计 算 机 科 学 丛 书

原书第6版

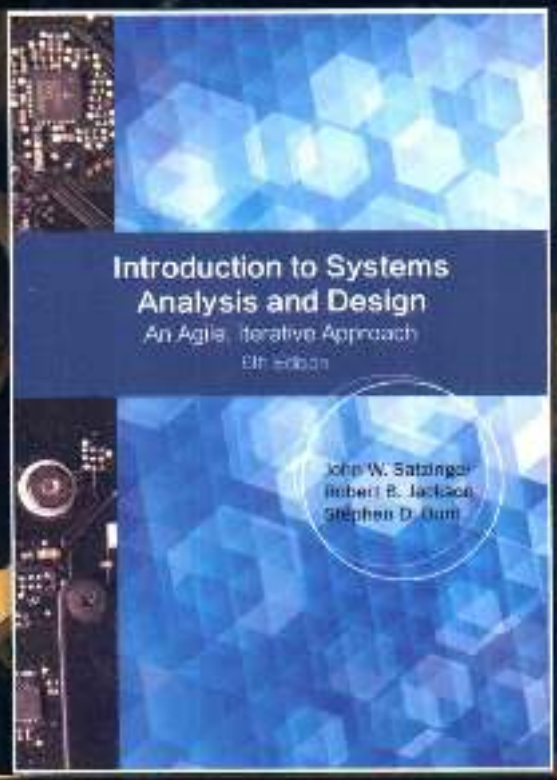
系统分析与设计

敏捷迭代方法

约翰 W. 萨茨辛格（John W. Satzinger）
[美] 罗伯特 B. 杰克逊（Robert B. Jackson） 著
史蒂芬 D. 伯德（Stephen D. Burd）
沈群力 译

Introduction to Systems Analysis and Design

An Agile, Iterative Approach Sixth Edition



机械工业出版社
China Machine Press

系统分析与设计 敏捷迭代方法 原书第6版

Introduction to Systems Analysis and Design An Agile, Iterative Approach Sixth Edition

本书主要讨论系统分析、系统设计和项目管理三大主题，第6版对全书内容进行了重新组织，以一个完整的系统开发案例贯穿始终，更加注重敏捷和迭代方法，并通过UML来模使面向对象方法的讲解更加细致和深入。新版在引入新方法的同时，依然保留了对传统结构化开发过程的介绍，并且突出了项目管理部分。

本书特色

- 经典和新颖的内容。既介绍系统开发的基本规则，也表指与时俱进的新技术，用新方法传授新概念，提高学生的参与程度和学习效率。
- 完整和实用的案例。除了贯穿始终的RMO案例，每章开篇还有精选的小案例。迭代项目的过程在第1章中便全景呈现，先观概貌，再探细节。
- 灵活和高效的教学。针对侧重面向对象、传统方法和项目管理的不同课程，教师可自主选择内容，学生在入门后亦可轻松实现自学。

作者简介

约翰 W. 萨茨辛格（John W. Satzinger） 美国密苏里州立大学计算机信息系统学院教授，拥有20多年的CIS和MIS大学课程教学和研究经验，研究方向包括系统分析与设计、图形用户界面设计和客户-服务器系统开发等。

罗伯特 B. 杰克逊（Robert B. Jackson） 美国杨百翰大学信息系统系退休教师，研究方向包括面向对象系统分析与设计、电子商务、项目管理等。

史蒂芬 D. 伯德（Stephen D. Burd） 美国新墨西哥大学副教授，从事管理信息系统、网络、数据库、硬件/软件课程的教学已逾30年。

CENGAGE Learning
www.cengageasia.com

销售电话：(010) 88378604
客服电话：(010) 88378991 88361066
购书电话：(010) 68326294 88378649 68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn

计算机/软件工程
ISBN 978-7-111-55641-1
9 787111 556411 >
定价：79.00元

计 算 机 科 学 丛 书

原书第6版

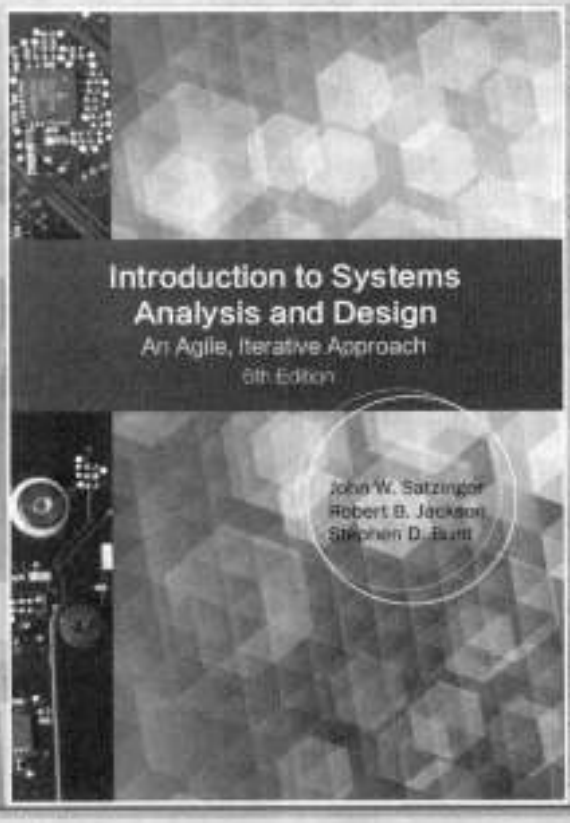
系统分析与设计

敏捷迭代方法

约翰 W. 萨茨辛格（John W. Satzinger）
[美] 罗伯特 B. 杰克逊（Robert B. Jackson） 著
史蒂芬 D. 伯德（Stephen D. Burd）
沈群力 译

Introduction to Systems Analysis and Design

An Agile, Iterative Approach Sixth Edition



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

系统分析与设计：敏捷迭代方法（原书第6版）/（美）约翰W. 萨茨辛格（John W. Satzinger）等著；沈群力译．—北京：机械工业出版社，2017.1
（计算机科学丛书）
书名原文：Introduction to Systems Analysis and Design: An Agile, Iterative Approach, Sixth Edition

ISBN 978-7-111-55641-1

I. 系… II. ①约… ②沈… III. ①信息系统－系统分析－高等学校－教材 ②信息系统－系统设计－高等学校－教材 IV. G202

中国版本图书馆CIP数据核字（2016）第317545号

本书版权登记号：图字：01-2012-8909

John W. Satzinger, Robert B. Jackson, Stephen D. Burd, Introduction to Systems Analysis and Design: An Agile, Iterative Approach, Sixth Edition.

Copyright © 2012 by Course Technology, a part of Cengage Learning.

Original edition published by Cengage Learning. All Rights reserved.

China Machine Press is authorized by Cengage Learning to publish and distribute exclusively this simplified Chinese edition. This edition is authorized for sale in the People's Republic of China only (excluding Hong Kong, Macao SAR and Taiwan). Unauthorized export of this edition is a violation of the Copyright Act. No part of this publication may be reproduced or distributed by any means, or stored in a database or retrieval system, without the prior written permission of the publisher.

Cengage Learning Asia Pte. Ltd.

151 Lorong Chuan, #02-08 New Tech Park, Singapore 556741.

本书原版由圣智学习出版公司出版，版权所有，盗印必究。

本书中文简体字翻译版由圣智学习出版公司授权机械工业出版社独家出版发行。此版本仅限在中华人民共和国境内（不包括香港、澳门特别行政区及台湾地区）销售。未经授权的本书出口将被视为违反版权法的行为。未经出版者预先书面许可，不得以任何方式复制或发行本书的任何部分。

本书封面贴有Cengage Learning防伪标签，无标签者不得销售。

本书是系统分析与设计的入门教材，以信息系统开发生命周期为主线，以贯穿系统开发始终的完整案例为引导，全面涵盖面向对象方法和UML，强调系统结构、用户界面和系统界面。在此基础上，还重点讲解了项目计划和项目管理方法。书中知识点巧妙融于实际案例，内容由浅入深，并配有大量习题，十分易于教学。

本书适合作为高等院校计算机、信息管理与信息系统等相关专业的本科生教材，也可作为系统开发或管理人员的参考书。

出版发行：机械工业出版社（北京市西城区百万庄大街22号 邮政编码：100037）
责任编辑：曲 熠　　责任校对：董纪丽
印　　刷：北京诚信伟业印刷有限公司　　版　　次：2017年1月第1版第1次印刷
开　　本：185mm×260mm 1/16　　印　　张：20.25
书　　号：ISBN 978-7-111-55641-1　　定　　价：79.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991 88361066　　投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259　　读者信箱：hzjsj@hzbook.com

版权所有·侵权必究
封底无防伪标均为盗版
本书法律顾问：北京大成律师事务所 韩光/邹晓东

文艺复兴以来，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的优势，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自1998年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与Pearson, McGraw-Hill, Elsevier, MIT, John Wiley & Sons, Cengage等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出Andrew S. Tanenbaum, Bjarne Stroustrup, Brian W. Kernighan, Dennis Ritchie, Jim Gray, Afred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, Abraham Silberschatz, William Stallings, Donald E. Knuth, John L.Hennessy, Larry L. Peterson等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专门为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com
电子邮件：hzjsj@hzbook.com
联系电话：（010）88379604
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

