第 1 章

微服务的前世今生

本章将详细探讨微服务架构的起源和发展过程，深入分析微服务架构的优势与面临的挑战，并展望其未来的发展方向。通过深入了解微服务架构的发展历程和优势，读者将能够更深刻地理解微服务架构的核心理念及其适用的应用场景。

1.1 软件架构的演化之路

软件架构是软件系统的基础和核心，它决定了系统的整体结构、功能特性、性能、可维护性和可扩展性等关键属性。随着软件技术的持续迭代和更新，软件架构也在不断演进，以适应业务需求的增长和技术创新的方向。软件架构的演进既是技术发展的自然结果，也是为了满足不断变化的业务需求和用户期望而进行的积极探索和变革。

在信息技术迅猛发展的时代，软件架构作为软件开发的中心蓝图，经历了漫长且不断变革的发展过程。它的演进不仅反映了技术的进步，也积极回应了业务需求和用户期望的变化。优秀的软件架构不是一朝一夕就能形成的，而是在不断的实践和探索中逐渐演化而来的。因此，深入学习软件架构及其演化历程至关重要。架构的演变过程主要经历了如图 1-1 所示的几个阶段。

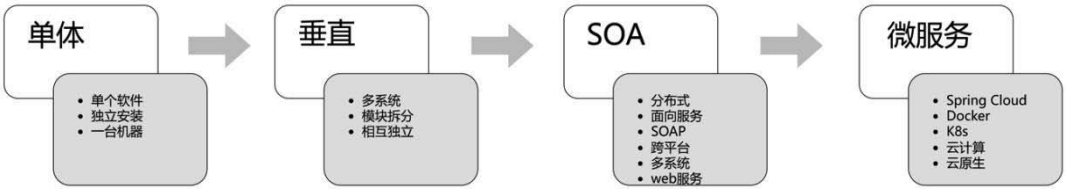

图 1-1　大型系统架构的演化之路

1. 单体应用架构

互联网发展初期，系统业务简单，流量较小。因此，一个系统可以实现所有业务功能，系统的开发、部署和维护都相对简单。在这一阶段，对技术架构的要求通常不高，单体架构能够满足需求。例如，早期的电商系统的所有代码、业务逻辑和功能模块都集中在一个项目中，如图 1-2 所示。

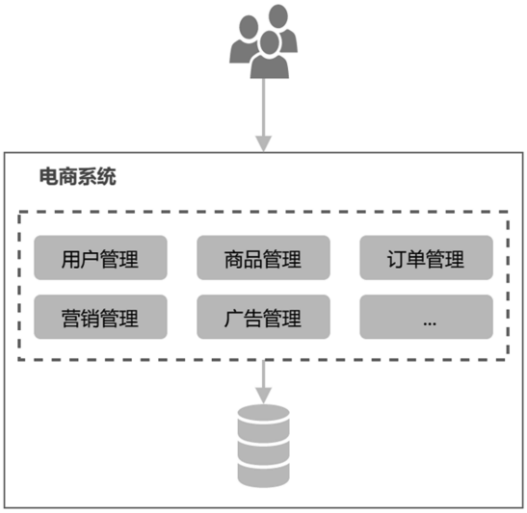

图 1-2　单体架构图

单体应用架构的优点在于：架构简单，开发成本低，维护容易。然而，它的缺点也显而易见：由于所有功能都集成在一个工程中，模块之间的耦合度高，这严重影响了系统的扩展性。

2. 垂直应用架构

随着系统业务逐渐复杂化，系统数据量和访问量逐渐增大，项目参与人员逐渐增多，单体架构的系统变得越来越臃肿，代码也越来越难以维护。这导致了开发效率低下，测试成本增加和运维过程复杂。因此，将系统按功能模块拆分势在必行，垂直应用架构应运而生。

