【剖析 | SOFARPC 框架】系列之链路追踪剖析

原创：SOFARPCLab 金融级分布式架构 昨天

> **SOFA**
> **S**calable **O**pen **F**inancial **A**rchitecture
> 是蚂蚁金服自主研发的金融级分布式中间件，包含了构建金融级云原生架构所需的各个组件，是在金融场景里锤炼出来的最佳实践。
>
> 本文为《剖析 | SOFARPC 框架》第二篇，本篇由畅为/碧远/卓与共同出品。
> 《剖析 | SOFARPC 框架》系列由 SOFA 团队和源码爱好者们出品，
> 项目代号：<SOFA:RPCLab/>，文章尾部有参与方式，欢迎同样对源码热情的你加入

一. 前言

微服务已经被广泛应用在工业界，微服务带来易于团队并行开发、独立部署、模块化管理等诸多优点。然而微服务将原单体拆分多个模块独立部署，各模块之间链接变得错综复杂，在大规模分布式系统中这种复杂链路给维护带来了诸多困难。 如果对整个微服务架构不能了然于胸，便很难理清各模块之间的调用关系。 例如修改一个服务接口，对哪些服务造成影响不能快速定位。

SOFARPC 在 5.4.0 以后提供了链路追踪技术，可以有效协助开发运营人员进行故障诊断、容量预估、性能瓶颈定位以及调用链路梳理。

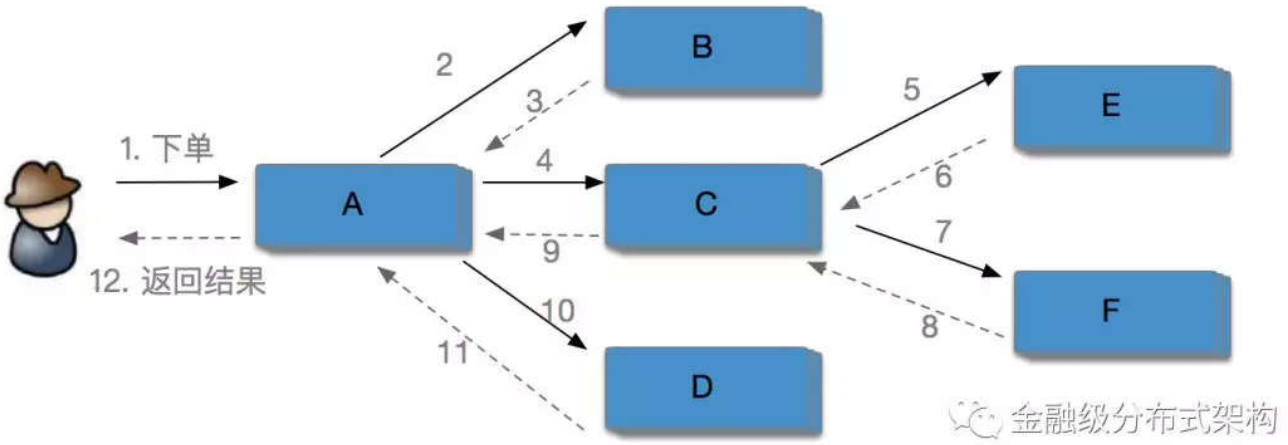
如思维导图所示，本文将从以下两个方面介绍目前已经开源的 SOFARPC 的链路追踪技术：

- 什么是链路追踪技术
- SOFARPC 链路追踪设计原理

二. 什么是链路追踪技术

链路追踪技术主要是收集、存储、分析分布式系统中的调用事件数据，协助开发运营人员进行故障诊断、容量预估、性能瓶颈定位以及调用链路梳理。 链路追踪技术包含了数据埋点、收集、存储、分析等相关技术，是一套技术体系。 大部分的链路追踪框架都是参考 Google 链路追踪系统 Dapper 的一篇设计论文（《Dapper, a Large-Scale Distributed Systems Tracing Infrastructure》），SOFARPC 的 SOFATracer 的设计灵感也是来自这篇著名论文。

以大规模分布式电商系统为例，用户下单购买某款产品时后端需要调用各系统或子模块进行协作，共同完成一个用户请求。 如下图所示，用户的下单行为涉及到了A、B、C、D、E、F 6 个系统的协同工作，这些系统都是以集群形式部署。整个链路中最长的链路调用是3层，如 A-> C -> E 或 A -> C -> F。



