Evaluation Warning : The document was created with Spire.PDF for .NET.

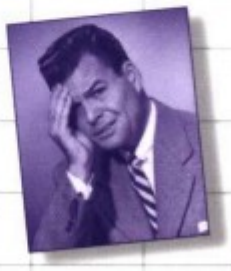
重视大脑的学习指南

本文档由Linux公社 www.linuxidc.com 收集整理

Head First 设计模式

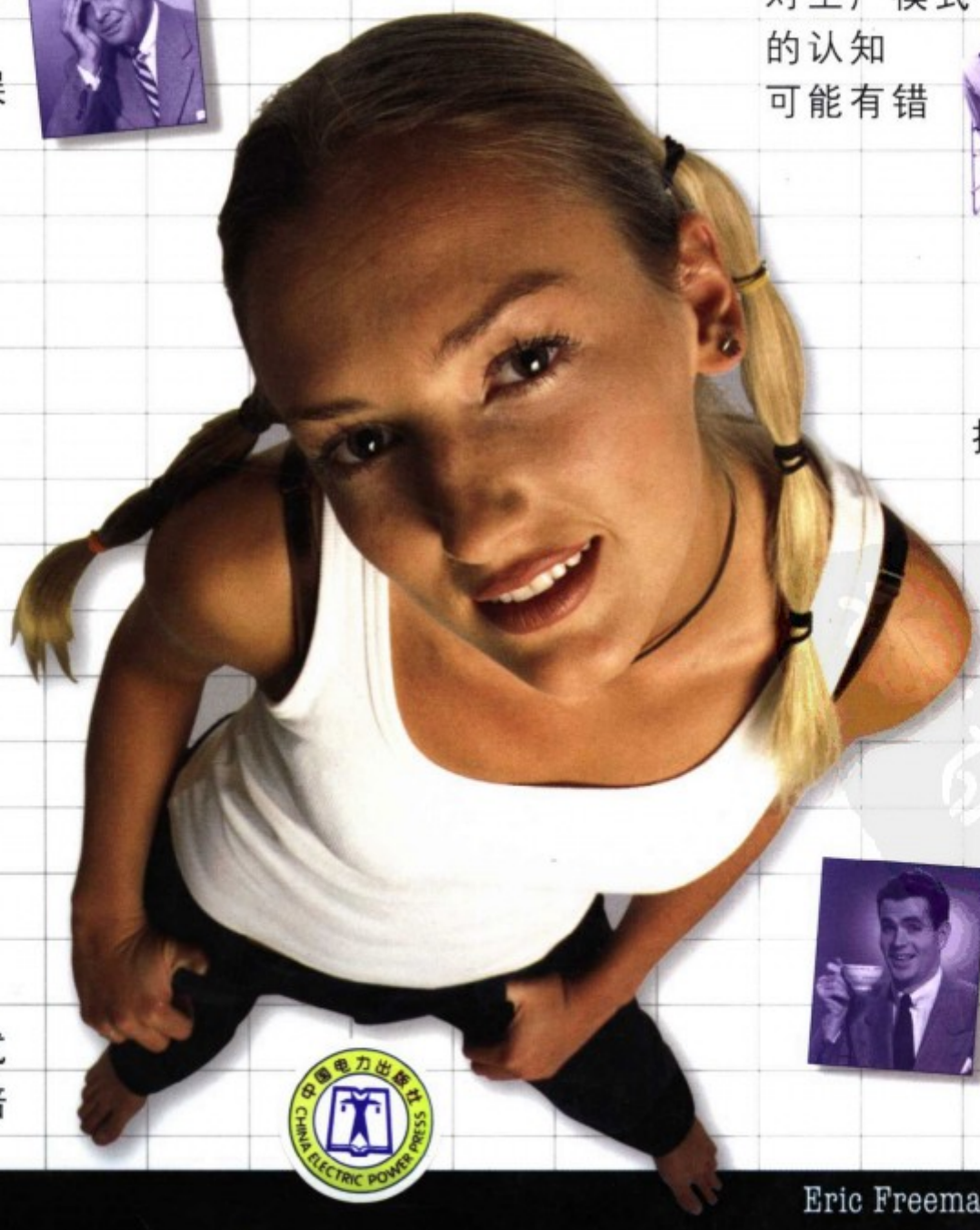
（中文版）

避免一些
尴尬的
耦合错误



学习为何朋友们
对工厂模式
的认知
可能有错

发掘模式大师
的秘密



把事关紧要的模式
直接装入脑海里

探究星巴兹咖啡
如何以装饰者模式
让自己的股价翻倍

瞧瞧Jim
为何拒绝继承后
改善了爱情生活

Eric Freeman & Elisabeth Freeman
with Kathy Sierra & Bert Bates 著
O'Reilly Taiwan公司 译　UMLChina 改编

O'REILLY® 中国电力出版社

Head First设计模式（中文版）

本文档由Linux公社 www.linuxidc.com 收集整理

Software Development/Java

"我昨天收到了这本书后就开始读……我简直欲罢不能。这真是太酷了！不但有趣，涵盖面广，而且切中要点。这本书让我感到印象深刻。"

— *Erich Gamma*,
IBM 杰出工程师、
《设计模式》作者之一

"我感到读这本书的效果等同于读一千磅重的同类书的效果。"

— *Ward Cunningham*,
Wiki 发明者、
Hillside Group 创始人

"本书趋近完美，因为它在提供专业知识的同时，仍然具有相当高的可读性。口吻权威、阅读轻松。"

— *David Gelernter*,
耶鲁大学计算机科学系教授

"这是我阅读过的最有趣且最聪明的软件设计书籍之一。"

— *Aaron LaBerge*,
ESPN.com 技术副主席

本书荣获2005年第十五届Jolt通用类图书震撼大奖。

你不想重新发明轮子（或者更差的是，漏气的轮子），所以你从设计模式中寻求协助——设计模式是过去人们面对同样的软件设计问题所学来的经验。有了设计模式，你就可以利用他人实践经验的精华，省下的时间可以用在……其他的事情上，一些更有挑战性的事情、更复杂的事情、更有趣的事情。你想要学习：

- 事关紧要的模式
- 何时使用某个模式，为何使用该模式
- 如何在自己的设计中马上采用这些模式
- 何时不该使用模式（如何避免对模式过度狂热）
- 模式是基于哪些面向对象设计原则而设计出来的

更重要的是，你在学习设计模式的过程中不会感到昏昏欲睡。如果你曾经读过任何一本Head First系列书籍，就知道你能够从本书中得到的是：透过丰富的视觉效果让你的大脑充分地工作。本书的编写运用了许多最新的研究，包括神经生物学、认知科学，以及学习理论，这使得这本书能将这些设计模式深深地烙在你的脑海中，不容易被遗忘。你将更擅长于解决软件设计中的问题，并能够和你的团队成员用模式的术语沟通。

Eric Freeman和Elisabeth Freeman是作家、讲师，以及技术顾问。原本在迪士尼公司领导了四年的数字媒体，以及Internet的开发，后来，他们将这些经验应用在他们自己的媒体中，包括本书。Eric具有耶鲁大学的计算机科学博士学位，Elisabeth具有耶鲁大学的计算机科学硕士学位。

Kathy Sierra（javaranch.com的创始者）和Bert Bates是畅销的Head First系列书籍的创立者，也是Sun公司Java开发员认证考试的开发者。

O'REILLY®
www.oreilly.com

ISBN 978-7-5083-5393-7
9 787508 353937 >
定价：98.00元

O'Reilly Media, Inc.授权中国电力出版社出版

本文档由Linux公社 www.linuxidc.com 收集整理

TP311.5/195
2007

Head First设计模式

（中文版）

Eric Freeman,
Elisabeth Freeman,
Kathy Sierra &
Bert Bates 著
O'Reilly Taiwan公司 译
UMLChina 改编

O'REILLY®

Beijing • Cambridge • Köln • Paris • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc.授权中国电力出版社出版

中国电力出版社

本文档由Linux公社 www.linuxidc.com 收集整理

图书在版编目（CIP）数据

Head First设计模式（中文版）/（美）弗里曼（Freeman, E.）等著；O'Reilly Taiwan公司译，UMLChina改编－北京：中国电力出版社，2007

书名原文：Head First Design Pattern

ISBN 978-7-5083-5393-7

I. H… II.①弗… ②O… ③U… III. 软件设计 IV. TP311.5

中国版本图书馆CIP数据核字（2007）第031335号

北京版权局著作权合同登记

图字：10-2005-5994号

©2004 by O'Reilly Media, Inc.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and China Electric Power Press, 2007. Authorized translation of the English edition, 2004 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由O'Reilly Media, Inc.出版2004。

简体中文版由中国电力出版社出版，2007。英文原版的翻译得到 O'Reilly Media, Inc.的授权。此简体中文版的出版和销售得到出版权和销售权的所有者 —— O'Reilly Media, Inc.的许可。

版权所有，未得书面许可，本书的任何部分和全部不得以任何形式重制。

书　　名/	Head First设计模式（中文版）
书　　号/	ISBN 978-7-5083-5393-7
责任编辑/	胡顺增，张旻
封面设计/	Ellie Volckhausen，张健
出版发行/	中国电力出版社（www.infopower.com.cn）
地　　址/	北京三里河路6号（邮政编码100044）
印　　刷/	北京盛通印刷股份有限公司
开　　本/	787毫米×980毫米 16开本 42.5印张 928千字
版　　次/	2007年9月第1版 2007年9月第1次印刷
印　　数/	0001－5000册
定　　价/	98.00元（册）

敬告读者

本书封面贴有防伪标签，加热后中心图案消失。

本书如有印装质量问题，我社发行部负责退换。

版权所有　翻印必究

欢迎点击这里的链接进入精彩的Linux公社 网站

Linux公社（www.Linuxidc.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

Linux公社是专业的Linux系统门户网站，实时发布最新Linux资讯，包括Linux、Ubuntu、Fedora、RedHat、红旗Linux、Linux教程、Linux认证、SUSE Linux、Android、Oracle、Hadoop、CentOS、MySQL、Apache、Nginx、Tomcat、Python、Java、C语言、OpenStack、集群等技术。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

Linux公社 主站网址：www.linuxidc.com 旗下网站：www.linuxidc.net

包括：**Ubuntu 专题** **Fedora 专题** **Android 专题** **Oracle 专题** **Hadoop 专题** **RedHat 专题** **SUSE 专题** **红旗 Linux 专题** **CentOS 专题**

Linux公社 www.Linuxidc.com

Linux 公社微信公众号：linuxidc_com

本文档由Linux公社 www.linuxidc.com 收集整理

O'Reilly Media, Inc.介绍

为了满足读者对网络和软件技术知识的迫切需求，世界著名计算机图书出版机构O'Reilly Media, Inc.授权中国电力出版社，翻译出版一批该公司久负盛名的英文经典技术专著。

O'Reilly Media, Inc.是世界上在 Unix、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时也是联机出版的先锋。

从最畅销的《The Whole Internet User's Guide & Catalog》（被纽约公共图书馆评为二十世纪最重要的 50 本书之一）到 GNN（最早的 Internet 门户和商业网站），再到 WebSite（第一个桌面PC的Web服务器软件），O'Reilly Media, Inc.一直处于Internet发展的最前沿。

许多书店的反馈表明，O'Reilly Media, Inc.是最稳定的计算机图书出版商 —— 每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly Media, Inc.具有深厚的计算机专业背景，这使得 O'Reilly Media, Inc.形成了一个非常不同于其他出版商的出版方针。O'Reilly Media, Inc.所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly Media, Inc.还有许多固定的作者群体 —— 他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly Media, Inc.依靠他们及时地推出图书。因为 O'Reilly Media, Inc.紧密地与计算机业界联系着，所以 O'Reilly Media, Inc.知道市场上真正需要什么图书。

本文档由Linux公社 www.linuxidc.com 收集整理

译者序

设计模式（Design Pattern）很重要，不需要我多说。你瞧，程序员几乎人手一本四人组（Erich Gamma、Richard Helm、Ralph Johnson、John Vlissides）所著的《设计模式》。打个比喻：信耶稣的人都要读圣经，而信OO的人都要读四人组的《设计模式》，这就是OO的圣经。更有趣的是，有人还不只买这本书的原版书、连它的光盘版、和中译本也一并买了收藏，可见这是一本多么受到重视的书。我打探过这本书的销售量，它畅销的程度令人咋舌。

许多人反映，四人组的《设计模式》不容易阅读。对于不容易阅读的书，会有已经悟道的人写出白话版或注释版，以飨后进。所以圣经和佛经都有注释版，用更白的方式阐述其中的道理，而我认为《Head First 设计模式》也是因应这样的需求而产生，它可以被视为是白话版、搞笑版、漫画版的《设计模式》。《Head First 设计模式》比起《设计模式》好读得多了，内容也相当有趣。相信我，要写出这样的一本书绝对比写一本正儿八经的书难上许多，可见作者煞费苦心。作者的用心换来空前的成功。《Head First 设计模式》得到相当正面的读者响应，连《设计模式》原创者Erich Gamma也慨然为《Head First 设计模式》写一段推荐文来“作保证”。《Head First 设计模式》还得到2005年的Jolt Award大奖，风光至极。

本书大纲

本书共有14章，每章都介绍了几个设计模式，完整地涵盖了四人组版本全部23个设计模式。前言先介绍这本书的用法；第1章到第11章陆续介绍的设计模式为Strategy、Observer、Decorator、Abstract Factory、Factory Method、Singleton、Command、Adapter、Facade、Template Method、Iterator、Composite、State、Proxy。最后三章比较特别。第12章介绍如何将两个以上的设计模式结合起来成为新的设计模式（例如著名的MVC模式），作者称其为复合设计模式（这是作者自创的名称，并非四人组的标准名词），第13章介绍如何进一步学习设计模式，如何发觉新的设计模式等主题，至于第14章则很快地浏览尚未介绍的设计模式，包括Bridge、Builder、Chain of Responsibility、Flyweight、Interpreter、Mediator、Memento、Prototype、Visitor。

第1章还介绍了四个OO基本概念（抽象、封装、继承、多态），而第 1 章到第9章也陆续介绍了九个OO原则（Principle）。千万不要轻视这些OO原则，因为每个设计模式背后都包含了几个OO原则的概念。很多时候，在设计时有两难的情况，这时候我们必须回归到OO原则，以方便判断取舍。可以这么说：OO原则是我们的目标，而设计模式是我们的做法。

本文档由Linux公社 www.linuxidc.com 收集整理

本书特色

强大的写作阵容。本书作者Eric Freeman和Elisabeth Freeman是作家、讲师和技术顾问。Eric拥有耶鲁大学的计算机科学博士学位，Elisabath拥有耶鲁大学的计算机科学硕士学位。Kathy Sierra（javaranch.com的创始人）和Bert Bates是畅销的Head First系列书籍的创立者，也是Sun公司Java开发员认证考试的开发者。

本书的产品设计应用神经生物学、认知科学，以及学习理论，这使得这本书能够将这些知识深深地印在你的脑海里，不容易被遗忘。本书的编写方式采用引导式教学，不直接告诉你该怎么做，而是利用故事当作引子，带领读者思考并想办法解决问题。解决问题的过程中又会产生一些新的问题，再继续思考、继续解决问题，这样可以加深体会。作者以大量的生活化故事当背景，例如第1章是鸭子，第2章是气象站，第 3 章是咖啡店，书中搭配大量的插图（几乎每一页都有图），所以阅读起来生动有趣，不会感觉到昏昏欲睡。作者还利用歪歪斜斜的手写字体，增加“现场感”。精心设计许多爆笑的对白，让学习过程不会太枯燥。还有模式告白节目，将设计模式拟人化成节目来宾，畅谈其内在的一切。

本书大量采用UML的Class Diagram（Static Structure Diagram）。书中的例子程序虽然都是用Java编写，但是本书所介绍的内容对于任何OO语言的用户都适用，包括C++和C#。每一章都有数目不等的测验题。每章最后有一页要点整理，这也是精华所在，我都是利用这一页做复习。

我认为，这本书的作者全都是“变态”！唔，我是说，好的那种“变态”。毕竟要把这么枯燥的主题写得这么有趣而学习效果又好，不是“变态”的作者还真是做不到呢！

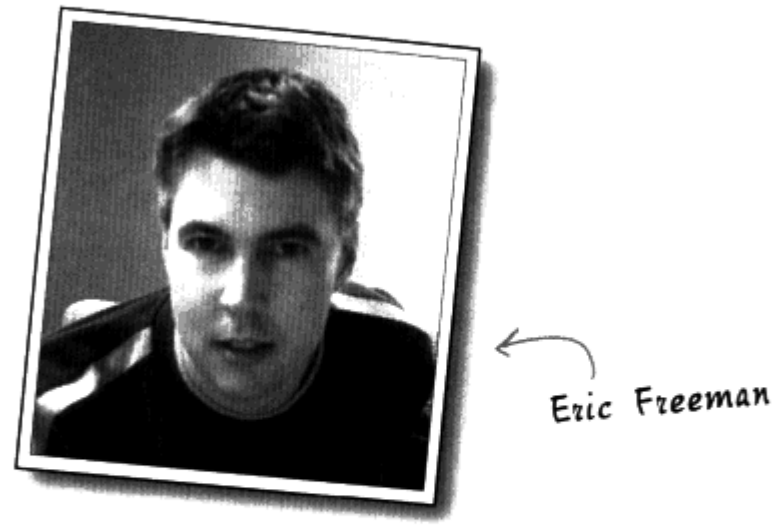
本文档由Linux公社 www.linuxidc.com 收集整理

《Head First设计模式》的作者/开发者

Elisabeth是作者、软件开发人员及数字艺术家。她很早就开始进行Internet相关的研究，也是Ada Project的共同发起人（Ada Project是一个针对在计算机界工作的女性而设计的网站，曾获得大奖，现在已经并入ACM）。最近她带领迪士尼的数字媒体研发力量与他人共同发明了一个名为Motion的内容系统，此系统每天传送巨量的数字内容给迪士尼、ESPN及Movies.com的用户。

Elisabeth本质上是一个计算机科学家，拥有耶鲁大学和印第安那大学的计算机科学硕士学位。她的工作领域很广，包括视觉语言、RSS内容整合与Internet系统。她也很积极提倡女性从事计算机工作。今天，你可以发现她在她的Mac上使用Java或Cocoa，但是其实，她最希望的是全世界都使用Scheme。从小在苏格兰长大，Elisabeth喜欢在大自然踏青及户外活动。一旦她在户外，相机总是不离手。她热爱骑单车，是个素食主义者，也很喜欢动物。

她的电子邮件信箱是beth@wickedlysmart.com，你可以发电子邮件给她。



Eric是一个计算机科学家，热衷于软件架构和媒体。

他刚刚花了四年的时间在一个梦寐以求的工作上：在迪士尼指导Internet宽带与无线应用。现在，他回到写作的岗位上，用Java和Mac创造很酷的软件。

在90年代，Eric和David Gelernter一起花了大量的时间，寻找Desktop metaphor的替代品。（他们“仍然”在问：我干嘛不得不给计算机文件取个名字）。也因为这样的研究，Eric在1997年获得耶鲁大学的博士学位。他也与他人一同创立了Mirror Worlds Technologies公司（已经被收购），将他的论文内容商业化，创建了一套软件Lifestreams。

以前，Eric为网络和超级计算机写软件，你可能通过《JavaSpaces Principles Patterns and Practice》这本书得知他的名号。他曾在Thinking Machine CM-5上实现了元组空间系统（tuple-space system），也在80年代末期为NASA创建了第一个Internet信息系统，他为此深感自豪。

Eric目前住在圣达菲附近的沙漠中，当他不写书或代码时，他总是花更多时间摆弄他的家庭影院，而不是观看影片，他利用空档时间试着修复80年代的经典视频游戏Dragon Lair。他也不介意在晚上兼差当个电音DJ。

给他的E-mail可以写到eric@wickedlysmart.com，你也可以去参观他的Blog，网址在http://www.ericfreeman.com。

本文档由Linux公社 www.linuxidc.com 收集整理

Head First系列的创立者（以及本书共同策划者）

Kathy Sierra

Bert Bates

Kathy自从开始设计游戏以来（她为Virgin、MGM、Amblin等都编写过游戏），一直对学习理论很感兴趣。Head First系列的大多数格式都出自她的手，具体来说，都是她在为UCLA Extension（加利福尼亚大学洛杉矶分校）的“Entertain ment Studies”研究项目教授“New Media Authoring”（新媒体创作）课程时完成的。最近，她成为Sun公司的一名高级培训人员，负责教Sun的Java讲师如何讲授最新的Java技术，并参与开发了多个Sun的认证考试，其中就包括SCBCD考试。与Bert Bates一道，她积极地使用Head First概念来教成千上万的开发人员。她还是世界上最大的Java群体网站javeranch.com的创始人之一，这家网站赢得了2003年和2004年《软件开发》杂志生产力大奖。有时你还会看到她在Java Jam Geek Cruise（geekcruises.com）给学生上Java认证课程。

她最近从加州搬到了科罗拉多，在这里，她得学习一些新的词汇，包括“刨冰机”、“羊绒大衣”（译注），但是在这里的字典里找不到闪电两个字。

喜欢的事：跑步、滑雪、滑板、和她养的冰岛马玩、以及怪力乱神的玩意儿。不喜欢：Entropy（混乱）。

你可以在javaranch.com找到她，偶而她也会出现在java.net的blog中。写给她的信可以寄到kathy@wickedlysmart.com。

译注：加州会打雷，科罗拉多州会下雪。

Bert很早就是一位软件开发者和建构师，不过由于在人工智能领域有近十年的经历，使得他对学习理论和基于技术的培训发生了兴趣。从那以后，他一直在教客户学习编程。最近，他成为Sun的Java认证考试开发小组的一员。

在他软件生涯的最初十年，他全世界游历，向Radio New Zealand、Weather Channel和Arts&Entertaininent Network(A&E)这样一些客户提供帮助。他最得意的项目是为Union Pacific Railroad构建了一个全轨系统仿真应用。

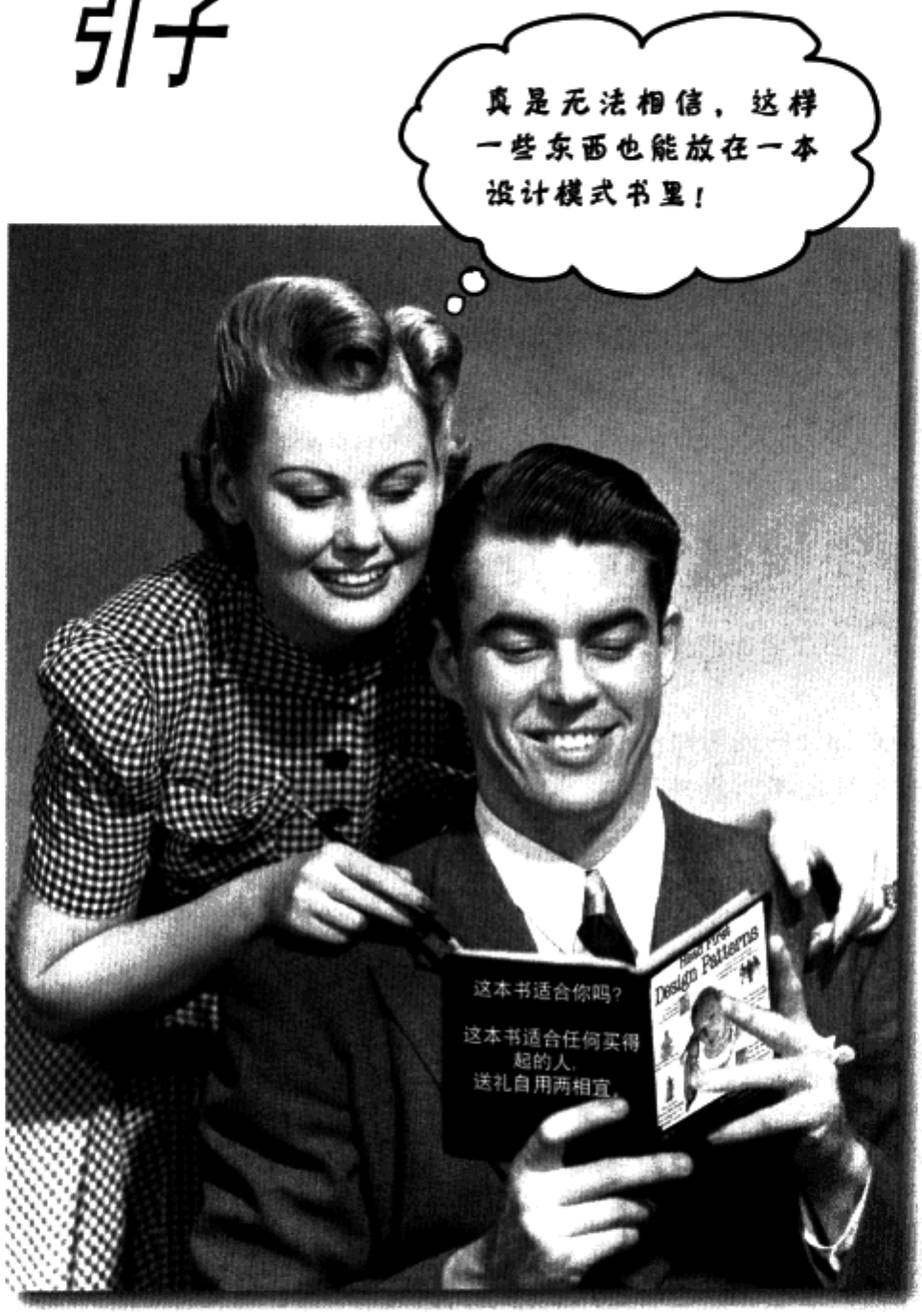
长久以来，Bert一直是无可救药的围棋玩家，玩围棋的时间已经长得超乎想象。他的吉他弹得不错，现在更意图染指Banjo（五弦琴或称斑鸠琴）。

你可以在Javaranch.com找到他，或者在IGS go Server上找到他。你也可以通过terrapin@wickedlysmart.com给他写信。

本文档由Linux公社 www.linuxidc.com 收集整理

如何使用这本书

引子



有一个问题真是听得我们耳朵都磨出茧了，这就是："你们为什么要把这样一些东西放在一本设计模式书里呢？"这一节正是要回答这个问题。

本文档由Linux公社 www.linuxidc.com 收集整理

谁适合读这本书？

如果对下面的所有问题你都能肯定地回答“是”：

① 你懂Java吗？（不过不要求精通。）

如果你会C#可能也可以。

② 你想学习、了解、记得并应用设计模式，以及其所基于的OO设计原则吗？

③ 你是不是更喜欢一种轻松的氛围，就像在餐桌上交谈一样，而不愿意被动地听技术报告似的枯燥乏味的说教？

那么，本书正是你需要的。

谁暂时还不适合读这本书？

如果对下面任何一个问题你能回答“是”：

① 你是不是对Java一无所知？（你不需要是高手，甚至你只会C#但不会Java也没关系，因为两者的相似度是80%。如果你只有C++背景，其实也应该没关系。）

② 你是不是一个很棒的OO设计者/开发人员，正在找一本参考书？

③ 你是不是一个架构师，想找企业设计模式？

④ 你是不是对新鲜事物都畏手畏脚？你是不是宁愿接受牙根管治疗，也不愿意接受苏格兰花格裙？你是不是觉得，如果把Java组件都拟人化了，这样的一本书肯定不是一本正儿八经的技术书？

那么，太遗憾了，本书不适合你。

[营销备注：本书适合所有有信用卡的人。]

本文档由Linux公社 www.linuxidc.com 收集整理

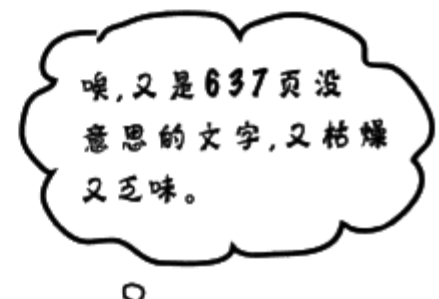
我们知道你在想什么

“这算一本正儿八经的编程书吗？”
“这一堆图是干什么的？”
“我真的能这样学吗？”

