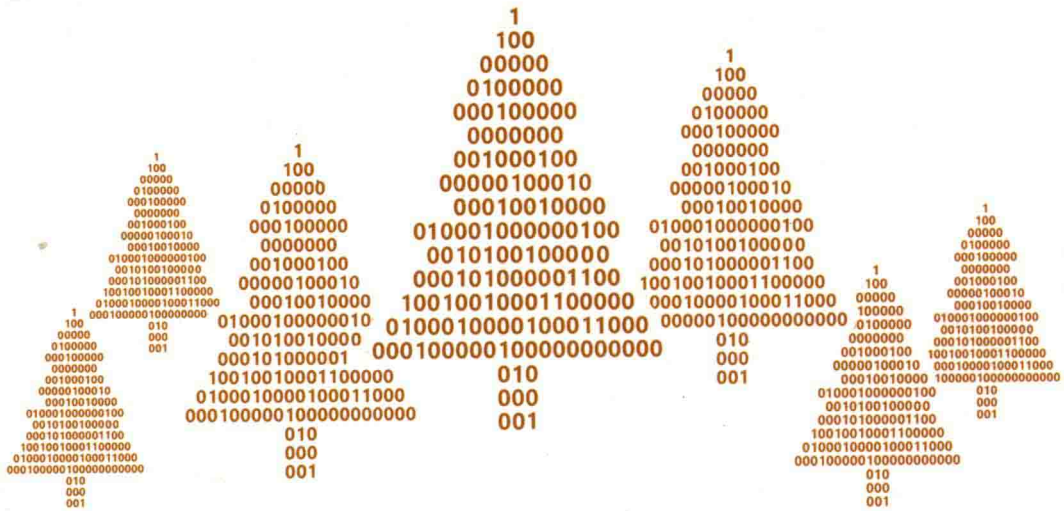
TURING
图灵程序
设计丛书

图解Java多线程设计模式

【日】结城浩 著　侯振龙 杨文轩 译



精选12种设计模式 | 轻松学习多线程编程

264张图表 | 解析Java多线程和并发处理

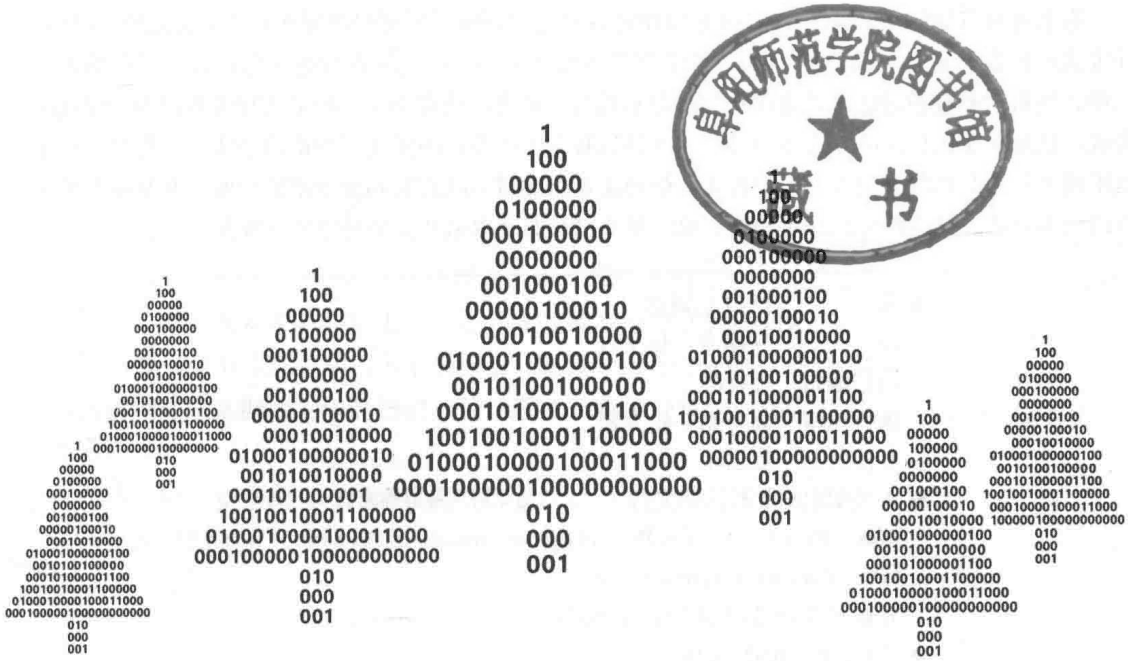
日本经典多线程入门书 | 原版长销11年

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

TURING
图灵程序
设计丛书

图解Java多线程设计模式

【日】结城浩 著　侯振龙 杨文轩 译



人民邮电出版社
北　京

图书在版编目（CIP）数据

图解 Java 多线程设计模式 /（日）结城浩著；侯振龙，杨文轩译 . -- 北京：人民邮电出版社，2017.8
（图灵程序设计丛书）
ISBN 978-7-115-46274-9

