<<Erlang/OTP并发编程实战>>

图书基本信息

书名：<<Erlang/OTP并发编程实战>>

13位ISBN编号：9787115285591

10位ISBN编号：7115285594

出版时间：2012-8

出版时间：人民邮电出版社

作者：[美] Martin Logan,[美] Eric Merritt,[瑞典] Richard Carlsson

页数：333

字数：526000

译者：连　城

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。

<<Erlang/OTP并发编程实战>>

内容概要

《Erlang/OTP并发编程实战》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。

《Erlang/OTP并发编程实战》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。

《Erlang/OTP并发编程实战》面向Erlang程序员，以及对Erlang/OTP感兴趣的开发人员。

作者简介

Martin Logan 从1999年开始活跃于Erlang社区，后来全职从事Erlang研发。
目前任职于全球最大的在线旅游公司Orbitz Worldwide，为基于大规模分布式服务的基础设施开发解决方案。
Erlware联合创始人，核心开发人员，Erlang/OTP软件包管理系统Faxien的主要开发者。

Eric Merritt 专注并发编程和分布式系统。
曾任职于Amazon.com，现为eCD Market软件工程师。
Erlware联合创始人，Erlang芝加哥用户组核心成员。
Erlware团队开源产品核心开发人员，Erlang/OTP构建系统Sinan的主要开发者。

Richard Carlsson 瑞典乌普萨拉大学高性能Erlang计划（HiPE）早期成员，研究Erlang技术达17年，曾为标准库、Erlang编译器、运行时系统和Erlang语言本身都作出过不少贡献。
此外，他还是Erlang文档系统EDoc和单元测试框架EUnit的创建者。
目前加入了Kreditor，致力于高可用性支付系统的Erlang开发。

书籍目录

目　录

章节摘录

版权页： 插图： 1.4.1 调度器 经过多年的演进，ERTS的进程调度器提供了其他平台无法比拟的灵活性。
它最初的设计目标是在单CPU上并发运行轻量级Erlang进程，而不用关心底层用的是什么操作系统。
ERTS运行的时候通常就是单个操作系统进程（在操作系统的进程列表中一般名为beam或werl）。
这个进程中，就跑着管理所有Erlang进程的调度器。
随着线程在大多数操作系统中的普及，ERTS也有所变化，开始将I／O系统这类东西从运行Erlang进程的线程中拿出来，放到独立的线程中去，但完成主体工作的线程仍然只有一个。
如果你用的是多核系统，就必须在同一台机器上运行多个ERTS实例。
不过从2006年5月起，Erlang／OTP第11版中增加了对称多处理器（SMP）支持。
这是一项重大突破，令Erlang运行时系统可以在内部使用不止一个进程调度器，每个占用一个独立的操作系统线程。
这意味着现在Erlang进程可以以n：m的方式映射到操作系统线程。
每个调度器处理一个进程池。
可并行运行的Erlang进程最多能有m个（每个调度器线程执行一个），但同一池内的进程仍像之前所有进程共用一个调度器那样分时运行。
在此基础之上，进程可以在进程池之间迁移以便维 持可用调度器上的负载均衡。
在最新的Erlang／OTP发布版中，甚至可以根据机器上CPU的拓扑情 况将进程绑定到特定的调度器上，从而更好地利用硬件的缓存架构。
这意味着，大多数时候，作为一名Erlang程序员你不用担心手头有多少CPU或有多少个核：你只要中规中矩地写程序，并尽量将程序切分为尺寸适中的并行任务就好，负载均衡之类的事情就让Erlang运行时系统去操心吧。
不管是单核还是128核——都一样，只会更快。
Erlang新手常犯的一个错误就是过分依赖时序，这往往导致那些在他们的笔记本或工作站上运行良好的程序一迁移到多核服务器上就出错，因为在多核服务器上时序的不确定性要大得多。
要发现这类问题，只能借助测试。
好在现在的笔记本基本上都至少是双核的了，这类错误也可以被尽早发现。
Erlang的调度器还涉及运行时系统的另一个重要特性：I／O子系统。
这正是我们的下一个主题。
1.4.2 I／O与调度 很多并发语言都有的一个毛病就是它们没怎么拿I／O当回事儿。
单个进程进行I／O时，它们几乎都存在整个系统或大半系统阻塞的问题。
这真是既恼人又没有必要，尤其是Erlang早在二十年前就已经解决了这个问题。
在前一节，我们曾讨论过Erlang的进程调度器。
除了处理进程调度，调度器还替系统优雅地处理了I／O问题。
在系统的最底层，Erlang以事件驱动的方式处理所有I／O，当数据进出系统时，程序可以以非阻塞方式完成数据处理。
这降低了连接建立和断开的频次，还避免了OS层面上的加锁开销和上下文切换。
这是一种高效的I／O处理方法。
可惜，程序员往往难以分析和理解这种技术，这也是为什么只有在明确要求高可靠性和低延迟的系统中才能见到这种技术。
早在2001年，Dan Kegel就在他的论文The C10K Problem中描述过这个问题，虽然现在已经略显过时，但这篇文章仍然很值得一读。
它针对这个问题及可能的解决方案给出了良好的综述。
这些方案实现起来全都既复杂又痛苦，这正是Erlang运行时系统替你包办这些问题的原因。
Erlang在进程调度器中整合了基于事件的I／O系统。
事实上，你一点儿都不用操心就能享受一切便利。

这让用Erlang／OTP构建高可靠性系统变得轻松了很多。
我们要讲解的最后一个ERTs特性就是内存管理。
它对进程所起的作用超出你的想象。
1.4.3 进程隔离与垃圾回收器 如你所想，Erlang跟Java等大部分现代语言一样，会自动管理内存。
这儿没有显式的释放操作。
相反，垃圾回收器会定期搜寻和回收不再使用的内存。
垃圾回收（GC）算法是一片广袤而复杂的研究领域，我们无法在这儿给出详尽的阐述；不过针对那些对此有一定了解又心怀好奇的读者，可以告诉你Erlang当前使用的是一个简单明了的分代复制式垃圾回收器。

编辑推荐

《Erlang/OTP并发编程实战》讲述了如果将Erlang语言看成才华横溢的钢琴家，那么OTP平台就是一架能让钢琴家把才能发挥得淋漓尽致的钢琴。
《Erlang/OTP并发编程实战》除了全面介绍Erlang语言和OTP平台的基础知识外，还通过一系列实用案例引领你深入了解OTP的高级特性，一步步构建一个大型生产系统。
并加以优化和完善。
三位作者在Erlang领域拥有极其丰富的实战经验，细致入微地剖析了OTP开发与部署的全过程。
要想真刀真枪地上战场。
《Erlang/OTP并发编程实战》才是你明智的选择！

名人推荐

“惊艳！
不管是对于初学者还是Erlang高手，本书绝对都是不容错过的好书。
”——Amazon.com书评“多核处理器和并发编程是将来的重头戏。
ErIang在下一代编程语言中可谓独领风骚！
”——DZone书评“Erlang开发者必备两本书，一本是ErIang之父Joe Arrrlstrong的《Erlang程序设计》，另一本就是本书——务实、高效又不失幽默风趣的好书啊！
”——slashdot.org书评

<<Erlang/OTP并发编程实战>>

版权说明