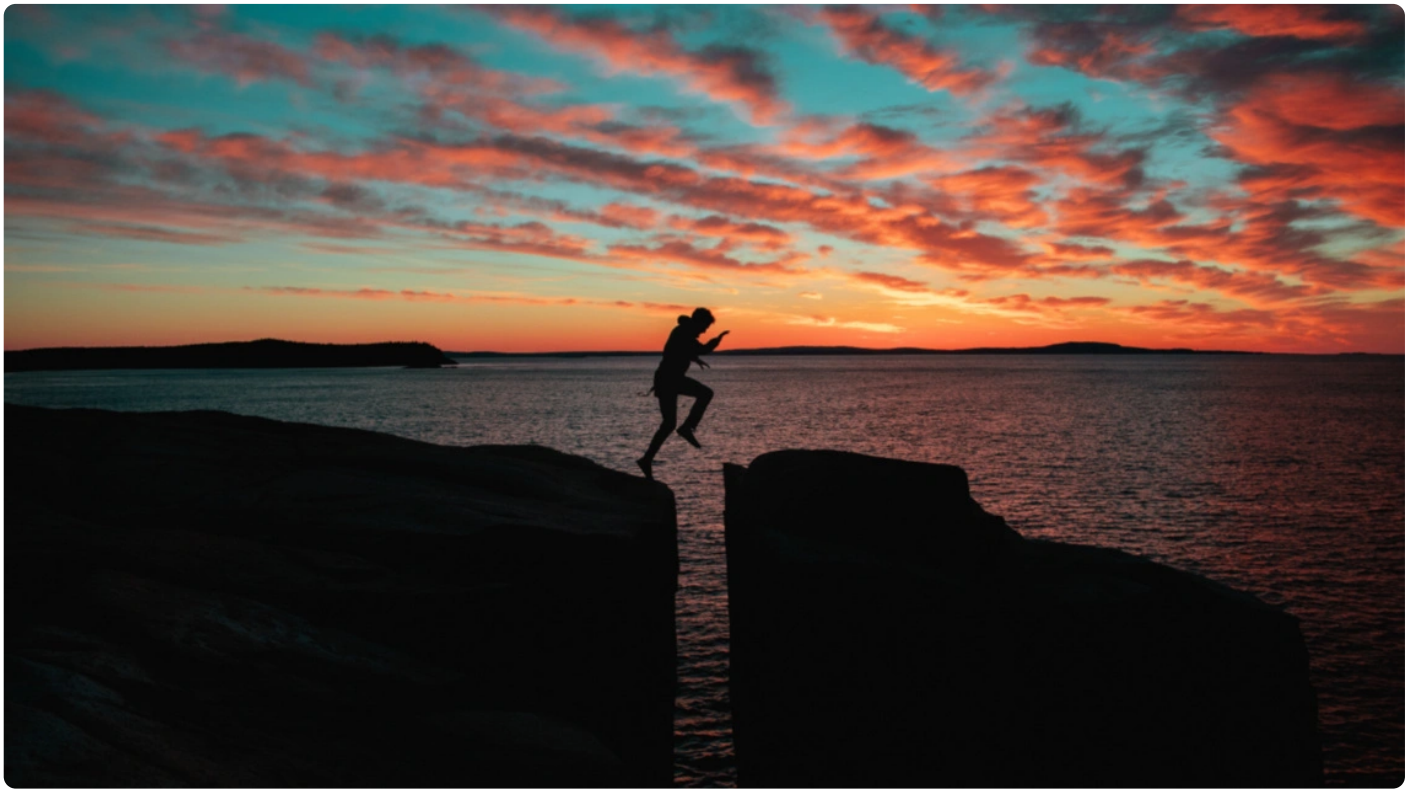
34 | 深入理解微服务架构：银弹 or 焦油坑?

