普通高等教育
软件工程 “十三五”规划教材

13th Five-Year Plan Textbooks
of Software Engineering

软件设计模式

（慕课版）

朱洪军 ◎ 编著

非外借

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

普通高等教育
软件工程 “十三五”规划教材

13th Five-Year Plan Textbooks
of Software Engineering

软件设计模式

（慕课版）

朱洪军 ◎ 编著

人民邮电出版社
北京

图书在版编目（CIP）数据

软件设计模式 : 慕课版 / 朱洪军编著. -- 北京 : 人民邮电出版社, 2018.10
普通高等教育软件工程“十三五”规划教材
ISBN 978-7-115-48976-0

Ⅰ. ①软… Ⅱ. ①朱… Ⅲ. ①JAVA语言－软件设计－高等学校－教材 Ⅳ. ①TP312.8

中国版本图书馆CIP数据核字(2018)第169656号

内 容 提 要

本书全面介绍了软件设计模式的基本概念、应用方法和行业案例，内容有：学习基础—包括软件工程基础知识、UML、面向对象软件开发方法等；面向对象软件设计—主要是 SOLID 设计原则、GRASP 设计模式、简单工厂设计模式等；GoF 设计模式—包括 23 种 GoF 经典设计模式的意图、结构和使用案例等。书中涉及的开源工程有 Struts、Spring MVC、Hibernate、MyBatis、JSF、Eclipse、Gson、Event Bus、Apache Tika、Android 等。此外，本书还提供开放免费的配套 MOOC 和案例源码。

本书可作为高等院校计算机、软件工程等相关专业本科生或研究生的教材或参考书，也可供从事软件开发行业的相关人员参考。

◆ 编　　著　朱洪军
责任编辑　刘　博
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
固安县铭成印刷有限公司印刷

◆ 开本：787×1092　1/16
印张：16.75　　2018 年 10 月第 1 版
字数：435 千字　　2018 年 10 月河北第 1 次印刷

定价：49.80 元

读者服务热线：(010)81055256　印装质量热线：(010)81055316
反盗版热线：(010)81055315

献给：

我的家人

序

软件是现代各类信息系统的灵魂。当今软件系统越来越庞大，功能越来越丰富，设计越来越复杂，维护越来越困难。软件工程作为一门独立的学科已发展多年，软件生命周期模型、面向对象技术、软件体系结构和设计模式方法等概念逐步构成其核心理论和技术方法。

设计模式提供了一种抽象的软件实现方法论。其所包含的各种设计模式源自众多软件开发人员长期成功或失败经验的总结，是软件开发过程中所面临的一般性问题的解决方案。不同的设计模式展示了专业工作者思考和解决真实问题的过程，隐含于实践中的认识历程和行动逻辑更为重要。

对于高级软件开发人员而言，深入理解和熟练应用设计模式这一独特的软件设计方法非常关键。一方面，使用设计模式可以重用大量程序代码，使得复杂的代码更容易被他人理解，有效地提升软件系统的可靠性和可维护性。更重要的是，使用设计模式可以超越简单的代码复用而实现软件方案复用，以一种简便的方式复用成功的设计方案和体系结构，提升复杂系统的设计能力。另一方面，设计模式作为一种讨论软件设计的公共语言，使得熟练设计者的经验可以被初学者和其他设计者掌握。对于经验丰富的开发人员，学习设计模式有助于了解在软件开发过程中所面临问题的最佳解决方案；对于经验不足的开发人员，学习设计模式有助于通过一种简单快捷的方式理解软件设计。

据我们所知，中科大软件学院是国内最早开设“设计模式”软件工程硕士课程的。大约 2010 年，我们有幸邀请到美国德州阿灵顿大学软件工程教授龚振和老师为软件工程硕士研究生讲授高级软件工程相关课程。在与龚老师的交流中我们了解了设计模式的概念，认为它是构建软件工程硕士知识体系不可或缺的重要内容，便决定邀请龚老师开设“设计模式”课程。朱洪军老师从课程设置开始就作为龚老师的助教参与课程的教学辅导工作。经过深入学习和研究实践，朱老师自 2013 年起逐步独立地多次开设了该课程，取得了非常好的教学效果。历届选课学生的实习反馈也充分证明了这一课程的成功。

