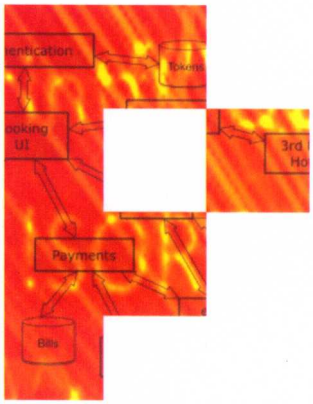
Python Microservices Development




# Python 微服务开发

[法] 塔里克·齐亚德（Tarek Ziadé）著
和坚 张渊 译

清华大学出版社

# Python 微服务开发

[法] 塔里克·齐亚德(Tarek Ziadé) 著

和坚 张渊 译

清华大学出版社

北 京

图书在版编目(CIP)数据

Python微服务开发 / (法)塔里克·齐亚德(Tarek Ziadé) 著；和坚，张渊 译. —北京：清华大学出版社，2019

书名原文：Python Microservices Development

ISBN 978-7-302-52412-0

Ⅰ. ①P… Ⅱ. ①塔… ②和… ③张… Ⅲ. ①软件工具—程序设计 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字(2019)第 041781 号

责任编辑：王　军　韩宏志
封面设计：周晓亮
版式设计：孔祥峰
责任校对：牛艳敏
责任印制：李红英

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 刷 者：北京鑫丰华彩印有限公司
装 订 者：三河市溧源装订厂
经　　销：全国新华书店
开　　本：170mm×240mm　　印　　张：17.75　　字　　数：358 千字
版　　次：2019 年 4 月第 1 版　　印　　次：2019 年 4 月第 1 次印刷
定　　价：59.00 元

产品编号：080803-01

# 译 者 序

近年来，“微服务”技术风靡全球。随着传统互联网和移动互联网的蓬勃发展，企业在快速迭代中积累了大规模服务化开发和运维经验。为通过提高 IT 的响应能力来提升竞争力，微服务架构成为传统企业的“救命稻草”。开发团队在变革中改造和重建技术架构，企业管理者们切实感受到微服务带来的巨大好处。

但不可否认，微服务架构带来了额外复杂性。对开发者提出了更高的要求，开发者不仅要编写业务代码，还需要具有部署和运维等能力。

十多年前，当 Ruby on Rails 和 Django 发布时，人们热衷于追逐包罗万象、开箱即用的全栈式 Web 框架，只需要运行几行脚手架命令，就能快速编写一个包含 Web 页面和数据存储的 Todo List 应用。但时至今日，那些小巧灵便的框架变得越来越受欢迎，开发者更愿意选择“微框架”，通过谨慎地综合运用不同工具来开发应用；随着开发的进行，灵活地升级或替换其中的某些部分。Flask 即是这种微框架之一。

本书模拟真实场景，从一个单体 Flask 应用开始，提出问题，分析和比较方案，作出权衡，最终解决问题(可能引出又一个“问题”——但并非当前的优先级)，逐渐拆分出多个微服务，解决随之而来的部署、监控、安全等新问题，在最后利用异步编程优化性能。期间牵涉大量工具，但本书并未详细介绍它们，只是“点到为止”。本书内容紧贴实用，面向想要开阔眼界和动手实践的开发者和架构师。通过阅读本书，读者将能对微服务开发实践有系统性认识，以便在开发早期进行规划和技术选型。

这里要感谢清华大学出版社的编辑们，他们为本书的翻译投入了巨大热情并付出了很多心血。没有他们的帮助和严谨的要求，本书不可能顺利付梓。

本书涉及大量的实践和专业术语，虽然译者倾力而为，力求译文准确易懂。但毕竟水平有限，失误在所难免，如有任何意见和建议，请不吝指正！本书主要章节由和坚和张渊翻译，参与本书翻译的还有张坤、吴邦、刘易斯、赵汝达、何睿智、刘娟娟、罗冬哲等。

