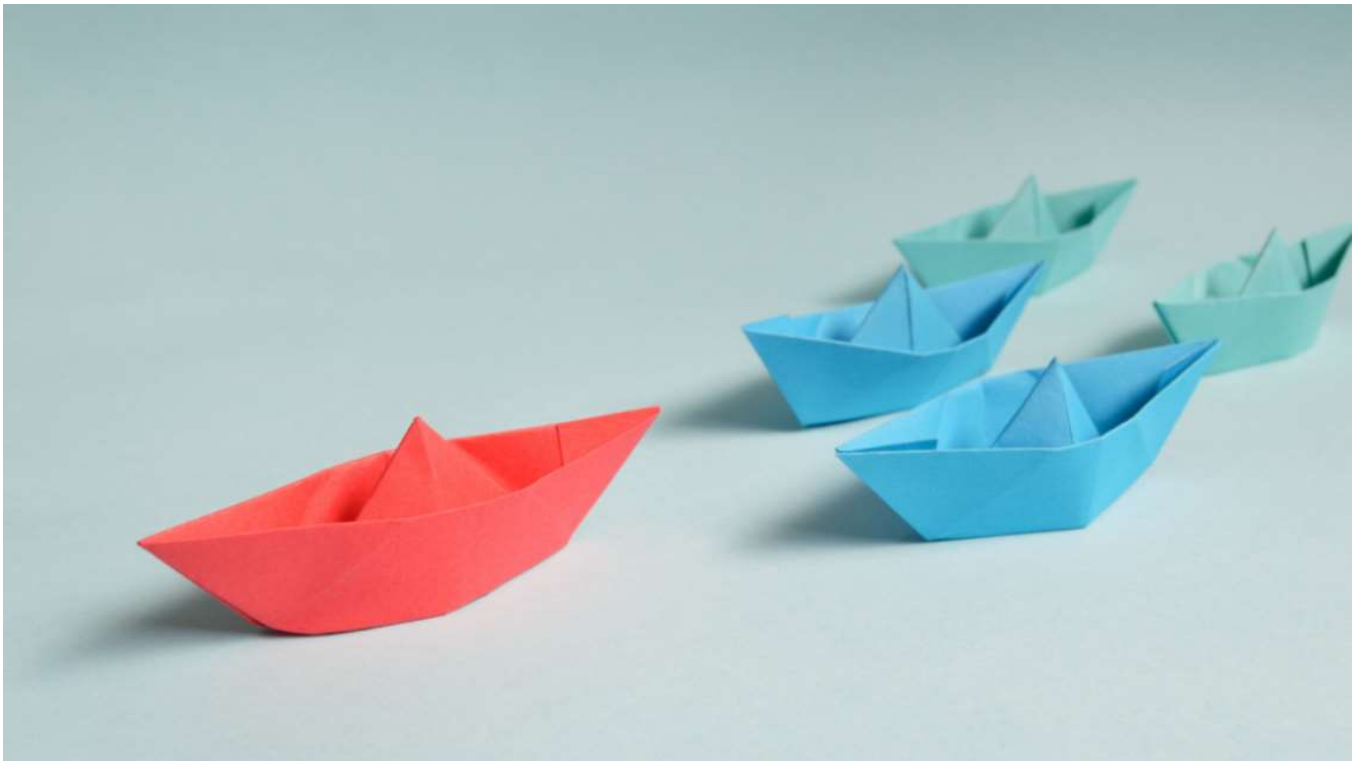
08 | 如何追踪微服务调用？

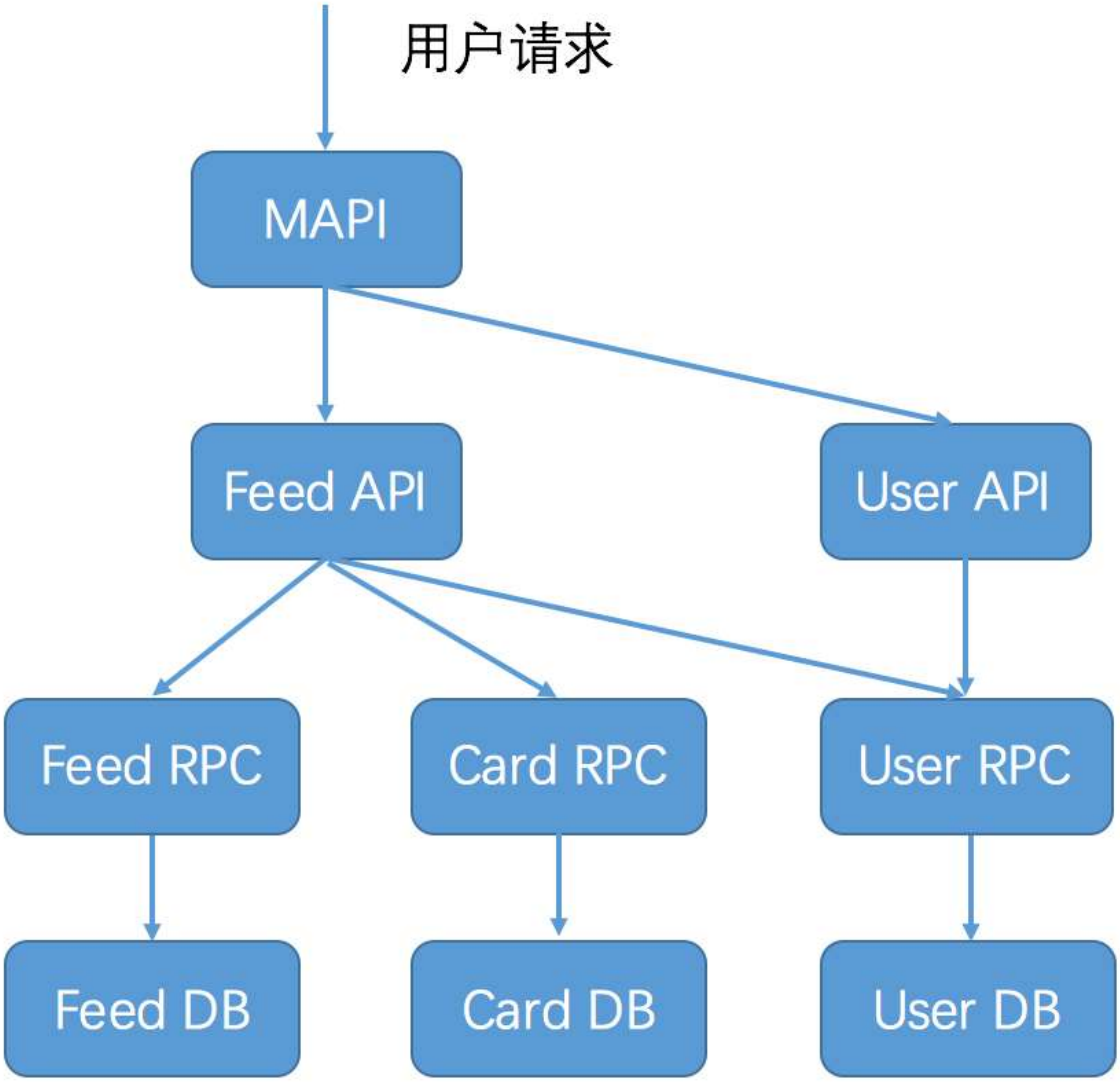
2018-09-08 胡忠想



08 | 如何追踪微服务调用？
朗读人：胡忠想　11′31″ | 5.28M

在微服务架构下，由于进行了服务拆分，一次请求往往需要涉及多个服务，每个服务可能是由不同的团队开发，使用了不同的编程语言，还有可能部署在不同的机器上，分布在不同的数据中心。

下面这张图描述了用户访问微博首页，一次请求所涉及的服务（这张图仅作为示意，实际上可能远远比这张图还要复杂），你可以想象如果这次请求失败了，要想查清楚到底是哪个应用导致，会是多么复杂的一件事情。



如果有一个系统，可以跟踪记录一次用户请求都发起了哪些调用，经过哪些服务处理，并且记录每一次调用所涉及的服务的详细信息，这时候如果发生调用失败，你就可以通过这个日志快速定位是在哪个环节出了问题，这个系统就是今天我要讲解的服务追踪系统。