华章教育

华章科技图书出版中心

译者序

Introduction to Systems Analysis and Design: An Agile, Iterative Approach, Sixth Edition

本书以信息系统开发的生命周期为主线，以一个贯穿系统开发始末的完整案例为引导，全面涵盖面向对象方法和UML，使学生快速熟悉系统分析模型和技术，在此基础上介绍系统设计的概念，强调系统结构、用户界面和系统界面。本书前7章中介绍了大量有关系统分析与设计的知识，学生在理解这些内容的基础上，就会明白管理系统开发项目的重要性，包括敏捷开发项目。整本教材按照系统分析、系统设计、项目管理、系统支持的顺序，结合实际案例来设计教学内容，将知识点穿插于具体案例之中，并通过大量习题来加以巩固。这些教学内容的安排由浅入深，循序渐进地讲授系统开发的基本规则，同时，这一版还增加了一些最近几年出现的新技术、新方法，提倡用新方法来传授新概念。对初学者来说，这是一本很容易入门的教材。

本教材已在上海商学院信息管理与信息系统专业使用过3届，并配有相关教学文件及教学资料，如需下载，可以登录cc.sbs.edu.cn（上海商学院课程中心），点击“课程资源”栏下的“精品课程”，找到“信息系统分析与设计”，具体网址为http://cc.sbs.edu.cn/G2S/Template/View.aspx?action=view&courseType=0&courseId=26461。在“信息系统分析与设计”课程主页上有“资料下载”一栏，在此便可下载课件、实验材料、优秀学生作品等相关教学资料。

本课程建议采用54学时（27理论+27实验）或者72学时（36理论+36实验）来设计教学内容，实验部分围绕CASE工具的使用，教师也可以根据自己的教学需要灵活安排学时。我在实际教学中根据教学内容的要求，以Project及Visio为工具设计了约8个独立实验（30学时左右），其中安排了6~8个学时的Project软件应用实验，目的是从IT项目的角度来计划和管理一个完整信息系统的开发项目。在系统分析与设计阶段，采用Visio来绘制UML模型及其他系统设计图形，以实验来巩固课堂理论教学，相关实验指导及实验任务课程均可访问网站中的链接进行查看及下载。

本书的翻译得到了上海商学院高亚楠和乐嘉斌同学的帮助，还得到了上海商学院张晶老师及上海商校贾廷琦老师的修改和校正，嘉兴瑞恺公司的聂在手经理也参与其中。在翻译过程中，我们尽量尊重原文的意思，一些新的及不常见的专业术语的翻译虽经过多次斟酌，但可能还会存在一定的歧义，错误之处也在所难免，恳请读者批评指正。

沈群力

shenql@sbs.edu.cn

2016年11月

当我们撰写本教材的第 1 版时，系统开发正处于一个重要的转变过程中——从结构化方法到面向对象方法。在同类教材中，我们较早地系统性引进面向对象方法，第 6 版在讲解面向对象技术时将继续保持这一领先优势。

然而世界变化不断。如今，大量创新成果和技术已经牢牢嵌入系统开发领域。首先是无处不在的全球互联网。这导致了大规模的连通性，并且意味着当今的项目团队将分散于世界各地。此外，技术供应商巨头（比如微软）以及一些零散的小供应商为我们提供了功能丰富、多种多样的软件开发环境。

为了管理当今系统开发团队的广分布、快节奏、强连接及千变万化的环境，软件开发技术在不断升级，项目管理方法也在不断发展。基于基础项目管理原则，新的途径和理念提供了类似迭代、增量式开发等更新潮、更易成功的方法论。而这些都在本书中有所体现。

尽管本书全面涵盖了各种主题（如用例、面向对象建模、综合项目管理、统一建模语言以及敏捷技术）且在领域内持续领先，但也是时候采取新的教材设计了。第 6 版使用创新方法来讲授系统分析与设计，并利用了新兴的教学工具和技术。因此，本书不但使学生更易于学习系统分析与设计，而且使教学工作能够更顺利地进行，为学生和老师都提供了极大的便利。

在这一版本中，我们主要做了三个方面的更新。首先，我们讲授系统开发的基本规则，这些规则必须紧跟今天普遍连接的环境。其次，我们讲授并解释由于广泛互联而在当今获得使用的新兴方法和技术。最后，我们重新组织并修订了书中内容，使得它能更好地用新的方法传授新的概念。