最后，希望读者能通过本书对微服务开发有更深入的理解，并能继续探索解决已知问题的工具和方法。

# 作者简介

Tarek Ziadé是一位 Python 开发人员，在 Mozilla 的服务团队工作，已使用法语和英语撰写多本 Python 书籍。Tarek 创建了一个名为 Afpy 的法国 Python 用户组，现居住在法国第戎市郊区。在工作之余，Tarek 不忘陪伴家人。他另有两个爱好：跑步和吹小号。

可访问 Tarek 的个人博客(Fetchez le Python)，并在 Twitter 上关注他(@tarek_ziade)。还可在亚马逊上找到他撰写的另一本书 *Expert Python Programming*，该书已由 Packt 出版。

感谢Packt团队，以及帮助过我的以下技术精英：Stéfane Fermigier、William Kahn-Greene、Chris Kolosiwsky、Julien Vehent和Ryan Kelly。

感谢 Amina、Milo、Suki 和 Freya 给予我的爱和耐心支持。

希望在阅读时，你能享受到和我写本书时同样的乐趣！

# 审校者简介

自 20 世纪 90 年代末以来，William Kahn-Greene 一直在编写 Python 代码和构建 Web 应用。

他在 Mozilla crash ingestion pipeline 的 crash-stats 小组工作，并维护着多种 Python 库，如 bleach。在等待 CI 测试代码改动时，William 会摆弄木制品，照料他种的番茄，并烹饪 4 个人的饭食。

# 序　言

7 年前，当我开始在 Mozilla 工作时，为一些 Firefox 功能编写 Web 服务。它们中的一些最终蜕变成微服务。这种变化是随着时间的推移逐渐发生的。促成这种转变的第一个因素是，我们将所有服务转移到云厂商上，并开始与一些第三方服务交互。在云服务上托管应用时，微服务架构成为自然之选。另一个驱动因素是 Firefox 的 Account 项目。我们想在 Firefox 上为用户提供独立身份，以便用户与我们的服务交互。这样一来，所有服务必须与同一个身份提供方(Identity Provider)交互，一些服务器端部分开始重新设计为微服务，以便更高效地工作。

许多 Web 开发者有类似经历，或正在经历这个过程。我也相信 Python 是用来编写小型和高效微服务的最佳语言。Python 生态系统生机勃勃，最新的 Python 3 的特性让它在这个领域中能与过去 5 年中迅猛发展的 Node.js 一决高下。

这就是本书的全部内容。我想分享自己使用 Python 编写微服务的经验，并为此创建了一个简单示例——Runnerly。它位于 GitHub，可供你学习。你可在 GitHub 上与我直接交流，请指出你看到的任何错误，我们可共同切磋如何编写优秀的 Python 应用。

# 前　言

为将 Web 应用部署到云，代码需要与很多第三方服务进行交互。使用微服务架构，可构建能管理这些交互的大型应用。但这带来一系列挑战，每项挑战都有独特的复杂性。这本通俗易懂的指南旨在帮助你克服这些挑战。书中将介绍如何以最合理的方式设计、开发、测试和部署微服务，紧贴实用的示例将帮助 Python 开发者用最高效的方式创建 Python 微服务。阅读完本书，读者将掌握基于小型标准单元构建大型应用的技能。本书将使用成熟的最佳实践，并分析如何规避常见陷阱。此外，对于正将单体设计转换成新型“微服务”开发范式的社区开发者来说，本书也颇具价值。

## 本书内容

第 1 章“理解微服务”定义什么是微服务，以及微服务在现代 Web 应用中扮演的角色。还介绍 Python，并解释为什么用 Python 构建微服务是上佳之选。

第 2 章“Flask 框架”介绍 Flask 的主要特性。通过一个 Web 应用示例来展示这个框架，Flask 是构建微服务的基础。

第 3 章“良性循环：编程、测试和写文档”，介绍测试驱动开发方法和持续集成方法，以及在构建和打包 Flask 应用的实践中如何使用这些方法。

第 4 章“设计 Runnerly”基于应用特性和用户案例，首先构建一个单体应用，然后讲述如何将其拆解成微服务，并实现微服务之间的数据交互。还将介绍用来描述 HTTP API 的 Open API 2.0(ex-Swagger)规范。

