bobby · 更新于 2022-11-16

上一节 17. 常见的幂等... 2. 链路追踪技术... 下一节

分布式链路追踪（Distributed Tracing），也叫 分布式链路跟踪，分布式跟踪，分布式追踪 等等。
本文使用分布式Trace来简称分布式链路追踪。
本篇文章只是从大致的角度来阐述什么是分布式Trace，以及一个分布式Trace系统具备哪些要点和特征。

场景

先从几个场景来看为什么需要分布式Trace

场景1

开发A编写了一段代码，代码依赖了很多的接口。一个调用下去没出结果，或者超时了，Debug之后发现是接口 `M` 挂了，然后找到这个接口 `M` 的负责人B，告知B接口挂了。B拉起自己的调用和Debug环境，按照之前传过来的调用方式重新Debug了一遍自己的接口，发现NND是自己依赖的接口 `N` 挂了，然后找到接口 `N` 负责人C。C同样Debug了自己的接口（此处省略一万个‘怎么可能呢，你调用参数不对吧’），最终发现是某个空判断错误，修复bug，转告给B说我们bug修复了，B再转告给A说，是C那个傻x弄挂了，现在Ok了，你试一下。
就这样，一个上午就没了，看着手头的需求越堆越高，内心是这样

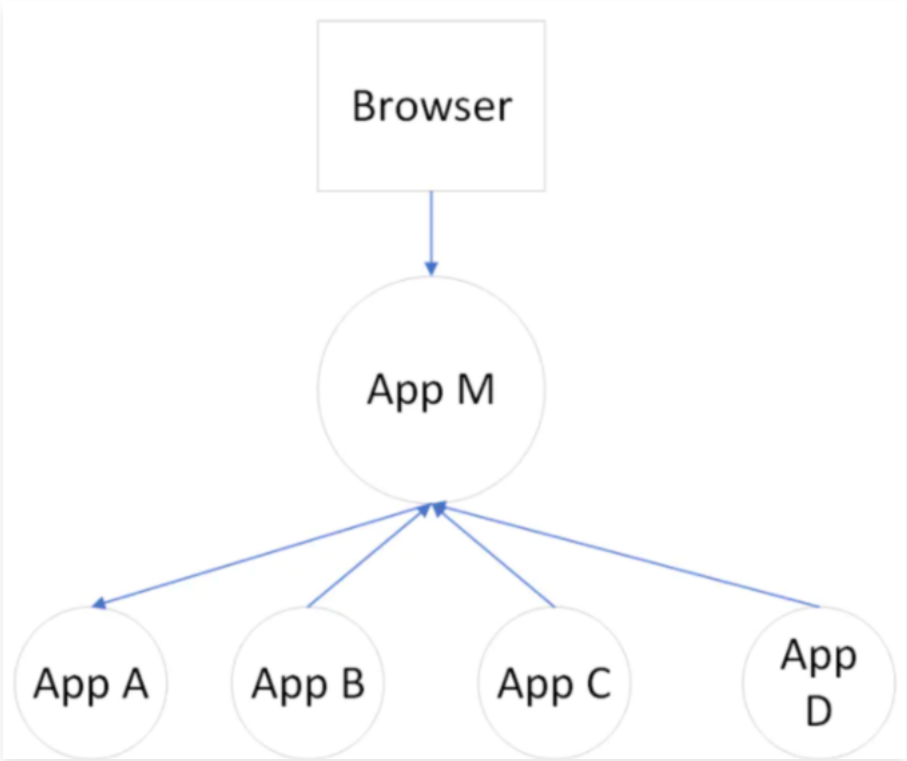
场景2

哪一天系统完成了开发，需要进行性能测试，发现哪些地方调用比较慢，影响了全局。A工程师拉起自己的系统，调用一遍，就汇报给老板，时间没啥问题。B工程师拉起自己的系统，调用了一遍，也没啥问题，同时将结果汇报了给老板。C工程师这时候发现自己的系统比较慢，debug发现原来是自己依赖的接口慢了，于是找到接口负责人。。balabala，和场景1一样，弄好了。老板一一把这些都记录下来，满满的一本子。哪天改了个需求，又重新来一遍，劳民伤财。