服务追踪的作用

在介绍追踪原理与实现之前，我们先来看看服务追踪的作用。除了刚才说的能够快速定位请求失败的原因以外，我这里再列出四点，它们可以帮你在微服务改造过程中解决不少问题。

第一，优化系统瓶颈。

通过记录调用经过的每一条链路上的耗时，我们能快速定位整个系统的瓶颈点在哪里。比如你访问微博首页发现很慢，肯定是由于某种原因造成的，有可能是运营商网络延迟，有可能是网关系统异常，有可能是某个服务异常，还有可能是缓存或者数据库异常。通过服务追踪，可以从全局视角上去观察，找出整个系统的瓶颈点所在，然后做出针对性的优化。

第二，优化链路调用。

通过服务追踪可以分析调用所经过的路径，然后评估是否合理。比如一个服务调用下游依赖了多个服务，通过调用链分析，可以评估是否每个依赖都是必要的，是否可以通过业务优化来减少服务依赖。

还有就是，一般业务都会在多个数据中心都部署服务，以实现异地容灾，这个时候经常会出现一种状况就是服务 A 调用了另外一个数据中心的服务 B，而没有调用同处于一个数据中心的服务 B。

根据我的经验，跨数据中心的调用视距离远近都会有一定的网络延迟，像北京和广州这种几千公里距离的网络延迟可能达到 30ms 以上，这对于有些业务几乎是不可接受的。通过对调用链路进行分析，可以找出跨数据中心的服务调用，从而进行优化，尽量规避这种情况出现。

第三，生成网络拓扑。

通过服务追踪系统中记录的链路信息，可以生成一张系统的网络调用拓扑图，它可以反映系统都依赖了哪些服务，以及服务之间的调用关系是什么样的，可以一目了然。除此之外，在网络拓扑图上还可以把服务调用的详细信息也标出来，也能起到服务监控的作用。

第四，透明传输数据。

除了服务追踪，业务上经常有一种需求，期望能把一些用户数据，从调用的开始一直往下传递，以便系统中的各个服务都能获取到这个信息。比如业务想做一些 A/B 测试，这时候就想通过服务追踪系统，把 A/B 测试的开关逻辑一直往下传递，经过的每一层服务都能获取到这个开关值，就能够统一进行 A/B 测试。

服务追踪系统原理

讲到这里，你一定很好奇，服务追踪有这么多好处，那它是怎么做到的呢？

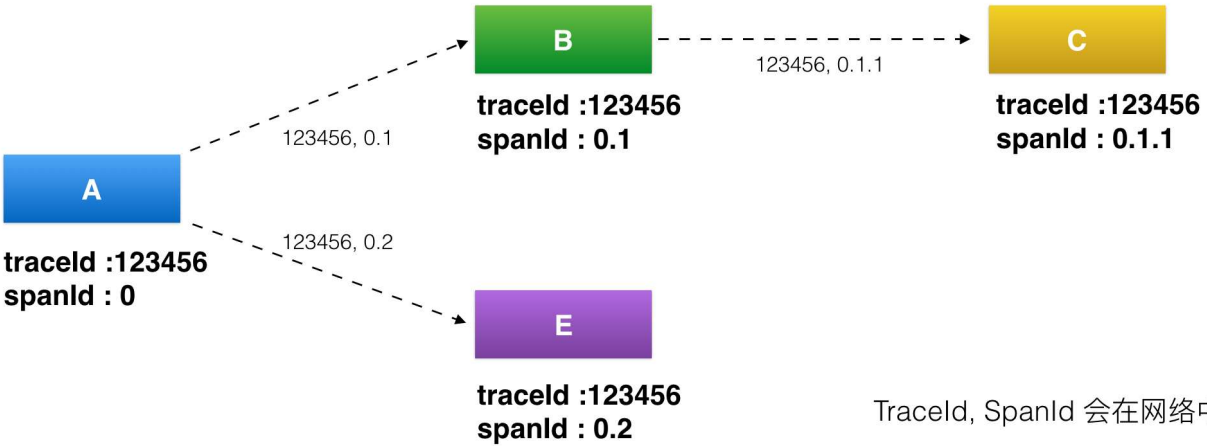
这就不得不提到服务追踪系统的鼻祖：Google 发布的一篇的论文Dapper, a Large-Scale Distributed Systems Tracing Infrastructure，里面详细讲解了服务追踪系统的实现原理。它的核心理念就是调用链：通过一个全局唯一的 ID 将分布在各个服务节点上的同一次请求串联起来，从而还原原有的调用关系，可以追踪系统问题、分析调用数据并统计各种系统指标。