第 5 章“与其他服务交互”介绍一个服务如何与后台服务进行交互，如何处理网络拆分问题，以及其他交互问题，另外介绍如何独立地测试一个服务。

第 6 章“监控服务”介绍如何在代码中添加日志和指标，清晰地掌控全局，确定发生了什么，并能追查问题和了解服务利用率。

第 7 章“保护服务”介绍如何保护微服务，如何处理用户身份验证、服务间身份验证以及用户管理。还介绍针对服务的欺诈和滥用，以及如何缓解这些问题。

第 8 章“综合运用”描述在终端用户界面中，如何设计和构建一个使用微服务的 JavaScript 应用。

第 9 章“打包和运行 Runnerly”描述如何打包、构建和运行整个应用。开发者必须能够将应用打包到一个开发环境中，确保所有部分都可以运行。

第 10 章“容器化服务”解释什么是虚拟化，如何使用 Docker，如何将服务做成 Docker 镜像。

第 11 章“在 AWS 上部署”首先介绍当前的云服务厂商和 AWS 世界。然后演示如何使用 AWS 来实例化一个基于微服务架构的应用。另外介绍 CoreOS，这是一个专门用于在云上发布 Docker 容器的 Linux 分支。

第 12 章“接下来做什么？”总结全书，在如何构建独立于云厂商和虚拟化技术的微服务问题上，给出一些提示来避免将鸡蛋放入同一个篮子里。还将帮助你巩固第 9 章中学到的知识。

## 阅读本书需要准备什么

要执行本书的命令和应用，系统需要安装 Python 3.*x*、virtualenv 1.*x* 和 Docker CE。正文中也会根据需要详细列出安装说明。

## 读者对象

作为一名开发者，如果你了解 Python 基本概念、命令行，以及基于 HTTP 的应用设计原则，并想学习如何构建、测试、扩展和管理 Python 3 微服务，那么本书适合你。阅读本书，你不必具有用 Python 编写微服务的任何经验。

## 本书约定

代码块按以下样式显示：

```
import time

def application(environ, start_response):
    headers = [('Content-type', 'application/json')]
    start_response('200 OK', headers)
return bytes(json.dumps({'time': time.time()}), 'utf8')
```

会用粗体来显示需要重点关注的代码：

```
from greenlet import greenlet
    def test1(x, y):
    z = gr2.switch(x+y)
print(z)
```

任何命令行的输入或输出都按以下样式显示：

```
docker-compose up
```

警告或重要注释会这样显示。

提示和技巧会这样显示。

## 读者反馈

欢迎读者提出反馈意见，这样我们能了解你对本书的看法，喜欢什么或不喜欢什么。反馈意见很重要，能帮助我们开发读者真正想了解的主题。只需要发邮件给feedback@packpub.com，并在邮件标题中提及本书，即可将反馈意见发给我们。

如果你是某个主题的专家，有兴趣写书，或愿意为写书做贡献，请到www.packtpub.com/authors 页面查阅作者指南。

## 下载示例代码

本书相关的代码放在 GitHub 上，网址是 https://github.com/PacktPublishing/Python-Microservices-Development。还有其他代码包和视频，欢迎通过 https://github.com/PacktPublishing/页面下载。

另外，读者可扫描本书封底的二维码直接下载代码。

下载文件后，用以下工具的最新版本来解压缩：

- 在 Windows 系统中使用 WinRAR /7-Zip。
- 在 Mac 系统中使用 Zipeg /iZip /UnRarX。
- 在 Linux 系统中使用 7-Zip /PeaZip。