例如，第 1 章介绍了一个新系统开发中的完整迭代方法。学生可以在学习抽象的原理或记忆专业术语前，从头到尾（通过实验和测试）看到完整的迭代方法。我们对这种新的方法感到兴奋，这些更新使新的教学素材和工具更适合系统分析与设计课程的教学。教师会发现这本书直观、丰富且易用。学生则会在本书中有更好的参与感和主动权。通过上课和教师提供的指导，学生可以自学书中呈现的大多数理论。这本书将会给老师和学生带来非凡的讲授和学习体验。

更新

这一版在许多方面做了更新，包括从传统方法到面向对象方法的重要概念，涉及用例驱动的面向对象方法（通过 UML 建模使其更详细且有深度），强调敏捷和迭代开发，并在项目管理中采用了新的概念。同时，材料的重组为学生学习系统分析与设计提供了更好的支持。

涵盖面向对象和传统的分析与设计

本书的一大特点是整合了一些重要的系统建模概念，这些概念在传统的结构化分析方法和面向对象的分析方法中都会涉及——基于用户目标和事件而建立的系统用例，加上对象 / 实体，这些都是系统问题域的一部分。我们用一章来介绍如何确定用例，再用一章介绍如何对关键的对象 / 实体建模，包括实体 – 联系图，同时强调 UML 域模型类图。与传统方法有

所不同的是，本书先假定学生从一开始就了解面向对象的关键概念，包括结构化概念，因为近年来许多教师更强调面向对象方法的教学。

全面涵盖 UML 和面向对象方法

本书中呈现的面向对象方法基于 OMG（对象管理组织）的统一建模语言（UML 2.0），它是由 Grady Booch、James Rumbaugh 和 Ivar Jacobson 发明的。模型驱动方法首先分析用例和场景，然后定义涉及用户工作的问题域类。需求建模包含用例图、领域建模、用例描述、活动图和系统顺序图。FURPS+ 模型就是用来强调功能性和非功能性需求的。

本书深入讨论了设计原理和设计模式，通过使用 UML 组件图和包图来建立系统体系结构的模型。特别关注使用 CRC 卡、顺序图和类图设计来实现用例，并详细地讨论了模型的细节设计。

涵盖项目管理

许多本科课程需要系统分析与设计这门课程所讲授的项目管理知识。为满足这一需要，我们通过“双管齐下”的方法来学习项目管理。第一，项目管理技术和任务在本书中得到了突出体现，我们将讲解系统开发生命周期的各种活动（包括迭代开发）该如何使用特定的项目管理方法。第二，用独立的一章完整地介绍了项目的计划和管理。

重组内容以实现高效学习

第 6 版的结构焕然一新，它从一个贯穿系统开发始末的完整例子开始，使学生快速了解系统分析模型和技术，然后介绍系统设计的概念，强调系统结构、用户界面和系统界面。学生会在前 7 章中了解许多有关分析和设计的知识。接下来，在学生理解了系统开发真正包含的内容后，就会明白本书理论知识的重点是管理系统开发项目，包括敏捷开发。最后，本书覆盖了详细的设计主题与部署主题，从而帮助学生更深入地了解现代方法，如统一过程（UP）、极限编程（XP）和 Scrum 开发方法。

篇章结构和用书建议

本书包含了很多简洁、现代和集中的主题，这些主题对信息系统开发者来说是必不可少且非常重要的。

本书中涉及三个主要的课程领域：系统分析、系统设计和项目管理。其他没有那么重要的领域则不会深入讲解，比如系统的安装、启用、测试和调整。除此之外，我们也采用了一些与其他书籍不一样的方法。因为学生已经在第 1 章中对系统分析与设计有了基本了解，所以我们加深了关于系统分析与设计的概念，并在以后的章节中增加了项目管理的主题。这使得学生在理解系统分析与设计的元素之后能更好地学习项目管理的概念。我们认为这对于学生学习此门课程是很有意义的。

第一部分：系统开发导论

第一部分即第 1 章，主要内容是系统开发的概述。第 1 章开篇清晰地解释了系统分析与设计的目标，然后用一个详细、具体的例子来阐述在典型的软件开发项目中都需要什么。很多参加编程课程的学生认为编程就是必须会开发软件和调试系统，本章和本书中的其他内容

