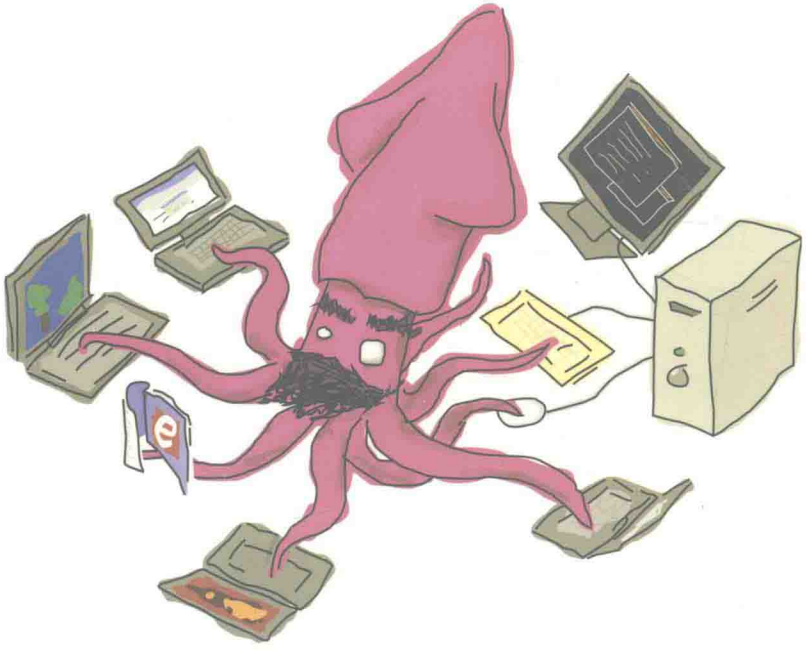
no starch press

·新·锐·编·程·语·言·集·萃·

Erlang趣学指南

Learn You Some Erlang for Great Good!

[加] Fred Hébert 著　　邓辉 孙鸣 译



中国工信出版集团　人民邮电出版社 POSTS & TELECOM PRESS

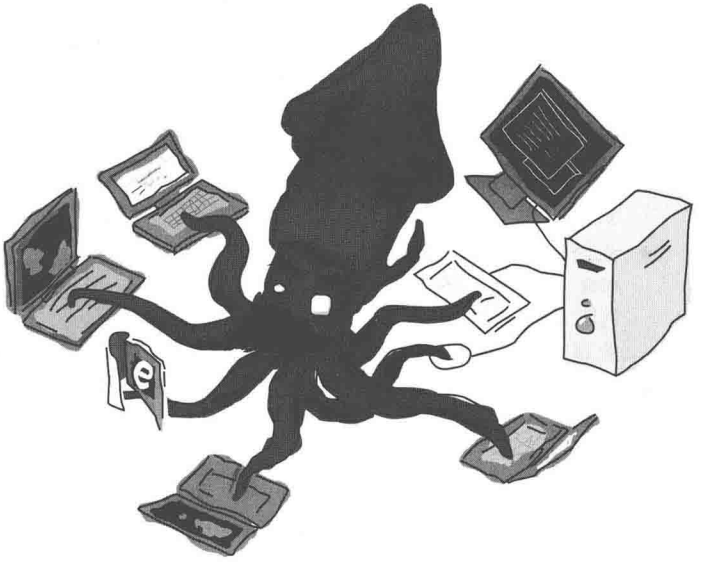
no starch press

·新·锐·编·程·语·言·集·萃·

Erlang趣学指南

Learn You Some Erlang for Great Good!

