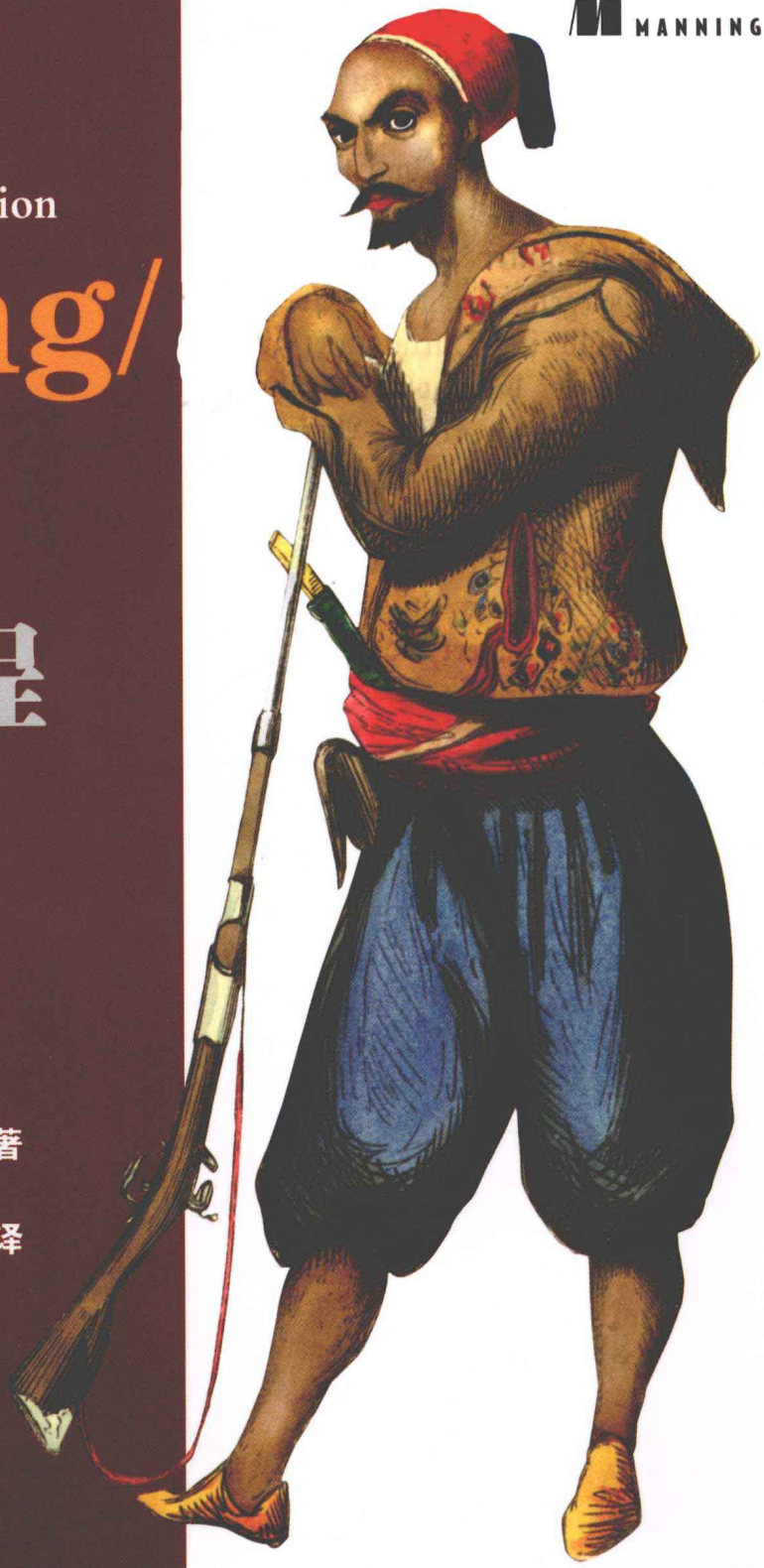
TURING 图灵程序设计丛书

MANNING

Erlang and OTP in Action

# Erlang/OTP 并发编程实战



[美] Martin Logan
[美] Eric Merritt
[瑞典] Richard Carlsson
著

连城 译

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

Erlang and OTP in Action

# Erlang/OTP 并发编程实战

[美] Martin Logan
[美] Eric Merritt
[瑞典] Richard Carlsson
著

连城 译

人民邮电出版社
北京

**图书在版编目（CIP）数据**

Erlang/OTP并发编程实战 / (美) 洛根 (Logan, M.), (美) 梅里特 (Merritt, E.), (瑞典) 卡尔森 (Carlsson, R.) 著 ; 连城译. -- 北京 : 人民邮电出版社, 2012.8
(图灵程序设计丛书)
书名原文: Erlang and OTP in Action
ISBN 978-7-115-28559-1

Ⅰ. ①E… Ⅱ. ①洛… ②梅… ③卡… ④连… Ⅲ. ①程序语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2012)第122366号

