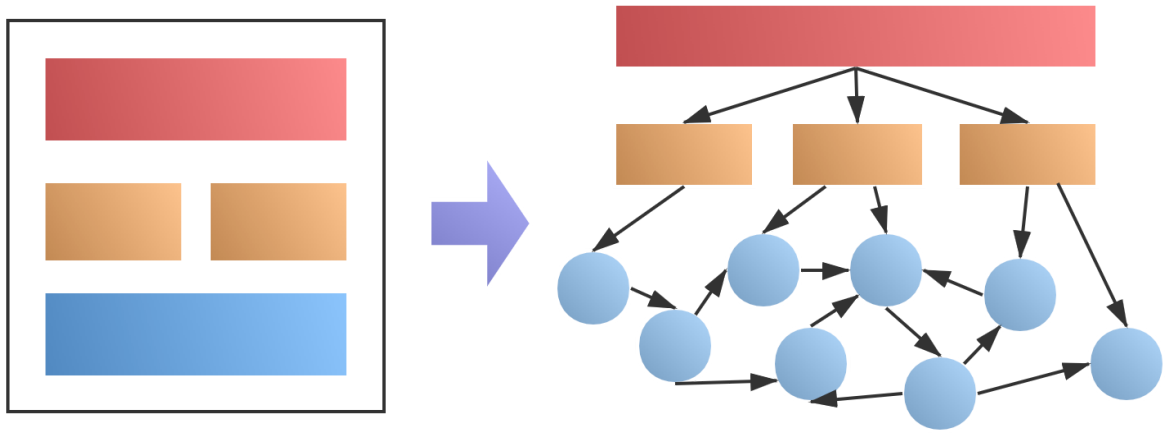
分布式服务拆分以后，系统变得日趋复杂，业务的调用链也越来越长，如何快速定位线上故障，就需要依赖分布式调用跟踪技术。下面我们一起来看下分布式调用链相关的实现。

为什么需要分布式调用跟踪

随着分布式服务架构的流行，特别是微服务等设计理念在系统中的应用，系统架构变得越来越分散，如下图所示。



集中式架构　　分布式微服务架构

可以看到，随着服务的拆分，系统的模块变得越来越多，不同的模块可能由不同的团队维护，一个请求可能会涉及几十个服务的协同处理， 牵扯到多个团队的业务系统。

假设现在某次服务调用失败，或者出现请求超时，需要定位具体是哪个服务引起的异常，哪个环节导致的超时，就需要去每个服务里查看日志，这样的处理效率是非常低的。

另外，系统拆分以后，缺乏一个自上而下全局的调用 ID，如何有效地进行相关的数据分析工作呢？比如电商的活动转化率、购买率、广告系统的点击链路等。如果没有一个统一的调用 ID 来记录，只依靠业务上的主键等是很难实现的，特别是对于一些大型网站系统，如淘宝、京东等，这些问题尤其突出。

分布式调用跟踪的业务场景

分布式调用跟踪技术就是解决上面的业务问题，即通过调用链的方式，把一次请求调用过程完整的串联起来，这样就实现了对请求调用路径的监控。

分布式调用链其实就是将一次分布式请求还原成**调用链路**，显式的在后端查看一次分布式请求的调用情况，比如各个节点上的耗时、请求具体打到了哪台机器上、每个服务节点的请求状态等。

