计 算 机 科 学 丛 书

设计模式

可复用面向对象软件的基础

Design Patterns

Elements of Reusable Object-Oriented Software

(美) Erich Gamma Richard Helm Ralph Johnson John Vlissides 著

李英军 马晓星 蔡敏 刘建中 等译

吕 建 审校

机械工业出版社 China Machine Press

Addison-Wesley

计算机科学丛书

设计模式

可复用面向对象软件的基础

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
著

李英军 马晓星 蔡敏 刘建中 等译
吕建 审校

机械工业出版社
China Machine Press

本书结合设计实例从面向对象的设计中精选出23个设计模式，总结了面向对象设计中最有价值的经验，并且用简洁可复用的形式表达出来。本书分类描述了一组设计良好、表达清楚的软件设计模式，这些模式在实用环境下特别有用。本书适合大学计算机专业的学生、研究生及相关人员参考。

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software.

Original edition copyright © 1995 by Addison Wesley Longman, Inc.

Chinese edition published by arrangement with Addison Wesley Longman, Inc. All rights reserved.

本书中文简体字版由美国Addison Wesley 公司授权机械工业出版社独家出版。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。

版权所有，侵权必究。

本书版权登记号：图字：01-1999-2859

图书在版编目（CIP）数据

设计模式：可复用面向对象软件的基础 /伽玛等著；李英军等译. - 北京：机械工业出版社，2000.9

(计算机科学丛书)

书名原文：Design Patterns: Elements of Reusable Object-Oriented Software

ISBN 7-111-07575-7

I. 设… II. ①伽… ②李… III. 面向对象语言-程序设计 IV.TP312

中国版本图书馆CIP数据核字（2000）第39439号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：周 桦

北京市密云县印刷厂印刷 · 新华书店北京发行所发行

2000年9月第1版 · 2001年3月第3次印刷

787mm × 1092 mm 1/16 · 16.5印张

印数：13 001-21 000册

定价：35.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

部分学者对《设计模式：可复用面向对象软件的基础》的赞誉

“这本书是我长期以来所读过的写得最好、最富洞察力的书籍之一……该书不是泛泛而论，而是结合实例，以最佳的方式确立了模式的合法地位。”

-- Stan Lippman, *C++ Report*

“..... Gamma, Helm, Johnson和Vlissides的这本新书将对软件设计领域产生重要且深远的影响。因为《设计模式》这本书将自己定位于面向对象软件技术，恐怕面向对象圈子以外的设计者也许会忽视它的价值，但这将是一件憾事。该书将能使从事软件设计的每个人从中获益。所有软件设计者都在使用模式，而更好地理解这种对我们工作的可复用的抽象只会使我们做得更好。”

-- Tom DeMarco, *IEEE Software*

“总的来讲，这本书表达了一种极有价值的东西，对软件设计领域有着独特的贡献，因为它捕获了面向对象设计的有价值的经验，并且用简洁可复用的形式表达出来。它将成为我在寻找面向对象设计思想过程中经常翻阅的一本书；这正是复用的真实含义所在，不是吗？”

-- Sanjiv Gossain, *Journal of Object-Oriented Programming*

“这本众人期待的书达到了预期的全部效果。“模式”的讲法来自一位建筑师的书，该书云集了经过时间考验的可用设计。作者从多年的面向对象设计经验中精选出了23个模式，这构成了该书的精华部分，每一个精益求精的优秀程序员都应拥有这本《设计模式》。”

-- Larry O’Brien, *Software Development*

“《设计模式》在实用环境下特别有用，因为它分类描述了一组设计良好、表达清楚的面向对象软件设计模式。整个设计模式领域还很新，本书的四位作者也许已占据了在这方面造诣最深的专家中的半数，因而他们定义模式的方式可以作为后来者的榜样。如果要知道怎样恰当定义和描述设计模式，我们应该可以从他们的专业知识中获得启发。”

-- Steve Bilow, *Journal of Object-Oriented Programming*

“《设计模式》是一本深刻有力的书。在花费了相当的时间研究该书后，绝大部分C++程序员都能够使用模式构造出更好的软件。本书发挥了一种智能杠杆作用：提供具体工具帮助我们进行思维并有效地表达我们自己。它也许能从根本上改变你对程序设计的看法。”