2018-07-14 李运华 来自北京

《从0开始学架构》



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。

微服务的定义相信你早已耳熟能详，参考维基百科，我就来简单梳理一下微服务的历史吧（https://en.wikipedia.org/wiki/Microservices#History）：

> 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，反而还可能有副作用。

今天我们就来深入理解微服务，到底是银弹还是焦油坑。

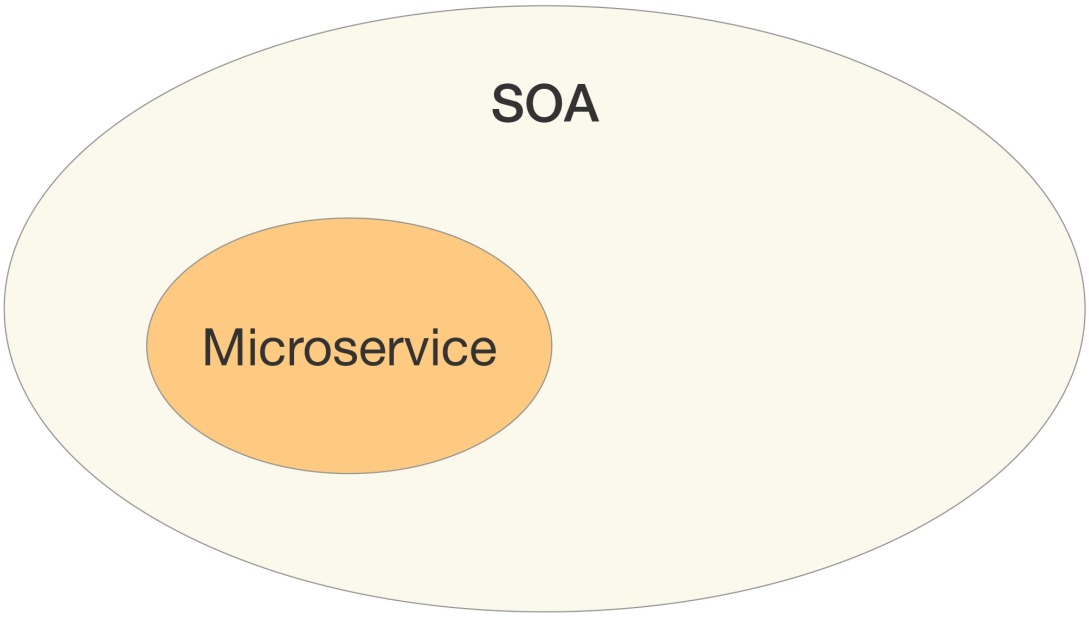
微服务与 SOA 的关系

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。

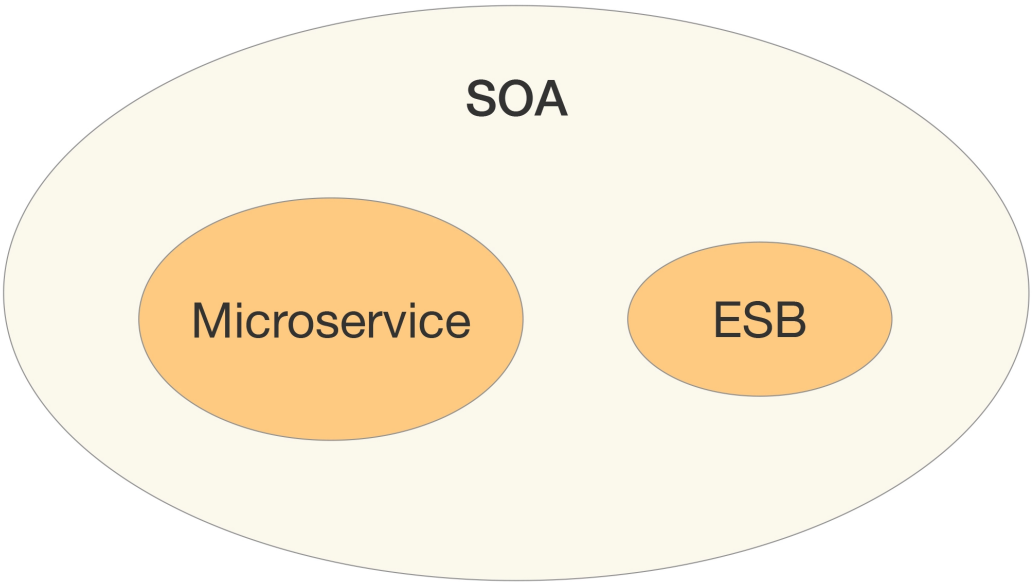
微服务是 SOA 的实现方式

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”。



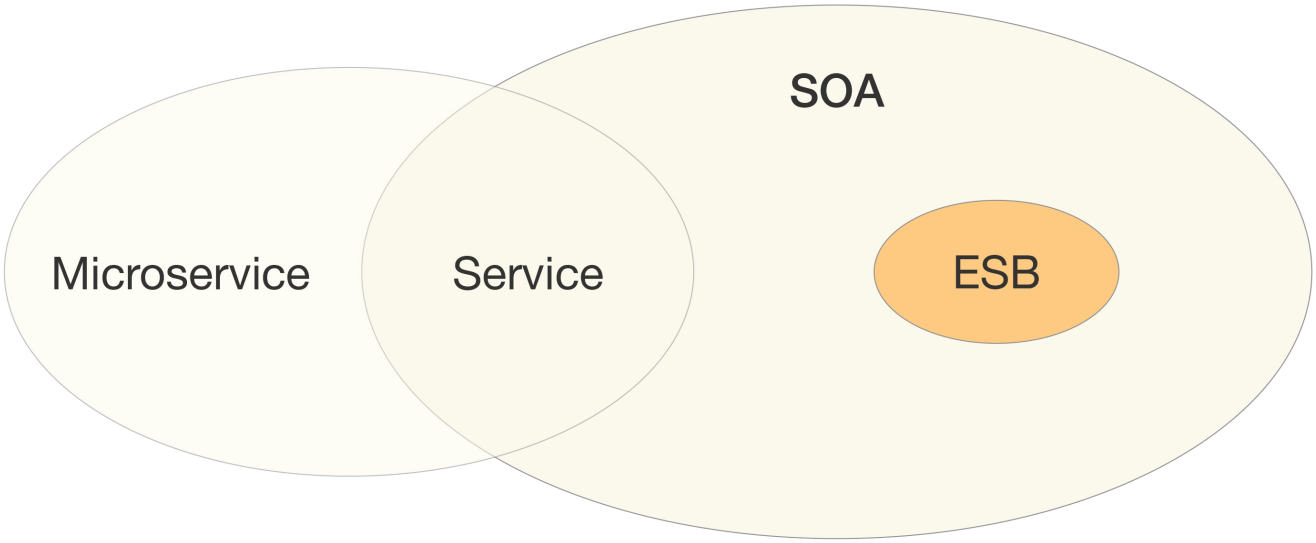
微服务是去掉 ESB 后的 SOA

