

二

34 深入理解微服务架构：银弹 or 焦油坑？

微服务是近几年非常火热的架构设计理念，大部分人认为是 Martin Fowler 提出了微服务概念，但事实上微服务概念的历史要早得多，也不是 Martin Fowler 创造出来的，Martin 只是将微服务进行了系统的阐述（原文链接：<https://martinfowler.com/articles/microservices.html>）。不过不能否认 Martin 在推动微服务起到的作用，微服务能火，Martin 功不可没。

2005 年：Dr. Peter Rodgers 在 Web Services Edge 大会上提出了“Micro-Web-Services”的概念。

2011 年：一个软件架构工作组使用了“microservice”一词来描述一种架构模式。

2012 年：同样是这个架构工作组，正式确定用“microservice”来代表这种架构。

2012 年：ThoughtWorks 的 James Lewis 针对微服务概念在 QCon San Francisco 2012 发表了演讲。

2014 年：James Lewis 和 Martin Fowler 合写了关于微服务的一篇学术性的文章，详细阐述了微服务。

由于微服务的理念中也包含了“服务”的概念，而 SOA 中也有“服务”的概念，我们自然而然地会提出疑问：**微服务与 SOA 有什么关系？有什么区别？为何有了 SOA 还要提微服务？**这几个问题是理解微服务的关键，否则如果只是跟风拿来就用，既不会用，也用不好，用了不但没有效果，反而还可能副作用。