-- Tom Cargill, *C++ Report*

序　言

所有结构良好的面向对象软件体系结构中都包含了许多模式。实际上，当我评估一个面向对象系统的质量时，所使用的方法之一就是要判断系统的设计者是否强调了对象之间的公共协同关系。在系统开发阶段强调这种机制的优势在于，它能使所生成的系统体系结构更加精巧、简洁和易于理解，其程度远远超过了未使用模式的体系结构。

模式在构造复杂系统时的重要性早已在其他领域中被认可。特别地，Christopher Alexander和他的同事们可能最先将模式语言（pattern language）应用于城市建筑领域，他的思想和其他人的贡献已经根植于面向对象软件界。简而言之，软件领域中的设计模式为开发人员提供了一种使用专家设计经验的有效途径。

在本书中，Erich Gamma、Richard Helm、Ralph Johnson和John Vlissides介绍了设计模式的原理，并且对这些设计模式进行了分类描述。因此，该书做出了两个重要的贡献：首先，它展示了模式在建造复杂系统过程中所处的角色；其次，它为如何引用一组精心设计的模式提供了一个实用方法，以帮助实际开发者针对特定应用问题使用适当的模式进行设计。

我曾荣幸地有机会与本书的部分作者一同进行体系结构设计工作，从他们身上我学到了许多东西，并相信通过阅读该书你同样也会受益匪浅。

Rational 软件公司首席科学家
Grady Booch

试读结束：需要全本请在线购买：　www.ertongbook.com

前　言

本书并不是一本介绍面向对象技术或设计的书，目前已有不少好书介绍面向对象技术或设计。本书假设你至少已经比较熟悉一种面向对象编程语言，并且有一定的面向对象设计经验。当我们提及“类型”和“多态”，或“接口”继承与“实现”继承的关系时，你应该对这些概念了然于胸，而不必迫不及待地翻阅手头的字典。

另外，这也不是一篇高级专题技术论文，而是一本关于**设计模式**的书，它描述了在面向对象软件设计过程中针对特定问题的简洁而优雅的解决方案。设计模式捕获了随时间进化与发展的问题的求解方法，因此它们并不是人们从一开始就采用的设计方案。它们反映了不为人知的重新设计和重新编码的成果，而这些都来自软件开发者为了设计出灵活可复用的软件而长时间进行的艰苦努力。设计模式捕获了这些解决方案，并用简洁易用的方式表达出来。

设计模式并不要求使用独特的语言特性，也不采用那些足以使你的朋友或老板大吃一惊的神奇的编程技巧。所有的模式均可以用标准的面向对象语言实现，这也许有时会比特殊的解法多费一些功夫，但是为了增加软件的灵活性和可复用性，多做些工作是值得的。

一旦你理解了设计模式并且有了一种“Aha！”（而不是“Huh？”）的应用经验和体验后，你将用一种非同寻常的方式思考面向对象设计。你将拥有一种深刻的洞察力，以帮助你设计出更加灵活的、模块化的、可复用的和易理解的软件——这也是你为何着迷于面向对象技术的源动力，不是吗？

当然还有一些提示和鼓励：第一次阅读此书时你可能不会完全理解它，但不必着急，我们在起初编写这本书时也没有完全理解它们！请记住，这不是一本读完一遍就可以束之高阁的书。我们希望你在软件设计过程中反复参阅此书，以获取设计灵感。

我们并不认为这组设计模式是完整的和一成不变的，它只是我们目前对设计的思考的记录。因此我们欢迎广大读者的批评与指正，无论从书中采用的实例、参考，还是我们遗漏的已知应用，或应该包含的设计模式等方面。你可以通过Addison-Wesley写信给我们，或发送电子邮件到：design-patterns@cs.uiuc.edu。你还可以发送邮件“send design pattern source”到design-patterns-source@cs.uiuc.edu获取书中的示例代码部分的源代码。

另外我们有一个专门的网页报道最新的消息与更新：

http://st-www.cs.uiuc.edu/users/patterns/DPBook/DPBook.html.