我们也知道你的大脑在想什么。

你的大脑总是渴求一些新奇的东西，它一直在搜寻、审视、期待着不寻常的事情发生。大脑的构造就是如此。正是这一点才让我们不至于固步自封，能跟着时代前进。

如今，一般是不太可能被老虎吃掉的。然而，你的大脑还是一直在注意着周围是否有潜伏的老虎。只不过你自己没有意识到而已。但是我们每天都会遇到许多按步就班的事情，这些事情很普通，对于这样一些例行的事情或者平常的东西，你的大脑又是怎么处理的呢？它的做法很简单，就是不让这些平常的东西妨碍大脑真正的工作。那么什么是大脑真正的工作呢？这就是记住那些确实重要的事情。它不会费心地去记乏味的东西；就好像大脑里有一个筛子，这个筛子会筛掉“显然不重要”的东西，如果遇到的事情枯燥乏味，这些东西就无法通过这个筛子。



那么你的大脑怎么知道到底哪些东西重要呢？打个比方，假如你某一天外出旅行，突然一只大老虎跳到你面前，此时此刻，你的大脑里会发生什么呢？

看到这只大老虎，你的神经元会“点火”，情绪爆发，释放出一些化学物质。

好了，这样你的大脑就会知道……

这肯定很重要！可不能忘记了！

不过，假如你正待在家里或者坐在图书馆里，这里很安全，很温暖，肯定没有老虎。你正在刻苦学习，准备应付考试。也可能想学一些比较难的技术，你的老板认为掌握这种技术需要一周时间，最多不超过十天。这就存在一个问题。你的大脑很想给你帮忙。它会努力地把这些显然不太重要的内容赶走，保证这些东西不去侵占本不算充足的脑力资源。这些资源最好还是用来记住确实重要的事情，比如大老虎，再比如火灾险情。如果你曾经只身着短衣裤被大雪围困，这件事肯定不会忘却，你的大脑会记住绝不要让这种情况再发生第二次。

我们没有一种简单的方法来告诉大脑：“嘿，大脑，真是谢谢你了，不过不管这本书多没意思，也不管我对它是多么的无动于衷，但我确实希望你能帮助我把这些东西记下来。”

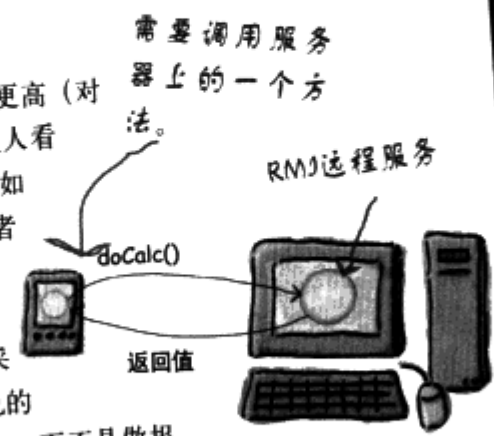
本文档由Linux公社 www.linuxidc.com 收集整理

我们认为“Head First”的读者就是要学习的人

那么，怎么来学习呢？首先，必须了解，然后要保证自己确实不会忘记。这可不是填鸭式的硬塞。根据认知科学、神经生物学和教育心理学的最新研究，学习的途径相当丰富，绝非只是通过书本上的文字。我们很清楚怎么让你的大脑兴奋起来。

下面是一些Head First学习原则：

看得到。与单纯的文字相比，图片更能让人记得住，通过图片，学习效率会更高（对于记忆和传递型的学习，甚至能有多达89%的效率提升）。而且图片更能让人看懂。以往总是把图片放在一页的最下面，甚至放在另外的一页上，与此不同，如果把文字放在与之相关的图片内部，或者在图片的周围写上相关文字，学习者的能力就能得到多至两倍的提高，从而能更好地解决有关的问题。



采用一种针对个人的交谈式风格。最新的研究表明，如果学习过程中采用一种第一人称的交谈方式直接向读者讲述有关内容，而不是用一种干巴巴的语调介绍，学生在学习之后的考试中成绩会提高40%。正确的做法是讲故事，而不是做报告。要用通俗的语言。另外不要太严肃。如果你面对着这样两个人，一个是你在餐会上结识的很有意思的朋友，而另一个学究气十足，喋喋不休地对你说教，在这两个人中，你会更注意哪一个呢？

抽象方法真是简单。这些方法是没有身体的。

```
abstract void roam();
```

没有方法体（方法内的定义）！用一个分号结束。

让学习的人想得更深。换句话说，除非你很积极地让神经元活动起来，否则你的头脑里什么也不会发生。必须引起读者的好奇，促进、要求并鼓励读者去解决问题、得出结论、产生新的知识。为此，需要提出挑战，留下练习题和拓宽思路的问题，并要求读者完成一些实践活动，让左右脑都开动起来，而且要利用多种思维。

说浴缸是一个（IS-A）浴室合适吗？能不能说浴室是一个浴缸？或者，是不是应该说这是一种“有一个”（HAS-A）关系？

引起读者的注意，而且要让他一直保持注意。我们可能都有这样的体验，“我真的想把这个学会，不过看过一页后就变得昏昏欲睡了”。你的大脑注意的是那些不一般、有意思、有些奇怪、抢眼的、意料之外的东西。学习一项有难度的新技术并不一定枯燥。如果学习过程不乏味,你的大脑很快就能学会。

影响读者的情绪。现在我们知道了，记忆能力很大程度上取决于所记的内容对我们的情绪有怎样的影响。如果是你关心的东西，就肯定记得住。如果让你感受到了什么，这些东西就会留在你的脑海中。不过，我们所说的可不是什么关于男孩与狗的伤心故事。这里所说的情绪是惊讶、好奇、觉得有趣。想知道“什么……”，还有就是一种自豪感，如果你解决了一个难题，学会了所有人都觉得很难的东西，或者发现你了解的一些知识竟是那些自以为无所不能的傲慢家伙所不知道的，此时就会有一种自豪感油然而生。

本文档由Linux公社 www.linuxidc.com 收集整理

元认知：有关思考的思考

如果你是真的想学，而且想学得更快、更深入，就应该注意你怎样才能集中注意力。考虑自己是怎样思考的，并了解自己的学习方法。

我们中间大多数人长这么大可能都没有上过有关元认知或学习理论的课程。我们想学习，但是很少有人教我们怎么来学习。

不过，这里可以做一个假设，如果你手上有这本书，你想学设计模式，而且可能不想花太多时间。另外，因为你要参加考试，所以需要记住你读到的所有内容。为此必须理解这些内容。想要最大程度地掌握这本书或其他任何一本书中介绍的知识，就要让你的大脑负起责任来，要求它记住这些内容。

怎么做到呢？技巧就在于要让你的大脑认为你在学习的新东西确实很重要，对你的生活有很大影响。就像老虎出现在面前一样。如若不然，你将陷入旷日持久的拉锯战中，虽然你很想记住所学的新内容，但是你的大脑却会竭尽全力地把它们拒之门外。

那么，究竟怎样才能让你的大脑把设计模式看作是一只饥饿的老虎呢？

这有两条路：一条比较慢，很乏味；另一条路不仅更快，还更有效。慢方法就是大量地重复。你肯定知道，如果反反复复地看到同一个东西，即使再没有意思，你也能学会并记住它。如果做了足够的重复，你的大脑就会说“尽管看上去这对他来说好像不重要，不过，既然他这样一而再、再而三地看同一个东西，那么我就假定这是很重要的。”

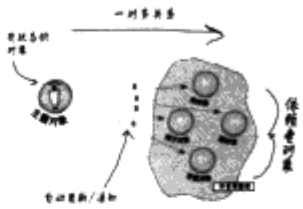
更快的方法是尽一切可能让大脑活动起来，特别是开动大脑来完成不同类型的活动。如何做到这一点呢？上一页列出的学习原则正是一些主要的可取做法，而且经证实，它们确实有助于让你的大脑全力以赴。例如，研究表明，把文字放在所描述图片的中间（而不是放在这一页的别处，比如作为标题，或者放在正文中），这样会让你的大脑更多地考虑这些文字与图片之间有什么关系，而这就会让更多的神经元点火。让更多的神经元点火＝你的大脑更有可能认为这些内容值得注意，而且很可能需要记下来。

交谈式风格也很有帮助，当人们意识到自己在与“别人”交谈，往往会更加关注，这是因为他们总想跟上谈话的思路，并能做出适当的发言。让人惊奇的是，大脑并不关心“交谈”的对方究竟是谁，即使你只是与一本书“交谈”，它也不会不乎！另一方面，如果写作风格很正式，干巴巴的，你的大脑就会觉得像坐在一群人当中被动地听人做报告一样，很没意思，所以不必在意对方说的是什么，甚至可以打嗑睡。

不过，图片和交谈风格还只是开始而已，能做的还有很多。

本文档由Linux公社 www.linuxidc.com 收集整理

我们是这么做的：



我们用了很多图，因为你的大脑更能接受看得见的东西，而不是纯文字。对你的大脑而言，一幅图顶得上1024个字。如果既有图片又有文字，我们会把文字放在图片当中，因为文字处在所描述的图片中间时，大脑的工作效率更高，倘若把这些描述文字作为标题，或者“湮没”在别处的大段文字中，那就达不到这种效果了。

我们采用了重复手法，会用不同的方式，采用不同类型的媒体、运用多种思维手段来介绍同一个东西，目的是让有关内容更有可能储存在你的大脑中，而且能够在多个区中都有容身之地。

模式大师

我们会用你想不到的方式运用概念和图片，因为你的大脑喜欢新鲜玩艺；在提供图和思想时，至少会含着一些情绪因素，因为如果能产生情绪反应，你的大脑就会投入更大的注意。而这会让你感觉到这些东西更有可能要被记住，其实这种感觉可能只是有点幽默，让人奇怪或者比较感兴趣而已。

我们采用了一种针对个人的交谈式风格，因为当你的大脑认为你在参与一个交谈，而不是被动地听一场演示汇报时，它就会更加关注。即使你实际上在读一本书，也就是说在与书“交谈”，而不是真正与人交谈，但这对你的大脑来说并没有什么分别。

在这本书里，我们加入了40多个实践活动，因为与单纯的阅读相比，如果能实际做点什么，你的大脑会更乐于学习，更愿意去记。练习都是我们精心设计的，有一定的难度，但是确实能做出来，因为这是大多数人所希望的。

要点

我们采用了多种学习模式，因为尽管你可能想循序渐进地学习，但是其他人可能希望先对整体有一个全面认识，另外可能还有人只是想看一个代码示例。不过，不管你想怎么学，要是同样的内容能以多种方式来表述，这对每一个人都会有好处。

这里的内容不只是单单涉及左脑，也不只是让右脑有所动作，我们会让你的左右脑都开动起来，因为你的大脑参与得越多，你就越有可能学会并记住，而且能更长时间地保持注意力。如果只有一大脑在工作，通常意味着另一半有机会休息，这样你就能更有效率地学习更长时间。

谜题

我们会讲故事，留练习，从多种不同的角度来看同一个问题，因为如果要求大脑做一些评价和断，它就能更深入地学习。

你会看到我们给出的一些练习，还要回答一些问题，这些问题往往不是直截了当就能做出回答的，通过克服这些挑战，你就能学得更好，因为让大脑真正做点什么的话，它就更能学会并记住。想想吧，如果只是在健身馆里看着别人流汗，这对于保持你自己的体形肯定不会有什么帮助，正所谓临渊羡鱼，不如退而结网。不过另一方面，我们会竭尽所能不让你钻牛角尖，把劲用错了地方，而是能把功夫用在点子上。也就是说，你不会为搞定一个难懂的例子而耽搁，也不会花太多时间去弄明白一段晦涩难懂而且通篇行话的文字，我们的描述也不会太过简洁而让人无从下手。

我们用了拟人手法。在故事中，在示例中，还有在图中，你都会看到人的出现。这是因为你本身是一个人，不错，这就是原因。如果和人打交道，相对于东西而言，你的大脑会表示出更多的注意。

我们充分利用了80/20方法，我们认为，如果你真的要攻读软件设计博士的话，这本书肯定不会是你唯一的设计模式书，所以我们不打算面面俱到。这里只提供了你真正需要的东西。

本文档由Linux公社 www.linuxidc.com 收集整理

沿着虚线剪下，贴在冰箱上。

可以用下面的方法让你的大脑就范

好了，我们该做的已经做了，剩下的就要看你自己的了。这些提示只是个开头：听一听你的大脑是怎么说的，弄清楚对你来说哪些做法可行，哪些做法不能奏效。还可以做些新的尝试。

---

1 慢一点，你理解的越多，需要记的就越少。

不要光是看看而已。停下来，好好想一想。书中提出问题的时候，你不要直接去翻答案。可以假想成真的有人在问你问题。你让大脑想得越深，就越有可能学会并记住。

2 勤做练习，自己记笔记。

我们给你留了练习，但是如果这些练习的解答也由我们一手包办，那和有人替你参加考试有什么区别？不要只是坐在那里看着练习发呆。拿出笔来，写一写、画一画。大量研究都证实，学习过程中如果能实际动动手，将改善你的学习效果。

3 阅读“There are no Dumb Questions”部分。

顾名思义，这些问题可不是可有可无的旁注，它们绝对是核心内容的一部分！千万不要把它们跳过去不看。

4 上床睡觉之前不要再看别的书了,或者至少不再看其他有难度的东西。

学习中有一部分是在你合上书之后完成的（特别是，要把学到的知识长久地记住，这往往无法在看书的过程中做到）。你的大脑也需要有自己的时间来再做一些处理。如果在这段处理时间内你又往大脑里灌输了新的知识，那么你刚学的一些东西就会被丢掉。

5 要喝水，而且要多喝点水。

如果能提供充足的液体，你的大脑才能有最佳表现。如果缺水（可能你觉到口渴之前，就已经缺水了），学习能力就会下降。

6 大声说出来。

说话可以刺激大脑的另一部分。如果你想看懂什么，或者想更牢地记住它，就要大声说出来。更好的办法是，大声地解释给别人听。这样你会学得更快，而且可能会有一些新的认识，而这是以前光看不说的时候未曾发现的。

7 听听你的大脑怎么说。

注意一下你的大脑是不是负荷太重了，如果发现自己开始浮光掠影地翻看，或者刚看的东西就忘记了，这说明你该休息一会儿了。达到某个临界点时，如果还一味地向大脑里塞，这对加快学习速度根本没有帮助，甚至还可能影响正常的学习。

8 要有点感觉！

你的大脑需要知道这是很重要的东西。要真正融入到书中的故事里。为书里照片加上你自己的说明。你可能觉得一个笑话很憋脚，不太让人满意，但这总比根本无动于衷要好。

9 设计一些东西！

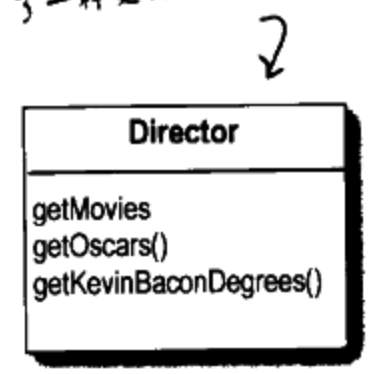
将学来的知识应用到新项目中，甚至重构旧项目。反正就是尽量应用知识，获取实践经验。你所需要的是一枝铅笔和一个难题，试着应用数个设计模式解决这个难题。

本文档由Linux公社 www.linuxidc.com 收集整理

Readme

这是一本体验式学习的书，不是一本参考书。对于学习过程有所阻挠的东西，我们都予以排除。读完第一次之后，你需要从头再读一次，因为本书对读者的背景知识做了一些假设。

我们对UML有所修改，使用了一种更简单的"类"UML.



我们使用简单的"类"UML图（注意，可不是UML类图，而是指与UML图很相似）。

书中用到了UML，但是我们没有详细介绍UML，而UML也不是本书必备的预备知识。如果你以前没见过UML，也别担心。我们会沿路告诉你一些UML的基本用法。换句话说，你根本不需要同时担心UML和设计模式。我们的图示法是"类"UML图——虽然我们试着用真正的UML，但是基于自私的写作必要，我们终究还是做了一些小改变。

我们没有包含所有的设计模式。

设计模式实在是太多了，GoF的基础模式、Sun的J2EE模式、JSP模式、架构模式、游戏设计模式…… 我们希望这本书的重量能比读者的体重更轻，所以自然不可能涵盖所有的设计模式。我们从GoF模式中，取出更重要的一部分模式，作为本书的焦点，并确保你能够真正地、深入地、彻底地了解如何使用这些模式，以及何时使用这些模式。对于GoF的其他模式，我们也会在附录中概略地介绍。我们相信，读过本书之后，你可以很快地从其他资源中学到本书没有介绍的模式，并且游刃有余。

书里的实践活动不是可有可无的。

这里的练习和实践活动并非可有可无的装饰和摆设；它们也是这本书核心内容的一部分。其中有些练习和活动有助于记忆，有些则能够帮助你理解，还有一些对于如何应用你所学的知识很有帮助。所以，请不要略过这些练习。填字游戏是你唯一可以不理会的部分，但是它们可以帮助大脑回想本章的内容。

当我们提到"组合"（composition）一词，我们指的是OO一般概念中的composition，而不是UML严格定义的composition。

当我们说"一个对象和另一个对象组合在一起"，我们的意思是"有一个"（HAS-A）的关系。在一般的OO概念及GoF的书中，都是采用这样的用法。最近UML对于composition有严谨的定义，如果你是UML专家，你还是可以读这本书，只是要注意到此名词定义上的差异。

本文档由Linux公社 www.linuxidc.com 收集整理

我们有意安排了许多重复，这些重复非常重要。

Head First系列图书有一个与众不同的地方，这就是，我们希望你确确实实地掌握这些知识。另外，我们希望在学完这本书之后你能记住学过了什么。尽管重复很有必要，不过，多数参考书都不认为重复和回顾是一个重要的环节，但是在这本书里，你会看到一些概念会一而再、再而三地出现很多次。

代码示例尽可能短小精悍。

有读者告诉我们，如果查了200行代码才能找到要理解的那两行代码，这是很让人郁闷的。这本书里大多数示例往往都开门见山，作为上下文的代码会尽可能地少，这样你就能一目了然地看到哪些东西是需要你学习的。别指望这些代码很健壮，要知道这里的代码甚至是不完整的——毕竟我们的代码是辅助学习之用，所以不见得一定功能完整。

在某些例子中，我们并未将所有需要的package都import进来，但如果你是Java程序员，你应该知道ArrayList类是属于java.util package。如果package不属于J2SE API，我们会特别说明。我们已经将所有的代码都放在网络上，可供下载。网址在：http://wickedlysmart.com/headfirstdesignpatterns/code.html。

为了方便学习与测试程序，我们在书中并没有将我们的类放在package中（换句话说，所有的类都是在Java默认的package中）。我们不建议你在真实世界中也这么做。如果你到我们的网站下载代码，会发现这些类都放在适当的package中。

“Brain Power”习题没有答案。

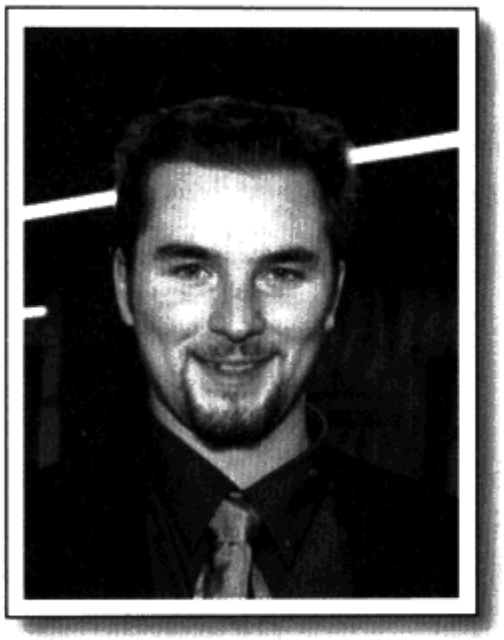
对于某些人来说，“Brain Power”习题没有对的答案；对于另一些人来说，动动脑习题所带来的学习经验在于决定是否你的答案是对的，以及何时你的答案是对的。在某些动动脑习题中，我们会提供暗示，为你指引正确的方向。

本文档由Linux公社 www.linuxidc.com 收集整理

技术审校群

Jeb Cumps

Valentin Crettaz



Barney Marispini

Ike Van Atta

本书狂热的审校团队的无畏的队长

Johannes deJong

Jason Menard

Mark Spritzler

Dirk Schreckmann

本文档由Linux公社 www.linuxidc.com 收集整理

Philippe Maquet

怀念Philippe Maquet

1960—2004

你那惊人的技术专长、不懈的热忱、为学习者的深思熟虑，
将永远激励我们。

我们永远缅怀你。

致谢

致O'Reilly：

在O'Reilly我们对Mike Loukides致以最大的谢意，感谢他开始这一切，并将Head First观念形成一个系列。对Head First幕后的推手Tim O'Reilly致以衷心的感谢。感谢聪明的Head First“系列之母”Kyle Hart，还有摇滚明星Ellie Volkhausen和她灵感十足的封面设计，还有Colleen Gorman的核心编辑。最后，感谢Mike Hendrickson支持这本“设计模式”的书，并建立了整个的团队。

我们大无畏的审校者：

我们特别感谢技术审校的队长Johannes deJong。Johannes，你是我们的英雄。我们深深地感谢Javaranch 审校团队共同管理者的贡献，已故的Philippe Maquet，你以只手照亮了上千开发人员的生活，永远地影响了他们（还有我们）的生活。

Jef Cumps总是能在我们的草拟章节中找出问题，并再三地造成本书巨幅的改变，谢了！Jef!

Valentin Cretazz（专搞AOP的人），他从第一本Head First开始就跟着我们，总是适时地提供我们刚好需要的技术专长，以及他的洞察力。你真行，Valentin。

Head First审阅团队有两位新人，Barney Marispini和Ike Van Atta担任专挑本书毛病的工作，你们两位给我们真正严酷的反馈，谢谢你们的加入。

我们还从Javaranch的主考人/大师——Mark Spritzler、JasonMenard、Dirk Schreckmann、Thomas Paul与Margarita Isaeva等人那里得到杰出的技术帮助。一如平常，要特别感谢javaranch.com Trail的老板Paul Wheaton。

感谢参加“挑选HFDP封面”竞赛的最后决赛入围者。赢家是Si Brewster，他提交了获胜的文字，说服我们选用本书封面的女人。其他入围最后决赛的有：Andrew Esse、Gian Franco Casula、Helen Crosbie、PhoTek、Helen Thomas、Sateesh Kommineni及Jeff Fisher。

本文档由Linux公社 www.linuxidc.com 收集整理

还有更多人要感谢*

来自Eric和Elisabeth的感谢

与两位令人惊讶的“导游”——Kathy Sierra和Bert Bates——共同写一本Head First书，是一次放肆的旅行。你们把所有写书的惯例一股脑儿地抛弃，而带我们进入充满说故事、学习理论、认知科学及流行文化的世界，这是读者成为主宰的世界。谢谢两位让我们进入你们的神奇世界，我们希望作这本Head First是正确的。老实说，我们还震惊不已。谢谢你们细心地指导、推动我们向前走，而最主要的就是信任我们（还有你的宝贝）。我们知道，两位都是相当地“鬼灵精”，也都是最时尚的29岁，所以……接下来呢？

要大大地感谢Mike Loukides和Mike Hendrickson。Mike L.从头到尾都伴随着我们。Mike，你有深刻见解的反馈帮助了本书的形成，你的鼓励让我们继续往前走下去。Mike H.，感谢你持续五年来游说我们写一本关于模式的书；我们终于做到了，并且很高兴等到了Head First系列。

特别感谢Erich Gamma，他所做的已经远超过审校本书的责任（甚至在他度假时，都带着本书的草稿）。Erich，你对本书的关注激励了我们，而你彻底的技术审校，无可估量地改善了这本书。同样感谢整个四人组的支持和关注，并且还特别出现在对象村。我们也从Ward Cunningham和模式社群处受惠不少，他们创建了波特兰模式库（Portland Pattern Repository）——我们写本书时不可或缺的资源。

写本技术书需要集结一些人的智慧与力量：Bill Pugh和Ken Arnold在单件模式上，给了我们专业的建议；Joshua Marinacci提供了Swing的技巧和建议；John Brewer的“为什么是鸭子”产生了模拟鸭子的设想（我们很高兴，他也喜欢鸭子）；Dan Friedman激发了小单件的例子；Daniel Steinberg担任我们的技术联络和感情网络；再感谢Apple的James Dempsey，允许我们使用他的MVC歌曲。

最后，私下感谢Javaranch审校团队，为我们做最高级别的校对，以及温馨的支持。还有更多人，没有写在这里……。

来自Kathy和Bert的感谢

我们很想感谢Mike Hendrickson找到Eric和Elisabeth……但是不能。因为这两人，我们发现（令我们恐怖的是），已经不只有我们可以写Head First的书了。不过，如果读者想要相信，在书里所有的“酷事”都是Kathy和Bert的作为，那么，“我们”是谁，可以让他们循规蹈矩？

* 之所以要感谢这么多人，是因为我发现了这样一条定律，书中致谢部分里提到的每个人都至少会买一本书，可能还会买好几本书,给亲戚和周围的所有人都送上一本,如果你希望我们在下一本书的致谢里提到你,而且你们家族的人很多的话,可以写信给我们。

本文档由Linux公社 www.linuxidc.com 收集整理

目录（概览）

	引子	xxvii
1	欢迎来到设计模式世界：设计模式入门	1
2	让你的对象知悉现况：观察者模式	37
3	装饰对象：装饰者模式	79
4	烘烤OO的精华：工厂模式	109
5	独一无二的对象：单件模式	169
6	封装调用：命令模式	191
7	随遇而安：适配器与外观模式	235
8	封装算法：模板方法模式	275
9	管理良好的集合：迭代器与组合模式	315
10	事物的状态：状态模式	385
11	控制对象访问：代理模式	429
12	模式中的模式：复合模式	499
13	真实世界中的模式：与设计模式相处	577
14	附录A：剩下的模式	611

目录（真正的目录）

引子

让你的大脑来学设计模式。你想学些东西，但是你的大脑却在帮倒忙，不让你记住这些东西。你的大脑在想，“还是把空间留给更重要的事情吧，比方说要躲避的野兽，还有，光着身子滑雪不太好吧。”那么你该如何骗过大脑，让它认为要是不知道设计模式你就活不下去了？

谁适合读这本书？	xxviii
我们知道你的大脑在想什么	xxix
元认知	xxxi
让你的大脑就范	xxxiii
技术审校	xxxvi
致谢	xxxvii

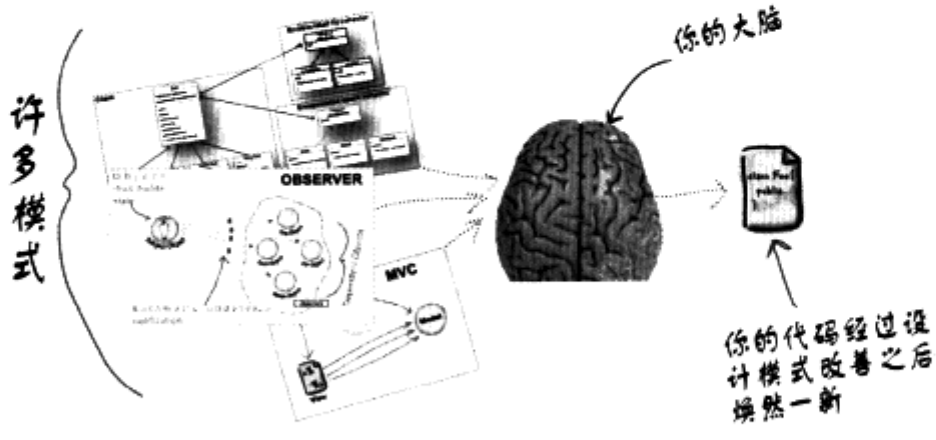
本文档由Linux公社 www.linuxidc.com 收集整理

设计模式入门

1 欢迎来到设计模式世界

有些人已经解决你的问题了。在本章，你将学到为何（以及如何）利用其他开发人员的智慧与经验。他们遭遇过相同的问题，也顺利地解决过这些问题。本章结束前，我们会先看看设计模式的用途与优点，再看一些关键的OO设计原则，并通过一个实例来了解模式是如何运作的。使用模式最好的方式是："把模式装进脑子里，然后在你的设计和已有的应用中，寻找何处可以使用它们。"以往是代码复用，现在是经验复用。

