# 从单体架构到分布式架构的演进

任何架构都是随着用户量和数据量的增长而不断演变来的。而架构的发展可能都会经历 单体架构、垂直化和集群、SOA（面向服务器架构）、微服务架构等。当然不是所有 的公司都会按照这个架构顺序来演进，每个公司遇到的问题不一样，架构的发展过程也 不一样。当用户量爆发式的增长，处理方式一定是先扛住流量然后在流量平稳之后再来 优化架构。

## 单体架构

通常来说，如果一个war包或者jar包里面包含了一个应用的所有功能，则我们称这种 架构为单体架构。很多传统的互联网公司或者创建型公司基本都会采用这样的架构，因 为这样的架构足够简单，能够快速开发和上限，对于项目初期用户量不大的情况，也足 以支持业务的正常运行。

## 集群及垂直化

当公司开发的产品（采用单体架构）被越来越多的人使用，可能会面临以下挑战：

1. 当用户量越来越大，网站的访问量不断增长，导致后端服务器的负载越来越高。当服务器的负载越来越高的时候，如果不进行任何处理，用户在网站上操作的响应会越来越慢，甚至出现无法访问的情况，对于非常注重用户体验的互联网产品来说，这是无法容忍的。

2）用户量大了，需要满足不同用户的需求来留住用户，使得业务场景越来越多并且越

来越复杂。业务场景越多越复杂，就意味着war包中的代码量会持续上升，并且各

个业务代码之间的耦合度也会越来越高，后期的代码维护和版本发布涉及的测试和 上限，也会很困难。

因此，我们可以从两个方面进行优化：

1. 通过横向增加服务器，把单台机器变成多台机器的集群。
2. 按照业务的垂直领域进行拆分，减少业务的耦合度，以及降低单个war包带来的伸缩性困难问题。、

例如，将一个单体架构的电商系统按照业务维度进行了垂直拆分：用户子系统、库存子 系统、商品子系统，每个子系统由不同的业务团队负责维护并且独立部署。同时，我们 针对Tomcat服务器进行了集群部署，将多台Tomcat服务器通过网络进行连接组合， 形成一个整体对外提供服务，这样做的好处是能够在改变应用本身的前提下，通过增加 服务器来进行水平扩容从而提升整个系统的吞吐量。考虑到Tomcat服务器能够承载的 流量大了之后，如果流量都传导到了一个数据库上，会给数据库造成比较大的压力，所 以对数据库进行分库分表操作，也采用分而治之的思想。

**插图**

## SOA

虽然采用了集群和垂直化的架构方式，但是也存在一些问题，如以下场景：

1. 将商品系统采用集群和垂直化架构划分为了三个子系统：用户子系统，库存子系统、订单子系统、商品子系统。用户在执行下单操作时，需要检查商品的库存，只有当库存充足的情况下才会提交订单，但这个检查库存的逻辑是放在订单子系统中还是库存子系统中呢？在整个系统中，一定会存在非常多类似的共享业务的场景，这些业务场景肯定会被重复创建，从而产生非常多的冗余的业务代码，这些冗余代码的维护成本随着时间的推移越来越高，能不能把这些共享业务逻辑抽离出来形成可重用的服务呢？
2. 在一个集团公司下通常有很多子公司，每个子公司都有自己的业务模式和信息沉淀，各个子公司之间不进行交互和共享，这个时候每个子公司虽然能够创造一定的价值，但是由于子公司信息不是互联互通的，彼此之间形成了信息孤岛，使得价值无法最大化。

基于这些问题，就引入了SOA（Service-Oriented Architecture），也就是面向服务的架 构，它的核心目标是把一些通用的、会被多个上层服务调用的共享业务提取成独立的基 础服务，这些提取出来的共享服务相对来说比较独立，并且可以重用。在SOA中，服 务是最核心的抽象手段，业务被划分为一些粗粒度的业务服务和业务流程。

在SOA中，会采用ESB（企业服务总线）来作为系统和服务之间的通信桥梁，ESB本 身还提供服务地址的管理、不同系统之间的协议和数据格式转化等。调用端不需要关心 目标服务的位置，从而使得服务之间的交互式动态的，这样做的好处是实现了服务的调 用者和服务的提供者之间的高度耦合。总得来说，SOA主要解决的问题是：信息孤岛、 共享业务的重用。

**插图**

## 微服务架构

业务系统实施服务化改造后，原本共享的业务被拆分形成可复用的服务，可以在最大程 度上避免共享业务的重复建设等问题。那么被拆分的服务是否也需要以业务功能为维度 来进行拆分和独立部署，以降低业务的耦合、提升容错性呢？微服务就是这样一种解决 方案，SOA（面向服务）微服务本质上都是服务化思想的一种体现。如果SOA是面向 服务开发思想的雏形，那么微服务就是针对可重用业务服务的更进一步优化，可以把 SOA看成微服务的超集，也就是多个微服务可以组成一个SOA服务。 伴随着服务粒 度的细化，会导致原本10个服务可能拆分成了100个微服务，一旦服务规模扩大就意 味着服务的构建、发布、运维的复杂度也会成倍增加，所以实施微服务的前提是软件交 付链路及基础设施的成熟化。因此微服务的本质是服务化思想的最佳实践方向。