E.G. 于加州Mountain View

R.H. 于蒙特利尔

R.J. 于伊利诺Urbana

J.V. 于纽约 Hawthorne

1994年8月

读者指南

本书包括两个主要部分，第一部分（第1章和第2章）介绍了什么是设计模式以及它如何帮助你设计面向对象的软件系统。该部分包含了一个设计案例研究，展示了如何将设计模式应用于实际工作。第二部分（第3、4、5章）则是实际设计模式的分类描述。

模式的分类描述构成了本书的主要部分，书中的章节根据模式的性质将其划分为三种类型：创建型（creational），结构型（structural）和行为型（behavioral）。可以从多个角度使用这个模式分类描述，例如，你可以从头至尾地阅读每一个模式，也可以随机浏览其中的任何一个模式。另外一种方法是研究其中的一章，这将有助于理解原本密切关联的模式如何相互区分。

模式描述中的交叉引用将给你提供寻找其他相关模式的逻辑路径，它将帮助你看清楚模式是如何相互关联的、一个模式怎样与其他模式进行组合、以及哪些模式能在一起工作。图1-1将用图示方法展现这种关系。

阅读模式分类描述的另一种方法是问题导向法，你可以翻到书中的第1.6节查找有关设计可复用的面向对象系统过程中经常见到的问题，然后阅读解决这些问题的有关模式。有些读者首先通读模式分类描述，然后运用问题导向的方法将模式应用于他们的项目之中。

如果你不是一个有经验的面向对象设计人员，我们建议你应该从那些最简单常用的模式出发：

- Abstract Factory(3.1)
- Adapter(4.1)
- Composite(4.3)
- Decorator(4.4)
- Factory Method(3.3)
- Observer(5.7)
- Strategy(5.9)
- Template Method(5.10)

很难找到一个面向对象软件系统，它没有使用书中描述的若干模式。许多大型软件系统几乎用到了所有的这些模式。上述这组模式将有助于你进一步理解设计模式本身及一般意义下的优秀的面向对象设计。

目　录

第1章 引 言

设计面向对象软件比较困难，而设计可复用的面向对象软件就更加困难。你必须找到相关的对象，以适当的粒度将它们归类，再定义类的接口和继承层次，建立对象之间的基本关系。你的设计应该对手头的问题有针对性，同时对将来的问题和需求也要有足够的通用性。你也希望避免重复设计或尽可能少做重复设计。有经验的面向对象设计者会告诉你，要一下子就得到复用性和灵活性好的设计，即使不是不可能的至少也是非常困难的。一个设计在最终完成之前常要被复用好几次，而且每一次都有所修改。

有经验的面向对象设计者的确能做出良好的设计，而新手则面对众多选择无从下手，总是求助于以前使用过的非面向对象技术。新手需要花费较长时间领会良好的面向对象设计是怎么回事。有经验的设计者显然知道一些新手所不知道的东西，这又是什么呢？

内行的设计者知道：不是解决任何问题都要从头做起。他们更愿意复用以前使用过的解决方案。当找到一个好的解决方案，他们会一遍又一遍地使用。这些经验是他们成为内行的部分原因。因此，你会在许多面向对象系统中看到类和相互通信的对象（communicating object）的重复模式。这些模式解决特定的设计问题，使面向对象设计更灵活、优雅，最终复用性更好。它们帮助设计者将新的设计建立在以往工作的基础上，复用以往成功的设计方案。一个熟悉这些模式的设计者不需要再去发现它们，而能够立即将它们应用于设计问题中。

以下类比可以帮助说明这一点。小说家和剧本作家很少从头开始设计剧情。他们总是沿袭一些业已存在的模式，像“悲剧性英雄”模式(《麦克白》、《哈姆雷特》等)或“浪漫小说”模式(存在着无数浪漫小说)。同样地，面向对象设计员也沿袭一些模式，像“用对象表示状态”和“修饰对象以便于你能容易地添加/删除属性”等。一旦懂得了模式，许多设计决策自然而然就产生了。

我们都知道设计经验的重要价值。你曾经多少次有过这种感觉——你已经解决过了一个问题但就是不能确切知道是在什么地方或怎么解决的？如果你能记起以前问题的细节和怎么解决它的，你就可以复用以前的经验而不需要重新发现它。然而，我们并没有很好记录下可供他人使用的软件设计经验。

这本书的目的就是将面向对象软件的设计经验作为**设计模式**记录下来。每一个设计模式系统地命名、解释和评价了面向对象系统中一个重要的和重复出现的设计。我们的目标是将设计经验以人们能够有效利用的形式记录下来。鉴于此目的，我们编写了一些最重要的设计模式，并以编目分类的形式将它们展现出来。