[加] Fred Hébert 著　　邓辉 孙鸣 译



人民邮电出版社
北京

图书在版编目（CIP）数据

Erlang趣学指南 /（加）弗莱德·赫伯特著 ；邓辉，孙鸣译. -- 北京 ：人民邮电出版社，2016.9
（新锐编程语言集萃）
书名原文：Learn You Some Erlang for Great Good!
ISBN 978-7-115-43190-5

Ⅰ. ①E… Ⅱ. ①弗… ②邓… ③孙… Ⅲ. ①程序语言－程序设计－指南 Ⅳ. ①TP312-62

中国版本图书馆CIP数据核字(2016)第200850号

内容提要

这是一本讲解 Erlang 编程语言的入门指南，内容通俗易懂，插图生动幽默，示例短小清晰，结构安排合理。书中从 Erlang 的基础知识讲起，融汇所有的基本概念和语法。内容涉及模块、函数、类型、递归、错误和异常、常用数据结构、并行编程、多处理、OTP、事件处理，以及所有 Erlang 的重要特性和强大功能。

本书适合对 Erlang 编程语言感兴趣的开发人员阅读。

◆ 著　[加] Fred Hébert
译　邓 辉 孙 鸣
责任编辑　杨海玲
责任印制　焦志炜

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路 11 号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：31
字数：693 千字　2016 年 9 月第 1 版
印数：1－3 000 册　2016 年 9 月北京第 1 次印刷

著作权合同登记号　图字：01-2012-7094 号

定价：79.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315

版权声明

Copyright © 2013 by Fred Hébert. Title of English-language original: *Learn You Some Erlang for Great Good!: A Beginner's Guide*, ISBN 978-1-59327-435-1, published by No Starch Press. Simplified Chinese-language edition copyright © 2016 by Posts and Telecom Press. All rights reserved.

本书中文简体字版由美国 No Starch 出版社授权人民邮电出版社出版。未经出版者书面许可，对本书任何部分不得以任何方式复制或抄袭。

版权所有，侵权必究。

对本书的赞誉

作者 Fred Hébert 作为活跃在一线的有丰富实战经验的工程师，不仅把入门教程写得清晰易懂、深入浅出，更难能可贵的是从 Erlang 应用程序的完整生命周期角度把涉及设计、开发、测试、部署、调优的关键特性表现得淋漓尽致。跟着他的节奏，读者会很容易登堂入室，此外，书中配的插图幽默、诙谐、有爱，也为本书增色不少！

——余锋，阿里云研究员负责阿里云数据库

译者序

在我近 20 年的软件开发工作中，除了 Erlang，还使用过许多其他编程语言。有工作需要的 C/C++、Java，也有作为业余爱好使用的 Lisp、Haskell、Scala 等，其中我最喜欢的当属 Erlang。除了因为我的电信软件开发背景外，还有一个很重要的原因是 Erlang 独特的设计哲学和解决问题方式。

大家听说 Erlang，往往是因为其对高并发的良好支持。其实，Erlang 的核心特征是容错，从某种程度上讲，并发只是容错这个约束下的一个副产品。容错是 Erlang 语言的 DNA，也是和其他所有编程语言的本质区别所在。

我们知道，软件开发中最重要的一部分内容就是对错误的处理。所有其他的编程语言都把重点放在“防”上，通过强大的静态类型、静态分析工具以及大量的测试，希望在软件部署到生产环境前发现所有的错误。而 Erlang 的重点则在于“容”，允许软件在运行时发生错误，但是它提供了用于处理错误的机制和工具。如果把软件系统类比为人体，那么其他编程语言只关注于环境卫生，防止生病；而 Erlang 则提供了免疫系统，允许病毒入侵，通过和病毒的对抗，增强免疫系统，提高生存能力。

这个差别给软件开发带来的影响是根本性的。大家知道，对于大型系统的开发、维护来说，最怕的就是无法控制改动的影响。我们希望每次改动最好只影响一个地方，我们通过良好的模块化设计和抽象来做到这一点。但是如果这个更改不幸逃过了静态检查和测试，在运行时出了问题，那么即使这个改动在静态层面确实是局部的，也照样会造成整个系统的崩溃。而在 Erlang 中，不仅能做到静态层面的变化隔离，还可以做到运行时的错误隔离[①]，让运行时的错误局部化，从而大大降低软件发布、部署的风险。另外，分布式系统中错误出现的必然性更加凸显了 Erlang 容错哲学的价值。