解决方案

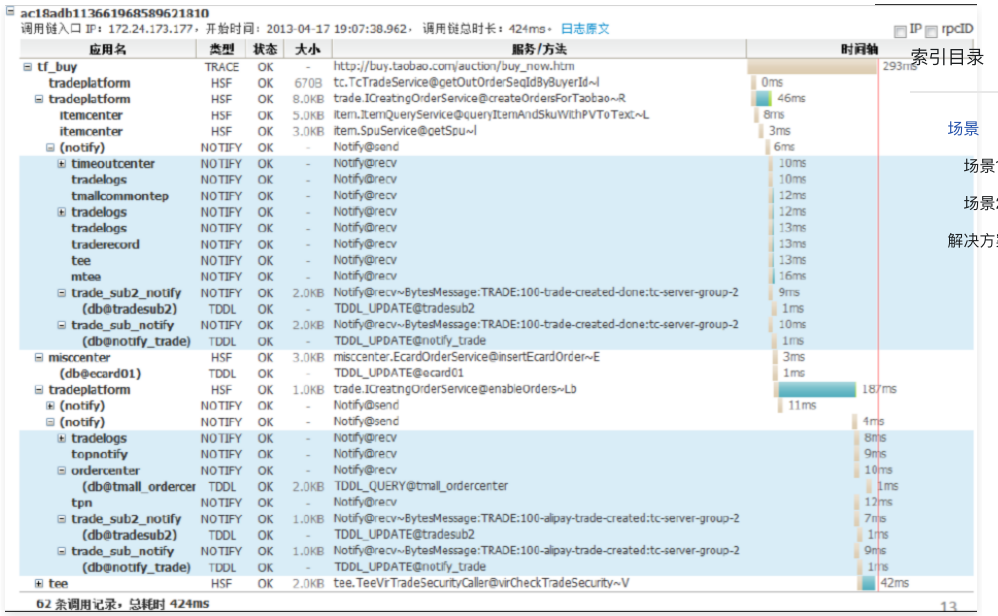
这两种场景只是缩影，假设这时候有这样一种系统，

它记录了所有系统的调用和依赖，以及这些依赖之间的关系和性能。打个比方，一个网页访问了应用M，应用M又分别访问了A，B，C，D四个应用，如下面这样的结构



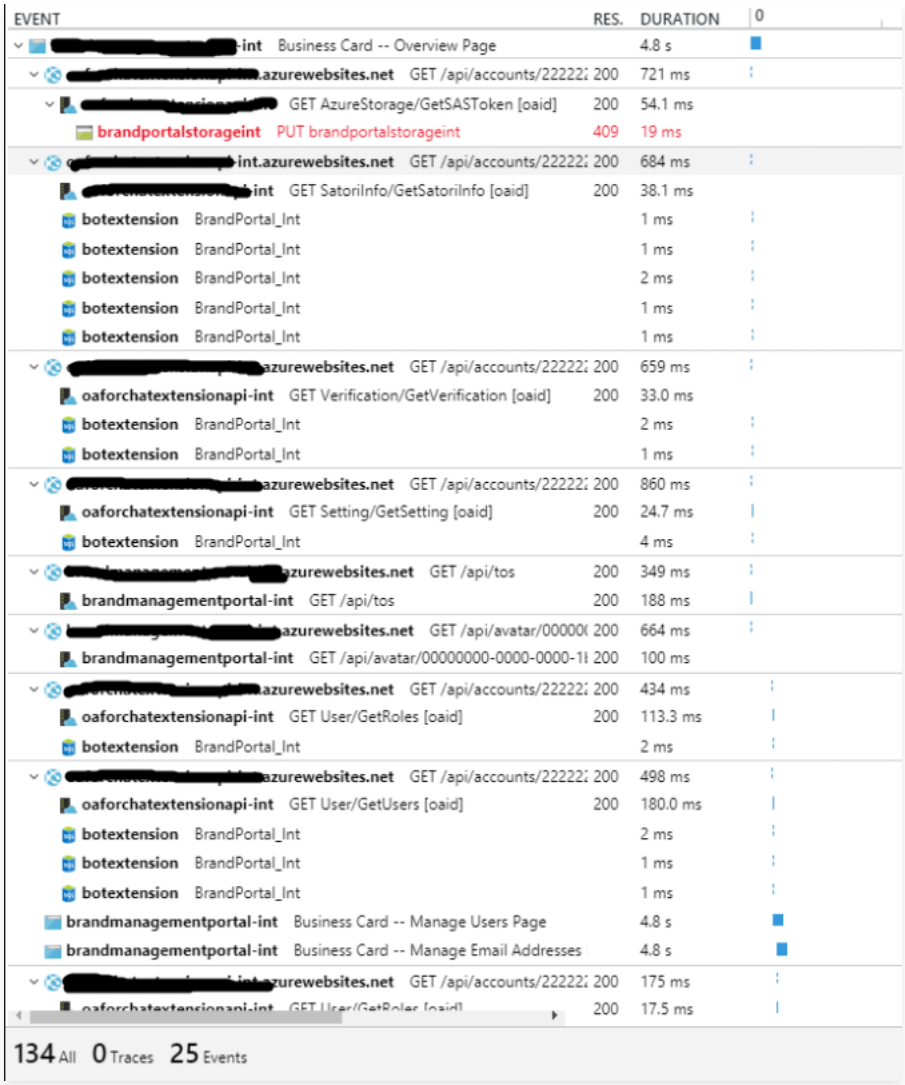
那么在这个系统中就能够看到，一个网页Request了一个应用M，花费了多少时间，请求的IP是多少，请求的网络开销是多少。应用M执行时间是多久，是否执行成功，访问A，B，C，D分别花了多少时间，是否成功，返回了什么内容，测试是否通过。 然后到下一步，A，B，C，D四个应用本次执行的时间是多久，有没有超时，调用了多少次DB，每次调用花费了多少时间。

作为示例，给出一个阿里鹰眼的trace图：



trace就犹如一张大的json表，同一层级的数据代表同一层级的应用，越往下代表是对下层某个应用的依赖。从图中可以很方便的看到每一个应用调用的名称，调用花费的时间，以及是否成功。

下面这张图是我们使用微软的application insights生成的tracing图



有些敏感数据打上了马赛克，尽请谅解。不过还是可以清晰的看到应用之间的依赖关系，有处标红来代表此次调用出现了问题。

有了这个系统，场景1和场景2中的需求就能解决吗？如果有了分布式trace，这些场景中的问题又是怎么解决呢？

