Web 开发与设计

Rust 全栈开发

[美] 帕布 • 埃什瓦拉(Prabhu Eshwarla) 著

王志强 刘 畅 译

清華大學出版社

北 京

北京市版权局著作权合同登记号 图字：01-2024-5557

Prabhu Eshwarla
Rust Servers, Services,and Apps
EISBN: 9781617298608

图书在版编目（CIP）数据

Rust 全栈开发 / (美) 帕布·埃什瓦拉 (Prabhu Eshwarla) 著 ; 王志强, 刘畅译. -- 北京 : 清华大学出版社, 2025. 3. -- (Web 开发与设计).
ISBN 978-7-302-68171-7

Ⅰ. TP312

中国国家版本馆 CIP 数据核字第 2025F1930T 号

责任编辑：王 军
封面设计：高娟妮
版式设计：恒复文化
责任校对：马遥遥
责任印制：沈 露

出版发行：清华大学出版社
网 址：https://www.tup.com.cn，https://www.wqxuetang.com
地 址：北京清华大学学研大厦 A 座 邮 编：100084
社 总 机：010-83470000 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：涿州汇美亿浓印刷有限公司
经 销：全国新华书店
开 本：170mm×240mm **印 张：**18.25 **字 数：**422 千字
版 次：2025 年 3 月第 1 版 **印 次：**2025 年 3 月第 1 次印刷
定 价：98.00 元

产品编号：106326-01

拥抱 Rust 全栈开发，开启高效安全编程新纪元

当今时代，软件开发领域风起云涌，新技术、新框架层出不穷。作为一名 Rust 开发者和技术爱好者，我深知开发者在享受技术红利的同时，也面临着日益严峻的挑战：如何在保障系统高性能、高并发的同时，兼顾代码的安全性和可维护性？如何在快速迭代的开发节奏中，降低错误发生的概率，提升开发效率？传统的编程语言在面对这些挑战时，往往显得力不从心。而 Rust 语言的出现，为我们提供了一种全新的、极具潜力的解决方案。

我有幸提前拜读了“Web 开发与设计”丛书的新作《Rust 全栈开发》，深感这是一本能够引领开发者进入 Rust Web 全栈开发新世界、并能切实解决上述挑战的优秀著作。

为什么你需要阅读这本书？

在信息爆炸、技术快速迭代的今天，选择一本适合自己的书，就如同获得了一位经验丰富的导师，能够帮助你少走弯路，快速掌握核心技能。

以下是我对本书内容的一个重点摘要，方便广大读者查看本书是否适合你。

1. 系统掌握 Rust Web 开发从后台到前端的全貌：无论你是 Rust 新手，还是有一定经验的开发者，本书都能为你提供系统全面的 Rust Web 开发知识。它从 Rust 基础、Web 服务概念讲起，逐步深入到 Actix Web 框架的使用、数据库操作、异步编程、错误处理、模板引擎、客户端测试，乃至 P2P 网络编程和 Docker 部署。

2. 从实践中学习，提升实战能力：本书最大的特点在于其实战性。它以一个名为 EzyTutors 的在线教育平台项目为贯穿全书的案例，从需求分析、架构设计到代码实现，手把手教你如何使用 Rust 构建一个真实的 Web 应用。通过这个项目，你不仅可以学习 Rust Web 开发的各项技术，更能够理解如何将这些技术应用到实际项目中，以解决实际问题。

3. 深入理解 Rust 核心优势：本书不仅教你“如何做”，还会深入讲解“为什么这样做”。通过本书，你将深刻理解 Rust 语言在内存安全、并发安全、零成本抽象等方面的独特优势，以及这些优势如何帮助你构建更加健壮、高效、安全的 Web 应用。

4. 掌握前沿技术，拓展技术视野。本书不仅停留在基础知识和常见用例。它前瞻性地引入了异步 Rust 和 P2P 网络编程的内容。

- **异步 Rust：**在现代 Web 应用中，处理高并发、I/O 密集型任务是常态。本书详细讲解了 Rust 的异步编程模型，包括 async/await 语法、future 的使用，以及如何利用 Tokio 运行时构建高效的异步 Web 服务。

- **libp2p：**去中心化应用是未来的趋势。本书专门用一章介绍了如何使用 Rust 和 libp2p 库构建 P2P 应用。libp2p 是一个模块化的网络协议栈，用于构建点对点应用程序，它提供了传输层、安全性、多路复用、对等路由、内容发现等功能。通过学习第 11 章，你将掌握构建去中心化应用的基础知识，为未来的 Web 3.0 开发做好准备。

阅读本书你将收获什么？

1. 扎实的 Rust Web 开发基础：你将掌握使用 Rust 编写 Web 服务、处理 HTTP 请求、操作数据库、渲染模板、进行客户端测试等核心技能。

2. 构建高性能、高并发 Web 应用的能力：你将理解 Rust 在并发编程方面的优势，学会如何利用 Rust 的异步编程模型构建高吞吐量、低延迟的 Web 服务。

3. 编写安全、可靠代码的信心：你将深入理解 Rust 的所有权系统和借用检查器，掌握编写内存安全、无数据竞争代码的方法。

4. 解决实际问题的实战经验：通过 EzyTutors 项目的实践，你将获得构建真实 Web 应用的宝贵经验，提升你的项目开发能力。

5. 拥抱未来技术趋势的前瞻视野：本书介绍的异步编程，libp2p，为你打开通向区块链技术的大门。

寄语

Rust 语言近年来发展迅猛，凭借其独特的优势，已成为系统编程、Web 开发、嵌入式开发等领域的新宠。本书的出版，为中文开发者学习和掌握 Rust 全栈开发提供了难得的机遇。

我强烈推荐每一位对 Rust 语言、Web 开发、以及对构建高性能、安全、可靠的应用程序感兴趣的开发者阅读本书。我相信，通过本书的学习与实践，你定能在 Rust 的世界中乘风破浪，构建出属于自己的精彩应用！

张汉东
资深独立咨询师，《Rust 编程之道》的作者

译 者 序

在当今这个信息爆炸、技术迅速更新换代的时代，编程语言的迭代似乎永不停歇。然而，Rust 这门语言以其独特的魅力和强大的功能，成功地在众多语言中脱颖而出。我在早期的职业生涯，一直使用 C++进行程序开发。近年来，当接触到 Rust，并在几个项目实际应用后，我便被 Rust 的诸多特性深深吸引。Rust 的安全性、性能和并发能力让我在许多新项目中毫不犹豫地选择了它，来替代 C++。