## 勘误

尽管我们已经非常小心地确保内容的准确性，但还是会发生失误。如果你在书中发现了错误，可能是文本错误或代码错误，你能向我们报告此事，我们将不胜感激。通过这样做，可减少其他读者的阅读痛苦，并帮助我们改进本书的后续版本。如果你发现任何勘误，请访问 http://www.packtpub.com/submit-errata 页面来报告它们，选择你购买的书，单击 Errata Submission Form 链接，输入勘误的详细信息。一旦填写的勘误被确认，你的提交将被接受，然后勘误将被上传到我们的网站上，或添加到任何现有的勘误列表中。现有的勘误列表位于 Errata 标题的下面。

要查看之前提交的勘误，可访问 https://www.packtpub.com/books/content/support，然后在查找输入框内输入书名。要查找的信息会显示在 Errata 下面。

## 盗版行为

互联网上的盗版行为是所有媒体一直头疼的问题。在 Packt，我们将尽力处理盗版问题。我们会非常认真地对待版权和许可证的保护。如果你在互联网上遇到任何我们作品的非法拷贝，请立即向我们提供网址或网站名称，以便我们能采取补救措施。

请通过 copyright@packtpub.com 联系我们，并附带上有侵权嫌疑的材料。

非常感激你能帮助保护我们的作者以及我们的工作。这样我们可持续为你带来有价值的内容。

## 问题

关于本书的任何问题，欢迎通过 questions@packtpub.com 联系。

# 目　录

# 第1章

# 理解微服务

软件行业一直在尝试改良软件构建方式。不用说，与穿孔卡片时代相比，当今的软件构建过程已改良了很多。

微服务是过去几年里涌现出的一种改良方法，部分原因是很多公司想缩短发布周期。他们希望能尽快向客户交付新产品或新特性，希望通过迭代达到“敏捷”目的，希望能做到交付、交付、再交付。

如果有大量客户正使用你的服务，相对于发布之前反复测试产品，直接在运行的产品上推送一个试验性功能或移除一项无用的功能是更好的实践方法。

现在，Netflix 等公司正在倡导持续交付技术，这是一种可让小改动频繁上线，然后在一小部分用户中进行测试的技术。他们开发了很多工具，例如 Spinnaker(http://www.spinnaker.io/)就是通过自动执行尽可能多的步骤来更新生产环境，改动的特性通过相互独立的微服务发布到云上。

但如果阅读 Hacker News 或 Reddit，你会发现，梳理出哪些概念真正有用，哪些概念只是随波逐流的新闻体裁是非常困难的。

“写一篇承诺救赎的论文，使它成为‘结构化’或‘虚拟化’的东西，或使用抽象、分布式、高阶、可适用等概念，你几乎肯定在宣扬一门新邪教。”

——Edsger W. Dijkstra

本章将讲解什么是微服务，然后重点介绍多个使用 Python 实现微服务的方法。本章要点如下：

- 面向服务架构
- 使用单体方式构建应用
- 使用微服务方式构建应用

- 微服务的益处
- 微服务的缺陷
- 使用 Python 实现微服务

希望当读到本章结尾时，你能深入了解微服务的构建，并明白微服务是什么，以及如何使用 Python。

## 1.1 SOA 的起源

关于微服务有很多种定义，并没有一个官方标准。在试着解释微服务时，人们通常会提到面向服务架构(Service-Oriented Architecture，SOA)。

SOA 早于微服务，其核心原则是将应用组织成一个独立的功能单元，可远程访问并单独进行操作和更新。

——Wikipedia

上述定义中的每个单元都是一个独立服务，它实现业务的一个方面，并通过接口提供功能。

虽然SOA清楚地指出服务应当是独立的进程，但并未强制使用哪种协议进行交互，对如何部署和编排应用还是相当模糊的。

在少数专家于 2009 年发布的 SOA 宣言(http://www.soa-manifesto.org)中，甚至没有提及服务是否通过网络进行交互。

SOA服务可在同一个机器上使用套接字(socket)通过IPC(Inter-Process Communication，进程间通信)方式来交互，如使用共享内存、间接消息队列或远程过程调用(Remote Procedure Call，RPC)。选项非常广泛，只要没有在单个进程中运行所有应用，SOA就可以是任何东西。

