JAVA 技术丛书

Java模式，卷2

Patterns in Java, Volume 2

Java模式

Patterns in Java

[美] Mark Grand 著

亢 勇 豆庆华 等译

WILEY

PHEI

電子工業出版社

Publishing House of Electronics Industry

http://www.phei.com.cn

Java 技术丛书

Java 模式

Patterns in Java

［美］Mark Grand 著

亢 勇 豆庆华 等译

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

内容简介

软件模式已成为软件工程领域内的一个热门话题，它可以解决软件开发中的复用问题，并且促进了面向对象软件技术的发展。应用软件模式往往使得软件更加简洁、灵活、易于理解且富有灵活性。

本书将Java语言、统一建模语言（UML）与模式相结合，从软件工程的各个阶段对模式进行了描述，全书共9章，第1章介绍了软件模式的概念和发展，读者可以从中了解模式的相关知识；第2章简要阐述了UML的相关知识，包括类图、协作图、状态图等；第3章详细描述了软件生命周期，并配有具体的实例；第4章到第9章分别介绍了通用职责分配软件模式（GRASP）、图形用户界面（GUI）设计模式、代码编制模式、代码优化模式、代码健壮模式和测试模式。

本书是一本关于Java模式的权威且通俗易懂的指南，可供有经验的程序员掌握所介绍的模式。对于初学者，本书也是一本难得的参考书。

Mark Grand: **Patterns in Java.**
ISBN 0-471-25841-5

版权贸易合同登记号　图字：01-2002-5367

图书在版编目（CIP）数据

Java模式/（美）格兰德（Grand, M.）著；亢勇等译. -北京：电子工业出版社，2004.1
（Java技术丛书）
书名原文：Patterns in Java
ISBN 7-5053-9327-8

Ⅰ.J... Ⅱ.①格... ②亢... Ⅲ.JAVA语言-程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2003）第103248号

责任编辑：赵红燕　王思斯
印　　刷：北京兴华印刷厂
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
经　　销：各地新华书店
开　　本：787 × 980　1/16　　印张：13　字数：270千字
印　　次：2004年1月第1次印刷
印　　数：5000册　定价：25.00元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换；若书店售缺，请与本社发行部联系。
联系电话：（010）68279077。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。

译 者 序

一个优秀的体系架构包含了很多模式，随着模式设计的发展，模式的概念已经深入人心，模式不仅仅在设计中体现，还体现在分析、编码、测试等各个阶段，本书所讨论的正是软件工程各个阶段的软件模式。

软件模式是一种具有可重用性的方案，用来解决软件开发工程中重复出现的问题。它是人们从长期的设计软件、组织软件等实践过程中总结出来的，也是分析问题、管理开发过程的有效工具。同时，软件模式还为我们提供了一种描述抽象事物的标准，可以大大促进软件开发过程中人与人之间的交流。应用软件模式往往使得软件更加简洁、灵活、易于理解且富有灵活性。

本书的作者 Mark Grand 是分布式系统、面向对象设计和 Java 领域的专家，而且是“Patterns in Java”（《Java 模式》）系列丛书的作者。本书是该系列丛书的第二卷，第一卷主要介绍一般设计模式，第三卷系统介绍了设计模式和体系结构模式，这些模式可应用于分布式和企业应用程序。

本书不同于其他的模式书籍，它包含了多种软件模式，并展示了模式在软件开发各个阶段的重要作用。此外，本书选择了纯面向对象语言 Java 和统一建模语言（UML）来对模式进行具体描述。