我有幸翻译了这本《Rust 全栈开发》，希望通过我的努力，将 Rust 的精髓和魅力介绍给更多对技术充满热情的朋友们。

在翻译本书的过程中，我深刻感受到了 Rust 语言的严谨与优雅。它不仅在语法上追求简洁，更在运行时保证了内存安全，有效避免了传统编程语言常见的内存泄漏和数据竞争问题。虽然对初学者来说，Rust 的所有权系统和生命周期概念可能有些难以理解，但这正是 Rust 确保内存安全的核心。随着我对这些概念的深入学习和实践，我逐渐发现它们其实是既直观又强大的工具，能够帮助开发者编写出既安全又高效的代码。

本书内容丰富，涵盖了 Rust 的基础知识和实战项目，能够帮助读者打下坚实的基础，并引导他们将理论知识应用到实践中，逐步提升编程技能。无论是数据库编程、网络编程、错误处理还是 Web 应用开发，Rust 都能提供稳定而高效的解决方案。

在翻译这本书的过程中，我努力保持了原作的准确性和可读性，同时也尽量让语言更加贴近中文读者的阅读习惯。我相信，无论是 Rust 的初学者还是有一定基础的开发者，都能从本书获得宝贵的知识和启发。

最后，我要特别感谢本书的二译刘畅先生，没有他的强有力帮助，就无法如此顺利地完成本书的翻译工作。同时，我也要感谢我的爱人沈平女士，以及清华大学出版社的王军老师，还有所有支持和帮助我完成这项工作的人。虽然翻译是一项既费时又费力的工作，但是每每想到能将优秀的知识传播给更多的人，我就会深感荣幸和满足。希望本书能成为读者学习 Rust 编程的良师益友。

愿 Rust 在国内的生态越来越繁荣，愿大家的代码都无 bug，愿我们的技术之路越走越宽广。

关 于 作 者

Prabhu Eshwarla 目前是一家初创公司的 CTO，该公司正在使用 Rust 构建一层区块链。Prabhu 对 Rust 编程语言有浓厚的兴趣，自 2019 年 7 月以来一直在积极地学习和研究 Rust 语言。他之前曾在惠普担任软件工程师和技术管理等职位。

致　谢

在快节奏的科技领域中撰写《Rust 全栈开发》，无疑需要投入大量的时间和精力。

首先，我要感谢我的家人，他们牺牲了大量的时间支持我写书。我对他们的感激之情溢于言表。

同时，我还要感谢 Manning 出版团队的许多人，他们以各种方式帮助我，使我能够以更高的效率完成这本书的编写。我要感谢 Mike Stephens 给我这个宝贵的机会。感谢各位编辑，特别是 Elesha Hyde，她坚定的支持、指导和耐心，帮助我克服了无数挑战，最终完成本书。非常感谢制作人员，促成了这本书最终的呈现。最后，我真诚地感谢技术编辑 Alain Couniot。如果没有他们的辛勤付出，就不会有这本书。感谢 Alain 耐心而细致地审阅章节，改进代码和提升内容的技术质量、相关性，为读者带来更好的阅读体验!

最后，我还要感谢所有的审稿人，他们对稿件提出了宝贵的意见：Adam Wendell、Alessandro Campeis、Alex Lucas、Bojan Djurkovic、Casey Burnett、Clifford Thurber、Dan Sheikh、David Paccoud、Gustavo Gomes、Hari Khalsa、Helmut Reiterer、Jerome Meyer、Josh Sandeman、Kent R. Spillner、Marcos Oliveira、Matthew Krasnick、Michal Rutka、Pethuru Raj Chelliah、Richard Vaughan、Slavomir Furman、Stephane Negri、Tim van Deurzen、Troi Eisler、Viacheslav Koryagin、William Wheeler 和 Yves Dorfsman。我还要感谢 MEAP 的读者，他们在 liveBook 论坛上提了许多有趣的问题和意见，并发现了一些文字错误。

前　言

构建高性能网络服务对于任何一种编程语言来说都是挑战。Rust 所具有的特性可以大大降低这些“挑战”的门槛。

事实上，Rust 从一开始就被设计成一种用于高并发和安全系统的语言。尽管已有几种编程语言(如 C、C++、Go、Java、JavaScript 和 Python)用于开发高性能和可靠的网络服务，且这些服务既可以在单个节点上运行，也可以作为多节点分布式系统的一部分，无论是在内部部署的数据中心或云中，但是以下几点仍旧让 Rust 成为一种更佳的选择。

- 占用空间小(内存和 CPU 使用率完全可控)
- 安全性和可靠性(编译器保障内存和数据竞争的安全性)
- 低延迟(无垃圾收集器)
- 现代语言特性

本书讲授了如何通过各种工具、技术，用 Rust 构建高效可靠的网络服务和应用程序；还介绍了 Rust 中的网络服务和 Web 应用程序，从标准库原语构建的基本单节点、单线程服务器到高级多线程异步分布式服务器，跨越协议栈的不同层。内容涵盖：

- Rust 标准库中的网络原语
- 基本 HTTP 服务
- 由关系数据库支持的 REST API 服务器
- P2P 网络分布式服务器
- 高并发异步服务器

本书通过教程式方法讲解如何使用 Rust 开发 Web 服务和应用程序，通过一个个示例，不同的章节，逐步强化、深入。希望你能在本书中找到乐趣。本书内容丰富，可实践性强，对实际工作定有助益。

关 于 本 书

本书并非参考指南，更像一个引言，指引大家如何利用 Rust 开发网络服务。本书采用实践教程的形式，最大限度地方便读者的学习和理解。

本书读者对象

本书主要面向从事服务器端、Web 后端和 APl 开发，或对这些感兴趣的后端软件工程师；想探索用 Rust 替代 Go、Java 或 C++的分布式系统工程师；以及从事机器学习、人工智能、物联网、图像/视频/音频处理等领域的低延迟服务器、应用程序、实时系统的后端开发的软件工程师。

要想从本书中获得更大的收获，最好既有后端开发经验，又熟悉 Rust。如果你是后端开发人员，最好熟练掌握 Web 服务概念，包括 HTTP、JSON、使用 ORM 访问数据库以及任何高级语言(如 Java、JavaScript、Python、C#、Go 或 Ruby)的 API 开发。如果你是进阶的初学者或中级 Rust 程序员，最好了解如何复制和修改开源教程和仓库，并熟悉 Rust 的以下方面：

- Rust 基础(数据类型)、用户定义的数据结构(结构、枚举)、函数、表达式和控制循环(if、for 和 while 循环)
- 不可变性、所有权、引用和借用
- 使用 Result 和 Option 结构进行错误处理
- Rust 中的基本功能结构
- Rust 工具链，包括用于构建和依赖性管理的 Cargo，以及代码格式化、文档生成和自动化测试工具

