Broadview
www.broadview.com.cn

MANNING

Functional and Reactive Domain Modeling

函数响应式领域建模

[美] Debasish Ghosh 著
李 源 译

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

Functional and Reactive Domain Modeling

函数响应式领域建模

[美] Debasish Ghosh 著
李 源 译

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

内 容 简 介

传统的分布式应用不会切入微服务、快速数据及传感器网络的响应式世界。为了捕获这些应用的动态联系及依赖，我们需要使用另外一种方式来进行领域建模。由纯函数构成的领域模型是以一种更加自然的方式来反映一个响应式系统内的处理流程，同时它也直接映射到了相应的技术和模式，比如 Akka、CQRS 以及事件溯源。本书讲述了响应式系统中建立领域模型所需要的通用且可重用的技巧——首先介绍了函数式编程和响应式架构的相关概念，然后逐步地在领域建模中引入这些新的方法，同时本书提供了大量的案例，当在项目中应用这些概念时，可作为参考。

版权贸易合同登记号 图字：01-2016-9180

图书在版编目（CIP）数据

函数响应式领域建模 /（美）德巴斯什 • 戈施（Debasish Ghosh）著；李源译. —北京：电子工业出版社，2018.1
书名原文：Functional and Reactive Domain Modeling
ISBN 978-7-121-32392-8

Ⅰ. ①函… Ⅱ. ①德… ②李… Ⅲ. ①函数－程序设计 Ⅳ. ①TP311.1

中国版本图书馆CIP数据核字（2017）第185481号

责任编辑：张春雨
印　　刷：三河市双峰印刷装订有限公司
装　　订：三河市双峰印刷装订有限公司
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：787×980　1/16　印张：18.5　字数：383.6 千字
版　　次：2018 年 1 月第 1 版
印　　次：2018 年 1 月第 1 次印刷
定　　价：79.00 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

开发人员正淹没在各种错综复杂的问题中，需要借助多核处理器以及分布式基础架构的优势，来应对产生数据越来越多的高要求用户规模的迅猛增长，以确保更低的延迟以及更高的吞吐率。所以开发人员不得不在消费者日益苛刻的紧张截止时间前按时交付。

开发人员的工作从来没有轻松过。为了能保持多产的同时又能享受工作，需要采用合适的工具集——这些工具可以通过优化资源的使用来管理日益增长的复杂性以及需求。通常，并不是简单地追逐最新、最炫的东西——尽管这很诱人。所以必须要回顾总结，从过去艰难获胜的经验中学习，看是否可以将其应用到今天的场景以及挑战中。我认为开发人员开发的那些非常有用的工具中所包含的领域驱动设计（domain-driven design，DDD）、函数式编程（FP）以及响应式原则，都可以帮助我们管理复杂事务的某个方面。

- 领域复杂性：领域驱动设计帮助我们挖掘并理解领域的不同特性与语义。通过跟利益相关方用他们的语言进行沟通，DDD 可以更容易地创建可扩展的领域模型来映射真实世界，同时允许持续的变化。
- 解决方案复杂性：函数式编程可以帮助我们保持合理性及可组合性。通过可重用的纯函数并使用稳定（不可变）值，函数式编程提供了一个伟大的工具集，通过不会“撒谎”的代码来得出运行时间、并发性以及抽象过程。

- 系统复杂性：正如在 *The Reactive Manifesto*（http://www.reactivemanifesto.org）中所定义的，响应式原则能帮助我们管理日益复杂的世界，包括多核处理器、云计算、移动设备以及物联网。在这里，所有新系统本质上都是分布式系统。要运作这个世界是非常困难而且很有挑战的，但同样，也拥有很多有趣的新的机会。这种变化迫使我们的行业去反思过去一些围绕系统架构以及设计方面的最佳实践。

我非常喜欢阅读这本书，它完全体现了我在过去十几年的自身经历。我从 OO 实习生开始——白天埋头于 C++ 和 Java，而晚上阅读经典的 *Gang of Four*[1]。2006 年我开始阅读 Eric Evan 关于领域驱动建模的书[2]，它对我或多或少有所启发。然后我就变成一个 DDD 狂热爱好者，在所有可能的地方去应用它。多年后，我又开始使用 Erlang[3]，然后是 Scala[4]，它们都让我再次感受到了函数式编程的魅力并深深地爱上了它。我在大学期间学过函数式编程，但当时并没有真正意识到它的威力。在这段时间里，我开始对 Java 在并发性、适应性以及可伸缩性方面的“最佳实践”逐渐失去信仰。在 Erlang 方式，特别是 actor 模型[5]（我认为这是一个更好的做事方式）的指引下，我开始了 Akka 项目，我相信这会有助于将响应式原则带入主流。

这本书之所以能吸引我是因为它设立了一个更加宏大的目标，将 3 个完全不同的工具（领域驱动设计、函数式编程以及响应式原则）用可实践的方式整合到了一起。它教会你诸如边界上下文、领域事件、函数、monad、applicative、future、actor、流以及 CQRS[6] 等内容是如何使复杂性保持可控的。如果内心不够强大，那么这本书将不适合你，阅读它很费劲。但如果花上数小时，你就会收获一些基础概念。亲爱的读者，幸运的你已经迈出了第一步，接下来所需要做的就是继续读下去。

JONAS BONÉR
Lightbend 创始人兼 CFO
Akka 创始人

---

1 Erich Gamma、Richard Helm、Ralph Johnson与John Vlissides所著的*Design Patterns: Elements of Reusable Object-Oriented Software*（Addison-Wesley, 1994年），也是我们常说的*Gang of Four*。

2 Eric Evans所著的*Domain-Driven Design: Tackling Complexity in the Heart of Software*（Addison-Wesley, 2003年）。

3 一种面向并发的编程语言，由瑞典电信设备制造商爱立信所辖的CS-Lab开发。——译者注

4 一种类似Java的多范式编程语言，设计初衷是实现可伸缩的语言，并集成面向对象编程和函数式编程的各种特性。——译者注

5 https://en.wikipedia.org/wiki/Actor_model。

6 command query responsibility segregation，命令查询责任分离。——译者注

序

在2014年夏天，Manning出版社希望出版*DSLs in Action*（https://www.manning.com/books/dsls-in-action）的升级版本，因为DSL的所有新特性都围绕编程语言的设计和实现。巧合的是正好在那个时间，我用函数模式对一个复杂的领域模型成功地进行了重构。

跟一群刚毕业进入Scala函数式编程世界的软件工程师们一起，我将域行为建模为纯粹的函数，将域对象设计为代数数据类型，并开始意识到代数API设计的价值。团队的每个成员人手一本Paul Chiusano和Rúnar Bjarnason刚完成的*Functional Programming in Scala*（中文版为《Scalo函数式编程》，由电子出版社出版）。

我们的域模型非常复杂，实现严格遵守Eric Evans在他的著作*Domain-Driven Design: Tackling Complexity in the Heart of Software*（Addison-Wesley，2003年）中所阐述的领域驱动设计（DDD）的原则。不过我们没有用面向对象的方式，而是决定采用函数式编程。一切的开始都像是一个实验，但在最后证明这是一次非常成功并且令人满意的经历。现在当我回头看时，发现DDD的内容与软件工程的通用规则非常协调一致。因此也不用担心函数式、领域驱动设计会显得像是领域建模的典型范例。

这本书是我们成功运用函数式编程进行领域模型开发的证据。我决定跟读者分享我们遵守的实践、采用的原则，以及在实现中所使用的Scala风格。Manning出

版社完全同意这个想法并决定继续该项目。

不管你的领域模型是什么样的，定义实现成功的一个关键标准是应用的响应能力。没有一个用户喜欢盯着屏幕上的等待光标，根据我们的经验来看，这通常是因为架构师非必要地阻塞了主线程的执行。需要花费时间执行的昂贵的操作应该用异步的方式来执行，把主线程空出来给其他用户行为。*The Reactive Manifesto*（www.reactivemanifesto.org）中定义了建模所需要使用的特性，以便保证应用程序是非阻塞、响应及时的，并避免巨大延迟带来的恶劣影响。这也是我要在书中写的另一个方面。在经过与 Manning 团队多次友好的商讨后，我们决定在这本书中将函数与响应式编程结合起来。

于是本书就诞生了。通过这个项目，我收获了巨大的乐趣，也希望读者能有类似的体验。我收到了无数读者、评审者、良好祝愿者们的留言，他们陪着我一起提升了这本书的质量。我也非常感谢来自 Manning 出版社经验丰富的编辑以及评审者团队的巨大支持。

致谢

我要感谢很多人，他们直接或间接地参与了这本书的创作。

首先，我要感谢 Martin Odersky，Scala 编程语言的创建者，我用 Scala 完成了所有函数响应式领域建模的案例。同时也非常感谢你建立了 Scalaz，这个有趣的库使我们在用 Scala 语言进行纯函数编程时充满乐趣。

Twitter 是一个非常酷的沟通方式，承载了各种各样的讨论。我在上面和一些牛人就函数式编程有过很多非常激烈的讨论。感谢每一位牛人，是你们促使我完成了这本书。

感谢所有的评审者：Barry Alexander、Cosimo Attanasi、Daniel Garcia、Jan Nonnen、Jason Goodwin、Jaume Valls、Jean-François Morin、John G. Schwitz、Ken Fricklas、Lukasz Kupka、Michael Hamrah、Othman Doghri、Rintcius Blok、Robert Miller、Saleem Shafi、Tarek Nabil，以及 William E. Wheeler。时间可能是我们拥有的最宝贵的资源，我非常感谢他们愿意在这本书上花费时间，每个评审者都给了我很棒的建议，极大地提升了这本书的质量。

感谢所有购买了 MEAP[1] 的读者，在作者在线论坛里的定期沟通，一直鼓励着我完成这本书。特别要感谢 Arya Irani，她贡献的一个 pull 请求帮助我更新了 monad

1 Manning Early Access Program，在书的写作过程中，你可以阅读到刚完成的章节，并第一时间得到最终版本的电子版。——译者注

代码（从基于 Scalaz 7.1 到 7.2）。同样要特别感谢 Thomas Lockney 和 Charles Feduke，他们对每个不同的 MEAP 版本做了彻底的技术评审。

我还要感谢 Manning 出版社再次信任我。在我写第一本书的时候，我们有过非常美好的合作，而再次合作甚至更有乐趣。我要感谢以下 Manning 员工的杰出工作。

- 感谢 Michael Stephens 和 Christina Rudloff 促使我启动这个项目。
- 感谢 Jennifer Stout 在 10 个章节的漫长过程中不屈不挠地纠正了我所有的错误。
- 感谢 Alain Gouniot 在整个过程中提供了深入的技术评审。
- 感谢 Gandace Gilhooley 与 Ana Romac 帮助推动这本书。
- 感谢 Mary Piergies、Kevin Sullivan、Maureen Spencer，以及所有幕后工作人员（包括 Sharon Wilkey、Alyson Brener、April Milne，以及 Dennis Dalinnik），他们帮助我把一个粗糙的草稿变成一本真正的书。

感谢 Jonas Bonér 为我的书写序。我很荣幸，我与 Jonas 已经相识了很长时间，他也是我很多软件开发项目的重要灵感来源。

最后，我要感谢我的妻子、母亲以及我的儿子 Aarush，他们给我提供了最完美的“生态环境”，在那里，写一本关于函数式编程的书这种创造性任务才有可能完成。

关于本书

本书内容涉及如何使用函数式编程实现领域模型，以及如何通过使用响应式原则（诸如非阻塞计算和异步消息）来确保模型的响应性。

领域模型都是针对问题领域的，可以通过很多方式实现一个解决方案框架——能提供与问题领域模型相同的函数性，通常会使用面向对象技术来设计领域模型。本书中使用了一种正交方式——用纯函数对领域行为建模，用代数数据类型对领域实体建模，并将不变性作为设计空间的一个主关注点。作为读者，你能学到基于代数技术的函数式设计模式，可以将其直接用于实现自己的领域模型。

这本书同样还包括了响应式编程——使用 future、promise、actor 以及 stream 来确保模型在有一定延迟的条件下有足够的响应性和可操作性。

书中使用 Scala 语言来实现领域模型。作为 JVM 的一个“常驻民”，在面向对象以及函数式编程准则的强力支持下，今天 Scala 已经是最广泛使用的语言之一。尽管如此，本书讨论的核心准则同样适用于其他函数式语言，比如 Haskell。

内容简介

第 1 章会对从书中能学习到什么做一个全面的讨论。这一章中会提供一个关于领域模型的概览，同时讨论一些领域驱动设计背后的概念。也会谈到函数式编程（FP）的核心原则，以及将领域模型设计得足够透明并且根据纯逻辑进行边界解耦后所能

得到的好处。这一章还定义了*响应式模式*，包括如何将 FP 和响应式设计这两个概念捆绑在一起使得模型更具有响应性和可伸缩性。

第 2 章会讨论用 Scala 作为函数响应式领域建模实现语言的好处。它会讨论静态类型的好处，以及 Scala 的高级类型系统如何让模型更加健壮并经得起考验。在这一章中，还会学到如何将 OO 和 FP 的力量结合起来实现模块化的干净的模型。

第 3 章从讨论一个代数 API 设计开始。先不考虑实现，可以基于抽象的代数来设计 API。这一章结合大量的细节以及现实世界中个人银行系统建模的一些例子，呈现了这种方式的优点。代数有它自身的法则，基于代数建设 API 时，要确保实现遵守这些法则。这一章以一些关于领域对象生命周期的讨论作为结尾，从工厂中生产出来开始，然后执行领域行为，最后持久化。

第 4 章聚焦在函数式设计模式上，这跟以前所学的面向对象的设计模式有极大的不同。一个函数式设计模式是基于代数方法的，它可以有很多种实现方式（或诠释），因此在重用性上会远超过 OO 设计模式。这一章将讨论 functor、applicative、monad，这些都是函数式编程语言中的最基本的可复用模式。这一章还会讨论一些使用案例，如基于这些模式的代数方法如何演变领域模型。

第 5 章是关于如何模块化领域模型。一个非凡的领域模型是一系列小模型的集合，每个小模型都被认为是边界上下文（bounded context）。这一章会解释如何将边界上下文设计成独立的加工品，以及如何确保多个边界上下文之前的通信在空间和时间上的解耦。这是领域驱动设计的核心概念之一，而且可以用异步消息来很容易地实现。本章还将介绍 free monad，另一种运用函数式编程概念的高级模块化技术。

第 6 章将讨论响应式领域模型。这一章涉及如何设计响应式 API，既不会阻塞主线程的执行，同时又能保证模型的响应性。这一章会提供领域对象和边界上下文（如 future、promise、actor 和 reactive stream）之间非阻塞式通信的各种方式，还会讨论一个使用场景，即在个人银行领域中如何运用 reactive stream。

第 7 章讲解了 reactive stream。用 Akka Stream 实现了一个中等规模的用例来证明 reactive stream 的威力。第 6 章涉及了 actor 模型的缺点，而第 7 章则告诉我们如何用 Akka Stream 实现类型 API 来克服这些缺点。

第 8 章覆盖了领域模型的持久化（persistence）。这一章从一个基于 CRUD[1] 持久化模型的评论开始，介绍运用事件驱动技术的响应式持久化的概念。这一章结合了当前模型的状况介绍领域事件的完整历史，讨论诸如 CQRS 和事件源的实现技术，这也产生了一个更有弹性的持久化模型。这一章也用 Slick[2] 演示了一个基于 CRUD 的实现，一种针对 RDBMS 的通用的函数型到关系型映射框架。

---

1 Create、Retrieve、Update、Delete。——译者注

2 一种响应式的jQuery插件。——译者注

第 9 章是关于测试领域模型的。它从基于 xUnit[1] 经典的测试方法论开始，然后列出其中的缺陷以及如何使用代数测试进行改进。这一章介绍了基于属性的测试，它允许用户编写代数属性，然后通过运行时自动生成的数据来验证它。这一章会使用 Scala 基于属性的测试库 ScalaCheck，结合前面章节已有的领域模型来讨论这种技术的实现方式。

本书最后在第 10 章中对核心概念进行回顾，并讨论领域建模未来的发展趋势。

代码约定及下载

书中所有源代码都用等宽字体来和普通文本区分。有时需要将一行代码分成两行甚至更多行来适应书页。我们会用这样的箭头➥来表示连续行。

代码中包含很多注释。在一些情况下，会用数字标记与文后的注解对应关联。

书中案例的代码可以从出版社网站 https://www.manning.com/books/functional-and-reactive-domain-modeling 以及 GitHub 网址 https://github.com/debasishg/frdomain 上进行下载。

测验与练习

书中包含一系列测验与练习，它们会帮助读者了解自身对讨论内容的理解程度。第 1 章和第 2 章包含一些基本概念的测验。每个“测验时间”之后的一页或两页就会有相应的“测验答案”，如以下例子所示。

测验时间 1.1　你认为这种模型最主要的缺点是什么？

测验答案 1.1　主要问题是易变性，它会从两个方面打击你：很难在并行设置下使用抽象，而且很难推理你的代码。

从第 3 章开始，练习会复杂一些。测验被编号的练习所替代——实际的建模问题，这些练习将聚焦在对应章节所讨论的概念上。练习如同以下例子。

练习 3.2 验证透镜规律

第 3 章的在线代码库中包含一个 `Customer` 实体的定义以及它的透镜。观察 `addressLens`，它将更新一个用户的地址，然后用 ScalaCheck 写出属性来验证透镜规律。

针对解决方案领域的一个特定用例建模可以通过不同的途径来完成。练习会讨

1　如JUnit。——译者注

论这些候选方案以及每个方案的优缺点。我们鼓励读者独立完成它们而不是直接看解决方案，这些方案同样可以在出版社网站（https://www.manning.com/books/functional-and-reactive-domain- modeling） 和 GitHub（https://github.com/debasishg/frdomain）上找到。

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/32392

关于作者

Debasish Ghosh 在领域建模方面有 10 年的工作经验，在过去的 5 年里他主要专注在函数式建模领域。他有着丰富的函数式编程经验，特别在使用 Scala 语言以及 Akka、Scalaz 库方面，这也是本书的基础。他同样是事件源、CQRS 最早的使用者，并在实际应用程序中实践了这些技术。Debasish 还是一本领域相关语言书籍——*DSLs in Action*（www.manning.com/books/dsls-in-action）的作者，该书由 Manning 出版社在 2010 年出版。

关于译者

李源，曾在华为技术有限公司工作8年，经历过开发、SE、PM和PQA等多个岗位，目前在途牛旅游网担任研发总经理一职，是美国质量协会（ASQ）注册质量工程师（CQE）；译者有丰富的开发、架构设计及研发管理经验，先后负责过多个大型项目的方案设计和系统规划，对于C++、Java以及设计模式等领域都有比较深入的研究；曾翻译《Java性能调优指南》一书。

读者可扫码联系译者：

目录