Ⅰ . ①图… Ⅱ . ①结… ②侯… ③杨… Ⅲ . ① JAVA 语言 - 程序设计 - 图解 Ⅳ . ① TP312.8-64

中国版本图书馆 CIP 数据核字（2017）第 170408 号

内 容 提 要

本书通过具体的 Java 程序，以浅显易懂的语言逐一说明了多线程和并发处理中常用的 12 种设计模式。内容涉及线程的基础知识、线程的启动与终止、线程间的互斥处理与协作、线程的有效应用、线程的数量管理以及性能优化的注意事项等。此外，还介绍了一些多线程编程时容易出现的失误，以及多线程程序的阅读技巧等。在讲解过程中，不仅以图配文，理论结合实例，而且提供了运用模式解决具体问题的练习题和答案，帮助读者加深对多线程和并发处理的理解，并掌握其使用技巧。本书适合对多线程、Java 编程、设计模式以及面向对象开发感兴趣的读者阅读。

◆ 著　　[日] 结城浩
译　　侯振龙　杨文轩
责任编辑　杜晓静
执行编辑　高宇涵　侯秀娟
责任印制　彭志环
◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京圣夫亚美印刷有限公司印刷
◆ 开本：787×1092　1/16
印张：33
字数：944 千字　　2017 年 8 月第 1 版
印数：1-4 000 册　　2017 年 8 月北京第 1 次印刷
著作权合同登记号　图字：01-2016-3943 号

定价：89.00 元

读者服务热线：(010)51095186 转 600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

译者序

提起多线程编程，恐怕许多开发人员都会摇头表示不懂。确实，在校生和刚就职的开发人员往往很少有机会能够实践多线程编程。多数情况下，他们都是在开发框架下编写单线程的业务代码，而多线程的部分则被封装在了框架内部。即使是经验丰富的开发人员也会感叹他们曾经在多线程上栽过的跟头。但不可否认的是，多线程的确是一把利器，活用多线程有助于提高程序的响应性和吞吐量。可以毫不夸张地说，多线程是开发人员在继续“升级”的过程中必须打倒的一只“怪物”。

“设计模式”一词也常常会让开发人员感到畏惧。其实设计模式不过是对代码设计方式的总结和归纳。在我们的代码中，设计模式无处不在，只是我们没有注意到它们而已。善用设计模式可以帮助我们编写出具有高可复用性且松耦合的代码。

那么，将“多线程”与“设计模式”这两个主题放在一起的这本书，恐怕书名就会让许多读者望而却步吧。但是软件开发就是这么一件有趣的事情——随着我们心中的恐惧与日俱增，想要试着挑战的心情也会越来越迫切。

下面就让我们看看这本书都讲了哪些内容吧。

本书整理了 12 种常用的多线程设计模式，“图、文、码”并茂地讲解了它们各自的优缺点、相互的关联以及适用场景。不过这并不表示本书只适合已经掌握多线程编程的开发人员阅读和参考，因为作者还在讲解各种设计模式的过程中体贴地为初学者穿插介绍了多线程的基本知识。相信无论是新手还是老鸟，都能在阅读本书的过程中有所收获。此外，除第 13 章外，本书每章末尾都配有练习题，读者可以通过做题检验自己是否掌握了各章的知识。

本书的另一大特点是在编写代码实现这 12 种设计模式的基础上，还讲解了如何使用 Java 并发包 `java.util.concurrent` 包去实现这些设计模式。`java.util.concurrent` 是自 J2SE 5.0 起加入的包，在实现并发时非常重要，是并发编程必须要掌握的知识点。

另外，请一定不要忘记学习本书附录 B、附录 C 和附录 D 中介绍的知识哦。

相信读者如果掌握本书中的知识和设计模式，再去理解框架代码或是编写 Swing 程序时一定会得心应手。

本书的出版，要感谢合作译者侯振龙以及图灵公司的高编辑和侯编辑。

最后祝大家都能乐享多线程编程！

杨文轩

2017 年 6 月

引言

大家好，我是结城浩。欢迎阅读《图解 Java 多线程设计模式》。

这是一本讲解 Java 多线程及并发处理模式的入门书。

如果我们在程序中巧妙地利用多线程，便能够并发执行多个处理；在 GUI 应用程序中巧妙地利用多线程，便能够提高对用户的响应性；在服务器上的应用程序中巧妙地利用多线程，便能够并发处理多个用户的请求。多线程是重要的编程技术之一。Java 语言从一开始就加入了多线程功能，所以非常便于初学者学习多线程编程。

一些在单线程程序中并不会发生的 Bug 却会在多线程程序中发生，例如数据可能会损坏，程序可能会发生死锁而无法运行。另外，相对于单线程，多线程可能会占用更多的资源。同时多线程程序中发生的 Bug 很难调试，甚至连 Bug 现象的重现都会非常困难。多线程程序的性能优化也是一项非常难的课题。由此可见，相比单线程编程，多线程编程需要注意的地方更多，因此我们在编程时不能随心所欲，而要采用常用的模式。

