

第6章 结 论

或许有人会认为本书并多大贡献。毕竟，它没有提出任何前所未见的新算法或者新程序设计技术。本书既没有给出一种严格的系统设计方法，也没有提出一套新的设计理论——它只是将现有的一些设计加以文档化。也许你会认为它是一本合适的入门指南，但对有经验的面向对象设计人员却并无多大帮助。

我们希望你不会上面这样的想法。这是因为对设计模式的分类整理是重要的，它为我们使用的各种技术提供了标准的名称和定义。如果我们不研究软件中的设计模式，就无法对它们进行改进，更难以提出新的设计模式。

本书仅仅是一个开始。它讨论了面向对象设计专家们所使用的某些最常见的设计模式，而人们常常也会在口头交谈或分析已有系统时听到和学到这些设计模式。曾有人看了本书的初稿后也将其使用的设计模式写下来，因此，本书就更应起到抛砖引玉的作用。我们希望这将标志着一场把软件从业人员专门知识和技能加以文档化运动的开始。

本章的讨论内容包括我们认为设计模式将带来的巨大影响，设计模式与其他设计工作的关系，以及你怎样发现和整理设计模式。

6.1 设计模式将带来什么

根据我们日常使用设计模式的经验，我们认为它们将在以下几个方面影响你设计面向对象软件的方式。

6.2 一套通用的设计词汇

对使用传统语言的程序设计专家们的研究表明，其知识和经验并非是简单地围绕语法来组织的，而是围绕着诸如算法、数据结构、习惯用语 [AS85,Cop92,Cur89,SS86] 和满足某特定目标的计划 [SE84] 等更大的概念结构来组织的。设计者可能考虑得更多的不是用来记录设计的表示方式，而是如何把当前的设计问题与已知的计划、算法、数据结构和习惯用语等进行匹配。

计算机科学家们对算法和数据结构进行命名和分类，但我们却很少为其他类型的模式命名。设计模式为设计者们交流讨论、书写文档以及探索各种不同设计提供了一套通用的设计词汇。设计模式使你可以在比设计表示或编程语言更高的抽象级别上谈论一个系统，从而降低了其复杂度。设计模式提高了你的设计及你与同事讨论这些设计的层次。

一旦你吸收了本书中的各设计模式，你的设计词汇就几乎肯定要有所改变。你会直接使用这些模式的名称来表示某个设计，比如你会说：“这里我们使用观察者模式”，或者，“让我们从这些类中抽出一个 Strategy”。

6.3 书写文档和学习的辅助手段

了解本书中的各设计模式可使你更容易理解已有的系统。大多数规模较大的面向对象系

统都使用了这些设计模式。人们在学习面向对象编程时常常抱怨系统中继承的使用令人费解以及难于理解控制流程。这在很大程度上是由于他们未能理解该系统中的设计模式。学习这些设计模式将有助于你理解已有的面向对象系统。

这些设计模式也能提高你的设计水平。它们为你提供一些常见问题的解决方案。当然，如果你长期从事面向对象系统的工作，迟早你也会自己学到这些设计模式。但通过本书你可以学得更快。学好这些模式将有助于一个新手做出像专家一样的设计。

而且，按照一个系统所使用的设计模式来描述该系统可以使他人理解起来容易得多，否则，就必须对该系统的设计进行逆向工程来弄清其使用的设计模式。有一套通用的设计词汇的好处是你不必描述整个设计模式，而只要使用它的名字，当他人读到这个名字就会理解你的设计。当然如果读者不知道这个设计模式，他就必须先去找学习该模式，即使这样也还是比逆向工程来的容易。

我们在自己的设计中使用这些模式，并发现它们有很多好处。我们还以某些可争议的幼稚方式使用这些设计模式。我们用它们来为类命名，思考和传授优秀的设计，并用一连串的设计模式来描述我们的设计。很容易想出更复杂的使用设计模式的方式，比如基于模式的CASE工具或超文本文档。不过即使没有复杂的工具，设计模式对我们也还是很有帮助的。

6.4 现有方法的一种补充

面向对象设计方法可用来促进良好的设计，教新手如何设计，以及对设计活动进行标准化。一个设计方法通常定义了一组（常常是图形化的）用来为设计问题各方面进行建模的记号（notation），以及决定在什么样情况下以什么样的方式使用这些记号的一组规则。设计方法通常描述一个设计中出现的问题，如何解决这些问题，以及如何评估一个设计。但设计方法还不能描述设计专家的经验。

