高级.NET开发系列

.NET Patterns
Architecture,Design,and Process

.NET模式
架构、设计与过程

[美] Christian Thilmany 著
张晓坤 汤涛 谭立平 译

- 所有.NET环境下软件开发人员的完全指南
- 使用真实案例讲述模式应用全过程
- 完整的代码，方便读者学习使用

CHRISTIAN THILMANY
FOREWORD BY DAVID CHAPPELL

中国电力出版社
www.infopower.com.cn

高级.NET开发系列

.NET Patterns
Architecture,Design,and Process

.NET模式
架构、设计与过程

[美] Christian Thilmany 著
张晓坤 汤涛 谭立平 译

中国电力出版社
www.infopower.com.cn

.NET Patterns:Architecture,Design,and Process (ISBN 0-32-113002-2)
Christian Thilmany
Copyright © 2004 Pearson Education, Inc.
Original English Language Edition Published by Addison Wesley, Inc.
All rights reserved.
Translation edition published by PEARSON EDUCATION ASIA LTD and CHINA ELECTRIC POWER PRESS, Copyright © 2005.

本书翻译版由 Pearson Education 授权中国电力出版社独家出版、发行。
未经出版者书面许可，不得以任何方式复制或抄袭本书的任何部分。

本书封面贴有 Pearson Education 防伪标签，无标签者不得销售。
北京市版权局著作权合同登记号 图字：01-2004-1491 号

图书在版编目（CIP）数据
.NET 模式：架构、设计与过程 /（美）希蒙尼著；张晓坤，汤涛，谭立平译.
—北京：中国电力出版社，2005
（高级.NET 开发系列）
ISBN 7-5083-3273-3
I.N... II.①希...②张...③汤...④谭... III.计算机网络－程序设计 IV.TP393

中国版本图书馆 CIP 数据核字（2005）第 028354 号

丛 书 名：高级.NET 开发系列
书　　名：.NET 模式：架构、设计与过程
编　　著：（美）Christian Thilmany
翻　　译：张晓坤 汤涛 谭立平
责任编辑：陈维宁
出版发行：中国电力出版社
　　地址：北京市三里河路 6 号　　邮政编码：100044
　　电话：（010）88515918　　传　　真：（010）88518169
印　　刷：汇鑫印务有限公司
开本尺寸：185×233　　印　　张：20.75　　字　　数：470 千字
书　　号：ISBN 7-5083-3273-3
版　　次：2005 年 6 月北京第 1 版　　2005 年 6 月第 1 次印刷
定　　价：39.80 元
版权所有　翻印必究

关于本书的赞誉

Christian 从.NET 的角度重新审视了模式，他干得很棒。他探索出一些非常实用的模式，并且提供了.NET 环境下的指导以及实现的代码。

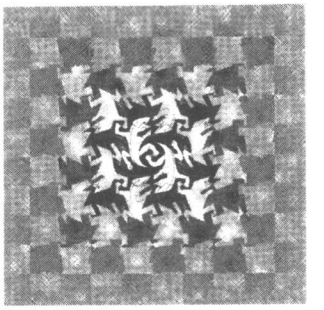
我尤其喜爱书中有关异常处理的冒泡模式的部分，因为我本人就笃信这种高级异常处理的管理。在这本书中，Christian 从表示层一直到数据层向读者展示了一系列的模式，并且重点讨论了 XML 和 XML Web 服务的场景。

Christian 还从模式和.NET 的实战经验中，提炼出一些切实可行的产品开发思路——这也是本书的精华所在。Christian 将这些内容组织在一起，放到了第六章中。这本书适用范围非常广泛：

- 刚刚转到.NET 下，并且开始学习面向对象的思想和模式的 VB6 开发人员
- 正考虑.NET 的模式应用的富有经验的面向对象开发人员
- 高级模式专家

Microsoft 区域主管和.NET 培训师
Stephen Fulcher

序



模式识别是人类智力的基本行为方式。自从 Christopher Alexander 意识到模式存在于良好的建筑中，软件开发人员也很快明显地谈论到存在于良好的代码中的模式。是什么在推动我们前进？重用（reusing）并非只有代码适用，对于我们创建代码的方式，它也同样有效。

如今可重用代码的最重要的部分之一就是类库，它是.NET 框架的一部分。这个软件的庞大集合提供了一些标准的方式，来操纵 XML 文档、创建 GUI（图形用户接口）、与其他系统通信以及更多其他功能。至少学会使用这些类库中某些部分，是任何 Windows 软件开发人员的任务。

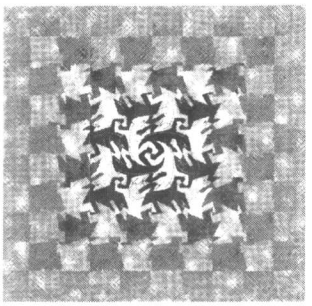
然而只学会这些技术仍然是不够的。理解.NET 命名空间如何工作，并不能自动为你产生高效地应用这种理解所需的智力工具。了解哪种模式在一个给定的.NET 上下文中工作得最好，是创建一个良好解决方案的关键部分。

这就是.NET 模式：架构、设计和过程产生的原因了。在本书中，Christian Thilmany 结合了设计模式世界的抽象和包含于.NET 中的具体开发解决方案。其结果便是一套观念，它们用于指导实践和更加正式的模式，而这些对于众多的.NET 开发人员来说都将是非常有用的。它们中有一些是很通用的，甚至可以被工作在任何软件环境中的任何人使用。还有一些依赖于.NET 的某些相关特征，在这一点上，它恰好是你期望从书中得到的。无论哪种情形，在.NET 世界中的软件专业人员都可以从这些以往的经验中获得好处。

设计模式活动在 Java 社区已经存在很长时间，并成为 Java 社区不可分割的部分。我非常高兴地看到随着.NET 的诞生，Microsoft 世界也明确地包含了这些观念。这本书就是迈向这条路上的极具价值的一步。

David Chappell
San Francisco,CA

此为试读，需要完整PDF请访问：www.ertongbook.com



前言

.NET 之路和本书

到目前为止，在软件架构、设计和“专业化”原则领域，我们已经被众多混乱的材料所淹没了。当一门新语言或者技术出现在市面上时，对于设计材料方面的需求似乎总会达到巅峰状态。当 Java 刚开始面世的时候，你可能看到开发人员涌入书店中，选择最新出版的《Learn Java in 10 Minutes》。Java 不仅仅是一门新语言，也是一个新平台，还是一种用来开发丰富的 Internet 应用程序的语言。开发人员不仅面临着一门新语法的挑战，而且还不得不学习这门新开发方法的新语义。对于那些没有接触过面向对象技术的人来说，这意味着更陡的学习曲线（learning curve）。在 Java 中每样东西都是对象，在精通了语法和基础库之后，另一个学习级别就是设计健壮和可重用的应用程序。自从 Java 诞生，我就从事 Java 方面的编程，我希望我能与您分享在一些材料所提供的新技术中我所看到的一些东西。这本书讲的几乎全是许许多多的“do's and don'ts（做什么和不做什么）”，这些都是在我学习.NET 并努力精通它的过程中所遇到的。

在作为一个.NET 架构师返回 Microsoft 和热忱地从事 Java 之前，我真切地努力使自己公平地对待 Sun 和 Microsoft 两大巨人之间的技术和语言战争。我从没确定对于任何一方的偏见。Java 毫无疑问有它的实力，并且在.NET 之前，因为许多原因它已经成为我最喜欢的面向对象语言。我喜欢它的语法、线程模型、类型系统，尤其是基类库。从 C/C++和 Visual Basic 走过来的人，接着无疑是使用 Java。你不用再在 C 运行时和扩展类库的无数重复功能性当中挣扎了。但我们很快发现 Java 并不是完美的。“Write Once，Run everywhere（一次编写，随处运行）”并不像最初承诺的那么好。开发一个自定义的 Java GUI 并不是那么简单，并且对于代码的生成有太多厂商选择。面向对象也成为了主流，对设计原则的需求使用它。

Microsoft 平台同样有它自身的实力。Visual Basic 为构建丰富的 GUI 应用程序，提供了快速的开发周期。随着 Visual Basic 的成熟，几乎不用为开发专门代码而进行调整，例如 COM

组件使用一种较低级的语言如 C++进行开发。Visual Basic（VB）可以为你提供这些，而不需要做太多的事情，当然下列情况除外，如你是在编写一个商业产品，需要非常专业化的行为或一个非常小的.DLL。使用 VB 可以在几个星期内让你的商业应用成为产品，而不需要花几个月时间。任何在 Java 或 VB 上从事过 GUI 开发的人，都能明白我所谈论的东西。从开发的角度来看，我喜欢 Java，但将使用 AWT、Swing 或我所使用的任何第三方类库的 GUI 应用集成到一起是一件痛苦的事情。随着 Java 的成熟，第三方会使得它更容易，但你会发现你淹没在太多的选择中了。选择某一个，你便冒着风险采取了一种可能没有支持或没有市场的技术。这似乎总是一个问题，因为这门语言的创建者并没有为所创建的语言，提供一个最广为使用的开发环境。这是 Java 的实际情况。

以我拙见，甚至在我加入 Microsoft 之前，我就感觉到它已经构造了一些最好的集成开发环境（IDE）。当然，有一些更强大的编辑器为开发人员提供了一种无所不知的感觉。然而对于大多数人来说，你想要的是一个设计良好的、紧密集成的和对用户友好的 IDE，它不仅要可靠而且很平常。在 UNIX 世界，使得 vi 如此盛行的原因也正是 Visual Studio 如此吸引人的原因——它随处都可以找得到。如果在 Windows 2000 上开发 Visual Basic 或 C++应用程序，Visual Studio 通常是你所使用的工具。而在 Java 世界却不是这样。你可以使用 BEA、Borland、Sun 和 IBM 等厂商的编辑器。作为一个 Java 顾问，我需要尝试所有主要厂商的编辑器，因为似乎没有一个编辑器是真正杰出的。对于资本主义制度来说，这种“开放市场”观念似乎是很有吸引力的，但对于工程学来说，它创建了一种回旋（convolution）。

我曾经对自己说，“要是有一个平台，它结合了 Java 和 Visual Basic 的优点，又能提供 C++的强大功能该多好啊。”这就是.NET（尤其是 C#）。我希望当我这么说的时候，并没有使得太多人脸红而想放下这本书。但对于双边的语言战争，我希望我有权利作出这样一个选择。.NET 框架包含了在 Java 中我喜欢的每一样东西，甚至更多。你不用再被迫选择一门语言，并且遭受实际开发中类库不一致的痛苦，更不用说我将要在本书中提到的为数众多的其他好处。作为第一代产品（.NET 框架 1.1 在本书中仍然算第一代产品），我对我们今天所拥有的功能性（functionality）感到惊讶，而这不过是冰山之一角。有希望的是，在随后的数年，.NET 将会继续改进，成为将来的跨平台开发框架，并为这些环境带来好处。考虑在.NET 框架 1.1 版本和 Visual Studio .NET 中已经提供的相当数量的功能性，我可以想像得到今后的两年.NET 平台将会更进一步。Microsoft 生产这个不同开发环境的速度是惊人的，使我想起在如此短的时间内 Internet Explorer 成为市面上重要的浏览器。这不只是因为良好的市场，还有良好的交付。如果你是一个 Java 开发人员，将.NET 当成一个重要的转变；我想你将不会后悔。

我开始这本书正是在.NET Beta 2 发布之后，这时.NET 类库和其运行时的大部分主要特征都已经完成了。一年之后，当你拿起本书的第一版时.NET 框架版本应该发布和公开了。令人感到高兴的是，在那时并没有太大的改变。只在原始发布的版本和 Beta 2 的稳定性方面谈论了许多。实际上，如果你刚好有一个旧版的 Beta 2 版本，你可以随意地测试本书中引用的代码。它应该是可以编译并运行的。毕竟，这是一本关于设计与架构的最佳实践的书。在那个时候，在一个未定的环境中编写代码是很难的，但更难的是试图开发出最好的实践。问

题在于编写的代码对于这个技术是不可知的。使用.NET 框架 1.1 对这一点做出严格的测试，我很高兴地发现一切都是兼容的。但本书最终出版时，.NET1.0 已经发布一年多了，并且成千上万的开发人员对于其内容、示例代码和使用的原则是哗声一片。我所做的事情是在 Beta 2 之上进行的。

我希望可以帮助你进入这个远离 Microsoft 传统开发平台的不同世界。Microsoft 不仅引入了等同的一套新语言，而且还提供了在 Java 世界吸引 Java 开发人员的相同的元素。Microsoft 现在为知道一些面向对象但没有真正经历过的 Visual Basic 开发人员，真正地提供了面向对象的特征。为.NET 的新味道干杯（bottom up）！我相信每个人都会对这道菜感到满意的。

本书布局

我把这本书分成了三个主要部分。第一部分介绍.NET 和设计的元素以及开始编写良好应用程序所需要的架构。包括大多数框架需要的较低级的管道组件（plumbing component）。这些“框架模式和实践”适用于.NET 开发的所有方面，包括异常处理和日志。第二部分强调了在一个架构层的一些特别的该做和不做的事情，将模式分类到三个层当中：表示层（presentation tier）、中间层（middle tier）和存储层（persistence tier）。这个突出的部分不仅是最好的实践和实现模式，也是更好的技术无关的（technology-agnostic）架构和设计模式。最后一章是自成一体的，因为它包括了更高级的模式，涵盖一些主题，例如异步行为、复杂线程和缓存。许多框架需要这些模式，但它们并不被认为是一种需求，因而，给出它们自己的一章。许多模式都符合建立“四件套（gang of four）”的模式，可以添加到你的设计工具区。

在第一章，我将给你一些简洁的技术概览，它们是 XML Web 服务、.NET 世界的面向对象，以及为设计解决方式的新手们提供的一些模式定义。贯穿本书，我将介绍一些新的.NET 主题，这些特征中有一些可能对影响你的设计和对内容的理解的其他框架是惟一的。我也将开始在由.NET 提供的这些服务中埋下伏笔，它们将在本书后面更重要的技术模式中产生影响。

第一部分：用.NET 构建框架

第一章——新的框架、新的模型、新的度量

在这里我简单地回顾了一下模式、模式的分类和它们在.NET 框架中的应用。我将为过去没有使用模式工作过的人介绍一下模式，并且提供它们如何应用的处理。我还将构造那些将所有来自 Microsoft 的开发版本中分离出来的内容，例如，与 XML Web 服务的结合。

本章还将简单回顾用作每个表示实践例证的.NET 元素。尽管许多模式是技术无关的，但对于实现和架构模式情况却不是如此。在这里我将简单回顾掌握更多相关技术模式所需的.NET 元素。这对于那些不熟悉.NET 基类库（BCL）的人来说更像是一种“应急课程”。

最后本章更深入地介绍了模式、模式分类和它们的历史。本章对于习惯于更线性编程风格的（例如 Visual Basic、Active Server Pages 等）程序员来说尤其重要，他们可能习惯于面向组

件 COM 的设计，但不习惯于 OO 设计。我将解释实现模式是什么，以及它们为什么不同于架构和设计模式。我在这里将通过表示一套最好的实践和“命名的”实现模式，来谈论好的和不好的.NET 设计、架构和开发。

我还将在讲述的过程中提供一些.NET 的基本要素，以便不熟悉的人知道在即将出现的模式图表中使用的“类库象形符号（library hieroglyphics）”。这些“基本要素”只是一些“小指南”，因为有非常多的材料更好更深地涵盖了这些主题背景。

第二章——框架模式：异常处理、日志记录和跟踪

由于其一般的适用性和重要性，在本章中将会涵盖像调试、错误处理和日志这样的主题。这些内容在本书的前面部分，因为我觉得它是一个良好而健壮的应用程序的最重要的元素之一。如果要开始任何良好框架的设计，你可以以这些实践和实现模式开始，并在你的应用程序中设置这个初始“管道（plumbing）”。在这一点上，你应该熟悉.NET 技术和一般的模式。这整个主题就是我所说的“框架模式（framework patterns）”,因为它们组成了框架的主干，并且对于任何应用来说都是非常重要的。它们包括异常处理、日志、跟踪和许多其他没有分类到表示层、中间层和存储层中的实践。虽然这些代码大多数都可以放在中间层，但对于所有的层来说都很重要，所以将它们如此分类。

其原则和模式的细节如下：

异常处理（Exception Handling）——.NET 下的结构化错误处理

异常日志（Exception Logging）——事件日志和检测实践

异常链（Exception Chaining）——将错误通过“冒泡”从一个层传递到另一个层

构建一个异常基类（Building a Base Exception Class）——为你的错误处理创建一个基类

跟踪和跟踪监听（Tracing and Trace Listening）——集中并简化你的检测方法

错误和调用堆栈（Error and Call Stacks）——表示并丰富你的错误表示信息

什么时候、在哪里以及如何写日志（When,Where and How to Log）——.NET 下的启发式错误处理

SOAP 异常和 SOAP 错误（SOAP Exception and SOAP Faults）——Web 服务环境中的错误

异常处理互操作（Interop Exception Handling）——分布式 EAI 环境中的错误

下面相关的实现模式将会详细地描述：

远程跟踪（Retomting Tracer）——跨机器边界的跟踪和错误处理

自定义 Soap 异常处理（Custom Soap Exception Handler）——Web 服务环境中的错误处理

系统异常包装（System Exception Wrapper）——处理.NET 系统异常

Soap 错误生成器（Soap Fault Builder）——产生通过 Web 服务客户端查看的错误

本书的这个部分主要是讲一个具体应用层的设计和架构模式的分类。有一些只是旧有模式的新的揉合（twist），而另一些却是独有的。有一些模式是技术无关的（technology-agnostic），而其他一些采用了.NET 所独有的框架特征，因此是与.NET 技术紧密相关的。

第二部分：创建框架的层

第三章——表示层模式

到这里，你应该已经熟悉.NET、Web 服务和模式了。现在我将开始讲述开发的所有前前后后的实现实践。它包括“瘦”和“胖”客户端实现实践，例如屏幕刷新、GUI 线程、自定义控件显示以及接口模板原则。甚至将会讨论更高级的主题，例如异步 Web 服务调用和客户端线程。如果你只会使用到中间层组件，那么可以随意地略过本章。

如你所推测，本章将表示层的模式进行了分类。它们包括：

通知线程管理器（Notifying Thread Manager）——创建一个线程，并且通知一个 Windows 窗体

轮询线程管理器（pollable Thread Manager）——创建一个线程，并且周期性地检查状态

多重同步线程管理器（MultiSync Thread Manager）——联合通知和轮询线程管理器

错误交叉引用生成器（Error Cross-Reference Generator）——异常处理过程中使用的错误 ID 生成器

Web Form 模板（Web Form Template）——用于快速结构布局的模板基类

神奇的驱动器（Stunt Driver）——用于测试组件的通用接口

动态程序集加载（Dynamic Assembly Loader）——动态加载和缓存驱动器

第四章——中间层模式

在本章中，我涵盖了六个中间层或“商业”层模式，它们可被真正地应用到任意级别。这个分类包含所有这些通常不被认为是“高级”的，但又不被认为与层和层、表示层或存储层（数据）相关的模式。对于那些需要设计必须表示单一入口点架构的人来说，有时候这样开始是有意义的。这也可以被认为是一个设计分布式包装器的机会，它可以使得你的所有中间件组件的可访问性，对于外部世界来说是安全的。创建这些模式以及框架模式对于任何架构来说，都是我首先要推荐的。第二章提供了构建一个真正动态和灵活的系统所需要的核心元素。这也将从基于将来的用户需求的改变中隔离出你的支持，并且可以让你的开发人员集中精力于业务规则定义而不是“管道式”问题。

这里所谈到的模式是：

链式服务工厂（Chained Service Factory）——为 Web 服务创建一个单一的入口点

非链式服务工厂（Unchained Service Factory）——Web 服务的一个晚期绑定入口点

产品管理器（Product Manager）——以托管的方式处理非托管代码

服务外观（Service Façade）——委托来自于 Web 服务的复杂逻辑

抽象包模式（Abstract Packet）——处理及传递复杂参数集

包翻译器（Packet Translator）——翻译这些复杂参数集

第五章——持久层模式

在这里我将重点讲述存储层架构和实现模式。它包括数据库访问、流处理和基于 XML 模式的 I/O。我将讲述.NET 的某些最强大的 XML 数据特征，并且向你展示如何利用它们。它包

括充分利用数据集、XML 模式、自动数据层代码生成、动态数据建模、强类型模式和 XML 的灵活性以及如何平衡它。

架构模式涵盖如下内容：

多模型（Poly Model）——使用 XML 模式提供一个“动态物理数据模型”

多模型工厂（Poly Model Factory）——对多模型应用工厂方法

模式字段（Schema Field）——存储模式、获取模式和模式管理

模式索引器（Schema Indexer）——使用模式和多模型构建动态索引器

抽象模式（Abstract Schema）——在 XML 模式世界应用抽象包模式

第六章——过程模式：把.NET 模式应用到商业产品中

在本章，我将谈论我所领导设计和架构的，一个名为 ProductX 的真实世界商业应用。我将会提供一些应用这个金融服务应用的一些实践，以及如何从基于传统 C++/COM 的平台移植到目标.NET 平台的过程。我还将描述在本书中所强调的模式是如何在被应用到实现的。在这个产品的开发周期中，可以说，我“吃了自己的狗食（ate my own dog food）”，在一个生动的商业应用中使用了本书中的大多数模式。

在这个商业应用程序中我所使用的来自本书的模式如下：

多模型（Poly Model）——应用多模型复合模式并将它集成到产品数据库

非链式服务工厂（Unchained Service Factory）——对于所有外部访问，由 ProductX 所使用的 Web 服务，采用晚期绑定单一入口点

产品管理器（Product Manager）——通过将所有金融组件托管在商业框架中，提供业务框架

服务外观（Service Facade）——从 Web 服务委托复杂逻辑，并且为金融服务定义框架特征集

第七章——高级模式

本章谈论的就是我所说的高级模式。这些模式可以被应用到任何级别，它们对于应用程序来说不是核心的，但却非常有用。当性能成为问题的时候，就可以应用这些模式，例如附加缓存和异步过程。增加这些模式可以增强设计，但只应该非常谨慎地在特定情形下使用。如果不正确地实现和应用任何这些模式，将会缩减它们的好处，并且失去任何应用它们的理由。

这些高级模式包括：

高级缓存（Abstract Cache）——使用缓存对象缓存框架

Web 服务接口（Web Service Interface）——基于接口的 Web 服务

松耦合事务处理器（Loosely Coupled Transactor）——高级异步商业事务处理

LCT Server——基于服务器的松耦合事务处理器

LCT Client——基于客户端的松耦合事务处理器

此为试读，需要完整PDF请访问：www.ertongbook.com

感谢

一本书的产生不只是作者编写它。它是某些观念的创作努力、与白色书写板的对话、艺术作品、编辑学，最重要的是时间的有机结合。有许多人有意或无意地参与了这 18 个月的工程。最容易忘记的人就是这些为进行这本书而贡献了无形资产（例如时间和耐心）的人。我要感谢那些帮助我从最初模糊的想法到设计、到编码、到内容再到产品的走过来的人们。他们不只是专心地准备了这里所使用的材料，并且也让我有时间在这个工程中使用它。还有像 Brett Walker 这样的人，给与了我在这本书上的时间，而这些时间本来可以用于立即获取更大利润。因此，我要感谢所有这些为从想法到产品做出贡献的人，按做出贡献的时间顺序排列。

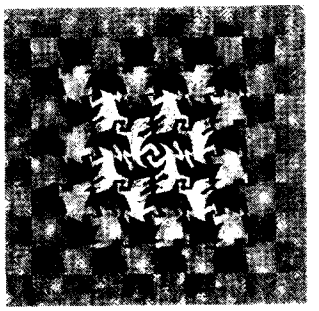
感谢 Paul Becker 花时间倾听我对这本书的想法，以及评价它的前提价值。特别感谢 John Neidhart 在 Addison 加速了这个工程，信任地（用了很少的时间考察）接纳了它的观念，并为我提供了更实用的交付方式。感谢产品组的所有成员（Kyle Howard、Marti Jones 和 Patti Guerrieri），为我写这本书尽可能减少痛苦所作的努力。

感谢 Brett Walker 使我在他的公司与他--同工作时，有时间继续写这本书。我被赋予使用这本书中的原则来设计产品应用的能力，并且 Brett 为我提供了最好的测试平台。没有这种合作和共同的努力，这本书中许多原则可能陷入理论的深坑。Brett Walker、David Mytchak、James Williams 在许多模式方面，给予了我最重要的反馈和创造性鼓舞。没有 David Mytchak 的帮助，自早期的那几个月我的想法将不会成为现实。没有 David 的想法和 James 的代码，我将无法跨越某些传统的思想，并创建第五章。在这里描述的多模型便是源自你们的“原始想法”。

我还要感谢在 2003 年早春加入公司之前，Microsoft 所有关心这本书的人。Microsoft 的每个人都对这项工程如此支持，并帮助我提炼本书的信息。在这里所使用的材料，非常漂亮地符合 Microsoft 的约定标记和它的自己的架构原则。它真正地向你展示了好的设计就是好的设计，无论你怎么打击它。特别感谢 Gulf Coast 开发组的成员（我是其中的一员）：Ed Draper、J Sawyer、Michael Lane Thomas 和 John Opalko，他们帮助我在他们的现场幻灯片的最后几分钟，加入了这本书的存在的消息。

感谢我的妻子 Hilari。到这本书出版时为止，我们刚刚度蜜月回来。我们现在可以回顾并享受我们劳动的成果。我们现在可以欣赏所有花费在我的电脑前的周末和周日夜晚，那时她不得不将食物悄悄地放在我的门后。非常感谢她对于这项工程的耐心和理解。感谢我的母亲所提供的所有的“小的协助”，让我节省了大量的时间，可以进行这本书。

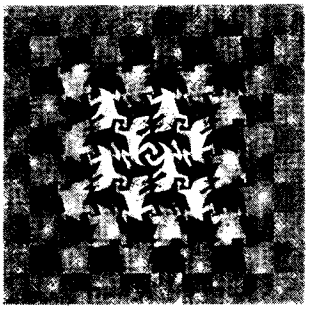
最后，我要感谢 Brian Eshelman，他是第三章所有内容的主要贡献者，这使得我节省了一些时光。这里是他的想法、设计和代码。没有他们，我的最后期限可能危难重重——非常感谢你，Brian，为这本书所作的一部分。你是一个杰出的设计师、开发者和朋友。



目录

第一部分

用.NET 构建框架



第一章

新的框架、新的模型、新的度量

请打住，这不是另一种语言！

回想 Microsfot 首次宣布符合其.NET 框架的一门新语言时，我表示很大的怀疑。这门语言就是 C#（发音为 C-Sharp），它也是本书示例所采用的主要语言。不过，这并不表示选择其他.NET 所支持编程语言（例如 VB.NET、J#、托管 C++等）的开发人员，就不能参考这些材料。事实上，本书的主要目标是尽可能提供.NET 实践的一种语言无关的观点。这种方式与 Microsoft 在.NET 中通过其运行时和其语言规范所实现的方式相同。本书的重点在于框架、它的设计元素以及从中提炼出的最佳实践和模式。因为我的个人喜好，所以，示例都是用 C#编写的，但这些观点中的每一个都可以使用 VB.NET、J#或者任何.NET 支持的其他编程语言实现。这同 Microsoft 架构.NET 框架的方式密切相关。这确实是这个产品最强大并且独特的特征。开发人员从.NET 类库中学习所有编程语言的公共元素，以及本书所教授的公共的设计原理，这时，他或她不必在乎具体的编程语法，根据自己的喜好进行选择。使用.NET，再多的公共设计模式（例如 Strategy 模式）现在都可以用多种编程语言实现。图 1.1 使用<< >>来指明实现 Strategy 模式各个实体的编程语言。这只是向设计者提供一个简单的演示，它展示了跨语言继承的灵活性。即使被普遍实现的模式（例如 Strategy 模式），在.NET 的辅助下，也可以推陈出新，变换花样。

如果你恰好处在选择哪种.NET 语言的十字路口，那么，要是我的话将会选择 C#。理由来自 Visual Basic 程序员方面的感触，他们普遍感到学习 VB.NET 存在技术上的鸿沟。因为这个框架是如此不同，它的学习曲线不是在于语法，而是在于环境。对于那些不熟悉面向对象技术的开发者，甚至是 VB.NET 也将是从 Visual Basic6.0 世界的一个相当大的飞跃。如果你想学习这个新系统，那么，我建议你选择一门可以找到最多示例的语言。当然，我相信那就是 C#，尽管 Microsoft 在其文档和在线帮助中同时展示了 VB.NET 和 C#的示例，并且为此做了大量的工作。即使.NET 将支持遵守公共语言基础架构（CLI）的任何语言，但是，C#是.NET 的编程

此为试读，需要完整PDF请访问：www.ertongbook.com