模拟鸭子应用	2
Joe想到继承	5
利用接口如何?	6
软件开发的不变真理	8
分开变化和不变部分	10
设计鸭子的行为	11
测试鸭子的代码	18
动态地设置行为	20
封装行为的大局观	22
"有一个"比"是一个"更好	23
策略模式	24
共享模式词汇的威力	28
我如何使用设计模式?	29
设计箱内的工具	32
习题解答	34



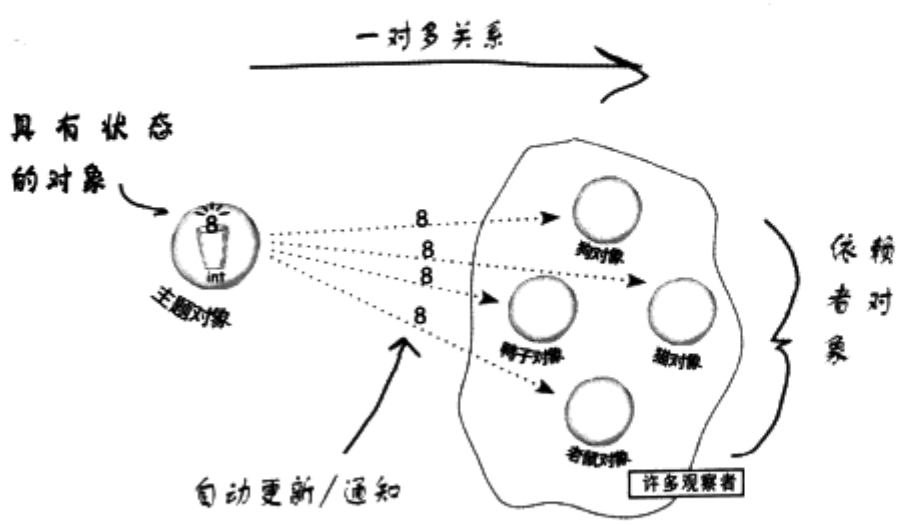
本文档由Linux公社 www.linuxidc.com 收集整理

2 观察者模式

让你的对象知悉现况

有趣的事情发生时，可千万别错过了！有一个模式可以帮你的对象知悉现况，不会错过该对象感兴趣的事。对象甚至在运行时可决定是否要继续被通知。观察者模式是JDK中使用最多的模式之一，非常有用。我们也会一并介绍一对多关系，以及松耦合（对，没错。我们说耦合）。有了观察者，你将会消息灵通。

气象观测站	39
认识观察者模式	44
出版者+订阅者=观察者模式	45
五分钟短剧：观察主题	48
定义观察者模式	51
松耦合的威力	53
设计气象站	56
实现气象站	57
使用Java内建的观察者模式	64
java.util.Observable的黑暗面	71
设计箱内的工具	74
习题解答	78



本文档由Linux公社 www.linuxidc.com 收集整理

3 装饰者模式

装饰对象

本章可以称为“给爱用继承的人一个全新的设计眼界”。我们即将再度探讨典型的继承滥用问题。你将在本章学到如何使用对象组合的方式，做到在运行时装饰类。为什么呢？一旦你熟悉了装饰的技巧，你将能够在不修改任何底层类代码的情况下，给你的（或别人的）对象赋予新的职责。

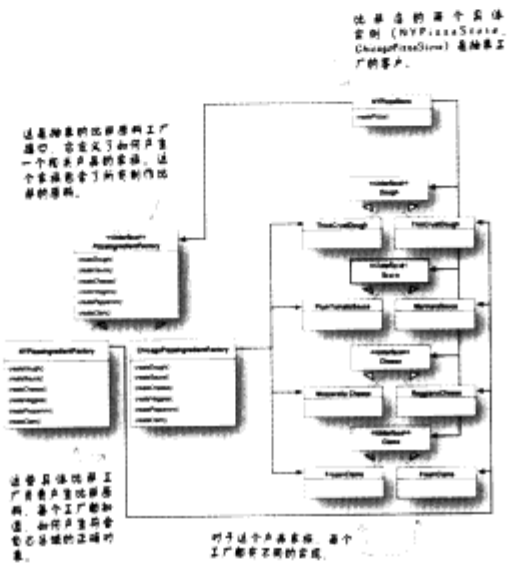
欢迎来到星巴兹咖啡 80
开放－关闭原则 86
认识装饰者模式 88
以装饰者构造饮料订单 89
定义装饰者模式 91
装饰饮料 92
写下星巴兹的代码 95
真实世界的装饰者：Java I/O 100
编写自己的Java I/O装饰者 102
设计箱内的工具 105
习题解答 106

本文档由Linux公社 www.linuxidc.com 收集整理

工厂模式

4 烘烤OO的精华

准备好开始烘烤某些松耦合的OO设计。除了使用new操作符之外，还有更多制造对象的方法。你将了解到实例化这个活动不应该总是公开地进行，也会认识到初始化经常造成“耦合”问题。你不希望这样，对吧？读下去，你将了解工厂模式如何从复杂的依赖中帮你脱困。



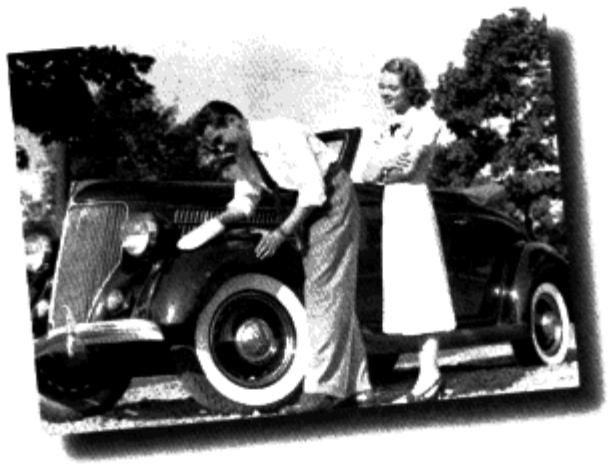
当看到“new”，就会想到“具体” 110
对象村比萨 112
封装创建对象的代码 114
建立一个简单比萨工厂 115
定义简单工厂 117
给比萨店使用的框架 120
允许子类做决定 121
让我们开一家比萨店吧 123
声明一个工厂方法 125
认识工厂方法模式 131
平行的类层级 132
定义工厂方法模式 134
一个很依赖的比萨店 137
看看对象依赖 138
依赖倒置原则 139
再回到比萨店…… 144
原料家族 145
建造原料工厂 146
看看抽象工厂 153
幕后花絮 154
定义抽象工厂模式 156
比较工厂方法和抽象工厂 160
设计箱内的工具 162
习题解答 164

本文档由Linux公社 www.linuxidc.com 收集整理

5 单件模式

独一无二的对象

单件模式：用来创建独一无二的，只能有一个实例的对象的入场券。告诉你一个好消息，单件模式的类图可以说是所有模式的类图中最简单的，事实上，它的类图上只有一个类！但是，可不要兴奋过头，尽管从类设计的视角来说很简单，但是实现上还是会遇到相当多的波折。所以，系好安全带，出发了！



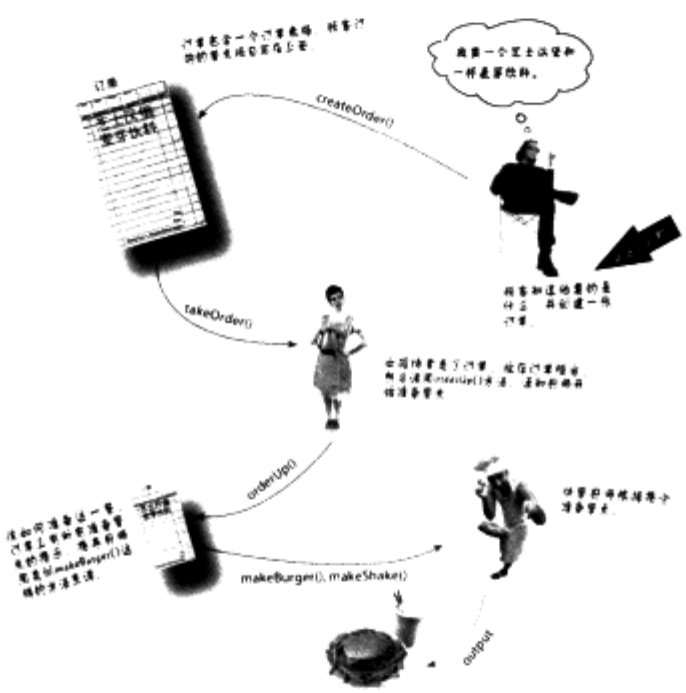
独一无二 170
小小单件 171
剖析经典的单件模式实现 173
单件的告白 174
巧克力工厂 175
定义单件模式 177
Hershey ~~Houston，~~我们遇到麻烦了…… 178
化身为JVM 179
处理多线程 180
单件Q&A 184
设计箱内的工具 186
习题解答 188

本文档由Linux公社 www.linuxidc.com 收集整理

6 命令模式

封装调用

在本章，我们将把封装带到一个全新的境界：把方法调用封装起来。没错，通过封装方法调用，我们可以把运算块包装成形。所以调用此运算的对象不需要关心事情是如何进行的，只要知道如何使用包装成形的方法来完成它就可以。通过封装方法调用，也可以做一些很聪明的事情，例如记录日志，或者重复使用这些封装来实现撤销。



巴斯特家电自动化公司 192
遥控器 193
看一下厂商的类 194
同时，回到餐厅…… 197
研究餐厅的交互 198
对象村餐厅的角色和职责 199
从餐厅到命令模式 201
第一个命令对象 203
定义命令模式 206
命令模式与遥控器 208
实现遥控器 210
逐步测试遥控器 212
写文档的时刻到了 215
使用状态实现撤销 220
每个遥控器都需要Party模式！ 224
使用宏命令 225
命令模式的更多用途：队列请求 228
命令模式的更多用途：日志请求 229
设计箱内的工具 230
习题解答 232

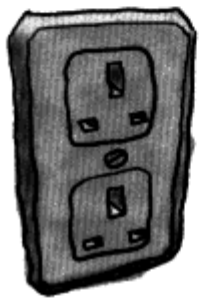
本文档由Linux公社 www.linuxidc.com 收集整理

7 适配器模式与外观模式

随遇而安

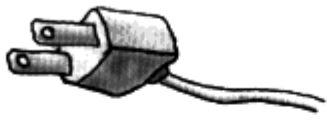
在本章，我们将要进行一项任务，其不可能的程度，简直就像是将一个方块放进一个圆洞中。听起来不可能？有了设计模式，就有可能。还记得装饰者模式吗？我们将对象包装起来，赋予它们新的职责。而现在则是以不同目的，包装某些对象：让它们的接口看起来不像自己而像是别的东西。为何要这样做？因为这样就可以在设计中，将类的接口转换成想要的接口，以便实现不同的接口。不仅如此，我们还要探讨另一个模式，将对象包装起来以简化其接口。

欧洲壁式插座

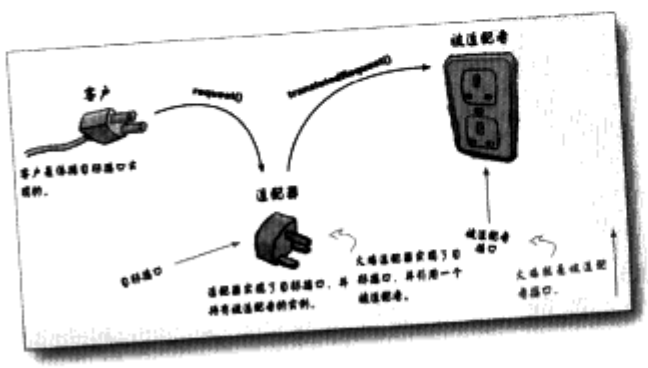


交流电适配器

标准的交流电插头



我们周围的适配器	236
面向对象适配器	237
适配器模式解析	241
定义适配器模式	243
对象和类的适配器	244
今夜话题：对象适配器和类适配器	247
真实世界的适配器	248
将枚举适配到迭代器	249
今夜话题：装饰者模式和适配器模式	252
甜蜜的家庭影院	255
灯光、相机、外观！	258
构造家庭影院外观	261
定义外观模式	264
“最少知识”原则	265
设计箱内的工具	270
习题解答	272

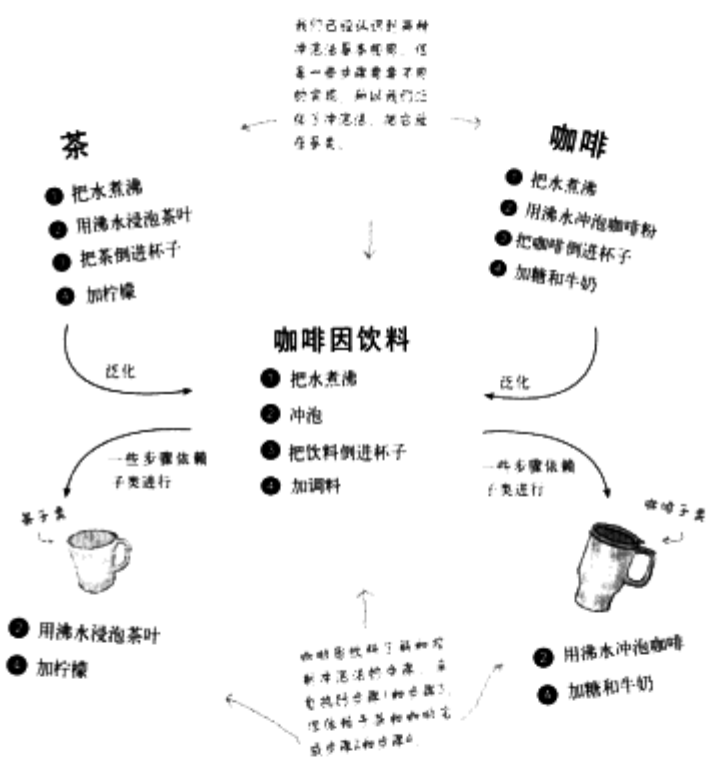


本文档由Linux公社 www.linuxidc.com 收集整理

8 模板方法模式

封装算法

直到目前，我们的议题都绕着封装转；我们已经封装了对象创建、方法调用、复杂接口、鸭子、比萨……接下来呢？我们将要深入封装算法块，好让子类可以在任何时候都可以将自己挂接进运算里。我们甚至会在本章学到一个受到好莱坞影响而启发的设计原则。



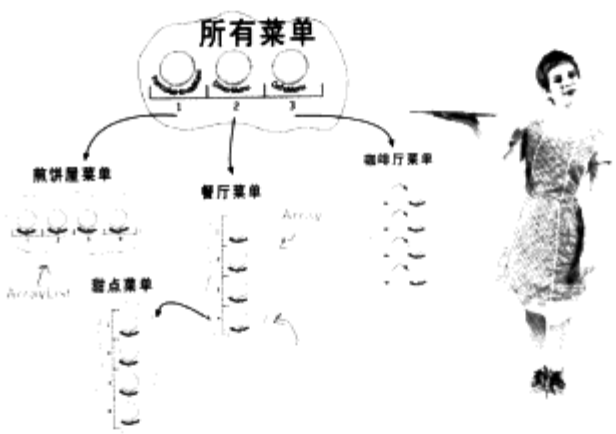
快速搞定几个咖啡和茶的类 277
抽取咖啡和茶 280
更进一步的设计…… 281
抽象prepareRecipe() 282
我们做了什么？ 285
认识模板方法 286
走，泡茶去 287
模板方法带给我们什么？ 288
定义模板方法模式 289
再靠近一点 290
对模板方法进行挂钩…… 292
使用钩子 293
咖啡？茶？执行测试程序 294
好莱坞原则 296
好莱坞原则和模板方法 297
荒野中的模板方法 299
用模板方法排序 300
来排序鸭子吧…… 301
比较鸭子 302
观察鸭子排序的内部运作 304
写一个Swing的窗口程序 306
Applet 307
今夜话题：模板方法和策略 308
设计箱内的工具 311
习题解答 312

本文档由Linux公社 www.linuxidc.com 收集整理

9 迭代器与组合模式

管理良好的集合

有许多种方法可以把对象堆起来成为一个集合。你可以把它们放进数组、堆栈、列表或者是散列表（Hashtable）中，这是你的自由。每一种都有它自己的优点和适合的使用时机，但总有一个时候，你的客户想要遍历这些对象，而当他这么做时，你打算让客户看到你的实现吗？我们当然希望最好不要！这太不专业了。没关系，不要为你的工作担心，你将在本章中学习如何能让客户遍历你的对象而又无法窥视你存储对象的方式；也将学习如何创建一些对象超集合（super collection），能够一口气就跳过某些让人望而生畏的数据结构。你还将学到一些关于对象职责的知识。



对象村餐厅和对象村煎饼屋合并了 316
比较菜单的实现 318
可以封装遍历吗？ 323
认识迭代器模式 325
在餐厅菜单中加入一个迭代器 326
鸟瞰目前的设计 331
利用java.util.Iterator来清理 333
这为我们带来什么好处？ 335
定义迭代器模式 336
单一责任 339
迭代器与集合 348
Java 5 的迭代器和集合 349
正当我们认为这很安全的时候…… 353
定义组合模式 356
利用组合设计菜单 359
实现组合菜单 362
闪回到迭代器 368
空迭代器 372
迭代器和组合凑在一起的魔力…… 374
设计箱内的工具 380
习题解答 381

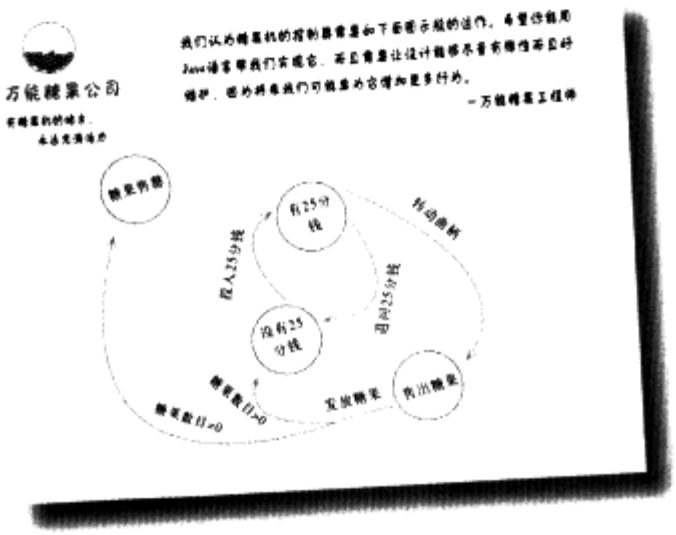
本文档由Linux公社 www.linuxidc.com 收集整理

10 状态模式

事物的状态

基本常识：策略模式和状态模式是双胞胎，在出生时才分开。

你已经知道了，策略模式是围绕可以互换的算法来创建成功业务的。然而，状态走的是更崇高的路，它通过改变对象内部的状态来帮助对象控制自己的行为。它常常告诉它的对象客户“跟着我念：我很棒，我很聪明，我最优秀了……”



如何实现状态？（办公室隔间对话） 387
状态机101 388
状态机代码的第一个版本 390
该来的躲不掉……变更请求！ 394
混乱的状态…… 396
定义状态接口和类 399
实现我们的状态类 401
重新改造糖果机 402
定义状态模式 410
状态 vs.策略模式 411
精神检查…… 417
我们差点儿忘了！ 420
设计箱内的工具 423
习题解答 424

本文档由Linux公社 www.linuxidc.com 收集整理

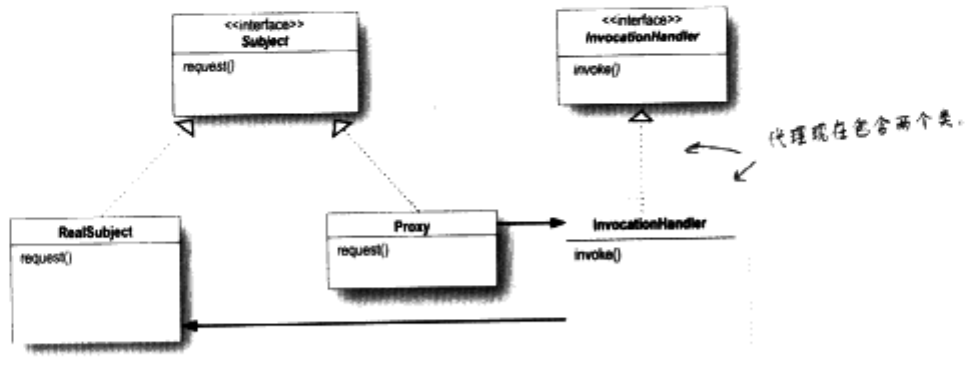
11

代理模式

控制对象访问

玩过扮白脸、扮黑脸的游戏吗？你是一个白脸，提供很好且很友善的服务，但是你不希望每个人都叫你做事，所以找了黑脸控制对你的访问。这就是代理要做的：控制和管理访问。就像你将看到的，代理的方式有许多种。代理以通过Internet为它们的代理对象搬运的整个方法调用而出名，它也可以代替某些懒惰的对象做一些事情。

监控糖果机	430
远程代理的角色	434
RMI浏览	437
GumballMachine远程代理	450
代理幕后花絮	458
定义代理模式	460
准备虚拟代理	462
设计CD封面虚拟代理	464
虚拟代理的幕后花絮	470
使用Java API的代理	474
五分钟短剧：保护主题	478
创建动态代理	479
代理动物园	488
设计箱内的工具	491
习题解答	492

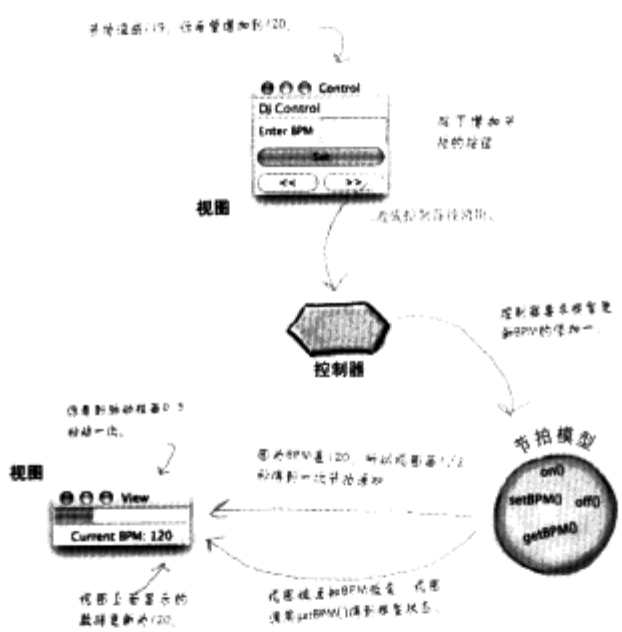


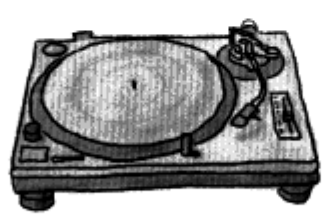
本文档由Linux公社 www.linuxidc.com 收集整理

12 复合模式

模式中的模式

谁料得到模式居然可以携手合作？ 你已经见识过围炉夜话的火爆场面（幸好，出版社事先请我们删除“死神来访”模式的篇章，好让本书不需附上“12岁以下读者必须家长陪同阅读”的警告标语，所以你没见识到闹出人命的那一集围炉夜话），谁料得到模式居然可以携手合作？这实在是太意外了。信不信由你，有一些威力强大的OO设计同时使用多个设计模式。准备让你的模式技巧进入下一个层次，现在是复合模式的时间。





复合模式	500
与鸭子重聚	501
加入一个适配器	504
加入一个装饰者	506
加入一个工厂	508
加入一个组合和一个迭代器	513
加入一个观察者	516
模式概览	523
鸭瞰：类图	524
模型－视图－控制器之歌	526
设计模式是MVC的钥匙	528
戴着模式的有色眼镜看MVC	532
利用MVC控制节拍……	534
模型	537
视图	539
控制器	542
探索策略	545
适配模型	546
现在我们准备写HeartController	547
MVC与Web	549
设计模式和Model 2	557
设计箱内的工具	560
习题解答	561

本文档由Linux公社 www.linuxidc.com 收集整理

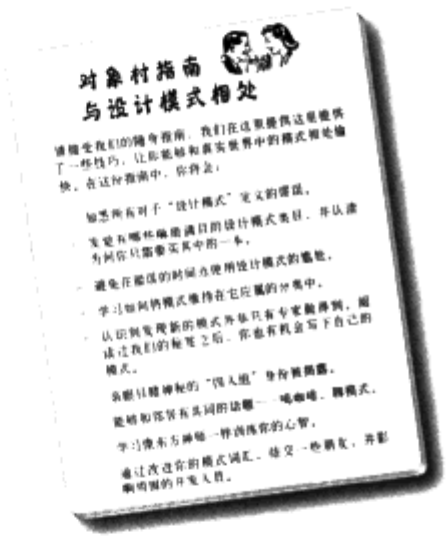
13

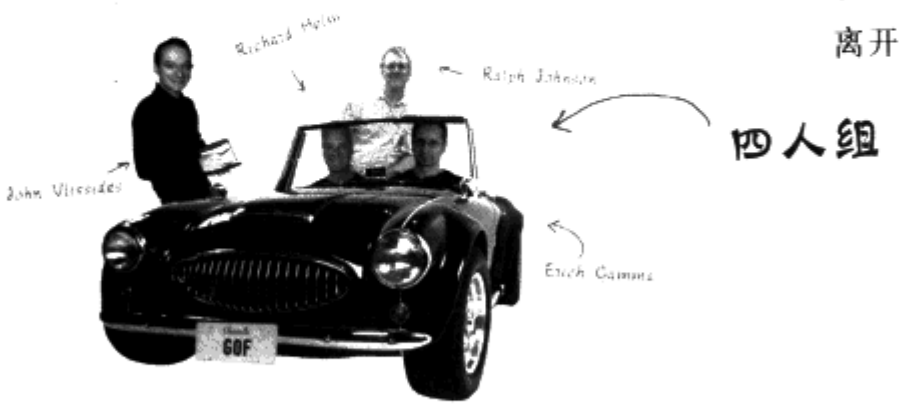
与设计模式相处

真实世界中的模式

现在你已经准备好迎接一个充满设计模式的崭新世界。但是，在你打开所有的机会大门之前，我们需要告诉你一些即将在真实世界中遇到的细节——没错，外面的世界比对象村来得复杂。来吧！从下页开始，我们会指引你的方向……

对象村指南	578
定义设计模式	579
更近地观察设计模式的定义	581
愿力与你同在	582
模式类目	583
如何创建模式	586
想当一个设计模式作家吗？	587
组织设计模式	589
用模式思考	594
使用模式的心智	597
别忘了共享词汇的威力	599
共享词汇的五种方式	600
和四人组一同巡游对象村	601
你的旅途刚刚开始……	602
其他设计模式资源	603
模式动物园	604
以反模式歼灭恶势力	606
设计箱内的工具	608
离开对象村……	609