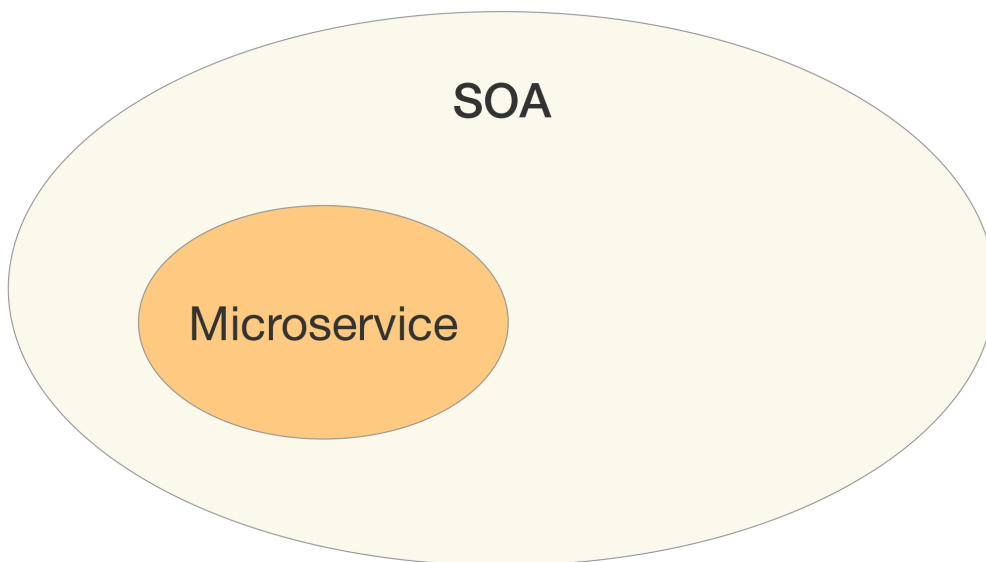
今天我们就来深入理解微服务，到底是银弹还是焦油坑。

微服务与 SOA 的关系

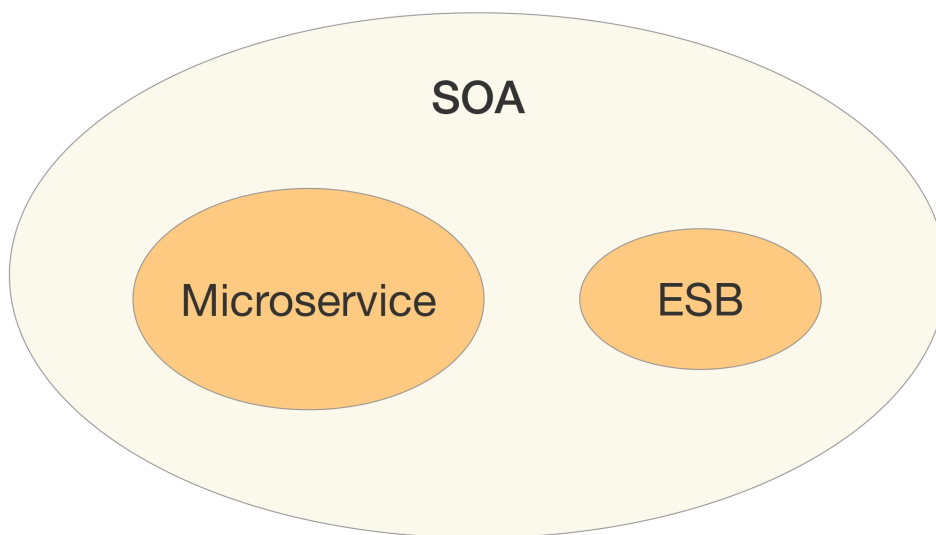
对于了解过 SOA 的人来说，第一次看到微服务这个概念肯定会有所疑惑：为何有了 SOA 还要提微服务呢？等到简单看完微服务的介绍后，可能很多人更困惑了：这不就是 SOA 吗？

关于 SOA 和微服务的关系和区别，大概分为下面几个典型的观点。

如下图所示，这种观点认为 SOA 是一种架构理念，而微服务是 SOA 理念的一种具体实现方法。例如，“微服务就是使用 HTTP RESTful 协议来实现 ESB 的 SOA”“使用 SOA 来构建单个系统就是微服务”和“微服务就是更细粒度的 SOA”。

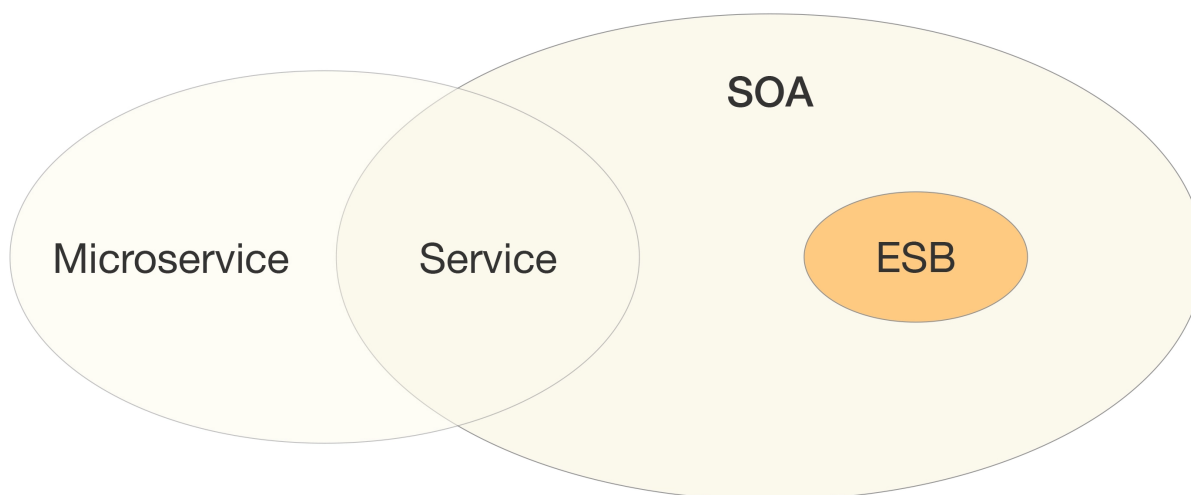


如下图所示，这种观点认为传统 SOA 架构最广为人诟病的就是庞大、复杂、低效的 ESB，因此将 ESB 去掉，改为轻量级的 HTTP 实现，就是微服务。



微服务是一种和 SOA 相似但本质上不同的架构理念

如下图所示，这种观点认为微服务和 SOA 只是有点类似，但本质上是不同的架构设计理念。相似点在于下图中交叉的地方，就是两者都关注“服务”，都是通过服务的拆分来解决可扩展性问题。本质上不同的地方在于几个核心理念的差异：是否有 ESB、服务的粒度、架构设计的目标等。



