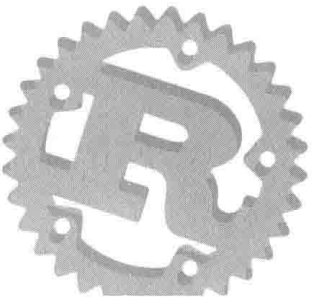
Broadview®
www.broadview.com.cn

Rust官方力荐!

Rust编程之道

张汉东◎著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn



Rust编程之道

张汉东◎著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

Rust 是一门利用现代化的类型系统，有机地融合了内存管理、所有权语义和混合编程范式的编程语言。它不仅能科学地保证程序的正确性，还能保证内存安全和线程安全。同时，还有能与 C/C++语言媲美的性能，以及能和动态语言媲美的开发效率。

本书并非对语法内容进行简单罗列讲解，而是从四个维度深入全面且通透地介绍了 Rust 语言。从设计哲学出发，探索 Rust 语言的内在一致性；从源码分析入手，探索 Rust 地道的编程风格；从工程角度着手，探索 Rust 对健壮性的支持；从底层原理开始，探索 Rust 内存安全的本质。

本书涵盖了 Rust 2018 的特性，适合有一定编程经验且想要学习 Rust 的初学者，以及对 Rust 有一定的了解，想要继续深入学习的进阶者。

图书在版编目（CIP）数据

Rust 编程之道 / 张汉东著. —北京：电子工业出版社，2019.1
ISBN 978-7-121-35485-4

Ⅰ. ①R… Ⅱ. ①张… Ⅲ. ①程序语言－程序设计 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字（2018）第 251334 号

策划编辑：刘恩惠
责任编辑：张春雨
印　　刷：三河市良远印务有限公司
装　　订：三河市良远印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编：100036
开　　本：787×1092　1/16　印张：36.25　字数：1018 千字
版　　次：2019 年 1 月第 1 版
印　　次：2019 年 1 月第 1 次印刷
定　　价：128.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）51260888-819，faq@phei.com.cn。

推荐序一

Even though I had to read this book through Google Translate, *The Tao of Rust* is an extremely interesting book. It starts off explaining exactly why it is different: it's a book that gets you to think about Rust, and its perspective on the world. I only wish I could read it in its native tounge, as I'm sure it's even better then! I have been working on Rust for six years now, and this book changed my perspective on some aspects of the language. That's very powerful!

即便我不得不通过谷歌翻译阅读这本书，但也不难发现《Rust 编程之道》是一本非常有趣的书。它解释了 Rust 为何与众不同：这本书可以让你思考 Rust，以及 Rust 语言所蕴含的世界观。我好希望能读懂中文原版书，因为我相信它会更精彩！ 我已经从事 Rust 的相关工作六年了，这本书改变了我对 Rust 语言的某些看法。这非常强大！

——Steve Klabnik，Rust 官方核心团队成员及文档团队负责人

推荐序二

I knew Rust was a notoriously difficult programming language to learn, but it wasn't until I read the preface to *The Tao of Rust*, by Alex Zhang, that I realized why it is so difficult. Alex writes:

> Rust covers a wide range of knowledge, including object-oriented, functional programming, generics, underlying memory management, type systems, design patterns, and more.

Alex covers all of these topics and more in *The Tao of Rust*. A single text that ties all of this together will be invaluable for Rust learners. So far I've read a couple of chapters translated from the original Chinese, and I can't wait to read more.

Rust 语言难学，这已经是众所周知的了。但是直到我看到 Alex（张汉东）的《Rust 编程之道》的前言时，我才明白它为什么如此难学，Alex 写道：

> “Rust 涉及的知识范围非常广泛，涵盖了面向对象、函数式、泛型、底层内存管理、类型系统、设计模式等知识。”

《Rust 编程之道》一书涵盖了所有这些主题和内容，并且将这些内容有机地联系在一起，这对于 Rust 的学习者来说是非常宝贵的。我阅读了本书部分内容的英文译稿后，就已经迫不及待地想要阅读更多的内容了。

—— Patrick Shaughnessy，《Ruby 原理剖析》原著作者

推荐序三

三年前，当我们决定为 TiDB 开发自有的分布式 key-value 存储系统 TiKV 时，我们首先要面对的就是选择什么语言的问题。当时，摆在我们面前的有多个选择：Go、C++、Java 和 Rust。在仔细评估之后，我们决定使用 Rust，虽然那时候 Rust 并没有太多成功的项目案例。我清晰地记得当时选择一门语言的条件如下。

