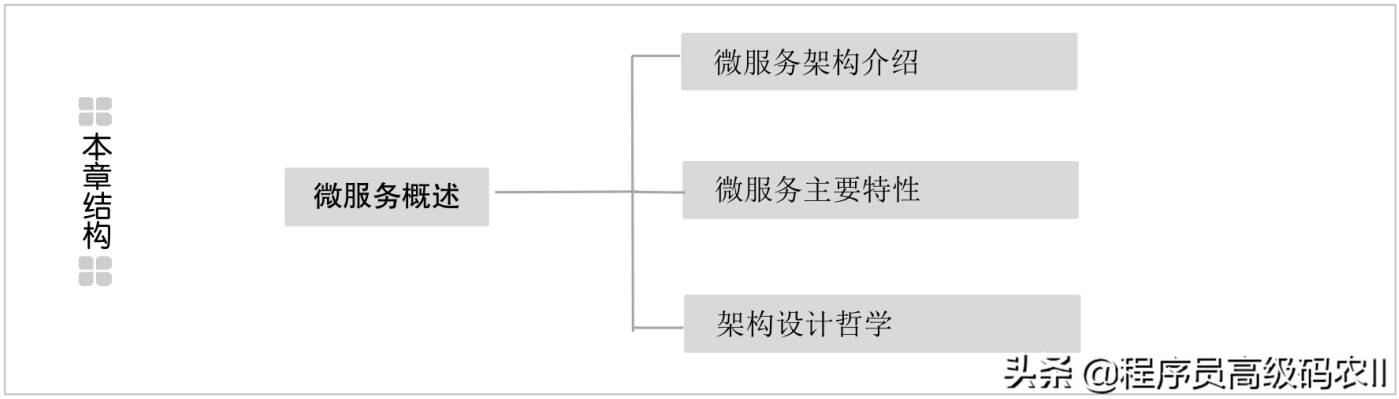
微服务架构深度解析微服务定义是什么？微服务与云原生有何关联？

原创

该用户快成仙了 于 2021-08-12 22:23:25 发布 226 收藏 1

微服务概述



微服务的概念来源于Martin Fowler 的一篇知名博文 ：MicroServices。在博文中，“微服务架构”这个术语用来描述一种将软件应用程序设计为可独立部署的服务套件的特定方式。