垂直应用架构的核心思想是将原来的单体系统拆分成多个独立的应用。比如，我们可以将上述电商系统拆分成如下几个模块（见图 1-3）：

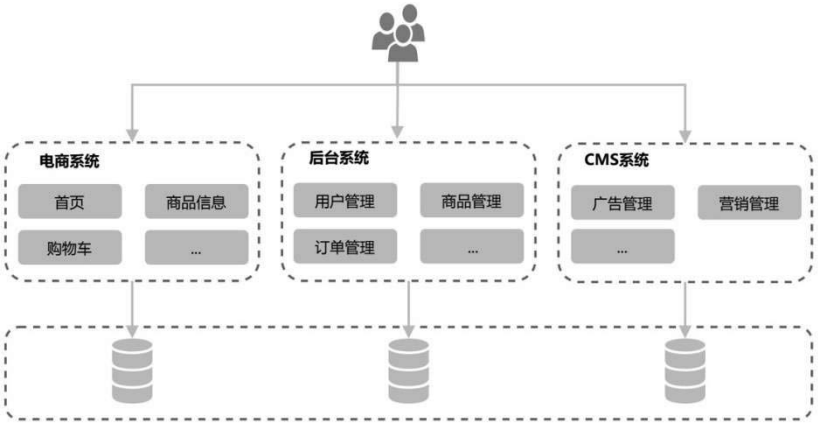

图 1-3　垂直应用架构图

如图 1-3 所示，各子系统之间相互独立，互不影响，从而使业务迭代变得更加高效。不

过，因为各系统之间无法互相调用，所以部分模块的功能需要在不同系统中重复实现。比如，用户信息模块可能在订单系统和用户系统中都需要实现。

3. SOA 架构

随着垂直应用数量的增加，重复的业务代码也会相应增多，导致即使是小的需求变更也可能需要修改多个应用。这不仅增加了维护成本，还提高了开发风险。因此，有必要将重复的业务逻辑和代码抽象并提取出来，构建成一个统一的业务层，作为独立的服务存在。这种做法是面向服务架构（Service-Oriented Architecture，SOA）的核心理念。

SOA 架构允许根据需求通过网络对松散耦合的、粗粒度的应用组件（即服务）进行分布式的部署、组合和使用。例如，订单系统、后台管理系统、产品系统、广告系统等不同的功能模块可以被定义为独立的服务。这些服务通过企业服务总线（Enterprise Service Bus，ESB）相互连接，实现高效的通信和数据交换。具体架构如图 1-4 所示。

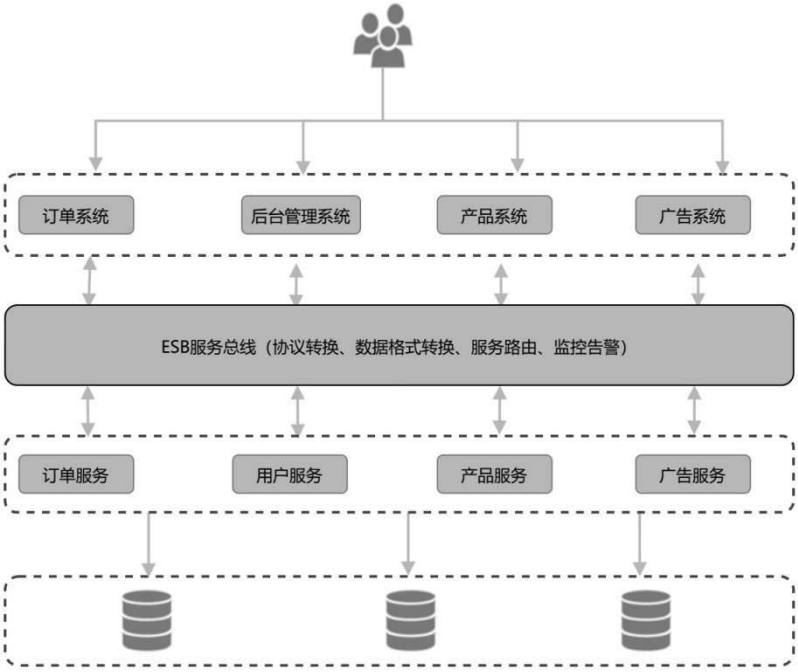

图 1-4　SOA 架构图

4. 微服务架构

随着移动互联网的飞速发展，SOA 架构变得越来越臃肿，已经无法满足移动互联网时代对于系统业务的快速迭代、高并发、高扩展的要求。因此，微服务架构应运而生。

微服务架构在某种程度上也是面向服务的架构，但是，它更加强调服务的彻底拆分和业务彻底的组件化和服务化，将原有的业务系统拆分为多个可以独立开发、设计和运行的小应用。这些小应用之间通过 RESTful 或 RPC（Remote Procedure Call，远程过程调用）进行交互和集成。具体架构如图 1-5 所示。

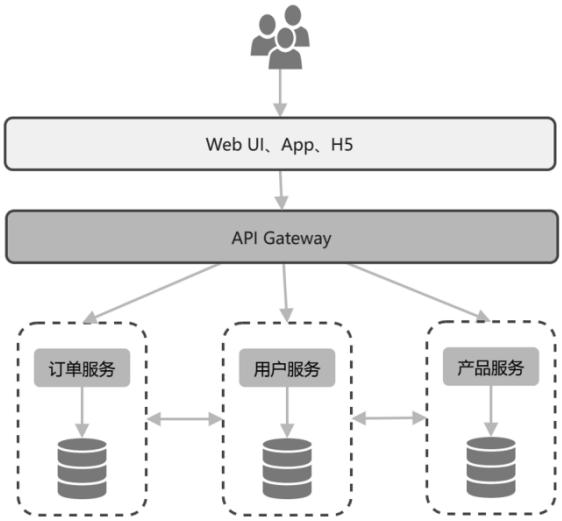

图 1-5　微服务架构图

通过对服务进行原子化拆分，实现独立打包、部署和升级，可以确保每个微服务职责清晰，方便扩展。服务之间的松散耦合有助于避免因一个模块的问题导致整个系统崩溃。当然，微服务架构的技术成本高（容错、分布式事务等），随着服务的增加，运维的压力也随之增大。因此，需要一个统一的服务治理解决方案来应对这些挑战。

5. 总结

尽管单体架构、垂直架构和 SOA 架构都存在一些不足，微服务架构也并非万能的解决方案。正如业界常说的，没有最好的架构，只有最适合的架构。选择合适的架构必须综合考虑业务复杂度、数据规模、团队的技术能力、时间成本等因素，从全局出发，寻找最能满足项目需求的架构方案。

1.2　什么是微服务

磨刀不误砍柴工。要真正了解微服务架构，首先需要了解什么是微服务。微服务背后又

有着什么样的故事？接下来，让我们走进微服务的世界，探究它的前世今生。

1. 微服务的定义

微服务架构最早由 Martin Fowler 和 James Lewis 于 2014 年提出，是一种将单个应用程序构建为一系列小型服务的架构风格。每个微服务在独立的进程中运行，并通过轻量级的通信机制（例如 HTTP API）与其他服务进行协作和通信。每个微服务都围绕特定的业务功能构建，具有高度自治性，能够独立开发、部署、扩展和维护。

微服务架构允许将大型复杂的应用程序分解为多个专注于单一任务或业务功能的小型服务。根据业务需求和负载情况，这些服务可以独立伸缩和升级。这种架构模式提高了开发团队的敏捷性，加快了应用的开发和迭代速度，同时增强了系统的灵活性、可扩展性和容错性。

2. 微服务的特性

微服务是一种架构风格，它通过使用一组功能独立的服务来开发和构建应用的。该架构将应用程序拆分成一组小型、独立的服务。微服务架构具有以下特性。

- 服务自治性：每个服务都是独立的，拥有自己的数据存储、业务逻辑和用户界面，能够独立部署和运行，不会影响其他服务的运行。
- 松耦合：每个服务都是相互独立的，它们之间通过 API 进行通信，不需要共享代码或数据库。
- 可替换性：由于每个服务都是独立的，因此可以轻松地替换或升级其中的一个服务，而不会影响整个系统的运行。
- 可伸缩性：每个服务都可以独立扩展，可以根据需要添加或删除服务，从而提高系统的性能和可靠性。
- 弹性设计：由于每个服务都是独立的，因此可以通过在运行时动态添加或删除服务来应对系统负载的变化。
- 去中心化管理：每个服务都有自己的团队负责开发和维护，因此可以更好地分散风险和责任。
- 技术异构性：每个服务都可以使用不同的技术栈和编程语言，因此能够选择最适合特定任务的工具。

综上所述，微服务架构具有自治性、松耦合、可替换性、可伸缩性、弹性设计、去中心化管理和技术异构性等特性，这些特性使得微服务架构成为构建大型、复杂应用程序的理想选择。

3. 微服务的优势

微服务具有以下显著优势。

（1）独立部署与扩展：每个微服务都能独立部署，可依据自身的负载及需求灵活扩展，从而有效提升资源利用率和系统性能。

（2）技术选型灵活：各微服务可根据其特定业务需求自由选择适宜的技术栈，充分发挥不同技术的优势。

（3）敏捷开发与迭代：开发团队能够专注于单个微服务，降低了代码库的复杂度和团队间的协调成本，加快了开发速度，更契合敏捷开发理念。

（4）高可用性保障：单个微服务的故障不会累及整个系统，极大地提高了系统的整体可用性。

（5）易于维护与更新：较小的服务代码库更易于理解和维护，更新服务时风险相对较小。

（6）更好的容错性：某一微服务出现故障时，仅影响其自身功能，不会导致整个系统崩溃，便于快速定位和修复问题。

（7）促进团队自治：不同团队可负责不同的微服务，减少团队间的依赖和冲突，提升团队的自主性和责任感。

（8）适应业务变化：能够快速响应并适应不断变化的业务需求，通过新增或修改微服务即可实现新功能或者业务调整。

4. 微服务的劣势

微服务架构虽然具备众多吸引人的特质，但是也存在着一些问题。当使用微服务架构时，我们可能会面临如下挑战。

- 分布式系统复杂性增加。服务间的通信、协调以及分布式事务的处理等，使得系统的整体复杂性提升，需要解决诸多分布式系统特有的难题。
- 开发与运维成本上升。需要构建完善的自动化工具和基础设施来支持服务的开发、部署、监控和治理，这会增加技术投入和人力成本。
- 数据一致性挑战加剧。各个微服务管理自身的数据，容易导致数据一致性难以保证，需要额外的机制和策略来维护数据的一致性。
- 测试难度增大。对整个系统的集成测试变得更加复杂，需要确保多个微服务之间交互正常。
- 服务间的通信开销不可忽视。微服务之间的频繁通信可能会带来一定的性能损耗。
- 部署与管理复杂度提高。众多微服务的部署和管理需要高效的流程和工具，否则容易出现混乱。

- 对团队协作要求更高。不同团队负责不同微服务的开发和维护，需要良好的沟通和协作机制，以确保服务之间的集成和协调顺畅。

1.3　为什么需要微服务

前面介绍了微服务的本质，相信读者对微服务已经有了一个大致的了解。读者可能会问：为什么需要微服务？它解决了什么问题？下面我们来回答这些问题。

1. 传统软件架构面临的挑战

随着技术的持续进步，软件系统架构设计正面临着前所未有的挑战。这些挑战包括应用场景的不确定性、大规模数据处理、云计算和虚拟化等新技术的融入。具体而言，传统软件架构所面临的挑战主要表现在以下几个方面。

- 大规模：现代软件系统常常需要处理庞大的数据量和用户基数，这就要求系统必须具备高性能、高可靠性和高可扩展性。
- 复杂性：随着功能的不断扩展，软件系统的复杂性也在增长。因此，需要采用合适的架构和设计模式来有效管理和维护系统。
- 安全性：面对日益增多的网络攻击，软件系统必须具备强大的安全性能，以确保用户数据和隐私的安全。
- 可维护性：软件系统应具有良好的可维护性，这样在系统发生故障或需要进行升级时，可以迅速进行修复和更新。
- 敏捷性：为了能够迅速响应用户需求和市场变化，软件系统需要具备敏捷性。
- 数据管理：数据量的激增要求软件系统拥有高效的数据管理能力，以便快速地存储、检索和分析数据。
- 云化：云计算技术的兴起使得软件系统越来越多地需要具备云化能力，以支持在云端的部署和运行。

综上所述，当前软件系统面临的挑战涵盖了大规模处理、系统复杂性、安全性保障、可维护性、敏捷性、数据管理以及云化等多个方面。为应对这些挑战，必须采用合适的技术和架构策略。

2. 传统架构的问题

早期的软件架构通常以典型的 MVC 框架为基础构建而成，涵盖前端（包括 Web 端和手机端）、中间业务逻辑层以及数据库层。此架构在业务逻辑和数据规模相对较小时，运行状

况良好。然而，伴随系统规模的持续拓展，逐渐显露出如下问题。

- 复杂性逐渐增加：例如，有的项目有几十万行代码，模块间界限模糊，逻辑混乱。代码越多，系统复杂度越高，问题也越难解决。
- 技术债务逐渐增加：员工流动在公司中是再正常不过的现象。有些员工在离职前对代码质量缺乏自我约束，留下来很多“坑”。由于单体项目的代码量过于庞大，这些“坑”往往难以被发现，给新员工带来了极大的困扰。人员流动越频繁，留下的“坑”就越多，技术债务也随之增加。
- 部署速度逐渐变慢：这个很好理解。单体架构的模块非常多，代码量巨大，导致项目部署时间不断延长。有的项目启动就需要一二十分钟，这是多么恐怖的事情。启动几次项目，一天的时间就过去了，留给开发者开发的时间就非常少了。
- 阻碍技术创新：比如，以前的某个项目是使用 struts2 架构开发的，各模块之间有着千丝万缕的联系，代码量大，逻辑不够清晰。如果现在尝试用 Spring MVC 来重构这个项目，难度和成本都非常高。因此，很多时候公司不得不硬着头皮继续使用旧的 struts 架构，阻碍了技术的创新。
- 无法按需伸缩：比如电影模块是 CPU 密集型的，而订单模块是 IO 密集型的。如果需要提升订单模块的性能，比如加大内存、增加硬盘，但由于所有模块都在一个架构下，因此在扩展订单模块的性能时不得不考虑其他模块的因素，不能因扩展某个模块的性能而损害其他模块的性能，从而难以实现按需伸缩。

在项目初期，传统软件架构由于开发、测试和部署过程相对简单，能够顺利运行。但是，随着需求的持续增长和开发团队规模的增加，代码库迅速膨胀。这导致架构逐渐变得臃肿，其可维护性和灵活性逐步下降，同时维护成本也在持续上升。

3. 微服务能否解决这些问题

微服务架构在一定程度上可以解决当前软件架构设计所面临的挑战。通过将大型软件系统拆分为多个小型、自治的服务，微服务能够有效应对系统规模和复杂性的问题。这使得每个服务都可以独立开发、测试、部署和扩展，从而提高系统的可维护性、可扩展性和灵活性。

此外，微服务架构通过松耦合设计提升了系统的可替换性和可伸缩性，增强了系统的弹性和可靠性。同时，它支持去中心化的管理和技术异构性，更好地适应不同的业务需求和技术栈。然而，微服务架构也面临一些挑战，例如安全性、数据管理、敏捷性和云化等方面的问题，需要采用适当的技术和架构来应对这些挑战。

1.4　微服务与单体、SOA 的区别

众所周知，在当前流行的软件架构领域中，存在着微服务、单体应用以及 SOA（面向服务的架构）这三种截然不同的架构模式。那么，微服务与单体应用和 SOA 之间究竟存在着何种区别与联系呢？本节将揭晓答案。

1. 微服务与单体架构的区别

在单体架构中，所有模块都包含在同一工程内，并且共用一个数据库，这导致存储方式较为单一。相比之下，在微服务架构中，每个服务都可以选择不同的存储解决方案（例如，某些服务可能采用 Redis，而其他服务可能使用 MySQL），并且每个服务都有其专用的数据库。

与传统的单体架构相比，单体架构中的所有模块紧密耦合，代码量庞大且难以维护。微服务架构允许每个服务独立运作，类似于独立项目，这显著减少了代码量，并简化了问题解决的过程。

此外，在微服务架构下，每个服务可以采用不同的技术栈，从而提供了更大的开发灵活性和多样性，如图 1-6 所示。

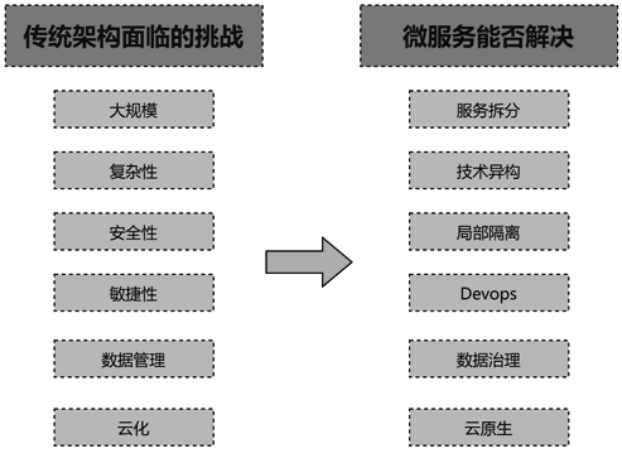

图 1-6　单体架构与微服务架构图

2. 微服务与 SOA 的区别

SOA（面向服务的架构）是一种软件架构风格，它将应用程序的功能分解为独立的服务单元，并通过企业服务总线（ESB）进行通信和集成。SOA 强调服务的可重用性和松耦合性，适用于构建大型企业级应用。

微服务架构将应用程序构建为一组小型、独立的服务，每个服务在自己的进程中运行，并通过轻量级的通信机制进行交互。这些服务专注于单一业务功能，具有高度的自治性和独立性，这有助于开发团队进行敏捷开发和快速迭代，同时提高了系统的容错性和可靠性。

SOA 与微服务架构如图 1-7 所示。

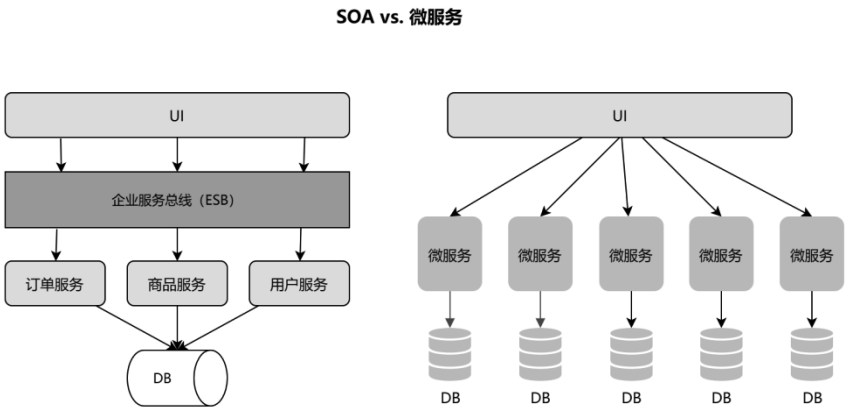

图 1-7　SOA 与微服务架构图

微服务架构和 SOA 在表面上可能看起来有些相似，以至于早期有人将微服务视作细粒度的 SOA。然而，它们之间实际上存在显著差异，主要表现在以下方面。

（1）服务粒度：微服务架构中的服务粒度较细，每个服务专注于单一业务功能的执行；而 SOA 中服务的粒度较粗，通常覆盖更广泛的业务流程。

（2）架构风格：微服务架构倾向于去中心化，允许各个服务独立选择技术栈和数据存储方式；SOA 则通常具有更集中的管理和控制，强调服务的规范化和标准化。

（3）通信方式：微服务之间的通信通常采用轻量级的 HTTP/RESTful 协议，简单而高效；SOA 则常依赖企业服务总线（Enterprise Service Bus，ESB）进行消息路由和转换，通信过程相对复杂。

（4）部署方式：微服务支持独立部署，使得部署速度更快，更易于进行扩展和更新；SOA 中的服务部署往往更为复杂，更新和扩展的难度较高。

（5）数据管理：在微服务架构中，每个服务通常拥有独立的数据管理，并通过特定机制确保数据一致性；SOA 则更倾向于采用集中式的数据管理。

（6）技术选型：微服务架构为每个服务提供了更大的技术选型自由度；SOA 则更注重技术的统一性和兼容性。

（7）敏捷性：微服务架构更能适应快速变化的业务需求，具有更短的开发和部署周期，展现出更高的敏捷性；相比之下，SOA 在应对变化时的灵活性可能稍显不足。

3. 总结

总的来说，微服务、单体应用和 SOA 是三种不同的软件架构风格，每种风格都有其独特的优势和适用场景。选择恰当的架构风格应基于具体的业务需求和技术环境来做出决策。

1.5　什么场景适合微服务

此前已阐述了微服务的定义、采用微服务的缘由，以及微服务与单体、SOA 等架构的差异。那么，什么样的系统适合采用微服务架构呢？

适合微服务架构的系统通常具有以下特点：

- 大规模和复杂性：当项目规模较大，功能模块较多，需要支持多种客户端、多种协议和多种数据源时，采用微服务架构可以更好地管理和维护系统。
- 高可用和高性能：当项目需要支持高并发和高可用性时，采用微服务架构可以通过横向扩展来提高系统的性能和可用性。
- 技术异构性：当项目需要支持多种技术栈和开发语言时，采用微服务架构可以更好地适应不同的技术栈和开发语言，从而提高开发效率和灵活性。
- 敏捷开发：当项目需要支持快速迭代和敏捷开发时，采用微服务架构可以通过自治性和松耦合的设计来提高开发效率和灵活性。
- 云原生：当项目需要支持云原生应用开发和部署时，采用微服务架构可以更好地适应云环境的要求，从而提高部署效率和可伸缩性。

需要注意的是，微服务架构并非适用于所有场景。对于一些规模较小、功能单一的项目，采用微服务架构可能会不必要地增加开发和维护的成本。因此，应根据项目的具体情况来做出选择。

此外，微服务的划分应基于业务功能的独立性。对于涉及操作系统内核、存储、网络、数据库等基础层面的系统，它们属于底层架构，各功能之间的依赖关系通常较为紧密。在这种情况下，如果强行将这些系统拆分为多个小服务单元，可能会导致集成过程变得复杂，工作量大幅增加，并且难以实现业务隔离和服务的独立部署运行。因此，这类系统可能不适合采用微服务架构。

1.6　微服务架构的形态

既然微服务有着众多优点，又是应用系统架构的大趋势，本节就来深入了解其具体形态。

许多人声称自己的架构是微服务架构，那么微服务架构究竟是怎样的呢？

1. 微服务需要解决的问题

微服务架构，简而言之，是将单体应用进一步分解，拆分为更为细小的服务，每个服务均为能够独立运行的项目。然而，一旦采用微服务系统架构，必然会面临如下几个问题：

（1）数量众多的小服务，应当如何对其进行管理？
（2）数量众多的小服务，它们之间应当怎样实现通信？
（3）数量众多的小服务，客户端应通过何种方式对其进行访问？
（4）数量众多的小服务，一旦产生问题，应当如何进行排错？

针对上述问题，任何一位微服务设计者均无法回避，故而大部分微服务架构解决方案针对每个问题都提供了相应的解决策略或组件。

2. 微服务架构到底长什么样

从整体结构来看，微服务架构类似于由多个小型、独立的模块构成的系统。

每个微服务都是一个独立的应用单元，具备自己的业务逻辑、数据存储和运行环境。它们专注于实现特定的业务功能。例如，用户管理微服务处理用户的注册、登录和信息管理；订单管理微服务则负责订单的创建、处理和跟踪等业务流程。

这些微服务通过轻量级的通信机制进行交互，通常是基于 HTTP 协议的 API 接口。它们可以部署在不同的物理服务器上，或在同一服务器的不同容器中，以实现灵活的部署和扩展策略，如图 1-8 所示。

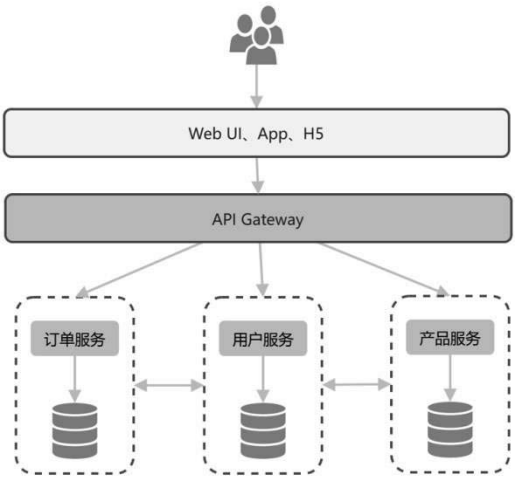

图 1-8　微服务架构模型图

通过将服务进行原子化拆分，确保每个微服务都有明确的任务划分和单一的业务职责，这有助于实现服务的扩展性。微服务之间通过 RESTful 等轻量级 HTTP 协议进行相互调用，保持了松耦合的状态。它们能够独立地进行打包、部署和升级，有效避免了因单一模块问题导致的整个服务崩溃的风险。

3. 微服务治理核心概念

我们在学习微服务的时候，通常会遇到各种概念和组件，例如服务治理、服务调用、服务网关和服务容错等。那么，这些组件究竟是什么？它们各自有什么作用呢？

1）服务注册与发现

服务注册与发现是分布式或微服务架构中的核心，它能让各微服务动态地了解彼此的状态，有利于系统动态扩展、灵活部署，提高整体的可靠性与可维护性。

- 服务注册：各微服务将自身信息注册到注册中心。
- 服务发现：其他微服务或客户端通过注册中心获取目标服务信息（如地址和端口），以建立连接和通信。

2）服务调用

服务调用是指一个微服务向另一个微服务发起请求，并接收相应的响应的过程。通过定义明确的接口和协议，实现不同服务之间的通信与协作，这有助于促进服务的复用和功能的解耦。目前，主流的远程服务调用技术包括基于 HTTP 请求的 RESTful 接口和基于 TCP 的 RPC 协议。

- RESTful（Representational State Transfer）：RESTful 是一种基于 HTTP 协议的软件架构风格和设计原则，用于构建网络应用程序的架构。
- RPC（Remote Promote Call）：RPC 是一种通信协议，用于在不同的进程或计算机之间进行通信和远程调用。它允许像调用本地服务一样调用远程服务。

表 1-1 给出了 RESTful 和 RPC 的区别和联系。

表1-1　RESTful和RPC的区别和联系

比　较　项	RESTful	RPC
通信协议	HTTP	一般是 TCP
性能	略低	较高
灵活度	高	低
应用	微服务架构	SOA 架构

3）服务网关

随着微服务数量的增加，不同的微服务可能具有不同的网络地址，而外部客户端可能需要调用多个服务接口以满足特定的业务需求。如果客户端直接与各个微服务通信，可能会遇到以下问题：

- 客户端需要管理众多不同的微服务地址，导致调用逻辑复杂，增加了开发和维护成本。
- 微服务之间可能使用不同的通信协议和数据格式，增加了客户端的适配难度。
- 缺少统一的安全入口，使得安全策略难以实施，降低了数据和服务的安全性。
- 难以集中管理流量，例如限流、熔断和降级等，影响了系统的稳定性和可靠性。
- 容易产生跨域请求问题，增加了通信障碍。
- 客户端需要处理不同微服务的版本兼容性问题。

为了解决这些问题，服务网关应运而生。服务网关作为微服务架构的统一入口，接收外部请求并将其路由到后端相应的服务。它还承担身份验证、授权、请求限流、协议转换等职能，如图 1-9 所示。

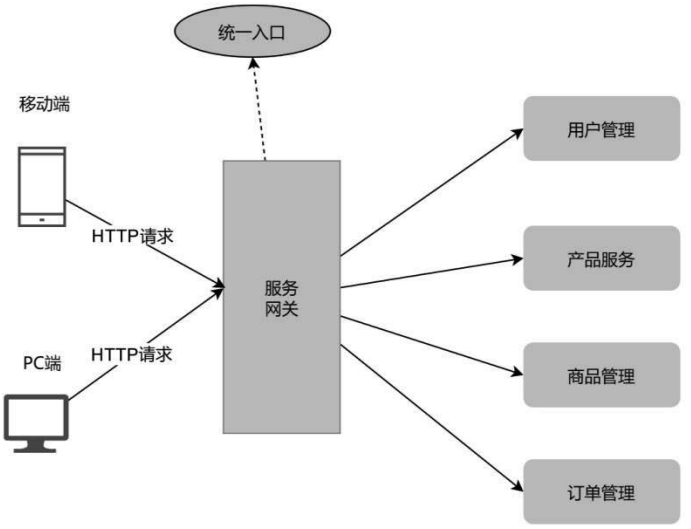

图 1-9　服务网关结构图

4）服务容错

服务容错旨在处理服务调用过程中可能出现的错误和异常情况。常见的容错机制包括断路器模式、重试机制、降级处理等。其作用是当某个服务出现故障时，能够避免故障扩散，以保证整个系统的稳定性和可用性，尽量减少对业务的影响，如图 1-10 所示。

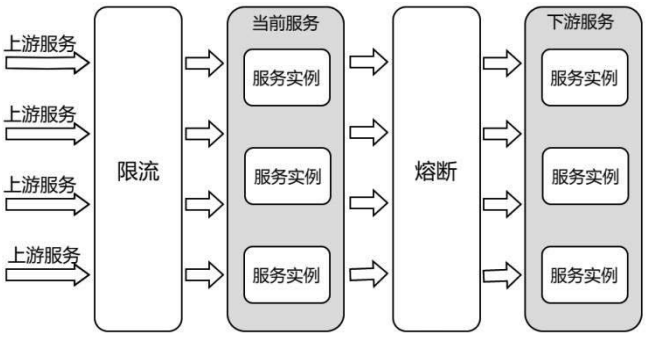

图 1-10　服务容错机制

总之，这些组件和概念相互配合，共同构建了一个可靠、高效、灵活的微服务架构体系。

1.7　本章小结

在当今互联网时代，微服务架构已成为构建高效、稳定的分布式系统的首选架构方案。然而，随着微服务架构的广泛应用，它所面临的挑战也日益显著。本章首先介绍了微服务架构的起源、基本概念和演变历程，然后详细论述了微服务的优势和挑战，探讨了采用微服务的原因以及微服务适用的场景，最后阐释了微服务的具体实现形态。

通过本章的学习，读者应已掌握微服务的基本准则、设计模式和最佳实践，明确了微服务的定义、架构特点以及适合采用微服务的项目类型，为进一步深入学习奠定了基础。