设计模式使人们可以更加简单方便地复用成功的设计和体系结构。将已证实的技术表述成设计模式也会使新系统开发者更加容易理解其设计思路。设计模式帮助你做出有利于系统复用的选择，避免设计损害了系统复用性。通过提供一个显式类和对象作用关系以及它们之间潜在联系的说明规范，设计模式甚至能够提高已有系统的文档管理和系统维护的有效性。简而言之，设计模式可以帮助设计者更快更好地完成系统设计。

本书中涉及的设计模式并不描述新的或未经证实的设计，我们只收录那些在不同系统中

多次使用过的成功设计。这些设计的绝大部分以往并无文档记录，它们或是来源于面向对象设计者圈子里的非正式交流，或是来源于某些成功的面向对象系统的某些部分，但对设计新手来说，这些东西是很难学得到的。尽管这些设计不包含新的思路，但我们用一种新的、便于理解的方式将其展现给读者，即：具有统一格式的、已分类编目的若干组设计模式。

尽管该书涉及较多的内容，但书中讨论的设计模式仅仅包含了一个设计行家所知道的部分。书中没有讨论与并发或分布式或实时程序设计有关的模式，也没有收录面向特定应用领域的模式。本书并不准备告诉你怎样构造用户界面、怎样写设备驱动程序或怎样使用面向对象数据库，这些方面都有自己的模式，将这些模式分类编目也是件很有意义的事。

1.1 什么是设计模式

Christopher Alexander说过："每一个模式描述了一个在我们周围不断重复发生的问题，以及该问题的解决方案的核心。这样，你就能一次又一次地使用该方案而不必做重复劳动"[AIS+77，第10页]。尽管Alexander所指的是城市和建筑模式，但他的思想也同样适用于面向对象设计模式，只是在面向对象的解决方案里，我们用对象和接口代替了墙壁和门窗。两类模式的核心都在于提供了相关问题的解决方案。

一般而言，一个模式有四个基本要素：

1. **模式名称**（pattern name） 一个助记名，它用一两个词来描述模式的问题、解决方案和效果。命名一个新的模式增加了我们的设计词汇。设计模式允许我们在较高的抽象层次上进行设计。基于一个模式词汇表，我们自己以及同事之间就可以讨论模式并在编写文档时使用它们。模式名可以帮助我们思考，便于我们与其他人交流设计思想及设计结果。找到恰当的模式名也是我们设计模式编目工作的难点之一。

2. **问题**(problem) 描述了应该在何时使用模式。它解释了设计问题和问题存在的前因后果，它可能描述了特定的设计问题，如怎样用对象表示算法等。也可能描述了导致不灵活设计的类或对象结构。有时候，问题部分会包括使用模式必须满足的一系列先决条件。

3. **解决方案**(solution) 描述了设计的组成成分，它们之间的相互关系及各自的职责和协作方式。因为模式就像一个模板，可应用于多种不同场合，所以解决方案并不描述一个特定而具体的设计或实现，而是提供设计问题的抽象描述和怎样用一个具有一般意义的元素组合（类或对象组合）来解决这个问题。

4. **效果**(consequences) 描述了模式应用的效果及使用模式应权衡的问题。尽管我们描述设计决策时，并不总提到模式效果，但它们对于评价设计选择和理解使用模式的代价及好处具有重要意义。软件效果大多关注对时间和空间的衡量，它们也表述了语言和实现问题。因为复用是面向对象设计的要素之一，所以模式效果包括它对系统的灵活性、扩充性或可移植性的影响，显式地列出这些效果对理解和评价这些模式很有帮助。

出发点的不同会产生对什么是模式和什么不是模式的理解不同。一个人的模式对另一个人来说可能只是基本构造部件。本书中我们将在一定的抽象层次上讨论模式。《设计模式》并不描述链表和hash表那样的设计，尽管它们可以用类来封装，也可复用；也不包括那些复杂的、特定领域内的对整个应用或子系统的设计。本书中的设计模式是对被用来在特定场景下解决一般设计问题的类和相互通信的对象的描述。

一个设计模式命名、抽象和确定了一个通用设计结构的主要方面，这些设计结构能被用

试读结束：需要全本请在线购买： www.ertongbook.com

来构造可复用的面向对象设计。设计模式确定了所包含的类和实例，它们的角色、协作方式以及职责分配。每一个设计模式都集中于一个特定的面向对象设计问题或设计要点，描述了什么时候使用它，在另一些设计约束条件下是否还能使用，以及使用的效果和如何取舍。既然我们最终要实现设计，设计模式还提供了C++ 和Smalltalk示例代码来阐明其实现。