更多 Rust 的新资讯参阅本节最后的“其他在线资源”。

本书学习路线图

本书由一系列实践项目组成，每个项目都涉及可以选用 Rust 开发的特定类型的网络服务。你将通过检查代码和编写代码的方式来学习。我们会在项目中解读相关理论知识，并鼓励你尝试一些编码练习。

本书共有 12 章，分为三部分。第 I 部分介绍了 Web 应用程序的基本概念，为其他部分奠定了基础。其中我们将开发一个日益复杂的 Web 应用程序后端，最终达到接近生产

就绪的阶段。第Ⅰ部分由以下几章组成。

- 第 1 章介绍关键概念，如分布式体系结构和 Web 应用程序，以及将在本书中开发的应用程序。最后，总结 Rust 的优势，并提供一些关于何时使用和不使用 Rust 的提示。
- 第 2 章是本书其余部分的“热身”章节，会开发一些基于 TCP 的组件，以了解 Rust 在该领域的能力。
- 第 3 章展示如何在已经存在的丰富生态系统中用 Rust 和 crates 构建 RESTful Web 服务(并持续迭代)；解释什么是应用程序状态以及如何管理它。
- 第 4 章论述在数据库中持久化数据的必要性，将使用一个简单但高效的与 SQL 数据库交互的 crate。
- 第 5 章重点讨论在调用已开发的 Web 服务时，应对不可预见情境的关键点。
- 第 6 章旨在展示在用 Rust 开发时，随着 Web 服务 API 变得越来越强大和复杂，重构代码如何变得更加简单和安全。

第Ⅱ部分介绍处理 Web 应用程序的另一部分——前端及其图形用户界面(GUl)。本书采用了一种简单的方法，该方法依赖于服务器端渲染，而非在浏览器中运行的复杂 Web 框架。本部分由以下三章组成。

- 第 7 章详细介绍选定的服务器端渲染框架，并阐述如何引导用户输入以及如何处理项目列表。此外，还会展示如何与第Ⅰ部分开发的后端 Web 服务进行交互。
- 第 8 章着重探讨在服务器端使用的模板引擎，详细展示如何借助几个表单来实现用户注册功能。
- 第 9 章会更加深入地介绍 Web 应用程序，如用户身份验证、路由，以及如何高效地使用 RESTful Web 服务以 CRUD(创建、读取、更新、删除)的方式维护数据。

第Ⅲ部分包含三个高级主题，这些主题与前面章节已经构建的 Web 服务和 Web 应用程序没有直接关联，但对于有兴趣构建复杂的 Rust 服务器并为其生产部署做准备的人来说，非常重要。

- 第 10 章介绍异步编程以及 Rust 如何支持异步编程范式。然后，通过几个简单的例子说明异步编程。
- 第 11 章展示使用 Rust 和精选的 crate 开发 P2P 应用程序时 Rust 的强大功能。
- 第 12 章详细介绍如何将 Web 应用程序打包成 Docker 映像，以便在各种环境(从本地到云端)中进行部署。

关于代码

本书的源代码可在 GitHub 上获取：https://github.com/peshwar9/rust-servers-services-apps。也可扫描封底二维码下载。全部代码按章节存放，且为各章代码的最终形式。

对于有 Rust 开发经验的人来说，配置环境应该不难：所需要的只是标准的 Rust 工具链和一个合适的 IDE(集成开发环境)，如 VS Code，以及一些 Rust 扩展(建议使用 Rust 扩展包；Rust 语法和 Rust 文档查看器也不错)。为了尽可能从 GitHub 和版本控制中受益，

还应该安装 Git，但这不是强制性的，毕竟还可以从 GitHub 上以 zip 存档的形式下载源代码。这本书包含了许多源代码的例子，既有带编号的代码清单，也有直接引用的代码。在这两种情况下，源代码都以这样固定宽度的字体进行格式化，以将其与普通文本分开。

在书中，大多原始源代码已被重新格式化；添加了换行符并重新调整了缩进，以适应排版。在极少数情况下，代码清单中还用到了换行标记(➥)。此外，若文中详细讲解过代码，则代码清单中不再对相应的源代码做注释。许多代码清单中都带有代码注释，用于突出重要概念。

其他在线资源

Rust 编程语言由 Rust 创建者管理的优质在线资源以及其他独立资源(如 Medium)提供支持。以下是一些推荐资源。

- *The Rust Book*——Rust 开发者的官方指南(www.rust-lang.org/learn)。这本在线书籍有一个关于编写网络服务程序的部分，相对比较基础。
- *Rust by Example*——*The Rust Book* 的姊妹篇(https://doc.rust-lang.org/rust-by-example/index.html)。
- *The Cargo Book*——另一本来自 Rust 官网的书，专门介绍 Cargo 包管理(https://doc.rust- lang.org/cargo/index.html)。

关于封面插图

本书封面上的形象源自“Homme Toungouse”，又名“通古斯人”，取自 Jacques Grasset de Saint-Sauveur 于 1788 年出版的一本合集。其中每幅插图都由手工精细绘制及着色而成。

在过去，很容易通过穿着来识别人们的居住地、职业和社会地位。Manning 通过将再现几个世纪前地区文化的丰富多样性的藏画做封面，来赞颂计算机行业的创造力和积极性。

目　录

第 I 部分

Web 服务器及 Web 服务

Rust 是一种卓越的编程语言，如今正日益受到广泛认可。它最初被定位为系统编程语言，与 C 语言和 Go(lang)语言等齐名。的确，它正逐渐融入 Linux 内核：目前仅用于驱动程序和模块开发，但其核心优势——出色的表现力、内存安全性和性能——无疑将为其在操作系统中更多关键领域的应用敞开大门。在浏览器和无服务器的云环境中，Rust 正以较慢的速度渗透到仍属于保密范畴的 WebAssembly(WASM)中。

与 Go 语言一样，富有创新精神的开发者已经证明，Rust 不仅限于系统编程，还可以用于由数据库支持的高效 Web 应用程序后端开发。

本书的第 I 部分将使用 REST Web 服务开发一个简单而有代表性的 Web 应用程序，该服务由相关数据库支持。UI 方面的问题放在本书的第 II 部分讲解。本部分为 Web 应用程序奠定基础——毕竟既要拥有广阔的视野，也要踏踏实实从小事做起。然后讨论日益专业化的主题，如数据库持久化、错误处理以及 API 维护和重构。

学完这一部分后，便能够使用 Rust 设置和开发强大的应用程序后端，包括路由和错误处理。然后，开始第 II 部分的学习。

第1章

为什么 Rust 可用于 Web 应用程序

本章内容

- 现代 Web 应用程序简介
- 为 Web 应用程序选择 Rust
- 可视化示例应用程序

通过互联网运行的 Web 应用程序构成了现代企业和人类数字生活的支柱。作为个人，我们使用以消费者为中心的应用程序进行网络社交和通信、电子商务购物、旅行预订、支付和理财、教育和娱乐等。同样，以业务为中心的应用程序在企业中几乎用于所有职能和流程。