## 内 容 提 要

本书侧重生产环境下的 Erlang 开发，主要讲解如何构建稳定、版本控制良好、可维护的产品级代码，凝聚了三位 Erlang 大师多年的实战经验。

本书主要分为三大部分：第一部分讲解 Erlang 编程及 OTP 基础；第二部分讲解如何在实际开发中逐一添加 OTP 高级特性，从而完善应用，作者通过贯穿本书的主项目——加速 Web 访问的分布式缓存应用，深入浅出地阐明了实践中的各种技巧；第三部分讨论如何将代码与其他系统和用户集成，以及如何进行性能调优。

本书面向 Erlang 程序员，以及对 Erlang/OTP 感兴趣的开发人员。

图灵程序设计丛书

**Erlang/OTP并发编程实战**

◆ 著 [美] Martin Logan [美] Eric Merritt
[瑞典] Richard Carlsson
译 连 城
责任编辑 毛倩倩
执行编辑 刘美英

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 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：22.25
字数：526千字 2012年8月第1版
印数：1-4 000册 2012年8月北京第1次印刷

著作权合同登记号 图字：01-2011-0610号

ISBN 978-7-115-28559-1

定价：79.00元

**读者服务热线：(010)51095186转604 印装质量热线：(010)67129223**
**反盗版热线：(010)67171154**

# 版权声明

# 译 者 序

本科毕业后，我的第一份工作是即时通信服务研发。大约在2007年年底的时候，出于工作原因，我对XMPP产生了兴趣。在调研过程中，我找来了各种XMPP服务器进行比较。令我惊讶的是，业内公认最为优秀的分布式XMPP服务器ejabberd，竟然是用一种样貌诡异的冷僻语言写成的——这便是我与Erlang的第一次碰面。相较于日常惯用的C++和Java，Erlang对我来说既陌生又怪异。囫囵吞枣地过了一遍相关资料之后，我惊讶地发现这门语言竟然已有近三十年的历史，而且功能完备、羽翼丰满。然而，晦涩的语法和文档却令我晕头转向，加上初识函数式语言，思维方式一时难以转变，这第一次亲密接触没过多久便宣告结束。

一年多之后，Facebook发布了基于XMPP的即时通信服务Facebook Chat，所用的服务器正是经过定制的ejabberd[①]。刚好那段时间正在琢磨分布式一致性相关的问题，迫切需要一门便于实现分布式算法的语言。于是我开始在官方文档和*Concurrent Programming in Erlang*的指引下学习Erlang。很快我便发现，只要适应了尾递归和单次赋值等函数式编程的特点，Erlang语言本身非常地简单明了。经过初步的摸索，对Erlang的认识也逐渐完整起来。最初以为这是一门阳春白雪的学院派语言，后来却发现大错特错。自打诞生之日起，Erlang就是一门目的性和工程性极强的语言。它的特性集合历经电信行业的千锤百炼，几乎不带一丝一毫的水分。尤为有趣的是，正如本书简介中所述，Erlang的历史与工程型语言的另一典范C惊人地相似。但是，由于思想过于前卫，这门优秀的语言却一直未能受到足够的关注。在服务器端的开发者们都在兴致勃勃地探讨如何通过避免内存复制来提高单机性能时，复制式消息传递简直就是异端邪说！过去十年间，并发处理的复杂性已经随着硬件瓶颈的到来而凸显出来。然而，对于一线应用的开发者而言，近年来大规模互联网应用的爆炸式增长才切实将并发处理变成了一个亟待解决的现实问题。

没想到很快冷水便劈头盖脸地浇了下来——Erlang的并发、容错机制只是整个体系的基石，要想真正发挥出它们的威力，还要仰仗一套叫做OTP的东西。这玩意儿可真是折腾死人了！OTP行为模式迷宫般的回调逻辑把我绕得晕头转向；在解决具体问题时，我总是搞不清楚到底该用`gen_server`、`gen_fsm`还是`gen_event`；好不容易在Erlang shell里跑通一段代码，想打包部署到其他机器上做多机实验时又撞了一头包：应用、发布镜像、变幻莫测的配置项、漫天飞舞的版本号、语焉不详的官方文档……所有这些魔鬼般的细节无不蚕食着我所剩不多的耐心。

译完本书之后再回过头来看，其实OTP的核心概念并不复杂，当年最让我搓火儿的还是实战

① 参见http://blog.process-one.net/facebook_chat_supports_xmpp_with_ejabberd/。

过程中各种剪不断理还乱的繁琐细节。Erlang/OTP的官方文档并不缺细节，但这些细节却水银泻地般地散落在各个角落里，缺乏一条将它们有机地贯穿起来的主线。梳理这一主线，正是本书的任务。全书以交付产品级代码为目标，以一个现实而鲜活的虚拟项目为舞台，深入浅出地讲解了OTP中最为重要的机制和概念，并清晰地呈现了它们之间的内在联系。如果说Erlang/OTP的官方文档是个繁华的大都市，那么这本书就是一本地图。明白了关键机制和概念之间的内在联系之后，按图索骥深入学习Erlang /OTP就不再是什么难事了。