可以说后面的诞生各种服务追踪系统都是基于 Dapper 衍生出来的，比较有名的有 Twitter 的 Zipkin、阿里的鹰眼、美团的MTrace等。

要理解服务追踪的原理，首先必须搞懂一些基本概念：traceId、spanId、annonation 等。Dapper 这篇论文讲得比较清楚，但对初学者来说理解起来可能有点困难，美团的 MTrace 的原理介绍理解起来相对容易一些，下面我就以 MTrace 为例，给你详细讲述服务追踪系统的实现

原理。虽然原理有些晦涩，但却是你必须掌握的，只有理解了服务追踪的基本概念，才能更好地将其实现出来。

首先看下面这张图，我来给你讲解下服务追踪系统中几个最基本概念。



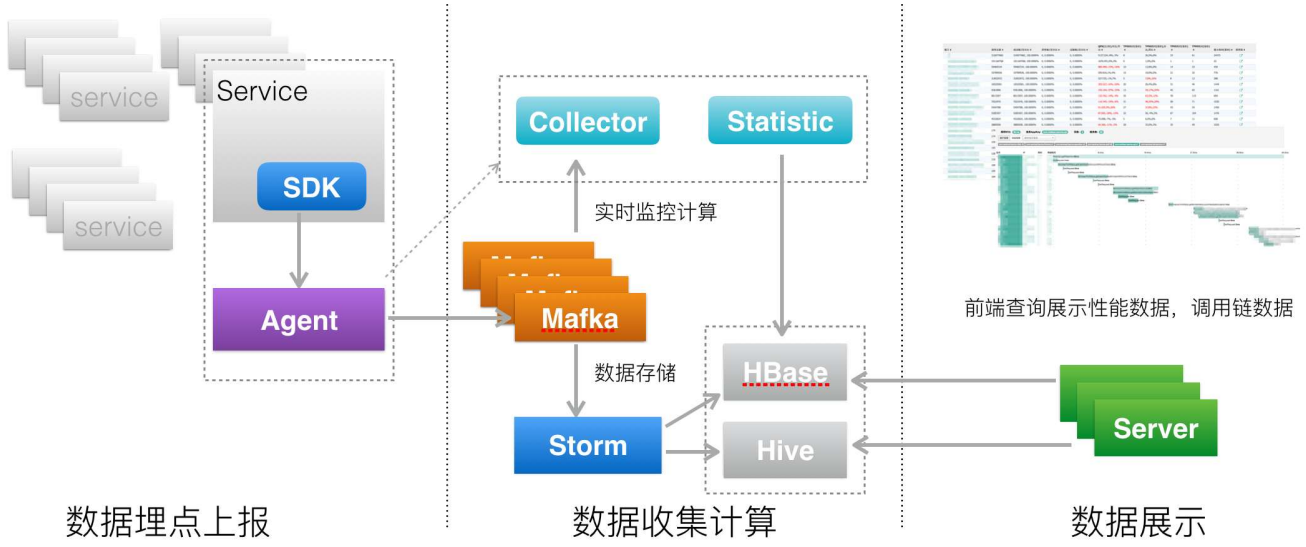
TraceId, SpanId 会在网络中传递

- traceId，用于标识某一次具体的请求 ID。当用户的请求进入系统后，会在 RPC 调用网络的第一层生成一个全局唯一的 traceId，并且会随着每一层的 RPC 调用，不断往后传递，这样的话通过 traceId 就可以把一次用户请求在系统中调用的路径串联起来。
- spanId，用于标识一次 RPC 调用在分布式请求中的位置。当用户的请求进入系统后，处在 RPC 调用网络的第一层 A 时 spanId 初始值是 0，进入下一层 RPC 调用 B 的时候 spanId 是 0.1，继续进入下一层 RPC 调用 C 时 spanId 是 0.1.1，而与 B 处在同一层的 RPC 调用 E 的 spanId 是 0.2，这样的话通过 spanId 就可以定位某一次 RPC 请求在系统调用中所处的位置，以及它的上下游依赖分别是谁。
- annotation，用于业务自定义埋点数据，可以是业务感兴趣的想上传到后端的数据，比如一次请求的用户 UID。

