EagleEye——分布式调用的跟踪者

综述

阿里巴巴电子商务平台现在是一个由很多个应用集群组成的非常复杂的分布式系统。这些应用里面主要有处理用户请求的前端系统和有提供服务的后端系统等，各个应用之间一般有RPC调用和异步消息通讯两种手段，RPC 调用会产生一层调一层的嵌套，一个消息发布出来更会被多个应用消费。另外，应用还会访问分库分表的数据库、缓存、存储等后端，以及调用其他外部系统如支付、物流、机彩票等。

请试想一下，现在淘宝一个买家点击下单按钮所产生的网络请求到达淘宝服务器之后，就会触发淘宝内网数百次的网络调用。这些调用中有哪些出问题会影响这次交易，有哪些步骤会拖慢整个处理流程，双十一的交易高峰需要给应用集群分配多少台机器，这些都是支撑双十一需要考虑的。但是调用环境的复杂度，已经很难用人力去做准确的分析和评估了，这时候EagleEye就派上了用场。

4.1、EagleEye

EagleEye （鹰眼）是Google 的分布式调用跟踪系统 Dapper 在淘宝的实现。分布式调用跟踪的意思，就是对一次前端请求产生的分布式调用都汇总起来做分析。同一次请求的所有相关调用的情况，在EagleEye里称作调用链。同一个时刻某一台服务器并行发起的网络调用有很多，怎么识别这个调用是属于哪个调用链的呢？我们可以在各个发起网络调用的中间件上下手。

在前端请求到达服务器时，应用容器在执行实际业务处理之前，会先执行EagleEye的埋点逻辑（类似Filter的机制），埋点逻辑为这个前端请求分配一个全局唯一的调用链ID。这个ID在EagleEye 里面被称为 TraceId，埋点逻辑把TraceId 放在一个调用上下文对象里面，而调用上下文对象会存储在ThreadLocal里面。调用上下文里还有一个ID非常重要，在EagleEye里面被称作RpcId。RpcId用于区分同一个调用链下的多个网络调用的发生顺序和嵌套层次关系。对于前端收到请求，生成的RpcId固定都是0。

当这个前端执行业务处理需要发起RPC调用时，淘宝的RPC调用客户端HSF会首先从当前线程ThreadLocal上面获取之前EagleEye设置的调用上下文。然后，把RpcId递增一个序号。在EagleEye里使用多级序号来表示RpcId，比如前端刚接到请求之后的RpcId是0，那么它第一次调用RPC服务A时，会把RpcId改成0.1。之后，调用上下文会作为附件随这次请求一起发送到远程的HSF服务器。

HSF服务端收到这个请求之后，会从请求附件里取出调用上下文，并放到当前线程ThreadLocal上面。如果服务A在处理时，需要调用另一个服务，这个时候它会重复之前提到的操作，唯一的差别就是RpcId会先改成0.1.1再传过去。服务A的逻辑全部处理完毕之后，HSF在返回响应对象之前，会把这次调用情况以及TraceId、RpcId都打印到它的访问日志之中，同时，会从ThreadLocal清理掉调用上下文。如图4-1展示了一个浏览器请求可能触发的系统间调用。