既然是译者序，那么再说说译书的那些事儿吧。2009年至2010年间，我在Erlang China社区内发起了CPiE-CN项目，召集了一批志愿译者共同完成了*Concurrent Programming in Erlang (Part I)*中文版《Erlang并发编程（第一部分）》的翻译。后来，正是这一项目促使我成为了本书的译者。在此我要对CPiE-CN的几位志愿译者表示感谢，他们是王飞、赵宇坤、张驰原、丁豪、赵卫国和吴峻。在本书近一年的翻译过程中，我要感谢图灵教育的傅志红老师、李松峰老师和刘美英老师的帮助；感谢他们容忍了我非常规的交稿方式（以及蜗牛般的进度）。我还要感谢几位协助校对部分中间译稿的早期读者，他们是赵卫国、田中博和倪华杰。最后，特别感谢我的妻子雅莉：这一年中，本职工作的繁重程度远远超出了我的想象，所剩不多的业余时间完全被这本书消耗殆尽，如果没有她的支持和监督，身为重度拖延症患者的我也许根本就坚持不下来。

好啦，闲话少说，预祝各位读者在享用本书的过程中玩儿得开心！

# 序

长久以来，Erlang界仅有一本书流传于世，即1993年出版、1996年修订的“红宝书”[①]。花上100多美元Erlang的拥趸现在还能购买到这本书的印刷版。“红宝书”出版已逾十年，书中的内容早已过时。这门语言几经演变，又新增了一些强有力的编程结构。广泛应用于现代Erlang程序的高阶函数、列表速构（list comprehension）[②]和比特位语法（bit syntaz）等，“红宝书”都未曾收录。不过，发布于1996年的Erlang应用开发框架——开放电信平台（OTP，Open Telecom Platform）——才是全书空缺内容中最关键的一环。Erlang并不难学，OTP却恰恰相反。像本书作者Martin Logan这种自1999年便开始接触Erlang的早期用户，基本上只能靠不断试错硬碰硬地学习OTP。

在过去的几年中，大量Erlang相关图书不断出版，足以证明语言本身的魅力。我们曾获悉还有几本新书即将问世，其中最受瞩目的便是Martin Logan、Eric Merritt和Richard Carlsson合著的这本书。如今它终于面世了。

我是从1993年开始接触Erlang编程的，那时我正在阿拉斯加的安克雷奇设计灾难应急系统。我购买了一套随QIC磁带发布的HP-UX预编译版Erlang。当时的Erlang规模比现在小，支持库的数量也少。我不得不自行设计数据访问结构、数据库管理器、协议解析器以及错误处理框架——不过我却醉心于此。那时候跟现在可没法比：随着同一年Mosaic浏览器的发布Web才刚刚兴起，开源的概念也还得再过五年才会为人所知。在这样的背景下，要想获取一套支持分布式计算和容错的编程框架，就只能狠狠地砸钱砸时间。我淘遍了市面上所有的相关工具，自认已经对各种商业方案了如指掌。当时的Erlang既生涩又不起眼，语法怪异、文档奇缺，但相较于其他工具，其核心理念却显得更为靠谱。

三年后，我已身处瑞典，就职于爱立信并担任史上最大的Erlang项目的首席设计师。我们正打算用Erlang构建传说中的电信级ATM交换系统，以及一套名为开放电信平台的全新框架。之所以采用这个名字，主要是为了迎合公司老大们的胃口——电信是我们的核心业务；开放是时下的流行词儿；主流观点又认为要想构建一套健壮的复杂系统，你就必须拥有一套用于解决冗余、远程配置支持、在线软件升级以及实时追踪调试等问题的平台。

---

① Joe Armstrong、Robert Virding、Claes Wikström和Mike Williams编写的*Concurrent Programming in Erlang*（Prentice Hall, 1993, 1996）。（另一本Erlang经典图书是Joe Armstrong编写的*Programming Erlang*，人称“Erlang圣经”，中文版《Erlang程序设计》已经由人民邮电出版社出版。——译者注

② 速构（comprehension）这一概念源自ZF集合论中的ZF速构，同时也是函数式编程语言中的一种常见语法结构。Erlang中的速构是一种在现有列表/位串上应用若干约束条件后重新构造新的列表/位串的方法。——译者注

开发工具销售并非爱立信的正业，但早在20世纪70年代早期，这儿就开始设计一些用于满足特定需求的编程语言。值得称道的是，爱立信于1998年开源了Erlang/OTP（当然也有出于自身利益的考虑）。于是全世界的爱好者得以投身其中。起初主要用于电信业，后来又逐步渗入到了其他领域。90年代的时候，我们曾试着向Web开发者们大力推荐Erlang，但那时的Web开发者所面临的挑战并非冗余、可伸缩、高响应度的电子商务站点的构建；这类系统的时代尚未来临，并发也尚未入得主流程序员的法眼。那时，并发是众所周知的难点，是人人都唯恐避之而不及的东西。既然如此，人们又何苦选用一种连写“hello world”都得牵扯上并发的语言呢？