对于场景1中的case，开发A发现自己的接口挂了或者比较慢，而且Debug发现并不是自己代码的错误，这时候他找到自己的这一次trace，图中就会列出来这一次trace的所有依赖和调用，以及各调用之间的关系。A发现，自己调用的链路到 N 接口那里就断了，并且调用 N 接口返回500错误，于是A直接和 N 接口的负责人C联系，C立马修复了错误。

在A调用出错的时候，系统自动检测出在 N 接口出错，系统立马生成一份错误报告发到A和C的邮箱，A拿到报告的时候就直接能够知道那个环节出错了，而C拿到报告的时候发现，A在调用我的接口，并且我的接口出错了。这就是出错的主动通知。

对于场景2，项目开发完成了，或者有新的pull request merge到主分支了，触发了自动化测试。测试下来同样生成一张链路分析图，不管是开发，测试，DBA，还是老板，很容易从里面看到哪些应用的响应速度慢了，读取DB的时间慢了，接口挂了这些参数。再也不用一个一个搜集评测报告了。加快了持续集成和持续迭代。

分布式Trace关乎到的不仅仅是开发，运维，还有测试，DBA，以及你老板的工作量。

上面的例子只是一个缩影，如果一个公司内部存在成千上万个接口调用，到时候接口负责人都找不到的时候，时间成本和沟通成本无法想象。

17. 常见的幂等性解决方案 上一节 下一节 2. 链路追踪技术选型

我要提出意见反馈