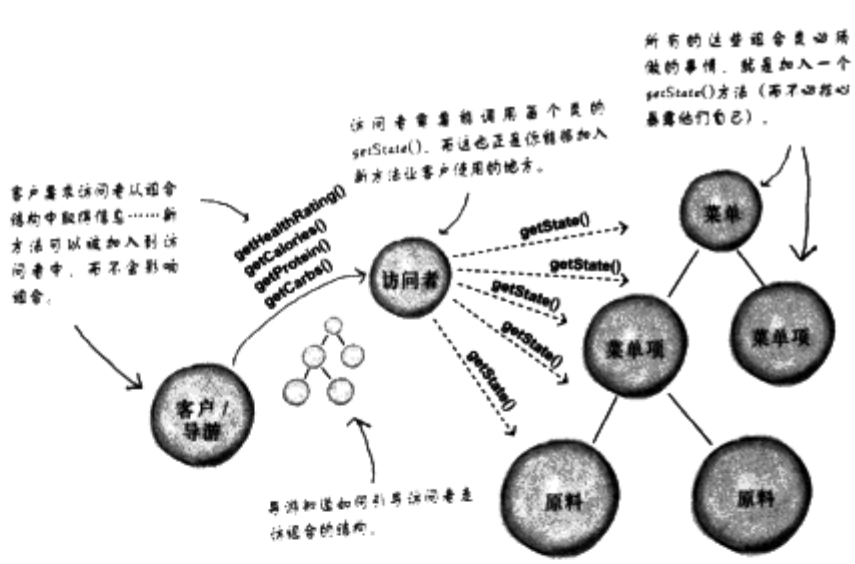




本文档由Linux公社 www.linuxidc.com 收集整理

A 附录A：剩下的模式

并非每个人都广受欢迎。过去10年来，事情改变了许多。自从《设计模式：可复用面向对象软件的基础》一书出版之后，开发人员就开始大量地采用这些模式。我们在此附录中所介绍的模式，都是成熟、典型、正式的四人组模式，只不过可能不像前面章节所探索的模式那么经常地被使用。但是这些模式本身也有相当可取之处，而如果你遇到了合适的情形，也应当毫不犹豫地采用它们。我们在此的目标，是希望能够让你通盘了解这些模式的意义。



桥接 612
生成器 614
责任链 616
蝇量 618
解释器 620
中介者 622
备忘录 624
原型 626
访问者 628

i 索引

631

本文档由Linux公社 www.linuxidc.com 收集整理

1 设计模式入门

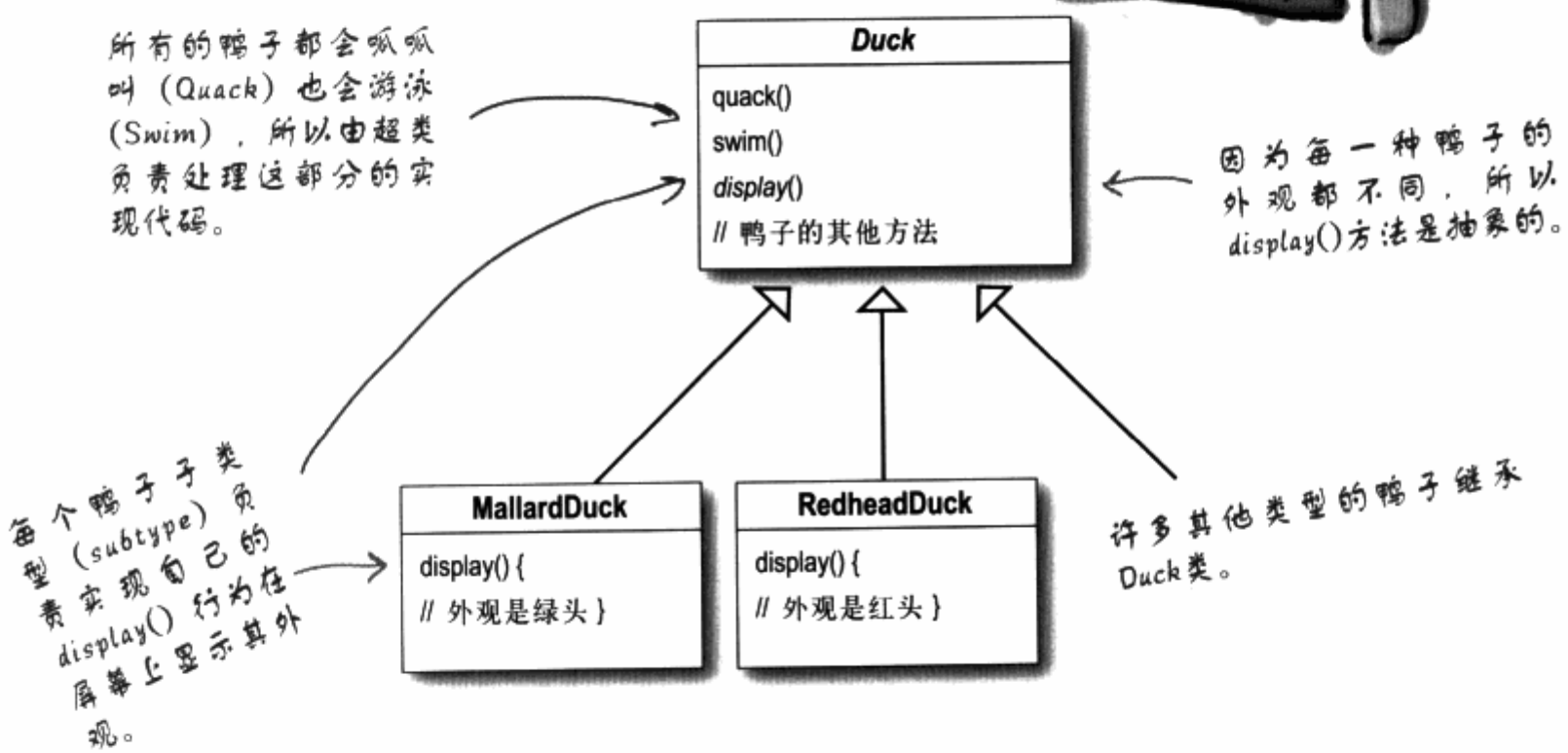
欢迎来到设计模式世界

有些人已经解决你的问题了。 在本章，你将学到为何（以及如何）利用其他开发人员的经验与智慧。他们遭遇过相同的问题，也顺利地解决过这些问题。本章结束前，我们会看看设计模式的用途与优点，再看一些关键的OO设计原则，并通过一个实例来了解模式是如何运作。使用模式最好的方式是："把模式装进脑子里，然后在你的设计和已有的应用中，寻找何处可以使用它们。"以往是代码复用，现在是经验复用。

本文档由Linux公社 www.linuxidc.com 收集整理

先从简单的模拟鸭子应用做起

Joe上班的公司做了一套相当成功的模拟鸭子游戏：SimUDuck。游戏中会出现各种鸭子，一边游泳戏水，一边呱呱叫。此系统的内部设计使用了标准的OO技术，设计了一个鸭子超类（Superclass），并让各种鸭子继承此超类。

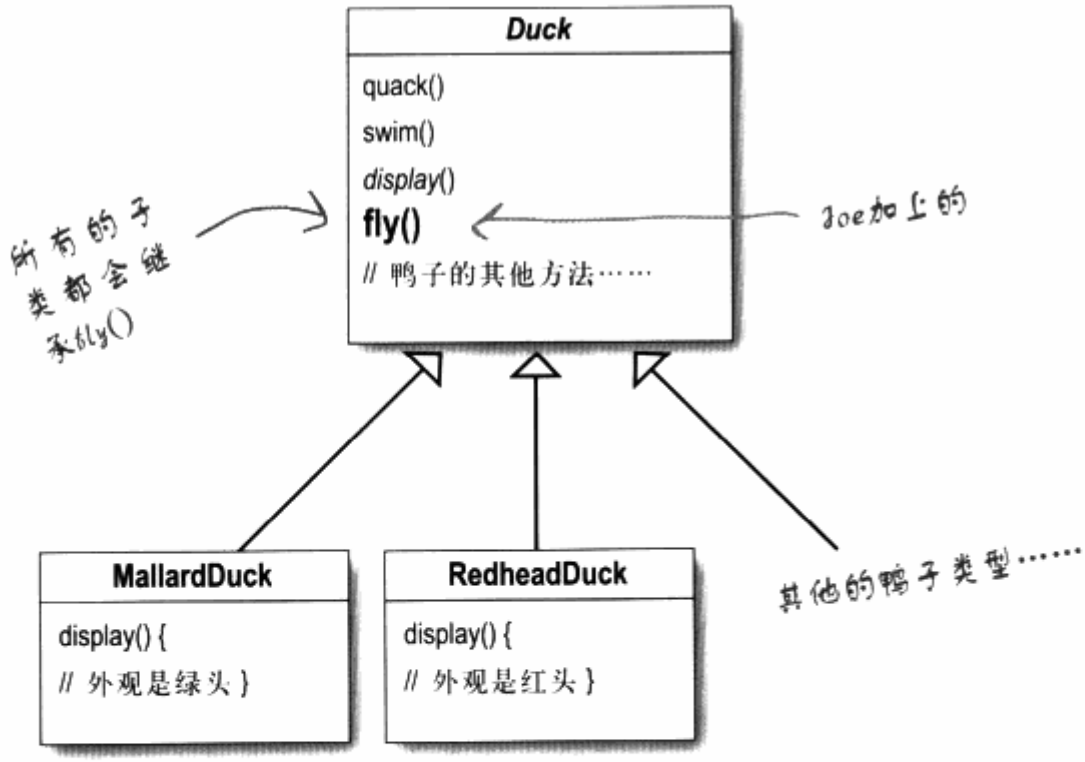


去年，公司的竞争压力加剧。在为期一周的高尔夫假期兼头脑风暴会议之后，公司主管认为该是创新的时候了，他们需要在“下周”毛伊岛股东会议上展示一些“真正”让人印象深刻的东西来振奋人心。

本文档由Linux公社 www.linuxidc.com 收集整理

现在我们得让鸭子能飞

主管们确定，此模拟程序需要会飞的鸭子来将竞争者抛在后头。当然，在这个时候，Joe的经理拍胸脯告诉主管们，Joe只需要一个星期就可以搞定。“毕竟，Joe是一个OO程序员……这有什么困难？”



本文档由Linux公社 www.linuxidc.com 收集整理

但是，可怕的问题发生了……

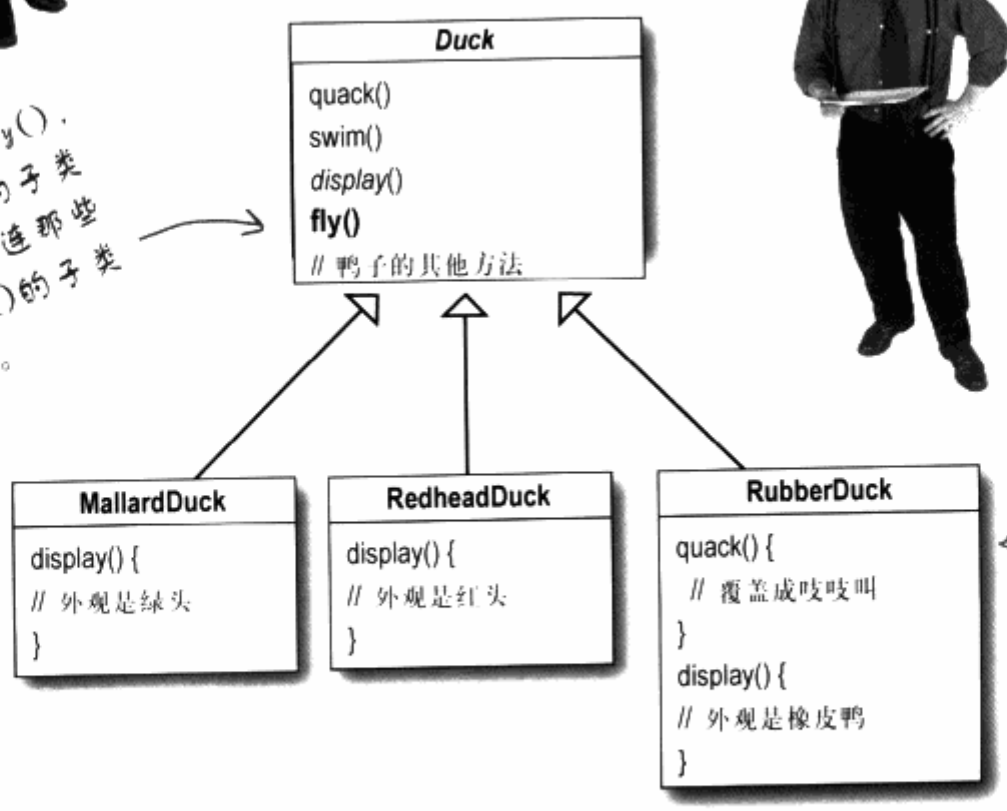
怎么回事？

Joe忽略了一件事：并非Duck所有的子类都会飞。Joe在Duck超类中加上新的行为，会使得某些并不适合该行为的子类也具有该行为。现在可好了！SimUDuck程序中有了一个无生命的会飞的东西。

对代码所做的局部修改，影响层面可不只是局部（会飞的橡皮鸭）！

他体会到了一件事：当涉及“维护”时，为了“复用”（reuse）目的而使用继承，结局并不完美。

在超类中加上fly()，就会导致所有的子类都具备fly()，连那些不该具备fly()的子类也无法免除。



橡皮鸭子不会呱呱叫，所以把quack()的定义覆盖成“吱吱叫”（squeak）。

本文档由Linux公社 www.linuxidc.com 收集整理

Joe想到继承

Sharpen your pencil

利用继承来提供Duck的行为，这会导致下列哪些缺点？（多选）

- ❑ A. 代码在多个子类中重复。
- ❑ B. 运行时的行为不容易改变。
- ❑ C. 我们不能让鸭子跳舞。
- ❑ D. 很难知道所有鸭子的全部行为。
- ❑ E. 鸭子不能同时又飞又叫。
- ❑ F. 改变会牵一发动全身，造成其他鸭子不想要的改变。

本文档由Linux公社 www.linuxidc.com 收集整理

利用接口如何？

Joe认识到继承可能不是答案，因为他刚刚拿到来自主管的备忘录，希望以后每六个月更新产品（至于更新的方法，他们还没想到）。Joe知道规格会常常改变，每当有新的鸭子子类出现，他就要被迫检查并可能需要覆盖fly()和quark()……这简直是无穷无尽的噩梦。

所以，他需要一个更清晰的方法，让“某些”（而不是全部）鸭子类型可飞或可叫。

你觉得这个设计如何？

本文档由Linux公社 www.linuxidc.com 收集整理

这真是一个超笨的主意，你没发现这么一来重复的代码会变多吗？如果你认为覆盖几个方法就算是差劲，那么对于48个Duck的子类都要稍微修改一下飞行的行为，你又怎么说？！

如果你是Joe，你要怎么办？

我们知道，并非“所有”的子类都具有飞行和呱呱叫的行为，所以继承并不是适当的解决方式。虽然Flyable与Quackable可以解决“一部分”问题（不会再有会飞的橡皮鸭），但是却造成代码无法复用，这只能算是从一个恶梦跳进另一个恶梦。甚至，在会飞的鸭子中，飞行的动作可能还有多种变化……

此时，你可能正期盼着设计模式能骑着白马来解救你离开苦难的一天。但是，如果直接告诉你答案，这有什么乐趣？我们会用老方法找出一个解决之道：“采用良好的OO软件设计原则”。

如果能有一种建立软件的方法，好让我们可以用一种对既有的代码影响最小的方式来修改软件该有多好。我们就可以花较少时间重做代码，而多让程序去做更酷的事……

本文档由Linux公社 www.linuxidc.com 收集整理

软件开发的一个不变真理

好吧！在软件开发上，有什么是你可以深信不疑的？

不管你在何处工作，构建些什么，用何种编程语言，在软件开发上，一直伴随你的那个不变真理是什么？

CHANGE

（用镜子来看答案）

不管当初软件设计得多好，一段时间之后，总是需要成长与改变，
否则软件就会“死亡”。

驱动改变的因素很多。找出你的应用中需要改变代码的原因，一一列出来。（我们写下了一些我们的原因，给你起个头。）

我们的顾客或用户需要别的东西，或者想要新功能。

我的公司决定采用别的数据库产品，又从另一家厂商买了数据，这造成数据格式不兼容。唉！

本文档由Linux公社 www.linuxidc.com 收集整理

把问题归零……

现在我们知道使用继承并不能很好地解决问题，因为鸭子的行为在子类里不断地改变，并且让所有的子类都有这些行为是不恰当的。Flyable与Quackable接口一开始似乎还挺不错，解决了问题（只有会飞的鸭子才继承Flyable），但是Java接口不具有实现代码，所以继承接口无法达到代码的复用。这意味着：无论何时你需要修改某个行为，你必须得往下追踪并在每一个定义此行为的类中修改它，一不小心，可能会造成新的错误！

幸运的是，有一个设计原则，恰好适用于此状况。

设计原则

找出应用中可能需要变化之处，把它们独立出来，不要和那些不需要变化的代码混在一起。

这是我们的第一个设计原则，以后还有更多原则会陆续在本书中出现。

换句话说，如果每次新的需求一来，都会使某方面的代码发生变化，那么你就可以确定，这部分的代码需要被抽出来，和其他稳定的代码有所区分。

下面是这个原则的另一种思考方式：“把会变化的部分取出并封装起来，以便以后可以轻易地改动或扩充此部分，而不影响不需要变化的其他部分”。

这样的概念很简单，几乎是每个设计模式背后的精神所在。所有的模式都提供了一套方法让“系统中的某部分改变不会影响其他部分”。

好，该是把鸭子的行为从Duck类中取出的时候了！

把会变化的部分取出并“封装”起来，好让其他部分不会受到影响。

结果如何？代码变化引起的不经意后果变少，系统变得更有弹性。

本文档由Linux公社 www.linuxidc.com 收集整理

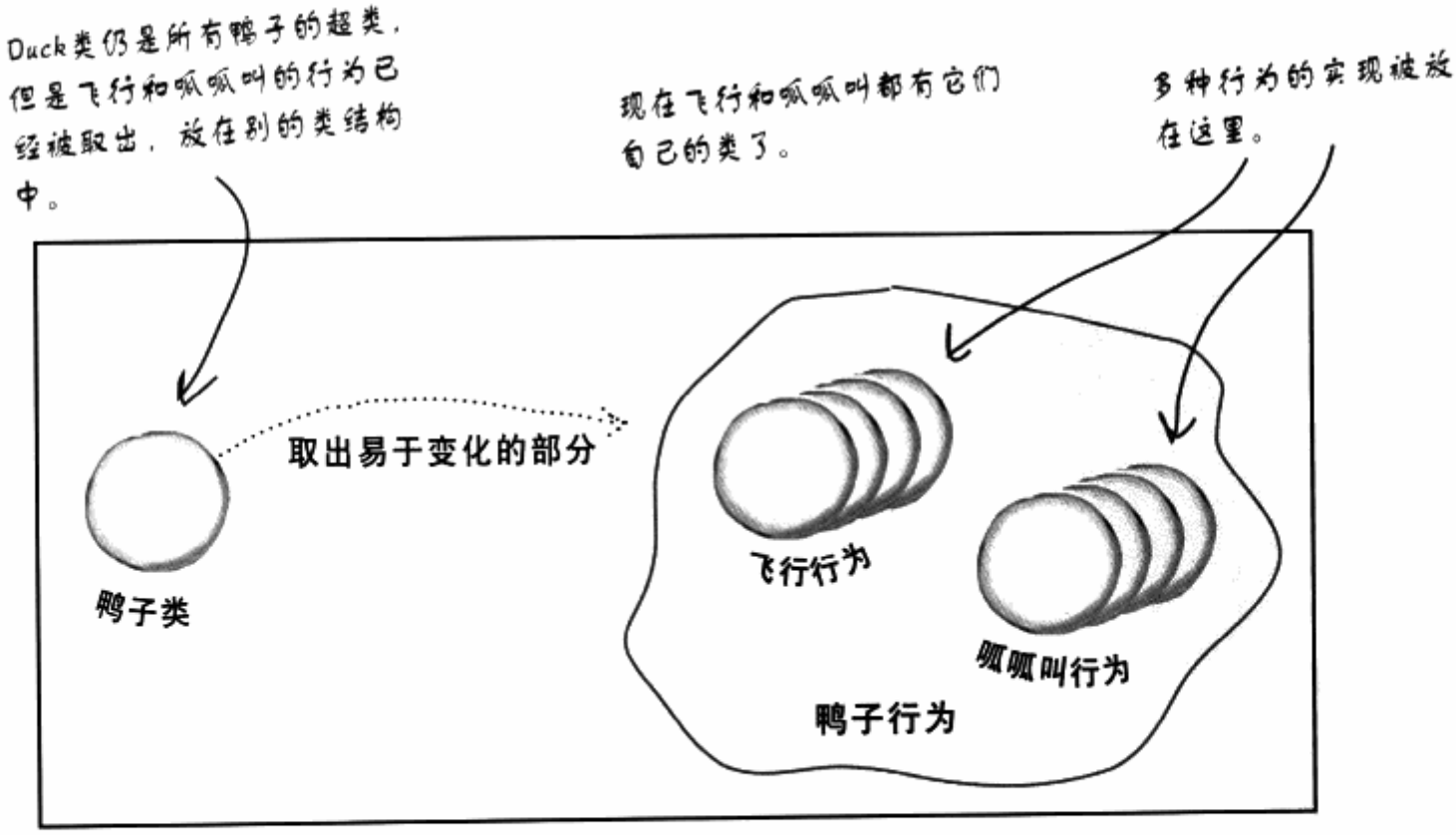
分开变化和不会变化的部分

从哪里开始呢？就我们目前所知，除了fly()和quack()的问题之外，Duck类还算一切正常，似乎没有特别需要经常变化或修改的地方。所以，除了某些小改变之外，我们不打算对Duck类做太多处理。

现在，为了要分开“变化和不会变化的部分”，我们准备建立两组类（完全远离Duck类），一个是“fly”相关的，一个是“quack”相关的，每一组类将实现各自的动作。比方说，我们可能有一个类实现“呱呱叫”，另一个类实现“吱吱叫”，还有一个类实现“安静”。

我们知道Duck类内的fly()和quack()会随着鸭子的不同而改变。

为了要把这两个行为从Duck类中分开，我们将把它们从Duck类中取出来，建立一组新类来代表每个行为。



本文档由Linux公社 www.linuxidc.com 收集整理

设计鸭子的行为

如何设计那组实现飞行和呱呱叫的行为的类呢？

我们希望一切能有弹性，毕竟，正是因为一开始鸭子行为没有弹性，才让我们走上现在这条路。我们还想能够“指定”行为到鸭子的实例。比方说，我们想要产生一个新的绿头鸭实例，并指定特定“类型”的飞行行为给它。干脆顺便让鸭子的行为可以动态地改变好了。换句话说，我们应该在鸭子类中包含设定行为的方法，这样就可以在“运行时”动态地“改变”绿头鸭的飞行行为。

有了这些目标要实现，接着看看第二个设计原则：

设计原则

针对接口编程，而不是针对实现编程。

我们利用接口代表每个行为，比方说，FlyBehavior与QuackBehavior，而行为的每个实现都将实现其中的一个接口。

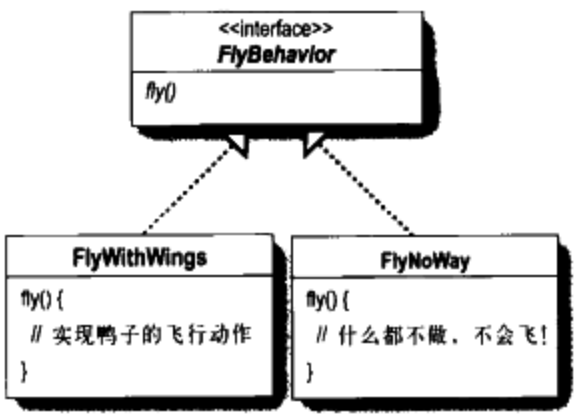
所以这次鸭子类不会负责实现Flying与Quacking接口，反而是由我们制造一组其他类专门实现FlyBehavior与QuackBehavior，这就称为“行为”类。由行为类而不是Duck类来实现行为接口。

这样的做法迥异于以往，以前的做法是：行为来自Duck超类的具体实现，或是继承某个接口并由子类自行实现而来。这两种做法都是依赖于“实现”，我们被实现绑得死死的，没办法更改行为（除非写更多代码）。

在我们的新设计中，鸭子的子类将使用接口（FlyBehavior与QuackBehavior）所表示的行为，所以实际的“实现”不会被绑死在鸭子的子类中。（换句话说，特定的具体行为编写在实现了FlyBehavior与QuakcBehavior的类中）。

从现在开始，鸭子的行为将被放在分开的类中，此类专门提供某行为接口的实现。

这样，鸭子类就不再需要知道行为的实现细节。



欢迎点击这里的链接进入精彩的Linux公社 网站

Linux公社（www.Linuxidc.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

Linux公社是专业的Linux系统门户网站，实时发布最新Linux资讯，包括Linux、Ubuntu、Fedora、RedHat、红旗Linux、Linux教程、Linux认证、SUSE Linux、Android、Oracle、Hadoop、CentOS、MySQL、Apache、Nginx、Tomcat、Python、Java、C语言、OpenStack、集群等技术。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

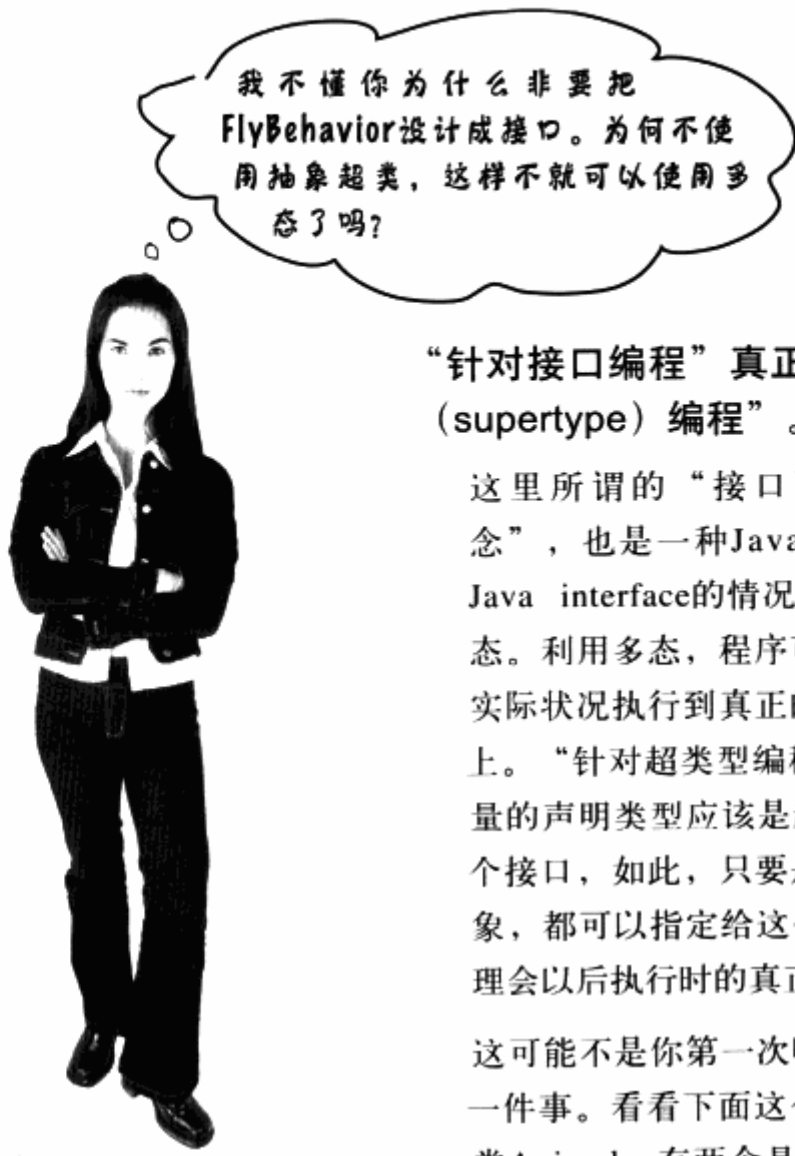
Linux公社 主站网址：www.linuxidc.com 旗下网站：www.linuxidc.net

包括：**Ubuntu 专题** **Fedora 专题** **Android 专题** **Oracle 专题** **Hadoop 专题** **RedHat 专题** **SUSE 专题** **红旗 Linux 专题** **CentOS 专题**

Linux公社 www.Linuxidc.com

Linux 公社微信公众号：linuxidc_com

本文档由Linux公社 www.linuxidc.com 收集整理



“针对接口编程”真正的意思是“针对超类型（supertype）编程”。

这里所谓的“接口”有多个含义，接口是一个“概念”，也是一种Java的interface构造。你可以在不涉及Java interface的情况下，“针对接口编程”，关键就在多态。利用多态，程序可以针对超类型编程，执行时会根据实际状况执行到真正的行为，不会被绑死在超类型的行为上。“针对超类型编程”这句话，可以更明确地说成“变量的声明类型应该是超类型，通常是一个抽象类或者是一个接口，如此，只要是具体实现此超类型的类所产生的对象，都可以指定给这个变量。这也意味着，声明类时不用理会以后执行时的真正对象类型！”

这可能不是你第一次听到，但是请务必注意我们说的是同一件事。看看下面这个简单的多态例子：假设有一个抽象类Animal，有两个具体的实现（Dog与Cat）继承Animal。做法如下：

“针对实现编程”

```
Dog d = new Dog( );
d.bark( );
```

声明变量“d”为Dog类型（是Animal的具体实现），会造成我们必须针对具体实现编码。

但是，“针对接口/超类型编程”做法会如下：

```
Animal animal = new Dog( );
animal.makeSound( );
```

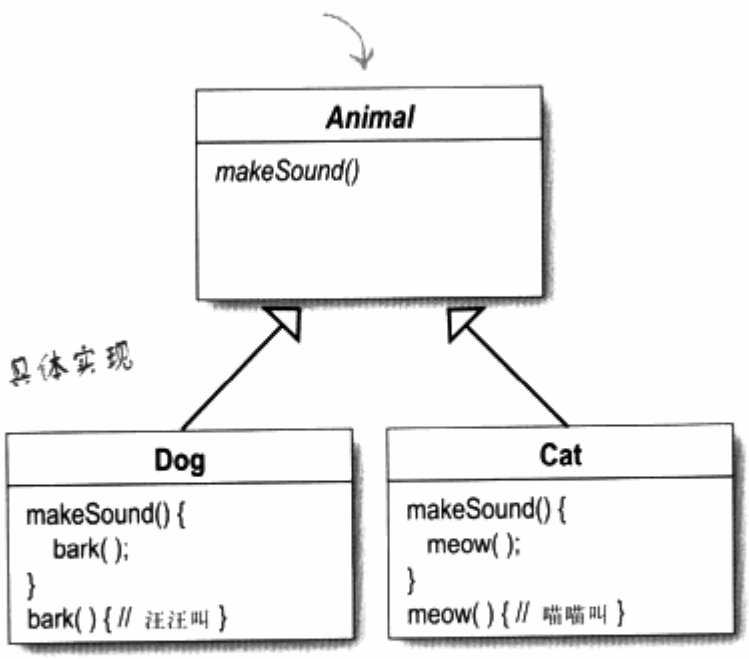
我们知道该对象是狗，但是我们现在利用animal进行多态的调用。

更棒的是，子类实例化的动作不再需要在代码中硬编码，例如new Dog()，而是“在运行时才指定具体实现的对象”。

```
a = getAnimal( );
a.makeSound( );
```

我们不知道实际的子类型是“什么”……我们只关心它知道如何正确地进行makeSound()的动作就够了。

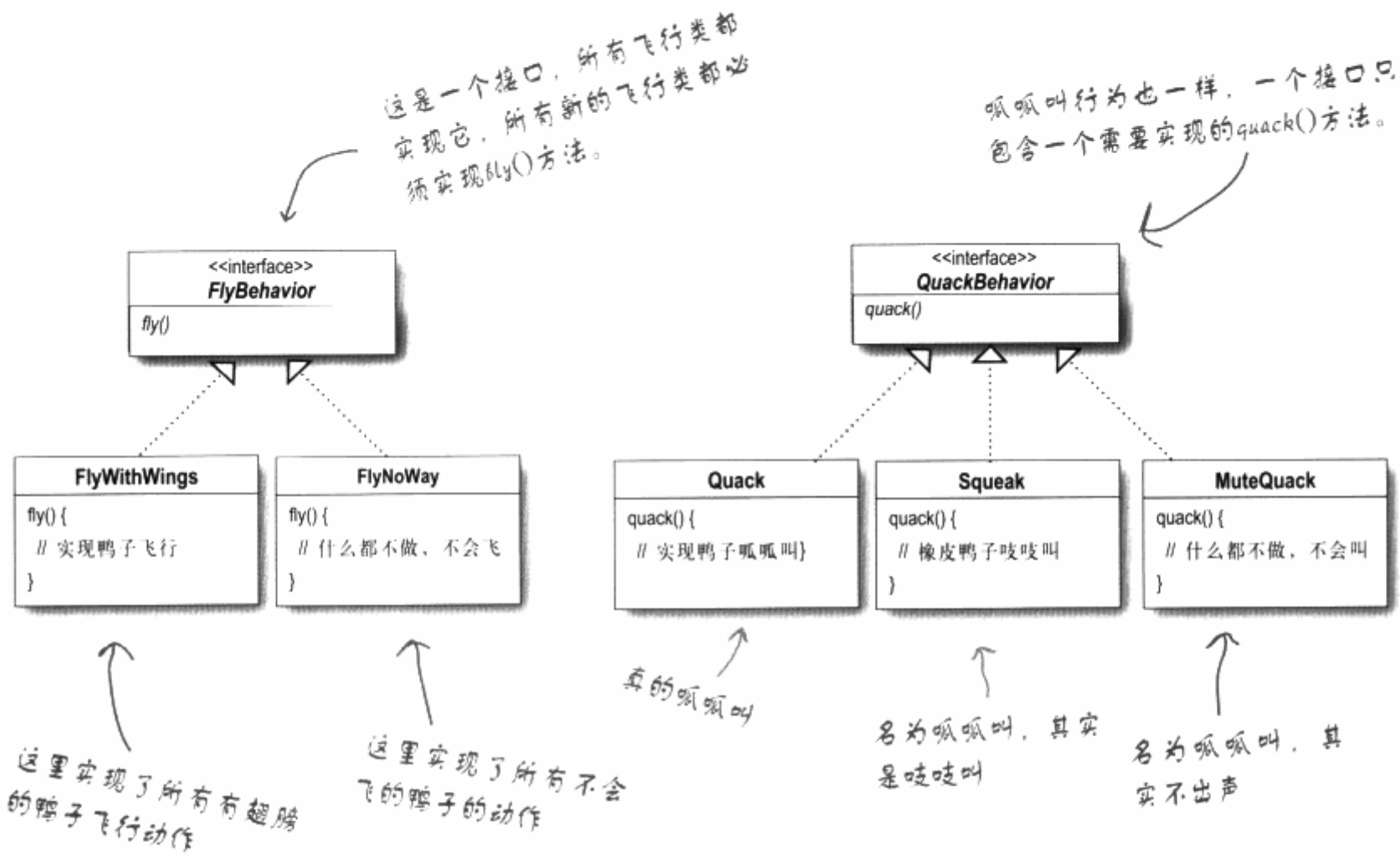
抽象超类型可以是抽象类“或”接口。



本文档由Linux公社 www.linuxidc.com 收集整理

实现鸭子的行为

在此，我们有两个接口，FlyBehavior和QuackBehavior，还有它们对应的类，负责实现具体的行为：



这样的设计，可以让飞行和呱呱叫的动作被其他的对象复用，因为这些行为已经与鸭子类无关了。

而我们可以新增一些行为，不会影响到既有的行为类，也不会影响“使用”到飞行行为的鸭子类。

这么一来，有了继承的“复用”好处，却没有继承所带来的包袱。

本文档由Linux公社 www.linuxidc.com 收集整理

there are no Dumb Questions

问： 我是不是一定要先把系统做出来，再看看有哪些地方需要变化，然后才回头去把这些地方分离&封装？

答： 不尽然。通常在你设计系统时，预先考虑到有哪些地方未来可能需要变化，于是提前在代码中加入这些弹性。你会发现，原则与模式可以应用在软件开发生命周期的任何阶段。

问： Duck是不是也该设计成一个接口？

答： 在本例中，这么做并不恰当。如你所见的，我们已经让一切都整合妥当，而且让Duck成为一个具体类，这样可以让衍生的特定类（例如绿头鸭）具有Duck共同的属性和方法。我们已经从Duck的继承结构中删除了变化的部分，原先的问题都已经解决了，所以不需要把Duck设计成接口。

问： 用一个类代表一个行为，感觉似乎有点奇怪。类不是应该代表某种“东西”吗？类不是应该同时具备状态“与”行为吗？

答： 在OO系统中，是的，类代表的东西一般都是既有状态（实例变量）又有方法。只是在本例中，碰巧“东西”是个行为。但是即使是行为，也仍然可以有状态和方法，例如，飞行的行为可以具有实例变量，记录飞行行为的属性（每秒翅膀拍动几下、最大高度和速度等）。

Sharpen your pencil

❶ 使用我们的新设计，如果你要加上一个火箭动力的飞行动作到SimUDuck 系统中，你该怎么做？

❷ 除了鸭子之外，你能够想出有什么类会需要用到呱呱叫的行为？

答案：

1）建立一个FlyRocketPowered 类，实现FlyBehavior 接口。

2）例如：鸭鸣器（DuckCall）（一种会产生鸭叫声的装置）。

本文档由Linux公社 www.linuxidc.com 收集整理

整合鸭子的行为

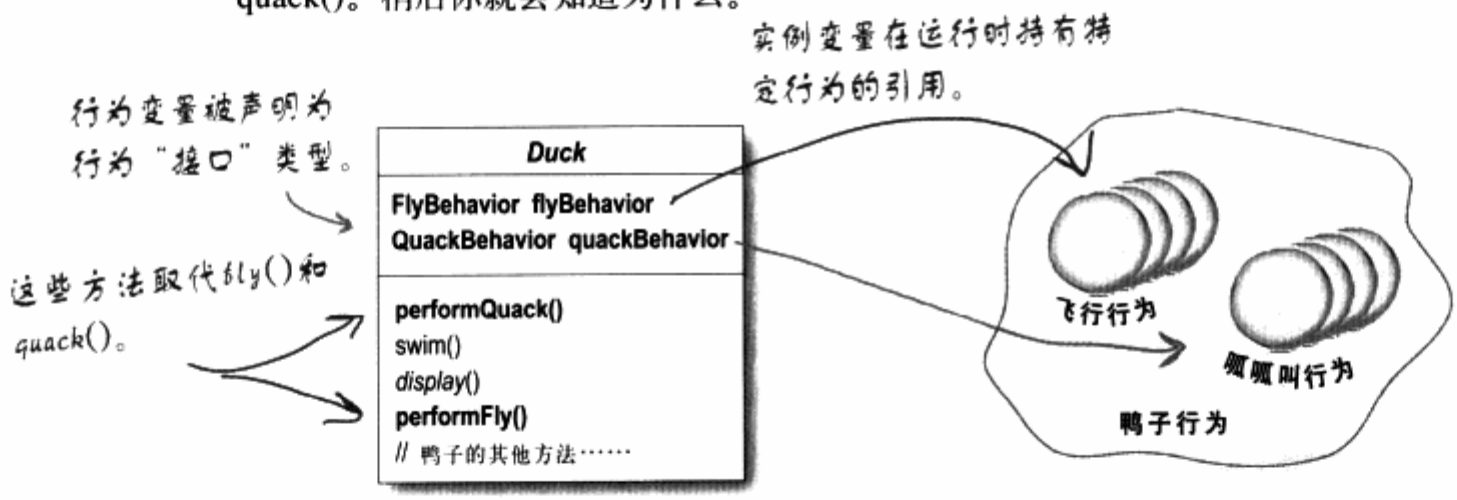
关键在于，鸭子现在会将飞行和呱呱叫的动作“委托”（delegate）别人处理，而不是使用定义在Duck类（或子类）内的呱呱叫和飞行方法。

做法是这样的：

❶ **首先，在Duck类中“加入两个实例变量”**，分别为“flyBehavior”与“quackBehavior”，声明为接口类型（而不是具体类实现类型），每个鸭子对象都会动态地设置这些变量以在运行时引用正确的行为类型（例如：FlyWithWings、Squeak等）。

我们也必须将Duck类与其所有子类中的fly()与quack()删除，因为这些行为已经被搬到FlyBehavior与QuackBehavior类中了。

我们用两个相似的方法performFly()和performQuack()取代Duck类中的fly()与quack()。稍后你就会知道为什么。



❷ **现在，我们来实现performQuack()：**

```
public class Duck {
   QuackBehavior quackBehavior;
   // 还有更多

   public void performQuack() {
     quackBehavior.quack();
   }
}
```

每只鸭子都会引用实现QuackBehavior接口的对象。

鸭子对象不亲自处理呱呱叫行为，而是委托给quackBehavior引用的对象。

很容易，是吧？想进行呱呱叫的动作，Duck对象只要叫quackBehavior对象去呱呱叫就可以了。在这部分的代码中，我们不在乎quackBehavior接口的对象到底是什么，我们只关心该对象知道如何进行呱呱叫就够了。

本文档由Linux公社 www.linuxidc.com 收集整理

更多的整合……

❸ 好吧！现在来关心“如何设定flyBehavior与quackBehavior的实例变量”。看看MallardDuck类：

```
public class MallardDuck extends Duck {

    public MallardDuck() {
        quackBehavior = new Quack();
        flyBehavior = new FlyWithWings();
    }

    public void display() {
        System.out.println("I'm a real Mallard duck");
    }
}
```

绿头鸭使用Quack类处理呱呱叫，所以当performQuack()被调用时，叫的职责被委托给Quack对象，而我们得到了真正的呱呱叫。

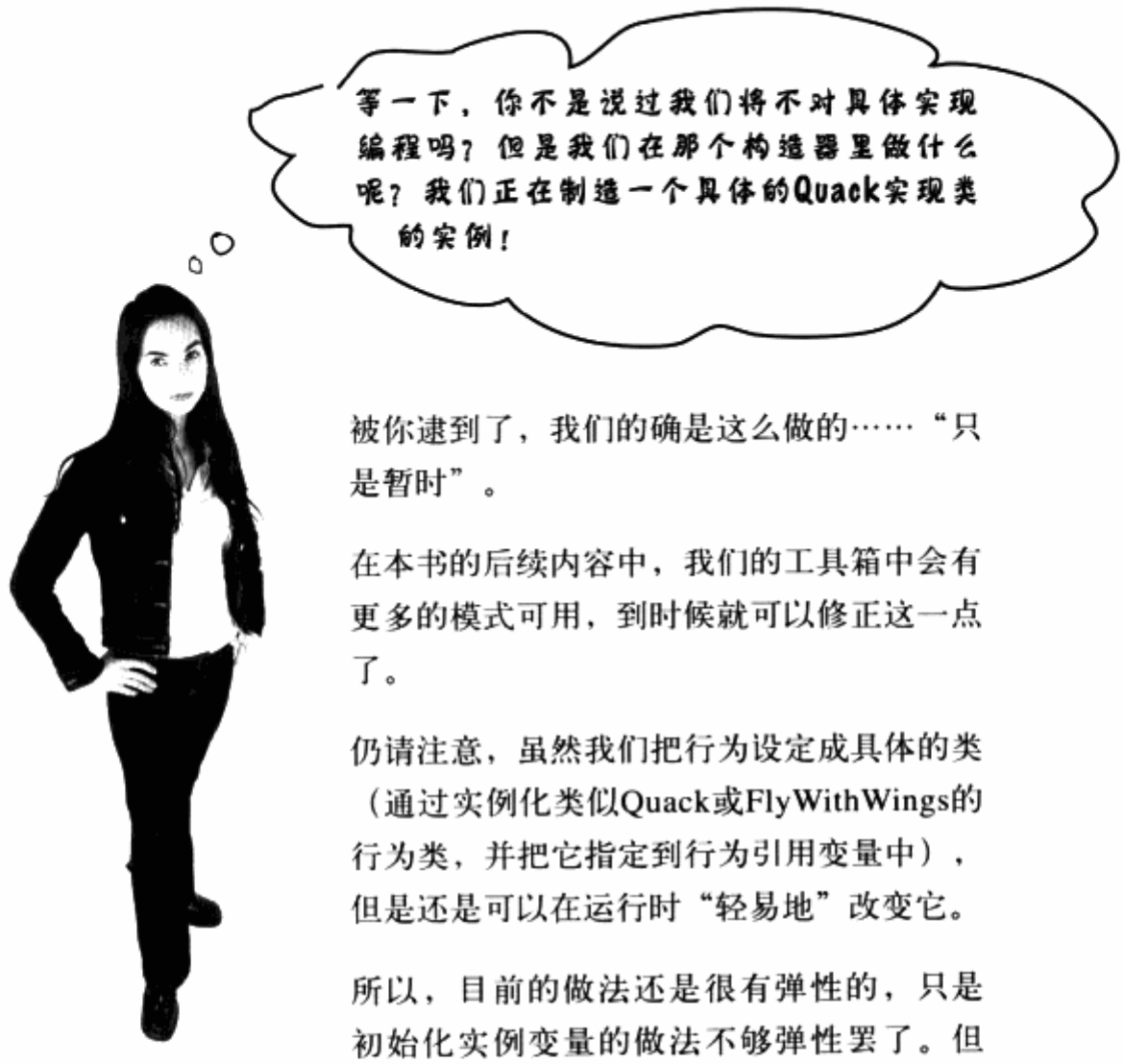
别忘了，因为MallardDuck继承Duck类，所以具有flyBehavior与quackBehavior实例变量。

使用FlyWithWings作为其FlyBehavior类型。

所以，绿头鸭会真的“呱呱叫”，而不是“吱吱叫”，或“叫不出声”。这是怎么办到的？当MallardDuck实例化时，它的构造器会把继承来的quackBehavior实例变量初始化成Quack类型的新实例（Quack是QuackBehavior的具体实现类）。

同样的处理方式也可以用在飞行行为上：MallardDuck的构造器将flyBehavior实例变量初始化成FlyWithWings类型的实例（FlyWithWings是FlyBehavior的具体实现类）。

本文档由Linux公社 www.linuxidc.com 收集整理



被你逮到了，我们的确是这么做的……“只是暂时”。

在本书的后续内容中，我们的工具箱中会有更多的模式可用，到时候就可以修正这一点了。

仍请注意，虽然我们把行为设定成具体的类（通过实例化类似Quack或FlyWithWings的行为类，并把它指定到行为引用变量中），但是还是可以在运行时“轻易地”改变它。

所以，目前的做法还是很有弹性的，只是初始化实例变量的做法不够弹性罢了。但是想一想，因为quackBehavior的实例变量是一个接口类型，我们能够在运行时，通过多态的魔力动态地给它地指定不同的QuickBehavior实现类。

花一点儿时间想一想，你如何实现一个其行为可以在运行时改变的鸭子。（几页以后，你就会看到做这件事的代码。）

本文档由Linux公社 www.linuxidc.com 收集整理

测试Duck的代码

❶ 输入并编译下面的Duck类（Duck.java）以及两页前的MallardDuck类（MallardDuck.java）。

```
public abstract class Duck {

   FlyBehavior flyBehavior;
   QuackBehavior quackBehavior;
   public Duck() {
   }

   public abstract void display();

   public void performFly() {
      flyBehavior.fly();
   }

   public void performQuack() {
      quackBehavior.quack();
   }

   public void swim() {
      System.out.println("All ducks float, even decoys!");
   }
}
```

为行为接口类型声明两个引用变量，所有鸭子子类（在同一个package中）都继承它们。

委托给行为类

❷ 输入并编译FlyBehavior接口（FlyBehavior.java）与两个行为实现类（FlyWithWings.java与FlyNoWay.java）。

```
public interface FlyBehavior {
   public void fly();
}
```

所有飞行行为类必须实现的接口。

```
public class FlyWithWings implements FlyBehavior {
   public void fly() {
       System.out.println("I'm flying!!");
   }
}
```

这是飞行行为的实现，给"会"飞的鸭子用……

```
public class FlyNoWay implements FlyBehavior {
   public void fly() {
       System.out.println("I can't fly");
   }
}
```

这是飞行行为的实现，给"不会"飞的鸭子用（包括橡皮鸭和诱饵鸭）。

本文档由Linux公社 www.linuxidc.com 收集整理

继续测试Duck的代码 ……

❸ 输入并编译QuackBehavior接口（QuackBehavior.java）及其三个实现类（Quack.java、MuteQuack.java、Squeak.java）。

```
public interface QuackBehavior {
   public void quack();
}
```

```
public class Quack implements QuackBehavior {
   public void quack() {
      System.out.println("Quack");
   }
}
```

```
public class MuteQuack implements QuackBehavior {
   public void quack() {
       System.out.println("<< Silence >>");
   }
}
```

```
public class Squeak implements QuackBehavior {
   public void quack() {
       System.out.println("Squeak");
   }
}
```

❹ 输入并编译测试类（MiniDuckSimulator.java）

```
public class MiniDuckSimulator {
   public static void main(String[] args) {
      Duck mallard = new MallardDuck();
      mallard.performQuack();
      mallard.performFly();
   }
}
```

这会调用MallardDuck继承来的performQuack()方法，进而委托给该对象的QuackBehavior对象处理（也就是说，调用继承来的quackBehavior引用对象的quack()）。

至于performFly()，也是一样的道理。

❺ 运行代码！

```
File Edit Window Help Yadayadayada
%java MiniDuckSimulator
Quack
I'm flying!!
```

本文档由Linux公社 www.linuxidc.com 收集整理

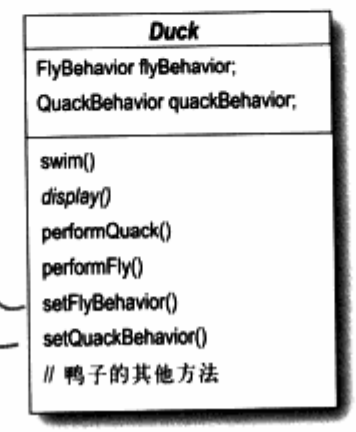
动态设定行为

在鸭子里建立了一堆动态的功能没有用到，就太可惜了！假设我们想在鸭子子类中通过“设定方法（setter method）”来设定鸭子的行为，而不是在鸭子的构造器内实例化。

1 在Duck类中，加入两个新方法：

```
public void setFlyBehavior(FlyBehavior fb) {
    flyBehavior = fb;
}

public void setQuackBehavior(QuackBehavior qb) {
    quackBehavior = qb;
}
```



从此以后，我们可以“随时”调用这两个方法改变鸭子的行为。

2 制造一个新的鸭子类型：模型鸭（ModelDuck.java）

```
public class ModelDuck extends Duck {
    public ModelDuck() {
        flyBehavior = new FlyNoWay();
        quackBehavior = new Quack();
    }

    public void display() {
        System.out.println("I'm a model duck");
    }
}
```

一开始，我们的模型鸭是不会飞的。

3 建立一个新的FlyBehavior 类型（FlyRocketPowered.java）

没关系，我们建立一个利用火箭动力的飞行行为。

```
public class FlyRocketPowered implements FlyBehavior {
    public void fly() {
        System.out.println("I'm flying with a rocket!");
    }
}
```

本文档由Linux公社 www.linuxidc.com 收集整理

❹ **改变测试类（MiniDuckSimulator.java），加上模型鸭，并使模型鸭具有火箭动力。**

```
public class MiniDuckSimulator {
   public static void main(String[] args) {
      Duck mallard = new MallardDuck();
      mallard.performQuack();
      mallard.performFly();

      Duck model = new ModelDuck();
      model.performFly();
      model.setFlyBehavior(new FlyRocketPowered());
      model.performFly();
   }
}
```

改变前

第一次调用performFly() 会被委托给flyBehavior对象（也就是FlyNoWay实例），该对象是在模型鸭构造器中设置的。

这会调用继承来的setter方法，把火箭动力飞行的行为设定到模型鸭中。哇！模型鸭突然具有了火箭动力飞行能力！

如果成功了，就意味着模型鸭可以动态地改变它的飞行行为。如果把行为的实现绑死在鸭子类中，可就无法做到这样了。

❺ 运行！

```
File Edit Window Help Yabadabadoo
%java MiniDuckSimulator
Quack
I'm flying!!
I can't fly
I'm flying with a rocket!
```

改变后

在运行时想改变鸭子的行为，只需调用鸭子的setter方法就可以。

本文档由Linux公社 www.linuxidc.com 收集整理

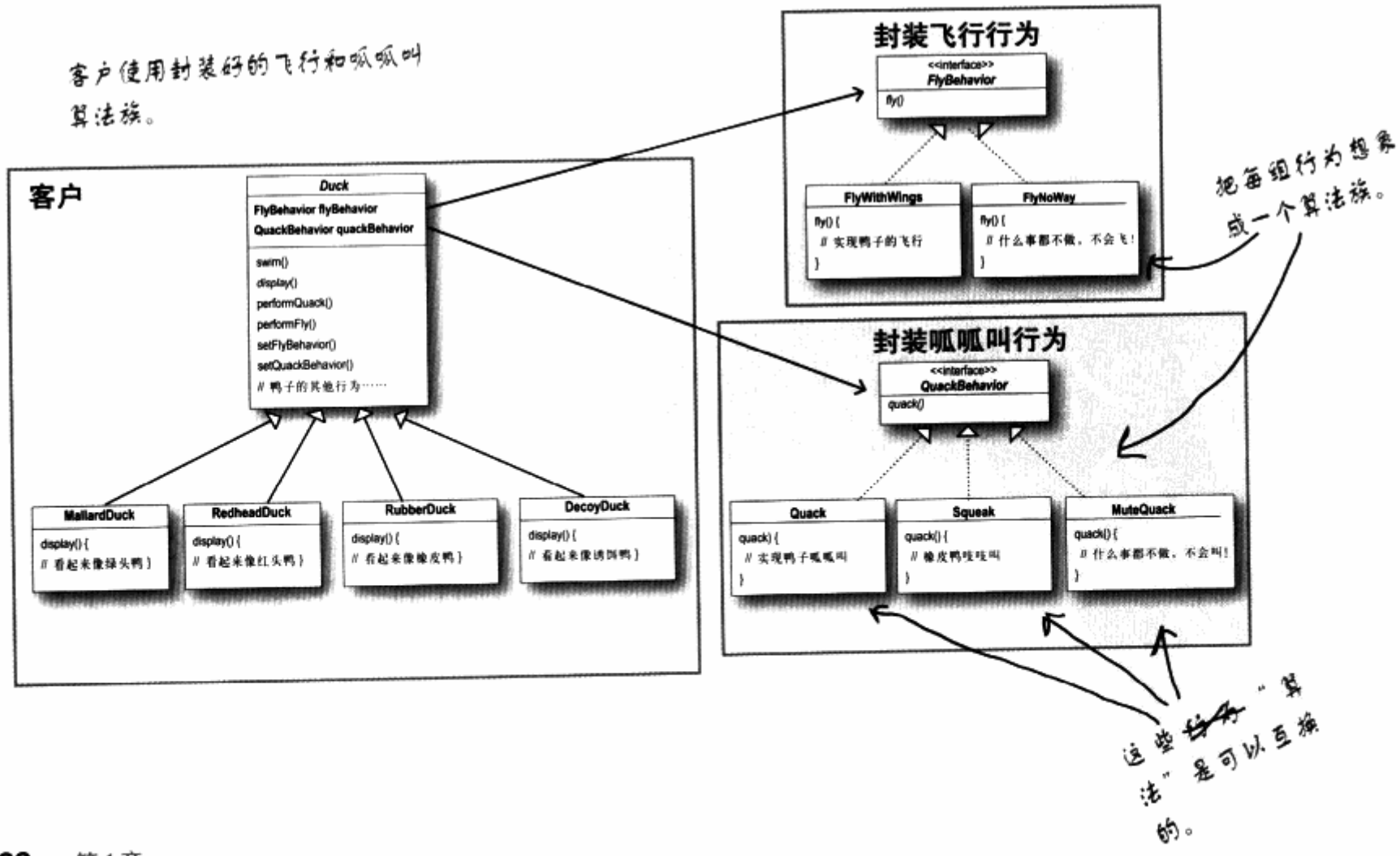
封装行为的大局观

好，我们已经深入研究了鸭子模拟器的设计，该是将头探出水面，呼吸空气的时候了。现在就来看看整体的格局。

下面是整个重新设计后的类结构，你所期望的一切都有：鸭子继承Duck，飞行行为实现FlyBehavior接口，呱呱叫行为实现QuackBehavior接口。

也请注意，我们描述事情的方式也稍有改变。不再把鸭子的行为说成是“一组行为”，我们开始把行为想成是“一族算法”。想想看，在SimUDuck的设计中，算法代表鸭子能做的事（不同的叫法和飞行法），这样的做法也能很容易地用于用一群类计算不同州的销售税金。