上面这三段内容我用通俗语言再给你小结一下，traceId 是用于串联某一次请求在系统中经过的所有路径，spanId 是用于区分系统不同服务之间调用的先后关系，而 annotation 是用于业务自定义一些自己感兴趣的数据，在上传 traceId 和 spanId 这些基本信息之外，添加一些自己感兴趣的信息。

服务追踪系统实现

讲到这里，你应该已经理解服务追踪系统里最重要的概念和原理了，我们先来看看服务追踪系统的架构，让你了解一下系统全貌。



上面是服务追踪系统架构图，你可以看到一个服务追踪系统可以分为三层。

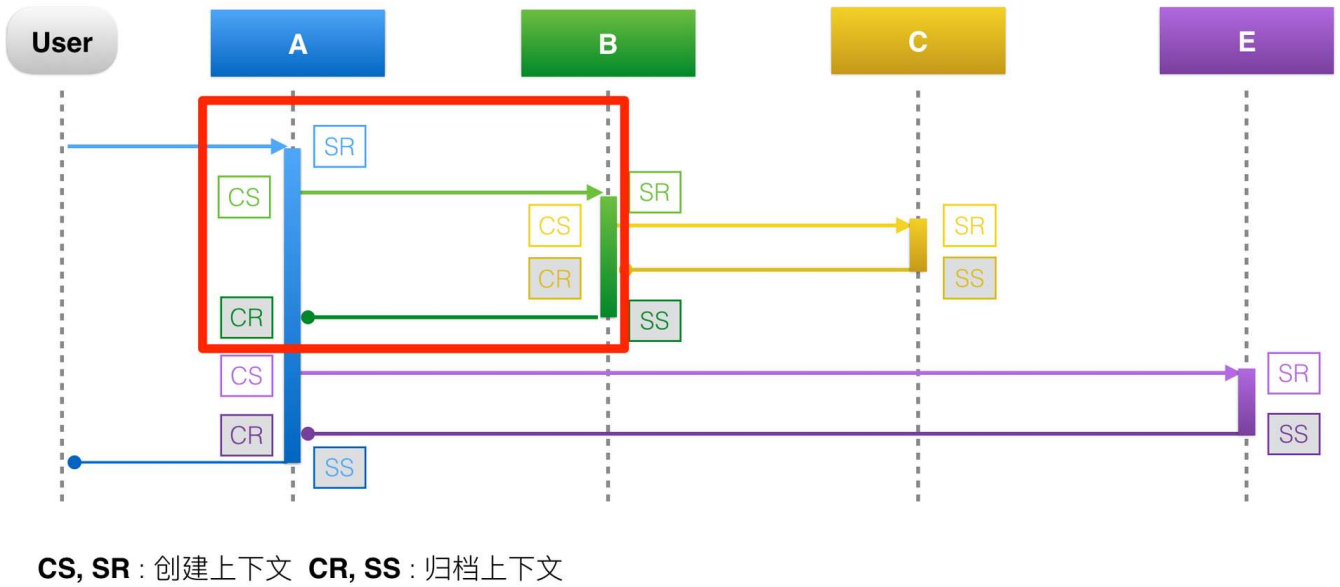
- 数据采集层，负责数据埋点并上报。
- 数据处理层，负责数据的存储与计算。
- 数据展示层，负责数据的图形化展示。