本书的形成源于朱老师多年的教学实践，强调使用模式解决代码设计问题的具体方法，而非抽象的软件工程理论，利于读者理解。书中以经典开源软件工程项目为案例，展示设计模式的应用场景，有利于学习者理论联系实际，极具特色。另外，作为 MOOC 式教学的积极实践者，朱老师还开发了与本书配套的线上课程，顺应了当前“互联网+”时代的线上线下相结合的学习模式，相信读者会有很好的收获。

中国科学技术大学软件学院

常务副院长　李曦

试读结束：需要全本请在线购买：　www.ertongbook.com

前　言

软件设计模式是从业人员应具备的最重要的知识或技能之一。从 GoF（Gang of Four，四人组）出版经典图书 *Design Patterns: Elements of Reusable Object-Oriented Software* 起，人们陆续发起了若干设计模式的讨论，有的侧重软件架构，有的强调程序的并发性能，还有的作为编程准则或规范等。当前，市场上对软件设计模式进行知识阐述与传播的图书已经足够多，它们从不同视角，以不同展示手法向读者传达了丰富的信息，但是仍然存在瓶颈，包括缺乏真实技术案例，过于偏重理论，抽象且入门门槛高，学习资源单调，等等。

针对以上问题，笔者做了以下努力。

1. 阅读大量开源技术的源码，抽取模式应用的真实案例

笔者使用约 1/3 的篇幅呈现了 20 多个开源技术源码中设计模式的应用案例，包括 Struts、Spring、Hibernate、MyBatis、Event Bus、Dom4j 等，详见人邮教育社区（http://www.ryjiaoyu.com）所提供的资源。

开源技术源码中的模式应用案例不仅提供了实例参考，也向学习者展示了资深（或权威）开发人员是如何使用设计模式解决实际业务问题的，这些对未入门（或刚入门）的读者来说是十分宝贵的学习指引。此外，依托经典案例，读者应侧重解读模式抽象理论与具体实践的不同，从而突破形式化的理论思维，达到灵活且恰当使用模式的目的。

例如，在书的第 4.3 节中讲到 GoF 定义了构造器模式的 Director 角色类，其职责为向客户端提供构造产品对象的接口；而在大多开源技术中，开发人员为了简化设计方案，并不单独定义 Director 角色类。还有很多其他模式案例亦是如此，读者可以留意相关内容。

2. 以解决具体设计问题为出发点，详细阐述模式的使用方法

针对每个具体模式的阐述，笔者均用 COS 系统（见附录）代码设计问题作为切入点，按分析问题→设计方案→实现代码的思路描述如何恰当地使用模式解决目标问题。特别地，在使用相关模式时，开发人员要注意它们的设计代价（即成本）。

相比于现有图书，笔者弱化了模式的抽象概念，并未过多纠缠模式的抽象理论。比如，GoF 原著作中会显式地分析不同模式的异同及相互关联，而本书并没有提及。因为，在笔者访谈的学习者中，**有很多人会十分在意模式的形式正确性，而忽略了模式作为问题解决方案的本质**，他们经常纠结于某个模式与其他模式的异同，却无法恰当且正确地使用它们。这显然是错误地定位了模式学习的着眼点。

模式是以设计问题为支撑的，如果没有设计问题，就不存在模式应用，学习者更不能脱离设计问题空谈模式。笔者尤其建议学习者不要拘泥于某个形式或理论，完全可以在实践中创建属于自己的模式。实际上，GoF 当时也是这样做的，所以才有了 GoF 模式。

3. 开发相配套的慕课（MOOC）视频和可运行源码等多维度学习资源

现有图书向学习者提供的参考资源相对单调，除了书籍自身呈现的内容，并无

太多辅助材料。为此，笔者特意在编写此书前，完成了软件设计模式 MOOC 视频的制作和 COS 案例源码的开发，并发布在公开平台上。读者可以访问人邮教育社区（http://www.ryjiaoyu.com）中提供的资源地址，按需获取开放的学习材料。

本书面向的读者为软件开发的从业人员，**适合编码人员、软件设计师、需求分析师、技术架构师、产品经理及项目经理等人群**。学习者在学习本书内容前，除应熟悉计算机及软件开发所需基础知识外，还需具备相关的领域知识，包括软件工程、Java 面向对象编程、UML、数据库系统应用等。本书采用 Java 编程语言展示示例代码，使用 UML 2.0 建模语言可视化设计模型。此外，读者应具备一定的项目（或代码）开发经验。