本书将通过具体的 Java 程序，逐章介绍多线程编程中常用的模式。首先介绍线程的基础知识，随后介绍线程的启动与终止、线程间的互斥处理与协作、线程的有效应用、线程数量的管理、性能优化的注意事项等。此外，本书还将介绍一些多线程编程时容易出现的失误及多线程程序的阅读技巧等。

自 J2SE 5.0 开始，Java 增加了易于多线程编程的类库——`java.util.concurrent` 包。大家在使用该包时，一定要充分理解 Java 多线程的相关内容，否则将无法充分理解该包提供的类的优势，或者会出现使用的类与自己的目的不匹配的危险情况。本书中随处可见使用 `java.util.concurrent` 包时的注意事项或建议，请务必正确地使用该包。

希望读者朋友们能够通过本书加深对多线程及并发处理相关内容的理解，并掌握其使用技巧。

本书的特点

◆多线程模式的讲解

本书第 1 章 ~ 第 12 章的每一章都会讲解一种多线程及并发处理的模式。另外，每章并不仅仅介绍各个模式的内容，还会讲解各个模式相关的 Java 语言功能，从而加深大家对 Java 语言的理解。

◆ Java 语言的示例程序

本书介绍的所有模式都配有具体的 Java 示例程序。为了便于大家通读整个程序，绝大多数示例程序都很短。另外，各示例程序中并无省略内容，每个程序都可单独编译并执行。

◆模式名称的讲解

模式名称均采用英文表述。本书还讲解了各模式名称的英文读法、含义及中文表述。因此，英文不好的人也可以很容易地记住各个模式并理解它们的内容。

◆练习题

除第 13 章外，每章末尾都配有练习题。为了扩展各章所学内容，以及了解这些模式在实际开

发中的应用，请大家一定要做一下这些练习题。附录 A 提供了所有练习题的答案，自学者也可以轻松学习。

◆主要的 API 文档

附录 D 总结了一些线程相关的主要 API 文档。读者在阅读本书时，可以根据需要自行查阅，也可以整体阅读，以复习相关内容。

◆ java.util.concurrent 包的介绍

J2SE 5.0 的标准库中增加了易于多线程编程的 `java.util.concurrent` 包。在本书各章中，笔者会结合示例程序，讲解 `java.util.concurrent` 包中一些主要的类的用法。另外，附录 E 将全面介绍 `java.util.concurrent` 包。

本书的读者

本书适合以下读者阅读。

- 对多线程感兴趣的人
- 对 Java 编程感兴趣的人
- 对设计模式感兴趣的人
- 对面向对象开发感兴趣的人

阅读本书需具备 Java 语言的基础知识。具体而言，需要能够理解类、实例、字段及方法等，并能够独自编译和运行书中提供的 Java 代码。

虽然本书讲解的是设计模式，但必要时也会对 Java 语言进行讲解，因此读者还可以在阅读本书的过程中加深对 Java 语言的理解。即使是对不怎么了解 Java 多线程的读者而言，本书也具有很大的参考价值。

对于 Java 语言零基础的读者来说，在阅读本书之前，可以先阅读笔者之前出版的《Java 语言编程教程（上・下）第 2 次修订版》[①]（见附录 G 中的 [Yuki05]）。

另外，对于想从零开始学习设计模式的读者来说，在阅读本书之前，可以先阅读笔者之前出版的《图解设计模式》[②]（见附录 G 中的 [Yuki04]）。

本书的结构

本书第 1 章 ~ 第 12 章的每一章都会讲解一种设计模式，这些设计模式都是笔者基于如下两个原则从附录 G 中的参考文献里的设计模式之中挑选出来的。

- 与多线程和并发处理相关的设计模式
- 实际编程中常用的设计模式

序章 1 “Java 线程” 将通过运行一个小程序，来介绍 Java 多线程的基础知识。

① 原书名为『改訂第 2 版　Java 言語プログラミングレッスン（上・下）』，尚无中文版。——译者注

② 杨文轩译，人民邮电出版社，2017 年 1 月。——译者注

序章 2 “多线程程序的评价标准” 将整理多线程程序的评价标准。

第 1 章 “Single Threaded Execution 模式——能通过这座桥的只有一个人” 将介绍多线程编程中最基础的一种设计模式——Single Threaded Execution 模式。该模式可以确保执行处理的线程只能是一个，这样就可以有效防止实例不一致。本章还将深入介绍 Java 语言的 `synchronized` 关键字，并给出计数信号量 `java.util.concurrent.Semaphore` 的示例程序。

第 2 章 “Immutable 模式——想破坏也破坏不了” 将介绍 Immutable 模式，即实例一旦创建完毕，其内容便不可更改的模式。在该模式下，由于实例不会不一致，所以无需执行互斥处理，程序性能也能提高。本章还将讲述 Java 语言中 `final` 的含义，并给出 `Collections.synchronizedList` 及 `java.util.concurrent.CopyOnWriteArrayList` 的示例程序。

