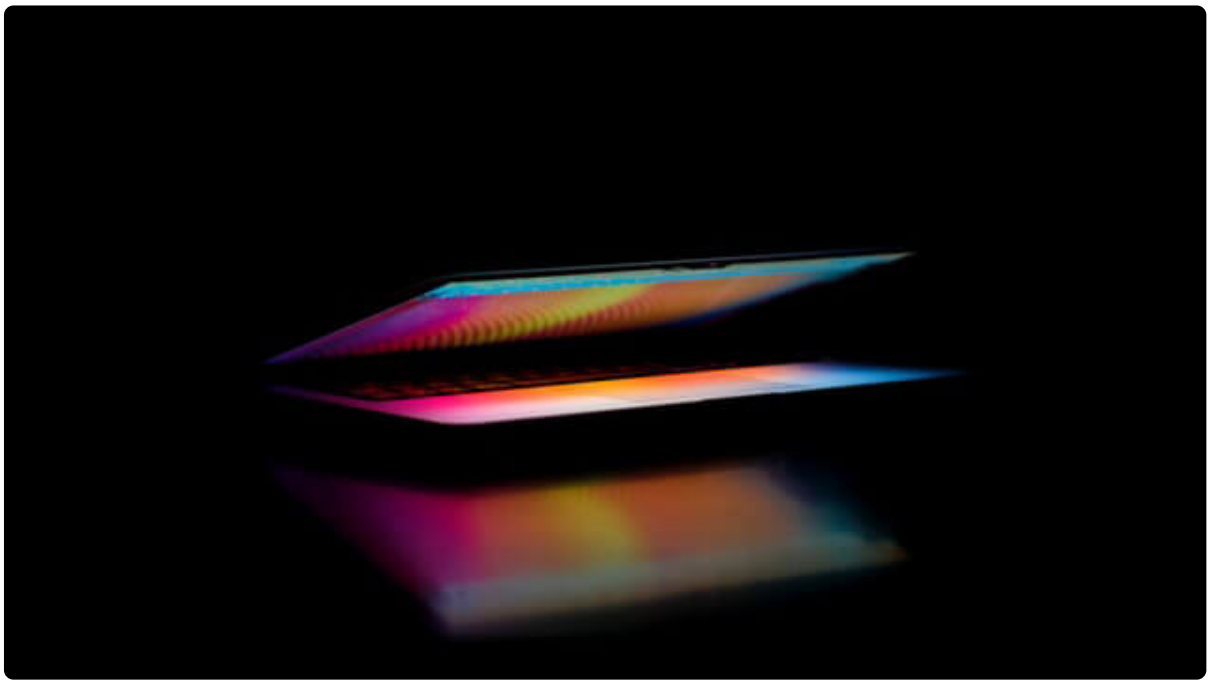
29 分布式链路跟踪和 Spring Cloud Sleuth

更新时间：2019-07-26 10:47:54



> 耐心和恒心总会得到报酬的。
>
> ——爱因斯坦

当代互联网服务，通常都是用复杂的、大规模分布式集群来实现的。互联网应用构建在不同的软件模块集上，而这些软件模块，可能是由不同的团队开发、使用不同的编程语言来实现，可能分布在了几千台服务器上、横跨多个不同的数据中心。因此，就需要一些可以帮助理解系统行为、用于分析性能问题的工具。

Spring Cloud Sleuth 的诞生便是为了帮助解决此类问题。在学习 Spring Cloud Sleuth 之前，我们需要先了解一下什么是分布式链路跟踪，为什么我们需要分布式链路跟踪，以及它解决了什么样的问题。

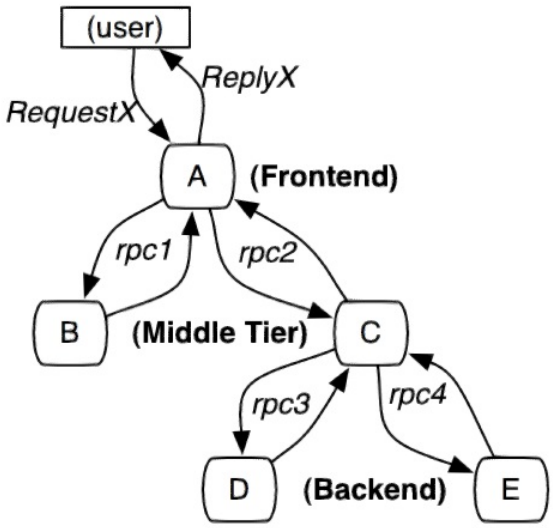
在微服务架构中，服务被切割成了很多的微服务，这些微服务相互之间通过 Http 的方式交互。在一个小型的微服务架构中，一次请求会涉及十几次不同项目之间的调用；在一个中型的互联网公司，一次请求平均会几百次的调用请求。因此在出现问题时，我们可能需要了解这些问题:

- 如何快速发现问题？
- 一次请求都调用了哪些服务？
- 为什么请求这么慢，到底是哪里出现了问题？
- 请求调用失败时，究竟是哪个服务调用失败了？

如果按照传统的方式根据日志来跟踪，那么当项目出现问题时，可能需要运维查询上百台甚至上万台服务日志来定位，仅日志收集一项就会产生巨大的工作量。即便使用 ELK 套件解决日志的收集问题，将这些日志进行关联、定位也将是一个巨大的工作量。有更好的解决方案吗？这就涉及到了分布式链路跟踪的概念。

分布式链路跟踪

现今业界分布式链路跟踪的理论基础主要来自于 Google 的一篇论文《Dapper, a Large-Scale Distributed Systems Tracing Infrastructure》。我们先根据一张图来了解一下一次请求的调用。



> 图来源于 Google Dapper

图中 A-E 分别表示五个服务，用户发起一次 X 请求到前端系统 A，然后 A 分别发送 RPC 请求到中间层 B 和 C，B 处理请求后返回，C 还要发起两个 RPC 请求到后端系统 D 和 E。

以上完整调用回路中，一次请求需要经过多个系统处理完成，并且追踪系统是持续跟踪到请求的每一步，也就是说分布式链路跟踪需要记录、跟踪一次请求的所有相关数据。在前端用户发起一次 X 请求的时候，就需要给这个请求生产一个唯一的 ID，在后面的所有请求调用中都需要带着这个 ID，最后根据这个 ID 将整个请求串联起来。

一个完成的分布式链路跟踪系统主要有三部分：数据收集、数据存储和数据展示。数据收集需要在调用的过程中，记录每一次请求的开始时间、结束时间、服务ID等其它相关数据；数据存储需要在短时间内快速存储大量的跟踪数据，并且需要满足快速检索的需求；数据展示，根据不同的维度以图形化的形式将收集的数据展示到页面，方便运营人员对问题进行分析、定位。

Spring Cloud Sleuth 属于分布式链路跟踪系统中数据收集的一个实现，它支持集成 Zipkin 等产品以图形化的方式展示分布式链路中收集的数据。

Spring Cloud Sleuth 介绍

Spring Cloud Sleuth 为 Spring Cloud 实现了分布式链路跟踪解决方案。Spring Cloud Sleuth 的实现过程也是充分吸收借鉴了 Google Dapper 的思想，并且沿用了一些 Google Dapper 术语。

Spring Cloud Sleuth 为服务之间调用提供了链路追踪，通过 Sleuth 可以很清楚的了解到一个服务请求经过了哪些服务，每个服务处理花费了多长时间，从而让我们可以很方便的理清各微服务间的调用关系。此外 Sleuth 还可以帮助我们：

- 耗时分析: 通过 Spring Cloud Sleuth 可以很方便的了解到每个采样请求的耗时，从而分析出哪些服务调用比较耗时；
- 可视化错误: 对于程序未捕捉的异常，可以通过集成 Zipkin 服务界面上看到；
- 链路优化: 对于调用比较频繁的服务，可以针对这些服务实施一些优化措施。

Sleuth 相关术语

Trace

服务追踪的追踪单元是从客户发起请求（request）抵达被追踪系统的边界开始，到被追踪系统向客户返回响应（response）为止的过程，称为一个“trace”。Trace 由一组 Span 形成树状结构，例如，如果运行分布式大数据存储，则可能由 PUT 请求形成 trace。

Span

每个 Trace 中会调用若干个服务，为了记录调用了哪些服务，以及每次调用的消耗时间等信息，在每次调用服务时，埋入一个调用记录，称为一个“span”。 Span 是 Sleuth 的基本工作单元，若干个有序的 Span 组成一个 trace。Span 由唯一的 64 位 ID 标识，还有另外一个 64 位 ID 标识其所属的 Trace。