与软件技术和工具一样，模式也是不断发展变化的，当前的模式只是软件发展过程中阶段性的总结和升华，我们还期待着更多、更新的模式的出现。

本书由亢勇、豆庆华、李鹏、张小明、丁伟、陈仕范、丁春庭翻译并整理。由于译者水平有限，疏漏之处在所难免，恳请读者批评指正。

作 者 简 介

Mark Grand 是 Java 和面向对象领域的专业顾问，他在该领域从业已经超过 20 年。除了为 John Wiley&Sons 公司编著了“Patterns in Java”系列丛书之外，他还为 O'Reilly & Associates 编写了两部 Java 参考手册，并且在众多杂志上发表了多篇论文。Mark 从事软件开发工作多年，作为顾问，他为众多公司和组织提供了 Java 和面向对象技术方面的指导和培训。此外，还为多个大规模商业 Java 工程提供咨询和技术指导。在他潜心研究 Java 之前，曾从事 4GLs 的设计和实现工作长达 11 年之久，在该领域中，他曾经担任过电子数据交换设备的设计师和工程的管理员。Mark 在许多 MIS 组织中工作过，担任过 Oracle 数据库设计师、网络设计师以及 Sun 系统的管理员。从 1982 年起，他开始研究面向对象的编程和设计。Mark Grand 已经获得了 Syracuse 大学计算机科学领域的 B.S.学位。如果需要与他联系，可以访问他的网页，网址为 http://www.mindspring.com/~mgrand。

目　录

第 1 章　软件模式简介

软件模式是一种可重用性的方案，用来解决软件开发工程中重复出现的问题。因为本书整体都是关于软件模式的，因此，在本书中，它们简称为模式。

一个优秀的、经验丰富的程序员往往比一个优秀但经验欠缺的程序员取得更大的成就，那么是什么原因导致了差距呢？其实，两者的差别仅仅取决于经验的多少。经验能够赋予程序员多方面的智慧。一旦一个程序员拥有了丰富的经验，当他遇到新问题时，就能敏锐地意识到新问题和他以前解决过的旧问题之间的相似之处。他们知道对于相似的问题可以通过重复的模式解决。应用这些模式知识，一个有经验的程序员可以在遇到问题时很快联想到应该使用的模式，并且马上应用该模式来解决问题，而不需要专门停下来去分析问题，然后重新提出可能的方案。

当一个程序员最初发现一个模式时，它还仅仅是思维深处的一种体会。在多数情况下，当它从一个无法描述清楚的体会变成一个程序员可以清晰地表达出来的东西的时候，它将发生令人惊奇的变化。这个变化过程也是非常有价值的一步。当一个程序员对一个模式的理解深刻到一定程度的时候，就会适时地将它描述出来，他们还可以充分利用自己的智慧和经验将该模式与其他模式结合到一起。非常重要的一点是，一旦一个模式以文字的形式表达出来，程序员就会对此进行讨论，这将会更有效地将程序员的智慧汇集在一起。他们都在考虑同样的解决方案，惟一的差别在于表达方式不同而已，运用同一种模式可以有效地避免不同程序员对同一问题的解决方案的争论

当新模式发布出来后，会给经验相对匮乏的程序员带来巨大的收获。一旦公布了一种模式，那些经验丰富的程序员就会将它传授给那些对模式还不熟悉的程序员。

编写本书目的在于用通俗的表达方式将模式介绍给大家，并且期待经验丰富的程序员对此提出宝贵的意见。同时，期望那些尚未发现一些模式的程序员能够从中获得相应的知识。

虽然本书涉及的模式范围非常广泛，但是仍然有一些模式没有收录在本书中。其中一些模式读者可能已经发现了，还有一些模式专业性很强，只有非常少的人会有兴趣，其他一些有价值并且具有吸引力的模式将会在本书的修订版中收录。有兴趣的朋友可以就这些模式与作者进行探讨，可发电子邮件给作者，地址为 mgrand@mindspring.com。

本书收录的模式对于缩短软件开发周期具有建设性的价值。另外，存在一些经常在程序中出现但对软件开发不具备建设性价值的模式，这些模式称为反模式（AntiPattern）。反模式抵消模式所带来的益处，本书不对它们进行收录。

1.1 模式发展史简述

软件模式的概念最初起源于建筑领域，一个名为 Christopher Alexander 的建筑师编写了两本描述建筑结构和城市规划模式的著作：*A Pattern Language: Towns, Buildings, Construction* (Oxford University Press, 1977)和 *The Timeless Way of Building* (Oxford University Press, 1979)。这些书中的理论不仅适用于建筑领域，同样对软件开发领域产生了深远的影响。