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。

1. 服务粒度

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。

2. 服务通信

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。

3. 服务交付

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。

4. 应用场景

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.

（https://martinfowler.com/articles/microservices.html）

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

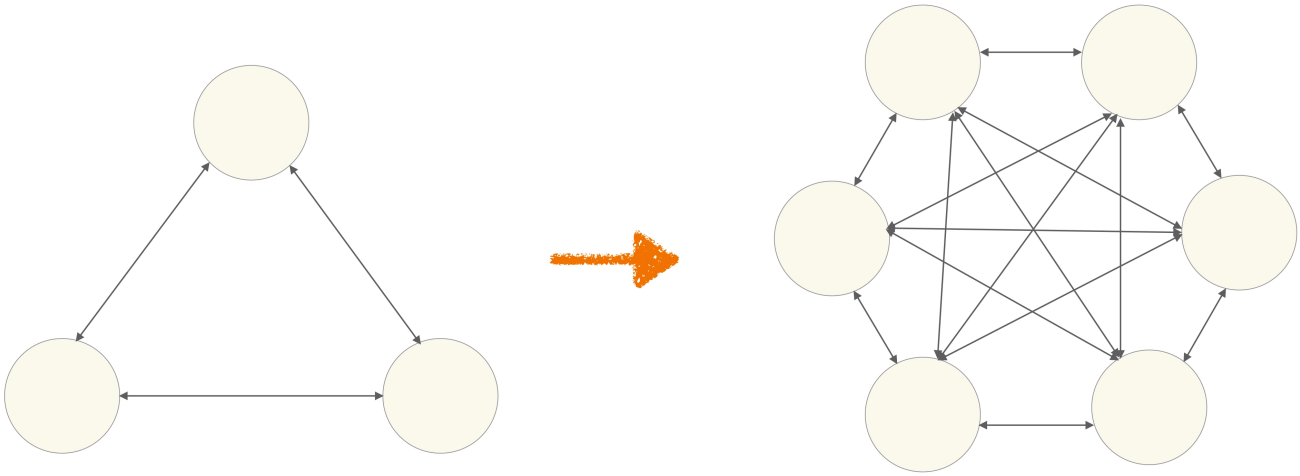
单纯从上面的对比来看，似乎微服务大大优于 SOA，这也导致了很多团队在实践时不加思考地采用微服务——既不考虑团队的规模，也不考虑业务的发展，也没有考虑基础技术的支撑，只是觉得微服务很牛就赶紧来实施，以为实施了微服务后就什么问题都解决了，而一旦真正实施后才发现掉到微服务的坑里面去了。