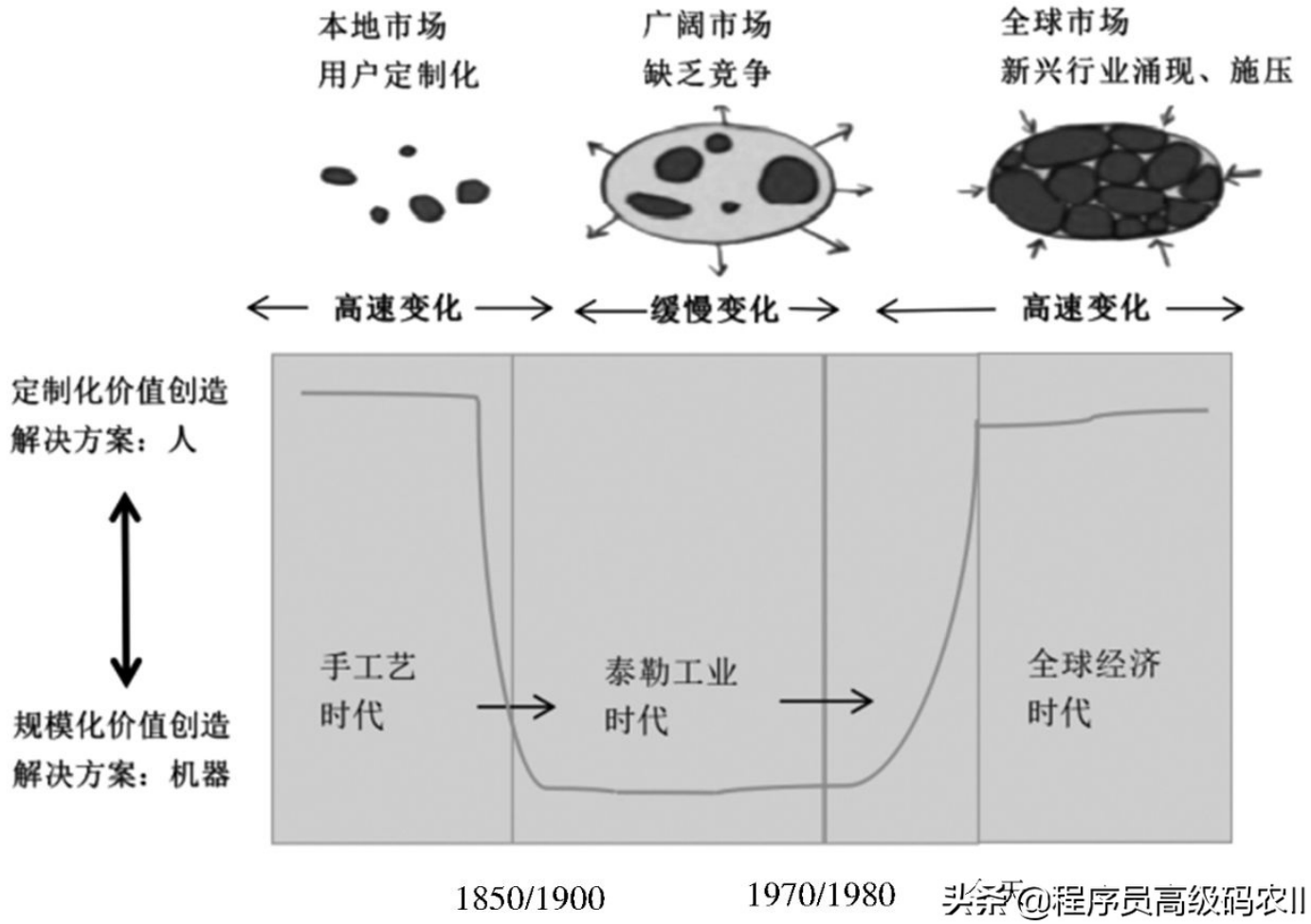
“细粒度自治服务”“自动化部署”“围绕业务能力”“端点智能”“语言和数据的分散控制”，从这些描述微服务架构特征的术语中，我们发现了一种越来越吸引人的软件系统风格。

微服务架构介绍

背景介绍

目前不仅各大互联网公司已经在大规模地应用微服务架构，而且传统行业也逐渐接受了这种架构模式，纷纷开始采用微服务架构构建业务系统。为什么微服务架构会如此受欢迎？微服务架构是设计而来还是演变而来的呢？要了解这些问题，我们需要从现代经济模式和企业组织架构入手来了解微服务架构崛起的时代背景。

Niels Pflaeging在Organize for Complexity一书中通过“浴缸曲线”（见下图）将西方从20世纪到现在的经济模式划分为三个时代：本地市场和用户定制化的“手工艺时代”；通过机械规模化提升效率和比拼成本，市场广阔而缺乏竞争的“泰勒工业时代”；以知识工人为主体，新兴行业涌现、施压，从而带来市场需求快速变化的“全球经济时代”。



在“手工艺时代”，产品的价值创造完全取决于掌握技艺的手工艺者，局部市场、高度动态化、定制化是这个时代的特点，但这种模式很难做到规模化地生产和持续地输出价值。在“泰勒工业时代”，主流的组织是上传下达的“命令控制型”组织，更适合简单、重复的规模化生产，但这种组织架构的不足是对市场响应慢，在应对复杂变化方面十分脆弱。而在“全球经济时代”，由新兴行业带领，逐渐兴起更多扁平或分散的复杂的自适应组织，这种架构模式更加倾向于跨职能混搭和协作，和市场直接对接，可以快速灵活地响应市场变化。

