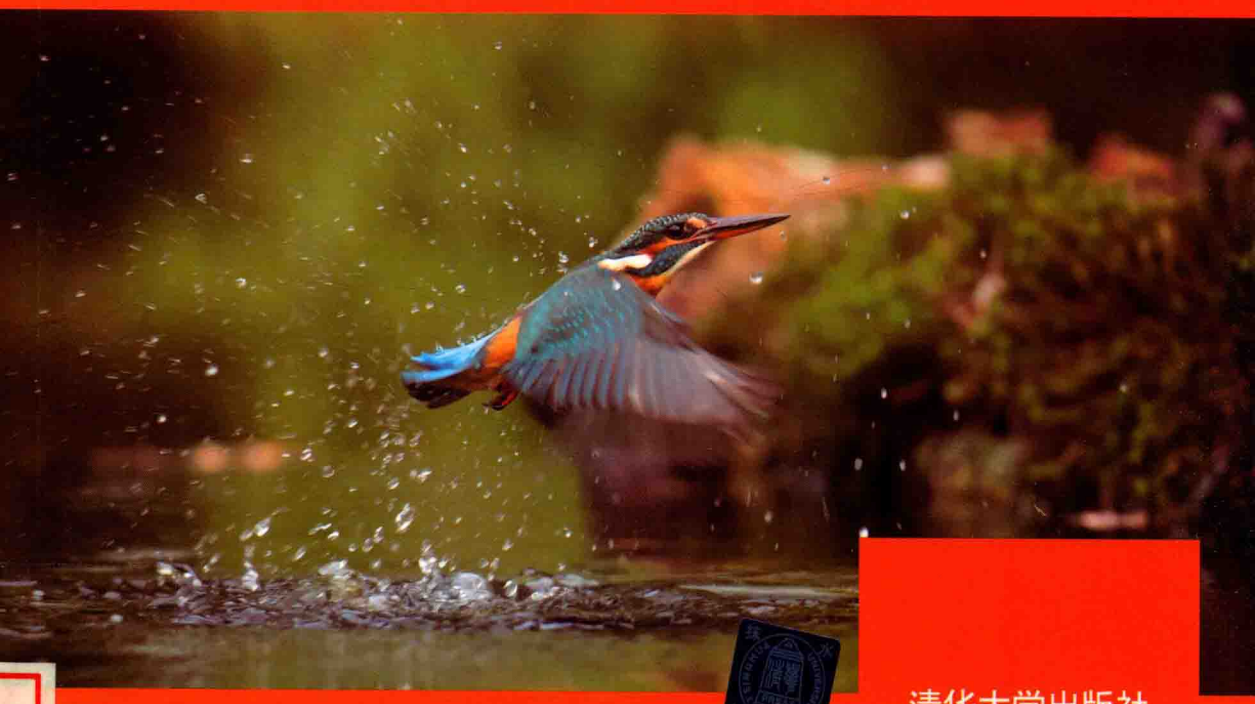
Microservice Patterns and Best Practices

微服务设计模式和最佳实践

[美] 维尼休斯 · 弗多萨 · 帕切科 著　　程晓磊 译



清华大学出版社

微服务设计模式和最佳实践

[美] 维尼休斯·弗多萨·帕切科 著

程晓磊 译

清华大学出版社

北 京

内 容 简 介

本书详细阐述了与微服务相关的基本解决方案，主要包括微服务概念、微服务工具、内部模式、微服务生态环境、共享数据微服务设计模式、聚合器微服务设计模式、代理微服务设计模式、链式微服务设计模式、分支微服务设计模式、异步消息微服务、微服务间的协同工作、微服务测试以及安全监测和部署方案等内容。此外，本书还提供了相应的示例、代码，以帮助读者进一步理解相关方案的实现过程。

本书适合作为高等院校计算机及相关专业的教材和教学参考书，也可作为相关开发人员的自学教材和参考手册。

北京市版权局著作权合同登记号 图字：01-2018-3298

图书在版编目（CIP）数据

微服务设计模式和最佳实践/（美）维尼休斯·弗多萨·帕切科著；程晓磊译. —北京：清华大学出版社，2019
书名原文：Microservice Patterns and Best Practices
ISBN 978-7-302-52041-2