虽然设计模式描述的是面向对象设计，但它们都基于实际的解决方案，这些方案的实现语言是Smalltalk 和C++等主流面向对象编程语言，而不是过程式语言(Pascal、C、Ada)或更具动态特性的面向对象语言(CLOS、Dylan、Self)。我们从实用角度出发选择了Smalltalk 和C++，因为在这些语言的使用上，我们积累了许多经验，况且它们也变得越来越流行。

程序设计语言的选择非常重要，它将影响人们理解问题的出发点。我们的设计模式采用了Smalltalk 和C++层的语言特性，这个选择实际上决定了哪些机制可以方便地实现，而哪些则不能。若我们采用过程式语言，可能就要包括诸如“继承”、“封装”和“多态”的设计模式。相应地，一些特殊的面向对象语言可以直接支持我们的某些模式，例如：CLOS支持多方法（multi-method）概念，这就减少了Visitor模式的必要性。事实上，Smalltalk和C++已有足够的差别来说明对某些模式一种语言比另一种语言表述起来更容易一些(参见5.4节Iterator 模式)。

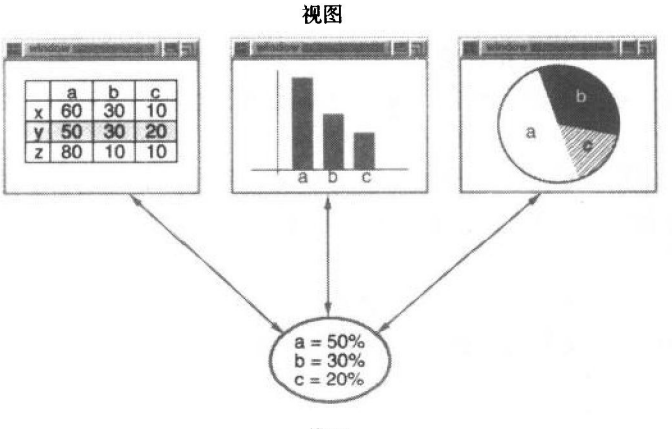
1.2 Smalltalk MVC中的设计模式

在Smalltalk-80中，类的模型/视图/控制器（Model/View/Controller）三元组(MVC)被用来构建用户界面。透过MVC 来看设计模式将帮助我们理解“模式”这一术语的含义。

MVC包括三类对象。模型Model是应用对象，视图View是它在屏幕上的表示，控制器Controller定义用户界面对用户输入的响应方式。不使用MVC，用户界面设计往往将这些对象混在一起，而MVC则将它们分离以提高灵活性和复用性。

MVC通过建立一个“订购/通知”协议来分离视图和模型。视图必须保证它的显示正确地反映了模型的状态。一旦模型的数据发生变化，模型将通知有关的视图，每个视图相应地得到刷新自己的机会。这种方法可以让你为一个模型提供不同的多个视图表现形式，也能够为一个模型创建新的视图而无须重写模型。

下图显示了一个模型和三个视图（为了简单起见我们省略了控制器）。模型包含一些数据值，视图通过电子表格、柱状图、饼图这些不同的方式来显示这些数据。当模型的数据发生变化时，模型就通知它的视图，而视图将与模型通信以访问这些数据值。



表面上看，这个例子反映了将视图和模型分离的设计，然而这个设计还可用于解决更一般的问题：将对象分离，使得一个对象的改变能够影响另一些对象，而这个对象并不需要知道那些被影响的对象的细节。这个更一般的设计被描述成Observer（5.7）模式。

MVC的另一个特征是视图可以嵌套。例如，按钮控制面板可以用一个嵌套了按钮的复杂视图来实现。对象查看器的用户界面可由嵌套的视图构成，这些视图又可复用于调试器。MVC用View类的子类——CompositeView类来支持嵌套视图。CompositeView类的对象行为上类似于View类对象，一个组合视图可用于任何视图可用的地方，但是它包含并管理嵌套视图。

上例反映了可以将组合视图与其构件平等对待的设计，同样地，该设计也适用于更一般的问题：将一些对象划为一组，并将该组对象当作一个对象来使用。这个设计被描述为Composite（4.3）模式，该模式允许你创建一个类层次结构，一些子类定义了原子对象（如Button）而其他类定义了组合对象（CompositeView），这些组合对象是由原子对象组合而成的更复杂的对象。