可以说，组织、系统架构和技术之间隐含着映射关系。从组织所采用的技术栈和架构特征可以快速推断出组织的业务模式和组织架构方式。单体架构、垂直架构、集中式数据库常见于泰勒型组织；而云计算、微服务、DevOps等技术更加适合于复杂的自适应组织。同时，技术架构反过来也受限于强势的组织架构和企业管理文化约束。回顾早期的软件系统，企业采用单体架构可以快速满足业务的简单诉求，然而随着项目规模的扩大、业务模块的耦合、组织人员的膨胀，使得单体架构冗余而数量庞大的代码越来越无法适应企业灵活应对变化的需求，大量紧耦合的代码导致应用的模块界限日益模糊，业务发展急需匹配高可用、可扩展、隔离性好、复用性强的应用系统架构，而传统的单体架构显然无法满足企业的需求。

在Microservice Pattens一书中，作者使用餐饮应用FTGO（Foodto Go）举例，说明了系统是如何一步步走向单体地狱的。系统的过度复杂使得业务逻辑耦合、开发速度缓慢、交付周期长、难以扩展，这给开发人员带来了极大的挫折感，生产效率也随之大幅下降。单体架构在运行状态下，出现故障难以隔离，局部异常问题往往会影响整个系统的正常运行，最终导致整体服务的不可用，给业务人员造成的收入损失、给客户造成的糟糕体验都让人无法忍受。

正是在这样的时代背景和业务诉求下，微服务架构成为了解决复杂问题的灵丹妙药。微服务架构在应对需求的变化、容错处理、服务复用及扩展、提升开发效率、简化交互等方面都有明显的优势。同时，敏捷、DevOps、持续集成/持续交付、容器技术、Spring Cloud框架、轻量级服务、领域驱动设计等的涌现也为微服务架构的发展奠定了基础。

综上所述，持续快速响应市场、高度动态化、应对复杂场景的能力已经成为企业的核心竞争力，企业越来越需要一个能够面对变化、并且能够主动拥抱变化的软件架构，而微服务架构正是在这样的时代大背景下逐渐发展壮大起来的。

微服务的定义

微服务并没有一个明确的官方定义，它可以解释为一种架构编程思维，更多地被描述为一种架构风格。微服务架构的概念可以说来源于技术专家多年的工作积累和最佳实践总结，是通过不断发展、演进逐渐形成的。

架构演进论在“技术雷达”里，微服务最早以“Micro-service”，而非“MicroService”出现，从架构演进的角度来说，微服务是从SOA（Services Oriented Architecture，面向服务架构）发展演进而来的，是更先进的细粒度的SOA实现方式。

SOA和微服务本质上都是分布式的架构模式，但是传统的基于ESB（Enterprise Service Bus，企业服务总线）的SOA架构和微服务架构要解决的问题其实并不相同。在J2EE（Java 2 PlatformEnterprise Edition，Java企业级应用平台）的企业架构中，ESB在异构系统的集成方面发挥功能。而微服务架构更多地采用轻量级的通信协议，围绕业务进行服务拆分、解耦、复用、隔离，实现细粒度的分布式服务构建和管理。

微服务架构对传统SOA架构最大的改变是强化端点（Endpoints）和弱化通道（Dumb Pipe），抛弃了ESB过度复杂的业务规则、编排、消息路由等功能。微服务强调服务的自治性，服务应用从一开始就可以独立地进行开发和演进。另外，微服务间的通信应尽可能地轻量化，微服务在SOA的基础上更加关注细粒度服务的独立性、可扩展性、伸缩性和容错性等。