Erlang 语言不仅内置了容错支持工具——完全隔离的进程、链接（link）和监控器（monitor），还提供了一套完整的系统级容错语义模型——监督树。基于这个语义模型，可以清晰、准确、声明性地表达出系统中数据状态的关键程度、系统部件之间的错误相关性和依赖关系、系统各部分之间的承诺保证等，整个系统的容错处理都固化到这个显式、一致、标准的程序结构中。用户只需编写处理正常情况的代码，这个程序结构会自动处理出现的错误。

基于这种独特的容错哲学和支持工具，用户就会以拥抱崩溃、拥抱失败、拥抱异常的态度构建自己的系统，这些原本令人恐惧的东西现在以一种受控的方式存在于系统中，它们不再是让人

① 借助操作系统的进程也可以做到运行时的错误隔离，不过粒度太大，也过于重量。

讨厌的破坏者，而是被转化为了一种简单、强大的工具，用来构建更大、更可靠的系统。

强大的并发支持也是 Erlang 的特色之一，在这一点上常常被其他语言争相模仿。不过，Erlang 和模仿者之间有个根本的不同点——公平调度。为了做到公平调度，Erlang 可谓“不择手段”，并做到“令人发指”的地步[①]。为什么要费劲做这些工作呢？对于一个高并发系统来讲，软实时、低延时、可响应性往往是渴求的目标，同时也是一项困难的工作。尤其是，在系统过载时，多么希望能具有一致的、可预测的服务降级能力。而公平调度则是达成这些目标的最佳手段，Erlang 也是目前唯一在并发上做到公平调度的语言。

由于 Erlang 在容错和并发的公平调度方面的独特性，可以说，这些年来 Erlang 一直被模仿，但是从未被超越。

从某种意义上讲，Erlang 不只是一门编程语言，更是一个系统平台。它不仅提供了开发阶段需要的支持，还提供了其他语言所没有的运行阶段的强大支持。其实，在静态检查和测试阶段发现的问题往往都是些“不那么有趣”的问题，那些逃逸出来的 bug 才是真正难对付的。特别是对于涉及并发和分布式的 bug，往往难以通过静态检查和测试发现，并且传统的调试手段也无法奏效[②]。而 Erlang 则提供了强大的运行时问题诊断、调试、解决手段。使用 Erlang 的 remote shell、tracing、自省机制以及强大的并发和容错支持，我们可以在系统工作时深入系统内部，进行问题诊断、跟踪和修正，甚至在需要时在线对其进行“高侵入性”的外科手术。一旦用户用这种方法解决过一个困难的问题，就再也离不开它了。如果要在静态类型和这项能力间进行选择，我会毫不犹豫地选择后者[③]。

对于 Erlang 存在的问题[④]，提得最多的有两个：一个是缺乏静态类型支持，另一个是性能问题。Erlang 是动态类型语言，往往会被认为不适合构架大型的系统。我自己也非常喜欢静态类型。一个强大的静态类型系统不但能够大大提升代码的可读性，而且提供了一个在逻辑层面进行思考、设计的强大框架。另外，还可以让编译器、IDE 等获取更深入的代码结构和语义信息，从而提供更高级的静态分析支持。

不过，在构建大型系统方面，我有些不同看法[⑤]。如果说互联网是目前最庞大的系统，相信没有人会反对，那么这个如此庞大的系统能构建起来的原因是什么呢？显然不是因为静态类型，根本原因在于系统的组织和交互方式。互联网中的每个部件都是彼此间隔离的实体，通过定义良好的协议相互通信，一个部件的失效不会导致其他部件出现问题。这种方式和 Erlang 的设计哲学是同构的。每个 Erlang 系统都是一个小型的互联网系统，每个进程对应一台主机，进程间的消息对应协议，一个进程的崩溃不会影响其他进程……Erlang 所推崇的设计哲学是面向崩溃

① 为了能够做到跨 OS 的高效调度，Erlang 放弃了基于时间片，采用了基于 reduction 的方式。几乎在系统的每个地方都会进行 reduction 计数，达到公平调度的目的。