- 我们需要一门安全的语言，让我们处理内存和多线程的时候更加游刃有余，不用担心类似垂悬指针、数据争用等问题。
- 我们需要一门高性能的静态语言，以便更好地与内存、CPU 打交道，不用担心 GC 引起的延迟突然上升等问题。
- 我们需要一个强大的包管理系统，以避免陷入编译构建工具的细节中，也不用为管理多个版本的库而发愁。
- 我们需要一个友善的社区，在需要时能从这个社区得到帮助，与大家一起成长。

以上这些条件，Rust 全部满足。事实也没有让我们失望。我们使用 Rust 快速地对 TiKV 进行了迭代。现在，TiKV 不仅大量用在生产环境中，还进入了 CNCF 基金会，成为了一个在 Cloud 上面构建其他服务的原生基础组件。

但是，我们使用 Rust 的历程并不是一帆风顺的。在早期，Rust 相关的文档非常稀缺，网上也没有很好的参考资料，更别提专业系统的 Rust 书籍了。所以，当我拿到汉东同学的《Rust 编程之道》时，我是非常兴奋的。本书不仅介绍了 Rust 的基础知识，还详细地解释了 Rust 里面非常难以理解的所有权系统、内存模型、并发编程等特性。尤其是所有权这个概念，对很多同学来说，所有权就是从其他语言切换到 Rust 的第一个拦路虎，而汉东同学在本书中进行了细致清晰的讲解，相信大家会有一种“哦，原来如此”的感慨。更难能可贵的是，本书还从工程角度讲解了如何使用 Rust 来编写健壮的应用程序，提升产品质量。

Rust 是一门相对难学的语言，我个人认为它的学习曲线比 C++的学习曲线更陡峭，但我相信，通过《Rust 编程之道》，大家能快速掌握 Rust，体验使用 Rust 编程的乐趣，也能更快地在项目中使用 Rust 来保证程序的健壮性。如果你遇到了困难，不用害怕，你可以很方便地从 Rust 社区得到帮助。

欢迎来到 Rust 的世界！！！

——唐刘，PingCAP 首席架构师，TiKV 负责人

序

当我 2015 年开始学习 Rust 的时候，我绝对没有想过要写一本 Rust 编程的书。

缘起

当时我刚刚翻译完《Ruby 原理剖析》一书，开始对底层开发产生了一点点兴趣。从 2006 年入行以来，我就一直和动态语言打交道。虽然自己也想学习底层开发，但能选择的语言几乎只有 C++。我在学校里浅浅地学过 C++这门语言，也许是第一印象作怪，总难以提起对 C++的兴趣。

当 Rust 1.0 发布时，我去官方网站了解了一下 Rust 语言，发现它的主要特点有以下几方面：

- 系统级语言
- 无 GC
- 基于 LLVM
- 内存安全
- 强类型+静态类型
- 混合编程范式
- 零成本抽象
- 线程安全

我一下子就被这些鲜明的特性“击中”了，从此开始了 Rust 的学习。

再一次爱上编程

第一次爱上编程是在上小学时。父亲给我买回来一台金字塔学习机，这台学习机有两种功能，一种是学习 Logo 语言，另一种是玩卡带游戏。编写 Logo 语言就是用小海龟画图，也许是因为太早了，也许是因为没有人引导，那时的我选择了痛快地玩游戏。总想着先玩游戏，再去学怎么编程，然后还幻想着能不能用 Logo 语言编写一个游戏。其实这时候的我对编程更多的是一种憧憬，并没有在学习编程上付出更多的实际行动。

第二次爱上编程是在大学初次学习 C 语言的时候。我本可以选择计算机科学专业，但是最后还是选了电子信息科学与技术专业。这样选是因为我想把软硬件都学了。想法是好的，可惜实施起来并不容易。最后的结果就是，软硬件都没学好。

第三次爱上编程是在遇到 Ruby 语言的时候。当时我在用 Java，并且已经完全陷入了 Java 语言和 Web 框架纷繁复杂的细节中，痛苦不堪。Ruby on Rails 框架的横空出世，把我从这种状态中解救了出来。Ruby 语言的优雅和自由，以及“让程序员更快乐”的口号深深地吸引了