随着Web的爆炸式增长和交互性渐强的Web应用的涌现，冷宫中的Erlang终于被解救出来。物理定律也出人意料地向我们伸出援手——通过提高CPU时钟频率来制造更快的单核芯片的技术终于达到了极限。硬件制造商们打出“免费午餐已经结束”的口号，促使开发者放弃对高速单核处理器的依赖，转而探索如何让程序扩展到多个较弱的核上。这对Erlang来说是绝佳的机遇。这意味着程序员中的许多佼佼者至少会开始注意Erlang，并去思考是什么令它如此特别。大部分人只会瞅上一眼，另一些人则会用自己熟悉的语言去模拟Erlang的理念。这是件好事，它意味着知晓并钟爱Erlang及其背后原理的人将更受市场的青睐。

目前OTP已经在电信以外的几个领域得到验证，那些学习并掌握了它的人无不对它赞不绝口。Erlang/OTP是一个非常强大的平台，但需要花时间来学习。在全新项目中应用它时就更是如此。有意思的是，往往那些在OTP项目中摸爬滚打多年的程序员，也不清楚该如何从头构建一个基于OTP的系统。这是因为应用开发者只需接触整个框架的一小部分。这恰恰是我们在大型项目中追求的目标；但小型初创公司的老板不能指望会有人挑灯夜战，逐个儿搞定OTP发布处理的细碎问题和其他犄角旮旯里的各种头疼事儿，所以必须有一套行之有效的示例和教程。

我们正迫切需要一本关于OTP的好书，而本书的出现正填补了这一空白。Martin Logan、Eric Merritt和Richard Carlsson都拥有大量Erlang实践经验，而且都为Erlang社区作出过杰出的贡献，合著本书可谓“强强联合”。我相信此书定会加速推动Erlang实用化的热潮。

尽请赏析！

Ulf Wiger

Erlang Solution 有限责任公司CTO

# 前　言

本书试图提炼出成就一名专业Erlang程序员所需的最关键的知识，借此我们才能让这门高效的语言发挥出其最大的潜力。Erlang/OTP功能强大，但直到目前为止，对初学者来说，通过研读OTP文档来自学OTP框架仍然是件令人望而生畏的事情（这些文档探究了很多细节，却缺乏全局观）。

本书三位作者长期从事Erlang相关工作，但各自的发展轨迹却很不一样。

Martin："我是在自己第一份'真正'的工作中接触到Erlang的。此前我一直从事C和C++开发，也有意思得很。我的第一任老板Hal Snyder甚至在20世纪90年代便对多线程深恶痛绝，后来邂逅了Erlang。我那时还只是个实习生，于是他给了我一个要用Erlang完成的项目。原因嘛，嗯，无非是我的工钱低，即便我搞砸了，公司在这桩买卖上的损失也就不过70美元。最后我并没搞砸，我自己写了一个1000行的监督进程，代码不好看，因为那时候我压根儿不知道OTP是什么东西，手边当然也不会有相关的书。在这个过程中，我爱上了这种'靠谱'的开发后端系统的方法，也爱上了Erlang。Erlang给了我洞悉未来的机会：我所写的复杂分布式系统，所用的高级算法，都是我那些使唤着命令式语言的同事们所梦寐以求的，不花上两年工夫码上一百万行代码他们压根儿实现不出来。读了数千页文档，写了数万行代码之后，我依然钟情于它。一路走来，我遇到了许多杰出的人物，能与其中的两位共同撰写本书令我激动不已。2004年我在ACM会议上发言时遇到了Richard，四年后我又遇到了Eric，并和他一同创建了Erlware———一个仍在蓬勃发展中的项目。多年来，Erlang在我的职业生涯和个人生活中一直扮演着重要的角色，今后也仍会如此。"

Eric："我研究Erlang完全是无心插柳。我曾想写一款大规模多人游戏。然而我明白，仅凭一人之力，即便是才华横溢，也干不完所有图形处理的活儿。于是我决定集中精力主攻游戏逻辑部分，觉得借助于合适的工具和语言我应该能解决这部分问题。在我的设计中，我喜欢在游戏中设立多个代理对象，每个代理对象都能随时间自主学习并独立而并发地行动。那时我能想到的唯一可行的办法，就是把每个代理对象都建模为某种并发的东西，但当时我还不知道这个东西是什么。我所掌握的语言没法让一个开发者单枪匹马拿下这么一款游戏。于是，我开始考察各种语言，前前后后一共花了大概五年时间，作了一些深入的研究。我很早就见识过Erlang，虽然很喜欢它的并发特性，但实在受不了它的语法和函数式特质。直到考察了很多编程语言之后，我才重新开始欣赏Erlang并用它编写代码。那款游戏我一直也没能写成，但我确信选择Erlang没有错，经过深入的研究和剖析，我发现这门语言在许多方面都大有用武之地。这大概是2000年或2001年的事情