1987 年，Ward Cunningham 和 Kent Beck 应用 Alexander 的一些理论开发了 5 种用户接口（UI）设计模式。他们在 OOPSLA-87 上发表了论文：*Using Pattern Languages for Object-Oriented Programs,* by Addison-Wesley [Beck-Cunningham87]。

20 世纪 90 年代初期，Erich Gamma，Richard Helm，Ralph Johnson 和 John Vlissides 开始潜心于 *Design Patterns* 一书的编写，该书 1994 年出版发行，在计算机领域产生了非常深远的影响，使得模式的概念深入人心。*Design Patterns* 也常称为 *Gang of Four* 或 *GoF*。

本书介绍的是自 *GoF* 出版后模式和对象的演变过程。*GoF* 中的应用举例采用的是 C++和 Smalltalk 语言。本书中的应用举例都采用 Java 语言，并且所有内容都是围绕着 Java 进行的。当 *GoF* 出版时，统一建模语言（UML）尚未出现，但是现在，它已经被广泛接受，成为面向对象分析和设计的首选工具。因此，UML 在本书中用于描述模式。

1.2 模式介绍

模式的描述通常有一种固定的格式，这种格式应当包括下面的信息：

- 问题描述，它包括一个具体的实例以及一个解决具体问题的专门方案。
- 推导出公式性通用解决方案的各种因素。
- 通用解决方案。
- 用指定的方案处理问题的结论，无论是好是坏。
- 相关模式的列表。

在介绍这些信息的时候，关于模式的不同书籍存在着差别。本书描述模式所采

用的方式因软件处于开发的不同阶段而不同，简而言之，模式的描述与软件生命周期不同阶段之间的差别有关。

模式名称 标题由模式名称和一个说明模式起源的参考书名组成，每个章节包括模式的出处和一般性质的信息。例如验收测试模式[Grand99]，其中，验收测试是模式名称，从参考书目中查询可以知道[Grand99]说明该模式是在本书中第一次正式介绍。模式都是在大量具体实践的基础之上产生的，所以，很多时候，模式的观点都是源于书目之外。通常，模式的编写者都不是最先发现并认可该模式观点的人，本书的作者也不是本书中收录的所有模式的原始发明人。根据模式名列举的参考书目可以提供给读者相应的信息，有利于探索模式本身的开发过程，而不只是基本概念。

概述 在这部分对模式进行了简要描述。概述表达了模式所提供的基本解决方案，首要是针对那些经验丰富的程序员的，他们或许已经知道这种模式，只是还没有给这种模式确定名称。对于经验丰富的程序员，通过名称和概述认定模式之后，往往不再需要有关这种模式的其他描述。

如果读者不能仅仅通过模式的名称和概述就认识一个模式，只要认真阅读有关模式的其他具体解释，就会对模式有较充分的了解。

环境 本节描述模式的链接地址。对于绝大多数的模式来说，“环境”一节介绍的是一个具体例子和一个设计方案的理论根据。

注意 本节归纳了影响解决问题的通用方案的因素，这些方案在“解决方案”中介绍。

解决方案 这部分是模式的核心部分，描述了模式基本问题的全方位的解决方案。

结果 这部分介绍了应用相应的解决方案带来的益处以及消极的影响。

实现 这部分描述实现解决方案时要着重考虑的因素，它可以描述一些通用的变量或简化的解决方案。不是每个模式都涉及这些内容，因此，一些模式的章节可能不含“实现”。

Java API 应用 当模式应用适当的核心 Java API 来举例时，在这部分中会具体指出。没有应用核心 Java API 的模式将不包含该节内容。

代码示例 本节列出了应用模式设计的典型实例的代码。对于一些模式，代码示例不是相关的，例如 GUI 设计模式。但在某些情况下，不同种类的示例却是相关的。

相关模式 本节列举了一系列的相关模式。

1.3 本书结构

本书比以往许多有关模式的书籍所覆盖的模式范围都要广，其目的就是尽可能多地将模式的种类包括在其中，但是由于时间的限制，收录的模式种类还不够全面，因此，请读者关注后续版本。