下面来看看具体每一层的实现方式是什么样的。

1. 数据采集层

数据采集层的作用就是在系统的各个不同模块中进行埋点，采集数据并上报给数据处理层进行处理。

那么该如何进行数据埋点呢？结合下面这张图来了解一下数据埋点的流程。



以红色方框里圈出的 A 调用 B 的过程为例，一次 RPC 请求可以分为四个阶段。

- CS（Client Send）阶段：客户端发起请求，并生成调用的上下文。
- SR（Server Recieve）阶段：服务端接收请求，并生成上下文。
- SS（Server Send）阶段：服务端返回请求，这个阶段会将服务端上下文数据上报，下面这张图可以说明上报的数据有：traceId=123456，spanId=0.1，appKey=B，method=B.method，start=103，duration=38。
- CR（Client Recieve）阶段：客户端接收返回结果，这个阶段会将客户端上下文数据上报，上报的数据有：traceid=123456，spanId=0.1，appKey=A，method=B.method，start=103，duration=38。

traceId 123456, spanId 0.1.1, appKey C, method C.method, start 106, duration 30, side server
traceId 123456, spanId 0.1.1, appKey B, method C.method, start 105, duration 33, side client
traceId 123456, spanId 0.1 , appKey B, method B.method, start 103, duration 38, side server
traceId 123456, spanId 0.1 , appKey A, method B.method, start 103, duration 38, side client
traceId 123456, spanId 0.2 , appKey E, method E.method, start 148, duration 12, side server
traceId 123456, spanId 0.2 , appKey A, method E.method, start 146, duration 15, side client
traceId 123456, spanId 0 , appKey A, method A.method, start 100, duration 82, side server

2. 数据处理层

数据处理层的作用就是把数据采集层上报的数据按需计算，然后落地存储供查询使用。

据我所知，数据处理的需求一般分为两类，一类是实时计算需求，一类是离线计算需求。

实时计算需求对计算效率要求比较高，一般要求对收集的链路数据能够在秒级别完成聚合计算，以供实时查询。而离线计算需求对计算效率要求就没那么高了，一般能在小时级别完成链路数据的聚合计算即可，一般用作数据汇总统计。针对这两类不同的数据处理需求，采用的计算方法和存储也不相同。

- 实时数据处理

针对实时数据处理，一般采用 Storm 或者 Spark Streaming 来对链路数据进行实时聚合加工，存储一般使用 OLTP 数据仓库，比如 HBase，使用 traceId 作为 RowKey，能天然地把一整条调用链聚合在一起，提高查询效率。

- 离线数据处理

针对离线数据处理，一般通过运行 MapReduce 或者 Spark 批处理程序来对链路数据进行离线计算，存储一般使用 Hive。

3. 数据展示层

数据展示层的作用就是将处理后的链路信息以图形化的方式展示给用户。

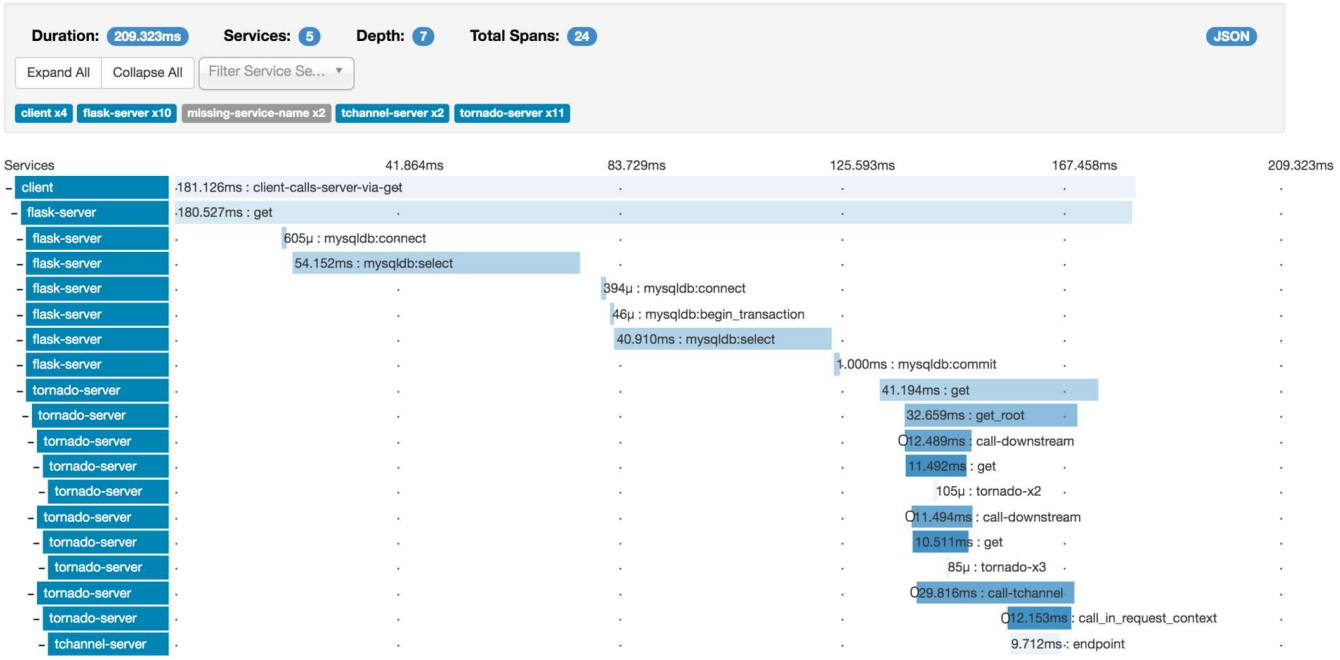
根据我的经验，实际项目中主要用到两种图形展示，一种是调用链路图，一种是调用拓扑图。