我们看一下微服务具体有哪些坑：

1. 服务划分过细，服务间关系复杂

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×(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：



粗粒度服务　　细粒度服务

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。

2. 服务数量太多，团队效率急剧下降

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。

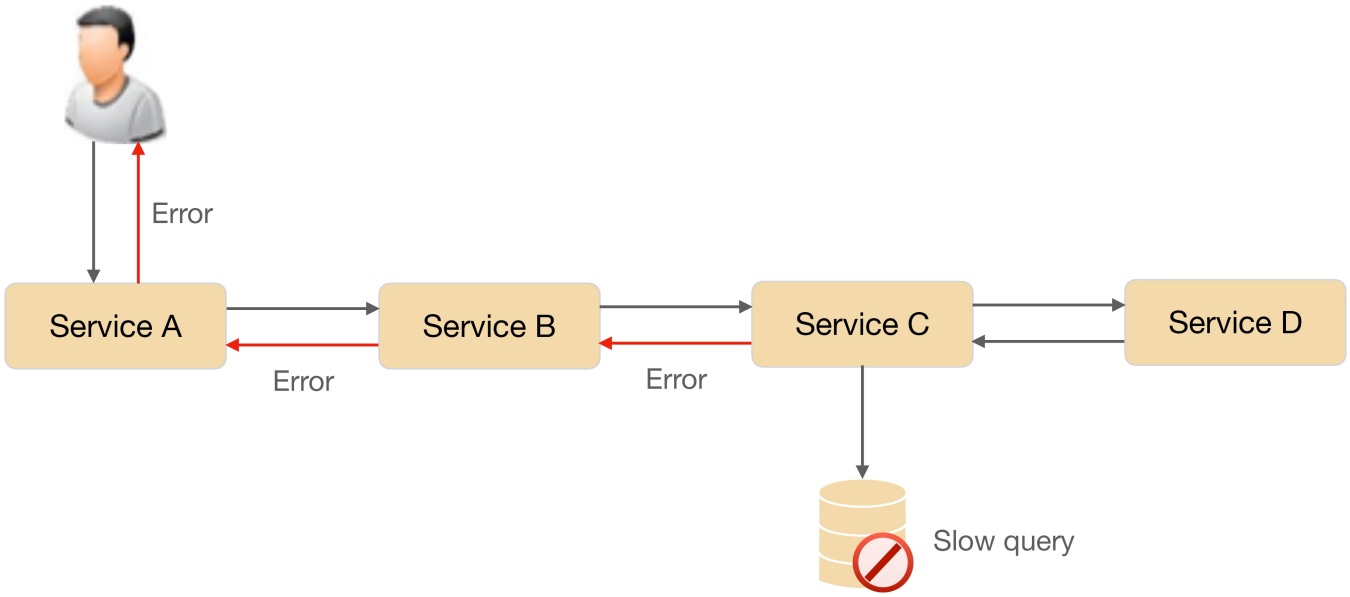
运维工程师每次上线都要操作多个微服务，并且微服务之间可能还有依赖关系。

3. 调用链太长，性能下降

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。

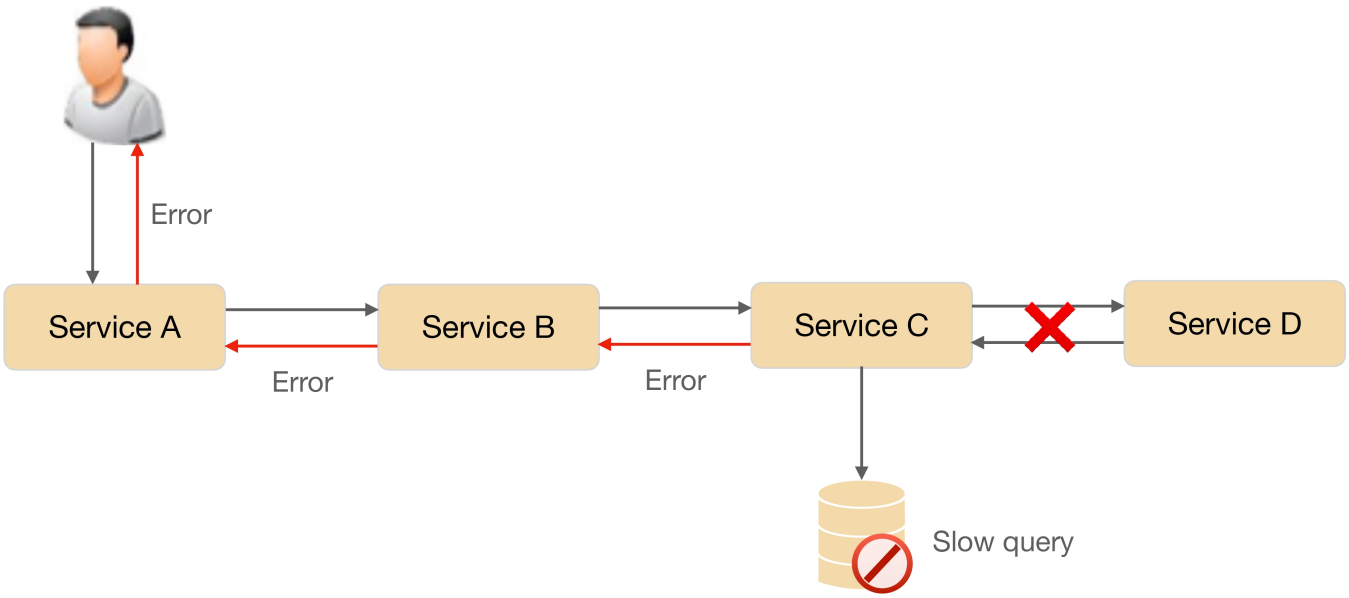
4. 调用链太长，问题定位困难

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。

5. 没有自动化支撑，无法快速交付

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。