以上观点看似都有一定的道理，但都有点差别，到底哪个才是准确的呢？单纯从概念上是难以分辨的，我来对比一下 SOA 和微服务的一些具体做法，再来看看到底哪一种观点更加符合实际情况。

整体上来说，SOA 的服务粒度要粗一些，而微服务的服务粒度要细一些。例如，对一个大型企业来说，“员工管理系统”就是一个 SOA 架构中的服务；而如果采用微服务架构，则“员工管理系统”会被拆分为更多的服务，比如“员工信息管理”“员工考勤管理”“员工假期管理”和“员工福利管理”等更多服务。

SOA 采用了 ESB 作为服务间通信的关键组件，负责服务定义、服务路由、消息转换、消息传递，总体上是重量级的实现。微服务推荐使用统一的协议和格式，例如，RESTful 协议、RPC 协议，无须 ESB 这样的重量级实现。Martin Fowler 将微服务架构的服务通讯理念称为“Smart endpoints and dumb pipes”，简单翻译为“聪明的终端，愚蠢的管道”。之所以用“愚蠢”二字，其实就是与 ESB 对比的，因为 ESB 太强大了，既知道每个服务的协议类型（例如，是 RMI 还是 HTTP），又知道每个服务的数据类型（例如，是 XML 还是 JSON），还知道每个数据的格式（例如，是 2017-01-01 还是 01/01/2017），而微服务的“dumb pipes”仅仅做消息传递，对消息格式和内容一无所知。

SOA 对服务的交付并没有特殊要求，因为 SOA 更多考虑的是兼容已有的系统；微服务的架构理念要求“快速交付”，相应地要求采取自动化测试、持续集成、自动化部署等敏捷开发相关的最佳实践。如果没有这些基础能力支撑，微服务规模一旦变大（例如，超过 20 个微服务），整体就难以达到快速交付的要求，这也是很多企业在实行微服务时踩过的一个

明显的坑，就是系统拆分为微服务后，部署的成本呈指数上升。

SOA 更加适合于庞大、复杂、异构的企业级系统，这也是 SOA 诞生的背景。这类系统的典型特征就是很多系统已经发展多年，采用不同的企业级技术，有的是内部开发的，有的是外部购买的，无法完全推倒重来或者进行大规模的优化和重构。因为成本和影响太大，只能采用兼容的方式进行处理，而承担兼容任务的就是 ESB。

微服务更加适合于快速、轻量级、基于 Web 的互联网系统，这类系统业务变化快，需要快速尝试、快速交付；同时基本都是基于 Web，虽然开发技术可能差异很大（例如，Java、C++、.NET 等），但对外接口基本都是提供 HTTP RESTful 风格的接口，无须考虑在接口层进行类似 SOA 的 ESB 那样的处理。

综合上述分析，我将 SOA 和微服务对比如下：

对比维度	SOA	微服务
服务粒度	粗	细
服务通信	重量级，ESB	轻量级，例如 HTTP RESTful
服务交付	慢	快
应用场景	企业级	互联网

因此，我们可以看到，**SOA 和微服务本质上是两种不同的架构设计理念，只是在“服务”这个点上有交集而已，因此两者的关系应该是上面第三种观点。**

其实，Martin Fowler 在他的微服务文章中，已经做了很好的提炼：

In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery.