全书共 6 章内容，概括如下。

第 1 章为学习基础，主要内容包括软件生命周期、软件开发方法、面向对象的概念与特征、UML 使用等。学习者如已熟悉该内容，可略过。

第 2 章是面向对象设计原则，阐述了 SOLID 原则基本内涵、使用示例等。笔者建议读者着重关注 SOLID 原则的意图与使用注意事项。此外，常用的面向对象设计原则还有很多，限于篇幅，本书不能一一详述，还请学习者按需查阅相关资料以进一步了解。

第 3 章是设计模式入门，**介绍了模式定义、GRASP 模式和简单工厂模式**。模式存在于生活的各个方面，是人们已有经验的总结与抽象。更重要地，模式是针对具体业务问题的已被无数次验证过的可行的解决方案。因此，通过本章学习，读者能掌握恰当运用模式解决代码设计问题的方法，并能熟知模式给设计带来好处的同时也具有对应的代价（或成本）。

第 4 章是 GoF 创建型模式。**GoF 创建型模式为对象创建的设计问题提供了解决方案参考，包括单例、原型、构造器等 5 种模式**，每一种模式所适用的问题场景均不相同，也都有相应的设计代价。本章同时展示了 JDK、Android SDK、Struts 等开源技术或平台的模式案例，使读者感受真实且实用的知识实践体验。

第 5 章是 GoF 结构型模式，介绍了适配器、桥、组合等 7 种模式。**这些模式灵活地运用组合/聚合、继承、接口等类关系设计了针对具体代码问题（例如解耦合、保护目标代码等）的实践方案**。通过对 Spring MVC、MyBatis 等技术源码的剖析，本章从不同视角着力呈现了模式理论与具体实践之间的差异与联系。

第 6 章为 GoF 行为型模式，内容主要涉及责任链、命令、解释器等 11 种模式。业务行为是软件功能需求的实施，是受需求变化影响最大的软件元素，也是开发人员设计软件最重要的内容之一。**GoF 行为型模式展示了 11 种面向对象行为设计问题的通用实践方案**。OGNL、Event Bus、NetBeans 等案例也会让读者领会到资深开发人员是如何恰当且灵活地运用模式解决代码设计问题的。

本书编撰得到了如下项目资助："教育部 2017 第一批产学合作协同育人"项目（名称：软件设计模式 MOOC 教材建设，编号：201701064015），"安徽省教育厅 2017 年度高等学校省级质量工程"教学研究项目（名称：软件工程硕士课程混合式教学模式研究——以《软件设计模式》为例，编号：2017jyxm0030），"安徽省教育厅 2017 年度高等学校省级质量工程"大规模在线开放课程（MOOC）示范项目（名称：软件设计模式，编号：2017mooc383）。

最后，感谢出版社刘博编辑和其他工作人员给予的建议与帮助！感谢我的

学生在文字校验、图形绘制等方面给予的大力帮助，他们是胡杭、叶晓曦、陈灏、姚德义、龙海军、程起、王红钰、龚雨濛等！感谢欧开亮、韦宇轩、何林明、杜亚文等为 MOOC 视频制作提供的技术支持！感谢我的家人给予的关爱和鼓励！谢谢你们！

由于笔者自身的技术经验与知识视野有限，书中难免有疏漏之处，欢迎读者多指正！

朱洪军

2018 年 3 月于苏州独墅湖畔

目 录

试读结束：需要全本请在线购买：　www.ertongbook.com

第 1 章 学习基础

1.1 软件工程简介

从始至今的软件工程发展进程中，软件工程与计算机科学之间的领域边界一直模糊不清，甚至很多人认为软件工程并非独立领域，而是隶属于计算机科学的子领域。笔者认为，软件工程与计算机科学的关系，正如计算机科学与数学之间的关系。然而，现在绝大部分人不会将计算机科学作为数学的子领域进行定义或归属。

软件工程的定义在业界有不同的表述。如，IEEE（Institute of Electrical and Electronics Engineers，美国电气与电子工程协会）在“系统与软件工程”的词条中定义软件工程为“系统地运用科学技术的知识、方法和经验，设计、实现、测试和文档化软件产品”；而在 IEEE 的软件工程术语词条中，则将软件工程定义为“运用系统的、规范的、定量的方法，开发、操作和维护软件产品”。