当今的 Web 应用程序是极其复杂的分布式系统。这些应用程序的用户通过 Web 或移动前端用户界面进行交互。但用户很少看到后台服务和软件基础架构组件的复杂环境，这些组件可以响应用户在应用程序界面发出的请求。主流的消费类应用程序拥有分布在全球数据中心的数千个后端服务和服务器。应用程序的每项功能都可以在不同的服务器上执行，使用不同的设计方案，用不同的编程语言编写，并且位于不同的地理位置。无缝衔接的应用内用户体验让事情看起来如此简单，但开发现代 Web 应用程序绝非易事。

发推特、在 Netflix 上看电影、在 Spotify 上听歌曲、预订旅行、订餐、玩在线游戏、叫出租车或使用众多在线服务，都是人们日常生活的一部分。如果没有分布式 Web 应用程序，企业和现代数字社会将停滞不前。

注意： *网站提供业务的相关信息。Web 应用程序为客户提供服务。*

本书将介绍使用 Rust 设计和开发分布式 Web 服务和应用程序所需的概念、技术和工具，这些服务通过标准互联网协议进行通信。在此过程中，还将通过实例展示 Rust 核心概念的实际应用。

如果你是 Web 后端软件工程师、全栈应用程序开发人员、云或企业架构师、技术产品的 CTO，或者只是一个对构建极其安全、高效、高性能且不会产生过高的操作及维护成本的分布式 Web 应用程序感兴趣的好奇学习者，那么本书非常适合你。通过在本书中开发一个工作示例，将向你展示如何使用纯 Rust 构建 Web 服务和传统 Web 应用程序前端。

正如你在各个章节中所注意到的，Rust 是一种通用语言，可以有效地支持许多不同类型应用程序的开发。本书介绍了一个单一的应用程序，但所演示的技术适用于使用相同或其他 crate 的许多其他情况(库在 Rust 术语中称为 crate)。

本章将回顾分布式 Web 应用程序的关键特性，了解 Rust 如何以及在何处发挥作用，并概述将在本书中共同构建的示例应用程序。

1.1 现代 Web 应用程序简介

首先介绍现代分布式 Web 应用程序的结构。分布式系统的组件可以分布在多个不同的计算处理器上，通过网络进行通信，并同时进行工作。从技术上讲，家用计算机就类似于一个网络分布式系统(具有现代多 CPU 和多核处理器)。

主流的分布式系统类型包括

- 分布式网络，如电信网络和互联网；
- 分布式客户端-服务器应用程序(大多数基于网络的应用程序都属于这一类)；
- 分布式 P2P 应用程序，如 BitTorrent 和 Tor；
- 实时控制系统，如空中交通管制和工业控制等；
- 分布式服务器基础设施，例如云计算、网格计算和其他形式的科学计算。

分布式系统大致由三部分组成：分布式应用程序、网络协议栈以及硬件和操作系统基础设施。

分布式应用程序可以使用多种网络协议在其组件之间进行内部通信。然而，由于其简单性和通用性，HTTP 是当今 Web 服务或 Web 应用程序与外界通信的压倒性选择。

Web 应用程序是使用 HTTP 作为应用层协议的程序，并为人类用户提供可通过标准 Internet 浏览器访问的功能。当 Web 应用程序不是单一的，而是由数十或数百个通过网络协作和通信的分布式组件组成时，这样的 Web 应用程序就被称为分布式 Web 应用程序。大规模分布式 Web 应用程序的示例包括 Facebook 和 Twitter 等社交媒体应用程序、Amazon 和 eBay 等电子商务网站、Uber 和 Airbnb 等共享经济应用程序、Netflix 等娱乐网站，甚至来自 AWS、Google 和 Azure 等提供商的用户友好型的云端应用。

图 1.1 是现代 Web 应用程序的分布式系统架构的代表性逻辑视图。在现实世界中，此类系统可以分布在数千台服务器上，但在此图中，只展示了通过网络协议栈连接的三台服务器。这些服务器可能全部位于单个数据中心内，也可能分布在云上。每个服务器内均显示了硬件和软件组件的分层视图。

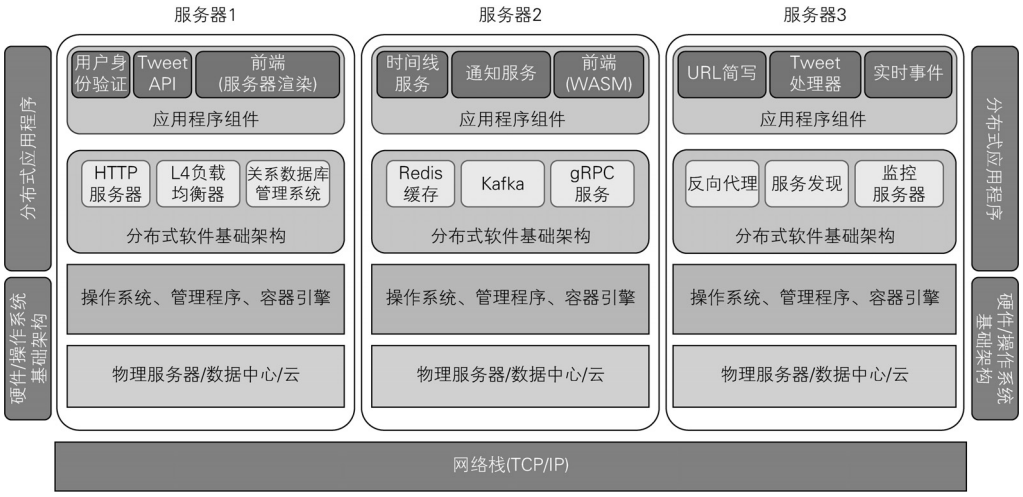

图 1.1　社交媒体应用程序的简化分布式系统架构

- **硬件和操作系统基础设施组件**——这些组件包括物理服务器(位于数据中心或云上)、操作系统以及虚拟化或容器运行时等。嵌入式控制器、传感器和边缘设备等也可以划分到这一层(未来，当超市货架上添加或移除带有 RFID 标签的商品库存时，连锁超市的社交媒体就会向关注者推送消息)。
- **网络协议栈**——网络协议栈由四层互联网协议套件组成，构成了分布式系统组件的通信主干，允许跨物理硬件相互通信。四个网络层(按抽象层次从低到高排序)是链路/接入层、网络层、传输层、应用层。

前三层通常在硬件或操作系统级别实现。对于大多数分布式 Web 应用程序，HTTP 使用的主要是应用层协议。REST、gRPC 和 GraphQL 等流行的 API 协议都使用 HTTP。有关 Internet 协议套件的更多详情，可访问 https://tools.ietf.org/id/draft-baker-ietf-core-04.html。