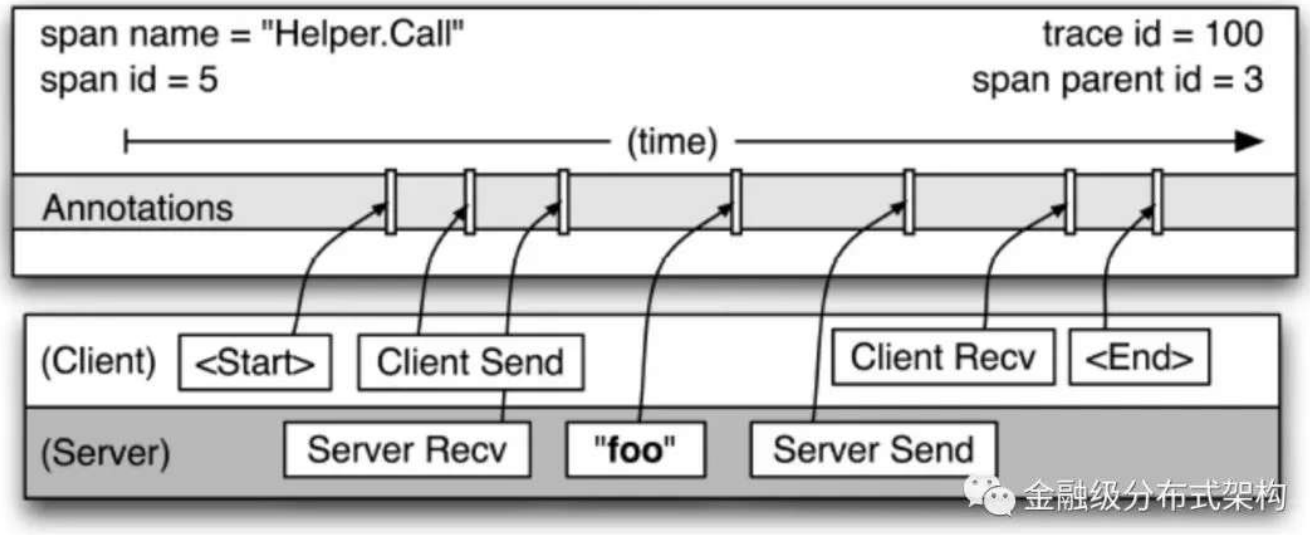
模块的增多加大了系统出错的概率，一旦因某系统/模块出错导致整个请求调用出错，在缺乏链路追踪的条件下很难定位具体出错的模块，只能通过日志搜索定位。 在实际生产环境下比较关注一个请求中的各个模块的耗时情况、连续调用情况、出错的节点等信息。

为了解决上述问题，Dapper 提供了一套解决方案。整个方案分为数据收集、存储和分析几个部分。分布式追踪技术会记录一个请求中各模块的调用信息；并通过一个处理集群把所有机器上的日志增量地收集到集群当中进行处理，将同一个请求的日志串联；最后可视化显示调用情况。

常用的数据收集方式为埋点，通过在公共组件如 RPC 等注入代码，收集服务调用的相关信息。目前大部分链路调用系统如 Dapper、鹰眼、Spring Cloud Sleuth 都在用这种模式。同样 SOFARPC 作为一个公共的通讯框架，在金融业务领域被广泛应用，因此适合作为埋点，这样无需业务端自行埋点，可做到对业务代码的无侵入性。

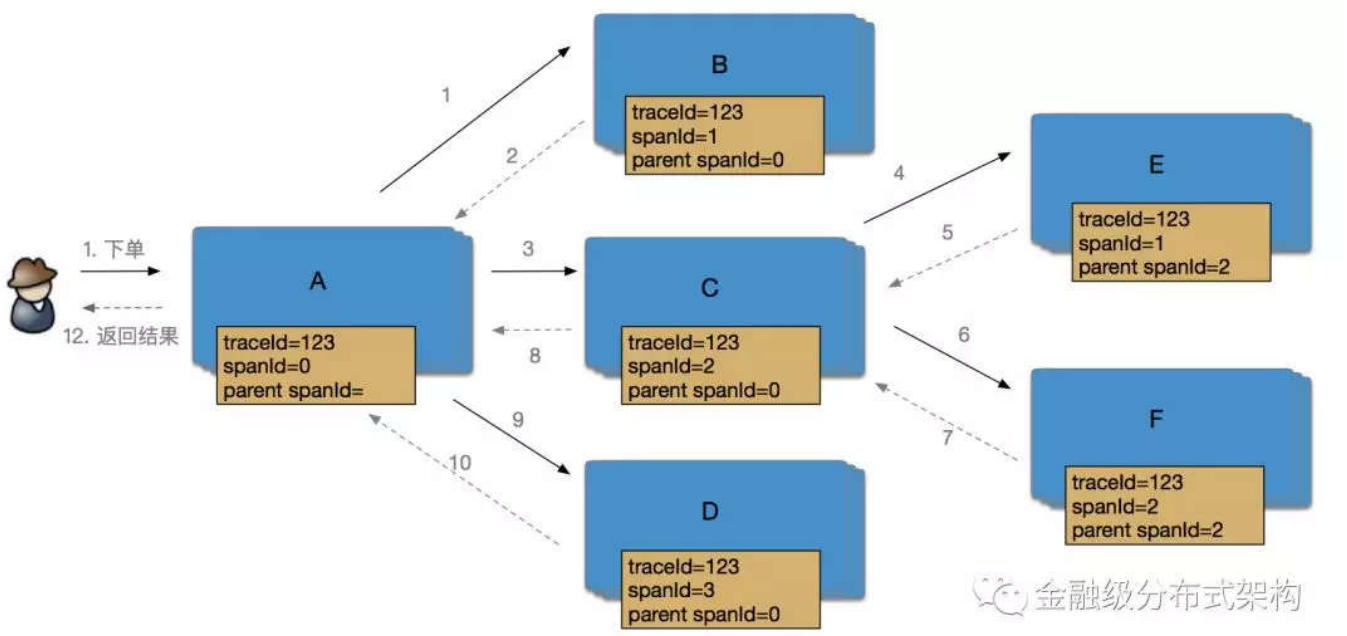
Dapper 将一个调用过程构建成一棵调用树(称为 Tracer)，Tracer 树中的每个节点表示链路调用中的一个模块或系统。 通过一个全局唯一的 traceId 来标识一个请求调用链。 并定义了span，span表示一个系统的调用，一个 span 包含以下阶段：