MVC允许你在不改变视图外观的情况下改变视图对用户输入的响应方式。例如，你可能希望改变视图对键盘的响应方式，或希望使用弹出菜单而不是原来的命令键方式。MVC将响应机制封装在Controller对象中。存在着一个Controller的类层次结构，使得可以方便地对原有Controller做适当改变而创建新的Controller。

View使用Controller子类的实例来实现一个特定的响应策略。要实现不同的响应策略只要用不同种类的Controller实例替换即可。甚至可以在运行时刻通过改变View的Controller来改变View对用户输入的响应方式。例如，一个View可以被禁止接收任何输入，只需给它一个忽略输入事件的Controller。

View-Controller关系是Strategy（5.9）模式的一个例子。一个策略是一个表述算法的对象。当你想静态或动态地替换一个算法，或你有很多不同的算法，或算法中包含你想封装的复杂数据结构，这时策略模式是非常有用的。

MVC还使用了其他的设计模式，如：用来指定视图缺省控制器的Factory Method(3.3)和用来增加视图滚动的Decorator(4.4)。但是MVC的主要关系还是由Observer、Composite和Strategy三个设计模式给出的。

1.3 描述设计模式

我们怎样描述设计模式呢？图形符号虽然很重要也很有用，却还远远不够，它们只是将设计过程的结果简单记录为类和对象之间的关系。为了达到设计复用，我们必须同时记录设计产生的决定过程、选择过程和权衡过程。具体的例子也是很重要的，它们让你看到实际的设计。

我们将用统一的格式描述设计模式，每一个模式根据以下的模板被分成若干部分。模板具有统一的信息描述结构，有助于你更容易地学习、比较和使用设计模式。

模式名和分类

模式名简洁地描述了模式的本质。一个好的名字非常重要，因为它将成为你的设计词汇表中的一部分。模式的分类反映了我们将在1.5节介绍的方案。

意图

是回答下列问题的简单陈述：设计模式是做什么的？它的基本原理和意图是什么？它解

决的是什么样的特定设计问题?

别名

模式的其他名称。

动机

用以说明一个设计问题以及如何用模式中的类、对象来解决该问题的特定情景。该情景会帮助你理解随后对模式更抽象的描述。

适用性

什么情况下可以使用该设计模式?该模式可用来改进哪些不良设计?你怎样识别这些情况?

结构

采用基于对象建模技术(OMT)[RBP+91]的表示法对模式中的类进行图形描述。我们也使用了交互图[JCJO92, BOO94]来说明对象之间的请求序列和协作关系。附录B详细描述了这些表示法。

参与者

指设计模式中的类和/或对象以及它们各自的职责。

协作

模式的参与者怎样协作以实现它们的职责。

效果

模式怎样支持它的目标?使用模式的效果和所需做的权衡取舍?系统结构的哪些方面可以独立改变?

实现

实现模式时需要知道的一些提示、技术要点及应避免的缺陷,以及是否存在某些特定于实现语言的问题。

代码示例

用来说明怎样用C++或Smalltalk实现该模式的代码片段。

已知应用

实际系统中发现的模式的例子。每个模式至少包括了两个不同领域的实例。

相关模式

与这个模式紧密相关的模式有哪些?其间重要的不同之处是什么?这个模式应与哪些其他模式一起使用?

附录提供的背景资料将帮助你理解模式以及关于模式的讨论。附录A给出了我们使用的术语列表。前面已经提到过的附录B则给出了各种表示法,我们也会在以后的讨论中简单介绍它们。最后,附录C给出了我们在例子中使用的各基本类的源代码。

1.4 设计模式的编目

从第3章开始的模式目录中共包含23个设计模式。它们的名字和意图列举如下,以使你有个基本了解。每个模式名后括号中标出模式所在的章节(我们整本书都将遵从这个约定)。

Abstract Factory(3.1):提供一个创建一系列相关或相互依赖对象的接口,而无需指定它们具体的类。

Adapter(4.1)：将一个类的接口转换成客户希望的另外一个接口。Adapter模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

Bridge(4.2)：将抽象部分与它的实现部分分离，使它们都可以独立地变化。

Builder(3.2)：将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。

Chain of Responsibility(5.1)：为解除请求的发送者和接收者之间耦合，而使多个对象都有机会处理这个请求。将这些对象连成一条链，并沿着这条链传递该请求，直到有一个对象处理它。