- **分布式应用程序**——分布式应用程序是分布式系统的子集。现代 *n* 层分布式应用程序由以下各项组合构建。
  - **应用程序前端**——这些可以是(在 iOS 或 Android 上运行的)移动应用程序或在互联网浏览器中运行的 Web 前端。这些应用程序前端与(通常位于数据中心或云平台的)驻留在远程服务器上的应用程序后端服务进行通信。终端用户与应用程序前端交互。
  - **应用程序后端**——其包含应用程序业务规则、数据库访问逻辑、图像或视频处理等计算密集型流程以及其他服务，被部署为物理机或虚拟机上运行的单独进程(例如 UNIX/Linux 上的 systemd 进程)，或者被部署为由容器编排环境(如 Kubernetes)管理的容器引擎(如 Docker)中的微服务。与应用程序前端不同，应用程序后端通过编程接口(API)公开其功能。应用程序前端与应用程序后端服务交互以代表用户完成任务。

- **分布式软件基础设施**——这包括为应用程序后端提供支持服务的组件。示例包括协议服务器、数据库、键/值存储、缓存、消息、负载均衡器和代理、服务发现平台以及使用的其他此类基础设施组件，这些组件用于分布式应用程序的通信、操作、安全和监控。应用程序后端与分布式软件基础设施交互，以实现服务发现、通信、生命周期支持、安全、监控等。

现在，你已经对分布式 Web 应用程序有了大致的了解，下面接着介绍使用 Rust 构建它们的好处。

1.2 为 Web 应用程序选择 Rust

Rust 可用于构建所有的三层分布式应用程序：前端、后端服务和软件基础设施组件。但每一层都涉及一组不同的关注点和特征。在讨论 Rust 的优势时，了解这些非常重要。

例如，客户端前端处理用户界面设计、用户体验、跟踪应用程序状态的变化和在屏幕上渲染更新视图以及构建和更新文档对象模型(DOM)等方面。

后端服务需要设计良好的 API，以减少往返次数，实现高吞吐量(以每秒请求数衡量)，在不同负载下保持短响应时间、对于视频流和在线游戏等应用提供低且可预测的延迟，同时降低内存和 CPU 占用率并支持服务发现和可用性。

软件基础设施层主要关注极低的延迟、网络和其他操作系统资源的底层控制、CPU 和内存的节约使用、高效的数据结构和算法、内置安全性、短启动和关闭时间，以及用于应用程序后端服务的符合人体工程学的 API。

诚然，单个 Web 应用程序包含至少具有三组特征和要求的组件。虽然其中每一个都可以独立成书，但本书将尽可能全面地介绍并重点关注一组对 Web 应用程序的所有三层都有广泛益处的共同特征。

1.2.1 Web 应用程序的特点

Web 应用程序可以分为不同的类型。

- 核心应用程序，例如车辆和智能电网的自主控制、工业自动化以及高速交易应用程序，其中成功的交易取决于快速可靠地响应输入事件的能力。
- 大批量交易和消息传递基础设施，例如电子商务平台、社交网络和零售支付系统。
- 近实时应用程序，例如在线游戏服务器、视频或音频处理、视频会议和实时协作工具。

这些应用程序的共同要求如下：

- 安全、可靠
- 节约资源

- 必须最大限度地减少延迟
- 支持高并发

此外，以下是此类服务应具备的要求：

- 快速的启动和关闭时间
- 易于维护和重构
- 必须提高开发者生产力

以上所有要求都可以在单个服务级别和架构级别上得到满足。例如，单个服务可通过采用多线程或异步 I/O 来实现高并发。同样，通过在负载均衡器后面添加多个服务实例来处理并发负载，可以在架构级别实现高并发性。本书在讨论 Rust 的好处时，仅关注单个服务级别，毕竟架构级别选项对于所有编程语言都是通用的。

1.2.2　Rust 对 Web 应用程序的好处

现代 Web 应用程序由 Web 前端、后端和软件基础设施组成。尽管 Rust 对于开发 Web 前端的好处，无论是替换还是补充部分 JavaScript 代码，都是当今的热门话题，但本书对此不做讨论。

这里将主要关注 Rust 对应用程序后端和软件基础设施服务的好处。Rust 满足 1.2.1 节中确定的此类服务的所有关键要求。具体如下。

1. Rust 是安全的

谈及程序安全时，需要考虑三个不同的方面：类型安全、内存安全和线程安全。

就类型安全而言，Rust 是一种静态类型语言。类型检查(验证并强制执行类型约束)发生在编译时，因此必须在编译时确定变量的类型。如果未指定变量的类型，编译器就会尝试推断它。如果无法做到，或者存在冲突，编译器便会告知并阻止用户继续。在这种情况下，Rust 类似于 Java、Scala、C 和 C++。Rust 中的类型安全由编译器强制执行，并提供有用的错误消息。这有助于消除一整类运行时错误。

内存安全可以说是 Rust 编程语言最独特的存在。为了公正地对待这个话题，不妨来详细分析一下。

主流编程语言可以根据它们提供内存管理的方式分为两组。第一组包括具有手动内存管理的语言，例如 C 和 C++。第二组包括具有垃圾收集器的语言，如 Java、C#、Python、Ruby 和 Go。

因为开发人员并不完美，手动内存管理意味着接受一定程度的风险，所以程序缺乏正确性。因此，对于那些不需要低级内存控制，峰值性能也不是主要目标的语言，垃圾收集在过去的 20 到 25 年里已经成了主流特性。垃圾收集使程序比手动管理内存更安全，但在执行速度、额外计算资源的消耗以及程序执行可能停顿方面存在局限性。此外，垃圾收集仅处理内存，而不处理其他资源，例如网络套接字和数据库句柄。

Rust 是第一个提出替代方案的流行语言——不需要垃圾收集的自动内存管理和内存安全，而是通过独特的所有权模式实现该目标。Rust 使得开发人员能够控制其数据结构的内存布局并明确所有权。Rust 的资源管理所有权模型围绕 RAII(资源获取即初始化，一

种 C++编程概念)和支持安全内存使用的智能指针建模。

在此模型中，Rust 程序中声明的每个值都会分配一个所有者。一旦某个值被赋予另一个所有者，就不能再为原始所有者使用。当值的所有者超出范围时，该值将自动销毁(内存被释放)。

Rust 还可以授予对值、另一个变量或函数的临时访问权限，这称为借用。Rust 编译器(特别是借用检查器)确保对值的引用不会比借用的值存在更久。要借用值，必须使用&运算符(称为引用)。引用有两种类型：不可变引用&T(允许共享但不允许更改)和可变引用&mut T(允许更改但不允许共享)。Rust 确保只要有一个对象的可变借用，就不会再有该对象的其他借用(无论是可变的还是不可变的)。所有这些都在编译时强制执行，从而消除了涉及无效内存访问的一整类错误。