我。这一次我是真正爱上了编程，并且积极付诸行动去学习和提升自己。此时也恰逢互联网创业大潮的开始，Ruby 语言的开发效率让它迅速成为创业公司的宠儿，因此，我也借着 Ruby 这门语言参与到了这股创业洪流中。

第四次爱上编程是在遇到 Rust 的时候。此时，创业洪流已经退潮。技术圈有句话，叫“十年一轮回”。当年喜欢 Ruby 给开发过程带来的快乐，但是随着时代的变革和业务规模的增长，我不禁开始重新思考一个问题：何谓快乐？真正的快乐不仅仅是写代码时的“酸爽”，更应该是代码部署到生产环境之后的“安稳”。Rust 恰恰可以给我带来这种“双重快乐”体验。

为什么是 Rust

社区中有人模仿阿西莫夫的机器人三大定律，总结了程序的三大定律[1]：

- 程序必须正确。
- 程序必须可维护，但不能违反第一条定律。
- 程序必须高效，但不能违反前两条定律。

程序的正确性，一方面可以理解为该程序满足了实际的问题需求，另一方面是指满足了它自身的程序规约。那么如何保证程序的正确性呢？首先，可以通过对程序的各种测试、断言和错误处理机制，来保证其满足实际的问题需求。其次，在数学和计算机科学已经融合的今天，通过较为成熟的类型理论即可保证程序自身的规约正确。

以我最熟悉的 Ruby 语言为例，程序的正确性必须依赖于开发者的水平，并需要大量的测试代码来保证正确性。即便在 100%测试覆盖率的条件下，也经常会遇到 NilError 之类的空指针问题。也就是说，Ruby 程序自身的正确性还没有得到保证。以此类推，C、C++、Python、Java、JavaScript 等语言都有同样的问题。

而函数式编程语言在这方面要好很多，尤其是号称纯函数式的 Haskell 语言，它具有融合了范畴理论的类型系统，利用了范畴理论自身的代数性质和定律保证了程序自身的正确性。然而，Haskell 也有比较明显的缺点，比如它不满足上述第三条定律，运行效率不高。

反观 Rust 语言，对程序的三定律支持得恰到好处。它借鉴了 Haskell 的类型系统，保证了程序的正确性。但还不止于此，在类型系统的基础上，Rust 借鉴了现代 C++的内存管理机制，建立了所有权系统。不仅保证了类型安全，还保证了内存安全。同时，也解决了多线程并发编程中的数据竞争问题，默认线程安全。再来看代码的可维护性，Rust 代码的可读性和抽象能力都是一流的。不仅拥有高的开发效率，还拥有可以和 C/C++媲美的性能。当然，没有银弹，但 Rust 就是我目前想要的语言。

目前 Rust 被陆续应用在区块链、游戏、WebAssembly 技术、机器学习、分布式数据库、网络服务基础设施、Web 框架、操作系统和嵌入式等领域。时代在变化，未来的互联网需要的是安全和性能并重的语言，Rust 必然会在其中大放异彩。

学习 Rust 带来了什么收获

Rust 是一门现代化的语言，融合了多种语言特性，而且 Rust 语言可以应用的领域范围非常广泛。在学习 Rust 的过程中，我发现自己的编程能力在很多方面存在短板。突破这些短板的过程实际上就是一次自我提升的过程。

---

1 https://medium.com/@schemouil/rust-and-the-three-laws-of-informatics-4324062b322b

Rust 是一门成长中的新语言，学习 Rust，跟随 Rust 一起成长，可以体验并参与到一门真正工业化语言的发展进程中，感觉就像在创造历史。虽然我并未给 Rust 语言提交过 PR，但也为 Rust 语言和社区多次提交过 Bug，以及文档和工具的改进意见。

Rust 自身作为一个开源项目，算得上是开源社区中的“明星”项目了。学习 Rust 的过程加深了我对开源社区的认识，也开拓了我的眼界。

为什么要写这本书

在学习 Rust 一年之后，我写下了《如何学习一门新语言》一文，其中记录了我学习 Rust 的心得，这篇文章颇受好评。也正因为这篇文章，电子工业出版社的刘恩惠编辑找到了我，并询问是否可以出一本 Rust 编程的书籍。我当时也正想通过一本书来完整地表达自己的学习心得，再加上中文社区中没有较全面系统的 Rust 书籍，于是，一拍即合。