第 3 章 “Guarded Suspension 模式——等我准备好哦” 将介绍 Guarded Suspension 模式，即在实例进入目标状态之前，防止线程继续执行的模式。该模式也可以防止实例不一致。通过本章还可以练习 Java 语言中的 `wait` 方法和 `notifyAll` 方法的使用。本章还将给出阻塞队列 `java.util.concurrent.LinkedBlockingQueue` 的示例程序。

第 4 章 “Balking 模式——不需要就算了” 将介绍 Balking 模式，即如果实例未进入目标状态，则中断方法执行的模式。该模式可防止执行无效的等待和多余的方法。

第 5 章 “Producer-Consumer 模式——我来做，你来用” 将介绍 Producer-Consumer 模式。在该模式下，多个线程能够协调运行。采用该模式时，生成数据的线程与使用数据的线程在并发运行时不会互相抢占。本章还将给出阻塞队列 `java.util.concurrent.ArrayBlockingQueue` 的示例程序。

第 6 章 “Read-Write Lock 模式——大家一起读没问题，但读的时候不要写哦” 将介绍 Read-Write Lock 模式，该模式会采用灵活的互斥处理。在该模式下，写数据的线程只能有一个，但读数据的线程可以有很多。该模式能够提高程序的整体性能。本章还将给出可重入的 `java.util.concurrent.locks.ReentrantReadWriteLock` 的示例程序。

第 7 章 “Thread-Per-Message 模式——这项工作就交给你了” 将介绍 Thread-Per-Message 模式，即将处理委托给其他线程的模式。在该模式下，线程可以将任务委托给其他线程，自己则直接处理接下来的工作。该模式能够提高程序的响应性。本章还将介绍 Java 语言中内部类的使用方法，并给出 `java.util.concurrent` 包中 `Executor` 和 `ExecutorService` 的示例程序。

第 8 章 “Worker Thread 模式——工作没来就一直等，工作来了就干活” 将介绍 Worker Thread 模式，即多个线程通过线程池进行等待，然后按照顺序接受工作并执行的模式。该模式可减少创建线程时的资源消耗，还可以通过调节等待线程的个数来控制可用的资源量。本章还将介绍 AWT 及 Swing（JFC）的线程处理方法，并给出通过 `java.util.concurrent` 包来使用线程池的示例程序。

第 9 章 “Future 模式——先给您提货单” 将介绍 Future 模式。在该模式下，可以同步获取交给其他线程的任务的结果。该模式适用于调用异步方法的情况。另外，本章还将给出 `java.util.concurrent.Future`、`FutureTask` 及 `Callable` 的示例程序。

第 10 章 “Two-Phase Termination 模式——先收拾房间再睡觉” 将介绍用于终止线程的 Two-Phase Termination 模式。该模式能够采用合适的终止处理来安全地终止线程。本章还将介绍线程的中断处理，并给出 `java.util.concurrent` 包中 `CountDownLatch`、`CyclicBarrier` 的示例程序。

第 11 章 “Thread-Specific Storage 模式——一个线程一个储物柜” 将介绍 Thread-Specific

Storage 模式。在该模式下，每个线程都会拥有自己的变量空间。采用该模式时，多个线程之间的变量空间是完全分离的，所以并不需要执行互斥处理。本章还将介绍 java.lang.ThreadLocal 类的使用方法。

第 12 章“Active Object 模式——接收异步消息的主动对象”将介绍 Active Object 模式。在该模式下，程序会创建主动对象。该主动对象将接收外部消息，并交由自己的线程来处理。采用该模式时，方法调用和方法执行是彼此分开的。本章还将给出使用了 java.util.concurrent 包中的类的示例程序。

第 13 章“总结——多线程编程的模式语言”将采用模式语言的形式归纳本书所介绍的 12 种模式之间的关系。

本书中的示例程序

支持的版本

本书中的示例程序都是基于 Windows 版的 J2SE 5.0（JDK 1.5.0）[①] 编写的，并已在 Windows XP 上进行了确认。

示例程序的获取方法

本书的示例程序可以从以下网址下载（点击“随书下载”）。

http://www.ituring.com.cn/book/1812

关于 Main 类

在 Java 中，只要类中定义了以下方法，那么无论该类取什么名字，都可以将其作为程序的起点。

```
public static void main(String[] args)
```

但是，为了便于读者理解代码，本书各章的示例程序都是使用 Main 类作为程序的起点的。

关于本书中术语的注意事项

接口和 API

接口这个术语有多个意思。

一般而言，在提到“某个类的接口”时，多是指该类持有的方法的集合。当想要对该类执行某些操作和处理时，需要调用这些方法。

但是在 Java 中，也将“使用关键字 interface 声明的代码”称为接口。

这两个“接口”的意思有些相似，在使用时容易混淆。因此本书中采用以下方式加以区分。

① 在 J2SE 5.0 以后的环境中，编译和运行结果可能与本书不同。——译者注

- 接口（API）：通常的意思（API 是 Application Programming Interface 的缩写）
- 接口：使用关键字 interface 声明的代码