了。后续几年间，我又自学了OTP。后来，在2005的时候，我在Amazon.com引入了Erlang，发布了Sinan的第一版，还认识了Martin Logan，并和他一起创办了Erlware。2008年，我搬到了芝加哥，开始写书并尽心打理Erlware项目。”

Richard：“我大概是在1995年前后接触Erlang的，那时我正在乌普萨拉大学为计算机科学硕士论文选题。这引导我后来成为高性能Erlang研究组的一名博士研究生，就Erlang编译器和运行时系统做了若干年的研究。我在瑞典和美国的会议上结识了Martin Logan和Eric Merritt，他们对Erlang的热情令我印象深刻，尽管那时候Erlang还是一门鲜为人知的语言——尤其是在美国。在攻读博士学位期间，我搞了几个业余项目，其中语法工具库和EDoc应用都源自我在编译器方面的研究成果，而EUnit原本是为了检查我的学生们的并发编程作业是否符合规范而设计的。走出学术界之后，我做了几年和Erlang无关的工作，基本上都是在用Python、Ruby和C++写程序。不过最近，我加盟了瑞典最成功的一家创业公司，再次全职投入Erlang，目前正投身于高速发展的高可用性支付系统领域。”

我们三人努力从共同的经验中提炼出尽可能多的内容，以便让你在迈向大师级Erlang程序员的道路上少走弯路；我们也希望能借助本书，最终让OTP框架成为每个Erlang程序员——而不是少数能将手册翻烂的人——都能掌握的东西。

# 致　谢

我们首先要感谢的是这个项目的牵头人Bob Calco，没有你就不会有这本书，我们希望这本书能让你满意。

感谢所有购买了预览版的读者，感谢你们耐心等待本书的完成定稿前我们差不多把这本书重写了三遍，是你们的热情助我们度过了难关。

感谢Jerry Cattell复查Java代码，感谢Francesco Cesarini对我们的帮助和为本书所做的宣传，感谢Ulf Wiger为本书作序，感谢Kevin A. Smith的驱动程序示例代码，感谢Ryan Rawson在Java和HBase方面提供的帮助，感谢Ken Pratt进行技术校对，还要感谢Alain O'Dea及其他所有对预览版给出反馈意见的读者。

这里要特别感谢各位审稿人，他们在写作过程的各阶段对本书手稿提出了宝贵意见，他们是：Chris Chandler、Jim Larson、Bryce Darling、Brian McCallister、Kevin Jackson、Deepak Vohra、Pierre-Antoine Grégoire、David Dossot、Greg Donald、Daniel Bretoi、James Hatheway、John S. Griffin、Franco Lombardo和Stuart Caborn。

我们还要对Manning的工作人员致以衷心的感谢，感谢你们的支持和耐心，特别是Tiffany Taylor、Katie Tennant和Cynthia Kane，你们从不言弃。

最后最为重要的是：Martin要感谢他的妻子Veronica在他马拉松式的写作过程中所展现出的耐心；Richard也要感谢他的妻子Elisabet，即便写作占用了丈夫所有的夜晚和周末，她还是给予了坚定的支持；Eric要感谢Rossana能倾听他的抱怨，还一直提醒他写作本身就是一个不断改进、不断完善的过程。

# 关于本书

本书主要讲解如何开发真实、稳定、版本组织合理且可维护的软件。其中理论讲得不多，更侧重于实战。我们几个人（指本书三位作者）都具有多年的系统开发经验，本书就是以实际的应用软件开发为目标，对这些经验进行提炼的结晶。Erlang编程语言本身并非我们的重点——市面上更适合用作语言教程的书有的是，本书讲述的是生产环境下的Erlang开发实践。

本书的目标是提升独立程序员或公司中的程序员团队的开发效率，因此其内容不仅涵盖了Erlang语言本身，更从头讲解了Erlang/OTP。Erlang自身具备构建强大应用的潜能，但只有借助OTP才能将这种潜能发挥出来。OTP既是一个框架，又是一组库，更是一套构建应用的方法学；它本质上是对语言的扩展。要正经学习Erlang，就一定要学习Erlang/OTP。

本书通过精心挑选的实例阐明了如何在实践中运用Erlang/OTP。通过亲手实现这些实例，你将了解如何构建稳固、版本组织合理的产品级代码，并以此榨干机架上32核机器的每个时钟周期！

