

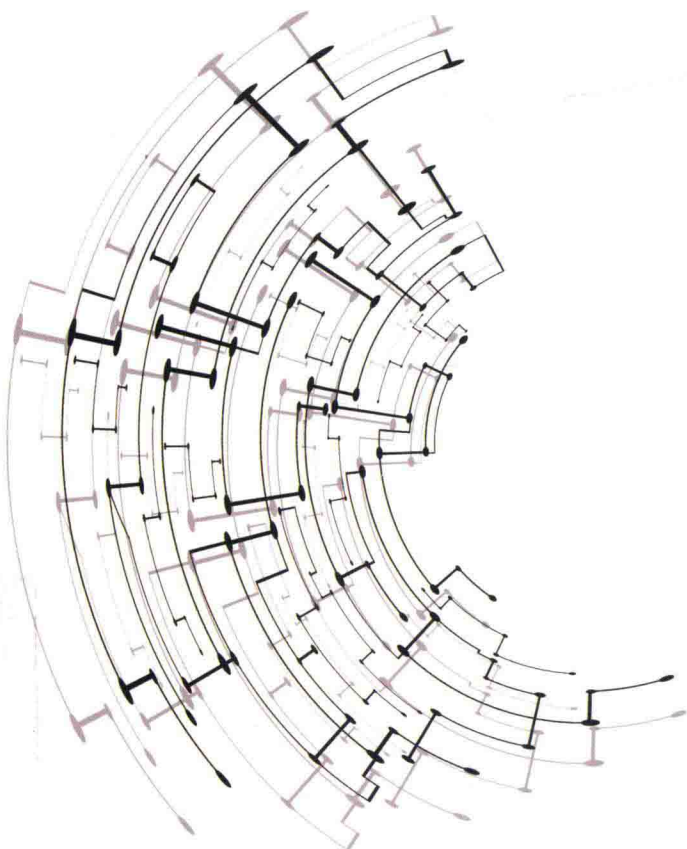
FUNCTIONAL PROGRAMMING IN C#, HOW TO WRITE BETTER C# CODE

C#函数式编程

编写更优质的C#代码

[美] 恩里科·博南诺(Enrico Buonanno) 著

张久修 译



 MANNING



清华大学出版社

C#函数式编程

编写更优质的 C#代码

[美]恩里科·博南诺(Enrico Buonanno) 著

张久修 译



清华大学出版社

北 京

Enrico Buonanno

Functional Programming in C#, How to Write Better C# Code

EISBN: 978-1-61729-395-5

Original English language edition published by Manning Publications, 178 South Hill Drive, Westampton, NJ 08060 USA. Copyright © 2017 by Manning Publications. Simplified Chinese-language edition copyright © 2018 by Tsinghua University Press. All rights reserved.

本书中文简体字版由 Manning 出版公司授权清华大学出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

北京市版权局著作权合同登记号 图字：01-2017-8973

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

图书在版编目(CIP)数据

C#函数式编程 编写更优质的 C#代码 / (美)恩里科·博南诺(Enrico Buonanno) 著；张久修译. —北京：清华大学出版社，2019

书名原文：Functional Programming in C#, How to Write Better C# Code

ISBN 978-7-302-51055-0

I. ①C… II. ①恩… ②张… III. ①C 语言—程序设计 IV. ①TP312.8

中国版本图书馆 CIP 数据核字(2018)第 192028 号

责任编辑：王 军 韩宏志

封面设计：周晓亮

版式设计：思创景点

责任校对：牛艳敏

责任印制：丛怀宇

出版发行：清华大学出版社

网 址：<http://www.tup.com.cn>, <http://www.wqbook.com>

地 址：北京清华大学学研大厦 A 座 邮 编：100084

社 总 机：010-62770175 邮 购：010-62786544

投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈：010-62772015, zhiliang@tup.tsinghua.edu.cn

印 装 者：三河市龙大印装有限公司

经 销：全国新华书店

开 本：170mm×240mm 印 张：24 字 数：484 千字

版 次：2019 年 1 月第 1 版 印 次：2019 年 1 月第 1 次印刷

定 价：98.00 元

产品编号：078166-01

译者序

历经数月，终于完成了本书的翻译工作。没有过多的轻松和兴奋之情，反而更多的是一份疼惜，一份不舍。从事技术书籍的翻译工作，既是一个付出的过程，也是一个收获的过程，更是对自己技术的一种沉淀。每当一本书的翻译工作结束后，我便会无所适从，仿佛少了些什么似的。每天都要花费几个小时的翻译工作，可让我的内心真正沉静下来，只专注于字里行间。那种状态、那种感觉真的很好，也正是如此，内心便充满依依不舍之情。

生活中充满了未知以及不确定性，谁也不能保证你的内心每时每刻都是清静的。在翻译本书的这段时间内，我也经历了或多或少会影响心境的一些琐事。可见，想要专心地做一件需要长期坚持且耗费精力的事情是多么艰难。因此，虽未曾写过书，但即使是翻译一本书，也能深深体会到作者付出的心血，可以想象到写书人夜以继日工作的情景。同时，对于能成为本书的译者，我感到特别荣幸。

正所谓“工欲善其事，必先利其器”。函数式编程作为主流编程的重要组成部分，必将是高级程序员手中的一大利器。而想要掌握并使用这件利器，首先需要了解和培养函数式思维，以不同的视角来看待代码。我们首先需要从宏观上了解函数式编程，了解它是什么，及其定义、特性、优点分别是什么，甚至其所存在的缺点或顾虑又有哪些。

顾名思义，函数式编程是一种特定的编程方式，将计算机运算视为函数的计算。简单来说它是一种“编程范式”，也就是如何编写程序的方法论。同时，它又属于“结构化编程”的一种，主要思想是将运算过程尽量写成一系列嵌套的函数调用。与指令式编程相比，函数式编程强调函数的计算比指令的执行重要；与过程化编程相比，函数式编程里函数的计算可随时调用。而函数编程语言最重要的基础是 λ 演算(lambda calculus)，而且 λ 演算的函数可接受函数当作输入(参数)和输出(返回值)。

函数式编程具有三个特性：闭包和高阶函数，惰性计算，以及递归。并有五个鲜明特点：函数是“一等公民”(first class)、只用“表达式”而非“语句”、没有“副作用”(side effect)、不修改状态、具有引用透明性。函数式编程的优点为：代码简洁且开发快速、接近自然语言而且易于理解、方便了代码管理、易于“并发编程”，且利于代码的热升级。

凡事有利必有弊。在良好的特性和优点的背后，必然也存在顾虑。函数式编程常被认为严重耗费 CPU 和存储器资源，主因有二：其一是，早期的函数式编程语言实现时并未考虑过效率问题；第二点是，有些非函数式编程语言为提升速度，不提供自动边界检查或自动垃圾回收等功能。同时，惰性求值亦为语言(如 Haskell)增加了额外的管理工作。

以上只是对函数式编程的一个宏观介绍；而通过本书，你可以更加系统、详细地了解并掌握函数式编程。本书由浅入深地讲解函数式编程的基本原理和技术，并通过现实中的各种示例场景带领你完成非函数式编程代码到函数式编程代码的不断重构，以逐渐让你树立函数式编程的思想，引领你从一个全新的视角看待代码。本书必将让你受益匪浅。

作为本书的译者，我本着“诚惶诚恐”的态度投入工作，为避免误导读者，文中的一词一句皆反复斟酌。但是鉴于译者水平有限，错误和失误在所难免，如有任何意见和建议，请不吝指正，感激不尽！

关于本书

如今，函数式编程(Functional Programming, FP)已成为主流编程的一个重要且令人兴奋的组成部分。近十年来创建的大多数语言和框架都是函数式的，这导致人们纷纷预测编程的未来也将是函数式的。与此同时，诸如 C#和 Java 的面向对象主流语言，在其每个新版本中都会引入更多函数式特性，从而实现多范式编程风格。

然而，C#社区关于函数式编程的推行却十分缓慢。为何会这样呢？我认为，其中一个原因是缺乏优秀的文献：

- 大多数 FP 文献都是用函数式语言(尤其是 Haskell)编写的。而对于具有 OOP(Oriented Object Programming, 面向对象编程)背景的开发人员来说，要学习其概念就必须跨越编程语言的障碍。尽管许多概念适用于诸如 C#的多范式语言，但同时学习一种新范式和新语言是一项艰巨的任务。
- 更重要的是，文献中的大部分书籍倾向于用数学或计算机科学领域的例子来阐明函数式编程的技术和概念。对于大部分终日从事业务(LOB)应用开发的程序员来说，这会产生一个领域差异，并使得他们难以知悉这些技术与实际应用间的相关性。

这些缺陷是我学习 FP 的主要绊脚石。有些书籍试图解释什么是“柯里化”，通过利用数字 3，来创建一个可将 3 添加到任意数字的函数，以展示 add 函数是如何被柯里化的(在你能想到的所有应用中，它有一点点实际用处吗？)。放弃此类书籍后，我决定追寻一条属于自己的研究之路。这包括学习 6 种函数式语言(最优秀的几种语言)，并探究 FP 中的哪些概念可在 C#中有效地应用，以及研究由大量开发人员有偿撰写的该类型的应用，并且最终撰写了本书。

本书展示如何利用 C#语言的函数式技术来弥补 C#开发人员的语言差异。还展示如何将这些技术应用于典型的业务场景来弥补领域差异。我采取了一种务实的方法，并且涵盖了函数式技术，使其在典型的 LOB 应用场景中非常有用，并省略了 FP 背后的大部分理论。

最终，你应该关注 FP，因其赋予了以下优势：

- **高效率**——这意味着可用更少的代码完成更多工作。FP 提高了抽象层级，使你可编写高级代码，同时将你从那些只是增加复杂性，却没有任何价值

的低级技术层面解放出来。

- **安全性**——在处理并发性时尤其如此。一个用命令风格编写的程序可能在单线程实现中运行良好，但当并发发生时会导致各种错误。函数式代码在并发场景中提供了更好的保障，因此，在多核处理器时代，我们很自然会看到开发人员对 FP 的兴趣激增。
- **清晰性**——相对于编写新代码，我们会花费更多时间来维护和使用现有的代码，所以让我们的代码清晰明了并且意义明确是非常重要的。当你转向函数式思维时，达到这种清晰性将水到渠成。

如果你已经用面向对象的风格进行了一段时间的编程，在本书的概念实现之前，可能需要做出一些努力和意愿去尝试。为确保学习 FP 是一个愉快而有益的过程，我有两个建议：

- **耐心**——你可能需要多次重复阅读一些章节。你可能会把这本书放下几个星期，当你再次拿起这本书时，突然间发现有些模糊的东西开始变得有意义了。
- **用代码进行实验**——实践出真知。本书提供了许多示例和练习，许多代码片段可在 REPL 中进行测试。

你的同事可能比你更不愿意去探索新东西。预料到他们可能会抗议你采用这种新风格，并对你的代码感到困惑，然后发问“为什么不只是做 x？”(其中 x 是枯燥的、过时的，并且通常是有害的)。不必过多地讨论。坐下来，看着他们最终转身，并用你的技术来解决他们屡次遇到的问题。

致 谢

感谢 Paul Louth, 他不但通过自己编写的 LanguageExt 库赋予我灵感(我从中借鉴了很多很棒的想法), 而且亲力亲为地在各个阶段对本书进行了审阅。

Manning 出版社的详尽编辑过程确保了本书的质量。为此, 我要感谢与本书合作的团队, 包括 Mike Stephens、开发编辑 Marina Michaels、技术编辑 Joel Kotarski 技术校对员 Jürgen Hoetzel, 以及版权编辑 Andy Carroll。

特别感谢 Daniel Marbach 和 Tamir Dresher 所给予的技术见解, 以及所有参与同行评审的人, 包括 Alex Basile、Aurélien Gounot、Blair Leduc、Chris Frank、Daniel Marbach、Devon Burriss、Gonzalo Barba López、Guy Smith、Kofi Sarfo、Pauli Sutelainen、Russell Day、Tate Antrim 和 Wayne Mather。

感谢 Scott Wlaschin 在 <http://fsharpforfunandprofit.com> 上分享的文章, 感谢所有通过文章、博客和开源代码分享自己的知识和热情的其他 FP 社区成员。

前言

本书旨在展示如何利用 C# 中的函数式技术编写简洁、优雅、健壮和可维护的代码。

本书读者对象

本书是为那些具有雄心壮志的开发人员所编写的。你需要了解 C# 语言和 .NET 框架。你需要具备开发实际应用的经验，熟悉 OOP 的概念、模式和最佳实践。并且，你正在寻求通过学习函数式技术来扩展编程技能，以便可以充分利用 C# 的多范式语言特性。如果你正在尝试或正在计划学习一门函数式语言，那么本书也将是非常有价值的，因为你将学习如何在—门你所熟悉的语言上进行函数式思考。改变自己的思考方式是很难的；而一旦做到，那么学习任何特定语言的语法将变得相对容易。

本书的组织结构

全书共 15 章，分为 3 个部分：

- 第 I 部分介绍函数式编程的基本技术和原理。我们将初窥函数式编程是什么，以及 C# 是如何支持函数式编程风格的。然后，将研究高阶函数的功能、纯函数及其与可测性的关系、类型和函数签名的设计，以及如何将简单的函数组合到复杂的程序中。在第 I 部分的最后，你将很好地感受到一个用函数式风格所编写的程序是什么样的，以及这种风格所带来的好处。
- 第 II 部分将加快速度，转向更广泛的关注点，例如函数式的错误处理、模块化和组合应用，以及理解状态和表示变化的函数式方法。到第 II 部分结束时，你将掌握一系列工具的用法，将能利用函数式方法来有效地完成许多编程任务。
- 第 III 部分将讨论更高级的主题，包括惰性求值、有状态计算、异步、数据流和并发性。第 III 部分的每章都介绍一些重要技术，它们可能彻底改变你编写软件的方式和思考方式。

你会在每章中找到更详细的主题分类，并在阅读任何特定章节之前，都能从本书的内封了解到需要预先阅读哪些章节。

为实际应用编码

本书旨在让实际场景保持真实。为此，很多例子都涉及实际任务，例如读取配置、连接数据库、验证 HTTP 请求；对于这些事情，你可能已经知道如何做了，但你将用函数式思维的新视角来重新看待它们。

在本书中，我使用了一个长期运行的例子来说明在编写 LOB 应用时，FP 是如何提供帮助的。为此，我选择了一个在线银行应用，它是虚拟的 Codeland 银行 (BOC)——我知道这或许有些生搬硬套了，但至少它有了必需的三个字母的缩写。由于大多数人都可访问在线银行设施，因此很容易想象其所需的功能，并且清楚地看到所讨论的问题是如何与实际应用关联的。

我也使用了场景来说明如何解决函数式风格中典型的编程问题。在实际的例子和 FP 概念之间的不断反复，将帮助我们弥合理论与实践之间的差异。

利用函数式库

诸如 C# 的语言具有函数式特性，但为了充分利用这些特性，你将经常使用便于实现常见任务的库。Microsoft 已经提供了几个库，以便进行函数式风格的编程，包括：

- **System.Linq**——这是一个功能库。我假定你是熟悉它的，因为它是 .NET 的一个重要组成部分。
- **System.Collections.Immutable**——这是一个不可变集合的库，第 9 章将开始使用它。
- **System.Reactive**——这是 .NET 的 Reactive Extensions 的实现，允许你使用数据流，第 14 章将讨论这些数据流。

当然还有其他许多重要的类型和功能未列举，这些都是 FP 的主要部分。因此，一些独立的开发人员已经编写了一些开源的代码库来填补这些空白。到目前为止，其中最完整的是 LanguageExt，这是由 Paul Louth 编写的一个库，用于在进行函数式编码时改进 C# 开发人员的体验。¹

本书并没有直接使用 LanguageExt；相反，将向你展示如何开发自己的函数式实用工具库，且将其命名为 LaYumba.Functional，尽管它与 LanguageExt 在很大程

1 LanguageExt 是开源的，可在 GitHub 和 NuGet 上找到：<https://github.com/louthy/language-ext>。

度上是重叠的，但这在教学方面会更有用，原因有如下几点：

- 在本书出版后，将保持代码的稳定。
- 你可以透过现象看本质，将看到看似简单实则强大的函数式构造。
- 你可以专注于基本要素：我将以最纯粹的形式向你展示这些构造，这样你就不会被一个完整的库所处理的细节和边缘情况分散注意力。

代码约定和下载

代码示例使用了 C# 7，大部分与 C# 6 兼容。C# 7 中专门介绍的语言特性仅用于第 10 章及之后章节(另外，1.2 节的几个示例中明确地展示了 C# 7)。可在 REPL 中执行许多较短的代码片段，从而获得动手练习的实时反馈。更多的扩展示例可通过 <https://github.com/la-yumba/functional-csharp-code> 下载，其中还配有练习的设置和解决方案。

本书中的代码清单重点讨论了正在讨论的主题，因此可能会省略命名空间(namespace)、using 语句、简单的构造函数，或先前代码清单中出现的并保持不变的代码段。如果你想查看代码清单的完整编译版本，可在代码存储库中找到它：<https://github.com/la-yumba/functional-csharp-code>。

另外，读者也可扫描封底的二维码下载相关资料。

图书论坛

购买本书后，可免费访问由 Manning 出版社运行的私人网络论坛，你可在这里提交有关本书的评论，询问技术问题，并获得作者和其他用户的帮助。可通过 <https://forums.manning.com/forums/functional-programming-in-c-sharp> 访问该论坛。你也可通过 <https://forums.manning.com/forums/about> 了解更多关于 Manning 论坛及论坛行为准则的信息。

Manning 出版社为读者提供一个场所，在这里，读者之间以及读者和作者之间可以进行有意义的对话。但不承诺作者的任何具体参与度，作者对论坛的贡献是自愿的(并且是无偿的)。我们建议你尝试向作者提出一些具有挑战性的问题，以免他的兴趣流失！只要本书还在市场上销售，论坛和之前所讨论的内容存档将可从出版商的网站上直接访问。

目 录

第 I 部分 核心概念

第 1 章 介绍函数式编程	3
1.1 什么是函数式编程	4
1.1.1 函数作为第一类值	4
1.1.2 避免状态突变	4
1.1.3 编写具有强力保证的 程序	5
1.2 C#的函数式语言	8
1.2.1 LINQ 的函数式性质	9
1.2.2 C# 6 和 C# 7 中的函数式 特性	10
1.2.3 未来的 C#将更趋函 数化	13
1.3 函数思维	13
1.3.1 映射函数	13
1.3.2 在 C#中表示函数	14
1.4 高阶函数	18
1.4.1 依赖于其他函数的 函数	18
1.4.2 适配器函数	20
1.4.3 创建其他函数的函数	20
1.5 使用 HOF 避免重复	21
1.5.1 将安装和拆卸封装到 HOF 中	23
1.5.2 将 using 语句转换为 HOF	24
1.5.3 HOF 的权衡	25

1.6 函数式编程的好处	27
练习	27
小结	28

第 2 章 为什么函数纯洁性很 重要	29
2.1 什么是函数的纯洁性	29
2.1.1 纯洁性和副作用	30
2.1.2 管理副作用的策略	31
2.2 纯洁性和并发性	33
2.2.1 纯函数可良好地并 行化	34
2.2.2 并行化不纯函数	35
2.2.3 避免状态的突变	36
2.3 纯洁性和可测性	38
2.3.1 实践：一个验证场景	39
2.3.2 在测试中引入不纯 函数	40
2.3.3 为什么很难测试不纯 函数	42
2.3.4 参数化单元测试	43
2.3.5 避免标头接口	44
2.4 纯洁性和计算的发展	47
练习	47
小结	48

第 3 章 设计函数签名和类型	49
3.1 函数签名设计	49
3.1.1 箭头符号	50

3.1.2 签名的信息量有多大.....	51	4.3 使用 Bind 来链接函数.....	85
3.2 使用数据对象捕获数据.....	52	4.3.1 将返回 Option 的函数 结合起来.....	85
3.2.1 原始类型通常不够 具体.....	53	4.3.2 使用 Bind 平铺嵌套 列表.....	87
3.2.2 使用自定义类型约束 输入.....	53	4.3.3 实际上,这被称为 单子.....	88
3.2.3 编写“诚实的”函数.....	55	4.3.4 Return 函数.....	88
3.2.4 使用元组和对对象来组 合值.....	56	4.3.5 函子和单子之间的 关系.....	89
3.3 使用 Unit 为数据缺失 建模.....	58	4.4 使用 Where 过滤值.....	90
3.3.1 为什么 void 不理想.....	58	4.5 使用 Bind 结合 Option 和 IEnumerable.....	91
3.3.2 使用 Unit 弥合 Action 和 Func 之间的差异.....	59	4.6 在不同抽象层级上编码.....	92
3.4 使用 Option 为数据可能 缺失建模.....	61	4.6.1 常规值与高级值.....	93
3.4.1 你每天都在使用糟糕 的 API.....	61	4.6.2 跨越抽象层级.....	94
3.4.2 Option 类型的介绍.....	62	4.6.3 重新审视 Map 与 Bind.....	95
3.4.3 实现 Option.....	65	4.6.4 在正确的抽象层级上 工作.....	96
3.4.4 通过使用 Option 而不是 null 来获得健壮性.....	68	练习.....	96
3.4.5 Option 作为偏函数的 自然结果类型.....	69	小结.....	97
练习.....	73	第 5 章 使用函数组合设计程序.....	99
小结.....	74	5.1 函数组合.....	99
第 4 章 函数式编程中的模式.....	77	5.1.1 复习函数组合.....	100
4.1 将函数应用于结构的内 部值.....	77	5.1.2 方法链.....	101
4.1.1 将函数映射到序列上.....	77	5.1.3 高级值界域中的组合.....	101
4.1.2 将函数映射到 Option.....	79	5.2 从数据流的角度进行 思考.....	102
4.1.3 Option 是如何提高抽象 层级的.....	81	5.2.1 使用 LINQ 的可组合 API.....	102
4.1.4 单子.....	82	5.2.2 编写可组合性更好 的函数.....	103
4.2 使用 ForEach 执行副 作用.....	83	5.3 工作流编程.....	105
		5.3.1 关于验证的一个简单 工作流.....	106
		5.3.2 以数据流的思想进行 重构.....	107

5.3.3 组合带来了更大的 灵活性	108
5.4 介绍函数式领域建模	109
5.5 端到端的服务器端 工作流	110
5.5.1 表达式与语句	112
5.5.2 声明式与命令式	112
5.5.3 函数式分层	113
练习	115
小结	115

第 II 部分 函数式风格

第 6 章 函数式错误处理	119
6.1 表示输出的更安全方式	120
6.1.1 使用 Either 捕获错误 细节	120
6.1.2 处理 Either 的核心 函数	123
6.1.3 比较 Option 和 Either	124
6.2 链接操作可能失败	125
6.3 验证: Either 的一个完美 用例	127
6.3.1 为错误选择合适的 表示法	128
6.3.2 定义一个基于 Either 的 API	129
6.3.3 添加验证逻辑	130
6.4 将输出提供给客户端应用 程序	131
6.4.1 公开一个类似 Option 的 接口	132
6.4.2 公开一个类似 Either 的 接口	134
6.4.3 返回一个 DTO 结果	134
6.5 Either 的变体	136
6.5.1 在不同的错误表示之间 进行改变	136

6.5.2 Either 的特定版本	137
6.5.3 重构 Validation 和 Exceptional	138
6.5.4 保留异常	141
练习	142
小结	142

第 7 章 用函数构造一个应用

程序	145
7.1 偏函数应用: 逐个提供 参数	146
7.1.1 手动启用偏函数应用	147
7.1.2 归纳偏函数应用	148
7.1.3 参数的顺序问题	150
7.2 克服方法解析的怪癖	150
7.3 柯里化函数: 优化偏函数 应用	152
7.4 创建一个友好的偏函数 应用 API	155
7.4.1 可文档化的类型	156
7.4.2 具化数据访问函数	157
7.5 应用程序的模块化及 组合	159
7.5.1 OOP 中的模块化	160
7.5.2 FP 中的模块化	162
7.5.3 比较两种方法	164
7.5.4 组合应用程序	165
7.6 将列表压缩为单个值	166
7.6.1 LINQ 的 Aggregate 方法	166
7.6.2 聚合验证结果	168
7.6.3 收获验证错误	169
练习	170
小结	171

第 8 章 有效地处理多参函数

8.1 高级界域中的函数应用 程序	174
----------------------------	-----

8.1.1	理解应用式	176	9.3.3	利用 F# 处理数据 类型	212
8.1.2	提升函数	177	9.3.4	比较不变性的策略：一场 丑陋的比赛	213
8.1.3	介绍基于属性的测试	179	9.4	函数式数据结构简介	214
8.2	函子、应用式、单子	181	9.4.1	经典的函数式链表	215
8.3	单子定律	182	9.4.2	二叉树	219
8.3.1	右恒等元	183	练习	223	
8.3.2	左恒等元	183	小结	224	
8.3.3	结合律	184			
8.3.4	对多参函数使用 Bind	185			
8.4	通过对任何单子使用 LINQ 来提高可读性	186	第 10 章	事件溯源：持久化的函数 式方法	225
8.4.1	对任意函子使用 LINQ	186	10.1	关于数据存储的函数式 思考	226
8.4.2	对任意单子使用 LINQ	188	10.1.1	为什么数据存储只能 追加	226
8.4.3	let、where 及其他 LINQ 子句	191	10.1.2	放松，并忘却存储 状态	227
8.5	何时使用 Bind 或 Apply	192	10.2	事件溯源的基础知识	228
8.5.1	具有智能构造函数的 验证	192	10.2.1	表示事件	228
8.5.2	使用应用式流来收集 错误	194	10.2.2	持久化事件	229
8.5.3	使用单子流来快速 失败	195	10.2.3	表示状态	230
练习	196		10.2.4	一个模式匹配的 插曲	231
小结	196		10.2.5	表示状态转换	234
			10.2.6	从过去的事件中重建 当前状态	235
第 9 章	关于数据的函数式思考	199	10.3	事件溯源系统的架构	236
9.1	状态突变的陷阱	200	10.3.1	处理命令	237
9.2	理解状态、标识及变化	202	10.3.2	处理事件	240
9.2.1	有些事物永远不会 变化	203	10.3.3	添加验证	241
9.2.2	表示非突变的变化	205	10.3.4	根据事件创建数据的 视图	243
9.3	强制不可变性	207	10.4	比较不可变存储的不同 方法	246
9.3.1	永远不可变	209	10.4.1	Datomic 与 Event Store	247
9.3.2	无样板代码的拷贝方法 的可行性	210			

10.4.2 你的领域是否受事件驱动?	247
小结	248

第Ⅲ部分 高级技术

第 11 章 惰性计算、延续以及单子组合之美	251
11.1 惰性的优点	251
11.1.1 用于处理 Option 的惰性 API	252
11.1.2 组合惰性计算	254
11.2 使用 Try 进行异常处理	256
11.2.1 表示可能失败的计算	257
11.2.2 从 JSON 对象中安全地提取信息	257
11.2.3 组合可能失败的计算	259
11.2.4 单子组合: 是什么意思呢?	260
11.3 为数据库访问创建中间件管道	261
11.3.1 组合执行安装/拆卸的函数	261
11.3.2 逃离厄运金字塔的秘方	263
11.3.3 捕获中间件函数的本质	263
11.3.4 实现中间件的查询模式	265
11.3.5 添加计时操作的中间件	268
11.3.6 添加管理数据库事务的中间件	269
小结	271

第 12 章 有状态的程序和计算	273
12.1 管理状态的程序	274
12.1.1 维护所检索资源的缓存	275
12.1.2 重构可测试性和错误处理	277
12.1.3 有状态的计算	278
12.2 一种用于生成随机数据的语言	279
12.2.1 生成随机整数	280
12.2.2 生成其他基元	281
12.2.3 生成复杂的结构	282
12.3 有状态计算的通用模式	284
小结	287
第 13 章 使用异步计算	289
13.1 异步计算	290
13.1.1 对异步的需要	290
13.1.2 用 Task 表示异步操作	291
13.1.3 Task 作为一个将来值的容器	292
13.1.4 处理失败	294
13.1.5 一个用于货币转换的 HTTP API	296
13.1.6 如果失败, 请再试几次	297
13.1.7 并行运行异步操作	297
13.2 遍历: 处理高级值列表	299
13.2.1 使用单子的 Traverse 来验证值列表	301
13.2.2 使用应用式 Traverse 来收集验证错误	302
13.2.3 将多个验证器应用于单个值	304

13.2.4	将 Traverse 与 Task 一起使用以等待多 个结果	305	14.4.1	检测按键顺序	330
13.2.5	为单值结构定义 Traverse	306	14.4.2	对事件源作出 反应	333
13.3	结合异步和验证(或其他 任何两个单子效果)	308	14.4.3	通知账户何时 透支	335
13.3.1	堆叠单子的问题	308	14.5	应该何时使用 IObservable?	337
13.3.2	减少效果的数量	310	小结		338
13.3.3	具有一个单子堆叠 的 LINQ 表达式	311	第 15 章	并发消息传递	339
小结		312	15.1	对共享可变状态的需要	339
第 14 章	数据流和 Reactive Extensions	315	15.2	理解并发消息传递	341
14.1	用 IObservable 表示数 据流	316	15.2.1	在 C# 中实现代理	343
14.1.1	时间上的一个序列 的值	316	15.2.2	开始使用代理	344
14.1.2	订阅 IObservable	317	15.2.3	使用代理处理并发 请求	346
14.2	创建 IObservable	318	15.2.4	代理与角色	349
14.2.1	创建一个定时器	319	15.3	“函数式 API”与“基于 代理的实现”	350
14.2.2	使用 Subject 来告知 IObservable 应何时发 出信号	320	15.3.1	代理作为实现 细节	351
14.2.3	从基于回调的订阅中 创建 IObservable	320	15.3.2	将代理隐藏于常规 API 的背后	352
14.2.4	由更简单的结构创 建 IObservable	321	15.4	LOB 应用程序中的并发 消息传递	353
14.3	转换和结合数据流	323	15.4.1	使用代理来同步对 账户数据的访问	354
14.3.1	流的转换	323	15.4.2	保管账户的注 册表	356
14.3.2	结合和划分流	325	15.4.3	代理不是一个 对象	357
14.3.3	使用 IObservable 进行 错误处理	327	15.4.4	融会贯通	359
14.3.4	融会贯通	329	小结		361
14.4	实现贯穿多个事件的 逻辑	330	结束语: 接下来呢?		363