上述英文的三个关键词分别是：small、lightweight、automated，基本上浓缩了微服务的精华，也是微服务与 SOA 的本质区别所在。

通过前面的详细分析和比较，似乎微服务本质上就是一种比 SOA 要优秀很多的架构模式，那是否意味着我们都应该把架构重构为微服务呢？

其实不然，SOA 和微服务是两种不同理念的架构模式，并不存在孰优孰劣，只是应用场景不同而已。我们介绍 SOA 时候提到其产生历史背景是因为企业的 IT 服务系统庞大而又复杂，改造成本很高，但业务上又要求其互通，因此才会提出 SOA 这种解决方案。如果我们将微服务的架构模式生搬硬套到企业级 IT 服务系统中，这些 IT 服务系统的改造成本可能远远超出实施 SOA 的成本。

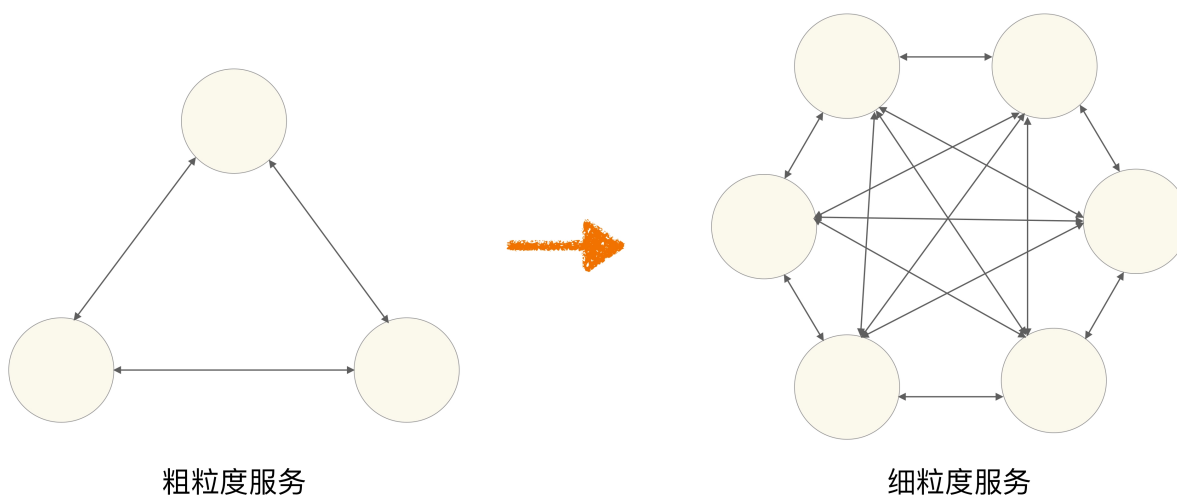
微服务的陷阱

单纯从上面的对比来看，似乎微服务大大优于 SOA，这也导致了很多团队在实践时不加思考地采用微服务——既不考虑团队的规模，也不考虑业务的发展，也没有考虑基础技术的支撑，只是觉得微服务很牛就赶紧来实施，以为实施了微服务后就什么问题都解决了，而一旦真正实施后才发现掉到微服务的坑里面去了。

我们看一下微服务具体有哪些坑：

服务划分过细，单个服务的复杂度确实下降了，但整个系统的复杂度却上升了，因为微服务将系统内的复杂度转移为系统间的复杂度了。

从理论的角度来计算， n 个服务的复杂度是 $n \times (n-1)/2$ ，整体系统的复杂度是随着微服务数量的增加呈指数级增加的。下图形象了说明了整体复杂度：



粗粒度划分服务时，系统被划分为 3 个服务，虽然单个服务较大，但服务间的关系很简单；细粒度划分服务时，虽然单个服务小了一些，但服务间的关系却复杂了很多。

微服务的“微”字，本身就是一个陷阱，很多团队看到“微”字后，就想到必须将服务拆分得很细，有的团队人员规模是 5 ~ 6 个人，然而却拆分出 30 多个微服务，平均每个人要维护 5 个以上的微服务。

这样做给工作效率带来了明显的影响，一个简单的需求开发就需要涉及多个微服务，光是微服务之间的接口就有 6 ~ 7 个，无论是设计、开发、测试、部署，都需要工程师不停地在不同的服务间切换。