最佳实践论

微服务的另外一种定义来源于架构大师们多年的最佳实践总结，Martin Fowler于2014年在他自己的博文中首次提出微服务的定义，概括总结如下：

“微服务架构”是一种将单个应用程序作为一套小型服务开发的方法，服务之间相互协调、互相配合，每个服务运行在其独立的进程中，并以轻量级机制（通常采用HTTP协议）进行交互通信。这些服务是围绕业务功能构建的，它们可以通过全自动部署机制进行独立部署。微服务采用去中心化的管理理念、可以用不同的编程语言编写，并使用不同的数据存储技术。总结起来微服务有以下几大特征：

- 通过服务组件化。
- 围绕业务能力组织。
- 是产品不是项目。
- 智能端点和哑管道。● 去中心化治理。
- 去中心化数据管理。
- 基础设施自动化。
- 为失效设计。
- 演进式设计。

可以看出，Martin Fowler试图将微服务定义为一个一般化的架构“最佳实践”集合。微服务的这些特征也很好地融合了领域驱动设计、自动化、DevOps、容器等先进的技术实践、架构理念和方法论。

与此同时，微服务的架构风格被描述为一种可以实现业务功能松散耦合（Loosely Coupled）的、具备一定服务边界（BoundedContext）的服务集合。这种架构风格使得大型、复杂的应用实现CI/CD（Continuous Integration/Continuous Delivery，持续集成/持续交付）成为可能，并且技术栈可以独立发展演进。

归根结底，微服务本质上还是一种分布式系统架构。它强调系统应该按照业务领域边界做细粒度的拆分和部署；它剔除了SOA中的企业服务总线，使用轻量级、标准化的HTTP（REST API）协议进行交互集成。微服务架构有利于应用应对业务的快速变化和规模化发展，通过对软件复杂性的有机治理，使系统易于有序化重构及扩展。

微服务与云原生

云原生（Cloud Native）可以理解为一系列技术及思想的集合，既包含微服务、容器等技术载体，也包含DevOps（开发与运维的合体）的组织形式和沟通文化。企业采用基于云原生的架构进行构建、运行、管理现代应用的技术模式能够平滑而快速地将业务迁移到云上，享受云的高效性和按需伸缩的能力。

云原生的提出

“效率”：天下武功、唯快不破，面对激烈的市场竞争，企业把服务产品快速交付的能力作为制胜的法宝。云原生架构的提出与应用的快速开发、快速交付的能力密不可分。作为对比，与传统企业为应用提供和部署软件按周、按月来计算，互联网公司经常在一天内可以进行上百次发布。这种快速迭代和部署是建立在云基础设施和自动化持续交付能力之上的。

“弹性”：随着用户规模和需求的增长，我们的应用需要能够快速扩展，提高服务能力。传统企业依靠购买硬件的方式来提升和扩展服务能力，而云原生架构可以通过虚拟化的技术实现按需扩展，动态地扩展服务实例以满足计算、存储、服务资源的弹性需求。

“可靠”：服务仅仅做到快速交付和弹性扩展是远远不够的，还需要兼顾系统的稳定性、可用性、持久性，而这种特征与服务的容错能力、故障隔离能力、可视化能力、系统快速恢复能力紧密相关，也就是系统所谓的“反脆弱”能力。现代应用如果想在这些“非功能需求”上得到保障，就需要采用云原生技术和云原生基础设施保障服务的高可用性。