- Start：发起调用
- cleint send(cs) : 客户端发送请求
- Server Recv(sr) ： 服务端收到请求
- Server Send(ss) : 服务端发送响应
- Client Recv(cr) : 客户端收到服务端响应
- End： 整个链路完成



（图来自 Dapper 论文）

每个 span 包含两个重要的信息 span id(当前模块的span id) 和 span parent ID (上一个调用模块的 span id)，通过这两个信息可以定位一个span 在调用链的位置。 通过以上信息我们可以定义用户下单过程的调用链：



SOFARPC 中的链路追踪技术主要是作为埋点部分，因此对于链路追踪系统的收集和分析部分本文不做详述，想要深入了解的可参看参考资料内容。链路追踪可以提供我们以下功能:

1. 服务耗时、瓶颈分析 ：分析每个服务的耗时情况，可以针对耗时长的服务进行优化，提高服务性能。
2. 可视化错误：快速定位服务链路中出错的环境，便于排查和定位问题。一般链路追踪中间件都提供了 ZipKin 页面支持。
3. 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。
4. 调用链路梳理：通过可视化界面，对整个调用链路有个清晰的认识。

在设计分布式链路框架时需要考虑一下几个问题：

1. 低损耗、高性能: 追踪系统对在线服务的影响应该做到足够小，不影响线上服务性能。
2. 应用透明: 对于业务开发人员来说，应不需要知道有跟踪系统这回事的。
3. 可扩展性：虽则业务规则增大、集群增多，监控系统都应该能完全把控住这种快速变化。

4. 数据采样设计：如果每条日志都记录，在高并发情况下对系统有一定的损耗。但收集数据太少可能对统计结果有所影响，所以应合理设计采样比例。

三. SOFARPC 链路追踪设计原理

SOFARPC 作为一个基础的通讯中间件，对服务调用有很强的感知能力，容易获取链路追踪所需的服务调用信息。因此很多链路追踪系统都会选择RPC 作为埋点对象，通过对 RPC中间件的埋点可以轻松做到对用户的无感知、透明化。 SOFARPC 在 5.4.0 以后开始支持分布式链路追踪，其技术实现主要依赖于所集成的 SOFATracer。

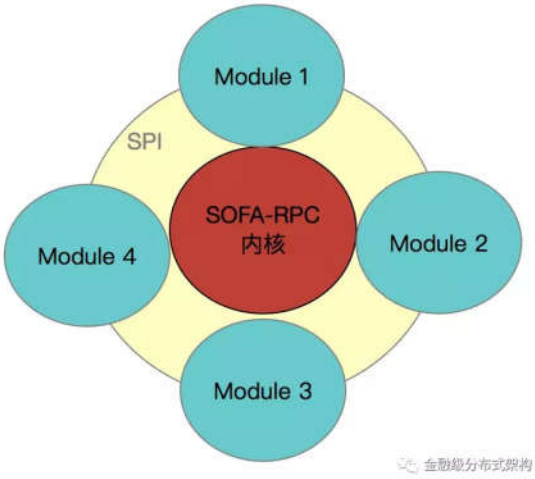
SOFARPC 不仅提供了埋点信息采集的能力, 还支持数据上报 zipkin。 通过 SOFARPC + SOFATracer + zipKin 可以快速搭建一套完整的链路追踪系统，包括埋点、收集、分析展示等。 收集和分析主要是借用 zipKin 的能力，本文重点讲 SOFARPC 中的数据埋点设计。SOFARPC 自身具备的微内核设计和可拓展性，使得在 SOFARPC 在不破坏开放封闭原则的前提下，比较轻松地集合 SOFATracer。

该部分主要从以下几个方面讨论 SOFARPC 的链路追踪设计思路：

1. 可插拔设计。 SOFARPC 采用了微内核设计，使得很容易扩展，增加新功能。
2. 总线设计。为数据埋点做提供一种无侵入的扩展方式。
3. 调用 trace 和 span
4. 数据采样设计
5. 异步刷新机制
6. 耗时计算：链路调用的耗时统计等信息获取。
7. 埋点数据透传，各模块之间的链路调用数据的透传机制。
8. 异步线程的链路调用。在异步多线程环境下如何保证 traceId 和 spanId 的有序性。
9. 链路调用日志数据的文件存储结构

3.1 可插拔设计

SOFARPC 自身具备的微内核设计和可拓展性，使得在 SOFARPC 在不破坏开放封闭原则的前提下，比较轻松地集合 SOFATracer。SOFARPC 采用了自己实现的一套 SPI 机制， 通过该 SPI 机制动态去加载其他模块、过滤器、协议等，实现灵活拓展。SOFARPC 为了集成 SOFATracer 也采用了这套机制，做到可插拔。