写书的过程可以形容为痛并快乐着。Rust 语言正值成长期，很多语言特性还在不断地完善。举一个极端的例子，比如写下某段代码示例并成功编译后，过了三天却发现它无法编译通过了。于是，我再一次跟进 Rust 的 RFC、源码、ChangeLog 去看它们的变更情况，然后再重新修订代码示例。这个过程虽然痛苦，但改完之后会发现 Rust 的这个改进确实是有必要的。在这个过程中，我看到了 Rust 的成长，以及 Rust 团队为保证语言一致性和开发者的开发体验所付出的努力，让我感觉自己花再多时间和精力去修改本书的内容都是值得的。

话说回来，任何人做事都是有动机或目的的，我也不例外。我写这本书的目的主要有以下三个。

- 为 Rust 中文社区带来一本真正可以全面系统地学习 Rust 的书。
- 以教为学。在写作的过程中，让自己所学的知识进一步内化。
- 传播一种自学方法。本书内容以 Rust 语言的设计哲学为出发点，按照从整体到细节的思路逐个阐述每个语言特性，希望读者可以产生共鸣。

结语

我自己作为本书的第一位读者，目前对这本书是非常满意的。衷心希望每一位读者都能从本书中收获新知。当然，我也知道不可能让每一位读者都满意。在我看来，写书不仅是在传播知识和思想，更是一种交流和沟通。所以，当你不满意的时候，可以来找我交流，提出更多建设性意见，帮助我成长。我争取在写下一本书的时候，让更多的人满意。而且，如果你的建议确实中肯，让我得到了成长，我也为你准备了不错的小礼物。

前言

在我刚开始学习 Rust 的时候，在社区里听到最多的声音就是“Rust 学习曲线陡”。已经有一定编程经验的人在学习一门新语言时，都喜欢直接上手写代码，因为这样可以快速体验这门语言的特色。对于大多数语言来说，这样确实可以达到一定的学习目的。但是当他们在初次学习 Rust 的时候，就很难通过直接上手来体验这种快感。

我第一次学习 Rust 时就遇到了这样的情况。我按以往的编程经验直接写下了代码，但是编译无法通过；可是有时候简单调换两行代码的顺序，程序就能顺利编译成功了，这让我非常困惑。我想这也是大多数人感觉“Rust 学习曲线陡”的原因吧。经过和 Rust 编译器的多次“斗争”之后，我不得不重新反思自己的学习方法。看样子，Rust 编译器暗含了某种规则，只要程序员违反了这些规则，它就会检查出来并阻止你。这就意味着，作为程序员，你必须主动理解并遵守这些规则，编译器才能和你“化敌为友”。

所以，我就开始了对 Rust 的第二轮学习，忘掉自己以往的所学，抱着初学者的心态，从零开始系统地学习 Rust。然而，事情并没有这么简单。

Rust 官方虽然提供了 *Rust Book*，但是内容的组织非常不友好，基本就是对知识点的罗列，系统性比较差。后来官方也意识到了这个问题，推出了第 2 版的 *Rust Book*，内容组织方面改善了很多，对学习者也非常友好，但系统性还是差了点。后来又看了国内 Rust 社区组织群友们合著的 *Rust Primer*，以及国外的 *Programming Rust*，我才对 Rust 建立了基本的认知体系。

直到此时，我才意识到一个重要的问题：Rust 学习曲线陡的根本原因在于 Rust 语言融合了多种语言特性和多种编程范式。这就意味着，Rust 涉及的知识范围非常广泛，涵盖了面向对象、函数式、泛型、底层内存管理、类型系统、设计模式等知识。从底层到上层抽象，从模式到工程化健壮性，无所不包。可以说，Rust 是编程语言发展至今的集大成者。对于大多数 Rust 语言的初学者来说，他掌握的知识体系范围是小于 Rust 所包含的知识量的，所以在学习 Rust 的过程中会遇到无法理解的内容。