云原生与微服务

“云原生”的本质和目标是一种应用模式，它能够帮助企业快速、持续、可靠、规模化地交付软件，其中关键的支撑组件总结如下。

● 容器化的抽象封装：标准化代码和服务，每个部分（应用程序、进程等）都封装在自己的容器中，有助于复用和资源隔离。实现方式代表有Docker、rkt。

● 动态管理：通过集中式的编排和调度系统来动态管理及优化资源。实现方式代表有Kubernetes、Swarm和Mesos。

● 面向微服务：应用程序基于微服务架构，显著提升开发效率、提高架构演进的灵活性和可维护性。实现方式代表有SpringBoot、Spring Cloud。

从云原生的代表组件可以看出，云原生的主要组成技术包括容器、服务编排管理和微服务技术。云原生可以说是从概念上统一了构建、交付、运行现代应用的最佳实践集合。在运行环境上，强调应用程序的运行环境是以容器和Kubernetes为主的云基础设施；在流程管理上，主要配合使用持续集成、持续发布以及DevOps能力；在软件开发上，基于微服务架构构建现代应用程序和软件。虽然微服务架构也可以运行在传统虚拟机或物理机上，但是微服务架构的最佳运行载体是以容器为代表的云原生环境。

云原生架构

云原生架构的思想和特征可以通过云原生12要素表达，我们在后面的章节中会详细讲解12要素的主要内容，从实践的角度，作为最早践行Cloud Native的Netflix，在云原生架构的目标、原则和措施等方面进行了详尽描述：

● 不可变性。使用易失效的基础设施来构建高敏捷（HighlyAgile）、高可用（Highly Available）服务。其目的是为了提高伸缩性（Scalability）、可用性（Availability）、敏捷性（Agility）、效率（Efficiency）。

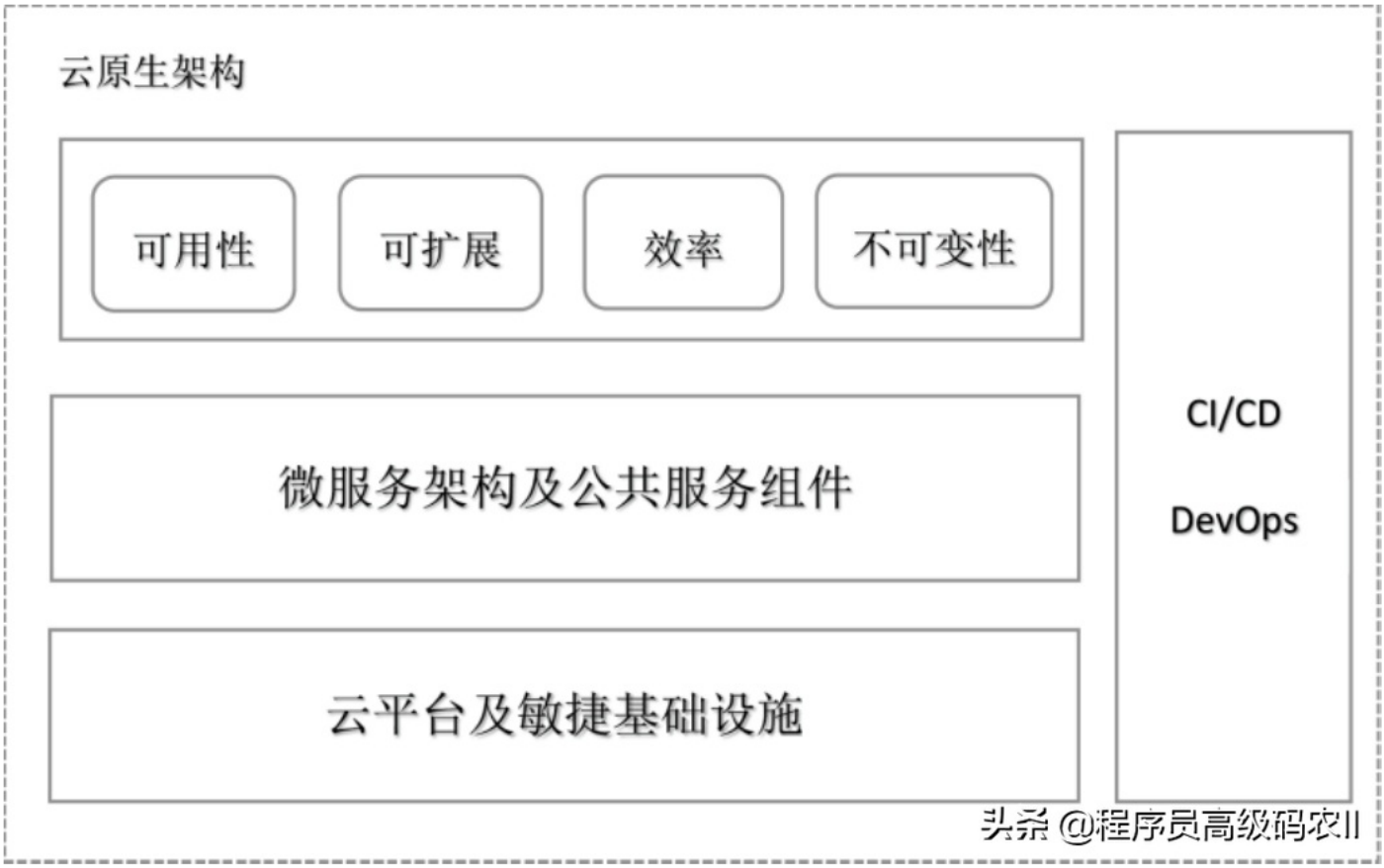
● 关注点分离。通过微服务架构实现关注点分离，避免出现“决策瓶颈”。实际上，实现关注点分离有助于提升系统的可扩展性和可用性。