请特别注意类之间的“关系”。拿起笔，把下面图形中的每个箭头标上适当的关系，关系可以是IS-A（是一个）、HAS-A（有一个）或IMPLEMENTS（实现）。



本文档由Linux公社 www.linuxidc.com 收集整理

“有一个”可能比“是一个”更好。

“有一个”关系相当有趣：每一鸭子都有一个FlyBehavior和一个QuackBehavior，好将飞行和呱呱叫委托给它们代为处理。

当你将两个类结合起来使用，如同本例一般，这就是组合（composition）。这种做法和“继承”不同的地方在于，鸭子的行为不是继承来的，而是和适当的行为对象“组合”来的。

这是一个很重要的技巧。其实是使用了我们的第三个设计原则：

设计原则

多用组合，少用继承。

如你所见，使用组合建立系统具有很大的弹性，不仅可将算法族封装成类，更可以“在运行时动态地改变行为”，只要组合的行为对象符合正确的接口标准即可。

组合用在“许多”设计模式中，在本书中，你也会看到它的诸多优点和缺点。

鸭鸣器（duckcall）是一种装置，猎人用鸭鸣器模拟出鸭叫声，以引诱野鸭。你如何实现你自己的鸭鸣器，而不继承Duck类？

大师与门徒……

大师：蚱蜢，告诉我，在面向对象的道路上，你学到了什么？

门徒：大师，我学到了，面向对象之路承诺了“复用”。

大师：继续说……

门徒：大师，借由继承，好东西可以一再被利用，所以程序开发时间就会大幅减少，就好像在林中很快地砍竹子一样。

大师：蚱蜢呀！软件开发完成“前”以及完成“后”，何者需要花费更多时间呢？

门徒：答案是“后”，大师。我们总是需要花许多时间在系统的维护和变化上，比原先开发花的时间更多。

大师：蚱蜢，这就对啦！那么我们是不是应该致力于提高可维护性和可扩展性上的复用程度呀？

门徒：是的，大师，的确是如此。

大师：我觉得你还有很多东西要学，希望你再深入研究继承。你会发现，继承有它的问题，还有一些其他的方式可以达到复用。

本文档由Linux公社 www.linuxidc.com 收集整理

讲到设计模式……

恭喜你，学会第一个模式了！

你刚刚用了你的第一个设计模式：也就是策略模式（Strategy Pattern）。不要怀疑，你正是使用策略模式改写SimUDuck 程序的。多亏这个模式，现在系统不担心遇到任何改变，主管们可以勾画他们的赌城狂欢之旅了。

为了介绍这个模式，我们走了很长的一段路。下面是此模式的正式定义：

> **策略模式**定义了算法族，分别封装起来，让它们之间可以互相替换，此模式让算法的变化独立于使用算法的客户。

当你需要给朋友留下深刻的印象，或是想影响关键主管的决策时，请使用“这个”定义。

本文档由Linux公社 www.linuxidc.com 收集整理

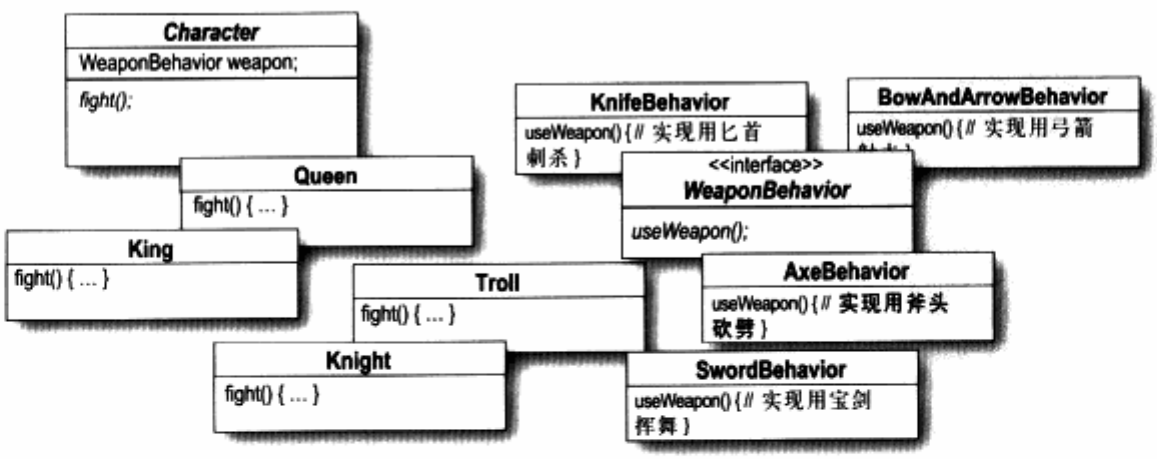
设计谜题

在下面，你将看到一堆杂乱的类与接口，这取自一个动作冒险游戏。你将看到代表游戏角色的类和角色可以使用的武器行为的类。每个角色一次只能使用一种武器，但是可以在游戏的过程中换武器。你的工作是要弄清楚这一切……

（答案在本章结尾处）

你的任务：

1. 安排类。
2. 找出一个抽象类、一个接口，以及八个类。
3. 在类之间画箭头。
   - a. 继承就画成这样（“extend”）。
   - b. 实现接口就画成这样（“implement”）。
   - c. “有一个”关系就画成这样。
4. 把setWeapon()方法放到正确的类中。




```
setWeapon(WeaponBehavior w) {
    this.weapon = w;
}
```

本文档由Linux公社 www.linuxidc.com 收集整理

在附近餐厅中无意间听到……

这两人点的餐有何不同？其实没有差异，都是一份单，只是Alice讲话的长度多了一倍，而且快餐店的厨师已经感到不耐烦了。

什么是Flo有的，而Alice没有？答案是，Flo和厨师之间有“共享的词汇”，Alice却不懂这些词汇。共享的词汇不仅方便顾客点餐，也让厨师不用记太多事，毕竟这些餐点模式都已经在他的脑海中了呀！

设计模式让你和其他开发人员之间有共享的词汇，一旦懂得这些词汇，和其他开发人员之间沟通就很容易，也会促使那些不懂的程序员想开始学习设计模式。设计模式也可以把你的思考架构的层次提高到模式层面，而不是仅停留在琐碎的对象上。

本文档由Linux公社 www.linuxidc.com 收集整理

在办公室隔间中无意间听到……

BRAIN POWER

除了面向对象设计和在餐厅点餐之外，你还能够想到有哪些例子需要共享词汇？（暗示：想一想汽车修理工、木工、大厨、航管）利用这些行话进行沟通的质量如何？

你能否想到OO设计的什么方面，能够和模式名称匹配的？“策略模式”这个名字是否传神？

本文档由Linux公社 www.linuxidc.com 收集整理

共享模式词汇的威力

你使用模式和他人沟通时，其实“不只是”和他人共享“行话”而已。

共享的模式词汇“威力强大”。当你使用模式名称和其他开发人员或者开发团队沟通时，你们之间交流的不只是模式名称，而是一整套模式背后所象征的质量、特性、约束。

“我们使用策略模式实现鸭子的各种行为。”这句话也就是告诉我们，鸭子的行为被封装进入一组类中，可以被轻易地扩充与改变。如果需要，甚至在运行时也可以改变行为。

模式能够让你用更少的词汇做更充分的沟通。当你用模式描述的时候，其他开发人员便很容易地知道你对设计的想法。

想想看，有多少次的设计会议中，你们一不小心就进入了琐碎的实现细节的讨论上。

将说话的方式保持在模式层次，可让你待在“设计圈子”久一点。使用模式谈论软件系统，可以让你保持在设计层次，不会被压低到对象与类这种琐碎的事情上面。

当你的团队开始利用模式分享设计想法与经验，你等于是建立了一个模式使用者的社区。

共享词汇可帮你的开发团队快速充电。对于设计模式有深入了解的团队，彼此之间对于设计的看法不容易产生误解。

考虑在你的组织内发起一个设计模式研讨会，说不定在学习的过程中，就开始得到回报了……

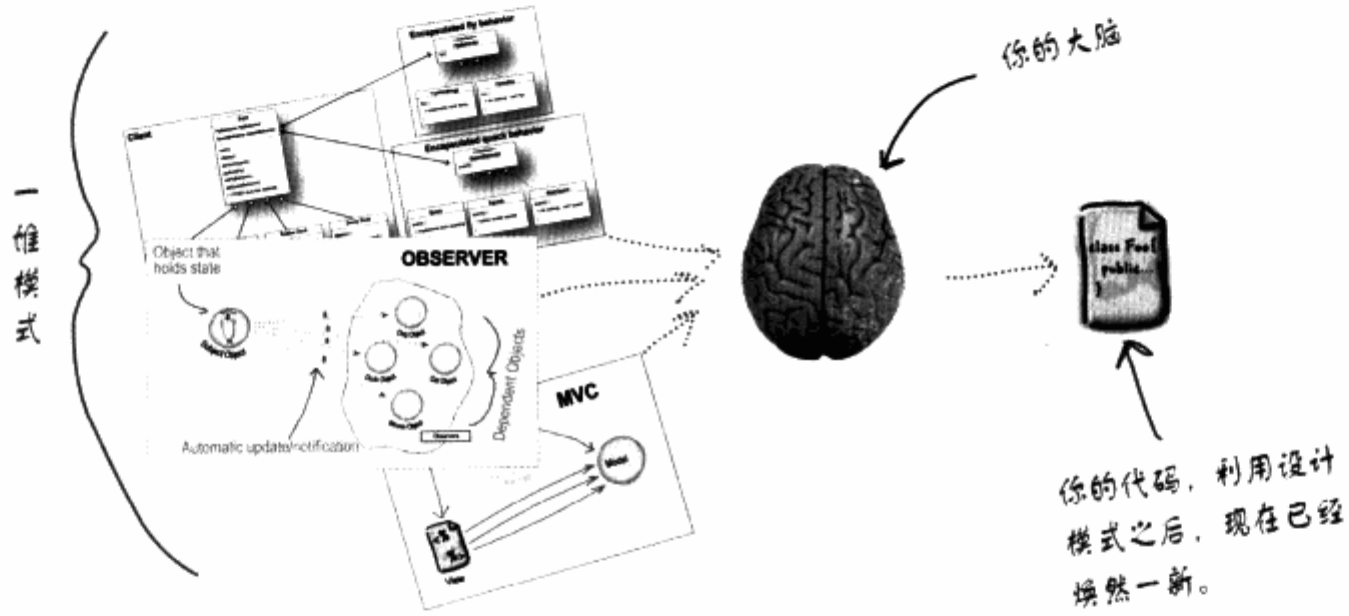
共享词汇能帮助初级开发人员迅速成长。初级开发人员向有经验的开发人员看齐。当高级开发人员使用设计模式，初级开发人员也会跟着学。把你的组织建立成一个模式使用者的社区。

本文档由Linux公社 www.linuxidc.com 收集整理

我如何使用设计模式？

我们全都使用别人设计好的库与框架。我们讨论库与框架、利用它们的API编译成我们的程序、享受运用别人的代码所带来的优点。看看Java API及它所带来的功能：网络、GUI、IO等。库与框架长久以来，一直扮演着软件开发过程的重要角色，我们从中挑选所要的组件，把它们放进合适的地方。但是……库与框架无法帮助我们将应用组织成容易了解、容易维护、具有弹性的架构，所以需要设计模式。

设计模式不会直接进入你的代码中，而是先进入你的“大脑”中。一旦你先在脑海中装入了许多关于模式的知识，就能够开始在新设计中采用它们，并当你的旧代码变得如同搅和成一团没有弹性的意大利面一样时，可用它们重做旧代码。



there are no Dumb Questions

问： 如果设计模式这么棒，为何没有人建立相关的库呢？那样我们就不必自己动手了。

答： 设计模式比库的等级更高。设计模式告诉我们如何组织类和对象以解决某种问题。而且采纳这些设计并使它们适合我们特定的应用，是我们责无旁贷的事。

问： 库和框架不也是设计模式吗？

答： 库和框架提供了我们某些特定的实现，让我们的代码可以轻易地引用，但是这并不算是设计模式。有些时候，库和框架本身会用到设计模式，这样很好，因为一旦你了解了设计模式，会更容易了解这些API是围绕着设计模式构造的。

问： 那么，没有所谓设计模式的库？

答： 没错，但是稍后你会看到设计模式类目。你可以在应用中利用这些设计模式。

本文档由Linux公社 www.linuxidc.com 收集整理

怀疑的开发人员　　友善的模式大师

开发人员：好吧！但是不都只是好的面向对象设计吗？我是说，我懂得运用封装、抽象、继承、多态，我真的还有必要考虑设计模式吗？运用OO，一切不是都很直接吗？这不正是我过去上了一堆OO课程的原因吗？我认为设计模式只对那些不懂好的OO设计的人有用。

大师：这是面向对象开发常有的谬误：以为知道OO基础概念，就能自动设计出弹性的、可复用的、可维护的系统。

开发人员：不是这样吗？

大师：不是！要构造有这些特征的OO系统，事实证明只有通过不断地艰苦实践，才能成功。

开发人员：我想我开始了解了，这些构造OO系统的隐含经验于是被收集整理出来……

大师：……是的，被整理成了一群“设计模式”。

开发人员：那么，如果知道了这些模式，我就可以减少许多体力劳动，直接采用可行的模式吗？

大师：对，在一定程度上可以这么说。不过要记住，设计是一门艺术，总是有许多可取舍的地方。但是如果你能采用这些经过深思熟虑，且经受过时间考验的设计模式，你就领先别人了。

本文档由Linux公社 www.linuxidc.com 收集整理

开发人员：如果我找不到模式，怎么办？

大师：有一些面向对象原则，适用于所有的模式。当你无法找到适当的模式解决问题时，采用这些原则可以帮助你。

开发人员：原则？你是说除了抽象、封装……之外，还有其他的？

大师：是的，建立可维护的OO系统，要诀就在于随时想到系统以后可能需要的变化以及应付变化的原则。

本文档由Linux公社 www.linuxidc.com 收集整理

设计工具箱内的工具

你几乎快要读完第1章了！你已经在你的设计工具箱内放进了几样工具，在我们进入第2章之前，先将这些工具一一列出。

OO基础

抽象

封装

多态

继承

我们假设你知道OO基础包括了多态的用法、继承就像按契约进行设计、封装是如何运作的。如果你觉得脑袋有一点生锈了，快快拿出你的《Head First Java》复习，然后再把这一章读一遍。

OO原则

封装变化

多用组合，少用继承

针对接口编程，不针对实现编程

我们会在后续的内容中更详细地看看这些原则，还会再多加一些原则到清单上。

阅读本书时，时时刻刻要思考着：模式如何仰赖基础与原则。

OO模式

策略模式——定义算法族，分别封装起来，让它们之间可以互相替换，此模式让算法的变化独立于使用算法的客户。

学了一个，还有更多！

要点

- 知道OO基础，并不足以让你设计出良好的OO系统。
- 良好的OO设计必须具备可复用、可扩充、可维护三个特性。
- 模式可以让我们建造出具有良好OO设计质量的系统。
- 模式被认为是历经验证的OO设计经验。
- 模式不是代码，而是针对设计问题的通用解决方案。你可把它们应用到特定的应用中。
- 模式不是被发明，而是被发现。
- 大多数的模式和原则，都着眼于软件变化的主题。
- 大多数的模式都允许系统局部改变独立于其他部分。
- 我们常把系统中会变化的部分抽出来封装。
- 模式让开发人员之间有共享的语言，能够最大化沟通的价值。

欢迎点击这里的链接进入精彩的Linux公社 网站

Linux公社（www.Linuxidc.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

Linux公社是专业的Linux系统门户网站，实时发布最新Linux资讯，包括Linux、Ubuntu、Fedora、RedHat、红旗Linux、Linux教程、Linux认证、SUSE Linux、Android、Oracle、Hadoop、CentOS、MySQL、Apache、Nginx、Tomcat、Python、Java、C语言、OpenStack、集群等技术。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

Linux公社 主站网址：www.linuxidc.com 旗下网站：www.linuxidc.net

包括：**Ubuntu 专题** **Fedora 专题** **Android 专题** **Oracle 专题** **Hadoop 专题** **RedHat 专题** **SUSE 专题** **红旗 Linux 专题** **CentOS 专题**

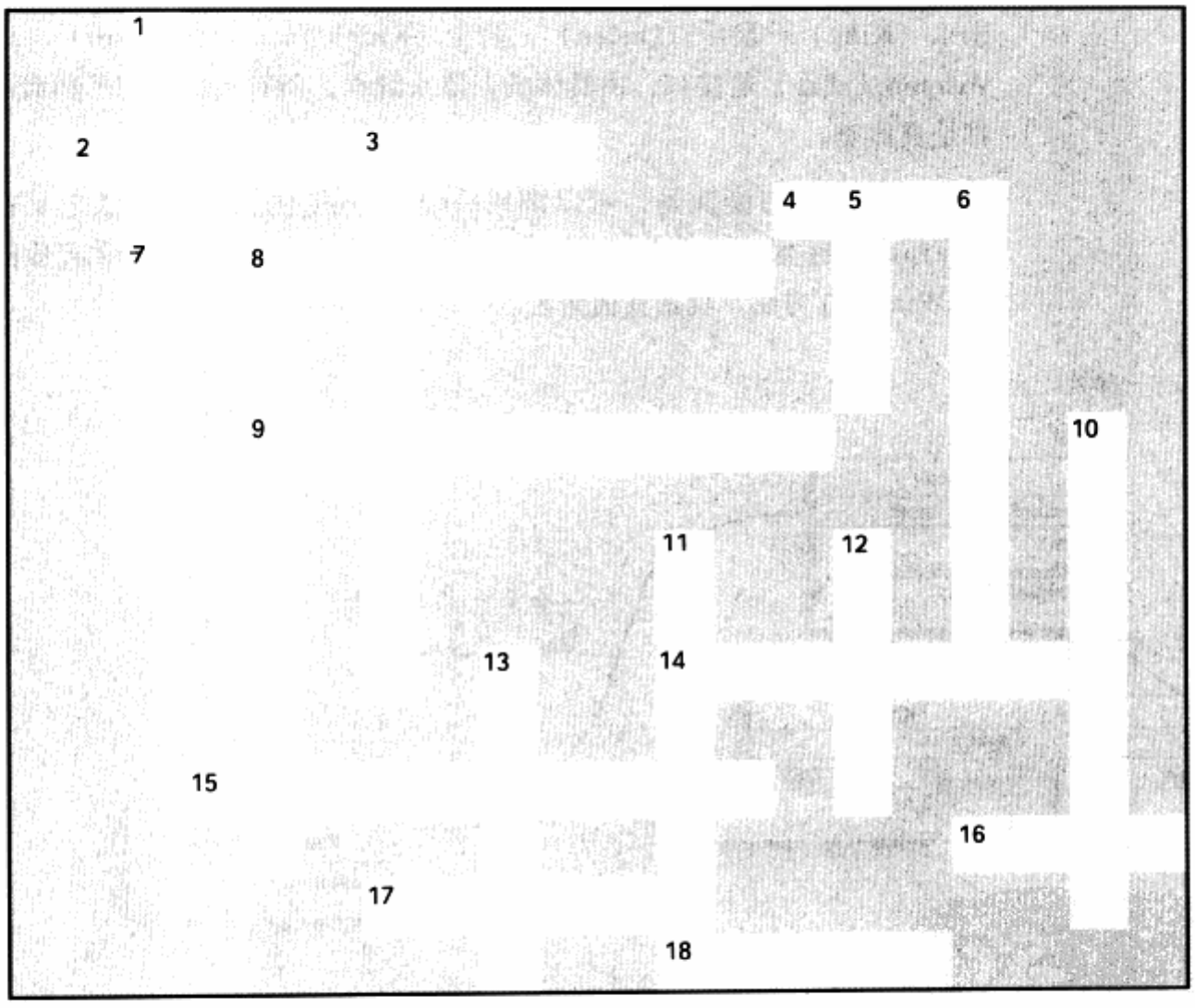
Linux公社 www.Linuxidc.com

Linux 公社微信公众号：linuxidc_com

本文档由Linux公社 www.linuxidc.com 收集整理

让标准填字游戏，动动你的右脑。

这是一个标准的纵横填字游戏，所有的词都来自本章。



横排提示：

2. Grilled cheese with bacon
4. Duck demo was located where
7. _____ what varies
9. Most patterns follow from OO _____
14. Pattern that fixed the simulator
15. Patterns give us a shared _____
16. Design patterns _____
17. Development constant
18. Patterns ____ in many applications

竖排提示：

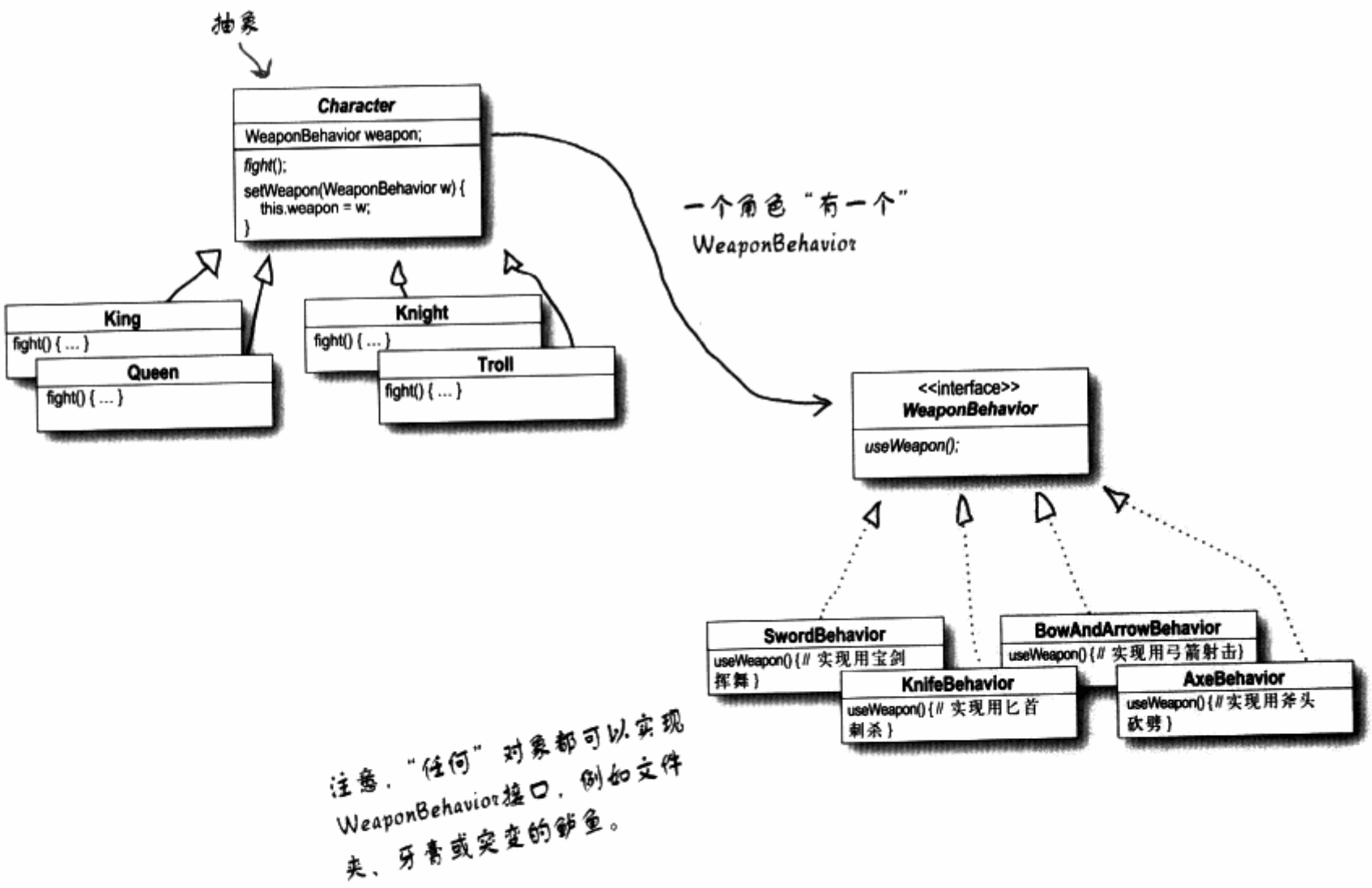
1. High level libraries
3. Learn from the other guy's _____
5. Java IO, Networking, Sound
6. Program to this, not an implementation
8. Favor over inheritance
10. Duck that can't quack
11. Rick was thrilled with this pattern
12. Patterns go into your _____
13. Rubberducks make a _____

本文档由Linux公社 www.linuxidc.com 收集整理

设计谜题解答

Character（角色）是抽象类，由具体的角色来继承。具体的角色包括：国王（King）、皇后（Queen）、骑士（Knight）、妖怪（Troll）。而Weapon（武器）是接口，由具体的武器来继承。所有实际的角色和武器都是具体类。

任何角色如果想换武器，可以调用setWeapon()方法，此方法定义在Character超类中。在打斗（flight）过程中，会调用到目前武器的useWeapon() 方法，攻击其他角色。



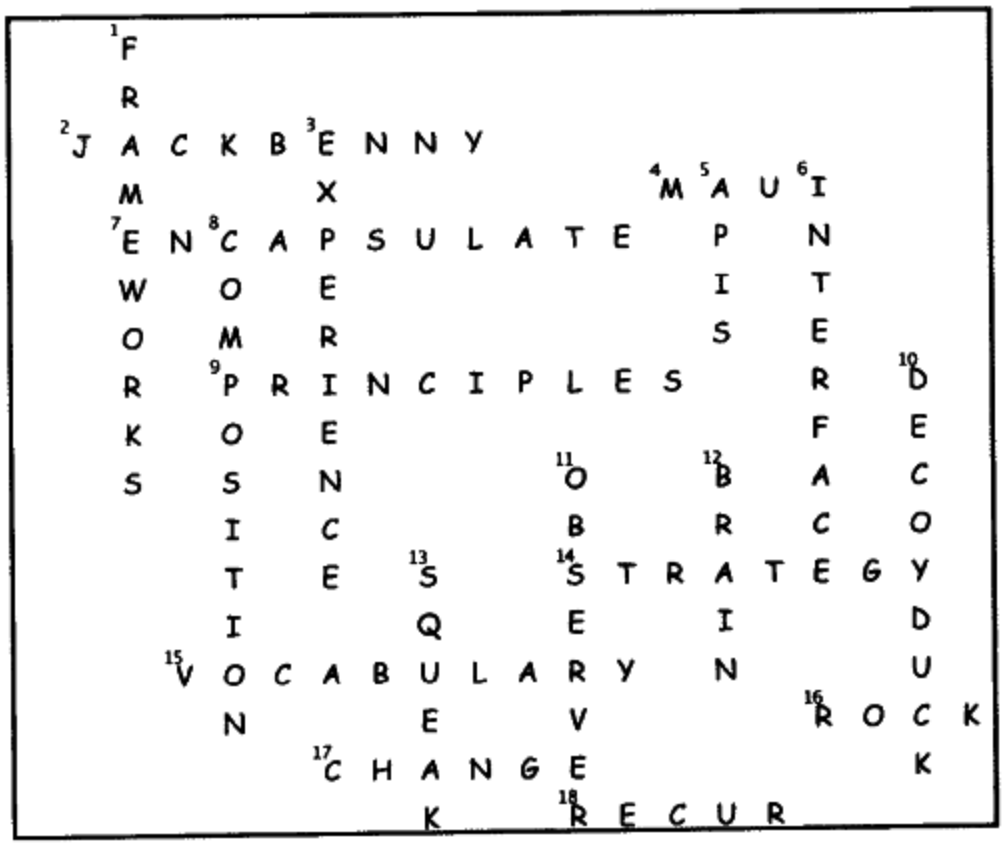
本文档由Linux公社 www.linuxidc.com 收集整理

答案

Sharpen your pencil

利用继承来提供Duck的行为，这会导致下列哪些缺点？（多选）

- ☑ A. 代码在多个子类中重复。
- ☑ B. 运行时的行为不容易改变。
- ☐ C. 我们不能让鸭子跳舞。
- ☑ D. 很难知道所有鸭子的全部行为。
- ☐ E. 鸭子不能同时又飞又叫。
- ☑ F. 改变会牵一发动全身，造成其他鸭子不想要的改变。