Span 可以启动和停止，它们可以追踪自己的时间信息，创建 span 后，必须在将来的某个时刻停止它。

> 启动 Trace 的初始 span 称为 root span，该 span 的 ID 值等于 trace ID。

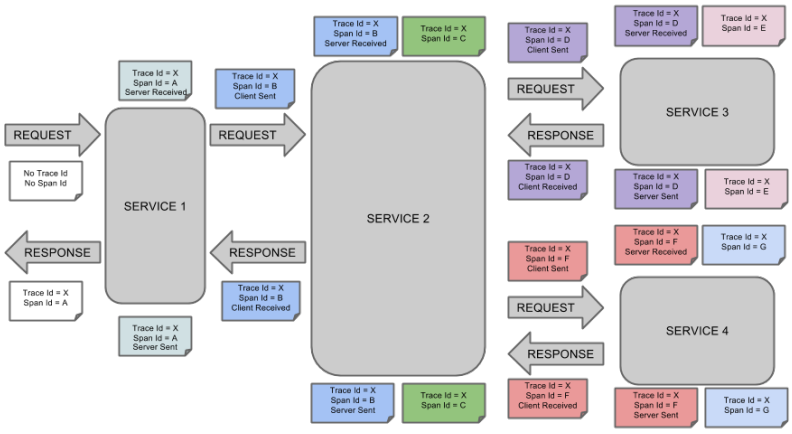
Annotation

Annotation 相当于 Span 记录的语法，描述 Span 现在所处的状态，它主要由四个概念：

- cs : Client Sent 客户端发送。表示一个 Span 的起始。
- sr : Server Received 服务端接收。表示服务端接收到请求，并开始处理。如果减去 cs 的时间戳，则表示网络传输时长。
- ss : Server Sent 服务端完成请求处理，应答信息被发回客户端。如果减去 sr 的时间戳，则表示服务端处理请求的时长。
- cr : Client Received 客户端接收。标志着 Span 的结束，客户端成功的接收到服务端的应答信息。如果减去 cs 的时间戳，则表示请求的响应时长。

记录过程

了解完这些概念之后，我们来看一下 Spring Cloud Sleuth 如何使用这些术语来完成一次 Trace 的记录。



> 图来源于 Spring Cloud Sleuth 官网。

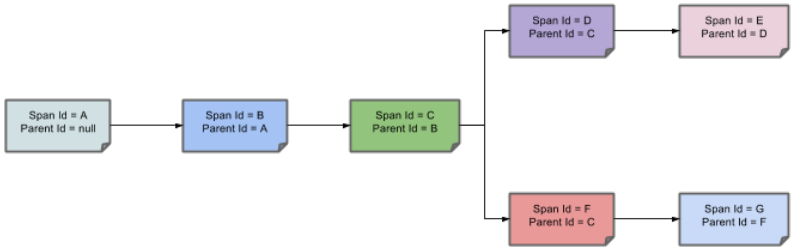
图中可以看出请求涉及到四个服务，每一次的请求和响应都会产生一个 Span 状态，在 Span 中会存储 Span id 和 Trace id 用来标记他们的所属关系，同时 Span 中会使用 Annotation 标记每一个 Span 当前的状态。通过以上信息的有序组合很直观的展示了一次请求（Trace）的调用过程。

图中标记的每种颜色表示一个 span（有七个 span — 从 A 到 G），Span 的格式如下：

```
Trace Id = X # 所属 Trace id
Span Id = D  # 自身 id
Client Sent  # 状态
```

此标记表示当前 Span 的 Trace Id 设置为 X，Span Id 设置为 D，此外，还发生了 Client Sent 事件。

Span 相互之间存在着父子关系，最开始的 Span 为初始 Span 没有父级，前面调用的 Span 是后面 Span 的父级。依此类推上图中七个 Span 的父子关系如下：



> 图来源于 Spring Cloud Sleuth 官网。

跟踪原理

当我们项目中引入 spring-cloud-starter-sleuth 包后，每次链路请求都会添加一串追踪信息，格式是 [server-name, main-traceId,sub-spanId,boolean] 。

- server-name：服务结点名称；
- main-traceId：一条链路唯一的 ID，为 TraceID；
- sub-spanId：链路中每一环的 ID，为 SpanID；
- boolean：是否将信息输出到 Zipkin 等服务收集和展示。

这种机制是如何实现的呢？我们知道 Spring Cloud 微服务是通过 Http 协议通信的，所以 Sleuth 的实现也是基于 Http 的，为了在数据的收集过程中不影响到正常业务，Sleuth 会在每个请求的 Header 上添加跟踪需求的重要信息，例如：

```
X-A1-TraceId：对应 TraceID；
X-A1-SpanId：对应 SpanID；
X-A1-ParentSpanId：前面一环的 SpanID；
X-A1-Sampled：是否被选中抽样输出；
X-Span-Name：工作单元名称。
```

这样在数据收集时，只需要将 Header 上的相关信息发送给对应的图像工具即可，图像工具根据上传的数据，按照 Span 对应的逻辑进行分析、展示。

小结

本节为大家介绍了分布式链路跟踪产生的背景，以及它解决了哪些问题。Spring Cloud Sleuth 是 Spring Cloud 体系内分布式跟踪解决方案的组件之一，使用 Spring Cloud Seuth 可以轻松收集微服务架构中，每个请求的调用数据。根据这些数据，我们能方便的解决服务性能优化、快速定位等问题。

参考出处：

https://storage.googleapis.com/pub-tools-public-publication-data/pdf/36356.pdf

https://www.ibm.com/developerworks/cn/web/wa-distributed-systems-request-tracing/index.html

28 Spring Cloud Stream 深入实践

30 Spring Cloud Sleuth 实践