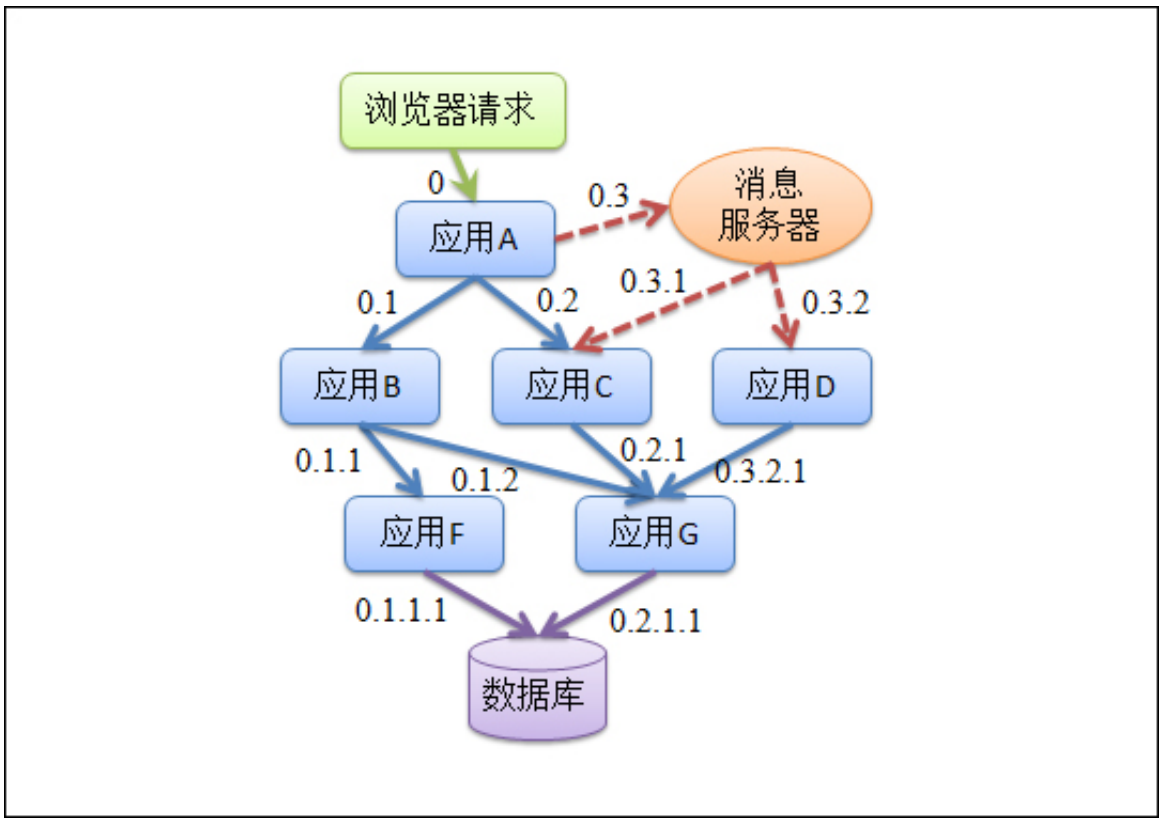


图4-1-一个浏览器请求可能触发的系统间调用

图4-1描述了EagleEye在一个非常简单的分布式调用场景里做的事情，就是为每次调用分配TraceId、RpcId，放在ThreadLocal的调用上下文上面，调用结束的时候，把TraceId、RpcId打印到访问日志。类似的其他网络调用中间件的调用过程也都比较类似，这里不再赘述了。访问日志里面，一般会记录调用时间、远端IP地址、结果状态码、调用耗时之类，也会记录与这次调用类型相关的一些信息，如URL、服务名、消息topic等。很多调用场景会比上面说的完全同步的调用更为复杂，比如会遇到异步、单向、广播、并发、批处理等等，这时候需要妥善处理好ThreadLocal上的调用上下文，避免调用上下文混乱和无法正确释放。另外，采用多级序号的RpcId设计方案会比单级序号递增更容易准确还原当时的调用情况。