我在学习 Rust 之前，所掌握的编程语言知识体系大多是和拥有 GC 的动态语言相关的，对于底层内存管理知之甚少。所以在我学习 Rust 所有权的时候，就很难理解这种机制对于内存安全的意义所在；而我所认识的一些拥有 C 语言编程经验的朋友，在学习 Rust 时面临的问题是，难以理解 Rust 支持的上层抽象，对他们来说，Rust 中融合的类型系统和编程范式就是他们学习道路上的“拦路虎”；对于拥有 Haskell 等函数式编程经验的朋友，会感觉 Rust 的类型系统很容易理解，但是底层的内存管理和所有权机制又成了需要克服的学习障碍；来自 C++编程圈的朋友，尤其是懂现代 C++的朋友，对 Rust 所有权机制理解起来几乎没有困难，但是类型系统和函数式编程范式可能会阻碍他们的学习。当然，如果正好你没有上述情况，那说明你的相关知识体系已经很全面了，你在 Rust 的学习之路上将会非常顺利。

这是不是意味着，在学习 Rust 之前需要把其他语言都学一遍呢？答案是否定的。

Rust 编程语言虽然融合了很多其他语言的特性和范式，但它不是进行简单的内容堆叠，而是有机地融合了它们。也就是说，Rust 遵循着高度的一致性内核来融合这些特性。我们只需要从 Rust 的设计哲学出发，牢牢地把握它的设计一致性，就可以把它的所有特性都串起来，从而达到掌握它的目的。这正是本书遵循的写作逻辑。

本书特点

从设计哲学出发，探索 Rust 语言的内在一致性。设计哲学是一门优秀编程语言保持语言一致性的关键所在。设计哲学是语言特性和语法要素设计的诱因和准则。理解 Rust 语言的设计哲学，有助于把握 Rust 语言的内核与一致性，把 Rust 看似纷繁复杂的特性都系统地串起来。

从源码分析入手，探索 Rust 地道的编程风格。Rust 是一门自举的语言，也就是说，Rust 语言由 Rust 自身实现。通过阅读 Rust 标准库和一些第三方库的源码，不仅可以深入理解 Rust 提供的数据类型和数据结构，更能体验和学习地道的 Rust 编程风格。

从工程角度着手，探索 Rust 对健壮性的支持。Rust 通过类型系统、断言、错误处理等机制保证内存安全的同时，还保证了系统的健壮性。从工程角度去看 Rust，才能看到 Rust 对系统健壮性的支持是多么优雅。

从底层原理开始，探索 Rust 内存安全的本质。只有深入底层，才能理解 Rust 所有权机制对于内存安全的意义。而且可以进一步理解 Rust 的类型系统，以及 Unsafe Rust 存在的必要性。

读者群体

适合本书的读者群体包括：

- 有一定编程经验，想要学习 Rust 的初学者。
- 对 Rust 有一定了解，还想对 Rust 深入学习的进阶者。

本书不适合完全没有编程基础的人学习。

如何阅读本书

对于 Rust 初学者，建议按照章节顺序去阅读。因为本书每一章内容基本都依赖于前一章内容的前置知识。

对于 Rust 有一定了解的朋友，可以选择你感兴趣的章节去阅读。因为本书的每一章也是对一个垂直主题的深入探讨。

一些章节的开头罗列出了通用概念，这是为了更通透地讲解相关知识的来龙去脉。如果你对这部分内容不了解，那么建议你把这部分内容（属于前置知识）认真看完再去看后面的内容。如果你对这部分内容已经有了充分的了解，那么完全可以跳过，直接选择你最关心的内容去阅读。

章节概述

第 1 章 新时代的语言。这一章将从 Rust 语言的发展历史概述开始，引出 Rust 的设计哲学，通过设计哲学进一步阐述 Rust 的语言架构。该语言架构也是本书组织内容时遵循的准则

之一。这一章还将介绍 Rust 语言社区的现状和未来展望。最重要的是，这一章将介绍 Rust 代码的执行流程，这对于理解本书后面的章节会有所帮助。

第 2 章 语言精要。学习任何一门语言时，首先要做的就是了解其语法。这一章将罗列 Rust 语言中的常用语法，但不是简单罗列，而是遵循一定的逻辑进行罗列。在介绍语法之前，这一章会先对 Rust 语言的基本构成做整体概述。然后将介绍一个非常重要的概念：表达式。它是 Rust 语法遵循的最简单的准则之一。接下来才会依次介绍 Rust 中最常用的语法，让读者对 Rust 语言有一个初步的了解。