## 本书结构

本书分为三部分。第一部分的目的是带你过一遍纯Erlang编程，并介绍一些OTP的基础知识。

- 第1章介绍了Erlang/OTP平台及其主要特性，例如进程、消息传递、链接、分布式以及运行时系统。
- 第2章是Erlang编程语言的一个简单教程，是每个专业Erlang程序员都应了解的内容的参考和总结。
- 第3章将带你开发一个通过TCP套接字通信的Erlang服务器，借此介绍OTP行为模式[①]的概念。
- 第4章介绍了OTP应用和监督树，展示了如何给第3章开发的服务器配上监督进程，并用EDoc生成文档，再一并打包成应用。
- 第5章展示了用于检测Erlang运行时系统的主要GUI工具：应用监视器、进程管理器、调试器和表查看器。

① Erlang/OTP中的“行为模式”（behaviour）相当于一类通用的模式框架，用户通过添加自定义回调便能够方便地实现相应的模式，详情请参见3.1.2节。——译者注

第二部分讨论实际开发相关的问题，我们将面对一些实际的开发任务，并逐渐向代码中添加各种更高级的OTP特性。

- 第6章将带你启动本书的主项目：一套用于提升Web服务器访问速度的缓存系统。本章展示一个更为复杂的多进程应用，还介绍了将监督进程用作进程工厂的方法。
- 第7章阐述了Erlang/OTP的日志和事件处理机制，并介绍了如何通过自定义事件处理程序的方式为缓存系统添加日志功能。
- 第8章介绍了分布式Erlang/OTP，解释了什么是节点，Erlang集群如何工作，节点间如何通信，以及如何借助Erlang shell的任务控制功能在远程节点上执行操作。紧接着我们将运用所学知识来实现一个分布式资源发现应用，用以发布和查询Erlang节点集群中的可用资源信息。
- 第9章将介绍Erlang内置的Mnesia分布式数据库，并展示如何利用分布式表将我们的缓存系统扩展至集群中的多个节点上。
- 第10章讲的是如何打包发布一个或多个Erlang/OTP应用。发布应用时既可以制作独立的最小安装，也可以将其附加于现有安装。另外还会介绍发布包的部署方法。

第三部分讨论如何让代码与大环境融合，如何与其他系统和用户集成，以及如何随复杂度的增加不断优化代码。

- 第11章展示了如何在TCP的基础之上为缓存应用增加REST风格的HTTP接口。通过自定义OTP行为模式，你将从头开发自己的Web服务器。
- 第12章解释了Erlang与其他语言开发的系统进行通信的基本机制。本章展示了三种接入第三方C库的途径：普通端口、端口驱动以及NIF。
- 第13章展示了用Jinterface库集成Java代码的方法，Java程序可以借此模拟成特殊的节点从而接入Erlang集群。随后我们将利用这种方法在缓存应用的后方接入一个用于提供后端存储支持的Hadoop HBase数据库。
- 第14章讲的是Erlang/OTP系统的性能评测和优化，解释了主要代码评测工具的使用并讨论了一些有助于系统调优的实现细节。

本书特意避开了`gen_fsm`行为模式，因为在实践中它的用途非常有限。本书详尽论述了有关OTP行为模式的最关键的内容，掌握这些内容，你便可以充分理解OTP行为模式，进而轻松地通过官方文档自学`gen_fsm`。`gen_fsm`的拿手好戏是二进制协议解析；但普通的`gen_server`配合恰当的模式往往更为适用，而且也更为灵活。对于希望学习`gen_fsm`的读者，我们很抱歉，不过总的来说其余的几个主要OTP行为模式会更为称手。

## 源代码

有别于普通文本，列于清单内或穿插于正文中的源码都使用等宽字体，像这样`fixed-width font like this`。许多代码清单都附有注释，用于强调重要概念。部分源码附有标号，分别对应于代码清单后的解释。

本书中的源码可以从http://github.com/erlware/Erlang-and-OTP-in-Action-Source下载（也可以直接到github.com上搜索书名）。出版社的站点也提供下载：www.manning.com/ErlangandOTPinAction或ituringbook. com. cn/book/828。

## 作者在线

购买本书的读者可以免费访问Manning出版社的专有论坛，在此你可以对本书发表评论，询问技术问题，或是向作者和其他用户寻求帮助。访问和订阅该论坛请移步http://www.manning-sandbox.com/forum.；spa?forumID=454。注册后便可在页面上找到论坛的访问方法、在此能获得哪一类帮助，以及论坛的行为准则。

Manning承诺为公众提供读者和读者之间以及读者和作者之间的交流途径，但无法承诺作者的参与度。作者在本书论坛上发帖都是自愿（且无偿）的。我们建议读者多向作者提一些有挑战的问题，好激发他们参与的兴趣。

作者在线论坛及本书出版至今的讨论合集都可通过出版社的网站访问。

## 关于封面插图

本书封面插图的标题叫“阿尔特温人”（An Artvinian），这是居住在土耳其东北部阿尔特温地区的居民。插图取自一本描绘奥斯曼帝国服饰的画册，这本画册由伦敦老邦德街的William Miller于1802年1月1日出版。画册的扉页已经丢失，因此很难推断准确的创作时间。画册的目录同时使用英语和法语标识插图，另外每张图片还附有两名创作它的艺术家的名字。他们一定想不到……自己的作品竟然会在两百年后被用作某本计算机编程图书的封面。