② 例如，一个和竞争有关的 bug，一旦加上断点，竞争可能就不会出现了。

③ 其实可以兼得，Erlang 现在已经支持类型定义、标注和推导。

④ 我们不讨论那些主观性太强的问题，例如，有人觉得 Erlang 语法怪异。

⑤ 这些看法只针对 Erlang。对于其他的动态类型语言，在程序规模变大后，确实有难以理解和维护的问题。在 Erlang 中，由于其 let it crash 哲学，很多动态类型语言的问题可以在很大程度上被避免。著名的 AXD301 就是用 Erlang 开发的，规模达百万行代码，历时 3 年。而其前身 AXE-N 是用具有静态类型支持的 C++语言开发的，开发 7 年后失败了。这个例子充分说明，对于大型、复杂的系统来说，语言的语义模型是成败的关键。

（crash-oriented）以及面向协议（protocol-oriented）是架构大型系统的最佳方式[①]。

当然现在，鱼和熊掌可以兼得，Erlang 已经支持丰富的静态类型定义和标注功能[②]，并且可以通过 Dialyzer 工具进行一定程度的类型推导和静态检查。

再来说说性能问题。在计算密集型领域，Erlang 确实性能不高[③]。因此，如果要编写的是需要大量计算的工具程序，那么 Erlang 是不适合的。不过，如果说涉及计算的部分只是系统中的一个局部模块，而亟需解决的是一些更困难的系统层面的设计问题——并发、分布式、伸缩、容错、短响应时间、在线升级以及调试运维等，那么 Erlang 则是最佳选择。此时，可以用 Erlang 作为工具来解决这些系统层面的难题，局部的计算热点可以用其他语言（如 C 语言）甚至硬件来完成。Erlang 提供了多种和其他语言以及硬件集成的方法，非常方便，可以根据自己的需要（安全性、性能）进行选择。

我前段时间曾经开发过一款 webRTC 实时媒体网关[④]，就是采用了 Erlang + C 的方案。其中涉及媒体处理的部分全部用 C 语言编写，通过 NIF 和 Erlang 交互，系统层面的难题则都交给 Erlang 完成。系统上线几个月，用户量就达到数百万。其间，系统运行稳定，扩容方便，处理性能也不错（尤其是高负载时的服务降级情况令人满意）。不是说使用其他语言无法做到，不过要付出的努力何止 10 倍[⑤]！

前面做了这么多的铺垫，主要是为了激起读者对 Erlang 的兴趣。有了兴趣之后，下面当然就是要选择一本好的介绍 Erlang 知识的书籍进行深入、系统的学习。而读者手上拿着的这本，就是一本广受好评的关于 Erlang 的图书。这本书甚至力压 Erlang 之父 Joe Armstrong 的《Erlang 程序设计》，被公认为是学习 Erlang 的一本佳作。这不是没有原因的。

首先，本书对 Erlang 和 OTP 平台进行了非常全面、详细的介绍，不仅讲解了语言的语法、常见的数据结构、基本的函数式编程和并发编程、套接字编程等内容，还深入讲解了 OTP 中的每个关键组件以及整个系统的发布方法。不仅如此，对于真实系统开发中会用到的关键知识内容，分别独立成章进行介绍，包括 EUnit、ETS、Common Test、Mnesia、类型和 Dialyzer 等，甚至还用了一个专门的章节介绍分布式系统设计的核心困难所在和应对策略。

其次，对于一些关键主题，尤其是那些复杂的主题，作者并没有蜻蜓点水、一带而过，而是深入原理，辅以实例，进行了深入浅出的讲解。作者的讲解风格轻松、幽默，让读者在不知不觉中就理解了原本不那么容易理解的内容。讲解的过程中有大量的编程实例，这些例子都是以循序渐进的方式编写的。和很多其他书籍不同的是，这本书中的示例代码质量很高，有些甚至达到了产品级质量。

最后，也是最重要的一点。虽然本书中传递的知识点很多，但是并非只是讲解这些知识点是什么，而是把它们放到了具体的领域背景和时代背景中，让读者理解问题是什么，做出这些决策

① 目前火热的 micro-service 架构在某种程度上和 Erlang 的哲学类似。在 Erlang 中，微服务只是一个语言特性。
② 无论如何，给程序加上类型标注都是一项好的实践。
③ 这方面的性能大概是 C 语言的 1/7。
④ 主要作用是完成浏览器的 webRTC 媒体流和 IMS 网络媒体流之间的互通，需要大量转码和控制。
⑤ 这个是我自己对比的数字。我曾经用 C++语言开发过类似的系统。