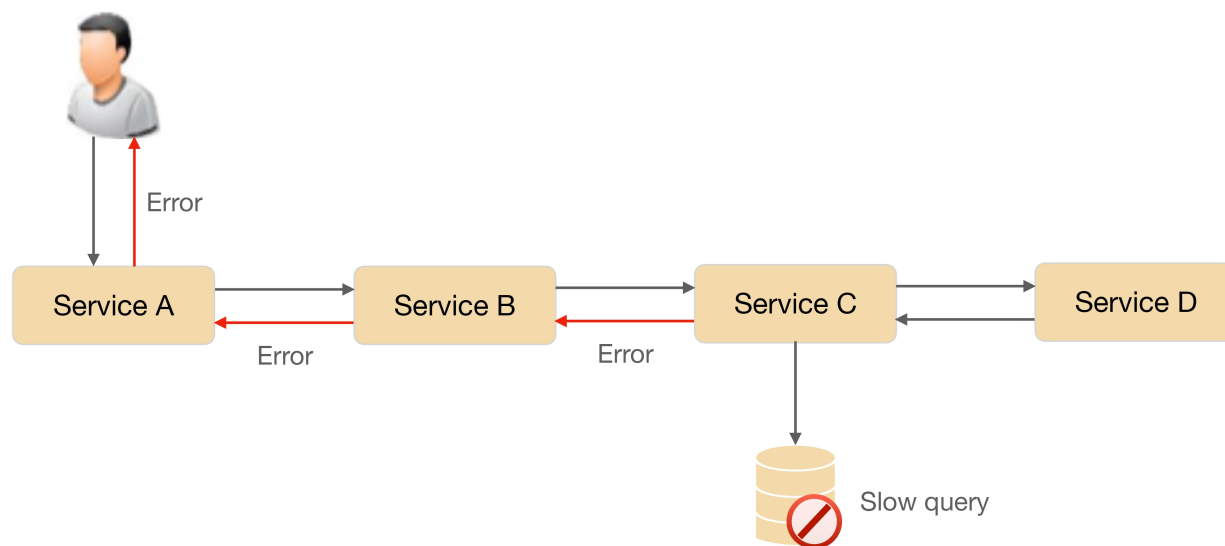
开发工程师要设计多个接口，打开多个工程，调试时要部署多个程序，提测时打多个包。

测试工程师要部署多个环境，准备多个微服务的数据，测试多个接口。

运维工程师每次上线都要操作多个微服务，并且微服务之间可能还有依赖关系。

由于微服务之间都是通过 HTTP 或者 RPC 调用的，每次调用必须经过网络。一般线上的业务接口之间的调用，平均响应时间大约为 50 毫秒，如果用户的一起请求需要经过 6 次微服务调用，则性能消耗就是 300 毫秒，这在很多高性能业务场景下是难以满足需求的。为了支撑业务请求，可能需要大幅增加硬件，这就导致了硬件成本的大幅上升。