我们相信设计模式是面向对象设计方法所缺少的一块重要内容。这些设计模式展示了如何使用诸如对象、继承和多态等基本技术。它们也展示了如何以算法、行为、状态或者需生成的对象类型来将一个系统参数化。设计模式使你可以更多地描述“为什么”这样设计而不仅仅是记录你的设计结果。设计模式的适用性、效果和实现部分都会帮助指导你做出各个必要的设计决定。

设计模式在将一个分析模型转换为一个实现模型的时候特别有用。尽管许多人声称面向对象分析可以平滑地向设计转换，但实践表明远非如此。一个灵活的可复用的设计常会包含一些分析模型中没有的对象。另外，你所使用的编程语言和类库也会影响设计。因此，为使设计可复用，常常需要重新设计分析模型。许多设计模式描述了这样的问题，这也是为什么我们称之为设计模式的原因。

一个成熟的设计方法不仅要有设计模式，还可为其他类型的模式，如分析模式，用户界面设计模式，或者性能调节模式等等。但是设计模式是最主要的部分，这在以前却被忽略了。

6.5 重构的目标

开发可复用软件的一个问题是开发者常常不得不重新组织或重构[OJ90]软件系统。设计模式可以帮助你重新组织一个设计，同时还能减少以后的重构工作。

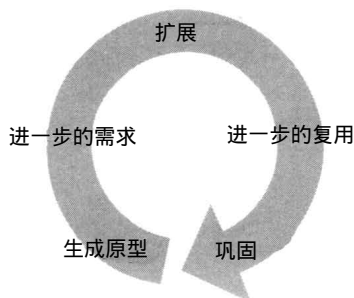
面向对象软件的生命周期常分为几个阶段。Brain Foote将其分为原型阶段、扩展阶段和

巩固阶段三个阶段[Foo92]。

在原型阶段，首先建立一个快速原型，在此基础上进行增量式的修改，直至能满足一组基本需求，然后进入“青春期”。此时，软件中的类层次通常直接反映了原始问题域中的各个实体。该阶段主要的复用方式是通过继承进行白箱复用。

一旦软件进入青春期并交付使用，其演化就由以下两个相互冲突的要求来决定：（1）该软件必须满足更多的需求。（2）该软件必须更易于复用。新的需求常常要求加入新的类和操作甚至增加整个类层次。于是该软件就要经过一个扩展阶段来满足新的需求。然而，这种扩展并不能持续很久。软件的不断扩展将使其变得过于滞胀僵硬而难以进一步修改。软件类层次不再与任何问题域匹配，而是多个问题域的混合反映，并且类中定义了许多不相关的操作和实例变量。

该软件若要继续演化就必须重新组织，这个过程称为重构（refactoring）。框架常常在这个阶段出现。重构工作包括将类拆分为专用和通用的构件，把各个操作在类层次上提或下放到合适的类中，并使各个类的接口合理化。这个巩固阶段将会产生许多新类型的对象，它们通常是通过分解而不是继承原有的对象而得到的。因而黑箱复用代替了白箱复用。满足更多需求和达到更高可复用性的要求推动面向对象软件不断重复扩展和巩固这两个阶段——扩展以满足新的需求，而巩固使软件更为通用（参见下图）。



这个循环是不可避免的。但好的设计者不仅知道哪些变化会促使重构，而且还知道哪些类和对象结构能够避免重构——它们的设计对于需求变化具有健壮性。对需求进行彻底分析有助于突出在软件的生命周期中易于发生变化的那些需求，而一个好的设计应对这些变化保持稳定。

我们的设计模式记录了许多重构产生的设计结构。在设计初期使用这些模式可以防止以后的重构。不过你即使是在系统建成以后才了解如何使用这些模式，它们仍可以教你如何修改你的系统。设计模式为你的重构提供了目标。

6.6 本书简史

分类整理设计模式肇始于 Erich 的博士论文[Gam91, Gam92]的部分工作。他的论文中大约有占本书半数的模式。到 OOPSLA '91 召开的时候它已正式成为一项独立的工作，并且 Richard 已加入进来与 Erich 一道从事这项工作。不久 John 也加入进来。到 OOPSLA '92 的时候，Ralph 也已加入到这个小组中。我们曾试图使我们的工作成果可以发表在 ECOOP '93 上，但我们很快意识到篇幅太长的论文是不会被录用的。所以我们将其简化为一个摘要发表在那次会议上。在那以后我们决定把我们分类整理的模式写成一本书。

在此过程中，我们改动了一些模式的名称。“Wrapper”变成了“Decorator”，“Glue”变成了“Facade”，“Solitaire”变成了“Singleton”，以及“Walker”变成了“Visitor”，并删掉了几个看起来不那么重要的模式。不过自1992年以来，这个分类体系中包含哪些模式没有多大变化，但各模式本身却有了巨大改进。

实际上，注意到某些东西是一个模式还是整个工作中相对容易的部分。我们四个人都经常从事建造面向对象系统的工作，发现当接触到足够多的系统时，发现模式并不困难。然而描述模式却要困难得多。

当你回过头来看你已经建好的一些系统时，会发现所做的工作中就存在着模式。但是，要很好地描述它们以使不熟悉的人也能理解并意识到它们为什么重要就很困难了。专家们能立即从我们模式的早期版本中意识到它们的价值，但也只有这些实际已经用过这些模式的人才能理解它们。

由于本书的主要目的之一在于教设计新手进行面向对象设计，所以我们必须改进模式的分类描述。我们将每个模式的篇幅进行了扩充，其中加入了较具体的说明动机的例子和示例代码，同时对模式的权衡以及实现模式的不同方式也进行了检查。这样就使模式学起来更容易一些。

在过去的一年中所做的另一个重要修改是更加强调一个模式所针对的问题。模式是问题的解决方案，是可以被重复使用的技术手段，这很容易明白；困难的是知道在什么情况下使用这个模式才是恰当的，也就是要刻画这个模式所针对的问题及其上下文，只有在这样的上下文中，这个模式才是最优解。一般而言，了解“做什么”要比“为什么”来的容易；而一个模式的“为什么”就是它要解决的问题。了解一个模式的目的也是重要的，它可以帮助我们选择要使用的模式，也可以帮助我们理解已有系统的设计。作为一个模式的作者，即使你已经知道了解决方案，你还是必须回过头来确定并刻画该模式所解决的问题。

6.7 模式界

我们并不是唯一的对写书来分类整理专家们使用的设计模式感兴趣的小组。我们属于一个更大的圈子，这个圈子里的人们对模式特别是有关软件的模式很感兴趣。建筑师 Christopher Alexander 第一个研究了建筑物和社区的模式，并开发了一个“模式语言”来生成它们。他的工作一次次地启发了我们。所以有必要将我们的工作与他的工作作一个比较，然后我们将看看其他有关软件模式方面的工作。

6.8 Alexander的模式语言

我们的工作在许多方面和 Alexander 的类似。二者都是在观察已有系统的基础上，发现其中的模式，都有描述模式的模板（尽管我们的模板有很大的不同），都是用自然语言和许多例子而不是用形式语言来描述模式，都给出了每个模式背后的原理。

不过我们的工作也在许多方面不同于 Alexander 的模式语言：

- 1) 人类从事建筑活动已有几千年的历史，积累下来许多经典的案例可供参考。相对而言建造软件系统的历史就短的多，很少有系统可称得上经典。
- 2) Alexander 给出了他的模式的使用顺序，而我们没有。
- 3) Alexander 的模式强调它们所针对的问题，而设计模式则更详细的描述了解决方案。

4) Alexander声称他的模式可以生成完整的建筑，而我们不能说我们的模式可以生成完整的程序。

Alexander声称可以通过简单地一个接一个地使用他的模式来设计一所房屋。这类似于一些面向对象设计方法学家的目标，他们也给出了一步步地进行软件设计的规则。Alexander并不否认创造的必要性，他的一些模式要求设计者理解所设计建筑物的使用者的生活习惯。而且，他对设计的“诗意[⊖]”的信仰暗示了存在某种高于模式语言本身的专业水平。不过他对模式怎样生成设计的描述却意味着模式语言可使设计活动成为一种确定的和可重复的过程。

Alexander的观点启发我们关注设计中的权衡问题——多种“力”共同决定了最终的设计结果。在他的影响下，我们慎重考虑了我们的设计模式的适用性及其效果。这也使我们不再试图定义模式的形式化表示。这是因为尽管这种形式化表示将使模式自动化成为可能，但目前更重要的是探索新的模式而不是将模式形式化。

依据Alexander的观点，本书的模式不能形成一个模式语言。考虑到人们建造的软件系统的多样性，我们很难给出一个“完备”的模式集合来指导人们一步步地设计出完整的应用。尽管对于某些特定类型的应用（例如报表生成系统）我们可以做到这一点。然而本书的模式体系仅仅是相关模式的集合，我们不能视之作为一种模式语言。