Sharpen your pencil 驱动改变的因素很多。找出你的软件中需要改变代码的地方，一一列出来。下面是我们的答案，你的答案可能和我们不一样。

我们的顾客或用户决定要别的做法，或者想要新功能。
我的公司决定采用别的数据库产品，又从另一家厂商买了数据，这造成数据格式不兼容。
唉！
唔！应对技术改变，我们必须更新代码，适用于新协议。
我们学到了足够的构建系统的知识，希望回去把事情做得更好。

本文档由Linux公社 www.linuxidc.com 收集整理

本文档由Linux公社 www.linuxidc.com 收集整理

2 观察者（Observer）模式

让你的对象知悉现况

喂，Jerry，我正在通知大家，模式小组会议改到周六晚上，这次要讨论的是观察者模式，这个模式最棒了！超级棒！你一定要来呀，Jerry。

有趣的事情发生时，可千万别错过了！ 有一个模式可以帮你的对象知悉现况，不会错过该对象感兴趣的事。对象甚至在运行时可决定是否要继续被通知。观察者模式是JDK中使用最多的模式之一，非常有用。我们也会一并介绍一对多关系，以及松耦合（对，没错，我们说耦合）。有了观察者，你将会消息灵通。

本文档由Linux公社 www.linuxidc.com 收集整理

恭喜你！

你的团队刚刚赢得一纸合约，负责建立Weather-O-Rama公司的下一代气象站——Internet气象观测站。

Weather-O-Rama气象站
100 Main Street
Tornado Alley, OK 45021

工作合约

恭喜贵公司获选为敝公司建立下一代Internet气象观测站！该气象站必须建立在我们专利申请中的WeatherData对象上，由WeatherData对象负责追踪目前的天气状况（温度、湿度、气压）。我们希望贵公司能建立一个应用，有三种布告板，分别显示目前的状况、气象统计及简单的预报。当WeatherObject对象获得最新的测量数据时，三种布告板必须实时更新。

而且，这是一个可以扩展的气象站，Weather-O-Rama气象站希望公布一组API，好让其他开发人员可以写出自己的气象布告板，并插入此应用中。我们希望贵公司能提供这样的API。

Weather-O-Rama气象站有很好的商业营运模式：一旦客户上钩，他们使用每个布告板都要付钱。最好的部分就是，为了感谢贵公司建立此系统，我们将以公司的认股权支付你。

我们期待看到你的设计和应用的alpha版本。

真挚的

Johnny Hurricane

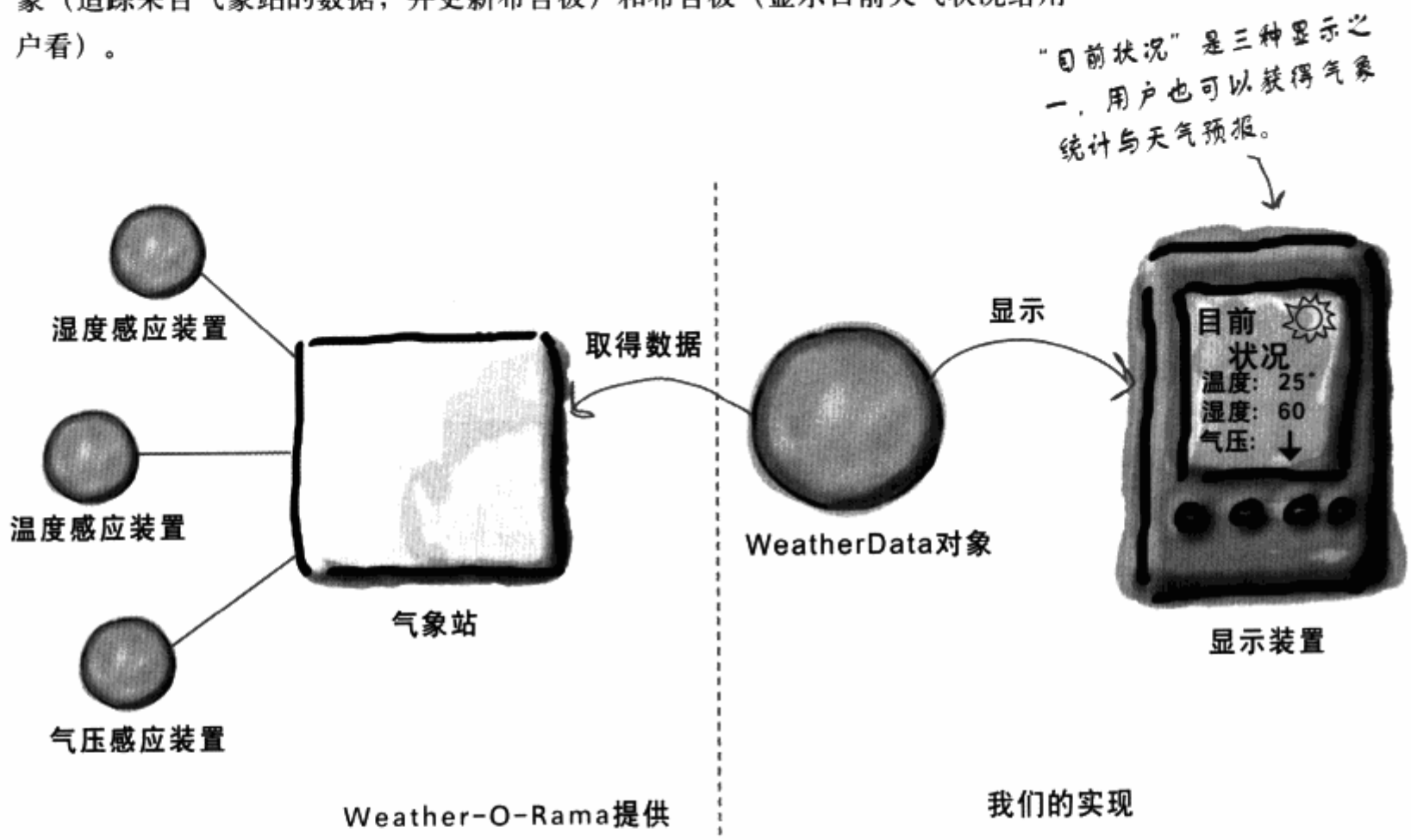
Johnny Hurricane——Weather-O-Rama气象站执行长

附注：我们正通宵整理WeatherData源文件给你们。

本文档由Linux公社 www.linuxidc.com 收集整理

气象监测应用的概况

此系统中的三个部分是气象站（获取实际气象数据的物理装置）、WeatherData对象（追踪来自气象站的数据，并更新布告板）和布告板（显示目前天气状况给用户看）。



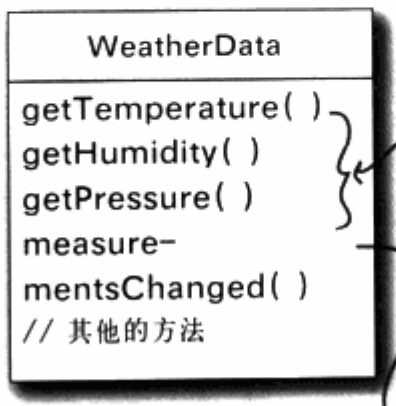
WeatherData对象知道如何跟物理气象站联系，以取得更新的数据。WeatherData对象会随即更新三个布告板的显示：目前状况（温度、湿度、气压）、气象统计和天气预报。

如果我们选择接受这个项目，我们的工作就是建立一个应用，利用WeatherData对象取得数据，并更新三个布告板：目前状况、气象统计和天气预报。

本文档由Linux公社 www.linuxidc.com 收集整理

瞧一瞧刚送到的WeatherData类

如同他们所承诺的，隔天早上收到了WeatherData源文件，看了一下代码，一切都很直接：



这三个方法各自返回最近的气象测量数据（分别为，温度、湿度、气压）。我们不在乎这些变量"如何"被设置，WeatherData对象自己知道如何从气象站获取更新信息。

```
/*
 * 一旦气象测量更新，此方法会被调用
 */
public void measurementsChanged() {
  // 你的代码加在这里
}
```

WeatherData.java

WeatherObject的开发人员留了一个线索，好让我们知道该加些什么……

再次提醒，这只是三个显示布告板中的一个。

显示装置

我们的工作是实现measurementsChanged()，好让它更新目前状况、气象统计、天气预报的显示布告板。

本文档由Linux公社 www.linuxidc.com 收集整理

我们目前知道些什么？

Weather-O-Rama气象站的要求说明并不是很清楚，我们必须搞懂该做些什么。那么，我们目前知道些什么呢？

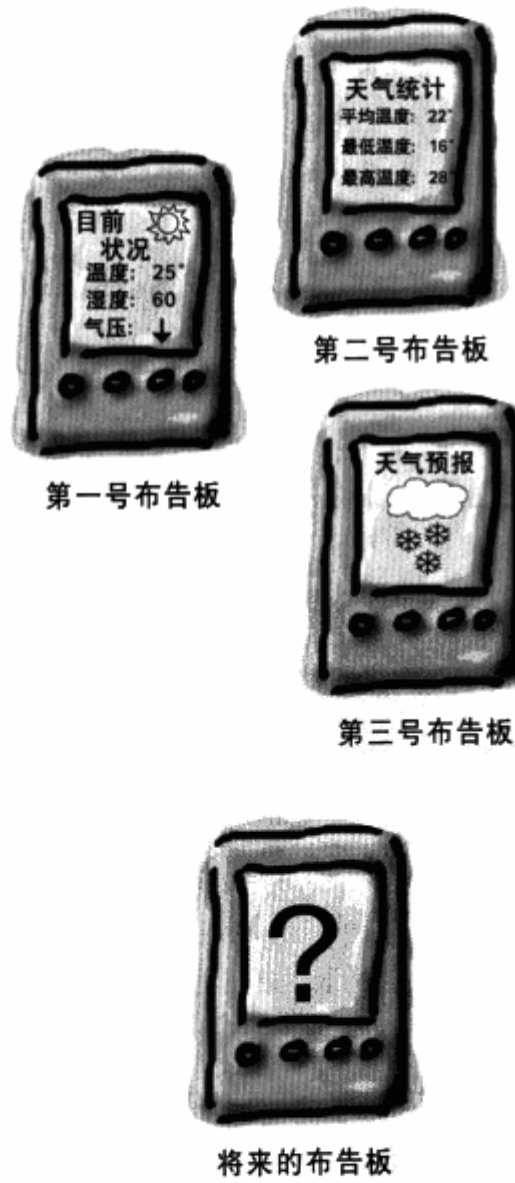
- WeatherData类具有getter方法，可以取得三个测量值：温度、湿度与气压。

```
getTemperature()
getHumidity()
getPressure()
```

- 当新的测量数据备妥时，measurementsChanged()方法就会被调用（我们不在乎此方法是如何被调用的，我们只在乎它被调用了）。

```
measurementsChanged()
```

- 我们需要实现三个使用天气数据的布告板：“目前状况”布告、“气象统计”布告、“天气预报”布告。一旦WeatherData有新的测量，这些布告必须马上更新。



第一号布告板

第二号布告板

第三号布告板

将来的布告板

- 此系统必须可扩展，让其他开发人员建立定制的布告板，用户可以随心所欲地添加或删除任何布告板。目前初始的布告板有三类：“目前状况”布告、“气象统计”布告、“天气预报”布告。

本文档由Linux公社 www.linuxidc.com 收集整理

先看一个错误示范

这是第一个可能的实现：我们依照Weather-O-Rama气象站开发人员的暗示，在measurementsChanged()方法中添加我们的代码：

```
public class WeatherData {

    // 实例变量声明

    public void measurementsChanged() {

        float temp = getTemperature();
        float humidity = getHumidity();
        float pressure = getPressure();

        currentConditionsDisplay.update(temp, humidity, pressure);
        statisticsDisplay.update(temp, humidity, pressure);
        forecastDisplay.update(temp, humidity, pressure);
    }

    // 这里是其他WeatherData方法
}
```

调用 WeatherData 的三个getXxx()方法，以取得最近的测量值。这些getXxx()方法已经实现好了。

现在，更新布告板……

调用每个布告板更新显示，传入最新的测量。

Sharpen your pencil

在我们的第一个实现中，下列哪种说法正确？（多选）

- ☐ A. 我们是针对具体实现编程，而非针对接口。
- ☐ B. 对于每个新的布告板，我们都得修改代码。
- ☐ C. 我们无法在运行时动态地增加（或删除）布告板。
- ☐ D. 布告板没有实现一个共同的接口。
- ☐ E. 我们尚未封装改变的部分。
- ☐ F. 我们侵犯了WeatherData类的封装。

SWAG的定义:Scientific Wild A** Guess

本文档由Linux公社 www.linuxidc.com 收集整理

我们的实现有什么不对？

回想第 1 章的概念和原则……

```
public void measurementsChanged( ) {

    float temp = getTemperature( );
    float humidity = getHumidity( );
    float pressure = getPressure( );

    currentConditionsDisplay.update(temp, humidity, pressure);
    statisticsDisplay.update(temp, humidity, pressure);
    forecastDisplay.update(temp, humidity, pressure);
}
```

改变的地方，需要封装起来。

至少，这里看起来像是一个统一的接口，布告板的方法名称都是update()，参数都是温度、湿度、气压。

针对具体实现编程，会导致我们以后在增加或删除布告板时必须修改程序。

我们现在就来看观察者模式，然后再回来看看如何将此模式应用到气象观测站。

本文档由Linux公社 www.linuxidc.com 收集整理

认识观察者模式

我们看看报纸和杂志的订阅是怎么回事：

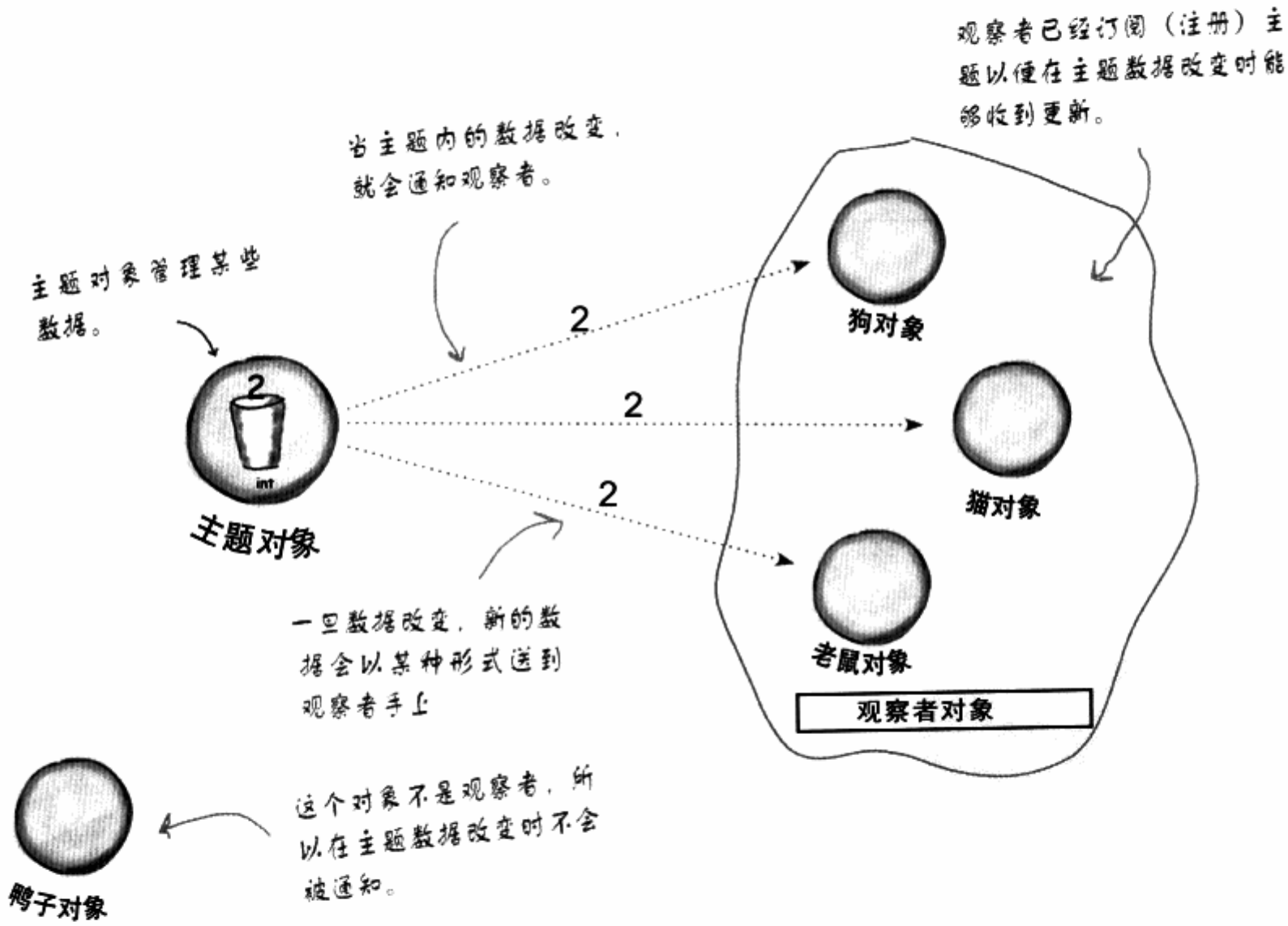
1. 报社的业务就是出版报纸。
2. 向某家报社订阅报纸，只要他们有新报纸出版，就会给你送来。只要你是他们的订户，你就会一直收到新报纸。
3. 当你不想再看报纸的时候，取消订阅，他们就不会再送新报纸来。
4. 只要报社还在运营，就会一直有人（或单位）向他们订阅报纸或取消订阅报纸。

本文档由Linux公社 www.linuxidc.com 收集整理

出版者+订阅者=观察者模式

如果你了解报纸的订阅是怎么回事，其实就知道观察者模式是怎么回事，只是名称不太一样：出版者改称为“主题”（Subject），订阅者改称为“观察者”（Observer）。