第 3 章 类型系统。类型系统是现代编程语言的重要支柱。这一章首先将以通用概念的形式介绍类型系统相关的概念，目的是帮助不了解类型系统的读者建立初步认知。接下来将从三方面阐述 Rust 的类型系统。为了理解 Rust 基于栈来管理资源的思想，有必要先了解 Rust 中对类型的分类，比如可确定大小类型、动态大小类型和零大小类型等。这一章还将介绍 Rust 类型推导功能及其不足。接下来将介绍 Rust 中的泛型编程。泛型是 Rust 类型系统中最重要的一个概念。最后会介绍 Rust 的“灵魂”，trait 系统。对类型系统建立一定的认知，有利于学习后面的内容。

第 4 章 内存管理。这一章首先将介绍底层内存管理的通用概念。在此基础上，围绕内存安全这个核心，从变量定义到智能指针，逐渐阐述 Rust 中资源管理的哲学。这部分内容是真正理解 Rust 所有权机制的基础。

第 5 章 所有权系统。这一章首先会介绍关于值和引用语义的通用概念，然后在此基础上探讨 Rust 的所有权机制。读者将看到，Rust 如何结合类型系统和底层内存管理机制，以及上层值和引用的语义形成现在的 Rust 所有权系统。然后，进一步围绕内存安全的核心，阐述借用检查和生命周期参数的意义。通过这一章的学习，读者将会对 Rust 的所有权系统有全面深入的了解。

第 6 章 函数、闭包和迭代器。在对 Rust 的类型系统和内存安全机制有了一定了解之后，我们将开始深入学习 Rust 编程最常用的语法结构。函数是 Rust 中最常用的语法单元。Rust 的函数承载了诸多函数式编程范式的特性，比如高阶函数、参数模式匹配等，同时也承载了面向对象范式的特性，比如为结构体及其实例实现方法，实际上就是一个函数调用的语法糖。然后将介绍闭包的用法和特性，帮助读者对闭包建立全面深入的认知，更重要的是，通过学习闭包的实现原理，进一步了解 Rust 中零成本抽象的哲学思想。最后介绍迭代器模式，以及 Rust 中的迭代器实现机制。迭代器也是 Rust 最常用的特性，通过这一章的学习，你将彻底了解迭代器。

第 7 章 结构化编程。这一章将对 Rust 混合范式编程进行探讨，会重点介绍 Rust 中的结构体和枚举体，以及它们如何在日常编程中以面向对象风格编程。同时，还将介绍三种设计模式，前两种是 Rust 标准库以及第三方库常用的设计模式，最后是一种合理利用 Rust 资源管理哲学的设计模式。通过学习这一章的内容，有利于掌握地道的 Rust 编程风格。

第 8 章 字符串与集合类型。字符串是每门编程语言最基本的数据类型，Rust 自然也不例外。出于内存安全的考虑，Rust 中的字符串被分为了多种，并且语言自身没有自带正则表达式引擎。这一章将从字符编码开始，围绕内存安全，对 Rust 中的字符和字符串做彻底梳理，并且阐述如何在没有正则表达式引擎的情况下，满足字符串进行匹配搜索等的需求。集合类型也是编程中必不可少的数据结构。这一章将着重介绍动态数组 Vector 和 Key-Value 映射集 HashMap 的使用，而且还会深入挖掘 HashMap 底层的实现原理，介绍 Rust 标准库提供的 HashMap 安全性，进一步探讨如何用 Rust 实现一个生产级的数据结构。最后将通过探讨一个

Rust 安全漏洞的成因，来帮助读者正确理解容量的概念，从而写出更安全的代码。

第 9 章 构建健壮的程序。对于如何构建健壮的系统，Rust 给出了非常工程化的解决方案。Rust 将系统中的异常分为了多个层次，分别给出了对应的处理手段。在这一章，读者将学习 Rust 是如何以分层的错误处理解决方案来帮助开发者构建健壮系统的。

第 10 章 模块化编程。现代编程语言的一大特色就是可以方便地进行模块化，这样有利于系统的设计、维护和协作。Rust 在模块化编程方面做得很好。这一章首先将介绍 Rust 强大的包管理系统 Cargo。然后会以真实的代码实例阐述 Rust 的模块系统，并且将包含 Rust 2018 版本中模块系统的重大改进。最后将以一个完整的项目为例阐述如何使用 Rust 开发自己的 crate。