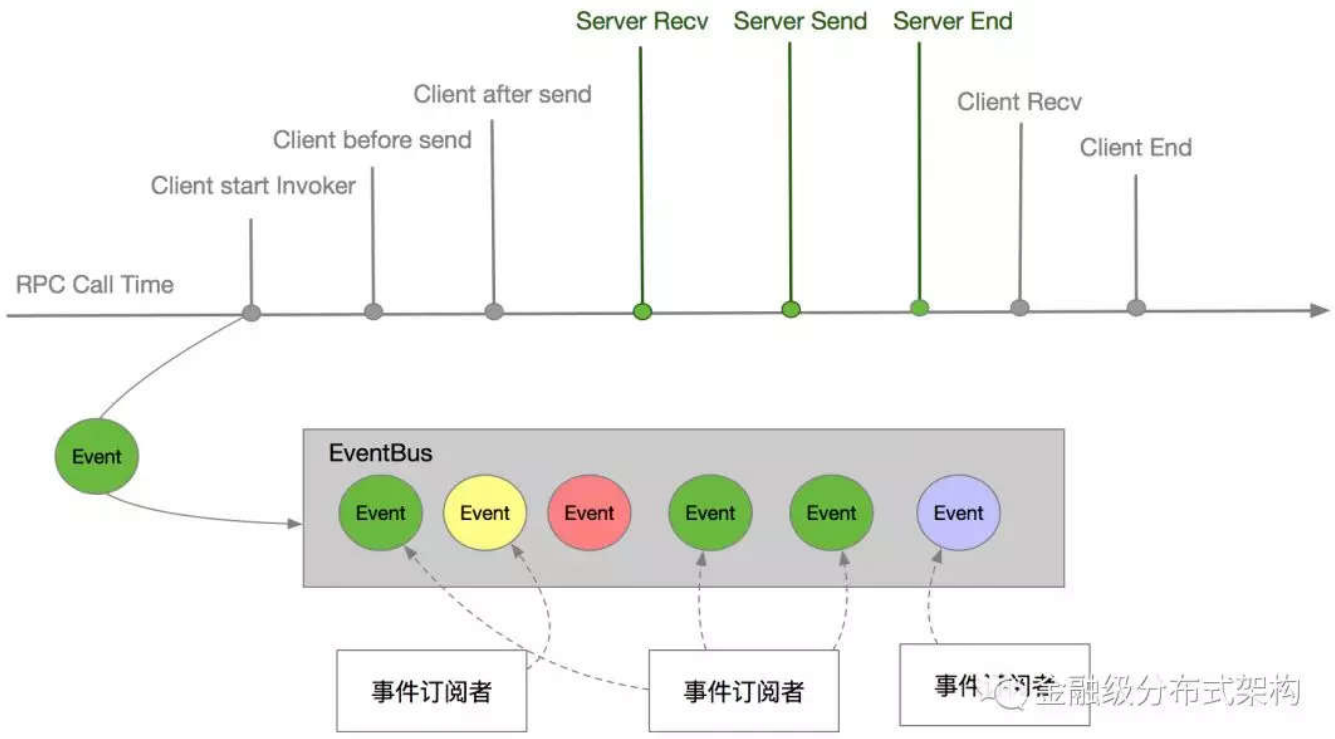


SofaTracerModule 类实现了 Module 接口，并增加 @Extension ("sofaTracer") 注解，方便SOFARPC 在启动时将相关模块加载进来。 SofaTracerModule 作为 SOFA-PRC 链路追踪的入口，在 SofaTracerModule 模块被加载时完成一些事件的订阅。

这里会订阅 9 种事件， 通过监听 SOFARPC 的这 9 种事件，来完成埋点数据的获取和异步磁盘写入操作。SOFARPC 通过事件总线设计来订阅这些事件，当事件发生时通知对应的订阅者做相应的操作。

3.2 事件总线设计

事件总线(EventBus)设计也是 SOFARPC 的一个具有很强扩展性的设计，EventBus 类似计算机数据总线，用于传输数据，EventBus 主要是传输事件数据。 EventBus 采用了发布-订阅设计模式，在SOFARPC 服务调用的整个过程中设置多个事件点，当这些事件发生时就创建事件写入到 EventBus，订阅者可以订阅总线中感兴趣的事件并处理。如图所示：



如上图所示，SOFATracer 订阅了在RPC调用周期的9种事件，当这些事件发生时会创建事件传入到 EventBus。 EventBus 中一旦发布新的事件就会通知所有感兴趣的订阅者，SAFA-Tracer 统一采用