角色

角色是本书中特有的说法。它是指模式（设计模式）中出现的类、接口和实例在该模式中所起的作用。例如，书中会有“由 MakerThread 类扮演 Producer 角色”这种表述。请注意，角色的名字与类和接口的名字不一定相同。

致谢

感谢《Java 并发编程：设计原则与模式（第二版）》[①]（见附录 G 中的 [Lea]）一书的作者 Doug Lea，他的著作给了笔者很大帮助。在本书的写作过程中，笔者通过邮件咨询问题时，他都给予了莫大的支持。对此笔者深表感谢。

感谢设计模式邮件列表[②]中的各位参加者。

然后，还要向阅读笔者拙作，包括图书、连载杂志和电子邮件杂志的读者们表示感谢。另外，还要向笔者 Web 主页上的朋友们表示感谢。

笔者在编写本书的原稿、程序以及图示的过程中，还将它们公布在了互联网上，以供大家评审。在互联网上招募的评审人员不限年龄、国籍、性别、住址、职业，所有交流都是通过电子邮件和网络进行的。在此，笔者要向参与本书评审的朋友们表示感谢，特别是对给予了笔者宝贵意见、改进方案，向笔者反馈错误以及一直鼓励笔者的以下各位表示最真挚的感谢（按五十音图排序）：

新真千惠、天野胜、石井胜、石川草子、井芹义博、植田训弘、植松喜孝、宇田川胜俊、宇野敦之、胡田昌彦、大内宽和、大谷晋平、绪方彰、冈庭祐、奥野皓市、小田浩之、大根田雄一、片冈孝浩、镰田淳、萱森孝、神崎雄一郎、木村明治、清田信行、黑川裕之、小松慎一、酒井敦、榊原知香子、坂本善隆、贞池克己、佐藤贵行、佐藤正明、佐山秀晃、泽田大辅、式见彰浩、清水顺、清水宏行、城生贵幸、助田雅纪、铃木健司、高江洲睦、高岛修、高津修一、高野兼一、高桥武士、高安厚思、土居俊彦、中井健介、中岛雷太、中林俊晴、西海秀俊、平田守幸、藤田宗典、藤田幸久、藤山博人、古川洋介、前原正英、松冈正恭、松本成道、三宅喜义、宫本信二、村田贤一郎、茂木正治、八木希仁、山本耕司、山本正和、丁农、吉田慎太郎、鹭崎弘宜。

此外，对其他参与了评审工作的人员也一并表示感谢。

另外，还要向软银出版股份有限公司图书总编野泽喜美男和编辑松本香织表示感谢。

最后要感谢笔者最爱的妻子和两个健康活泼的儿子。跟上你们的步伐简直比多线程的调试还要难啊。

本书献给在笔者上学时教会笔者如何选购参考书的姐姐。

“在书店看到好的参考书，就要赶紧买下来，要是卖出去不就买不到了吗？”

您的这句话简直就是在说这本书嘛。

结城浩

2002 年 6 月 于横滨

① 赵涌等译，中国电力出版社，2004 年 2 月。——编者注

② 亦称“邮件清单”，包含许多接收者地址的一个电子邮件列表之中，主要用来进行信息发布。这里提到的邮件列表地址为 http://www.hyuki.com/dp/dpml.html，现已停止活动。——编者注

写于“修订版”前

《图解 Java 多线程设计模式》拥有众多读者，笔者深感荣幸，在此再次向各位读者表示最真挚的感谢。

在这次修订中，笔者重新全面地审视了本书的内容和表述，还基于 J2SE 5.0 修改了示例程序，并新增了支持 `java.util.concurrent` 的示例程序。本次修订还参考了读者朋友们发送给笔者的无数反馈意见和建议，真心谢谢你们。

感谢对此次修订工作提供支持的软银出版股份有限公司的总编野泽喜美男和编辑中岛绫子。

希望本书也能在读者朋友的工作和学习中发挥些许作用。

结城浩

2006 年 3 月

关于 UML

UML

UML 是将系统可视化、让规格和设计文档化的表现方法，它是 Unified Modeling Language（统一建模语言）的简称。

本书使用 UML 来描述设计模式中的类和实例的关系，所以我们在这里先稍微了解一下 UML，以方便后面的阅读。但是请大家注意，在说明中我们使用的是 Java 语言的术语。例如，讲解时我们会用 Java 中的“字段”（field）取代 UML 中的“属性”（attribute），用 Java 中的“方法”取代 UML 中的“操作”（operation）。