第 11 章 安全并发。Rust 从两方面支持并发编程。首先，利用类型安全和内存安全的基础，解决了多线程并发安全中的痛点：数据竞争。Rust 可以在编译时发现多线程并发代码中的安全问题。其次，Rust 为了达成高性能服务器开发的目标，开始全面拥抱异步开发。这一章将从线程安全的通用概念开始，从 Rust 多线程并发讲到异步并发支持，带领读者逐步形成全面、深入、通透的理解。

第 12 章 元编程。元编程即程序生成程序的能力。Rust 为开发者提供了多种元编程能力。这一章将从反射开始介绍 Rust 中的元编程。虽然 Rust 的反射功能没有动态语言的那么强大，但是 Rust 提供了强大的宏系统。这一章将从 Rust 的编译过程出发，带领读者深入理解 Rust 的宏系统的工作机制，并且以具体的实例帮助读者理解编写宏的技巧。从声明宏到过程宏，再到编译器插件，以及第三方库 syn 和 quote 最新版的配合使用，都将在本章进行阐述。

第 13 章 超越安全的边界。前面的章节内容基本都是建立在 Safe Rust 的基础上的。而这一章将主要围绕 Unsafe Rust 的内容来构建，主要分为 4 大部分。首先将介绍 Unsafe Rust 的基本语法和特性。然后，围绕基于 Unsafe 进行安全抽象的核心，阐述 Unsafe Rust 开发过程中可能引起未定义行为的地方，以及相应的解决方案。然后介绍 FFI，通过具体的实例来阐述 Rust 如何和其他语言交互，涉及 C、C++、Ruby、Python、Node.js 等语言，还将介绍相关的第三方库。最后，将介绍未来互联网的核心技术 WebAssembly，以及 Rust 如何开发 WebAssembly 和相关的工具链。

相信通过这 13 章的内容，读者将会对 Rust 有全面、深入和系统的认识。

勘误及更多资源

有人的地方就有 Bug，此书当然也不例外。写书不仅是正确地传播知识和思想的途径，更是一种交流和沟通的方式。如果你发现本书中的任何错误、遗漏和解释不清楚的地方，欢迎提出反馈。

随书源码地址：https://github.com/ZhangHanDong/tao-of-rust-codes

勘误说明：

- 直接提交 issues。
- 标明具体的页码、行数和错误信息。
- 积极提出勘误者将获得合金 Rust 勋章一枚。

更多的学习资源：

- 官方 doc.rust-lang.org 列出了很多学习文档和资源。
- 订阅 Rust 每日新闻[1]，了解 Rust 社区生态发展，学习 Rust。

致谢

首先，我要感谢 Rust 社区中每一位帮助过我的朋友，没有你们的奉献，就没有这本书。

感谢 Mike 组织社区编写的免费书籍 *Rust Primer*。感谢 Rust 社区中不知名的翻译者翻译官方的 *Rust Book*。感谢知乎《Rust 编程》专栏作者辛苦的写作。感谢 KiChjang、ELTON、CrLF0710、F001、Lingo、tennix、iovxw、wayslog、Xidorn、42、黑腹喵等其他社区里的朋友们，你们在我学习的过程中给予了我无私的帮助和解答，Rust 社区有你们真好。感谢知道我写作并一直鼓励和支持我的朋友们。衷心希望 Rust 社区可以一直这么强大、温柔和友好。

然后，我要感谢电子工业出版社的刘恩惠编辑。感谢你给了我这个机会，让这本书从想法成为了现实。

最后，感谢我的妻子宋欣欣，因为她的温柔、大度、包容、信任和支持，才让我能够踏实且满含信心地做我自己想做的事。感谢我的父母，正是他们的培养，才使我具有积极、坚持不懈做事情的品格。

读者服务

轻松注册成为博文视点社区用户（www.broadview.com.cn），扫码直达本书页面。

- **下载资源**：本书如提供示例代码及资源文件，均可在 下载资源 处下载。
- **提交勘误**：您对书中内容的修改意见可在 提交勘误 处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **交流互动**：在页面下方 读者评论 处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口：http://www.broadview.com.cn/35485

1 https://github.com/RustStudy/rust_daily_news

目录