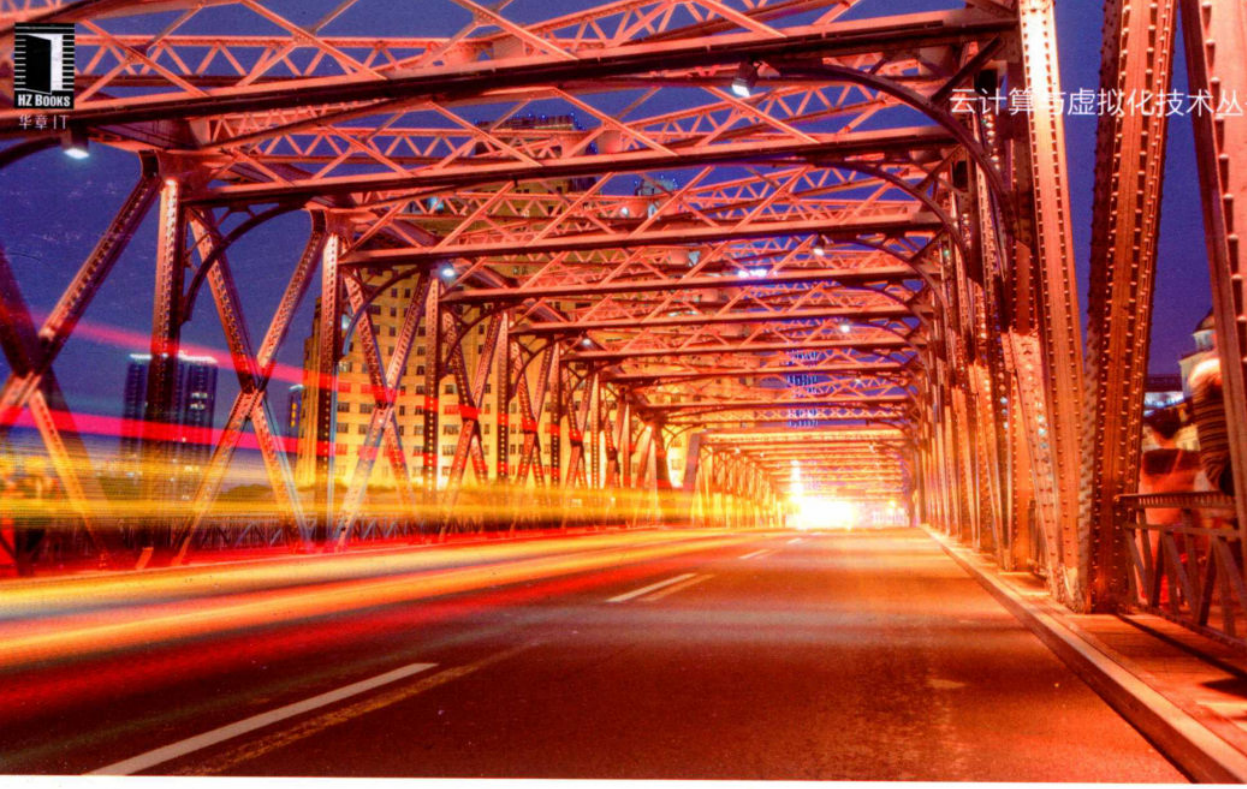


云计算与虚拟化技术丛

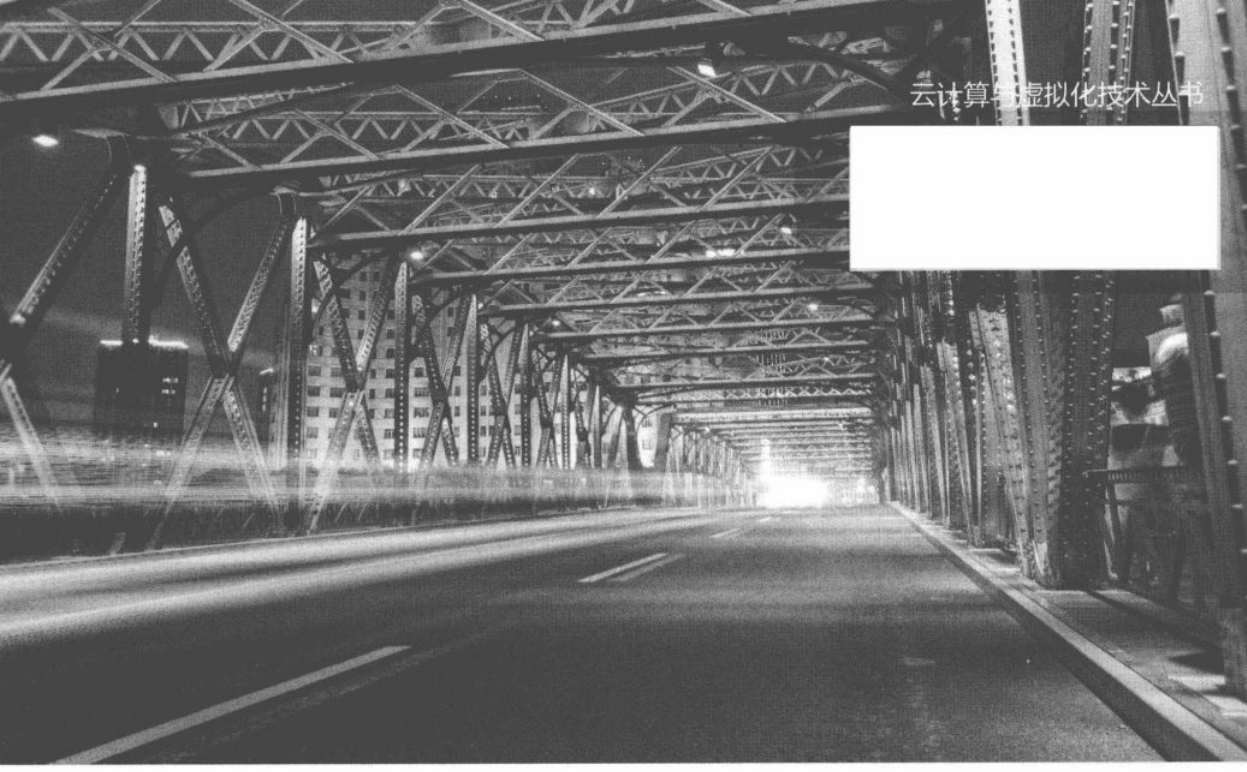
The Beginner's Guide to Spring Cloud

极简 Spring Cloud实战

胡劲寒 编著

精巧实用、快速入门，资深Spring Cloud践行者经验集结
组件应用与原理分析结合，并以综合案例融合微服务和DevOps实践

机械工业出版社
China Machine Press



云计算与虚拟化技术丛书

The Beginner's Guide to Spring Cloud

极简 Spring Cloud实战

胡劲寒 编著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

极简 Spring Cloud 实战 / 胡劲寒编著 . —北京：机械工业出版社，2019.8
（云计算与虚拟化技术丛书）

ISBN 978-7-111-63281-8

I. 极… II. 胡… III. 互联网络 - 网络服务器 IV. TP368.5

中国版本图书馆 CIP 数据核字（2019）第 148722 号

极简 Spring Cloud 实战

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：高婧雅　　责任校对：李秋荣
印　　刷：三河市宏图印务有限公司　　版　　次：2019 年 8 月第 1 版第 1 次印刷
开　　本：186mm × 240mm　1/16　　印　　张：13.75
书　　号：ISBN 978-7-111-63281-8　　定　　价：79.00 元

客服电话：（010）88361066　88379833　68326294　　投稿热线：（010）88379604
华章网站：www.hzbook.com　　读者信箱：hzit@hzbook.com