本书介绍的多种模式用于以下几个方面：

- 为类分派职责
- 设计图形用户界面
- 编写代码
- 测试软件

本书从 UML 子集的描述开始，第 3 章用来概述软件的生命周期，以此来为使用的模式提供环境。此外，第 3 章还包括应用了特定模式的实例研究。其余章节用来介绍不同类型的模式。

本书中使用的 Java 是基于 Java 2 版本的。书中的 UML 图表基于对象管理组织的 UML 标准，1.1 版。

第 2 章　UML 综述

统一建模语言（UML）是一种可用于面向对象分析和设计的符号表示法。本章对 UML 进行了概要描述，介绍 UML 子集和它的扩展。如果想了解完整的 UML，可以访问 http://www.rational.com/uml/documentation.html。

有关 UML 的特定源程序块访问的信息存储在实例类属性中，它们调用一个行为操作的类封装。术语 UML 并不特指某种实现语言。本书假设用户使用的实现语言为 Java，所以绝大多数地方都使用 Java 特定术语，语言符号表示法比术语更好，但是 Java 程序员很少熟悉它。例如，本书的属性（attribute）和变量（variable）是可互换的，但优先选择 Java 特定术语“变量”。此外，操作（operation）和方法（method）是可互换的，但优先选择 Java 特定术语“方法”。

UML 定义了许多不同种类的图，本书中出现的有类图、协作图和状态图。本章描述这三种图以及它们包含的元素。

2.1　类图

一个类图是一个展现类、接口和它们之间关系的图，类是类图的基本元素。图 2.1 提供了一个类的实例，展示了在类图中，一个类可以拥有的多个特征。

类一般画成矩形。矩形可以分成 2 个或 3 个部分。图 2.1 所示矩形的最顶层部分包含了类名、中间层部分列举了类的变量，底层部分列举了类的方法。

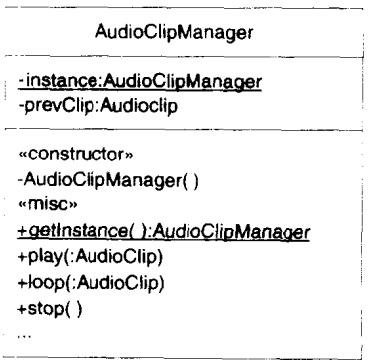

图 2.1　基础类

每个变量和方法前面的符号是可见性指示器（visibility indicators）。表 2.1 中有 3 种不同的可见性指示器。中间层中的变量显示成

```
visibilityIndicator name : type
```

因此，类中显示的两个变量是私有变量。第 1 个变量名是 instance，并且它是 AudioClipManager 类型。第 2 个变量名是 prevClip，AudioClip 类型。

表 2.1 可见性指示器

可见性指示器	含义
+	公有的
#	受保护的
-	私有的

尽管在图 2.1 中并没有显示初始值，但是一个初始值可以通过一个变量类型后跟一个“=”再加上一个值的方式为变量赋值，形式如下：

```
ShutDown:boolean = false
```

注意，类中显示的第一个变量有下划线。如果一个变量有下划线，则说明该变量是一个静态变量。这些规则将同样适用于方法，即有下划线的方法是静态方法。

底层部分的方法显示为：

```
visibilityIndicator name ( formalParameters ) : returnType
```

在图 2.1 中显示的类的 getInstance 方法返回一个 AudioClipManager 对象。

UML 通过省略“: returnType”来表示一个返回空值方法。因此，图 2.1 中显示的 stop 方法将不返回任何结果。

一个方法的形式参数由名称和一个类型组成，形式如下：

```
setLength(length:int)
```

如果一个方法含有多个参数，用逗号将它们分隔开来：

```
setPosition(x:int, y:int)
```

前面描述的类中的两个方法用双括号括住，形式如下：

```
«constructor»
```

在一个 UML 图中，一个双括号中的信息称为固定形式（stereotype）。固定形式类似于用一个形容词来修饰限定其后的内容。constructor 固定形式说明它后面的方法