最后，EagleEye分析系统把调用链相关的所有访问日志都收集上来，按TraceId汇总在一起之后，就可以准确还原调用当时的情况了。

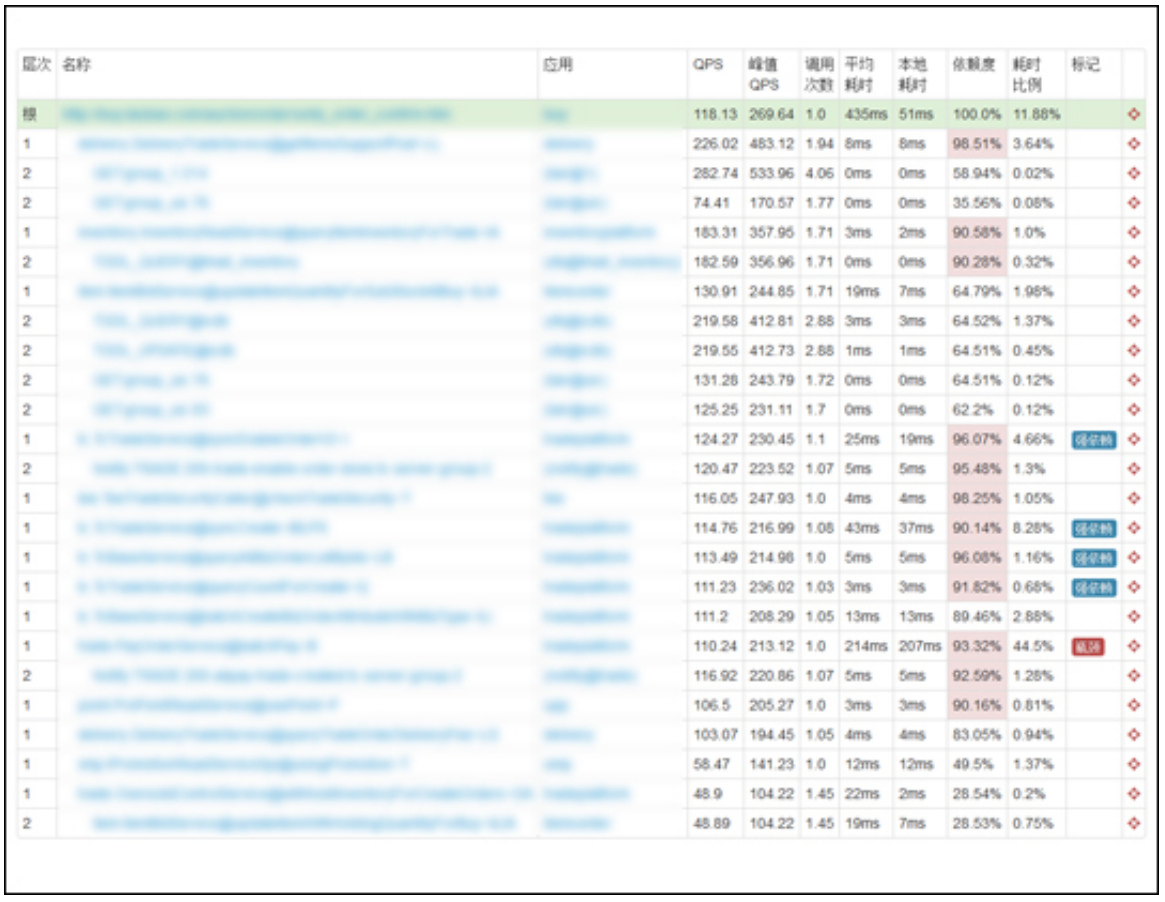
图4-2-一个典型的调用链

如图4-2所示，就是采集自淘宝线上环境的某一条实际调用链。调用链通过树形展现了调用情况。调用链可以清晰的看到当前请求的调用情况，帮助问题定位。如上图，mtop应用发生错误时，在调用链上可以直接看出这是因为第四

层的一个(tair@1)请求导致网络超时，使最上层页面出现超时问题。这种调用链，可以在EagleEye系统监测到包含异常的访问日志后，把当前的错误与整个调用链关联起来。问题排查人员在发现入口错误量上涨或耗时上升时，通过EagleEye查找出这种包含错误的调用链采样，提高故障定位速度。