这本画册是Manning的一位编辑从古董跳蚤市场上淘来的，那地方位于曼哈顿西26号大街的Garage。卖家是个住在土耳其安卡拉的美国人，谈这桩买卖的时候他正要收工回家。当时这位Manning编辑身上没带够钱，信用卡和支票又被婉拒。而卖家当晚就要飞回安卡拉，这么一来似乎就没指望了。那最后怎么办呢？两个人最后通过握手约定的老式君子协议解决了问题。卖家提议通过银行转账付款，于是我们的编辑在纸上记下银行信息后便掖着一包画册离开了。不用说，第二天我们就把钱转了过去。一直以来，我们都深深地感激这位曾对我们的同事抱以信任的陌生人。这真让人无比怀念那充满信任的朴素的旧时代。

奥斯曼画册上的那些画，正如出我们用在其他封面上的那些插图一样，栩栩如生地展现了两个世纪前的丰富多彩的服饰。抛开当今这个过度支配的时代，它们代表着那个年代以及其他历史时期的分隔和距离。衣着习俗从那时开始改变，此前不同地域的服饰的绚丽多姿开始逐渐消失。当今，来自不同大陆的人仅靠衣着已经很难区分开来。也许乐观地看，我们正是用文化上的多样性和视觉上的差异性才换来了多姿多彩的个人生活。亦或是一种更多样、更精彩的知识与技术生活。

我们Manning人把两个世纪前多样的地方生活融入书籍的封面，令画卷复苏，谨以此表达我们对计算机行业的创造性、首创性以及趣味性的赞颂。

# 引　言

Erlang是一门以进程为核心概念的语言。什么是进程？电脑上同时运行着的多个程序，比方说文字处理软件和Web浏览器，彼此便运行在各自的进程之中。文字处理软件若崩溃，通常不会影响浏览器——反之亦然，崩溃的浏览器也不会弄丢你正在编辑的文档。进程就像是某种在并行执行流程间起隔离保护作用的隔膜，Erlang便是完全围绕进程来构建起来的。

在Erlang中创建进程易如反掌——这就像是在Java等语言中创建对象一样简单。进程变得如此廉价，使得我们得以从不同的视角来看待系统。Erlang程序中的任何独立活动都可以被视作单独的进程。没有晦涩的事件循环，也没有线程池，所有这些烦人的实现细节统统都可以抛开。即便程序需要同时运行10 000个进程来完成某个任务，也可以轻松搞定。阅读本书时你会发现，Erlang可以极大地改变我们看待系统的方式。我们希望让你看到，系统可以以更直观（也更高效）的方式得以呈现。

Erlang还是一门函数式编程语言。别害怕，Erlang完全可以设计得更贴近你所熟悉的那些主流语言；即便不借助函数式编程，这里记述的各种特性也一样可以实现。但函数式编程所具有的引用透明性、高阶函数、不可变数据结构等几大特点，它们本身就很值得引入Erlang。函数式编程简洁优雅地融合并呈现了这些特点。要不是引入了函数式编程，Erlang只会变得更为复杂，也不可能那么令人愉悦。

## Erlang 诞生记

你头一回听说Erlang时，它多半是被定性为一门“函数式并发编程语言”，而你大概也会觉得它听起来更像是某种学院派的、不实用的玩具语言。但我们要强调的是，Erlang从一开始就致力于解决真实的大规模软件工程问题。为了把话说清楚，我们得先好好聊聊这门语言背后的历史。

## 与 C 比较

Erlang和C语言的背景颇为相似。首先，二者都源自大型电信公司，都是由某个环境相对轻松的研发部门里的几个人创造出来的。两门语言的创造者都是自在之人，但他们同时也都是试图去解决具体问题的务实的工程师。对于C来说，要解决的问题是如何在硬件资源受限（相对那时而言）的情况下，用比汇编更高级的语言来开发系统软件。对于Erlang来说，问题则在于如何让

程序员开发超大规模、高并发和强容错的软件，并彻底改善生产效率、减少软件缺陷的数量——这可真不是闹着玩儿的。

两种语言的第一批追随者都来自公司内部，这些人在各种内部项目和实际产品中使用它们，并就各种务实的细节向语言的缔造者们提出了宝贵的前期反馈。二者都花了大约十年时间才为公众所知，而在此之前，它们都出色地经受住了实战的检验。C诞生于1972年左右，流行于20世纪80年代。同样，Erlang成形于1988年，直到1998年才以开源的形式对外发布。在公司外部，两种语言先点燃的都是研究机构和大学的兴趣。当然，出于历史原因它们各有各的缺点，但一般来说我们并不在乎，因为也只有它们真正地解决了问题。

