这些数据上的关联，我们就可以达到如下的目标。

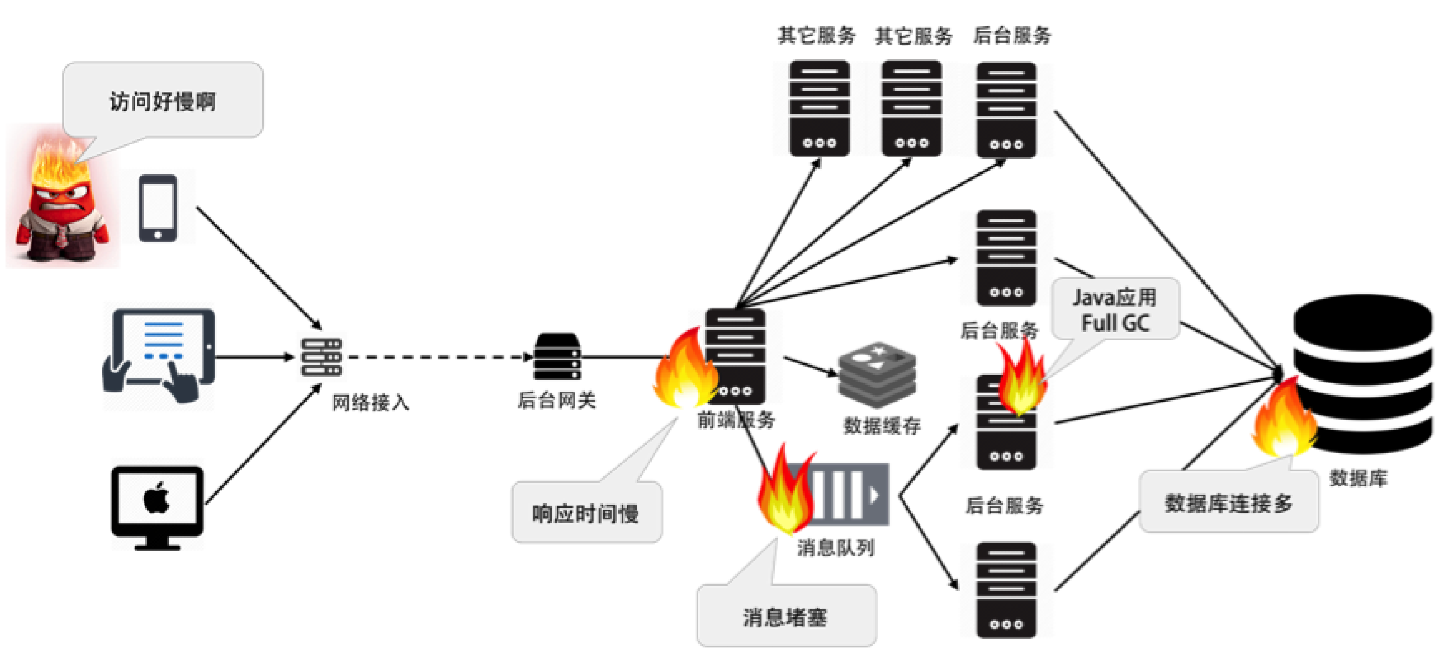
1. 当一台机器挂掉是因为CPU或I/O过高的时候，我们马上可以知道其会影响到哪些对外服务的API。
2. 当一个服务响应过慢的时候，我们马上能关联出来是否在做Java GC，或是其所在的计算结点上是否有资源不足的情况，或是依赖的服务是否出现了问题。
3. 当发现一个SQL操作过慢的时候，我们能马上知道其会影响哪个对外服务的API。
4. 当发现一个消息队列拥塞的时候，我们能马上知道其会影响哪些对外服务的API。

总之，我们就是想知道用户访问哪些请求会出现问题，这对于我们了解故障的影响面非常有帮助。

一旦了解了这些信息，我们就可以做出调度。比如：

- 一旦发现某个服务过慢是因为CPU使用过多，我们就可以做弹性伸缩。
- 一旦发现某个服务过慢是因为MySQL出现了一个慢查询，我们就无法在应用层上做弹性伸缩，只能做流量限制，或是降级操作了。

所以，一个分布式系统，或是一个自动化运维系统，或是一个Cloud Native的云化系统，最重要的事就是把监控系统做好。在把数据收集好的同时，更重要的是把数据关联好。这样，我们才可能很快地定位故障，进而才能进行自动化调度。



上图只是简单地展示了一个分布式系统的服务调用链接上都在报错，其根本原因是数据库链接过多，服务不过来。另外一个原因是，Java在做Full GC导致处理过慢。于是，消息队列出现消息堆积堵塞。这个图只是一个示例，其形象地体现了在分布式系统中监控数据关联的重要性。

小结

回顾一下今天的要点内容。首先，我强调了全栈系统监控的重要性，它就像是我们的眼睛，没有它，我们根本就不知道系统到底发生了什么。随后，从基础层、中间层和应用层三个层面，讲述了全栈监控系统要监控哪些内容。然后，阐释了什么才是好的监控系统，以及如何做出好的监控。最后，欢迎你分享一下你在监控系统中的比较好的实践和方法。

下一篇文章中，我将讲述分布式系统的另一关键技术：服务调度。

文末给出了《分布式系统架构的本质》系列文章的目录，方便你快速找到自己感兴趣的内容。