一般来说，分布式调用跟踪可以应用在以下的场景中。

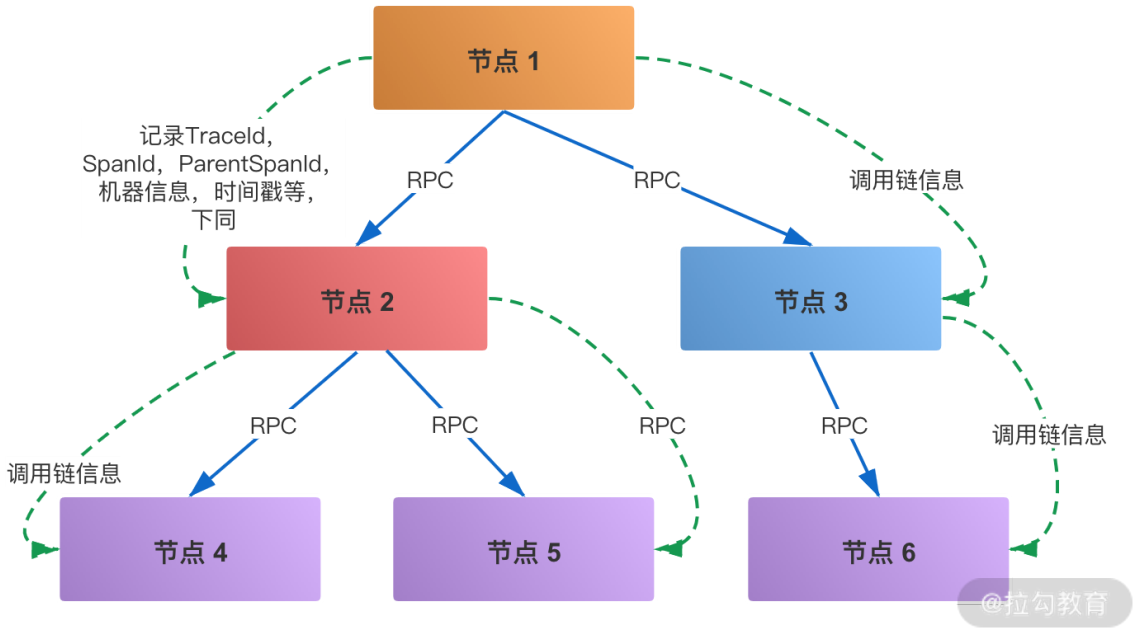
- **故障快速定位**：通过调用链跟踪，一次请求的逻辑轨迹可以完整清晰地展示出来。在开发的过程中，可以在业务日志中添加调用链 ID，还可以通过调用链结合业务日志快速定位错误信息。
- **各个调用环节的性能分析**：在调用链的各个环节分别添加调用时延，并分析系统的性能瓶颈，进行针对性的优化。
- **各个调用环节的可用性，持久层依赖等**：通过分析各个环节的平均时延、QPS 等信息，可以找到系统的薄弱环节，对一些模块做调整，比如数据冗余等。
- **数据分析等**：调用链是一条完整的业务日志，可以得到用户的行为路径，并汇总分析。

分布式调用跟踪实现原理

分布式链路跟踪的技术实现，主要是参考 Google 的 Dapper 论文，分布式调用跟踪是一种全链路日志，主要的设计基于 Span 日志格式，下面简单介绍这个日志结构。

Dapper 用 Span 来表示一个服务调用开始和结束的时间，也就是时间区间，并记录了 Span 的名称以及每个 Span 的 ID 和父 ID，如果一个 Span 没有父 ID 则被称之为 Root Span。

一个请求到达应用后所调用的所有服务，以及所有服务组成的调用链就像是一个树结构，追踪这个调用链路得到的树结构称之为 **Trace**，所有的 Span 都挂在一个特定的 Trace 上，共用一个 TraceId。



在一次 Trace 中，每个服务的每一次调用，就是一个 Span，每一个 Span 都有一个 ID 作为唯一标识。同样，每一次 Trace 都会生成一个 TraceId 在 Span 中作为追踪标识，另外再通过一个 parentSpanId，标明本次调用的发起者。

当 Span 有了上面三个标识后，就可以很清晰地将多个 Span 进行梳理串联，最终归纳出一条完整的跟踪链路。

确定了日志格式以后，接下来日志如何采集和解析，日志的采集和存储有许多开源的工具可以选择。一般来说，会使用离线 + 实时的方式去存储日志，主要是分布式日志采集的方式，典型的解决方案如 Flume 结合 Kafka 等 MQ，日志存储到 HBase 等存储中，接下来就可以根据需要进行相关的展示和分析。