在国内，推荐性国家标准 2006（GB/T 2006）在“信息技术与软件工程”术语词条中，将软件工程定义为“应用计算机科学理论和技术以及工程管理原则和方法，按预算和进度，实现满足用户要求的软件产品定义、开发、发布和维护的工程或进行研究的学科”。

软件工程作为正式术语使用，最早可追溯至 NATO（North Atlantic Treaty Organization，北大西洋公约组织）于 1968 年（部分文献标注为 1969 年）在德国举办的软件工程会议，此次会议邀请了来自企业届、学术届等领域的计算机科学家、编程人员共同讨论软件危机的应对策略。之后，随着操作系统、个人计算机以及面向对象语言的出现，软件开发过程模型在 20 世纪 80 年代被人们提出；到了 20 世纪 90 年代早期，开源软件逐步发展，给软件工程带来了巨大影响。Java 编程语言的出现使得软件开发更加关注软件架构，而 UML（Unified Modeling Language，统一建模语言）则统一了面向对象软件模型表达方式。同时，随着 Internet、分布式计算、移动互联网等技术的发展，软件工程则演变成更加关注软件产品的成本、友好度、程序效率等问题的学科。

总体上，**软件工程关注或解决的软件产品开发问题包括生产率（Productivity）、质量（Quality）、成本（Costs）、时间（Time）**。国际标准 ISO/IEC TR 19759 归纳的软件工程知识体系（Software Engineering Body of Knowledge，SWEBOK）涉及软件需求、软件设计、软件构建、软件测试、软件维护、软件配置管理、软件工程管理、软件工程过程、软件工程模型与方法、软件质量、软件工程经济学、计算机基础、数学基础及工程学基础等。

1.1.1 软件生命周期

软件生命周期（Software Lifecycle）指从软件计划开始直至软件销毁的整个周期，一般用软件开发周期（Software Development Life Cycle，SDLC）进行表达。软件开发周期一般包括 6 个阶段，分别为计划（Planning）、分析（Analysis）、设计（Design）、实现（Implementation）、测试与集成（Testing and Integration）、维护（Maintenance）。

最早被提出的软件开发周期模型为瀑布模型（Waterfall Model）。很多文献资料中一般会将瀑布模型的提出者标注为美国计算机科学家 Winston W. Royce，因为，其在 1970 年发表的 IEEE 论文“Managing the Development of Large Software Systems”中，首次描述了瀑布模型。但是，瀑布模型作为软件工程术语正式使用要推迟至 1976 年。

瀑布模型的得名因其将软件开发过程从上至下分成 6 个阶段，每一阶段都衔接在上一阶段之后，如图 1.1 所示。

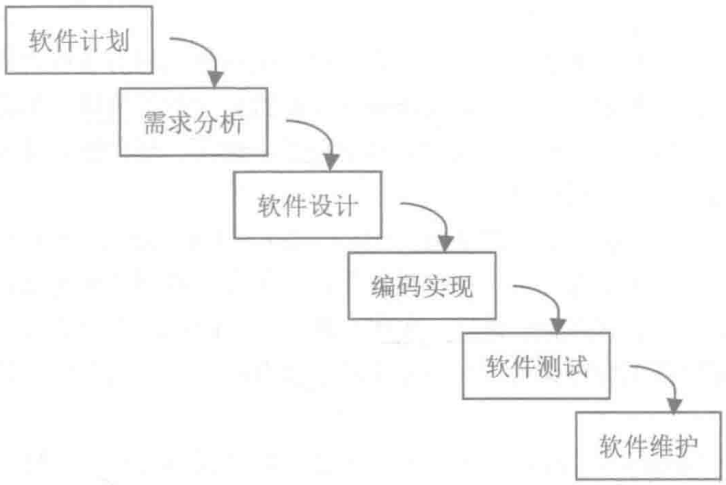

图 1.1 瀑布模型

图 1.1 中，从软件计划开始至软件维护阶段，自上而下形成了一个完整的软件开发周期。瀑布模型是一种理想软件开发周期模型。它假定软件需求是稳定不变的，要求下一阶段的软件开发活动必须在上一阶段活动完成后开始。然而，在实际的软件开发业务中，软件需求的易变特性往往会导致其他阶段活动在开展时反复实施，从而增加软件开发风险。

为了解决软件需求不明确或不稳定之类的问题，后来人提出了原型模型（Prototyping Model），如图 1.2 所示。

图 1.2 中，原型模型的软件开发阶段包括初步需求、原型设计、编码实现、原型测试、原型评估和软件交付。值得注意的是，由于软件需求不稳定，原型评估阶段需要决定是进行下一周期的原型优化，还是将软件交付给客户。如果需要进行下一周期的原型优化，软件开发人员会在原型评估的基础上继续进行原型设计、编码实现等，直到最终形成可交付的软件产品。

原型模型能够很好地解决软件需求不稳定或不明确可能造成的开发问题，但由于原型开发的成本不易控制，也会产生开发成本超支、需求分析不充分等问题。此外，在实践中，人们将原型模型分成快速原型（Rapid Prototyping）、增量原型（Incremental Prototyping）、迭代原型（Evolutionary Prototyping）和螺旋模型（Spiral Model）等。快速原型模型强调原型开发的时间短，主要是为了通过原型获取更精确的软件（用户）需求。根据快速原型模型所开发的软件原型一般都存在明显

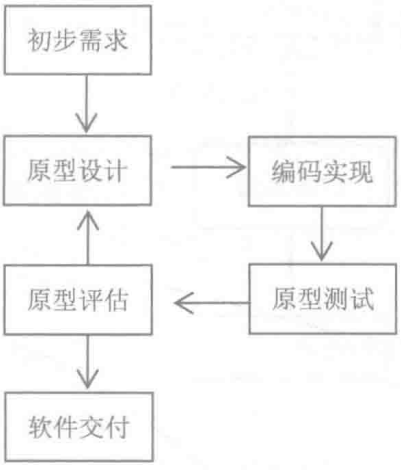

图 1.2　原型模型

的软件缺陷，最终一般都会被丢弃。增量原型则是在每个原型周期开发出最终软件产品的一部分功能子集，最终形成完整的软件产品。迭代原型与增量原型类似，只是更加注重在原型周期中开发出较为完整的软件，在不同的原型周期开发中优化和迭代上一周期的产品。螺旋模型是在原型模型的基础上加入了风险分析与控制的活动，使得原型产品开发的风险更加可控。

此外，还有一些软件开发周期模型，如 Rational 统一过程（Rational Unified Process，RUP）模型、敏捷过程（Agile Process）模型等，读者可以做进一步针对性了解和熟悉。

软件开发周期模型指出了每个开发阶段的活动或任务，但没有明确地指出如何具体实施软件开发计划，包括实施步骤、成果规范、工具或环境、实现技术等。因此，软件技术人员或学习者通过软件开发周期模型，仅仅形成了软件开发周期的概念模型，仍然需要进一步学习更多软件开发的实践方法或技术。

1.1.2　软件开发方法

软件开发方法定义了如何实施软件开发周期模型的每个阶段任务，包括计划、构建和控制这些任务时所使用到的方法、工具及技术等。常用的软件开发方法有结构化方法、面向对象方法、敏捷方法、可视化方法等。

结构化方法于 20 世纪 70 年代被提出，分为结构化分析（Structured Analysis，SA）方法和结构化设计（Structured Design，SD）方法。结构化分析采用自顶向下（Top Down）的方法，以数据流（Data Flows）的方式构建软件逻辑视图，将软件功能定义为数据流中的处理过程。结构化设计依据低耦合（Low Coupling）、高内聚（Hign Cohesion）的原则，使用结构图（Structure Chart，SC）、数据字典（Data Dictionary）等对软件模块结构及模块接口进行设计。

图 1.3 所示的是 COS 系统部分数据流图（Data Flow Diagram，DFD）示例。数据流图一般包括系统功能（加工/处理，用圆角矩形符号标识）、外部实体（用直角矩形标识）、数据存储（用开口矩形或平行线标识）和数据流向（用带箭头直线标识）。图 1.3 所示的数据流图中，客户向 COS（Cafeteria Ordering System，订餐系统）系统中输入订单信息，COS 系统生成订单并存储，COS 系统向配餐员发送配送指令。

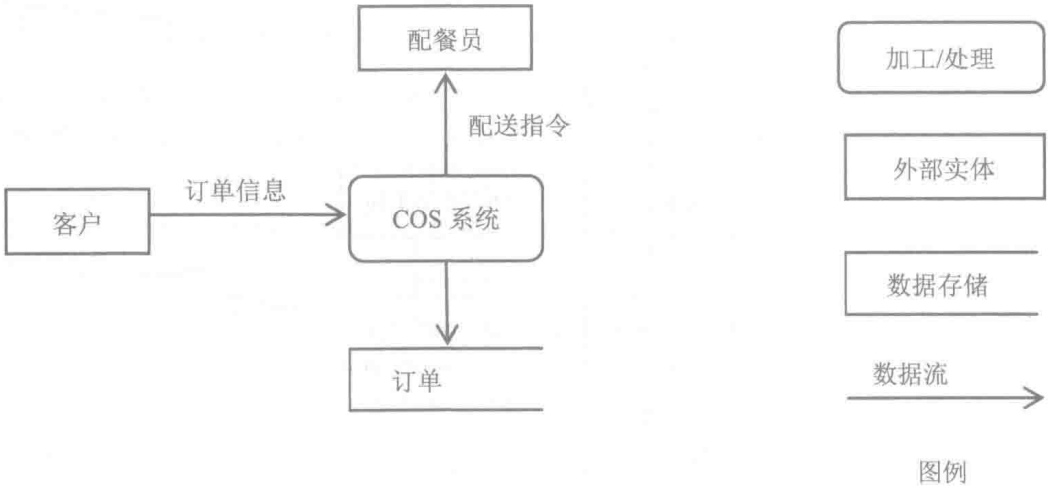

图 1.3　COS 系统部分数据流图示例

（注：COS 系统需求见附录 A）

结构图以“自顶向下”的视角对系统进行可视化建模。图 1.4 表达了 COS 系统中“下订单”“生成订单”“确认订单”“支付订单”等模块之间的逻辑关系结构图。图 1.4 中，“下订单”模块调用“生成订单模块”，并将“订单”数据发送至“确认订单”模块，最后调用“支付订单”模块获取支付结果。

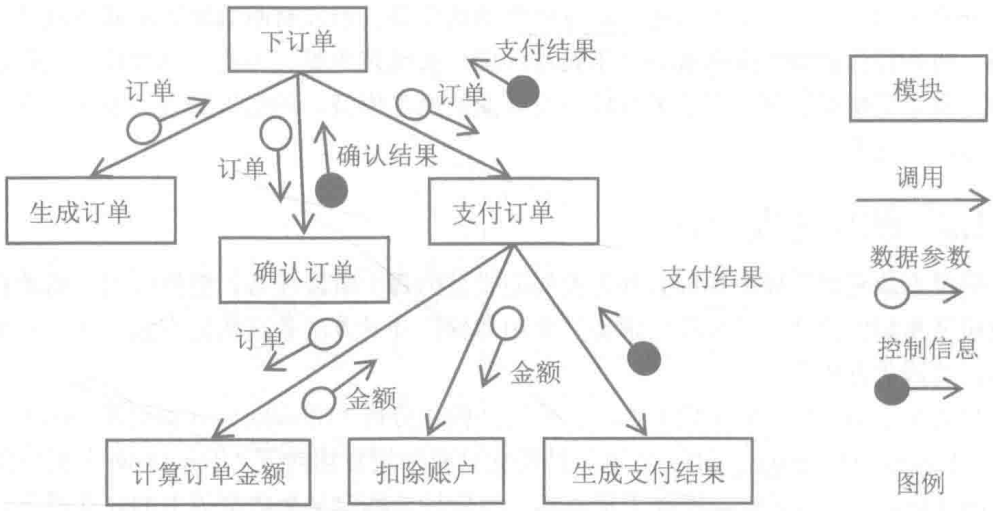

图 1.4　COS 系统订单模块部分结构图示例

使用面向结构（或过程）编程语言，如 Fortran、Basic、C 等，可以很好地实现结构化方法设计软件。

由于没有明确软件或程序设计的优化规范，也没有定义软件需求分析和设计文档标准；当软件系统规模或复杂度达到一定程度后，使用结构化方法进行软件开发会变得越来越困难。**“面向对象”提出了一种以对象为中心的软件系统分析、设计与实现的软件开发方法，能够在应对较大规模或较高复杂度的软件系统构建问题方面起到很好的作用。**

1.2　理解面向对象

对象以域（Field，也称为属性）的形式表达数据或状态，以方法（Method）的形式表达过程或行为；对象间可以相互访问或修改域，也可以调用行为；对象具有一定的生命周期（从初始化到最终消亡）；所有对象一起建立协作关系，向外部提供软件服务。如今，面向对象的编程语言已经成为应用最广的软件开发语言，如 Java、C#等。

1.2.1　面向对象的特征

在面向对象的概念中，对象具有状态变化，一般使用类（Class）定义对象的类型（Type）。类是对象的泛化和抽象，是静态的，可以通过面向对象编程语言进行描述。类经实例化生成具体的对象。面向对象的编程语言具有封装（Encapsulation）、继承（Inheritance）和多态（Polymorphism）等特征，用于实现软件系统业务模型时，具有天然优势。

1. 封装

封装是信息隐藏的一种形式。如果某个类将域或方法定义为私有（Private），则能够避免外部程序的干扰或错误访问。封装也能让程序员将业务相关性较强的数据或行为定义在一个类中，形成内聚度较强的代码单元，为软件解耦或复用提供便利。

下面的 Java 示例代码定义了 Employee 作为 COS 系统的员工类，该类将用户名（userName）、密码（password）域定义为受保护的（protected），登录行为（login()）定义为公共（public）方法，验证密码行为（verifyPassword()）定义为私有方法。Employee 类的封装避免了外部程序直接访问 Employee 类对象的用户名、密码等受保护的域和验证密码等私有行为，在一定程度上保护了数据和业务实现的安全；同时，登录行为和密码验证行为是具有强内聚特性的业务逻辑，Employee 类将其封装为一个独立的代码单元，减少了软件耦合，并增加了代码的复用便利。

Employee 类示例代码如下。

```
public class Employee{
    protected String userName;//用户名
    protected String password;//密码
        //其他域或行为
        /**
        * 登录行为
        */
    public boolean login() {
            //登录逻辑
         //return 语句
    }
        /**
         * 验证密码行为
         */
    private boolean verifyPassword() {
```

```
            //验证逻辑
    //return 语句
    }
}
```

2. 继承

继承是面向对象的重要特征之一，允许类以层次结构实现代码定义和复用。同时，继承也是物理世界中对象间关系的一种形式，能够使软件开发人员很容易地将目标领域的业务模型映射为技术模型。在继承关系中，被继承的类为父类，继承类为子类；子类可以继承父类的属性、行为和关系。

如下 Java 示例代码中定义了 Patron 作为 COS 系统的客户类，并且 Patron 类继承 Employee 类。程序员虽然没有在 Patron 类中定义登录行为，但 Patron 继承和复用了 Employee 类实现的登录行为，在一定程度上减少了代码开发成本。此外，这种复用方式还会为代码维护提供便利。如，当登录逻辑需求变更时，只需要修改父类 Employee 的 login()业务逻辑，即能实现所有 Employee 子类对象的登录行为修复。

Patron 类示例代码如下。

```
public class Patron extends Employee{
    private String name;//客户名称
    private PatronLevel level;//客户等级
        //其他域或行为
    //支付订单行为
    public void pay(){
        //支付订单逻辑
        //return 语句
    }
}
```

3. 多态

多态允许将父类对象的引用指向不同的子类对象，从而使得父类对象依据指向的子类对象执行不同的行为。多态也是一种抽象编程形式，可以向客户端屏蔽子类对象的差异，统一客户端对多态对象行为调用的形式，以达到客户端程序灵活适应需求变化的目的。

如下 Java 代码示例中的 PayOrder 类是 COS 系统的支付类，定义了支付订单行为（check ()）、抽象的支付费用行为（pay()）等。PayCard、PayRollDeduction 继承 PayOrder 父类，实现了卡和工资抵扣的支付费用行为 pay()。当客户端使用 PayOrder 类对象进行支付行为调用时，如果 PayOrder 对象引用指向 PayCard 类的对象，支付费用则为卡支付方式；如果 PayOrder 对象引用 PayRollDeduction 类的对象，支付费用即为工资抵扣支付方式。可见，对于调用 PayOrder 对象的客户端来说，支付行为表现为多态特性。

PayOrder 类示例代码如下。

```
public abstract class PayOrder {
    private Date payDate;//支付日期
    private float payAmount;//支付金额
```


试读结束：需要全本请在线购买： www.ertongbook.com