的原因是什么，有什么局限性。有了这些背景内容，读者可以更深刻地理解这些知识，在应用这些知识时，可以做出更准确的判断和权衡。更为难得的是，作者把自己在真实产品开发中积累的真知灼见、遇到的语言“坑”和对策也都呈现在这本书中。从某种程度上讲，本书其实是一本关于并发、容错、分布式系统设计的书，只是碰巧使用了这个领域中的 DSL（也就是 Erlang 语言）进行设计的表达。

无论你是初学者还是 Erlang 老手，尤其是你想在产品系统开发中使用 Erlang 时，我都强烈推荐你阅读本书。你一定不会失望的！

最后，如果发现译文中有任何问题，欢迎来信指正（dhui@263.net）。祝大家阅读愉快！

邓辉

2016 年 8 月于上海

序

学习编程很有趣，至少应该是一件有趣的事情。如果没有趣的话，读者就不会喜欢上它。在我的程序员职业生涯中，我曾经自学了几种不同的编程语言，在这个过程中，有时也会觉得不是那么有趣。在学习一门编程语言的过程中是否觉得有趣，很大程度上取决于对这门语言的介绍方式。

开始学习一门新的编程语言时，表面上看起来，好像就是学习这门新语言本身。但是，深入思考后会发现，所要学习的其实是意义更加深远的东西——一种新的思考问题的方法。而令人兴奋的正是这种新的思考方法，不是语言中那些微小的标点符号细节，也不是这门语言和你最喜欢的编程语言在外观上的不同。

在编程领域，函数式编程一直背负着“难学”的名声（并发编程更是如此），因此，编写一本关于 Erlang 语言的书，并在其中同时介绍函数式编程和并发编程，想想都令人望而却步。毫无疑问，介绍函数式编程不是一件容易的事情，介绍并发编程也很困难。除非具有非常特殊的才能，否则根本无法以轻松、幽默的方式同时介绍这两方面的内容。

Fred Hebert 向我们展示了他的这种特殊才能。他总能把复杂的概念以简单的方式介绍给大家。

在学习 Erlang 时有一个最大的障碍，这个障碍并不在于 Erlang 中的概念本身非常难以理解，而在于这些概念和在其他大多数语言中遇到的概念非常不同。为了学习 Erlang，得先暂时忘掉在其他语言中学到的东西。Erlang 中的变量不能改变。不要进行防御性编程。进程非常、非常的轻量，如果愿意的话，可以创建上千甚至上百万个进程。哦，还有就是 Erlang 的语法比较奇怪。Erlang 和 Java 不同，它没有方法或者类，也没有对象……甚至等号的含义也不是“等于”——它的意思是“匹配这个模式”。

Fred 完全无惧这些问题，他在处理这些内容时采用了一种巧妙的冷幽默方式，并且在教授这些复杂的主题时，他所采用的方法完全让我们感受不到复杂性的存在。

这是到目前为止第 4 本主要的 Erlang 书籍，并作为一部重量级的作品加入 Erlang 丛书之列。但是，这本书不仅仅是关于 Erlang 的。Fred 在这本书中介绍的许多概念同样适用于 Haskell、OCaml 以及 F#语言。

我希望大家能够像我一样喜欢 Fred 的这本书。学习 Erlang 的过程是一个令人愉悦、发人深省的过程，我也希望大家在学习的过程中能体会到这一点。如果在阅读本书的过程中，把书中的程序输入电脑并运行，那么你会学到更多的东西。编写程序要比阅读程序困难得多，第一步要做的就是让你的手指习惯于程序的录入，并尝试着去修正那些不可避免的语法小错误。随着阅读的

欢迎访问：电子书学习和下载网站（https://www.shgis.com）
文档名称：《Erlang趣学指南》（加）FRED HEBERT 著.pdf
请登录 https://shgis.com/post/3407.html 下载完整文档。
手机端请扫码查看: