第一节 为什么我们需要Service Mesh

纲要

- 云原生时代微服务的挑战
- 解决方案 -- 初级
- 解决方案 -- 中级
- 解决方案 -- 高级
- Service Mesh是什么
- Service Mesh能做什么
- 业界Service Mesh产品
- 我们需要Service Mesh吗

云原生时代微服务的挑战

随着近年来云计算技术的快速发展，软件开发也从传统的单体应用到SOA以及时下流行的微服务，均随着技术的演变发生巨大的变化, 无论是对开发人员还是运维人员的技术理念和思维都要求极大的转变。尤其是在云原生时代，微服务已经成为业界开发应用的主要方式，而一些云计算技术的出现如Docker使得开发和发布微服务更加容易。但是微服务架构并不是万能银弹，虽然一方面可收之桑榆，但另外也可能失之东隅。其中最具挑战性的是如何确保分布在复杂网络环境中微服务处理网络弹性逻辑及可靠地交付应用请求，正如L Peter Deutsch在分布式系统的谬误中论述，我们不能一厢情愿地认为：

- 网络是可靠的
- 网络零延迟
- 网络带宽是无限的
- 网络是安全的
- 网络拓扑一成不变
- 系统只有一个管理员
- 传输代价为零
- 网络是同质的

因此应用应当具有规避网络不可靠、丢包、延时等的能力。那么从开发人员和运维人员来说，怎么才能确保分布在复杂网络环境中的微服务具有处理网络弹性逻辑能力及可靠地交付请求呢？ 一些常用技术手段如：

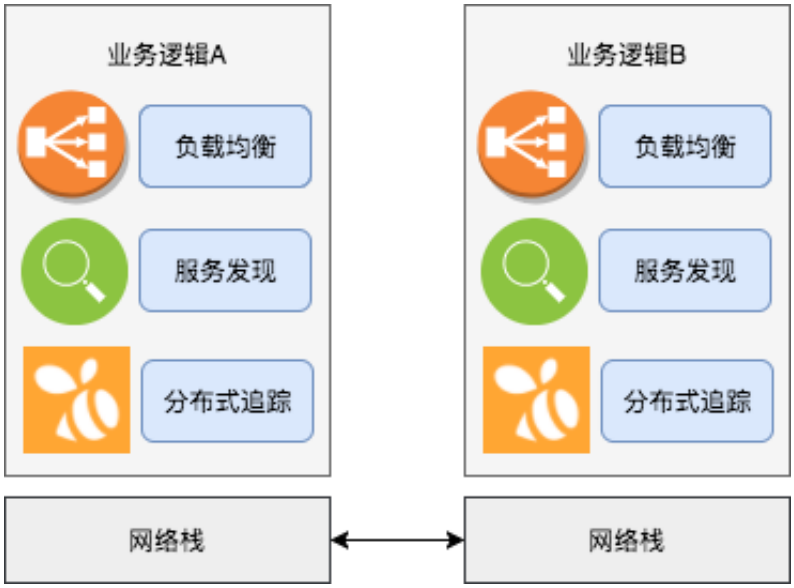
- 负载均衡
- 服务发现
- 运行时动态路由
- 熔断机制
- 安全通讯
- 指标和分布式追踪

当然，在微服务架构中，多语言、多协议支持以及透明监控等问题也需要得到足够的重视。

下面我们看处理这些挑战和问题时技术方案是如何演进的:

解决方案 -- 初级

在这种方案中，通常我们把上述的一些技术手段如负载均衡、服务发现或分布式追踪等跟业务逻辑代码一起封装起来，使得应用具有处理网络弹性逻辑的能力。该模式如:

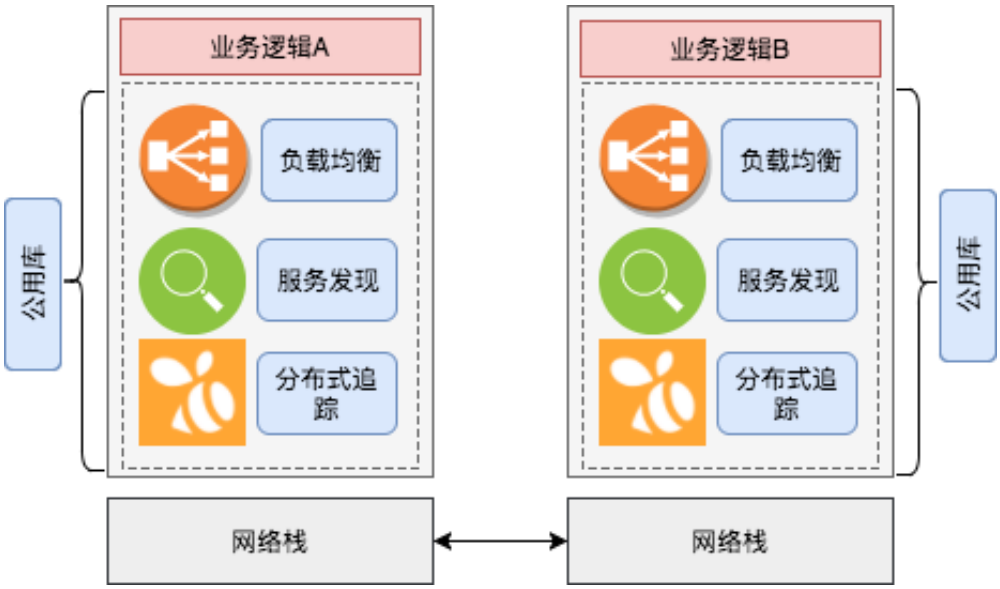


这种模式非常简单，但是从软件设计的角度，大家能很快发现它有很多缺点:

- 耦合性很高，每个应用都需封装负载均衡、服务发现、安全通讯以及分布式追踪等功能。
- 灵活性差，利用率低下，不同的应用需要重复的实现。
- 管理复杂，当其中一项如负载均衡逻辑发生变化，需要更新所有服务。
- 可运维性低，所有组件均封装在业务逻辑代码，不能作为一个独立运维对象。
- 对开发人员能力要求很高。

解决方案 -- 中级

关于第一种方案，虽然应用具有处理网络弹性逻辑能力，增强动态运行环境中如何处理服务发现、负载均衡等，向提供高可用、高稳定性、高SLA应用更进一步，与此同时，你也看到这种模式具有很多缺点。为此，我们是否可以考虑将由应用处理服务发现、负载均衡、分布式追踪、安全通讯等设计为一个公用库呢？这样使得应用与这些功能具有更低的耦合性，而且更加灵活、提高利用率及运维性，更主要的是开发人员只需要关注公用库有，而不是自己实现，从而降低开发人员的负担。这方面很多公司如Twitter、Facebook等走在业界前列，像Twitter提供给JVM的可扩展RPC库Finagle和Facebook的C++ HTTP框架Proxygen，Netflix的各种开发套件，还有如分布式追踪系统Zipkin, 这些库和开发套件的出现大量减少重复实现的工作。对于这种模式:

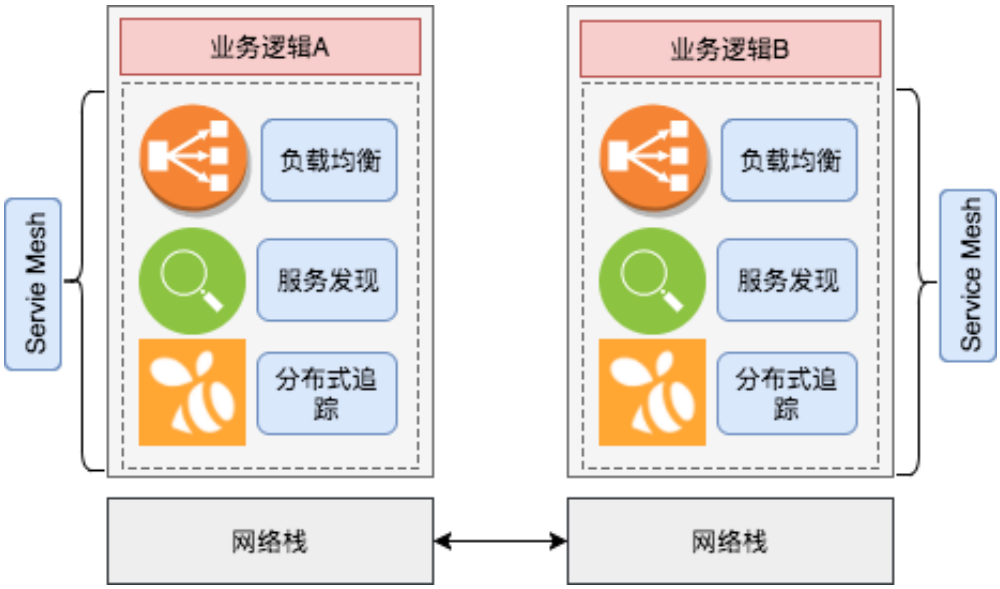


虽然相对前一种解决方案，新的方案在耦合性、灵活性、利用率等方面有很大的提升，但是仍然有些不足之处:

- 如果将类似Finagle, Proxygen或者Zipkin的库集成现有的系统中，仍然需要花费的大量的时间、人力和物力将其集成到现有生态圈，甚至需要调整现有应用的代码。
- 缺乏多语言支持，由于这些库只针对某种语言或者少数几种语言，这使得在一个多技术栈的公司需要限制开发语言和工具的选择。
- 虽然公共库作为一个独立的整体，但在管理复杂性和运维性这些方面仍然有更大的提升空间。
- 公共库并不能完全使得开发人员只关注业务代码逻辑，仍然需要对公共库很深的认识。

解决方案 -- 高级

显然，没有任何方案能一劳永逸的解决我们所有的问题，而各种问题驱使我们不断地向前演进、发展，探索新的解决方法。那么，针对我们现在所面临的问题，我们有更好的解决方案吗？答案当然有。相信大家对OSI七层模型应该不陌生，OSI定义了开放系统的层次结构、层次之间的相互关系以及各层所包括的可能的任务, 上层并不需要对底层具体功能有详细的了解，只需按照定义的准则协调工作即可，因此，我们也可参照OSI七层模型将公用库设计为位于网络栈和应用业务逻辑之间的独立层，即：透明网络代理，新的独立层完全从业务逻辑中抽离，作为独立的运行单元，与业务不再直接紧密关联。通过在独立层的透明网络代理上实现负载均衡、服务发现、熔断、运行时动态路由等功能，这样透明代理现在在业界有一个更加新颖时髦的名字: Service Mesh。率先使用这个Buzzword的产品恐怕非Buoyant的linkerd莫属了, 随后Lyft也发布了他们的Service Mesh实现Envoy，再有Istio也迎面赶上。新的模式如：



在这种方案中，由于Service Mesh作为独立运行层，它很好的解决了上述所面临的挑战，使应用具备处理网络弹性逻辑和提供可靠交互请求的能力。它使得耦合性更低、灵活性更强，跟现有环境的集成时间和人力代价更小，也提供多语言支持、多协议支持，运维和管理成本更低。最主要的是开发人员只需关注业务代码逻辑，而不是业务代码以外的其他功能，即Service Mesh对开发人员是透明的。下面我们将讲述什么是Service Mesh:

Service Mesh是什么

关于Service Mesh, William Morgan如是说:

> A service mesh is a dedicated infrastructure layer for handling service-to-service communication. It's responsible for the reliable delivery of requests through the complex topology of services that comprise a modern, cloud native application. In practice, the service mesh is typically implemented as an array of lightweight network proxies that are deployed alongside application code, without the application needing to be aware.

对此，可总结Service Mesh为:

- 专用基础设施层: 独立的运行单元。
- 包括数据层和控制层: 数据层负责交付应用请求，控制层控制服务如何通讯。
- 轻量级透明代理: 实现形式为轻量级网络代理。
- 处理服务间通讯: 主要目的是实现复杂网络中服务间通讯。
- 可靠地交付服务请求: 提供网络弹性机制，确保可靠交付请求。
- 与服务部署一起，但服务无需感知: 尽管跟应用部署在一起，但对应用是透明的。

Service Mesh能做什么

Service Mesh作为透明代理，它可以运行在任何基础设施环境，而且跟应用非常靠近，那么，Service Mesh能做什么呢?