调用链数据在容量规划和稳定性方面的分析

如果对同一个前端入口的多条调用链做汇总统计，也就是说，把这个入口URL下面的所有调用按照调用链的树形结构全部叠加在一起，就可以得到一个新的树结构（如图4-3所示）。这就是入口下面的所有依赖的调用路径情况。



层次	名称	应用	QPS	峰值QPS	调用次数	平均耗时	本地耗时	依赖度	耗时比例	标记	
根	[redacted]	[redacted]	118.13	269.64	1.0	435ms	51ms	100.0%	11.88%		
1	[redacted]	[redacted]	226.02	483.12	1.94	8ms	8ms	98.51%	3.64%		
2	[redacted]	[redacted]	282.74	533.96	4.06	0ms	0ms	58.94%	0.02%		
2	[redacted]	[redacted]	74.41	170.57	1.77	0ms	0ms	35.56%	0.08%		
1	[redacted]	[redacted]	183.31	357.95	1.71	3ms	2ms	90.58%	1.0%		
2	[redacted]	[redacted]	182.59	356.96	1.71	0ms	0ms	90.28%	0.32%		
1	[redacted]	[redacted]	130.91	244.85	1.71	19ms	7ms	64.79%	1.98%		
2	[redacted]	[redacted]	219.58	412.81	2.88	3ms	3ms	64.52%	1.37%		
2	[redacted]	[redacted]	219.55	412.73	2.88	1ms	1ms	64.51%	0.45%		
2	[redacted]	[redacted]	131.28	243.79	1.72	0ms	0ms	64.51%	0.12%		
2	[redacted]	[redacted]	125.25	231.11	1.7	0ms	0ms	62.2%	0.12%		
1	[redacted]	[redacted]	124.27	230.45	1.1	25ms	19ms	96.07%	4.66%	强依赖	
2	[redacted]	[redacted]	120.47	223.52	1.07	5ms	5ms	95.48%	1.3%		
1	[redacted]	[redacted]	116.05	247.93	1.0	4ms	4ms	98.25%	1.05%		
1	[redacted]	[redacted]	114.76	216.99	1.08	43ms	37ms	90.14%	8.28%	强依赖	
1	[redacted]	[redacted]	113.49	214.98	1.0	5ms	5ms	96.08%	1.16%	强依赖	
1	[redacted]	[redacted]	111.23	236.02	1.03	3ms	3ms	91.82%	0.68%	强依赖	
1	[redacted]	[redacted]	111.2	208.29	1.05	13ms	13ms	89.46%	2.88%		
1	[redacted]	[redacted]	110.24	213.12	1.0	214ms	207ms	93.32%	44.5%	慢SQL	
2	[redacted]	[redacted]	116.92	220.86	1.07	5ms	5ms	92.59%	1.28%		
1	[redacted]	[redacted]	106.5	205.27	1.0	3ms	3ms	90.16%	0.81%		
1	[redacted]	[redacted]	103.07	194.45	1.05	4ms	4ms	83.05%	0.94%		
1	[redacted]	[redacted]	58.47	141.23	1.0	12ms	12ms	49.5%	1.37%		
1	[redacted]	[redacted]	48.9	104.22	1.45	22ms	2ms	28.54%	0.2%		
2	[redacted]	[redacted]	48.89	104.22	1.45	19ms	7ms	28.53%	0.75%		

图4-3-对某个入口的调用链做统计之后得到的依赖分析

这种分析能力对于复杂的分布式环境的调用关系梳理尤为重要。传统的调用统计日志是按固定时间窗口预先做了统计的日志，上面缺少了链路细节导致没办法对超过两层以上的调用情况进行分析。例如，后端数据库就无法评估数据库访问是来源于最上层的哪些入口；每个前端系统也无法清楚确定当前入口由于双十一活动流量翻倍，会对后端哪些系统造成多大的压力，需要分别准备多少机器。有了EagleEye的数据，这些问题就迎刃而解了。