是构造器，Mist 固定形式表明它后面的方法是常规方法。有关固定形式的其他描述将在本章的后面介绍。

在图 2.1 中出现的最后元素是一个省略号(...)。如果一个省略号出现在一个类的底层部分中，说明这个类含有其他附加的方法，而这些方法在图中没有显示。如果一个省略号出现在一个类的中间层部分中，说明这个类含有其他附加的变量，而这些变量在图中没有显示。

通常，如图 2.1 所示，没有必要将一个类的众多细节都显示出来，而且这样做也并无过多的益处。一个类也可以如图 2.2 所示的那样，将类只分成两个部分。

当一个类只分成两个部分时，它的顶层部分中包含它的名称，而在它的底层部分中显示它的方法。如果一个类只分成两个部分，只是意味着其变量没有显示出来，并不意味它没有变量。

可见性指示器可能被方法和变量省略。当一个方法或变量表示成不带可见性指示器时，它的含义就是没有指示该方法或变量的可见性，也没有指示这些方法或变量是公有的、受保护的或是私有的。

如果已经省略了参数的返回值，就可以省略一个方法的参数。如图 2.3 所示，省略掉了一个类中的可见性指示器和方法参数。

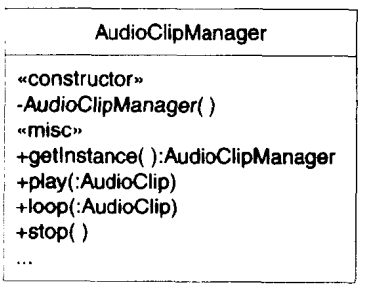

图 2.2　分为两个部分的类

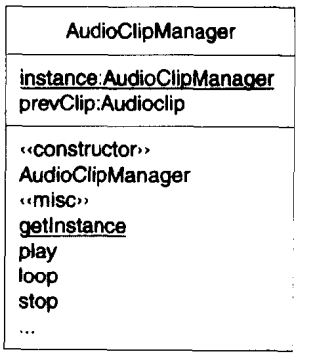

图 2.3　简化的类

图 2.4 展示了一个形式最简单的类，该类只有一个部分，只包含该类的名称。一个只有一个部分的类的表示法仅仅标识一个类，它不提供类中变量或方法的指示。

图 2.4　只有一个部分的类

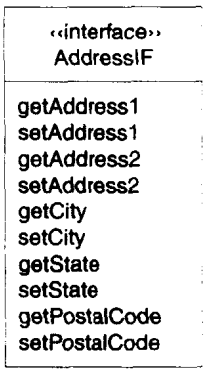

图 2.5 接口

2.1.1 接口

接口的格式与类的格式相似。惟一的区别是在位于顶层的名称前有一个«interface»的固定形式。图 2.5 给出了一个接口的实例。

类和接口是类图的重要组成元素。类图的其他元素表明了类和接口之间的关系，图 2.6 是一个典型的类图。

图 2.6 中的连线说明了类和接口之间的关系。图 2.7 所示的一个闭合箭头的连线表示一个子类和超类之间的继承和被继承关系，图 2.6 说明抽象类 Product 是 ConcreteProduct 的超类，抽象类名、抽象方法名用斜体字表示。