Ⅰ. ①微… Ⅱ. ①维… ②程… Ⅲ. ①互联网络-网络服务器 Ⅳ. ①TP368.5

中国版本图书馆 CIP 数据核字（2019）第 008130 号

责任编辑：贾小红
封面设计：刘 超
版式设计：魏 远
责任校对：马子杰
责任印制：宋 林

出版发行：清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦 A 座 邮 编：100084
社 总 机：010-62770175 邮 购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：清华大学印刷厂
经 销：全国新华书店
开 本：185mm×230mm **印 张**：19 **字 数**：377 千字
版 次：2019 年 3 月第 1 版 **印 次**：2019 年 3 月第 1 次印刷
定 价：99.00 元

产品编号：080091-01

译 者 序

“微服务”的概念是由 ThoughtWorks 公司的首席科学家 Martin 提出的，他是敏捷开发方法创始人之一。自然而然地，微服务的目的就是有效拆分应用，实现敏捷开发和部署。与之相对应的，此前的软件开发都是基于一体化架构。一体化架构的优点是开发简单直接、集中式管理、功能都在本地；但是，一旦应用程序变大、团队扩张，那么一体化架构的缺点就会立即凸显，庞大的一体代码库可能会让新手程序员望而生畏，再加上开发效率低、代码维护困难、部署不灵活、难以扩展应用等，这些很可能成为企业和开发人员难以逾越的障碍。微服务的出现解决了这些矛盾，它将系统拆分成进程独立的服务，进行分布式管理、自动化运维，通过 API 网关、服务间调用、服务发现、服务容错、服务部署和数据调用实现了开发简单、独立按需扩展、高可用性、持续集成和持续交付等，特别契合云平台应用程序开发的需要，这也正是它日益流行的原因。

本书将介绍不同阶段的微服务中应用程序开发的各种设计模式及其最佳实践。微服务模式和最佳实践始于对微服务关键概念的理解，并展示如何在设计微服务时做出正确的选择。本书将讨论内部微服务应用程序的各种方法，如缓存策略、异步机制、CQRS 和事件源等。随着过程的不断推进，读者将深入了解微服务的相关设计模式。

在本书的翻译过程中，除程晓磊外，刘璋、张博、刘晓雪、王烈征、张华臻、刘祎等人也参与了部分翻译工作，在此一并表示感谢。

由于译者水平有限，难免有疏漏和不妥之处，恳请广大读者批评指正。

译 者

前　言

微服务是一种软件体系结构策略，多年来一直在使用，其目标是提高服务的可伸缩性。由于当今的业务呈现快速、动态增长之势，单体应用程序与面向服务相比已不占优势。通过设计这种新的体系结构模型，面向对象的原则、标准、解耦和职责已经构成了超越自动化测试的基础内容。

微服务可使读者能够创建可维护、可伸缩的应用程序。在阅读本书后，读者将能够创建可互操作的微服务，同时兼具可测试性和高性能的特征。

适用读者

本书面向于具有 Web 开发经验，但想要改进其开发技术以创建可维护和可伸缩应用程序的读者，读者不需要具备微服务方面的经验。本书中演示的概念和标准，使读者能够开发易于理解和支持大量访问的应用程序。

本书内容

第 1 章整体介绍微服务的概念，以帮助读者理解体系结构背后的相关理念，例如客户优先方案以及域定义。

第 2 章讨论微服务应用中的常见工具。一旦了解了客户以及如何定义应用程序域，即可制定技术决策方案，包括所使用的语言、框架，以及如何验证微服务框架的功能。

第 3 章将探讨内部微服务的应用模式，如缓存策略、worker、队列以及异步机制。

第 4 章考察如何从单体应用程序中创建一组具有弹性和可伸缩的微服务。本章还将重点讨论如何在各自的容器中正确地分离微服务，并解释存储层的分布。

第 5 章涉及一些较为特殊的用例。通常，微服务在业务、测试、通信、连接和存储方面是完全独立的；而共享模式则是迁移概念的一种特殊模式（从单体到微服务）；总的来说，这是一种过渡模式。