SofaTracerSubscriber 订阅和处理这 9 种事件，最终链路追踪数据的获取操作都交给了 RpcSofaTracer 处理。

这种总线设计使得 SOFARPC 在集合 SOFATracer 时无需为了获取数据而破坏原来代码的封装性，使用无侵入的方式来完成埋点和数据的获取。

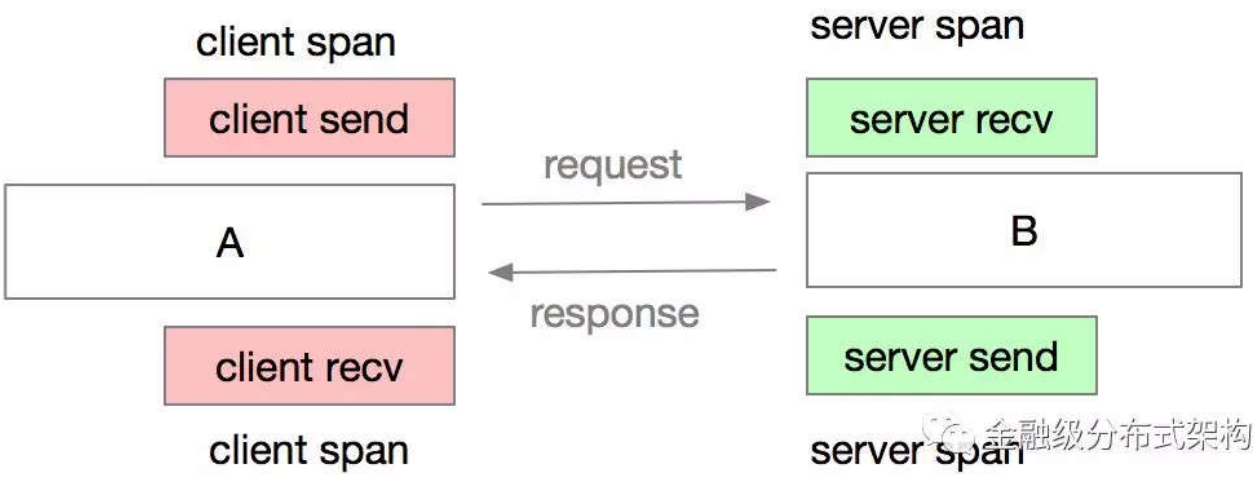
3.3 调用链 Trace 和 Span

SOFATracer 的设计思路也是来自 Dapper, 因此也提供了调用树 Trace 和 Span。

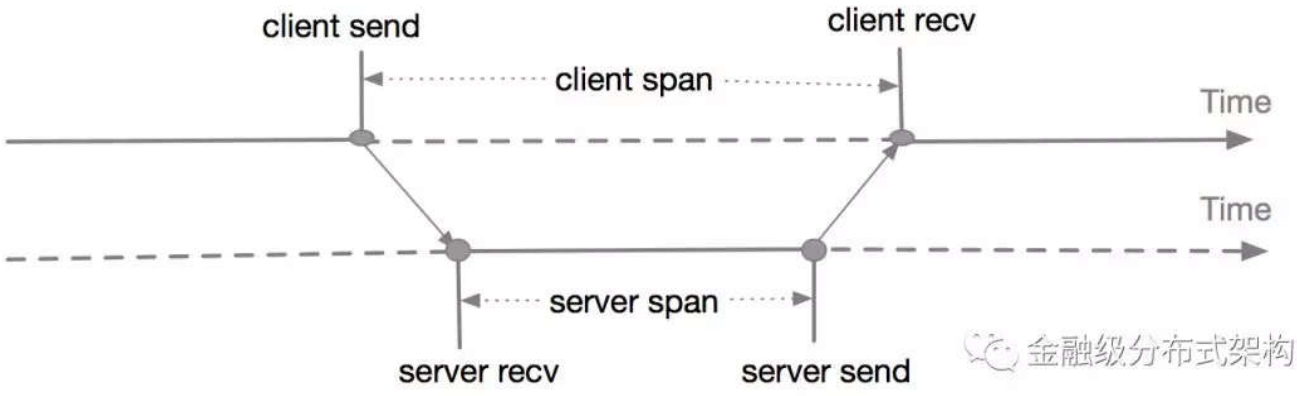
- Trace 是一次完整的跟踪，从请求到服务器开始，服务器返回 response 结束，跟踪每次rpc调用的耗时。 Trace 是一个类似树状调用链，树中的每个节点对应一个系统或服务节点。
- Span 是一个更小的单位，表示一个 RPC 调用过程。在 SOFARPC 中分为 ClientSpan 和 ServerSpan。 ClientSpan 记录从客户端发送请求给服务端，到接受到服务端响应结果的过程。ServerSpan 是服务端收到客户端时间 到 发送响应结果给客户端的这段过程。

一个 span 包含几种 Annotation:

1. cs (client send)
2. cr (client recv)
3. sr (server recv)
4. ss (server send)



如上图所示展示了两个系统调用中的 client span 和 server span 的关系， 一次 RPC 调用称为 span, 并产生一个 spanId, client span 的 spanId 和 server span的 spanId 是同一个，因为都在一个 RPC 调用中。 下图展示从时间的维度来解释这两者的关系：



3.3.1 TraceId 生成规则

TraceId 指的是 SOFATracer 中代表唯一一次请求的 ID，此 ID 一般由集群中第一个处理请求的系统产生，并在分布式调用下通过网络传递到下一个被请求系统。

traceId 应当保证全局唯一，如何做到全局唯一呢？TraceId 目前的生成的规则参考了淘宝的鹰眼组件：

```
服务器 IP + 产生 ID 时候的时间 + 自增序列 + 当前进程号
```

如下图所示：

8位	13位	4位	可变
IP地址	产生ID的时间	自增序列	进程pid
0ad1348f	1403169275002	1003	56636

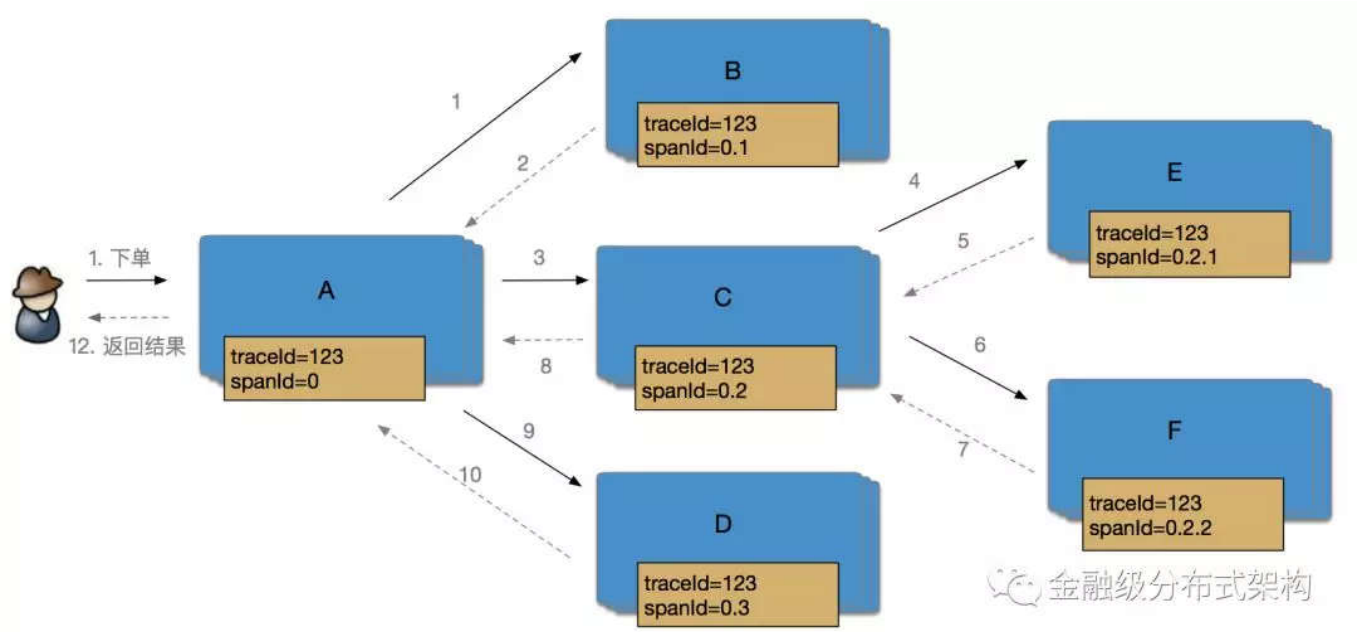
- 自增的序列，从 1000 涨到 9000，到达 9000 后回到 1000 再开始往上涨。