UML 标准的内容非常多，这里我们只对书中使用到的 UML 内容进行讲解。如果想了解更多 UML 内容，请访问以下网站。UML 的规范书也可以从以下网站下载。

- UML Resource Page

  `http://www.omg.org`

- UML Resource Center

  `http://www-306.ibm.com/software/rational/uml`

- UML 技术资料

  `https://www-01.ibm.com/software/cn/rational/index.html`

类图

UML 中的类图（Class Diagram）用于表示类、接口、实例等之间相互的静态关系。虽然名字叫作类图，但是图中并不只有类。

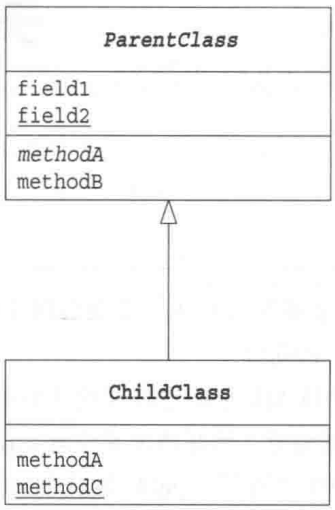
类与层次关系

图 0-1 展示了一段 Java 程序及其对应的类图。

图 0-1 展示类的层次关系的类图

```
abstract class ParentClass {
    int field1;
    static char field2;
    abstract void methodA();
    double methodB() {
        // ...
    }
}

class ChildClass extends ParentClass {
    void methodA() {
        // ...
    }
    static void methodC() {
        // ...
    }
}
```



该图展示了 ParentClass 和 ChildClass 这两个类之间的关系，其中的空心实线箭头表明了两者之间的层次关系，箭头由子类指向父类，换言之，这是表示继承（extends）的箭头。

ParentClass 是 ChildClass 的父类，反过来说，ChildClass 是 ParentClass 的子类。父类也称为基类或超类，子类也称为派生类或继承类。

图中的长方形表示类，长方形内部被横线自上而下分为了如下三个区域。

- 类名
- 字段名
- 方法名

有时，图中除了会写出类名、字段名和方法名等信息外，还会写出其他信息（可见性、方法的参数和类型等）。反之，有时图中也会省略所有不必关注的内容（因此，我们无法确保一定可以根据类图生成源程序）。

abstract 类（抽象类）的名字以斜体方式显示。例如，在图 0-1 中，ParentClass 是抽象类，因此它的名字以斜体方式显示。

static 字段（静态字段）的名字带有下划线。例如，在图 0-1 中，field2 是静态字段，因此它的名字带有下划线。

abstract 方法（抽象方法）的名字以斜体方式显示。例如，在图 0-1 中，ParentClass 类的 methodA 是抽象方法，因此它以斜体方式显示。

static 方法（静态方法）的名字带有下划线。例如，在图 0-1 中 ChildClass 类的 methodC 是静态方法，因此它的名字带有下划线。

▶▶ 小知识：Java 术语与 C++ 术语

Java 术语跟 C++ 术语略有不同。Java 中的字段相当于 C++ 中的成员变量，而 Java 中的方法相当于 C++ 中的成员函数。

▶▶ 小知识：箭头的方向

UML 中规定的箭头方向是从子类指向父类。可能会有人认为子类是以父类为基础的，箭头从父类指向子类会更合理。

关于这一点，按照以下方法去理解有助于大家记住这条规则。在定义子类时需要通过 extends 关键字指定父类。因此，子类一定知道父类的定义，而反过来，父类并不知道子类的定义。只有在知道对方的信息时才能指向对方，因此箭头方向是从子类指向父类。

接口与实现

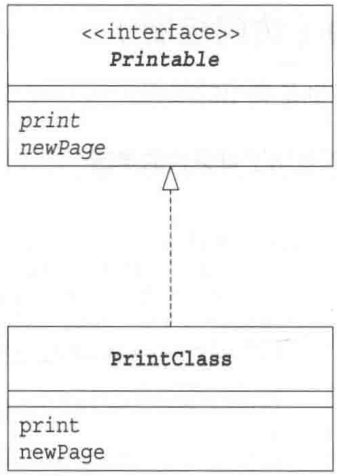
图 0-2 也是类图的示例。该图表示 PrintClass 类实现了 Printable 接口，其中接口名称为斜体。带有空心三角的虚线箭头代表了接口与实现类的关系，箭头从实现类指向接口。换言之，这是表示实现（implements）的箭头。

UML 以 <<interface>> 表示 Java 的接口。

图 0-2 展示接口与实现类的类图

```
interface Printable {
    abstract void print();
    abstract void newPage();
}

class PrintClass implements Printable {
    void print() {
        // ...
    }
    void newPage() {
        // ...
    }
}
```



聚合

图 0-3 也是类图的示例。

该图展示了 Color（颜色）、Fruite（水果）和 Basket（果篮）这三个类之间的关系。Basket 类中的 fruites 字段是可以存放 Fruite 类型数据的数组，在一个 Basket 类的实例中可以持有多个 Fruite 类的实例；Fruite 类中的 color 字段是 Color 类型，一个 Fruite 类的实例中只能有一个 Color 类的实例。通俗地说就是在篮子中可以放入多个水果，每个水果都有其自身的颜色。

我们将这种“持有”关系称为聚合（aggregation）。只要在一个类中持有另外一个类的实例——无论是一个还是多个——它们之间就是聚合关系。就程序上而言，无论是使用数组、java.util.ArrayList 还是其他实现方式，只要在一个类中持有另外一个类的实例，它们之间就是聚合关系。

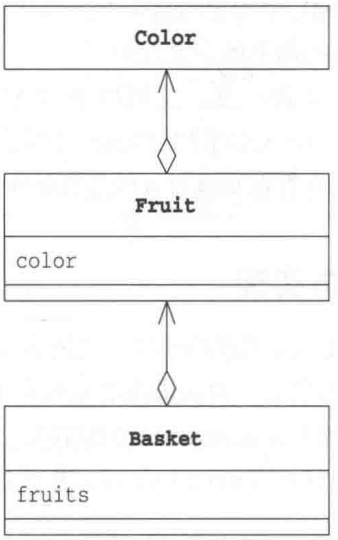
在 UML 中，我们使用带有空心菱形的实线表示聚合关系，因此可以进行联想记忆，将聚合关系想象为在菱形的器皿中装有其他物品。

图 0-3 展示聚合关系的类图

```
class Color {
    // ...
}

class Fruit {
    Color color;
    // ...
}

class Basket {
    Fruit[] fruits;
    // ...
}
```

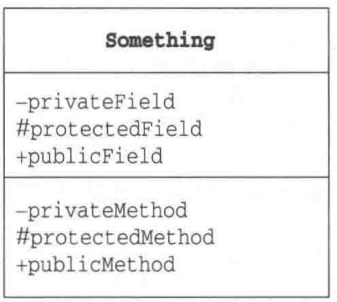


可见性（访问控制）

图 0-4 也是类图的示例。

图 0-4 标识出了可见性的类图

```
class Something {
    private   int privateField;
    protected int protectedField;
    public    int publicField;
    private   void privateMethod() {
    }
    protected void protectedMethod() {
    }
    public    void publicMethod() {
    }
}
```