● 反脆弱性。默认所有的依赖都可能失效，在设计阶段就要考虑如何处理这些失效问题。为了让系统更强壮，Netflix会不断地攻击自己、主动破坏，以提醒技术人员系统要进行反脆弱性设计。

● 高度信任的组织。Netflix基于信任的管理风格，相信自己的员工可以做出正确的决策，倡导给基层员工自主决策权。

● 共享。在Netflix，管理是比较透明的，共享能够促进技术人员的成长。

结合云原生的特征和云原生的架构实践，我们将云原生架构图描述如下（见下图），从架构层面来说，Cloud Native是构建在不可变基础设施（以容器技术为代表）和以微服务架构技术为基础的分布式系统之上的，这里的云包括私有云、公有云和混合云。微服务架构作为云原生概念的核心组成部分，本质上就是保证我们更好地适应云环境下的高效开发和运维。



云原生成熟度

对于系统采用云原生架构的程度，我们可以用云原生成熟度模型进行精确的等级划分和判断，成熟度模型一共包含4个等级（从Level0到Level 3成熟度逐级递增），从这些等级划分可以看到开发和运行应用程序的原则、模式和具体技术实践。如果你所在公司正在进行微服务的改造或者云原生架构的迁移，可以参考如下表所示的成熟度模型正确地评估你所在组织的云原生架构等级。

级别	描述	关键技术
Level 3：适应性	应用程序能够以全自动的方式检测或预测变化并对其做出反应。 应用程序管理和控制功能从应用程序中抽离出来，或者使用外部应用程序控制服务	人工智能分析决策 智能化运维 高度自动化 成熟的公共服务和基础设施
Level 2：抽象	应用程序必须与基础架构完全分离。 抽象出应用程序蓝图、部署策略、扩展策略、关联和布局规则等。 应用程序服务必须是弹性的和可适应的。应用程序也应该被设计，以便一个服务的失败不会级联到其他服务。 应用程序由多个服务组成，每个服务的设计都是弹性的、可适应的、可组合的、最小的和完整的	微服务治理平台 容器资源调度平台 CI/CD发布平台 分布式存储、消息中间件 分布式数据库
Level 1：松耦合	应用服务满足隔离、自治特性并与运行环境分离。 应用服务代码与配置文件存储和数据管理层解耦分离。 应用服务与网络依赖分离，可以实现动态服务注册与发现	微服务框架 初步的持续交付能力

级别	描述	关键技术
Level 0：虚拟化	应用程序运行在虚拟机或云实例上。 应用程序创建不可变的应用程序映像	虚拟化隔离 模块化