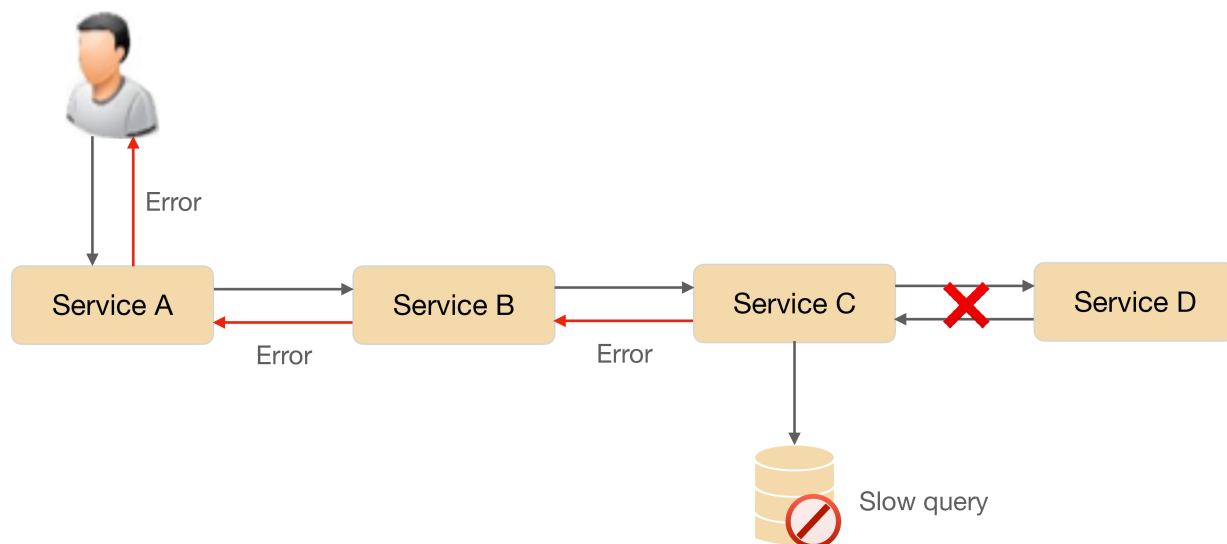
系统拆分为微服务后，一次用户请求需要多个微服务协同处理，任意微服务的故障都将导致整个业务失败。然而由于微服务数量较多，且故障存在扩散现象，快速定位到底是哪个微服务故障是一件复杂的事情。下面是一个典型样例。



Service C 的数据库出现慢查询，导致 Service C 给 Service B 的响应错误，Service B 给 Service A 的响应错误，Service A 给用户的响应错误。我们在实际定位时是不会有样例

图中这么清晰的，最开始是用户报错，这时我们首先会去查 Service A。导致 Service A 故障的原因有很多，我们可能要花半个小时甚至 1 个小时才能发现是 Service B 返回错误导致的。于是我们又去查 Service B，这相当于重复 Service A 故障定位的步骤.....如此循环下去，最后可能花费了几个小时才能定位到是 Service C 的数据库慢查询导致了错误。

如果多个微服务同时发生不同类型的故障，则定位故障更加复杂，如下图所示。



Service C 的数据库发生慢查询故障，同时 Service C 到 Service D 的网络出现故障，此时到底是哪个原因导致了 Service C 返回 Error 给 Service B，需要大量的信息和人力去排查。

如果没有相应的自动化系统进行支撑，都是靠人工去操作，那么微服务不但达不到快速交付的目的，甚至还不如一个大而全的系统效率高。例如：

没有自动化测试支撑，每次测试时需要测试大量接口。

没有自动化部署支撑，每次部署 6 ~ 7 个服务，几十台机器，运维人员敲 shell 命令逐台部署，手都要敲麻。

没有自动化监控，每次故障定位都需要人工查几十台机器几百个微服务的各种状态和各种日志文件。

信奉微服务理念的设计人员总是强调微服务的 lightweight 特性，并举出 ESB 的反例来证明微服务的优越之处。但具体实践后就会发现，随着微服务种类和数量越来越多，如果没有服务治理系统进行支撑，微服务提倡的 lightweight 就会变成问题。主要问题有：

服务路由：假设某个微服务有 60 个节点，部署在 20 台机器上，那么其他依赖的微服

务如何知道这个部署情况呢？

服务故障隔离：假设上述例子中的 60 个节点有 5 个节点发生故障了，依赖的微服务如何处理这种情况呢？

服务注册和发现：同样是上述的例子，现在我们决定从 60 个节点扩容到 80 个节点，或者将 60 个节点缩减为 40 个节点，新增或者减少的节点如何让依赖的服务知道呢？

如果以上场景都依赖人工去管理，整个系统将陷入一片混乱，最终的解决方案必须依赖自动化的服务管理系统，这时就会发现，微服务所推崇的“lightweight”，最终也发展成和 ESB 几乎一样的复杂程度。

小结

今天我为你讲了微服务与 SOA 的关系以及微服务实践中的常见陷阱，希望对你有所帮助。

这就是今天的全部内容，留一道思考题给你吧，你们的业务有采用微服务么？谈谈具体实践过程中有什么经验和教训。

[上一页](#)

[下一页](#)