第 6 章主要讨论了一些简单而常见的模式，主要包括对微服务的数据编排。

第 7 章阐述了代理模式，它与聚合器模式非常相似。对于该结构，当不需要连接数据并将其发送给终端用户时，可以对其加以使用。本章的目标是了解代理模式和聚合器模式之间的差异，以及针对微服务的正确的请求重定向操作。

第 8 章将学习链式模式。其中，发送至客户端的信息将整合至链中。本章的主要目标是解释信息整合问题。

第 9 章介绍分支模式，该模式可视作聚合器模式和链式模式的混合体，通常用于以下场合：在后端中，微服务未包含所有数据以完成某项任务；或者应直接通知另一个微服务。本章解释了该模式的应用时机，以及对业务的作用方式。

第 10 章解释了一种较为复杂的模式：在微服务级别使用异步机制。本章将讨论如何利用消息工具实现微服务的异步通信方式。

第 11 章对相关模式进行总结。在介绍了所有的微服务模式后，将考察如何将微服务进行整合，以使其可有效地协同工作。

第 12 章将讨论最为合理的测试机制及其简化方式。

第 13 章介绍了生产过程中维护微服务的必要条件以及最佳实践。

背景知识

如果读者了解一些 Go 语言 （golang）中的 OOP 和包结构，那么，阅读本书将变得更加轻松、有趣。

资源下载

读者可访问 http://www.packtpub.com 并通过个人账户下载示例代码文件。另外，http://www.packtpub.com/support，注册成功后，我们将以电子邮件的方式将相关文件发与读者。

读者可根据下列步骤下载代码文件。

（1）登录 www.packtpub.com 并注册我们的网站。

（2）选择 SUPPORT 选项卡。

（3）单击 Code Downloads & Errata。

（4）在 Search 文本框中输入书名并执行后续命令。

当文件下载完毕后，确保使用下列最新版本软件解压文件夹。

- Windows 系统下的 WinRAR/7-Zip。
- Mac 系统下的 Zipeg/iZip/UnRarX。
- Linux 系统下的 7-Zip/PeaZip。

另外，读者还可访问 GitHub 获取本书的代码包，对应网址为 https://github.com/PacktPublishing/Microservice-Patterns-and-Best-Practices。此外，读者还可访问 https://github.com/PacktPublishing/以了解丰富的代码和视频资源。

读者可访问 https://www.packtpub.com/sites/default/files/downloads/MicroservicePatternsandBestPractices_ColorImages.pdf 下载本书的彩色图像，以方便读者对比某些输出结果。

本书约定

代码块则通过下列方式设置：

```
class TestDevelopmentConfig(TestCase):

    def create_app(self):
        app.config.from_object('config.DevelopmentConfig')
        return app

    def test_app_is_development(self):
        self.assertTrue(app.config['DEBUG'] is True)
```

代码中的重点内容则采用黑体表示：

```
@patch('views.rpc_command')
def test_sucess(self, rpc_command_mock):
    """Test to insert a News."""
```

命令行输入或输出如下所示：

```
$ docker-compose -f docker-compose.yml up --build -d
```

图标表示较为重要的说明事项。

图标则表示提示信息和操作技巧。

读者反馈和客户支持

欢迎读者对本书的建议或意见予以反馈。

对此，读者可向 feedback@packtpub.com 发送邮件，并以书名作为邮件标题。若读者对本书有任何疑问，均可发送邮件至 questions@packtpub.com，我们将竭诚为您服务。

勘误表

尽管我们在最大程度上做到尽善尽美，但错误依然在所难免。如果读者发现谬误之处，无论是文字错误抑或是代码错误，还望不吝赐教。对此，读者可访问 http://www.packtpub.com/submit-errata，选取对应书籍，单击 Errata Submission Form 超链接，并输入相关问题的详细内容。

版权须知

一直以来，互联网上的版权问题从未间断，Packt 出版社对此类问题异常重视。若读者在互联网上发现本书任意形式的副本，请告知网络地址或网站名称，我们将对此予以处理。关于盗版问题，读者可发送邮件至 copyright@packtpub.com。