Preface 序

在互联网高速发展的时代，谁能够顺应趋势，快速拥抱变化，谁就能在未来的市场充满无限可能性。在滋润的互联网土壤上，各种技术框架、组件得到了蓬勃发展，而微服务无疑是这场技术狂欢中最受关注的热门技术之一。微服务的出现为高速发展的互联网企业带来了新的技术架构理念，松耦合的独立服务组件也使得微服务架构能够快速响应复杂业务的变化，加上对传统软件工程化的革新，也极大推动了自动化发展，以及持续集成与敏捷交付。

但是架构的演进也带来了技术的挑战，特别是服务治理层面的技术复杂性，例如：服务注册与发现、负载均衡、链路跟踪、监控与故障处理（熔断、降级）、APM、请求路由等，一系列的关键技术点都要求技术团队在微服务技术领域中持续投入、持续建设，对于那些缺乏足够技术储备能力的创业团队，技术投入的代价往往过大，如果能有一整套完整的微服务集成与治理的技术方案该有多好。

Spring 作为企业级技术框架中的佼佼者没有错过微服务这个风口。从它第一天出现就注定它的不平凡，Spring 通过其强大的抽象能力以及技术集成能力，结合 Netflix 成熟的开源服务套件，一出现就成为最热门的微服务技术集成方案。

Spring Cloud 也继承了 Spring 一如既往的风格，考虑了微服务的几乎所有功能，另外组件化的思维也为企业微服务架构技术落地提供了更多的灵活性。企业不仅可以通过 Spring Cloud 快速建立起自己的微服务技术体系，也可以通过整合 Spring Cloud 技术组件为已有的技术方案赋能。

劲寒在微服务领域研究多年，有着丰富的微服务落地实施经验，尤其对 Spring Cloud 有着极其深入的研究，在社区上也帮助很多人解决了 Spring Cloud 实际运用中的问题。在互联互通的时代，精华的知识应该开放分享，本着这个理念，劲寒将其对微服务的理解，将 Spring Cloud 研究心得与实战经验全都融合到本书中。

本书不仅介绍了微服务的背景与 Spring Cloud 技术体系价值，让读者快速了解全貌。

更在细节层面对 Spring Cloud 各个组件进行了细致的讲解与深入的原理剖析，最后以微服务技术的视角，结合时下热门的容器化、CI/CD 技术，介绍了微服务未来的运维形态和发展方向，真正做到了让读者既知其然，也知其所以然，更知其未来。

优秀的技术书籍阅读完后让人感觉如同品尝香醇的美酒，口感复杂却层次分明，这本书就带给了我这样的感觉。

——刘洋，同程金服 CTO

Preface 前　言

为什么要写这本书

从业近十年，一直从事服务端架构与基础平台等方向的工作，积累了些许服务端架构、微服务领域的心得，此前一直通过技术博客、GitHub、技术社区等方式与同行分享和交流。

Spring Cloud 作为 Spring 推出的一个基于微服务的完整解决方案，能够很方便地使企业进行微服务化转型。笔者结合 Spring Cloud 的微服务落地与推广工作，于实践中更加深了对微服务架构优势的理解。

一次偶然的机会，与策划编辑高婧雅交流后，达成出版意向，开始了写作之旅。

在写作过程中由于工作等个人原因，导致此书的写作计划数次延期，在此郑重对高婧雅编辑表示歉意。幸而在高编辑的不断鞭策与自身的努力下得以成稿。

本书特色

本书力求既全面又精巧，体现在以下方面：揭示 Spring Cloud 的核心特点、关键原理与应用，对于 Spring Cloud 中的每个组件，甚至每个可以支持自定义的扩展场景均有深入介绍。

本书从实战、进阶、全面配置三个层次展开介绍，分为三篇。**基础服务篇**介绍构建一个核心微服务架构不可缺少的部分。**任务与消息篇**则着重介绍 Spring Cloud 针对消息、任务、调用依赖等方面的支持方案。**微服务实战篇**基于 Spring Cloud+Docker 构建一个精简而又五脏俱全的小项目。