6. 没有服务治理，微服务数量多了后管理混乱

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。

欢迎你把答案写到留言区，和我一起讨论。相信经过深度思考的回答，也会让你对知识的理解更加深刻。（编辑乱入：精彩的留言有机会获得丰厚福利哦！）

精选留言 (91)

Tom 置顶

2018-08-10

我们公司的实践是比较粗粒度的子系统或服务，基本上没有太细粒度的微服务，以webapi为主。感觉更像微服务架构，只是服务粒度比较粗，从概念上算是SOA还是微服务架构呢?

> 作者回复: 我理解算微服务，千万不要理解为微服务就是将服务拆的很细，后面有具体实践技巧介绍这部分

25

鹅米豆发

2018-07-16

对于一个新事物的诞生，本能地套用已有的知识。特别是一个并不简单的东西，这算是一种高效的入门方法。微服务架构其实相当复杂，我是分成好几个阶段理解。
1、第一阶段，微服务架构就是去掉了ESB的SOA架构，只不过是通信的方式和结构变了。对于初级的使用者而言，这样理解没有太大问题。
2、第二阶段，没有了ESB，原本很多由ESB组件做的事儿，转到服务的提供者和调用者这里了。他们需要考虑服务的拆分粒。大体仍然算是SOA架构。
3、第三阶段，随着服务的数量大幅增加，服务的管理越来越困难，此时DevOps出现了。这个阶段的微服务架构，已经是跟SOA架构完全不同的东西了。
之前给一家大国企分享过一些经验。他们想从传统架构，转向微服务架构。

1、建设好基础设施，RPC、服务治理、日志、监控、持续集成、持续部署、运维自动化是基本的，其它包括服务编排、分布式追踪等。
2、要逐步演进和迭代，不要过于激进，更不要拆分过细，拆分的粒度，要与团队的架构相互匹配。（康威定律）
3、微服务与数据库方面，是个很大的难点，可以深入了解下领域驱动设计，做好领域建模，特别是数据库要随着服务一起拆分。
说完上面这些，他们的研发负责人说，我说的跟他们的架构师说的不一样，他们的架构师说，微服务就是各种拆分，不顾一切地拆分。

作者回复: 他们架构师是水货😂你的理解和分析是对的，后一篇就讲了

共 8 条评论 >　　137

空档滑行
2018-07-14

之前一家公司搞了一次完整的微服务改造，享受到了一些好处，但是文中说到的问题，大部分都碰上了。
先说下好处，原来的单体应用都服务化了，扩容简单很多。功能隔离后之前一个bug导致系统挂掉的现象没了。问题责任定位划分的更清楚，比如之前大量慢sql无人管，现在通过监控快速找到开发责任人。
再说下坏处，1.服务太多了，人不够啊。之前的架构师按照小的原则，把数据层，服务层，应用层严格拆分。一个人手上超过10几个服务...
2.服务化不彻底，太多事手工干。服务监控只能监控一半指标，各种远程调用异常没人解决。运维只有打包发布做了自动化。可以想象下开发人员基本下改bug和发布的死循环中。服务网关没有，服务调用就是一张密密麻麻的网
3.培训不到位，直接上阵，开发人员对微服务理解不到位，服务质量可想而知
4.没有专职测试，自动化测试靠开发写脚本，谁有空啊，单元测试能写一个就算相当有觉悟了
总结下来，做服务化改造首先问自己这些问题，业务真的需要微服务来解决吗？真的所有模块的问题都要微服务来解决吗？技术人员的配置和水平达到要求了吗？

共 5 条评论 >　　47

ant
2018-07-16

我们是一家社交公司，后端加厚的演变符合dubbo官网的那张图，在Mvc架构坚持了一年后，业务越来越大，工程越来越臃肿。后面我们一致同意进行服务话，开始用了dubbo。后面由于决策层的原因没有上，后面来了个架构师，又重启了服务拆分，到现在已经用于生产。我们使用的是Spring cloud，现在拆分暴露了很多问题:
1、个别服务没有熔断出来，出现过雪崩效应

2、服务拆分过细，服务调用链过长
3、开发人员能力不一样，代码水平不一样
4、没有监控措施
5、每个服务部署多台，日志查询就会死人的感觉
6、开发过程中经常出现访问不到该访问的接口，这是因为开发人员经常启动本地服务，就会导致30%的概率访问不到
7、使用了不合理的持久层框架，使用了JPA访问
大概就有上面的问题，总之微服务不是银弹，用得不好会发现无穷无尽的坑。当然，出现问题解决问题就是了。唯一的就像CEO说的: 你们他妈的是完全在拿用户当小白鼠使用

作者回复: 用户会用脚投票的😂😂

共 4 条评论 >

31

三棱镜
2018-08-17

我们目前全部微服务，踩坑踩了不少，拆分服务同时要把自动化运维系统和多维度监控系统，包括问题定位跟踪系统建立起来，要不然拆了就是噩梦。

作者回复: 感同身受啊😄

共 2 条评论 >

19

凡凡
2018-07-16

说到微服务，切分的粒度和基础设施都致关重要。

经历的项目有创业初期的单体服务，也有不太完善的服务切分的系统，也有微服务基础设施相对完善的公司。

单体服务致命就致命在随着项目的发展，项目会越来越臃肿，越不利于扩展开发。

微服务过程怕就怕基础不完善，人员配备不够盲目切分，导致工程师开发和维护的战线拉长，特别疲惫和泄气，容易产生一种抱怨的大气氛，从而导致微服务失败，重新合并一部分服务。