总之，可以使用 Rust 进行编程，而不必担心无效的内存访问，并且不需要垃圾收集器。Rust 提供编译时保证来防止以下类别的内存安全错误。

- 空指针解引用，程序因解引用的指针为空而崩溃。
- 段错误，程序尝试访问内存的受限区域。
- 悬空指针，与指针关联的值不再存在。
- 由于程序访问数组开头之前或超出数组末尾的元素而导致缓冲区溢出。Rust 迭代器不会越界运行。

在 Rust 中，内存安全和线程安全(这似乎是两个完全不同的问题)使用相同的基本所有权原则来解决。为了类型安全，Rust 默认情况下确保不存在由于数据竞争而导致的未定义行为。虽然某些 Web 开发语言能提供类似的保证，但 Rust 更进一步，能防止用户在线程之间共享非线程安全的对象。Rust 将某些数据类型标记为线程安全并为用户强制执行。大多数其他语言并不区分线程安全和线程不安全的数据结构。Rust 编译器明确防止所有类型的数据竞争，这使得多线程程序更加安全。

以下是一些深入了解 Rust 安全性的参考资料。

- Send 和 Sync trait：http://mng.bz/Bmzl
- Rust 无畏并发：http://mng.bz/d1W1

除了上面讨论过的，Rust 还有一些其他功能可以提高程序的安全性。

- Rust 中的所有变量默认都是不可变的，在改变任何变量之前需要显式声明。这强制开发人员思考数据的修改方式和位置以及每个对象的生命周期。
- Rust 的所有权模型不仅涵盖了内存管理，还扩展到了对其他资源的管理，例如网络套接字、数据库和文件句柄以及设备描述符。不使用垃圾收集器可以避免不确定的行为。
- match 语句(相当于其他语言中的 switch 语句)很详尽，这意味着编译器强制开发人员处理 match 语句中的每种可能的变体。这可以避免开发人员无意中错过处理某些可能导致意外运行时行为的代码流路径。
- 代数数据类型的存在使得更容易以简洁且可验证的方式表示数据模型。

Rust 的静态类型系统、所有权和借用模型、缺少垃圾收集器、不可变默认值以及详尽的模式匹配，所有这些都由编译器强制执行，为 Rust 开发安全应用程序提供了不容置

否的优势。

2. Rust 是资源高效的

多年来，诸如 CPU、内存和磁盘空间等系统资源越发便宜。虽然事实证明这对于分布式应用程序的开发和扩展非常有益，但也带来了一些问题。首先，软件团队普遍倾向于使用更多硬件来解决可扩展性的挑战——更多的 CPU、更多的内存和更多的磁盘空间。这是通过向服务器添加更多的 CPU、内存和磁盘资源(垂直扩展，又名向上扩展)或向网络添加更多计算机以分担负载(水平扩展，又称为横向扩展)来实现的。

这些方法变得流行的原因之一是当今主流 Web 开发语言的局限性。JavaScript、Java、C#、Python 和 Ruby 等高级 Web 开发语言不支持细粒度控制内存来限制内存使用。许多编程语言没有很好地利用现代 CPU 的多核架构。动态脚本语言无法进行有效的客户端内存分配，原因是变量的类型仅在运行时可知，不可能进行优化，这与静态类型语言不同。

Rust 提供了以下固有的特性，可以创建资源高效的服务。

- 源于其内存管理的所有权模型，Rust 很难编写泄露内存或其他资源的代码。
- Rust 允许开发人员严格控制其程序的内存布局。
- Rust 没有消耗额外 CPU 和内存资源的垃圾收集器。垃圾收集代码通常在单独的线程中运行并消耗资源。
- Rust 没有大型、复杂的运行时环境。这为开发人员提供了巨大的灵活性，即使在资源不足的嵌入式系统和微控制器(例如家用电器和工业机器)上运行 Rust 程序也是如此。Rust 可以在没有内核的裸机中运行。
- Rust 不鼓励对堆分配的内存进行深度复制，并且提供了各种类型的智能指针来优化程序的内存占用。Rust 没有运行时，并因此成为少数适合资源极少环境的现代编程语言之一。

Rust 结合了最好的静态类型、细粒度内存控制、多核 CPU 的高效使用以及内置异步 I/O 语义，所有这些都使其在 CPU 和内存利用率方面非常高效。所有这些都意味着降低服务器成本，并减轻小型和大型应用程序的运营负担。

3. Rust 的低延迟

往返网络请求和响应的延迟取决于网络延迟和服务延迟。网络延迟受传输介质、传播距离、路由器效率、网络带宽等多种因素影响。服务延迟取决于许多因素，例如处理请求时的 I/O 延迟、是否存在引入非确定性延迟的垃圾收集器、虚拟机暂停、上下文切换次数(例如，在多线程中)、序列化和反序列化成本，等等。

从纯编程语言的角度来看，Rust 作为系统编程语言，由于其底层硬件控制能力，提供了低延迟。Rust 没有垃圾收集器或运行时，并且原生支持非阻塞 I/O，拥有高性能异步(非阻塞)I/O 库和运行时的良好生态系统，同时将以零成本抽象作为语言的基本设计原则。此外，在 Rust 中，变量默认存储在栈上，这使得管理速度更快。

几个不同的基准测试显示，对于类似的工作负载，遵循典型编程风格的 Rust 与 C++ 性能相当，这比主流 Web 开发语言的结果更快。

4. Rust 已实现无所畏惧的并发

前文从程序安全的角度介绍了 Rust 的并发特性。接下来从更高的多核 CPU 利用率、吞吐量以及应用程序和基础设施服务性能的角度来看看 Rust 的并发性。

Rust 是一门并发友好的语言，支持开发人员利用多核处理器的强大功能。Rust 提供两种类型的并发：经典的多线程和异步 I/O。

- 多线程——Rust 的传统多线程支持提供共享内存和消息传递并发性。为共享值提供类型级保证。线程可以借用值、取得所有权并将值的范围转移到新线程。Rust 还提供数据竞争安全性，防止线程阻塞，从而提高性能。为了提高内存效率并避免复制跨线程共享的数据，Rust 提供了引用计数作为跟踪其他进程或线程对变量的使用的机制。当计数达到零时，该值将被删除，从而提供安全的内存管理。此外，Rust 中还可以使用互斥(mutex)来实现跨线程的数据同步。对不可变数据的引用不需要使用互斥。
- 异步 I/O——基于异步事件循环的非阻塞 I/O 并发原语内置于 Rust 语言中，具有零成本 future 和 async-await。非阻塞 I/O 确保代码在等待数据处理时不会挂起。

