

最近几年，我们一直在谈论各式各样的架构，如高并发架构、异地多活架构、容器化架构、微服务架构、高可用架构、弹性化架构等。还有和这些架构相关的管理型的技术方法，如DevOps、应用监控、自动化运维、SOA服务治理、去IOE等。面对这么多纷乱的技术，我看到很多团队或是公司都是一个一个地去做这些技术，非常辛苦，也非常累。这样的做法就像我们在撑开一张网里面一个一个的网眼。

其实，只要我们能够找到这张网的“纲”，我们就能比较方便和自如地打开整张网了。那么，这张“分布式大网”的总线——“纲”在哪里呢？我希望通过这一系列文章可以让你找到这个“纲”，从而能让你更好更有效率地做好架构和工程。

分布式系统架构的冰与火

首先，我们需要阐述一下为什么需要分布式系统，而不是传统的单体架构。也许这对你来说已经不是什么问题了，但是请允许我在这里重新说明一下。使用分布式系统主要有两方面原因。

- **增大系统容量。**我们的业务量越来越大，而要能应对越来越大的业务量，一台机器的性能已经无法满足足了，我们需要多台机器才能应对大规模的应用场景。所以，我们需要垂直或是水平拆分业务系统，让其变成一个分布式的架构。
- **加强系统可用。**我们的业务越来越关键，需要提高整个系统架构的可用性，这就意味着架构中不能存在单点故障。这样，整个系统不会因为一台机器出故障而导致整体不可用。所以，需要通过分布式架构来冗余系统以消除单点故障，从而提高系统的可用性。

当然，分布式系统还有一些优势，比如：

- 因为模块化，所以系统模块重用度更高；
- 因为软件服务模块被拆分，开发和发布速度可以并行而变得更快；

- 系统扩展性更高；

- 团队协作流程也会得到改善；

-

不过，这个世界上不存在完美的技术方案，采用任何技术方案都是“按下葫芦浮起瓢”，都是有得有失，都是一种trade-off。也就是说，分布式系统在消除上述问题的同时，也给我们带来了其他的问题。因此，我们需要清楚地知道分布式系统所带来的问题。

下面这个表格比较了单体应用和分布式架构的优缺点。

	传统单体架构	分布式服务化架构
新功能开发	需要时间	容易开发和实现
部署	不经常且容易部署	经常发布，部署复杂
隔离性	故障影响范围大	故障影响范围小
架构设计	难度小	难度级数增加
系统性能	响应时间快，吞吐量小	响应时间慢，吞吐量大
系统运维	运维简单	运维复杂
新人上手	学习曲线大（应用逻辑）	学习曲线大（架构逻辑）
技术	技术单一且封闭	技术多样且开放
测试和查错	简单	复杂
系统扩展性	扩展性很差	扩展性很好
系统管理	重点在于开发成本	重点在于服务治理和调度

从上面的表格我们可以看到，分布式系统虽然有一些优势，但也存在一些问题。

- 架构设计变得复杂（尤其是其中的分布式事务）。
- 部署单个服务会比较快，但是如果一次部署需要多个服务，部署会变得复杂。
- 系统的吞吐量会变大，但是响应时间会变长。
- 运维复杂度会因为服务变多而变得很复杂。
- 架构复杂导致学习曲线变大。
- 测试和查错的复杂度增大。
- 技术可以很多样，这会带来维护和运维的复杂度。
- 管理分布式系统中的服务和调度变得困难和复杂。

也就是说，分布式系统架构的难点在于系统设计，以及管理和运维。所以，分布式架构解决了“单点”和“性能容量”的问题，但却新增了一堆问题。而对于这些新增的问题，还会衍生出更多的子问题，这就需要我们不断地用各式各样的技术和手段来解决这些问题。

这就出现了我前面所说的那些架构方式，以及各种相关的管理型的技术方法。这个世界就是这样变得复杂起来的。

分布式系统的发展

从20世纪70年代的模块化编程，80年代的面向事件设计，90年代的基于接口/构件设计，这个世界很自然地演化出了SOA——基于服务的架构。SOA架构是构造分布式计算应用程序的方法。它将应用程序功能作为服务发送给最终用户或者其他服务。它采用开放标准与软件资源进行交互，并采用标准的表示方式。