微服务做的好的，也有所经历，公司的基础设施完全云化和统一管理，申请几台机器，一套缓存集群，一套mq，sql/nosql...特别容易，工程师愿意建独立的工程，因为很容易构建和部署。这种感觉有点像svn和git，git建分支特别轻量，大家都愿意用分支管理自己的代码，迭代

开发，应急处理都能自由切换，得心应手。

现在遇到一个问题，就是微服务内部系统大多使用rpc，但对接外部系统，或者跨外网传输到客户端就需要http-rest类的协议。也就是我们常说的网关，如果是纯http就很容易做到通用的转发机制，但是http转rpc就不知道有什么方式可以做到通用转发了，内部每增加一个rpc，网关就需要增加一个对应的服务处理逻辑。不知道，这个问题，有没有好的解决办法?

作者回复: 把HTTP转RPC做成规则，别硬编码每个接口，例如，规定HTTP URL为/service/interface/method?para1=xxx&para2=yyy

15

木头旮瘩
2018-12-12

让我想起了一次面试经历，一家小公司，技术团队大概7 8个人，然后他们要进行微服务架构改造，我问他们:你们这样满足康威定理么？他们一脸懵逼……我就果断闪人了

作者回复: 你可以去改造他们😂

共 2 条评论>

14

正是那朵玫瑰
2018-07-14

我们的业务也算是微服务吧，　　接手项目时，已经有好多服务，主要采用的语言有java，php，nodejs，存在以下问题：
1、没有一个统一的网关服务，前端请求后端服务都需要后端的服务A来充当安全校验，权限校验等，A服务充当了多重职责，变的职责不明确了，后来抽出网关系统，负责平台统一的流量入口。在构建微服务网关系统是至关重要的。
2、监控系统不完善，调用链跟踪，异常报警都不完善，对微服务是巨坑，查找问题如同一场噩梦，调用链很长，一旦发生异常不知道到底哪里出现问题，得一个一个去找。后来慢慢完善，变得好了很多，问题很快定位。
3、加入网关后，没有一个统一的服务发现注册中心，网关的路由依靠人工手动配置，变的很麻烦，也很容易出错。后来引入consul，得到改善。

作者回复: spring全家桶，你值得拥有😂

共 3 条评论>

10

子清
2018-07-15

我们公司的平台就是使用微服务架构，十多二十个微服务，但是没有自动化部署，监控，自动化测试这些，而且每次报错日志也特别难找，但是我们那架构师却不重视这些，只想继续升级平台的功能

作者回复: 让你们架构师来订阅架构专栏😂😂😂

9

Jussi Lee
2018-09-18

我们项目之前是把数据采集，和展示都是我们Java组去写的。后面公司为了统一，把采集部分交给了.net组。我们Java只负责数据展示。但是我们这边又按照领导的要求把整个业务拆分为模板解析层-》api层——》终端层。现在每次最烦的事就是找bug。一直感觉现阶段我们的任务和目标没有这么庞大，这与架构中的简单和合适原则想违背。搞的整个项目组一直在反复的开发工作中。都很心累

作者回复: 很好的案例👍

共 2 条评论>

8

修仙中
2020-12-18

我们之前5个人的小团队，老板一拍脑袋玩起了微服务。整了spring cloud那一套全家桶：
现有业务拆拆拆，什么注册中心，配置中心，链路追踪，网关，ELK统统往系统里面堆。
主要的带来了以下问题：
1、运维成本过高，部署物数量多、监控进程多导致整体运维复杂度提升。
2、接口大量的兼容版本
3、网络延迟，服务容错，可用性，负载复杂性大大提升
4、还引入了分布式事务
团队没有运维，我们只能硬着头皮上，自己用docker搭redis集群，mysql集群，mq集群，es集群
最终因为不够钱买服务器，整个叫停了，哈哈哈哈哈

作者回复: 最后的结局让人猝不及防😂😂

共 3 条评论>

7

Daniel大东

2019-06-10

能否来一篇中台战略和微服务的文章?