实际上，我们认为永远也不会有一个完备的软件模式语言。当然我们可以使模式系统更加完整，如可以加入包括框架及其怎样使用框架 [Joh92]，用户界面设计模式 [BJ94]，分析模式 [Coa92]，以及软件开发过程中的其他各个方面内容。设计模式仅仅是一个更大的软件模式语言的一部分。

6.9 软件中的模式

我们第一次集体研究软件体系结构是在 OOPSLA '91大会中一次由Bruce Anderson主持的讨论会上。那次讨论会致力于为软件体系结构设计者编写一本手册（从本书看来，我们认为“体系结构百科全书”这个名称要比“体系结构手册”更好一些）。此后又举行了一系列的会议，最近的一次是1994年8月召开的第一届程序模式语言大会，这次会议建立了一个群体，其兴趣是将软件经验文档化。

当然，也有其他人抱有同样的目标。Donald Knuth的《计算机程序设计的艺术》[Knu73]就是分类整理软件知识的最早尝试之一，只是他着重于描述算法。事实证明，即便如此，这项工作也还是工程浩大而难以完成。《Graphics Gems》系列[Gla90, Arv91, Kir92]是另一个同样着重于算法的设计知识分类体系。美国国防部发起的领域专用软件结构计划集中收集有关体系结构方面的信息。基于知识的软件工程界试图一般地表述软件相关知识。此外还有许多其他小组在为与我们相似的目标而努力。

James Coplien的《Advanced C++: Programming Styles and Idioms》[Cop92]一书也对我们产生了影响。相对于我们的设计模式，该书中描述的模式更加针对 C++语言，而且还包含了许多低层的模式。不过正如在我们的模式中已指出的那样，二者之间是有一些重复的。Jim在模式界很活跃，目前他正在研究那些用来描述软件开发组织中人的角色的模式。

你还可以从其他许多地方找到对模式的描述。Kent Beck是软件界中首先倡导学习Christopher Alexander的工作的先驱者之一。在1993年他开始在《The Smalltalk Report》上撰

⊖ 参见“ The poetry of the language ” [AIS+77]

写关于Smalltalk模式的一个专栏。Peter Coad开始收集模式也有一段时间了。在我们看来，他的关于模式的论文主要讨论的是分析模式 [Coad92]。我们知道他还在继续从事这方面的工作，但我们没有看到他最新的成果。我们也听说有好几本关于模式的书正在撰写之中，但目前一本也没有看到，所以我们只能告诉你它们就要出现了。其中有一本书将来源于 Pattern Language of Programming会议。

6.10 邀请参与

如果你对模式感兴趣的话，你能做些什么呢？首先，你可以在你的设计工作中使用这些设计模式，并寻找其他可用的设计模式。接下来几年里将会有许多有关模式的书和文章出现，所以不愁没地方找新的模式。不断积累和使用你的模式词汇，在与他人讨论你的设计时你可以使用它们，在构思和书写你的设计时也可以使用它们。

其次，提出你的批评。这个设计模式体系是许多人辛勤工作的成果，除了我们之外，还有几十个评论者提出了反馈意见。如果你发现了存在的问题或者觉得某些地方需要进一步解释的话，请和我们联系。同样，对于其他模式体系，也请给予你的反馈意见。模式的一个重要好处在于它提供的设计决策不再是模糊的直觉意向，模式的作者可以明确地说明他在各需求要素间所作的权衡取舍。这就为发现并与作者讨论其模式的不足之处提供了方便。你可以充分利用模式这个优越性。

再次，寻找你使用过的模式，并把它们写下来。把它们作为你的文档的组成部分，给别人看。你并不一定要在研究机构里才可以发掘模式。实际上，如果你没有某方面的实践经验，要发现相关的模式几乎是不可能的。你尽管写下你的模式体系，但一定要让其他人来帮助你使之成形！

6.11 临别感想

最佳的设计要用到许多设计模式，它们契合交织，形成一个更大的整体。正如 Christopher Alexander所说：

以一种松散的方式把一些模式串接在一起来建造建筑是可能的。这样的建筑仅仅是一些模式的堆砌，而不紧凑。这不够深刻。然而另有一种组合模式的方式，许多模式重叠在同一个物理空间里：这样的建筑非常紧凑，在一小块空间里集成了许多内涵；由于这种紧凑，它变得深刻。