根据这种方式计算出的 traceId 为 0ad1348f1403169275002100356696，可以保证 traceId 全局唯一。

3.3.2 SpanId 生成规则

SpanId 表示整个链路中一次rpc调用的序号，也代表了本次请求在整个调用链路中的位置或者说层次，比如 A 系统在处理一个请求的过程中依次调用了 B，C，D 三个系统，那么这三次调用的的 SpanId 分别是：0.1，0.2，0.3。如果 C 系统继续调用了 E，F 两个系统，那么这两次调用的 SpanId 分别是：0.2.1，0.2.2。

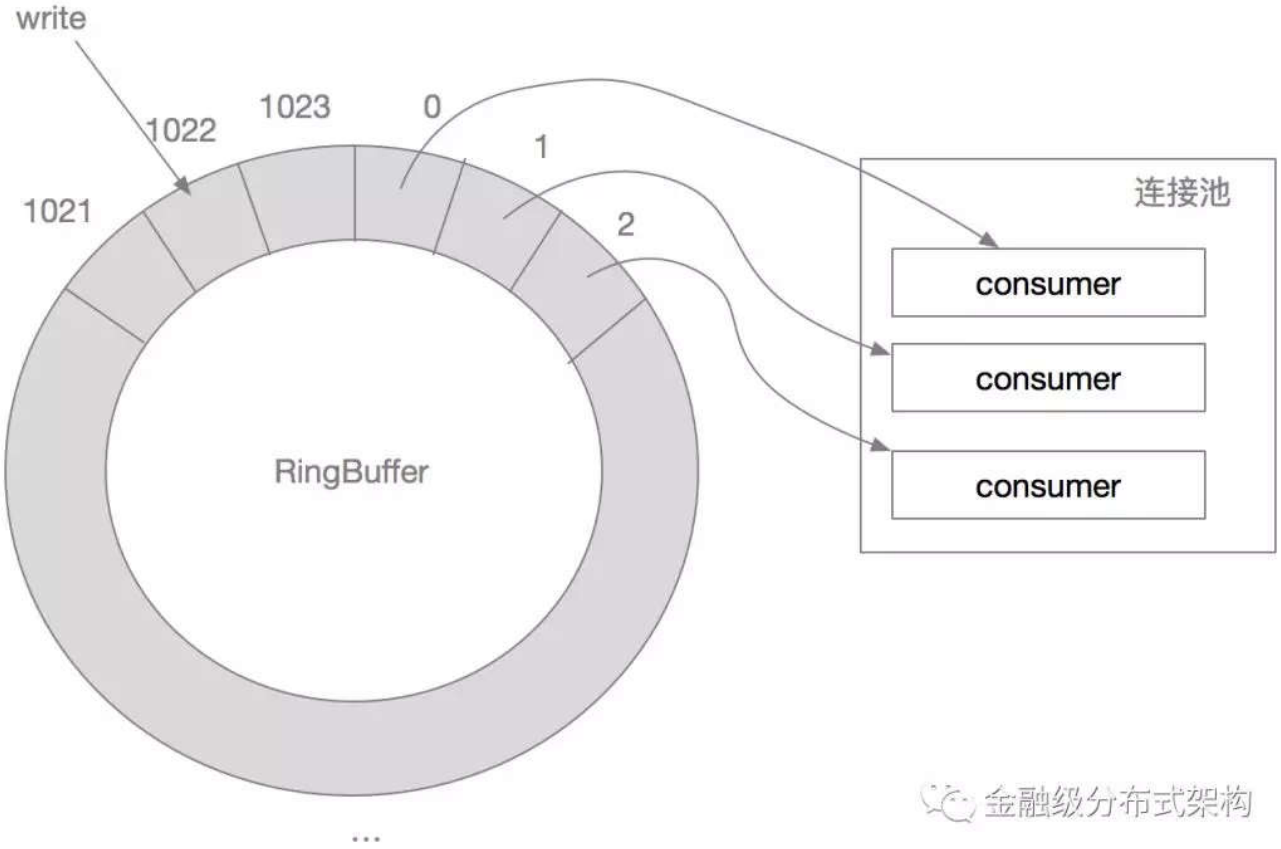
SOFARPC 和 Dapper不同，spanId中已经包含了调用链上下文关系，包含 parent spanId 的信息。如图所示：