该图标识出了方法和字段的可见性。在 UML 中可以通过在方法名和字段名前面加上记号来表示可见性。

“+”号表示 public 方法和字段，可以从类外部访问这些方法和字段。

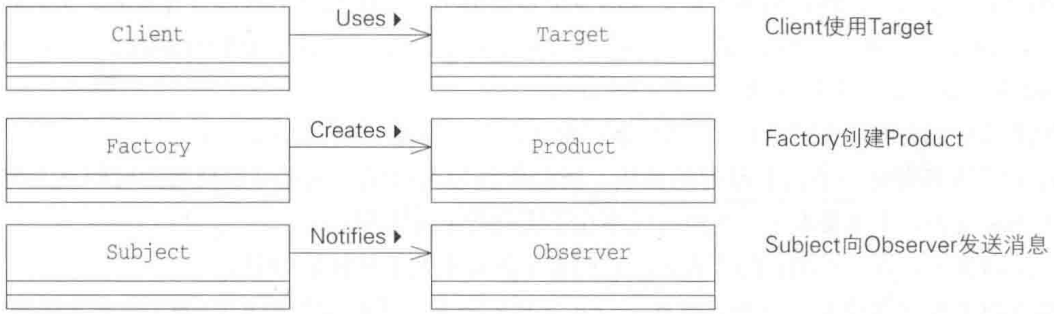
“-”号表示 private 方法和字段，无法从类外部访问这些方法和字段。

“#”号表示 protected 方法和字段，能够访问这些方法和字段的只能是该类自身、该类的子类以及同一个包中的类。

类的关联

可以在类名前面加上黑三角（▶）表示类之间的关联关系，如图 0-5 所示。

图 0-5 类的关联



时序图

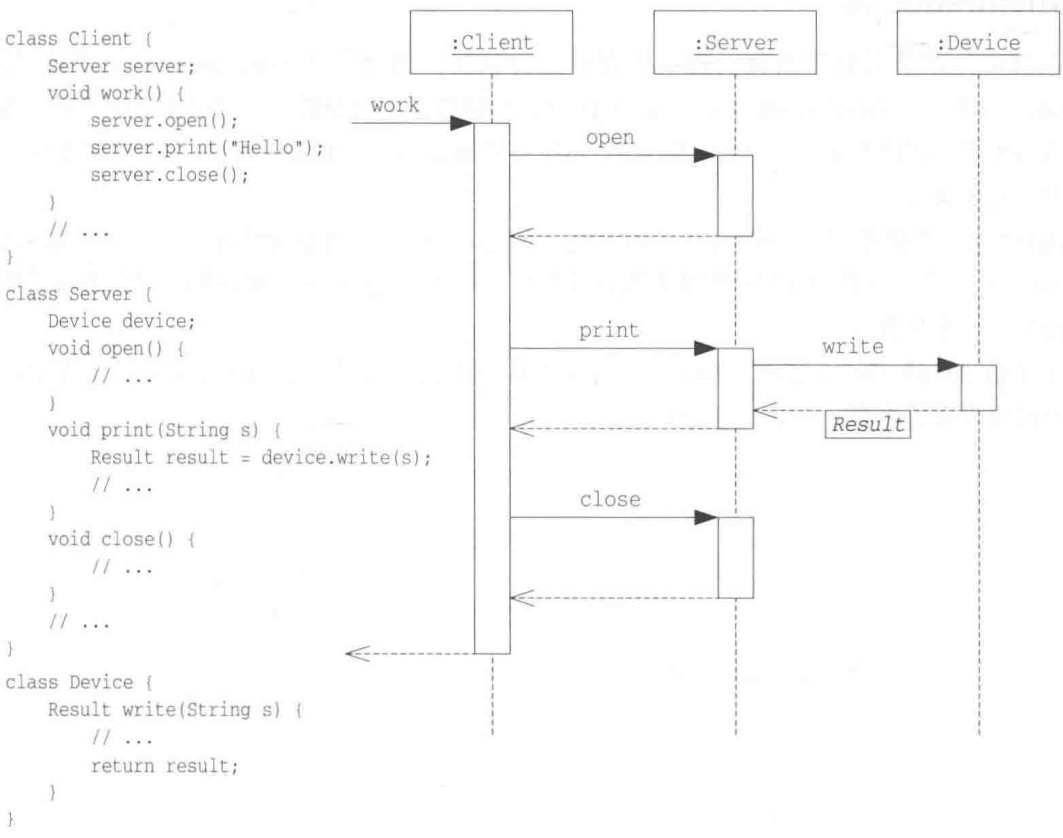
UML 的时序图（Sequence Diagram）用来表示在程序运行时，其内部方法的调用顺序，以及事件的发生顺序。

类图表示的是“不因时间流逝而发生变化的关系（静态关系）”，时序图则与之相反，表示的是“随时间流逝而发生变化的关系（动态行为）”。

处理流程与对象间的协作

图 0-6 展示的是时序图的一个例子。

图 0-6 时序图示例（方法的调用）



在图 0-6 中，右侧是时序图，左侧是与之对应的代码片段。

该图中共有三个实例，如图中最上方的三个长方形所示。在长方形内部写有类名，类名跟在冒号（：）之后，并带有下划线，如 :Client、:Server、:Device，它们分别代表 Client 类、Server 类、Device 类的实例。

如果需要，还可以在冒号（：）之前表示出实例名，如 server:Server。

每个实例都带有一条向下延伸的虚线，我们称其为生命线。这里可以理解为时间从上向下流逝，上面是过去，下面是未来。生命线仅存在于实例的生命周期内。

在生命线上，有一些细长的长方形，它们表示该对象处于某种活动中。

横方向上有许多箭头，请先看带有 open 字样的箭头。黑色实线箭头（—►）表示方法的调用，这里表示 client 调用 server 的 open 方法。当 server 的 open 方法被调用后，server 实例处于活动中，因此我们在 open 箭头处画出了一个细长的长方形来表示。

而在 open 箭头画出的长方形下方，还有一条指向 client 实例的虚线箭头（≪┈），它表示返回 open 方法。在上图中，我们画出了所有的返回箭头，但是有些时序图也会省略返回箭头。

由于程序控制权已经返回至 client，所以表示 server 实例处于活动状态的长方形就此结束了。

接着，client 实例会调用 server 实例的 print 方法。不过这次不同的是在 print 方法中，server 会调用 device 实例的 write 方法。

这样，我们就将多个实例之间的行为用图示的方式展示出来了。时序图的阅读顺序是沿着生命线从上至下阅读。然后当遇到箭头时，我们可以顺着箭头所指的方向查看实例间的协作。

另外，在本书中，如果返回值很重要，需要像 Device 类中 write 方法的返回值 Result 一样，在箭头下方加上一个长方形表示出来（此方式依据附录 G 中的 [POSA2]）。

Timethreads 图

本书中，如果用时序图难以表示线程的运行状况，则会采用 Timethreads 图（Timethreads Diagram）来表示。Timethreads 图并不是 UML 中的标准概念，而是附录 G 中的 [Lea] 使用的表示方法。本书主要采用该方法，而对象名则采用 UML 中的表示方法。Timethreads 图能够将线程的运行可视化，易于理解。

图 0-7 是一个简单的 Timethreads 图的示例。在该示例中，有两个线程对 Data 类的实例调用了 setValue 方法。左侧的线程获取实例的锁并执行 setValue 方法，而右侧的线程则在试图获取该锁时陷入阻塞状态。

在 Timethreads 图中，实例用圆角长方形来表示。而 :Data 中的长方形表示实例持有的锁。关于锁及线程阻塞的内容，序章 1 将会详细讲解。

图 0-7 Timethreads 图示例