让我们把目光转向过去。

## 20 世纪 80 年代中期，斯德哥尔摩：那个自由散漫的英国人

Erlang诞生自一个研究项目，该项目旨在寻求一种更好的编程方式，用以开发当时电信业内那些大流量、高并发、猛龙一般永生不死的控制系统。Joe Armstrong[①]于1985年加入这个项目，来到了位于瑞典斯德哥尔摩的爱立信计算机科学实验室。

这个项目的主要任务就是用尽可能多的编程语言来实现同一类通话控制系统。涉及的语言包括Ada、CLU、Smalltalk等。最终的结果结论性并不强，虽然很明显上上之选是用函数式和逻辑语言并采取高级的声明式风格来进行开发，但当时还没有哪种语言具备合适的并发模型。

但谁知道好的并发模型是个什么样呢？那时（以及之后近二十年间），并发方面的主要研究要么集中在CSP、pi演算这样的纯抽象进程模型和并发逻辑语言上，要么就集中在信号量（semaphore）、监视器（monitor）和信号（signal）这类底层机制上。

与此同时，工程师们仍然要解决各种大规模并发容错通信系统的实际问题。当时的爱立信已经有了独门秘籍，那是一套专有的揉合了编程语言和操作系统的混合解决方案，名为PLEX，AXE电话交换机的成功就要归功于它。

## 让人抓狂的需求

PLEX是一门相对常规的命令式编程语言，但它为后继者立下了一系列标杆：

- 进程必须是语言的核心；
- 任何进程不得损坏其他进程的内存空间，不得遗留悬空指针；
- 由于要同时跑数万乃至数十万个进程，进程创建和任务切换的速度必须要快，单个进程的内存占用量必须非常小；
- 必须能够隔离单个进程的故障；
- 必须能够在运行时对系统进行代码升级；

① 本节标题中的“英国人”指的就是Erlang之父Joe Armstrong。说他“自由散漫”是因为当时他在爱立信所参与的研究项目本身就比较发散，自由度比较大。——译者注

- 必须能够同时检测和处理软硬件错误。

与此最相近的语言当属Parlog和Strand等并发逻辑语言，但它们对进程有着不同的定义，进程的粒度更细，对单个进程的控制力也很弱。

## Erlang 的诞生

一天，Joe发现Prolog基于规则的编程风格可以很好地匹配他之前为描述通话控制问题而发明的手写标记法，于是他开始编写一个Prolog的元解释器[①]。通过这种方式，他扩展了Prolog，模拟出进程切换，用以并发运行多个电话呼叫。

很快，解释器所能识别的表达式形成了一个支持进程和消息传递的小型语言；虽然是用Prolog实现的，但它更简单，而且是函数式的，也没有用上Prolog的合一[②]和回溯特性。没多久，便有人一语双关地提议将之命名为Erlang（一方面指丹麦数学家A. K. Erlang，他因在通信系统统计领域的贡献而为电信工程师们所熟知；一方面它也是Ericsson Language的简写）。

就这样，开发小型但可工作的通话控制系统的需求驱动着早期Erlang的演化。特别是消息传递原语的设计，完全是出于大型电信系统的实际需要，绝非为了迎合某个特定的并发理论。这里所说的消息传递原语，指的是异步的消息发送操作符、自动消息缓存和乱序的选择性消息接收机制（这一设计深受用于规范复杂通信系统协议的CCITT SDL标记法的影响）。

1988年，一群真正的用户进行了初期实验，开发了一个全新的电信架构，事实证明该语言极大地提升了生产效率，但当时的实现实在是太慢了。于是从1990年起，Joe、Mike Williams和Robert Virding开始实现Erlang的第一个抽象机。这个抽象机名叫JAM，是一个用C写成的堆栈机，比起最初的Prolog实现要快上70倍。

1993年，第一本Erlang书籍出版，并于1996年再版。直到这时，Erlang才终于可以算得上是一门真正的语言了。[③]

## 发展壮大

在后来的年月里，Erlang又累积了许多特性，如分布式、记录语法、预处理器、Lambda表达式（fun语句）、列表速构、Mnesia数据库、二进制数据类型以及比特位语法等。整个系统也被移植到了Windows、VxWorks和QNX等非UNIX平台上。

1995年，随着一个巨型C++项目的分崩离析，Erlang在爱立信内部获得了空前的发展。该项目被推倒重来，这回用的正是Erlang以及“不过60个”程序员；另外还有一个靠谱的语言支持部门——OTP团队——来给他们做后盾。最终的成果便是取得了巨大成功的AXD301系统，整个系统由一百多万行Erlang代码锻造而成。

---

① 元解释器（meta-interpreter），这里指用Prolog写的Prolog解释器。——译者注

② 合一（unification），该译法取自Wikipedia（http://zh.wikipedia.org/wiki/合一）。——译者注

③ 这段历史可以参见Joe Armstrong发表于2007年的A History of Erlang，其中记载了大量Erlang的设计理念和趣闻。——译者注