读者对象

- 架构师

- 程序开发人员
- 运维管理人员
- 其他对微服务感兴趣的人员

如何阅读本书

本书分为三篇，共 14 章内容。

基础服务篇（第 1～9 章），本篇内容是实践微服务必备的知识点和技能，需要重点学习。

第 1 章对微服务演进历程以及 Spring Cloud 的全貌进行了提纲挈领的介绍，以期读者有全局性认知，使后面的学习不会碎片化。

第 2～8 章主要介绍了在分布式应用中几个核心场景的 Spring Cloud 解决方案，分别深入介绍 Spring Cloud 在服务调用、治理、调用链追踪、熔断及服务网关等方面的实现框架，这些内容是读者实践微服务的基础。

第 9 章主要介绍了 Spring Cloud 中注册中心的其他实现和快速调试、开发脚手架。

任务与消息篇（第 10～13 章），主要介绍消息处理以及任务流依赖处理方面的组件的使用及其实现原理。

微服务实战篇（第 14 章），本篇是基于 Spring Cloud、Docker、OAuth2 构建微服务的一个完整案例。

读者可以根据自身情况，全书阅读或者选择性重点阅读。然而，如果你是一名初学者，请在开始阅读本书之前，先进行一些分布式领域基础理论知识的学习。

勘误和支持

由于笔者水平有限，编写时间仓促，书中难免会出现一些错误或者不准确的地方，恳请读者批评指正。如果你有更多宝贵意见，欢迎发邮件至我的个人邮箱 wawzw123@163.com 进行讨论，我会尽量为读者提供最满意的解答。期待得到你们的真挚反馈，在技术之路上互勉共进。

致谢

感谢 Spring Cloud 官方文档，在写作期间提供给我最全面、最深入、最准确的参考材料，强大的官方文档支持是其他数据库所无法企及的。

感谢 Spring Cloud 中文社区的各位技术专家的博客文章，每次阅读必有所获，本书也多处引用了他们的观点和思想。

感谢所在公司的领导及同事，在微服务技术全面落地的过程中给予的极大信任与支持。

特别致谢

最后，我要特别感谢我的父母和妻子，我为写作这本书牺牲了很多陪伴他们的时间，但也正因为有了他们的付出与支持，我才能坚持写下去。

谨以此书献给我最亲爱的家人，以及众多热爱微服务架构的朋友们！

胡劲寒

目　录 *Contents*

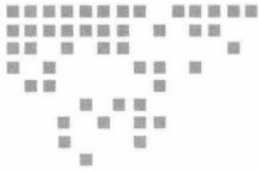
第二篇　任务与消息篇

第三篇　微服务实战篇

第一篇 Part 1

基础服务篇

本篇将为读者介绍微服务架构的演进过程，带领读者了解什么是微服务，为什么需要微服务，以及微服务与 Spring Cloud 之间是什么关系，为什么要选择 Spring Cloud 来实现微服务而不是市面上现存的其他解决方案。了解之后，相信读者会有自己的答案。



Chapter 1 第 1 章

微服务与 Spring Cloud

本章将带领读者从服务端架构的演进历程开篇，描述服务端架构演进至微服务的必然，同时介绍实现微服务的最佳方式——Spring Cloud，并对 Spring Cloud 的组成及其同业对比的优势进行初步介绍。读者通过本章能够了解微服务是什么，以及 Spring Cloud 如何帮助技术开发人员快速实现微服务。

1.1 架构演进

1.1.1 服务端架构发展

由于人们首先想到的是让两台或多台计算机相互通信，因此构思出了，如图 1-1 所示的简易通信模型。

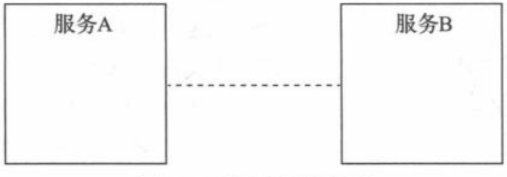


图 1-1 简易通信模型

互相通信的两个服务可以满足最终用户的一些需求。但这个示意图显然过于简单，缺少包括通过代码操作的字节转换和在线路上收发的电信号转换在内的多个层。虽然一定程度上的抽象对于讨论是必需的，但仍需添加网络协议栈（组件）以增加细节内容，如图 1-2 所示。

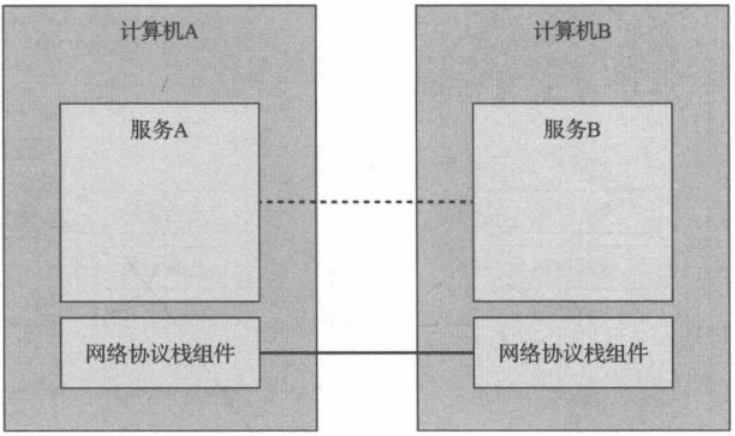


图 1-2　增加网络协议栈（组件）后

上述这个修改过的模型自 20 世纪 50 年代一直使用至今。一开始，计算机很稀少，也很昂贵，所以两个节点之间的每个环节都被精心制作和维护。随着计算机变得越来越便宜，连接的数量和数据量大幅增加。人们越来越依赖网络系统，工程师需要保证他们构建的软件能够达到用户所要求的服务质量。

当然，还有许多问题急需解决以达到用户要求的服务质量。人们需要找到解决方案让机器互相发现，通过同一条线路同时处理多个连接，允许机器在非直连的情况下互相通信，通过网络对数据包进行路由、流量加密等。

其中，有一种机制称为流量控制，下面以此为例。流量控制是一种防止一台服务器发送的数据包超过下游服务器可以承受上限的机制。这是必要的，因为在一个联网的系统中，至少有两台不同的、独立的计算机，彼此之间互不了解。计算机 A 以给定的速率向计算机 B 发送字节，但不能保证 B 可以连续地、以足够快的速度来处理接收到的字节。例如，B 可能正在忙于并行运行其他任务，或者数据包可能无序到达，并且 B 可能被阻塞以等待本应该第一个到达的数据包。这意味着 A 不仅不知道 B 的预期性能，还可能让事情变得更糟，导致 B 过载，B 现在必须对所有这些传入的数据包进行排队处理。

一段时间以来，大家寄希望于建立网络服务和应用程序的开发者能够通过编写代码来解决上面提出的挑战。在这个流程控制示例中，应用程序本身必须包含某种逻辑来确保服务不会因为数据包的原因而过载。这种重联网的逻辑与业务逻辑一样重要。抽象示意图如图 1-3 所示。

随着像 TCP/IP 这样的标准横空出世，流量控制和许多其他问题的解决方案被融入网络协议栈本身。这意味着这些流量控制代码仍然存在，但已经从应用程序转移到了操作系统提供的底层网络层，如图 1-4 所示。

这个模型相当成功。几乎任何一个组织都能够使用商业操作系统附带的 TCP/IP 协议栈来驱动他们的业务，即使有高性能和高可靠性的要求。

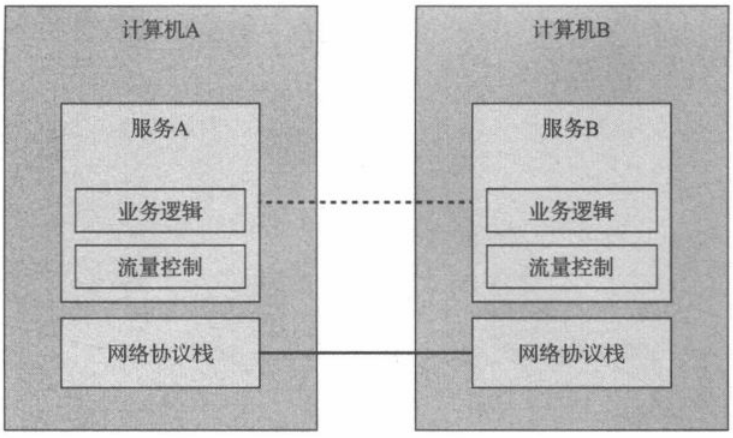


图 1-3 逻辑分离后

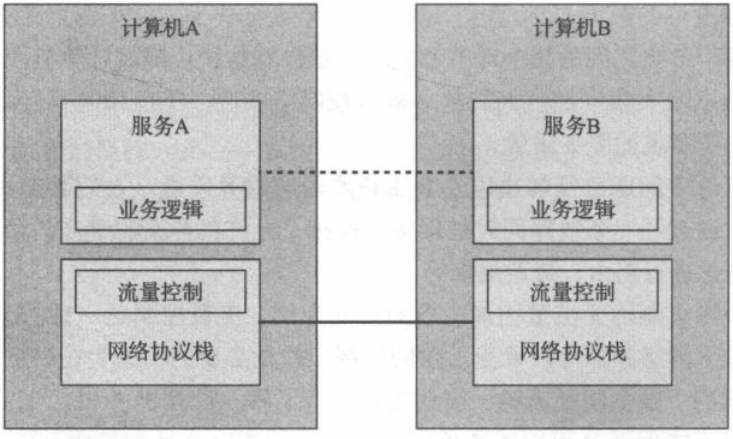


图 1-4 分层后

1.1.2 微服务架构

随着节点和稳定连接的数量越来越多，行业中出现了各种各样的网络系统：从细粒度的分布式代理和对象，到由较大但重分布式组件组成的面向服务的架构。这样的分布式系统带来了几个难题，有一些是新出现的，也有原有难题的“升级版”。

20 世纪 90 年代，Peter Deutsch 和他在 Sun 公司的同事撰写了《分布式计算的八大错误》一文，文中列出了人们在使用分布式系统时通常会做出的一些假设。Peter 认为，这些假设在更原始的网络架构或理论模型中可能是真实存在的，但在现代世界中是不成立的：

- 网络是可靠的；
- 延迟为零；
- 带宽是无限的；

- 网络是安全的；
- 拓扑是不变的；
- 管理员实时监控维护；
- 传输成本为零；
- 网络是同构的。

因此，工程师们必须处理这些问题。

为了处理更复杂的问题，需要转向更加分散的系统（我们通常所说的微服务架构），这在可操作性方面提出了新的要求。下面则列出了必须要处理的问题：

- 计算资源的快速提供；
- 基本的监控；
- 快速部署；
- 易于扩展的存储；
- 可轻松访问边缘；
- 认证与授权；
- 标准化的 RPC;

因此，尽管数十年前开发的 TCP/IP 协议栈和通用网络模型仍然是计算机之间相互通信的有力工具，但更复杂的架构引入了其他层面的问题。此时业界出现了微服务思想，以期解决上述问题。例如，微服务用服务发现与断路器技术来解决上面列出的几个弹性扩展和分布式问题，如图 1-5 所示。

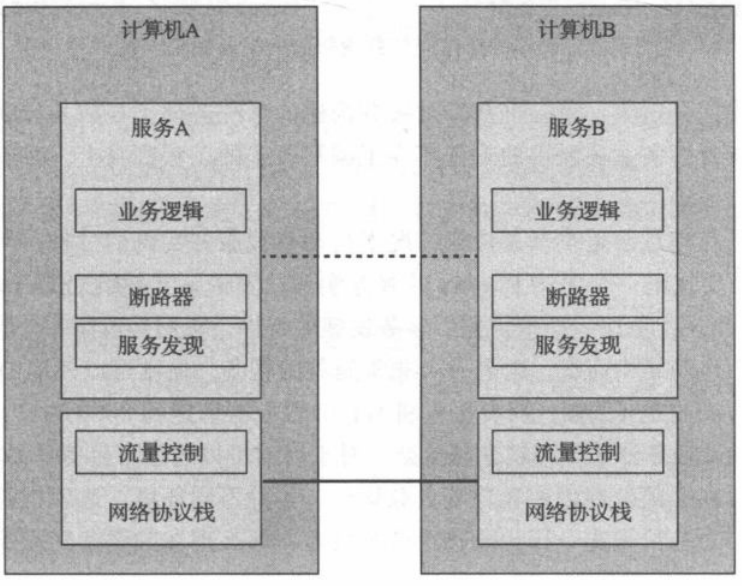


图 1-5　加入微服务层后

微服务架构风格是一种将一个单一应用程序开发为一组小型服务的方法，每个服务运行在自己的进程中，服务间通信采用轻量级通信机制（通常用 HTTP 资源 API）。这些服务围绕业务能力构建并且可通过全自动部署机制独立部署。这些服务共用一个最小型的集中式的管理，服务可用不同的语言开发，使用不同的数据存储技术。

我们为了将系统构建为微服务架构，除了服务是可独立部署、可独立扩展的之外，每个服务都提供一个固定的模块边界，甚至允许不同的服务用不同的语言开发，由不同的团队管理。图 1-6 展示了单体应用到微服务的简易图解。

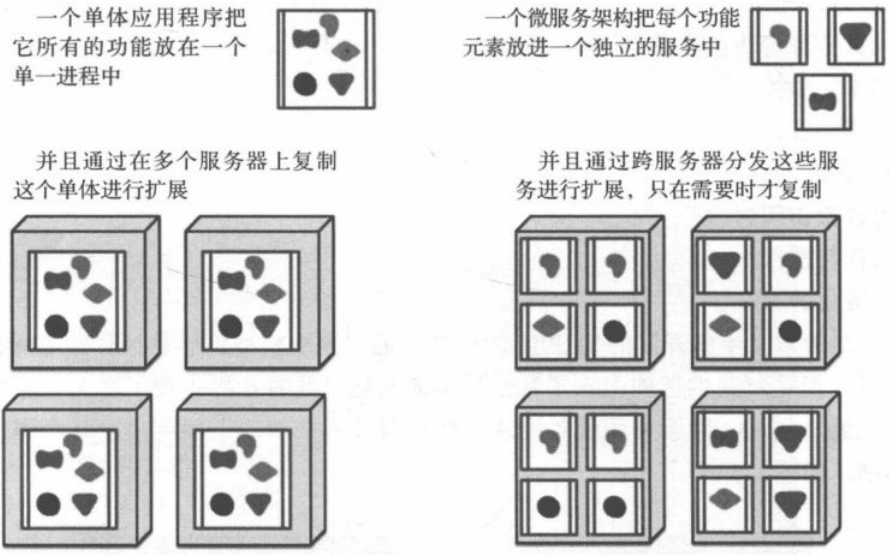


图 1-6 单体应用到微服务的简易图解

然而历史往往会重演，第一批基于微服务构建的系统遵循了与前几代联网计算机类似的策略。这意味着落实上述需求的责任落在了编写服务的工程师身上。我们以服务发现和断路器来说明。

服务发现是在满足给定查询条件的情况下自动查找服务实例的过程，例如，一个名叫 Teams 的服务需要找到一个名为 Players 的服务实例，其中该实例的 environment 属性设置为 production。当用户调用一些提供提供服务发现的组件，它们会返回一个满足条件的服务列表。对于中心化的架构而言，这是一个非常简单的任务，通常可以使用 DNS、负载均衡器和一些端口号的约定（例如，所有服务将 HTTP 服务器绑定到 8080 端口）来实现。而在更分散的环境中，任务开始变得越来越复杂，对于以前可以通过盲目信任 DNS 来查找依赖关系的服务，现在必须处理诸如客户端负载均衡、多种不同环境、地理位置上分散的服务器等问题。如果之前只需要一行代码来解析主机名，那么现在的服务则需要很多行代码来处理由分布式引入的各种问题。

断路器是由 Michael Nygard 在其编写的《 Release It 》一书中引入的模式，书中对该模