Command(5.2)：将一个请求封装为一个对象，从而使你可用不同的请求对客户进行参数化；对请求排队或记录请求日志，以及支持可取消的操作。

Composite(4.3)：将对象组合成树形结构以表示“部分-整体”的层次结构。Composite使得客户对单个对象和复合对象的使用具有一致性。

Decorator(4.4)：动态地给一个对象添加一些额外的职责。就扩展功能而言，Decorator模式比生成子类方式更为灵活。

Facade(4.5)：为子系统中的一组接口提供一个一致的界面，Facade模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。

Factory Method(3.3)：定义一个用于创建对象的接口，让子类决定将哪一个类实例化。Factory Method使一个类的实例化延迟到其子类。

Flyweight(4.6)：运用共享技术有效地支持大量细粒度的对象。

Interpreter(5.3)：给定一个语言，定义它的文法的一种表示，并定义一个解释器，该解释器使用该表示来解释语言中的句子。

Iterator(5.4)：提供一种方法顺序访问一个聚合对象中各个元素，而又不需暴露该对象的内部表示。

Mediator(5.5)：用一个中介对象来封装一系列的对象交互。中介者使各对象不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。

Memento(5.6)：在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态。这样以后就可将该对象恢复到保存的状态。

Observer(5.7)：定义对象间的一种一对多的依赖关系,以便当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动刷新。

Prototype(3.4)：用原型实例指定创建对象的种类，并且通过拷贝这个原型来创建新的对象。

Proxy(4.7)：为其他对象提供一个代理以控制对这个对象的访问。

Singleton(3.5)：保证一个类仅有一个实例，并提供一个访问它的全局访问点。

State(5.8)：允许一个对象在其内部状态改变时改变它的行为。对象看起来似乎修改了它所属的类。

Strategy(5.9)：定义一系列的算法,把它们一个个封装起来，并且使它们可相互替换。本模式使得算法的变化可独立于使用它的客户。

Template Method(5.10)：定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。Template Method使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。

Visitor(5.11)：表示一个作用于某对象结构中的各元素的操作。它使你可以在不改变各元素的类的前提下定义作用于这些元素的新操作。

1.5 组织编目

设计模式在粒度和抽象层次上各不相同。由于存在众多的设计模式，我们希望用一种方式将它们组织起来。这一节将对设计模式进行分类以便于我们对各族相关的模式进行引用。分类有助于更快地学习目录中的模式，且对发现新的模式也有指导作用，如表1-1所示。

表1-1 设计模式空间

		目 的		
		创 建 型	结 构 型	行 为 型
范围	类	Factory Method(3.3)	Adapter(类)(4.1)	Interpreter(5.3) Template Method(5.10)
	对象	Abstract Factory(3.1) Builder(3.2) Prototype(3.4) Singleton(3.5)	Adapter(对象)(4.1) Bridge(4.2) Composite(4.3) Decorator(4.4) Facade(4.5) Flyweight(4.6) Proxy(4.7)	Chain of Responsibility(5.1) Command(5.2) Iterator(5.4) Mediator(5.5) Memento(5.6) Observer(5.7) State(5.8) Strategy(5.9) Visitor(5.10)

我们根据两条准则(表1-1)对模式进行分类。第一是**目的**准则，即模式是用来完成什么工作的。模式依据其目的可分为**创建型**（Creational）、**结构型**(Structural)、或**行为型**(Behavioral)三种。创建型模式与对象的创建有关；结构型模式处理类或对象的组合；行为型模式对类或对象怎样交互和怎样分配职责进行描述。

第二是**范围**准则，指定模式主要是用于类还是用于对象。类模式处理类和子类之间的关系，这些关系通过继承建立，是静态的，在编译时刻便确定下来了。对象模式处理对象间的关系，这些关系在运行时刻是可以变化的，更具动态性。从某种意义上来说，几乎所有模式都使用继承机制，所以“类模式”只指那些集中于处理类间关系的模式，而大部分模式都属于对象模式的范畴。

创建型类模式将对象的部分创建工作延迟到子类，而创建型对象模式则将它延迟到另一个对象中。结构型类模式使用继承机制来组合类，而结构型对象模式则描述了对象的组装方式。行为型类模式使用继承描述算法和控制流，而行为型对象模式则描述一组对象怎样协作完成单个对象所无法完成的任务。