若读者针对某项技术具有专家级的见解，抑或计划撰写书籍或完善某部著作的出版工作，则可访问 www.packtpub.com/authors。

问题解答

若读者对本书有任何疑问，均可发送邮件至 questions@packtpub.com，我们将竭诚为您服务。

目　录

第 1 章　微服务概念

在编程领域，设计模式十分常见。同样，在 Web 开发中，这一点也不例外。随着互联网的普及以及 Web 2.0 的出现，许多为 Web 开发的模式被广泛传播，目的是使开发更动态、更简单，以适应新的特性。

诸如 MVC（模型-视图-控制器）、HMVC（层次模型视图控制器）和 MTV（模型模板视图）等模式激发了各种框架的创建，例如 Django、Ruby on Rails、Spring MVC 和 CodeIgniter。所有这些框架都非常适合快速创建 Web 应用程序，且无须对应用程序架构予以更多的关注。这是因为大部分工作都是由框架完成的，所有这些模式适用于包含所需业务规则的 Web 应用程序。通常，这些应用程序（其中，所有业务规则都位于同一代码库上）被称为单体。多年以来，在 Web 开发生态系统中，单体绝对占统治地位。许多公司通过全栈框架软件产品在市场上寻找销售空间。许多单体软件已大量应用于互联网上，随着时间的推移，这些单体应用也产生了一些问题。

对于单体应用程序来说，其开发过程存在以下困难：

- 鉴于整合操作难以实施，同一代码库中的维护变得更加复杂。
- 实现新特性的难度不断增加，开发周期往往超出预期的规定。
- 应用程序的可伸缩性。
- 部署新特性且不会对在线内容产生影响变得越加困难。
- 体系结构的变化使问题趋于复杂化。

上述内容只是单体应用程序中可能存在的部分问题。这些困难也使得人们思考，并将应用程序架构迁移到微服务中去。越来越多地，微服务已经成为软件工程行业和大多数公司所采取的方案。在这些行业中，“成功”一词意味着实践过程和可伸缩业务所面临的问题。微服务体系结构包含以下优点：

- 针对每个微服务设置专有的业务领域，以促进新特性的实现。
- 更好的业务定义，消除了它们之间的循环依赖关系。
- 独立部署。
- 简化错误的识别过程。
- 微服务之间技术的独立性。
- 团队间的独立性。
- 实现隔离。

- 针对特定微服务的可伸缩性。

本书旨在向读者探讨如何从单体应用程序过渡到微服务，应用适当的模式并向读者展示微服务的实现过程。相关内容涉及具有交互性的新闻门户网站、推荐系统、身份验证和授权系统。在本书中，当应用设计模式时，读者将经历迁移过程中的每一个步骤；当涉及微服务之间的通信层时，还会了解到内部和外部迁移过程中的具体实施方案。对此，读者首先需要了解一些重要的概念，以便有效地实现微服务。

1.1 理解应用程序

本书中的应用程序将成为所有概念解释和实际代码的来源。具体来说，基本系统是一个新闻门户网站，主要包括以下 3 方面内容：

- 新闻。
- 推荐系统。负责存储用户偏好设置，因此能够向用户提供特定的新闻，甚至根据用户配置文件形成唯一的主页。
- 用户，即基本的注册用户信息。

应用程序的所有业务内容都位于相同的源代码上，即单体软件。该应用程序是在 Django 框架上开发的，使用 PostgreSQL 作为数据库，并使用 Memcached 作为缓存系统（仅应用于数据库层）。

在此基础上，如果推荐级别超出负荷，那么所有应用程序都必须调整其规模，而不仅仅是引用推荐的部分，因为当前应用程序采用了单体结构。对应用程序来说，栈中所发生的变化其代价相对高昂。如果用户想更改所使用的缓存类型，那么所有的其他缓存内容都将丢失。

1.1.1 领域驱动设计

OOP 的概念也适用于微服务设计，这不仅体现于应用程序设计，还包括体系结构和业务逻辑的划分，但对于领域驱动设计（DDD）来说，情况则变得简单起来。

DDD 这一概念源自 Eric Evans 撰写的 Domain-Driven Design 一书。书中内容源自作者二十多年来 OOP 软件开发经验，其中列举了大量的模式分类。需要注意的是，OOP 不仅存在于继承机制、接口中，同时还涉及其他方面，如下所示：

- 代码与业务的一致性。
- 复用性。