作者回复: 我跟编辑聊聊看

6

乘风

2018-12-11

16年时，架构师引入了微服务架构，架构设计分为三层：网关层、业务处理层、数据处理层（getaway->ls>ds），自上而下依赖，业务层之间也互相依赖，构建过程中发现引入了分布式事务和调用链长（调用链的消耗时间无法接受），经常无故报错，定位问题慢，测试复杂等问题，而我们的大多数业务相对简单，所以重新划分服务粒度，分为两层：网关层：提供路由功能，业务处理层：处理请求，业务层中间可以相互调用，当出现有出现分布式事务时，采取MQ或者更新状态的方式解决，如果是充值+业务订单的这种会直接在一个事务里处理（其实这里就混乱一点了），层与层之间没有完全隔离，但是将调用链缩短，我觉得采用微服务架构确实解决了单体应用的一些问题，如资源问题（但消耗资源是比以前多的）、耦合问题、快速迭代问题，服务与服务的职责更加清晰，不同的服务粒度设计的系统是天差地别的。

作者回复: 是的，粒度太重要了

5

hello

2018-08-21

我记得有一次做业务，当时不知道什么原因，整体使用微服务，当时是个新业务，完全从0开始，没有任何用户基础，当时按照业务把服务拆成了7.8个微服务，但没有自动化，服务还是部署在一台机器上，服务之间也没有服务治理，服务之间的调用链很长，开发就4.5个人，定位问题花费时间长，典型的基础服务也跟不上。现在想想，当时团队不知道咋想的，觉得微服务比较新就上了，完全没有考虑到产生的问题。还好后来这块业务黄了。要不然估计坑更多。对于创业公司来说，刚开始的时候，其实一个服务就够了，真等用户上来了，业务有希望了，那时候再上其他架构模式也不迟，忌讳一开始就求新求全，说到底，对于大部分公司来说。技术是为业务服务的。

作者回复: 你们缺少一个真正的架构师😂

共 3 条评论>

4

旭东(Frank)
2018-08-09

个人觉得SOA只是提出了面向服务的编程，到但没有对服务粒度的定义，以及服务的治理问题做深入的分析。
提出微服务主要是为了解决面向服务架构后如何能够在实际工程中带来真正的红利。
微服务对SOA的技术关键点给出了指导意见。

微服务对分布式服务的诱惑力的确很大，但做但微服务都不会免费的，需要很多的基础设施来做铺垫和基石，才有可能搞定。

除了基础设施还有很多设计思想需要学习，我觉得DDD就是微服务的绝配。

作者回复: SOA是完整的解决方案哦，IBM等公司卖ESB都卖了好多钱😀

共 2 条评论 >

4

Neal
2018-07-16

1. 数据库拆不开
2. 人手不够就两人

4

narry
2018-07-14

我弄微服务遇到最大的坑，就是jvm与docker兼容性不好，导致每个微服务会消耗过多不必要的内存，我感觉从资源消耗上来说，java开发的微服务都不能算是微服务了，最近在转向go来改造

作者回复: 这还真是第一次听说jvm与docker不兼容，看看是不是有bug

4

冬阳(Wolfer Chen)
2018-11-08

我个人比较认同康威定律，微服务的拆分粒度一定要和组织结构匹配起来，组织结构和开发管理模式是微服务粒度的最重要参考指标之一。

作者回复: 是的，康威定律和微服务拆分是相关的

共 2 条评论 >

3

森林

2018-09-20

概念是会一直演化的，我相信最初的SOA的概念就是作者所说的形式，但是仅仅从"面向服务的架构"这几个字来说，难道他就不能是微服务的超集么？在微服务名字出来前，用dubbo的系统是怎么称呼其架构的？微服务定义一开始还要求一定是基于Rest的，难道用dubbo的就不是了？所以个人觉得，理解历史可以，但限定于SOA就一定是基于ESB做协议转换联通各个系统的一种架构是不妥的。

作者回复: 如果从历史产生的背景去理解，那就每个人都有自己的理解，没法沟通交流呢😊

3

衣申人

2018-07-15

同问:dubbo和motan这类rpc框架，是微服务框架？？

作者回复: dubbo可以算微服务基础设施的一部分，主要承担服务注册发现等

3