分布式调用跟踪的选型

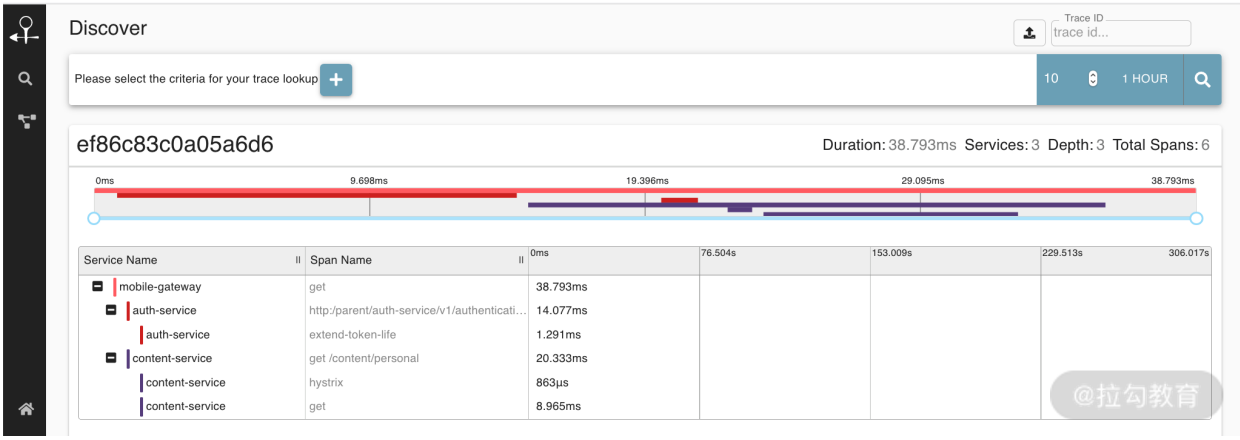
大的互联网公司都有自己的分布式跟踪系统，比如前面介绍的 Google 的 Dapper、Twitter 的 Zipkin、淘宝的鹰眼等。

Google 的 Drapper

Dapper 是 Google 生产环境下的分布式跟踪系统，没有对外开源，但是 Google 发表了“Dapper - a Large-Scale Distributed Systems Tracing Infrastructure”论文，介绍了他们的分布式系统跟踪技术，所以后来的 Zipkin 和鹰眼等都借鉴了 Dapper 的设计思想。

Twitter 的 Zipkin

Zipkin 是一款开源的分布式实时数据追踪系统，基于 Google Dapper 的论文设计而来，由 Twitter 公司开发贡献。其主要功能是聚集来自各个异构系统的实时监控数据，用来追踪微服务架构下的系统延时问题，Zipkin 的用户界面可以呈现一幅关联图表，以显示有多少被追踪的请求通过了每一层应用。



阿里的 EagleEye

EagleEye 鹰眼系统是 Google 的分布式调用跟踪系统 Dapper 在淘宝的实现，EagleEye 没有开源。下面这段介绍来自 阿里中间件团队：

> 前端请求到达服务器，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。埋点逻辑为这个前端请求分配一个全局唯一的调用链 ID，即 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。

> 当这个前端执行业务处理需要发起 RPC 调用时，RPC 调用客户端会首先从当前线程 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；之后，调用上下文会作为附件随这次请求一起发送到下游的服务器。

关于鹰眼的详细介绍，这里有一篇分享非常不错，即鹰眼下的淘宝：分布式调用跟踪系统。

总结

这一课时主要分享了分布式调用跟踪的应用场景、调用链的日志结构、分布式链路跟踪的选型实现等。

现在思考一下，了解了链路跟踪的日志格式，如果让你来设计一个调用跟踪系统，除了基本的链路跟踪功能，还需要满足哪些功能设计呢？

举个例子，在实际业务中，链路跟踪系统会有一个采样率配置，不会监控全部的链路，其实是考虑到对系统性能的影响。所以，作为非业务组件，应当尽可能少侵入或者无侵入其他业务系统，并且尽量少的占用系统资源。

精选评论

****伟:**

> 这里可以介绍一下skywalking....这个做的还挺好的

***坤:**

> Istio支持Jaeger zipkin kiali等 推荐一下