让我们来看得更仔细一点：

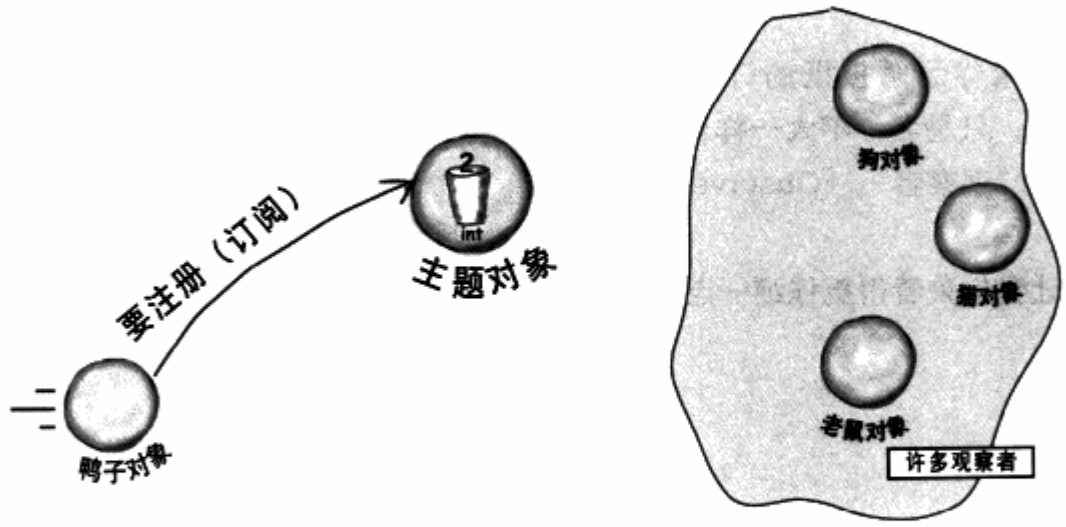


本文档由Linux公社 www.linuxidc.com 收集整理

观察者模式的一天

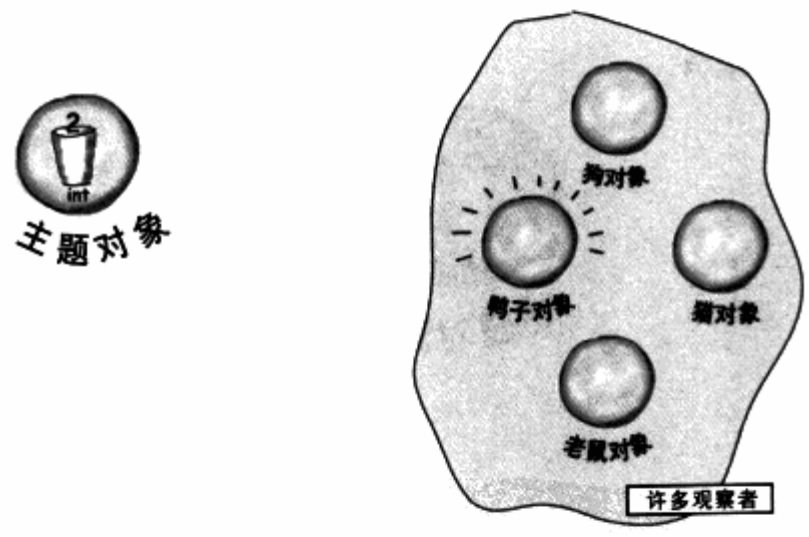
鸭子对象过来告诉主题，它想当一个观察者。

鸭子其实想说的是：我对你的数据改变感兴趣，一有变化请通知我。



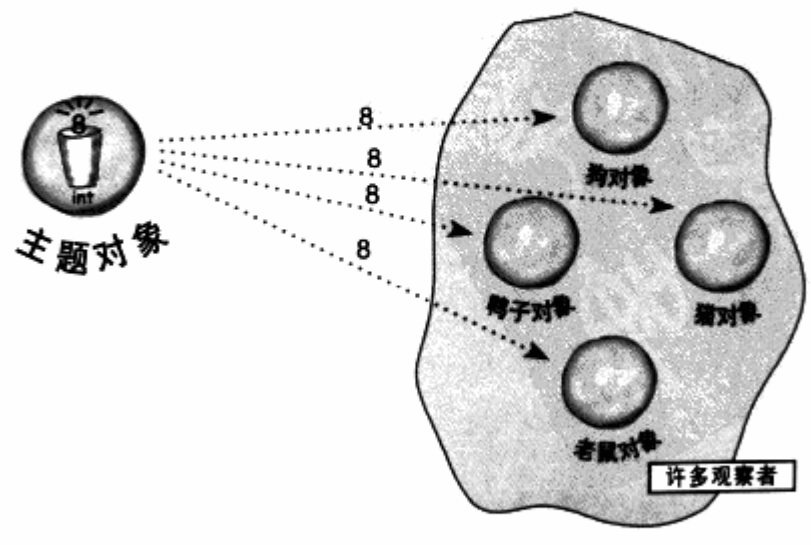
鸭子对象现在已经是正式的观察者了。

鸭子静候通知，等待参与这项伟大的事情。一旦接获通知，就会得到一个整数。



主题有了新的数据值！

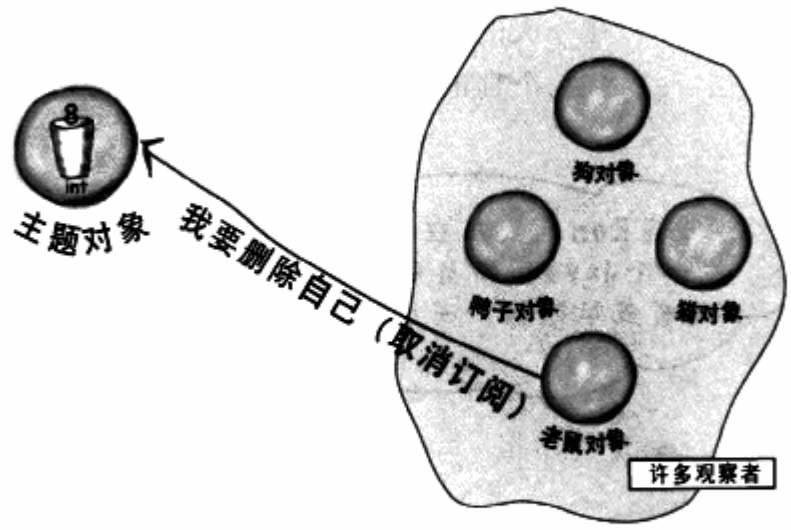
现在鸭子和其他所有观察者都会收到通知：主题已经改变了。



本文档由Linux公社 www.linuxidc.com 收集整理

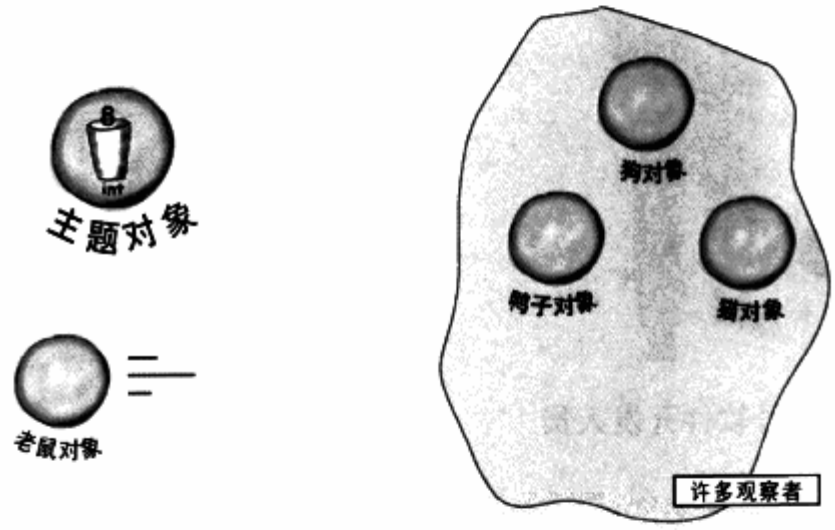
老鼠对象要求从观察者中把自己除名。

老鼠已经观察此主题太久，厌倦了，所以决定不再当个观察者。



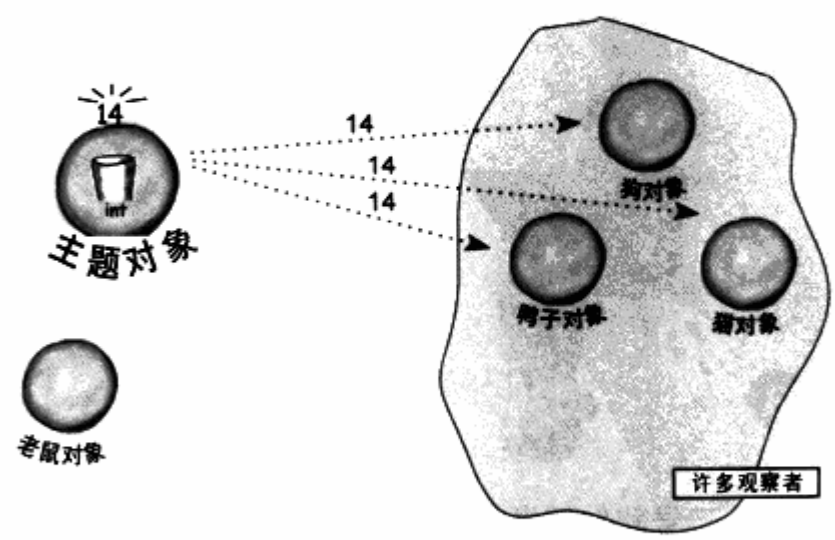
老鼠离开了！

主题知道老鼠的请求之后，把它从观察者中除名。



主题有一个新的整数。

除了老鼠之外,每个观察者都会收到通知，因为它已经被除名了。嘘！不要告诉别人，老鼠其实心中暗暗地怀念这些整数，或许哪天又会再次注册，回来继续当观察者呢！



本文档由Linux公社 www.linuxidc.com 收集整理

五分钟短剧：观察的主题

在今天的讽刺短剧中，有两个后泡沫时期的软件工程师，遇到一个真正的猎头……

一号软件开发人员

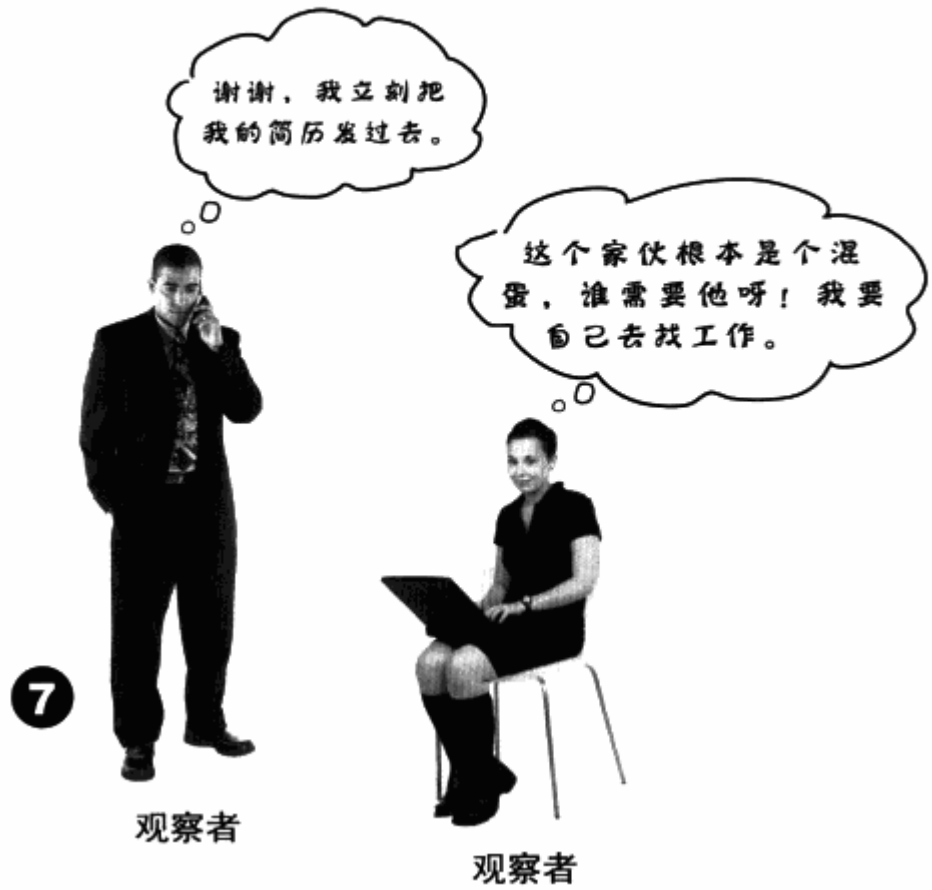
猎头/主题

二号软件开发人员

主题

本文档由Linux公社 www.linuxidc.com 收集整理

5 其间，Ron和Jill继续过自己的日子，如果Java工作来了，他们会接到通知，毕竟，他们是观察者嘛！



Jill自己找到工作了！

本文档由Linux公社 www.linuxidc.com 收集整理

两周后……

Jill热爱她的现状，她不再是观察者了。她还因为签约获得了一笔奖金，因为公司不用拨出一大笔钱给猎头。

但是，我们亲爱的Ron，又如何了？我们听说他设局把原来的猎头搞得毫无招架之力。他不只是一个观察者，也有了自己的求职者清单，只要付一笔钱给猎头，就可从其他求职者赚取更多钱。Ron既是一个主题，也是一个观察者，集两种角色于一身。

本文档由Linux公社 www.linuxidc.com 收集整理

定义观察者模式

当你试图勾勒观察者模式时，可以利用报纸订阅服务，以及出版者和订阅者比拟这一切。

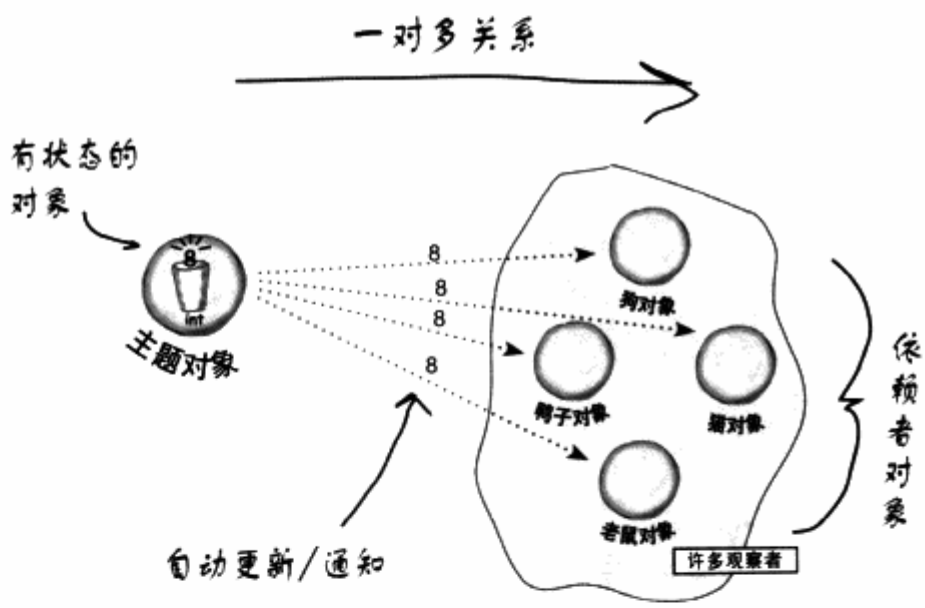
在真实的世界中，你通常会看到观察者模式被定义成：

> **观察者模式**定义了对象之间的一对多依赖，这样一来，当一个对象改变状态时，它的所有依赖者都会收到通知并自动更新。

观察者模式定义了一系列对象之间的一对多关系。

当一个对象改变状态，其他依赖者都会收到通知。

让我们看看这个定义，并和之前的例子做个对照：



主题和观察者定义了一对多的关系。观察者依赖于此主题，只要主题状态一有变化，观察者就会被通知。根据通知的风格，观察者可能因此新值而更新。

稍后你会看到，实现观察者模式的方法不只一种，但是以包含Subject与Observer接口的类设计的做法最常见。

让我们快来看看吧……

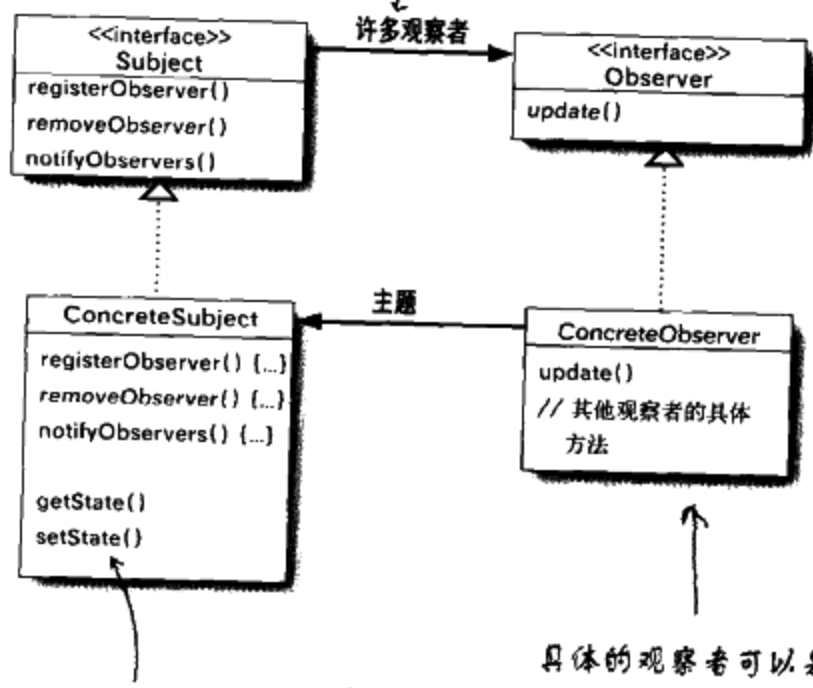
本文档由Linux公社 www.linuxidc.com 收集整理

定义观察者模式：类图

这是主题接口，对象使用此接口注册为观察者，或者把自己从观察者中删除。

每个主题可以有许多观察者

所有潜在的观察者必须实现观察者接口，这个接口只有update()一个方法，当主题状态改变时它被调用。



一个具体主题总是实现主题接口，除了注册和撤销方法之外，具体主题还实现了notifyObservers()方法，此方法用于在状态改变时更新所有当前观察者。

具体主题也可能有设置和获取状态的方法（稍后会进一步讨论）。

具体的观察者可以是实现此接口的任意类。观察者必须注册具体主题，以便接收更新。

there are no Dumb Questions

问： 这和一对多的关系有何关联？

答： 利用观察者模式，主题是具有状态的对象，并且可以控制这些状态。也就是说，有“一个”具有状态的主题。另一方面，观察者使用这些状态，虽然这些状态并不属于他们。有许多的观察者，依赖主题来告诉他们状态何时改变了。这就产生一个关系：“一个”主题对“多个”观察者的关系。

问： 其间的依赖是如何产生的？

答： 因为主题是真正拥有数据的人，观察者是主题的依赖者，在数据变化时更新，这样比起让许多对象控制同一份数据来，可以得到更干净的的OO设计。

本文档由Linux公社 www.linuxidc.com 收集整理

松耦合的威力

当两个对象之间松耦合，它们依然可以交互，但是不太清楚彼此的细节。

观察者模式提供了一种对象设计，让主题和观察者之间松耦合。

为什么呢？

关于观察者的一切，主题只知道观察者实现了某个接口（也就是Observer接口）。主题不需要知道观察者的具体类是谁、做了些什么或其他任何细节。

任何时候我们都可以增加新的观察者。因为主题唯一依赖的东西是一个实现Observer接口的对象列表，所以我们可以随时增加观察者。事实上，在运行时我们可以用新的观察者取代现有的观察者，主题不会受到任何影响。同样的，也可以在任何时候删除某些观察者。

你能够找到多少种不同的改变？

有新类型的观察者出现时，主题的代码不需要修改。假如我们有个新的具体类需要当观察者，我们不需要为了兼容新类型而修改主题的代码，所有要做的就是在新的类里实现此观察者接口，然后注册为观察者即可。主题不在乎别的，它只会发送通知给所有实现了观察者接口的对象。

我们可以独立地复用主题或观察者。如果我们在其他地方需要使用主题或观察者，可以轻易地复用，因为二者并非紧耦合。

改变主题或观察者其中一方，并不会影响另一方。因为两者是松耦合的，所以只要他们之间的接口仍被遵守，我们就可以自由地改变他们。

设计原则

为了交互对象之间的松耦合设计而努力。

松耦合的设计之所以能让我们建立有弹性的OO系统，能够应对变化，是因为对象之间的互相依赖降到了最低。

本文档由Linux公社 www.linuxidc.com 收集整理

Sharpen your pencil

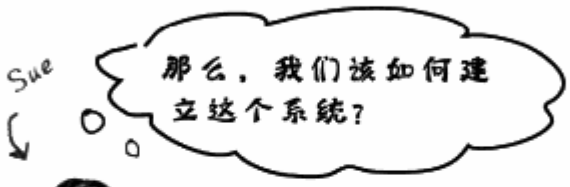
在继续后面的内容之前，请试着画出实现气象站所需要的类，其中包括WeatherData类及布告板组件。确定你的图能够显示出各个部分如何结合起来，以及别的开发人员如何能够实现他自己的布告板组件。

如果你需要一点小帮助，请阅读下一页，你的队友正在讨论如何设计气象站。

本文档由Linux公社 www.linuxidc.com 收集整理

办公室隔间对话

回到气象站项目，你的队友们已经开始全面思考这个问题了……



Mary：这个嘛！使用观察者模式啰！

Sue：是的……但是如何应用？

Mary：唔，我们再看一下定义好了：

观察者模式定义了对象之间的一对多依赖，这样一来，当一个对象改变状态时，它的所有依赖者都会收到通知并自动更新。

Mary：当你思考这个定义时，你会发现很有道理。我们的WeatherData类正是此处所说的“一”，而我们的“多”正是使用天气观测的各种布告板。

Sue：没错。WeatherData对象的确是有状态，包括了温度、湿度、气压，而这些值都会改变。

Mary：对呀！而且，当这些观测值改变时，必须通知所有的布告板，好让它们各自做出处理。

Sue：好棒！我现在知道如何将观察者模式应用在气象站问题上了。

Mary：还有一些问题有待理清，我现在还不太了解它们的解决方法。

Sue：什么问题？

Mary：其中一个问题是，我们如何将气象观测值放到布告板上。

Sue：回头去看看观察者模式的图，如果我们把WeatherData对象当作主题，把布告板当作观察者，布告板为了取得信息，就必须先向WeatherData对象注册。对不对？

Mary：是的……一旦WeatherData知道有某个布告板的存在，就会适时地调用布告板的某个方法来告诉布告板观测值是多少。

Sue：我们必须记得，每个布告板都有差异，这也就是为什么我们需要一个共同的接口的原因。尽管布告板的类都不一样，但是它们都应该实现相同的接口，好让WeatherData对象能够知道如何把观测值送给它们。

Mary：我懂你的意思。所以每个布告板都应该有一个大概名为update()的方法，以供WeatherData对象调用。

Sue：而这个update()方法应该在所有布告板都实现的共同接口里定义。

本文档由Linux公社 www.linuxidc.com 收集整理

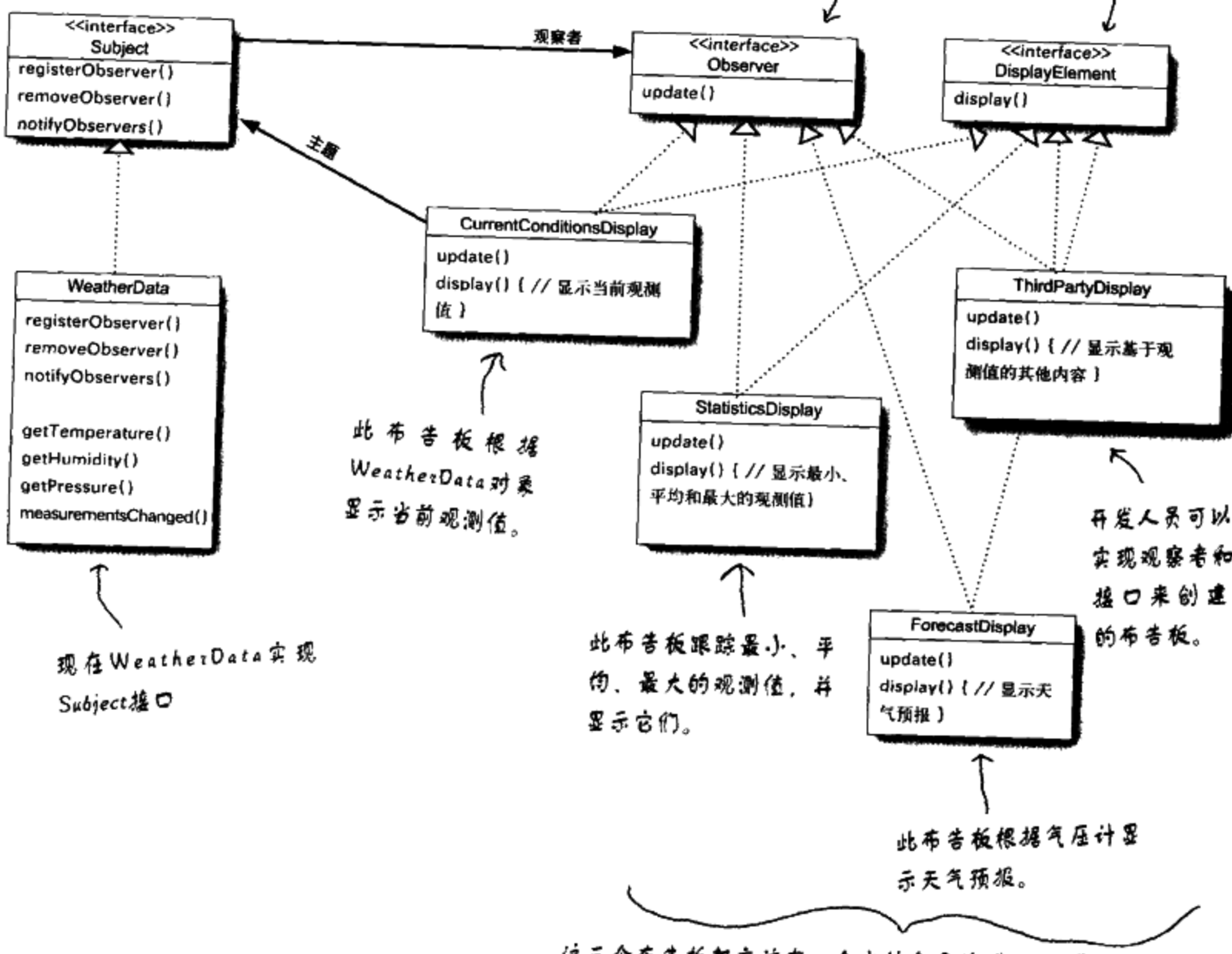
设计气象站

看看这个设计图，和你的设计图有何异同？

这是我们的主题接口，看起来应该不陌生。

所有的气象组件都实现此观察者接口。这样，主题在需要通知观察者时，有了一个共同的接口。

我们也为布告板建立一个共同的接口。布告板只需要实现display()方法。



此布告板根据WeatherData对象显示当前观测值。

开发人员可以通过实现观察者和显示接口来创建自己的布告板。

现在WeatherData实现Subject接口

此布告板跟踪最小、平均、最大的观测值，并显示它们。

此布告板根据气压计显示天气预报。

这三个布告板都应该有一个也被命名为"subject"的指针来指向WeatherData对象。但是，这张图没有画出这样的关系，以免太乱。

本文档由Linux公社 www.linuxidc.com 收集整理

实现气象站

依照两页前Mary和Sue的讨论，以及上一页的类图，我们要开始实现这个系统了。稍后，你将会在本章看到Java为观察者模式提供了内置的支持，但是，我们暂时不用它，而是先自己动手。虽然，某些时候可以利用Java内置的支持，但是有许多时候，自己建立这一切会更具弹性（况且建立这一切并不是很麻烦）。所以，让我们从建立接口开始吧：

```
public interface Subject {
    public void registerObserver(Observer o);
    public void removeObserver(Observer o);
    public void notifyObservers();
}
```

这两个方法都需要一个观察者作为变量，该观察者是用来注册或被删除的。

当主题状态改变时，这个方法会被调用，以通知所有的观察者。

```
public interface Observer {
    public void update(float temp, float humidity, float pressure);
}
```

当气象观测值改变时，主题会把这些状态值当作方法的参数，传送给观察者。

所有的观察者都必须实现update()方法，以实现观察者接口。在这里，我们按照Mary和Sue的想法把观测值传入观察者中。

```
public interface DisplayElement {
    public void display();
}
```

DisplayElement接口只包含了一个方法，也就是display()。当布告板需要显示时，调用此方法。

Mary和Sue认为：把观测值直接传入观察者中是更新状态的最直接的方法。你认为这样的做法明智吗？暗示：这些观测值的种类和个数在未来有可能改变吗？如果以后会改变，这些变化是否被很好地封装？或者是需要修改许多代码才能办到？

关于将更新的状态传送给观察者，你能否想到更好的方法解决此问题？

别担心，在我们完成第一次实现后，我们会再回来探讨这个设计决策。

本文档由Linux公社 www.linuxidc.com 收集整理

在WeatherData中实现主题接口

还记得我们在本章一开始的地方就试图实现WeatherData类吗？你可以去回顾一下。现在，我们要用观察者模式实现……

提醒你：为了节省篇幅，我们在代码中没有列出import和package语句。你可以到wickedlysmart网站找到完整的源代码，URL在本书的第xxxv页。

```
public class WeatherData implements Subject {
    private ArrayList observers;
    private float temperature;
    private float humidity;
    private float pressure;

    public WeatherData() {
        observers = new ArrayList();
    }

    public void registerObserver(Observer o) {
        observers.add(o);
    }

    public void removeObserver(Observer o) {
        int i = observers.indexOf(o);
        if (i >= 0) {
            observers.remove(i);
        }
    }

    public void notifyObservers() {
        for (int i = 0; i < observers.size(); i++) {
            Observer observer = (Observer)observers.get(i);
            observer.update(temperature, humidity, pressure);
        }
    }

    public void measurementsChanged() {
        notifyObservers();
    }

    public void setMeasurements(float temperature, float humidity, float pressure) {
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure = pressure;
        measurementsChanged();
    }

    // WeatherData的其他方法
}
```

WeatherData现在实现了Subject接口。

我们加上一个ArrayList来纪录观察者，此ArrayList是在构造器中建立的。

当注册观察者时，我们只要把它加到ArrayList的后面即可。

同样地，当观察者想取消注册，我们把它从ArrayList中删除即可。

这部分是Subject接口的实现。

有趣的地方来了！在这里，我们把状态告诉每一个观察者。因为观察者都实现了update()，所以我们知道如何通知它们。

当从气象站得到更新观测值时，我们通知观察者。

我们想要每本书随书赠送一个小型气象站，但是出版社不肯。所以，和从装置中读取实际的气象数据相比，我们宁愿利用这个方法来测试布告板。或者，为了好玩，你也可以写代码从网站上抓取观测值。

本文档由Linux公社 www.linuxidc.com 收集整理

现在，我们来建立布告板吧！

我们已经把WeatherData类写出来了，现在轮到布告板了。Weather-O-Rama气象站订购了三个布告板：目前状况布告板、统计布告板和预测布告板。我们先看看目前状况布告板。一旦你熟悉此布告板之后，可以在本书的代码目录中，找到另外两个布告板的源代码，你会觉得这些布告板都很类似。

此布告板实现了Observer接口，所以可以从WeatherData对象中获得改变。

它也实现了DisplayElement接口，因为我们的API规定所有的布告板都必须实现此接口。

```
public class CurrentConditionsDisplay implements Observer, DisplayElement {
    private float temperature;
    private float humidity;
    private Subject weatherData;

    public CurrentConditionsDisplay(Subject weatherData) {
        this.weatherData = weatherData;
        weatherData.registerObserver(this);
    }

    public void update(float temperature, float humidity, float pressure) {
        this.temperature = temperature;
        this.humidity = humidity;
        display();
    }

    public void display() {
        System.out.println("Current conditions: " + temperature
            + "F degrees and " + humidity + "% humidity");
    }
}
```

构造器需要weatherData对象（也就是主题）作为注册之用。

当update()被调用时，我们把温度和湿度保存起来，然后调用display()。

display()方法就只是把最近的温度和湿度显示出来。

there are no Dumb Questions

问： update()是最适合调用display()的地方吗？

答： 在这个简单的例子中，当值变化的时候调用display()，是很合理的。然而，你是对的，的确是有很多更好的方法来设计显示数据的方式。当我们谈到MVC（Model-View-Controller）模式时会再作说明。

问： 为什么要保存对Subject的引用呢？构造完后似乎用不着了呀？

答： 的确如此，但是以后我们可能想要取消注册，如果已经有了对Subject的引用会比较方便。

本文档由Linux公社 www.linuxidc.com 收集整理

启动气象站

❶ 先建立一个测试程序

气象站已经完成得差不多了，我们还需要一些代码将这一切连接起来。这是我们的第一次尝试，本书中稍后我们会再回来确定每个组件都能通过配置文件来达到容易“插拔”。现在开始测试吧：

```
public class WeatherStation {

    public static void main(String[] args) {
        WeatherData weatherData = new WeatherData();

        CurrentConditionsDisplay currentDisplay =
            new CurrentConditionsDisplay(weatherData);
        StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);
        ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);

        weatherData.setMeasurements(80, 65, 30.4f);
        weatherData.setMeasurements(82, 70, 29.2f);
        weatherData.setMeasurements(78, 90, 29.2f);
    }
}
```

首先，建立一个WeatherData对象。

如果你还不想下载完整的代码，可以将这两行注释掉，就能顺利执行了。

建立三个布告板，并把WeatherData对象传给它们。

模拟新的气象测量。

❷ 运行程序，让观察者模式表演魔术。

```
File Edit Window Help StormyWeather
%java WeatherStation
Current conditions: 80.0F degrees and 65.0% humidity
Avg/Max/Min temperature = 80.0/80.0/80.0
Forecast: Improving weather on the way!
Current conditions: 82.0F degrees and 70.0% humidity
Avg/Max/Min temperature = 81.0/82.0/80.0
Forecast: Watch out for cooler, rainy weather
Current conditions: 78.0F degrees and 90.0% humidity
Avg/Max/Min temperature = 80.0/82.0/78.0
Forecast: More of the same
%
```

本文档由Linux公社 www.linuxidc.com 收集整理

Sharpen your pencil

Johnny Hurricane（Weather-O-Rama气象站的CEO）刚刚来电告知，他们还需要酷热指数（HeatIndex）布告板，这是不可或缺的。细节如下：

酷热指数是一个结合温度和湿度的指数，用来显示人的温度感受。可以利用温度T和相对湿度RH套用下面的公式来计算酷热指数：

$$
\begin{aligned}
\text{heatindex} = {} & 16.923 + 1.85212 * 10^{-1} * T + 5.37941 * RH - 1.00254 * 10^{-1} * T * RH \\
& + 9.41695 * 10^{-3} * T^2 + 7.28898 * 10^{-3} * RH^2 + 3.45372 * 10^{-4} * T^2 * RH \\
& - 8.14971 * 10^{-4} * T * RH^2 + 1.02102 * 10^{-5} * T^2 * RH^2 - 3.8646 * 10^{-5} * T^3 \\
& + 2.91583 * 10^{-5} * RH^3 + 1.42721 * 10^{-6} * T^3 * RH + 1.97483 * 10^{-7} * T * RH^3 \\
& - 2.18429 * 10^{-8} * T^3 * RH^2 + 8.43296 * 10^{-10} * T^2 * RH^3 - 4.81975 * 10^{-11} * T^3 * RH^3
\end{aligned}
$$

开始练习打字吧！

开玩笑的啦！别担心，你不需要亲自输入此公式，只要建立你自己的HeatIndexDisplay.java文件并把公式从heatindex.txt文件中拷贝进来就可以了。

heatindex.txt文件可从wickedlysmart.com取得

这个公式是怎么回事？你可以参考《Head First气象学》，或者问问国家气象局的员工（或用Google搜索）。

当你完成后，输出结果应如下所示：

输出结果有改变的地方在这里。

File Edit Window Help OverdaRainbow

```
%java WeatherStation
Current conditions: 80.0F degrees and 65.0% humidity
Avg/Max/Min temperature = 80.0/80.0/80.0
Forecast: Improving weather on the way!
Heat index is 82.95535
Current conditions: 82.0F degrees and 70.0% humidity
Avg/Max/Min temperature = 81.0/82.0/80.0
Forecast: Watch out for cooler, rainy weather
Heat index is 86.90124
Current conditions: 78.0F degrees and 90.0% humidity
Avg/Max/Min temperature = 80.0/82.0/78.0
Forecast: More of the same
Heat index is 83.64967
%
```