常见的说法是，过年几年开始涌现的微服务是 SOA 的一种特定实现方式。它们实现了 SOA 的一些目标，也就是用独立组件来构建应用，组件之间进行着交互。

如果想给出微服务的完整定义，最好先分析一下大多数软件是如何设计架构的。

## 1.2 单体架构

让我们先通过一个非常简单的例子来介绍传统的单体应用：一个酒店预订网站。

除了静态的 HTML 内容，网站有一个预订功能，可让全球任何城市的用户通过网站预订酒店。用户可搜索酒店，然后用信用卡付款。

当用户搜索酒店网站时，应用将执行以下操作：

(1) 针对酒店数据库执行一些 SQL 查询。

(2) 给合作伙伴的服务发送 HTTP 请求，将更多酒店添加到列表中。

(3) 使用 HTML 模板引擎生成 HTML 结果页面。

一旦用户找到满意的酒店并单击“预订”，应用将执行以下步骤：

(1) 如有必要，在数据库中创建客户，然后进行身份验证。

(2) 通过与银行网络服务交互来完成付款。

(3) 按法律要求，应用需要将支付详情保存到数据库。

(4) 使用 PDF 生成器生成收据。

(5) 用电子邮件服务向用户发送一份用于确认的电子邮件。

(6) 用电子邮件服务将预订电子邮件转发给第三方酒店。

(7) 在数据库中添加用于追踪订单的条目。

上面是一个简化的过程，但紧贴实用。

应用和数据库的交互包括酒店信息、预订信息、支付信息和用户信息等。它还与外部服务进行交互来发送邮件，完成支付，从合作伙伴获取更多酒店。

在经典的 LAMP(Linux-Apache-MySQL-Perl/PHP/Python)架构中，每个传入的请求都会在数据库生成关联的 SQL 查询，以及少量对外部服务的网络请求，然后服务器使用模板引擎生成 HTML 响应。

图 1-1 描述了这种中心化架构。

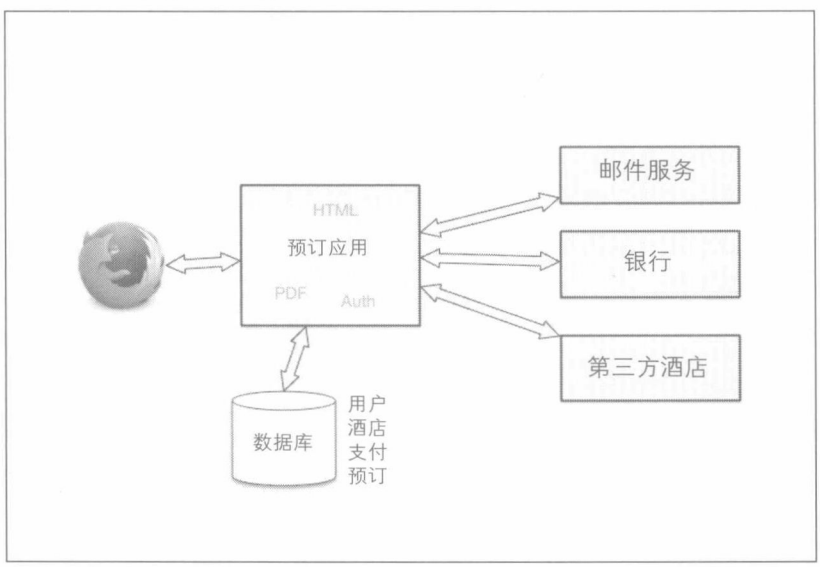



图 1-1　中心化架构