- 调用链路图

下面以一张 Zipkin 的调用链路图为例，通过这张图可以看出下面几个信息。

服务整体情况：服务总耗时、服务调用的网络深度、每一层经过的系统，以及多少次调用。下图展示的一次调用，总共耗时 209.323ms，经过了 5 个不同的系统模块，调用深度为 7 层，共发生了 24 次系统调用。

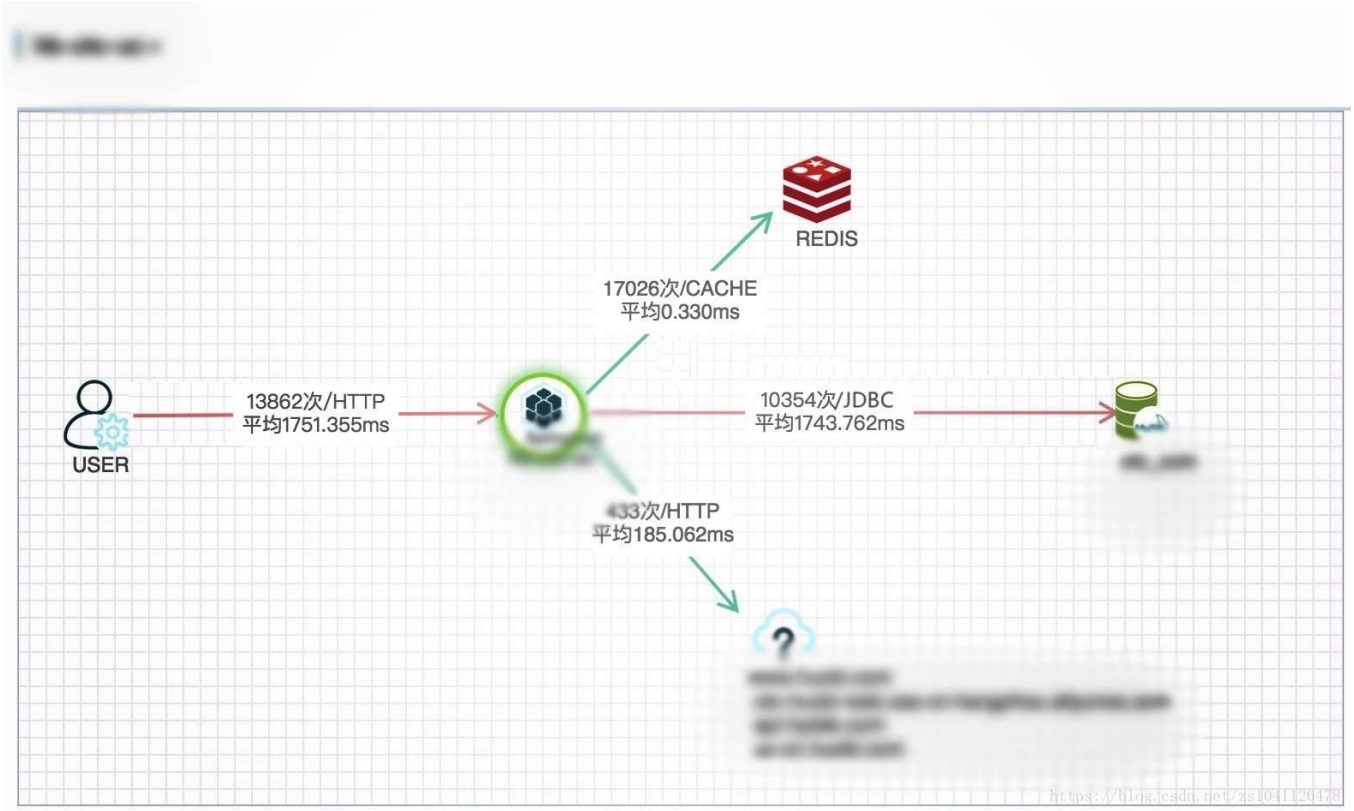
每一层的情况：每一层发生了几次调用，以及每一层调用的耗时。



根据我的经验，调用链路图在实际项目中，主要是被用来做故障定位，比如某一次用户调用失败了，可以通过调用链路图查询这次用户调用经过了哪些环节，到底是哪一层的调用失败所导致。

- 调用拓扑图

下面是一张服务追踪系统的调用链路拓扑图，通过这张图可以看出系统内都包含哪些应用，它们之间是什么关系，以及依赖调用的 QPS、平均耗时情况。



调用拓扑图是一种全局视野图，在实际项目中，主要用作全局监控，用于发现系统中异常的点，从而快速做出决策。比如，某一个服务突然出现异常，那么在调用链路拓扑图中可以看出对这个服务的调用耗时都变高了，可以用红色的图样标出来，用作监控报警。

总结

今天我给你讲解了服务追踪的基本原理以及实现方式，可以说服务追踪是分布式系统中必不可少的功能，它能够帮助我们查询一次用户请求在系统中的具体执行路径，以及每一条路径的上下游的详细情况，对于追查问题十分有用。

实现一个服务追踪系统，涉及数据采集、数据处理和数据展示这三个流程，有多种实现方式，具体采用哪一种要根据自己的业务情况来选择。关于服务追踪系统的选型我在专栏后面会详细展开介绍，这里你只需要了解它的基本工作原理就可以了。