此外，Rust 的不变性规则提供了高级别的数据并发性。

5. Rust 是一门高效的语言

尽管 Rust 首先是一门面向系统的编程语言，但也添加了高级和函数式编程语言舒适便捷的特性。以下是 Rust 中的一些高级抽象，可带来高效且令人愉快的开发体验：

- 带有匿名函数的闭包
- 迭代器
- 泛型和宏
- 枚举，例如 Option 和 Result
- 通过 trait 实现多态性
- 通过 trait 对象进行动态调度

Rust 不仅支持开发人员构建高效、安全和高性能的软件，还通过其表现力来优化开发人员的生产力。Rust 连续五年(2016—2020 年)成为 Stack Overflow 开发者调查中最受欢迎的编程语言并非浪得虚名。

注意：要更深入地了解高级开发人员为什么喜欢 Rust，可参阅 The Overflow 博客上的“Why the developers who use Rust love it so much”一文，网址为：http://mng.bz/rWZj。

到目前为止，已经介绍了 Rust 如何提供内存安全、资源效率、低延迟、高并发和开发人员生产力的独特组合。这些特性赋予了 Rust 系统编程语言的底层控制能力和高性能、高级语言的开发效率，以及独特的无垃圾收集器的内存模型。应用程序后端和基础设施服务直接受益于这些特性，在高负载下提供低延迟响应，同时高效地使用系统资源，例如多核 CPU 和内存。接下来看一下 Rust 的一些限制。

1.2.3　Rust 的欠缺之处

没有一种语言可以适合所有用例。此外，基于编程语言设计的本质，在一种语言中容易做到的事情在另一种语言中可能十分困难。为了有一个完整的视角来决定是否在 Web 上使用 Rust，需要了解以下几点。

- Rust 的学习曲线很陡峭。对于编程新手或从事动态编程或脚本语言的人来说，这绝对是一个更大的飞跃。即使对于经验丰富的开发人员来说，语法有时也难以阅读。
- 与其他语言相比，有些东西更难用 Rust 编程，例如单链表和双链表。这是由于语言的设计方式造成的。
- Rust 编译器目前比许多其他编译语言的编译器都慢。然而，编译速度在过去几年中已有所提高，并且一直在改进。
- 与其他主流语言相比，Rust 的库及其社区生态系统仍然处于成熟发展中。
- Rust 开发人员更稀缺。
- Rust 在大公司和企业中的采用仍处于早期阶段。Rust 缺乏一个天然的培育环境，例如 Java 的 Oracle、Golang 的 Google 或 C#的 Microsoft。

现在已经介绍了使用 Rust 开发应用程序后端服务的优点和缺点。1.3 节将介绍本书要构建的示例应用程序。

1.3　可视化示例应用程序

接下来的章节将使用 Rust 构建 Web 服务器、Web 服务和 Web 应用程序，并将通过完整的示例演示概念。注意，我们的目标不是开发功能完整或架构完整的分布式应用程序，而是学习如何在 Web 领域使用 Rust。

记住这一点很重要：我们只会探索一些路径——所有可能路径中数量非常有限的路径——而完全忽视其他可能同样有希望和有趣的路径。这是一个有意的选择，用以聚焦讨论重点。例如，仅开发 REST Web 服务，完全排除 SOAP 服务，尽管这看起来是多么武断。

本书也不会讨论现代软件开发的一些重要方面，例如持续集成/持续交付(CI/CD)。这些是今天实践中非常重要的主题，但没有任何特别针对 Rust 需要解释的内容，这已超出了本书的范畴。

另一方面，因为容器化是当今的主要趋势，而且我认为将 Rust 开发的分布式应用程序部署为容器很有趣，所以我将展示使用 Docker 部署和运行我们的示例应用程序是多么容易。

同样，本书的最后几章将简述点对点(P2P)网络领域，这是异步功能最引人注目的用法之一。不过，本书的这一部分与示例应用程序略显脱节，原因是我没有找到将 P2P 与其集成的令人信服的用例。因此，我们的示例应用程序使用 P2P 作为可以探索的练习。

接下来先看示例应用程序。

1.3.1 构建目标

本书将为导师建立一个名为 EzyTutors 的数字店面，导师可以在其中在线发布课程目录。导师可以是个人或培训企业。数字店面将成为导师的销售工具而非市场。

既然已经定义了产品愿景，接下来先谈谈范围，然后是技术栈。

店面允许导师自行注册然后登录。导师可以创建课程并将其与课程类别相关联。程序将为每位导师生成一个包含课程列表的网页，然后他们可以通过网络在社交媒体上分享该网页。还将有一个公共网站，允许学习者搜索课程、按导师浏览课程以及查看课程详细信息。图 1.2 说明了示例应用程序的逻辑设计。