3.4 数据采样设计

在 Dapper 论文中强调了数据采样的重要性，如果将每条埋点数据都刷新到磁盘上会增大链路追踪框架对原有业务性能的影响。如果采样率太低，可能会导致一些重要数据的丢失。 论文中提到如果在高并发情况下 1/1024 的采样率是足够的，也不必担心重要事件数据的丢失。因为在高并发环境下，一个异常数据出现一次，那么就会出现 1000 次。 然而在并发量不是很多的系统，并且对数据各位敏感时需要让业务开发人员手动设置采样率。

SOFATracer 的不支持配置 采样率，但采样率也不是一个固定写死的值，而是采用自适应采样率。 在 SOFATracer 中提供了 RingBuffer 的数据结构，设置一个 1024 的序列化槽位用于存储每个链路调用的埋点数据。 可以看做是一个圆形环状结构，RingBuffer 中的数据 从槽位 0 开始存储，一直存储到1023。当操作1023 时会从头开始存放，放在原来槽位的数据将被覆盖。



从上图所示，当数据写入的速度远远大于 3个 consumer 的处理速度，那么环上的数据在未被处理时就被覆盖。 通过覆盖的方式来自动调整采样率，并发性越高、写入速度越快时，采样率就越低。当并发性越低、写入速度也随之变慢，则采样率就变高。

在 SOFATracer 中默认开启三个线程去负责这些数据的持久化。假设每个线程的处理速度是 x/s(条/秒)，并发写入的速度是 y/s，那么 SOFATracer 的自动化采样率为 3x / y (前提是 y >= 3x, 否则就是 100% 采样率)。

3.5 异步刷新机制

埋点数据的本地化存储涉及到磁盘操作， 磁盘 IO 速度较慢，如果在高并发环境下同步刷新磁盘给原业务带来的性能损耗是非常可观的。 链路追踪系统在数据埋点的时候应尽可能的降低系统损耗，对原业务在逻辑和性能上做到无侵入性。

SOFATracer 采用了异步刷新机制，将 RingBuffer 的数据异步刷新到磁盘。 默认情况下 会启动 3个处理线程去处理 RingBuffer 的数据，将数据异步刷新到磁盘进行持久化。

当 RingBuffer 写入新数据时就会唤醒处理线程， 并将当前存入数据的槽位设置为可用槽位。 处理线程从睡眠点醒来后，便从原来处理位置往下获取数据并处理，直到所处理数据槽位大于可用槽位，则阻塞等待。

3.6 耗时计算

耗时计算不是为了集成 SOFATracer 而单独设置的，SOFARPC 框架自身带有耗时计算的逻辑，这些时间可以用于判断 RPC 调用是否超时等。因此加入链路追踪埋点时，不需要在扩展模块中计算耗时时间，SOFARPC中已经将调用的时间耗时等信息放在 RpcInternalContext 上下文中。 在计算 RPC 调用耗时时，对原有框架性能不影响，直接去上下文获取即可。

3.7 埋点数据透传

各模块部署后独立收集埋点日志，这些调用链日志通过 traceId 串联在一起。 在 SOFARPC中，下一个模块的 spanId 的创建依赖于上一个模块的 spanId。 因此这些埋点数据如 traceId 以及 spanId 需要透传给下游模块。

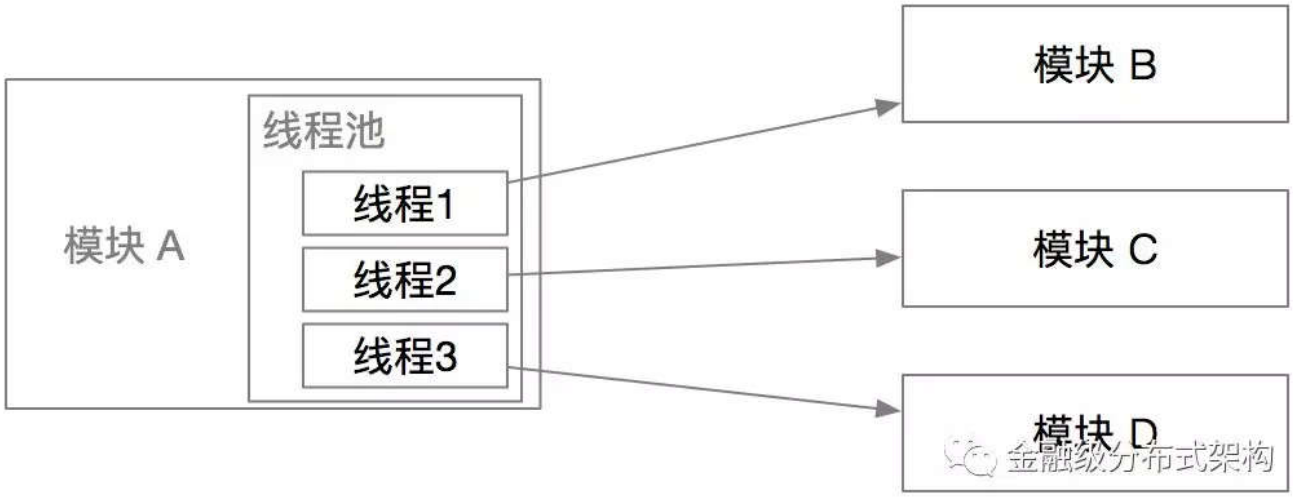
数据传输一般有两种：

1. 带内透传，即在原来的 rpc 调用请求网络宽带中加入埋点数据透传给下游；
2. 带外传播，通过单独提供一个宽带来传播，不影响原调用数据和网络。

Dapper 采用带外传播，这种方式可以不影响原有业务性能。带内透传数据意味着需要增加原来网络调用的负载。SOFARPC 采用的是带内透传，直接在原来的 RPCRequest 的扩展字段中加入埋点数据，直接透传给下游。SOFARPC 的 spanId 长度相对较短，所需传递的数据相对较小，从整体上看对原业务性能影响较小。