EagleEye还能得出一些特有的统计数据，帮助各个业务系统评估依赖上的一些潜在风险。我们知道，在正常的分布式系统里，异常的发生概率是很低的，长期运行的调用超过千亿次，发生错误也只会寥寥可数，错误信息可能只在日志中一闪而过就被其他信息湮没。EagleEye可以抓住线上真实环境"稍纵即逝"的错误机会，分析到调用路径上几乎每个依赖发生错误时对调用链整体的影响情况。

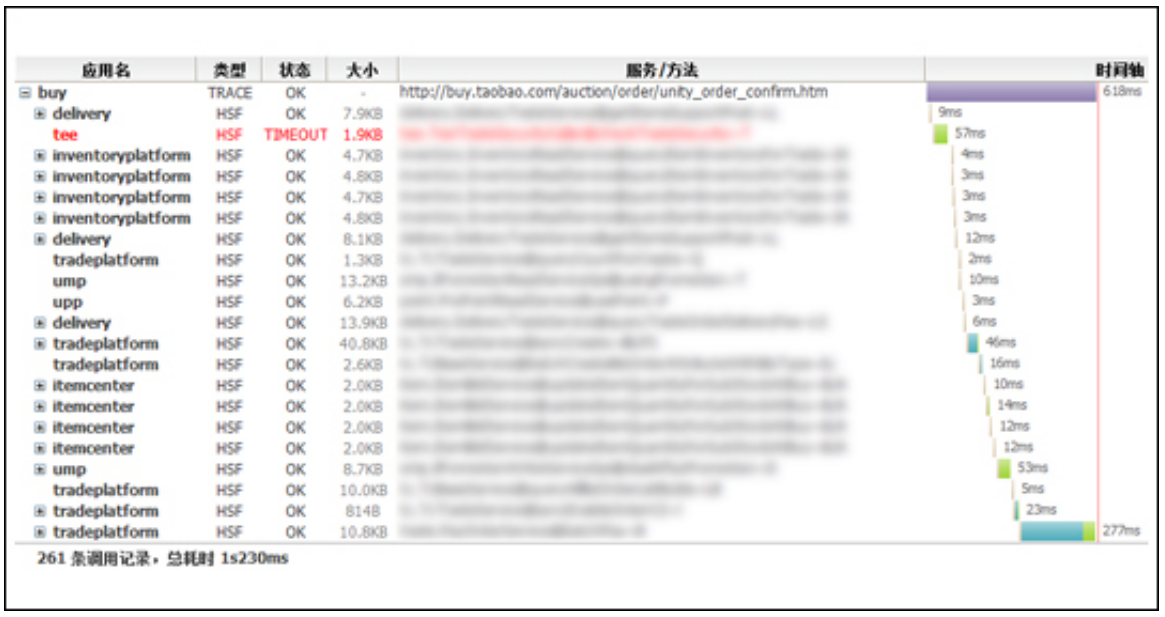


图4-4-弱依赖发生错误时的调用链

如图4-4所示，应用 tee由于存在调用错误，在 EagleEye 的调用链里被标记为红色，但是它的异常并没有影响调用链继续往下执行。EagleEye里面把这种依赖识别为弱依赖。那么强依赖又是怎样的呢？如图4-5所示，tradeplatform的这个调用出现了错误，导致当前的入口请求被直接中断（读者可以对比上图的弱依赖里面，tradeplatform调用正常返回之后还有很多其他的调用会发生），EagleEye把这种类型的依赖标识作强依赖。在双十一大并发的高压之下，任何错误出现的可能都会被成倍放大。对于系统内的依赖情况我们需要提前做好甄别，尽量降低在关键路径上的强依赖发生错误的机会，并把非关键依赖降级为弱依赖。