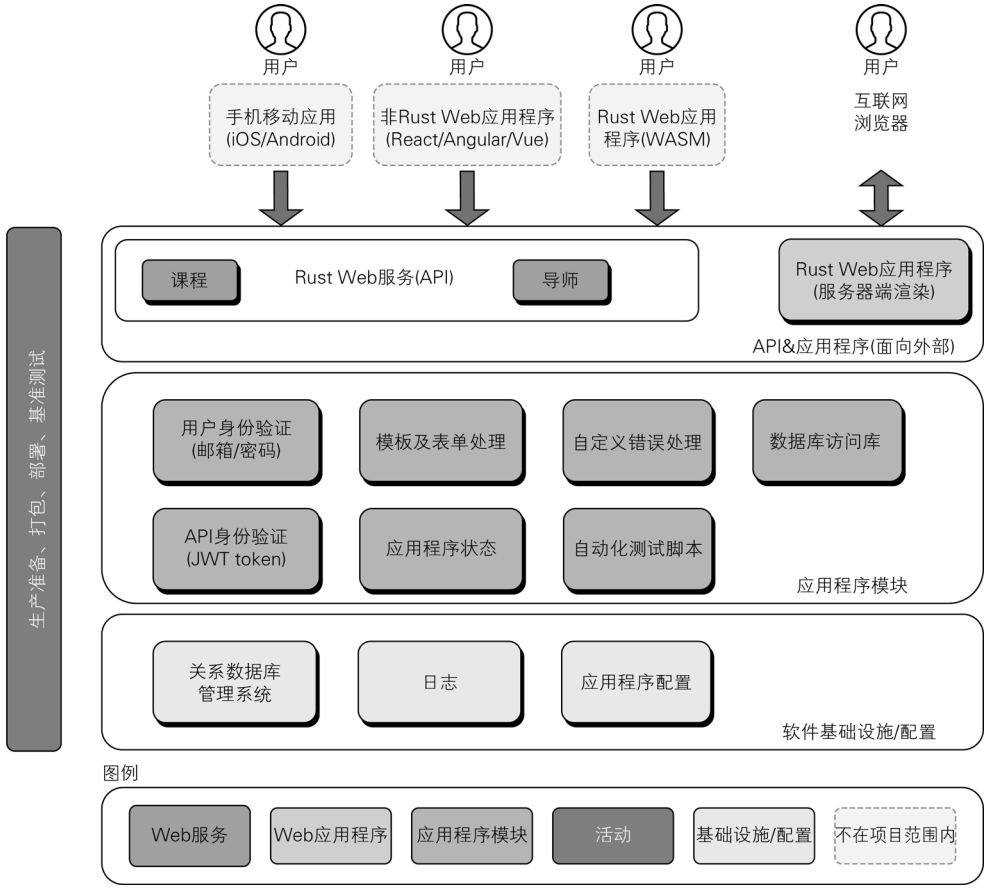

图 1.2 示例：EzyTutors 应用程序

技术栈将由一个 Web 服务和一个纯用 Rust 编写的服务器端渲染的 Web 应用程序组成。有几种非常流行的方法，例如使用成熟的 Web 框架(如 React、Vue 或 Angular)开发 GUI，但为了专注于 Rust，本书不会使用这些方法。关于这个主题还有很多其他优秀的书籍可供参考。

课程数据将保存在关系数据库中。我们将使用 Actix Web 作为 Web 框架，使用 SQLx 作为数据库连接，使用 Postgres 作为数据库。重要的是，设计始终是异步的。Actix Web 和 SQLx 都支持完全异步 I/O，这非常适合我们的 Web 应用程序工作负载，它的 I/O 负载大于计算负载。

首先，构建一个 Web 服务，该服务公开 RESTful API、连接到数据库并以特定于应用程序的方式处理错误和故障。然后，将通过增强数据模型并添加额外的功能来模拟应用程序生命周期的变化。这将需要重构代码和数据库迁移。这个练习将展示 Rust 的关键优势之一——借助强类型系统和严格但有用的编译器来无畏地重构代码(并减少技术债)。

除了 Web 服务，示例还将演示如何使用 Rust 构建前端；本书选择的示例将是服务器渲染的客户端应用程序。并且将使用模板引擎为服务器渲染的 Web 应用程序渲染模板和表单。也可以实现一个基于 WebAssembly 的浏览器内应用程序，但这样的任务超出了本书的范围。

本书的 Web 应用程序可以在 Rust 支持的任何平台上开发和部署：Linux、Windows 或 macOS。这意味着我们不会使用任何将应用程序限制到任何特定计算平台的外部库。应用程序将能够部署在传统的服务器中或任何云平台中，无论是作为传统的二进制文件还是容器化环境(例如 Docker 或 Kubernetes)。

示例应用程序选择的问题域是一个实际场景，但并不难理解。这将使我们能够专注于本书的核心主题——如何将 Rust 应用到 Web 领域。作为额外的收获，我们还将通过实践来加深对 Rust 的理解，例如特性(traits)、生命周期(lifetimes)、Result 和 Option、结构体(structs)和枚举(enums)、集合(collections)、智能指针(smart pointers)、可推导特性(derivable traits)、关联函数和方法(associated functions and methods)、模块和工作区(modules and workspaces)、单元测试(unit testing)、闭包(closures)，以及函数式编程(functional programming)等概念。

本书介绍 Rust Web 开发的基础知识，不会介绍如何配置和部署其他基础设施组件和工具，例如反向代理服务器、负载均衡器、防火墙、TLS/SSL、监控服务器、缓存服务器、DevOps 工具、CDN 等，因为这些不是 Rust 专用的主题(尽管大规模生产部署需要它们)。

除了在 Rust 中构建业务功能，示例应用程序还将展示良好的开发实践，例如自动化测试、便于维护的代码结构、将配置与代码分离、生成文档。当然，还有编写惯用的 Rust。

你准备好学习一些实用的 Web 领域的 Rust 知识了吗？

1.3.2　示例应用程序的技术准则

这不是一本关于系统架构或软件工程理论的书。不过，我想列举一些我在书中采用的基本准则，这些准则将帮助你更好地理解我在代码示例中选择设计的基本原理。

(1) 项目结构——本书将大量使用 Rust 模块系统来分离各个功能部分并保持组织有序。将使用 Cargo 工作区将相关项目分组在一起，其中可以包括二进制文件和库。

(2) 单一职责原则——每个逻辑上独立的应用程序功能都应该位于自己的模块中。例如，Web 层中的处理程序应该只处理 HTTP 消息。业务和数据库访问逻辑应该位于不同的模块中。

(3) 可维护性——以下准则与代码的可维护性相关。

- 变量和函数名称必须直观易懂。
- 使用 Rustfmt 可保持代码格式的统一。
- 随着代码的迭代发展，我们将编写自动化测试用例，以在代码迭代开发过程中检测和防止回归问题。
- 项目结构和文件名必须直观易懂。

(4) 安全性——本书将介绍使用 JSON Web Tokens(JWT)的 API 身份验证和基于密码的用户身份验证。基础设施和网络级安全不包括在内。然而，重要的是要记住，Rust 本质上提供了内存安全性，无需垃圾收集器，并且线程安全性可以避免竞争条件，从而避免几类难以查找和难以修复的内存、并发和安全错误。

(5) 应用配置——配置与应用分离是示例项目采用的原则。

(6) 外部 crate 的使用——将尽量减少外部库的使用。例如，本书中的自定义错误处理功能是从零开始构建的，而不是使用简化和自动化错误处理的外部库。这是因为使用外部库走捷径有时会阻碍学习和深入理解的过程。

(7) 异步 I/O——特意选择在示例应用程序中使用支持完全异步 I/O 的库，用于网络通信和数据库访问。

现在已经涵盖本书将讨论的主题、示例项目的目标以及将用来指导设计选择的准则，第 2 章开始深入研究 Web 服务器和 Web 服务。

1.4 本章小结

- 现代 Web 应用程序是数字生活和企业不可或缺的组成部分，但其构建、部署和操作都很复杂。
- 分布式 Web 应用程序包括应用程序前端、后端服务和分布式软件基础设施。
- 应用程序后端和软件基础设施由松散耦合、协作的面向网络的服务组成。它们需要满足特定的运行时特性，并会影响构建工具和技术的选择。
- Rust 因其安全性、并发性、低延迟和低硬件资源占用而成为一种非常适合开发分布式 Web 应用程序的语言。
- 本书适合考虑使用 Rust 进行分布式 Web 应用程序开发的读者。
- 本章研究了将在本书中构建的示例应用程序，并回顾了代码示例所采用的关键技术指南。