思考题

通过这两期的学习，你应该了解到服务追踪系统和服务监控系统的搭建都需要数据采集、处理和展示这三个步骤，你认为它们是否有相同和不同之处呢？

欢迎你在留言区写下自己的思考，与我一起讨论。

精选留言

拉欧 0

两者的维度不同:服务追踪系统关心的是单次调用的性能，这其中可能跨越多个服务；服务监控系统关心的是单个服务的性能，主要包括服务质量，甚至机器性能等指标；
两者是互为补充的关系，服务监控系统可以及时发现服务内部出现的问题，但是所有服务运行正常，不代表跨服务调用一定正常，因为网络本身就是不可靠的，所以需要服务追踪系统发现服务之间可能出现的问题，这样对于系统的监控才算完备

2018-09-08

云中漫步 0

一直不太懂，微服务怎么把一个交易组装起来，怎么编排，调用哪些服务。希望能得到这方面的知识。^_^

2018-09-08

Liam 0

一般排查bug都是从整体到局部，分布式链路追踪就是从整体把握业务执行的的过程，定位问题的区域后再具体分析，监控系统会收集每个节点的数据，包括日志，性能，资源，异常等，根据这些数据进一步定位问题的原因

2018-09-08

铂金小猪 0

链路不是http呢？

2018-09-08

作者回复

也可以的，就是埋点采集的代码不同

2018-09-08

Realm

0

1 监控是若干个局部，单独采集、分析、展示；追踪是全局视角，有链的上下游传递的概念，通过某个id串联相关的监控；

2 诊断故障一般从链上分析出现问题的点，然后定位到点上的监控数据，看具体原因；

2018-09-08

作者回复

嗯

2018-09-08

柏林

0

一楼沙发

2018-09-08