由于SOA和微服务两者的关注点不同，造成了两者有非常大的区别：SOA关注的是服 务的重用性、解决信息孤岛的问题。微服务关注的是解耦，虽然解耦和可重用性从特定 角度来看是一样的，但本质上是有区别的，解耦是降低业务之间耦合度，而重用性关 注的是服务的复用。并且微服务会更多地关注在DevOps的持续交付上，因为服务粒度 细化之后使得开发运维变得更加重要，因此微服务与容器化技术的结合更加紧。 （DevOps（Development和Operations的组合词）是一组过程、方法与系统的统称， 用于促进开发（应用程序/软件工程）、技术运营和质量保障（QA）部门之间的沟通、 协作与整合。）

将每个具体的业务服务构成可独立运行的微服务，每个微服务只关注某个特定功能，服 务之间采用轻量级通信机制REST API进行通信。

**插图**

但实际上，微服务到底要拆分到多大的粒度没有统一的标准，更多的时候是需要在粒 度和团队之间找平衡的，微服务的粒度越小，服务独立性带来的好处就越多，但是管理 大量的微服务也会越复杂。

# 微服务架构带来的挑战

从单体架构到微服务架构，技术架构随着产品的复杂度和访问的压力增大不断进行变化，

最终的目的都是更好地服务业务，使得用户在使用产品时获得更好的体验。

## 微服务架构的优点

微服务架构有很多好处，几个比较突出的优点如下：

1. 复杂度可控：通过对共享业务服务更细粒度的拆分，一个服务只需要关注一个特定的业务领域，并通过定义良好的接口清晰表述服务边界。由于体积小、复杂度低，开发、维护会更加简单。
2. 技术选型更灵活：每个微服务都由不同的团队来维护，所以可以结合业务特性自由选择技术栈。
3. 可扩展性更强：可以根据每个微服务性能要求和业务特定来对服务进行灵活扩展，比如通过增加单个服务的集群规模，提升部署了该服务的节点的硬件配置。
4. 独立部署：由于每个微服务都是一个独立运行的进程，所以可以实现独立部署，当某个微服务发生变更时不需要重新编译部署整个应用，并且单个微服务的代码量更小，使得发布更加高效。
5. 容错性：在微服务架构中，如果摸一个服务发生故障，我们可以使故障隔离在单个服务中，其他服务可以通过重试、降级等机制来实现应用层面的容错。

## 微服务器架构面临的挑战

微服务虽然具备非常多的优势，但是也给我们的开发工作带来了非常大的挑战。在拥抱 微服务架构过程中，我们经常会遇到数据库的拆分、API交互、大量的微服务开发和维 护、运维等问题，会面临以下一些挑战：

1. 故障排查：一次请求可能会经历多个不同的微服务的多次交互，交互的链路可能会比较长，每个微服务会产生自己的日志，在这种情况下如果出现一个故障，开发人员定位问题的根源会比较困难。
2. 服务监控：在一个单体架构中很容易实现服务的监控，因为所有的功能都在一个服务中。而在微服务架构中，服务监控开销会非常大，可以想象一下，在几百个微服务组成的架构中，我们不仅要对整个链路进行监控，还需要对每一个微服务都实现一套类似单体架构的监控。
3. 分布式架构的复杂性：微服务本身构件的是一个分布式系统，分布式系统涉及服务之间的远程通信，而网络通信中网络的延迟和网络故障是无法避免的，从而增加了应用的复杂度。
4. 服务依赖：微服务数量增加之后，各个服务之间会存在更多的依赖关系，使得系统整体更为复杂。
5. 运维成本：在微服务中，需要保证几百个微服务的正常运行，对于运维的跳转是巨大的，比如单个服务流量激增时如何快速扩容、服务拆分之后导致故障点增多如何处理、如何快速部署和统一管理众多的服务等。

# 如何实现微服务架构

不管是单体架构还是微服务架构，本质上都是为了更好地支持业务的发展。

## 微服务架构图

为了解决微服务面临的挑战，就必须引入更多的技术。简单整理实现微服务架构需要考 虑的功能，可以得到微服务架构图，如下：

**插图**

## 微服务架构下的技术挑战

微服务架构主要的目的是实现业务服务的解耦。随着公司业务的高速发展，微服务组件 会越来越多，导致服务与服务之间的调用关系越来越复杂。同时，服务与服务之间的远 程通信也会因为网络通信问题的存在变得更加复杂，比如需要考虑重试、容错、降级等 情况。那么这个时候需要进行服务治理，将服务之间的依赖转化为服务对服务中心的依 赖。除此之外，还需要考虑：分布式配置中心、服务路由、负载均衡、熔断限流、链路 监控。