3.8 异步线程的链路调动

在多线程并发调用环境下的数据链路埋点也是一个值得关注问题，当一个服务考虑性能问题可能会起多个线程同时调用其他不同的模块。链路系统如何保证这些调用还是符合 openTrace 规范，保证 traceId 和 spanId 有序。



一个链路调用在模块 A 是一个线程，链路调用的上下文信息如 traceId、spanId 等都是存放在 ThreadLocal。 按上图思路新起的线程 1、2、3 无法获取主线程的 ThreadLocal 数据，即无法获取调用

链路数据。那么在无法获取链路调用的上线文数据时进行模块 B、C、D 的调用操作会导致收集得到的埋点数据是乱序的脏数据。

为了避免启动新线程把 链路调用的上下文 信息丢失，SOFATracer 提供了SofaTracerCallable 类，只要使用该类来实现线程逻辑，SOFATracer 会自动将链路调用的上下文信息透传给 SofaTracerCallable，因此可以像单线程一样进行调用埋点。SOFATracer 将上下文中的一些字段设置为线程安全，同样保证了多线程环境下的数据安全问题。 因此建议在多线程环境下进行一步调用时尽可能考虑使用 SofaTracerCallable， 否则调用链数据与预期有些出路。

3.9 文件存储结构

SOFA 整体开源框架对日志做了很好地分类，将不同类型的日志存放在不同的文件夹下。一方面便于收集特定日志，如埋点数据；另一方面也便于查找问题方便，日志结构和内容清晰。

在 SOFARPC 的链路追踪技术中，埋点数据的存储也采用日志文件方式进行持久化存储。tracer 日志文件包含以下文件：

文件	功能
rpc-client-digest.log	记录client rpc 调用的链路调用数据
rpc-client-stat.log	记录 client rpc 链路调用的统计数据
rpc-server-digest.log	记录 server rpc 调用的链路调用数据
rpc-server-stat.log	记录 server rpc 链路调用的统计数据
static-info.log	统计信息日志
tracer-self.log	tracer 自身的日志记录

四. 总结

SOFARPC 的依靠集成 SOFATrace 来实现链路追踪技术，SOFARPC 作为公共组件在整个链路追踪技术系统中负责数据埋点工作。依赖 SOFARPC 自身强大的可扩展性设计，如微内核设计和事件总线设计，使得 SOFARPC 在不破坏开发封闭原则的基础上快速实现了链路追踪埋点工作。 SOFARPC 的链路追踪技术具有以下特点：

1. 作为公共基础的通讯组件，SOFARPC 的链路追踪埋点对业务代码实现零侵入。
2. 采用日志数据异步刷新机制，不影响正常业务性能。
3. 采用了自适应采样设计，巧妙平衡了数据采集和性能的问题。
4. 支持数据上报 zipkin, 通过与 zipkin 结合可以快速构建一个完整的连续追踪系统。
5. 解决了异步线程链路调用数据问题。
6. 采用了 OpenTracing 规范，因此可以和其他链路追踪手机和展示的技术框架快速整合。

五. 参考资料

- [1] 《Dapper, a Large-Scale Distributed Systems Tracing Infrastructure》
- [2] 全链路稳定性背后的数字化支撑：阿里巴巴鹰眼技术解密

欢迎加入 <SOFA:RPCLab/>，参与 SOFARPC 源码解析

SOFARPC 源码解析目录：

我们会逐步详细介绍每部分的代码设计和实现，预计会按照如下的目录进行，以下也包含目前的源码分析文章的认领情况：

- **【已完成】SOFARPC 框架之总体设计与扩展机制**
- **【已完成】SOFARPC 链路追踪剖析**
- **【已完成】SOFARPC 同步异步实现剖析**
- 【已认领】SOFARPC 线程模型剖析
- 【已认领】SOFARPC 连接管理与心跳剖析
- 【已认领】SOFARPC 单机故障剔除剖析
- 【待认领】SOFARPC 路由实现剖析
- 【待认领】SOFARPC 序列化比较
- 【待认领】SOFARPC 注解支持剖析

- 【待认领】SOFARPC 优雅关闭剖析
- 【待认领】SOFARPC 数据透传剖析
- 【待认领】SOFARPC 跨语言支持剖析
- 【待认领】SOFARPC 泛化调用实现剖析

领取方式：

直接回复本公众号想认领的文章名称，我们将会主动联系你，确认资质后，即可加入**<SOFA:RPCLab/>，It's your show time!**

除了源码解析，也欢迎提交 issue 和 PR：

SOFA: https://github.com/alipay

SOFARPC：https://github.com/alipay/sofa-rpc

SOFABolt：https://github.com/alipay/sofa-bolt

SOFATracer：https://github.com/alipay/sofa-tracer

长按关注，获取分布式架构干货

欢迎大家共同打造 SOFAStack https://github.com/alipay