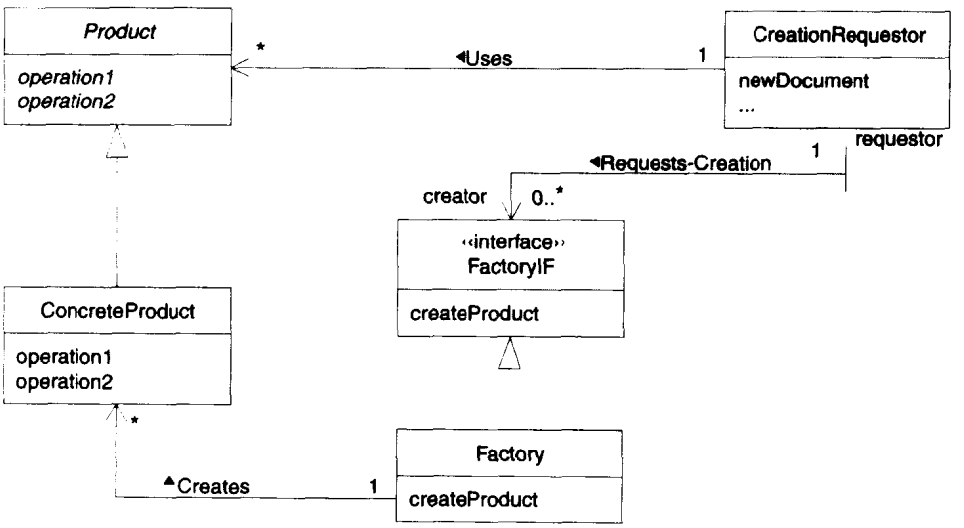

图 2.6 类图

图 2.8 所示的由点划线或虚线前面加一个闭合的箭头组成的连线指出了一个类对于接口的实现。在图 2.6 中，Factory 类实现了 FactoryIF 接口。

其他连线显示了类与接口之间的其他形式的关系。UML 称其他类型的关系为关联，关联关系提供许多相互联系的性质信息。下面的条目是可选的，本书在合适的场合常常使用它们。

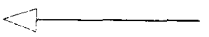

图 2.7 子类从超类中继承

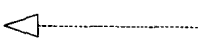

图 2.8 类实现接口

- **关联名**。在一个关联的中间部分会有一个关联名。关联名通常用大写字母开头，一个关联名可以跟在一个三角形的后面。三角形指示研究关联的方向，可以在图 2.6 中找到这样的例子，在 Factory 和 ConcreteProduct 类之间有一个名称为 Creates 的关联。
- **导航箭头**。在一个关联之后出现的箭头称为导航箭头。箭头指明了对关联进行导航的方向，见图 2.6 中名为 Creates 的关联，可以看到一个箭头从 Factory 类指向 ConcreteProduct 类。这说明 Factory 对象将有一个参考，容许 Factory 对象去访问 ConcreteProduct 对象，但不是通过其他的途径。

  因为创建的特性，这种意思会很清楚，即 Factory 类是 ConcreteProduct 类建立的实例类。但一些关联的特征并不明显，为了将这样的关联特征阐述清楚，需要为这些关联补充一些额外的信息。解决这个问题的一个通用途径就是为类在关联活动中充当的角色命名。
- **角色名**。为了将关联特性阐述清楚，每个关联中类的角色名可以出现在每个关联的末尾，紧挨着相应的类。角色名通常小写，而关联名通常大写，这样可以很容易地将角色名与关联名区别开来。图 2.6 的类图说明 CreationRequestor 类和 FactoryIF 接口在一个联合中充当名为 Requests-Creation 的角色。CreationRequestor 类在关联中充当的角色名为 requestor。FactoryIF 接口在关联中充当的角色名为 creator。
- **多重指示器**。通常有关关联的另外一个细节是：一个关联中的每个类究竟有多少实例。一个多重指示器可以位于一个关联的末尾，用来提供这样的信息。它可以是类似于 1 或者 0 的简单数字，也可以是给定范围的数字，如：

  ```
  0..2
  ```

  数值范围的上限值如果用*号表示，就意味着实例的个数为无限个。多重指示器 1..*表示最少为一个实例，0..*表示任意个实例，一个单独的*等同于 0..*。在图 2.6 中的多重指示器表示每一个联合都是多重关系。

图 2.9 的类图说明一个拥有多重子类的类。

图 2.9 的类图是非常完善和有效的，然而，UML 提供了一个更为完善的构思用来描述类和它的子类，可以像图 2.10 中显示的那样，将这些箭头结合起来。图 2.10 与图2.9 的意思相同。

有时候，需要传达比简单的多重关系结构更复杂的结构。一个对象包含一组其他对象的多重关系的类型称为聚合，用位于关联末尾的一个中空菱形表示聚合。出现在一个附属于类的关联末尾的中空菱形包含其他类的实例。图 2.11 中的类图说明一个聚合。