- 负载均衡: 运行环境中微服务实例通常处于动态变化状态，而且经常可能出现个别实例不能正常提供服务、处理能力减弱、卡顿等现象。但由于所有请求对Service Mesh来说是可见的，因此可以通过提供高级负载均衡算法来实现更加智能、高效的流量分发，降低延时，提高可靠性。
- 服务发现: 以微服务模式运行的应用变更非常频繁，应用实例的频繁增加减少带来的问题是如何精确地发现新增实例以及避免将请求发送给已不存在的实例变得更加复杂。Service Mesh可以提供简单、统一、平台无关的多种服务发现机制，如基于DNS, K/V键值对存储的服务发现机制。
- 熔断: 动态的环境中服务实例中断或者不健康导致服务中断可能会经常发生，这就要求应用或者其他工具具有快速监测并从负载均衡池中移除不提供服务实例的能力，这种能力也称熔断，以此使得应用无需消耗更多不必要的资源不断地尝试，而是快速失败或者降级，甚至这样可避免一些潜在的关联性错误。而Service Mesh可以很容易实现基于请求和连接级别的熔断机制。
- 动态路由: 随着服务提供商以提供高稳定性、高可用性以及高SLA服务为主要目标，为了实现所述目标，出现各种应用部署策略尽可能从技术手段达到无服务中断部署，以此避免变更导致服务的中断和稳定性降低，例如: Blue/Green部署、Canary部署，但是实现这些高级部署策略通常非常困难。关于应用部署策略，可参考Etienne Tremel的文章，他对各种部署策略做了详细的比较。而如果运维人员可以轻松的将应用流量从staging环境到产线环境，一个版本到另外一个版本，更或者从一个数据中心到另外一个数据中心进行动态切换，甚至可以通过一个中心控制层控制多少比例的流量被切换。那么Service Mesh提供的动态路由机制和特定的部署策略如Blue/Green部署结合起来，实现上述目标更加容易。
- 安全通讯: 无论何时，安全在整个公司、业务系统中都有着举足轻重的位置，也是非常难以实现和控制的部分。而微服务环境中，不同的服务实例间通讯变得更加复杂，那么如何保证这些通讯是在安全、授权情况下进行非常重要。通过将安全机制如TLS加解密和授权实现在Service Mesh上，不仅可以避免在不同应用的重复实现，而且很容易在整个基础设施层更新安全机制，甚至无需对应用做任何操作。
- 多语言支持: 由于Service Mesh作为独立运行的透明代理，很容易支持多语言。
- 多协议支持: 同多语言支持一样，实现多协议支持也非常容易。
- 指标和分布式追踪: Service Mesh对整个基础设施层的可见性使得它不仅可以暴露单个服务的运行指标，而且可以暴露整个集群的运行指标。
- 重试和最后期限: Service Mesh的重试功能避免将其嵌入到业务代码，同时最后期限使得应用允许一个请求的最长生命周期，而不是无休止的重试。

业界Service Mesh产品

当前，业界主要有以下相关产品:

- Buoyant的linkerd，基于Twitter的Fingle，长期的实际产线运行经验及验证，支持Kubernetes, DC/OS容器管理平台，CNCF官方支持的项目之一。
- Lyft的Envoy，7层代理及通信总线，支持7层HTTP路由、TLS、gRPC、服务发现以及健康监测等，也是CNCF官方支持项目之一。
- IBM、Google、Lyft支持的Istio, 一个开源的微服务连接、管理平台以及给微服务提供安全管理，支持Kubernetes、Mesos等容器管理工具，其底层依赖于Envoy。

我们需要Service Mesh吗

前面我们已经讲述了Service Mesh带来的各种好处，解决各种问题，作为下一代微服务的风口，Service Mesh可以使得快速转向微服务或者云原生应用, 以一种自然的机制扩展应用负载，解决分布式系统不可避免的部分失败，捕捉分布式系统动态变化, 完全解耦于应用等等。我相信Service Mesh在微服务或者云原生应用领域一定别有一番天地，下一节我们开始讲述Service Mesh的一个实现: linkerd。

参考资源

1. Fallacies of distributed computing
2. Pattern: Service Mesh
3. Fingle
4. Proxygen
5. Zipkin
6. OSI七层模型
7. linkerd
8. Envoy
9. Istio
10. What's a service mesh? And why do I need one?
11. Six Strategies for Application Deployment