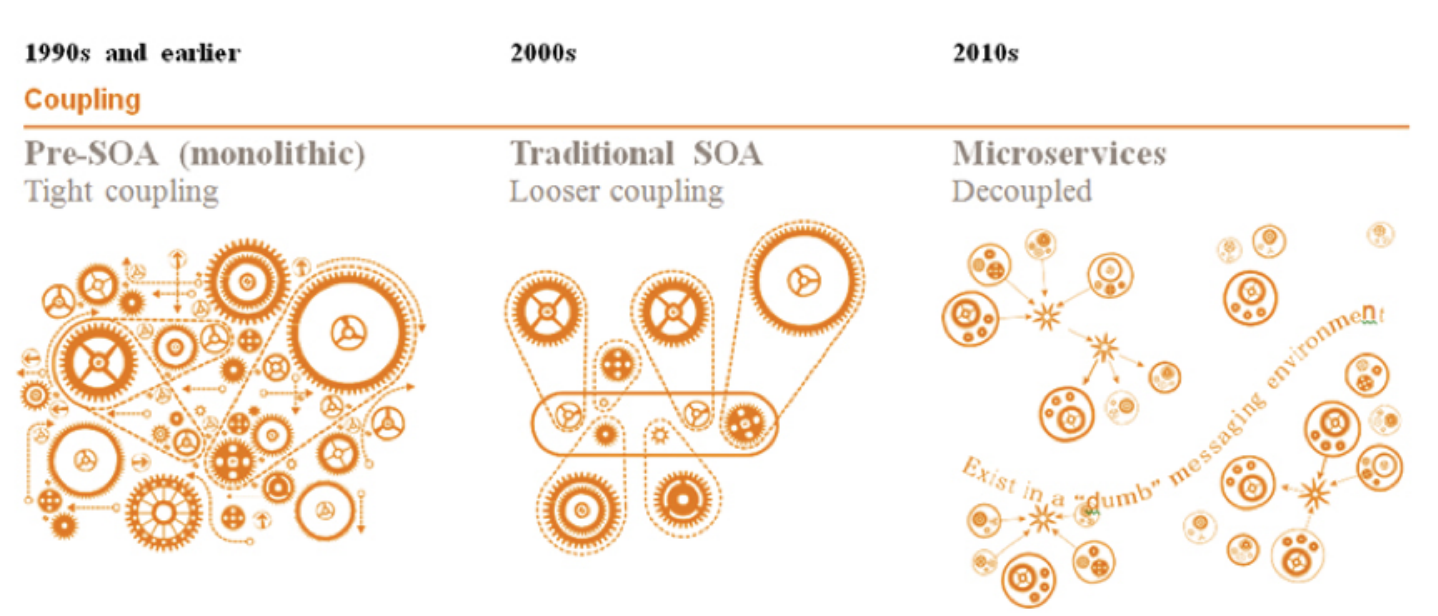
开发、维护和使用SOA要遵循以下几条基本原则。

- 可重用，粒度合适，模块化，可组合，构件化以及有互操作性。
- 符合开放标准（通用的或行业的）。
- 服务的识别和分类，提供和发布，监控和跟踪。

但IBM搞出来的SOA非常重，所以对SOA的裁剪和优化从来没有停止过。比如，之前的SOAP、WSDL和XML这样的东西基本上已经被抛弃了，而改成了RESTful和JSON这样的方式。而ESB（Enterprise Service Bus，企业服务总线）这样非常重要的东西也被简化成了Pub/Sub的消息服务.....

不过，SOA的思想一直延续着。所以，我们现在也不说SOA了，而是说分布式服务架构了。

下面是一个SOA架构的演化图。



我们可以看到，面向服务的架构有以下三个阶段。

- 20世纪90年代前，是单体架构，软件模块高度耦合。当然，这张图同样也说明了有的SOA架构其实和单体架构没什么两样，因为都是高度耦合在一起的。就像图中的齿轮一样，当你调用一个服务时，这个服务会调用另一个服务，然后又调用另外的服务.....于是整个系统就转起来了。但是这本质是比较耦合的做法。
- 而2000年左右出现了比较松耦合的SOA架构，这个架构需要一个标准的协议或是中间件来联动其它相关联的服务（如ESB）。这样一来，服务间并不直接依赖，而是通过中间件的标准协议或是通讯框架相互依赖。这其实就是IoC（控制反转）和DIP（依赖倒置原则）的设计思想在架构中的实践。它们都依赖于一个标准的协议或是一个标准统一的交互方式，而不是直接调用。
- 而2010年后，出现了微服务架构，这个架构更为松耦合。每一个微服务都能独立完整地运行（所谓的自包含），后端单体的数据库也被微服务这样的架构分散到不同的服务中。而它和传统SOA的差别在于，服务间的整合需要一个服务编排或是服务整合的引擎。就好像交响乐中需要有一个指挥来把所有乐器编排和组织在一起。

一般来说，这个编排和组织引擎可以是工作流引擎，也可以是网关。当然，还需要辅助于像容器化调度这样的技术方式，如Kubernetes。在Martin Fowler 的 [Microservices 这篇文章](#)中有详细描述。

微服务的出现使得开发速度变得更快，部署快，隔离性高，系统的扩展度也很好，但是在集成测试、运维和服务管理等方面就比较麻烦了。所以，需要一套比较好的微服务PaaS平台。就像Spring Cloud一样需要提供各种配置服务、服务发现、智能路由、控制总线.....还有像Kubernetes提供的各式各样的部署和调度方式。

没有这些PaaS层的支撑，微服务也是很难被管理和运维的。好在今天的世界已经有具备了这些方面的基础设施，所以，采用微服务架构，我认为只是一个时间问题了。

小结

好了，今天的内容就到这里。相信通过今天的学习，你应该已经对为什么需要分布式系统，而不是传统的单体架构，有了清晰的认识。并且对分布式系统的发展历程了然于心。下一篇文章，我将结合亚马逊的分布式架构实践，来谈谈分布式系统架构的技术难点及应对方案。

文末尾给出了《分布式系统架构的本质》系列文章的目录，希望你能在这个列表里找到自己感兴趣的内容。

- [分布式系统架构的冰与火](#)
- [从亚马逊的实践，谈分布式系统的难点](#)
- [分布式系统的技术栈](#)
- [分布式系统关键技术：全栈监控](#)
- [分布式系统关键技术：服务调度](#)
- [分布式系统关键技术：流量与数据调度](#)
- [洞悉PaaS平台的本质](#)
- [推荐阅读：分布式系统架构经典资料](#)
- [推荐阅读：分布式数据调度相关论文](#)

在文章的最后，很想听听大家在进行分布式系统开发，把一个单体应用拆解成服务化或是微服务中遇到的问题和难点是什么？踩过什么样的坑？你是如何应对的？

[戳此获取你的专属海报](#)