图4-5-强依赖发生错误时的调用链

对于弱依赖，我们也不能掉以轻心。EagleEye识别了另外一种潜在的问题模式。请看下图的调用链，delivery应用发生错误时，从调用链来看，尽管它是弱依赖，但是因为它的错误占用了3秒时间，导致整个页面的响应也长达3秒。从用户角度看，这是非常不好的体验；从系统层面看，一个弱依赖错误导致系统一个处理线程无法释放，意味着如果这个弱依赖真的挂了，系统此刻大部分处理线程会有可能都堵塞在这个服务调用上，从而整个系统的吞吐量会降低很多。

应用名	类型	状态	大小	服务/方法	时间轴
tmallbuy	TRACE	OK	-	http://buy.tmall.com/order/confirm_order.htm	3s130ms
tradeplatform	HSF	OK	1.3KB		3ms
delivery	HSF	TIMEOUT	845B		3s8ms
itemcenter	HSF	OK	10.8KB		9ms
itemcenter	HSF	OK	3.5KB		4ms
pointcenter	HSF	OK	681B		2ms
memberplatform	HSF	OK	1.3KB		3ms
inventoryplatfor	HSF	OK	4.7KB		3ms
delivery	HSF	OK	7.0KB		7ms
maybach	HSF	OK	12.5KB		42ms
tradeplatform	HSF	OK	732B		1ms
delivery	HSF	OK	12.3KB		7ms
tradeplatform	HSF	OK	3.3KB		18ms
tradeplatform	HSF	OK	816B		1ms

图4-6-弱依赖的超时异常导致线程堵塞

EagleEye可以得到相关的依赖列表，并且对依赖的错误影响作了一定判别，通过这份数据，可以再人为模拟出问题场景来验证这个异常是否确实存在，处理后再验证是否彻底修复。相关的内容可以利用后面提到的稳定性平台做检测。

4.2、EagleEye双11准备与优化

- 应对双十一的日志输出问题

  EagleEye 的调用日志是每次网络调用都会打印一条，在双十一高峰期日志的打印量会非常大，如何降低 EagleEye 的日志输出对应用的影响呢？

  采用通用日志框架会引入很多不需要的特性，带了更多性能损耗，为了提高性能和对写日志底层做更细致的控制，EagleEye 自己实现了日志输出：利用无锁环形队列做日志异步写入来避免hang住业务线程，调节日志输出缓冲大小，控制每秒写日志的IO次数等。

  另外还有很重要的一点就是设置全局采样开关，用来在运行期控制调用链的采样率。所谓调用链采样，就是根据TraceId来决定当前的这一次访问日志是否输出。比如采样率被设置为10时，只有hash(traceId) mod 10的值等于0的日志才会输出，这样可以保证有一部分调用链日志完全不输出，有一部分调用链会完整输出。由于核心入口的调用量都在每日百万次以上级别，样本空间足够大，实测开启1/10的采样下，调用量统计误差在0.1%左右，大于10万以上的调用统计点，误差超过5% 的概率为2% 左右，是可以接受的，因此采样会作为常态化的配置存在。

小结

EagleEye是基于网络调用日志的分布式跟踪系统，它可以分析网络请求在各个分布式系统之间的调用情况，从而得到处理请求的调用链上的入口URL、应用、服务的调用关系，从而找到请求处理瓶颈，定位错误异常的根源位置。同时，业务方也可以在调用链上添加自己的业务埋点日志，使各个系统的网络调用与实际业务内容得到关联。

系列文章：

《中间件技术及双十一实践之中间件总体介绍》http://jm-blog.aliapp.com/?p=3359

《中间件技术及双十一实践之软负载篇》http://jm-blog.aliapp.com/?p=3450

《中间件技术及双十一实践·服务框架篇》http://jm-blog.aliapp.com/?p=3462

《中间件技术及双十一实践·EagleEye篇》http://jm-blog.aliapp.com/?p=3465

如果觉得内容还行，请分享给更多的人...

转发：中间件技术及双十一实践之中间件总体介绍

转发：中间件技术及双十一实践之软负载篇