还有其他组织模式的方式。有些模式经常会被绑在一起使用，例如，Composite常和Iterator或Visitor一起使用；有些模式是可替代的，例如，Prototype常用来替代Abstract Factory；有些模式尽管使用意图不同，但产生的设计结果是很相似的，例如，Composite和Decorator的结构图是相似的。

还有一种方式是根据模式的“相关模式”部分所描述的它们怎样互相引用来组织设计模式。图1-1给出了模式关系的图形说明。

显然，存在着许多组织设计模式的方法。从多角度去思考模式有助于对它们的功能、差异和应用场合的更深入理解。

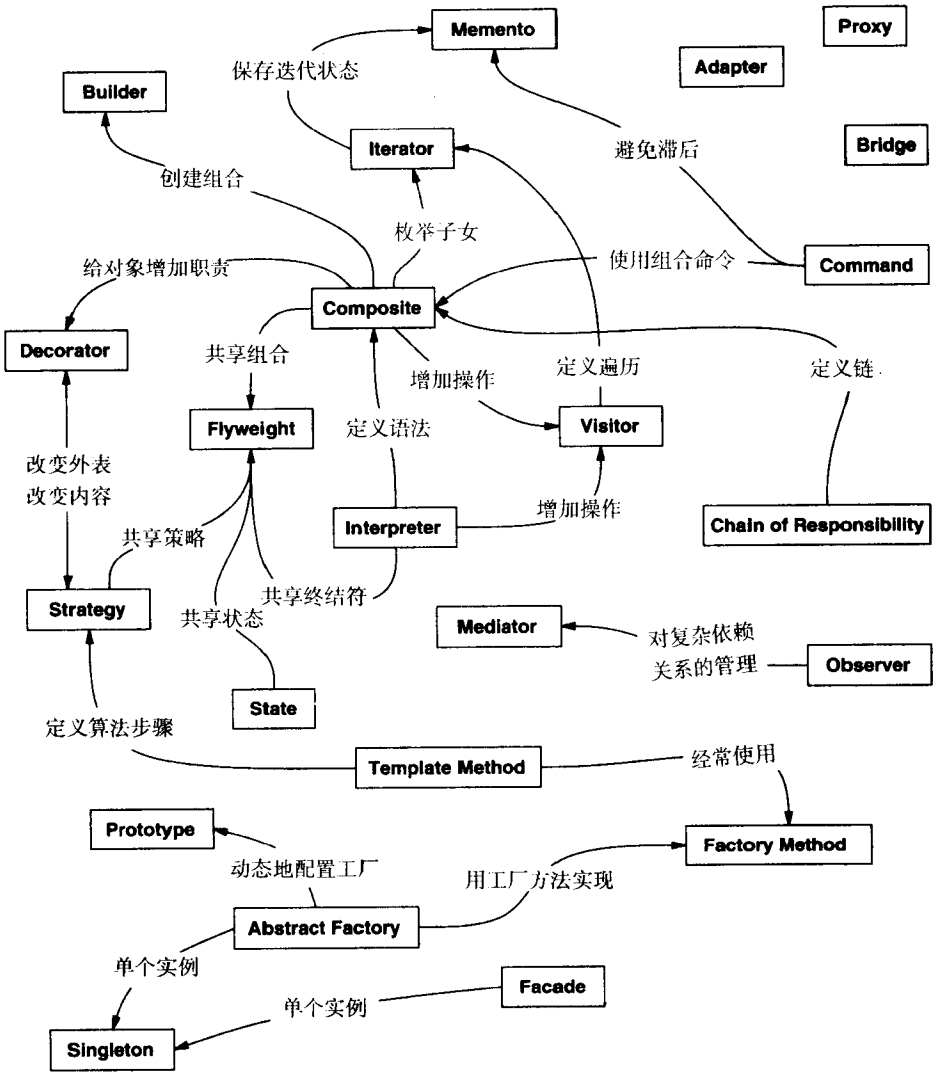


图1-1　设计模式之间的关系

1.6　设计模式怎样解决设计问题

设计模式采用多种方法解决面向对象设计者经常碰到的问题。这里给出几个问题以及使用设计模式解决它们的方法。

1.6.1　寻找合适的对象

面向对象程序由对象组成，**对象**包括数据和对数据进行操作的过程，过程通常称为**方法**或**操作**。对象在收到**客户**的**请求**(或**消息**)后，执行相应的操作。

试读结束：需要全本请在线购买：　www.ertongbook.com