会消除这个误解。

第二部分：系统分析活动

第 2 – 5 章详细阐述了系统分析。第 2 章讨论的是搜集某个商业问题的信息所需要的系统需求、分析活动和技术。只有充分理解问题所在，才有可能开发出相对适合的系统。第 2 章同时阐述了怎样确定利益相关者并使其参与进来，还介绍了模型和建模的概念。第 3 章和第 4 章讲述了以一种有用的方式捕获详细系统需求的建模技术。当我们提到信息系统时，两个核心概念是很有用的：一个是用例，它能正确定义出最终用户需要系统做什么；另一个是数据实体 / 域类，用户以此来完成他们的工作任务。这两个概念——用例和数据实体 / 域类——在任何一种系统开发方法中都是很重要的。第 5 章出现了更多深层次的需求模型，例如用例描述、用例图、系统顺序图以及状态机图。

这些建模技能使得分析员可以对用户需求进行深度分析并撰写需求说明。再次重申一下，系统分析的目的就是完全理解和说明用户需求。

第三部分：系统设计的要点

第 6 章和第 7 章的主要内容是系统设计和设计用户体验的基本概念。第 6 章全面介绍了系统设计和结构化设计的重要原则。这一章不但是各种设计原则的概述，而且为后续章节中所要学习的设计技术、任务、技巧和模型打下了基础。

第 7 章讲述了有关设计用户界面和系统界面的其他一些设计原则。设计用户界面是分析和设计的结合。与分析有关是因为它需要很多用户的参与，包括说明用户活动和期望。另一方面，这也是一个设计活动，因为它能创造出特殊的组件，这些组件会对编程结果的产生起到促进作用。必须精确设计界面，记录和其他用户的互动，这样才能作为最终系统的一部分而进行编程。当一个信息系统与另一个信息系统在没有人为干扰的情况下产生了交流和交互时，就需要设计系统界面。系统界面因为网站服务和云计算而变得越来越重要。

第四部分：项目和项目管理

通过这一部分，学生会对系统开发的所有组成元素有一个基本了解。第四部分会通过解释更多有关组织和管理开发项目的过程而将所有概念结合到一起。第 8 章描述了在当前环境下用不同方法进行的系统开发，包括几种重要的系统开发生命周期模型和敏捷开发。这是一个重要的章节，它能帮助你理解项目是怎样得以执行的。

第 9 章通过讲解项目计划和项目管理的基本原理扩充了这些概念。每个系统分析员都会参与到组织、协调和管理软件开发项目的过程中。除此以外，大多数优秀的学生最终会成为团队领导和项目管理者。第 9 章中呈现的这些原理对于一项成功的事业来说是必需的。

第五部分：高级设计和部署概念

第五部分会更深入地谈到系统设计、数据库设计，以及其他重要的有关有效且成功的系统开发和部署的论题。

第 10 章和第 11 章详细解释了用来设计软件系统的模型、技巧和技术。正像之前提到的，系统设计是一个相当复杂的活动，尤其是要把它做得很正确。这两章的任务是教会学生从简到难的各种技术，这些技术可以用来进行有效的软件系统设计。第 12 章描述了系统开

发的最终元素：最终测试、部署、维护和版本控制。

分析与设计课程的教学建议

分析与设计课程的教学方法有很多种，并且各个大学中教授这门课程的目的也不一样。在一些学院的信息系统专业，分析与设计课程是一门顶级课程，学生要将之前在数据库、电子信息和编程课程中学习到的知识运用到一个真实的分析与设计项目中。在其他的信息系统专业，分析与设计会作为系统开发领域的导论课，先于其他专业课。还有一些信息系统专业开设两门课程，第一个学期强调的是分析，而第二个学期则注重设计和应用。另外一些信息系统专业仅仅开设一门课程，同时讲授分析与设计。

由于要在强调传统结构的方法和面向对象方法中进行选择，所以这门课程的设计变得更加困难，这也使得它要依靠学校自身课程设置的优先顺序。此外，越来越多的迭代方法在开发中得到应用，这使得顺序排列分析与设计课题难上加难。例如，若采用迭代开发，则两门课程就不能轻易地分为分析和设计。

这些目标、课程内容、任务和项目会不断变化，我们所能提供的是使用这本书教授这门课程的一些建议。

面向对象分析与设计课程

面向对象设计在本书中得到了详细介绍。这门课程包括面向对象的分析和设计、用户和系统界面设计、数据库设计、控制和安全以及应用和测试。我们通常假设项目采用定制开发，包括 Web 开发。本课程强调迭代开发方法，包括三层结构、项目管理、信息收集和管理报告。为了学生学习的完整性，一学期制课程通常仅仅达到完成用户界面原型设计的目标。有时，这门课程也会横跨两个学期，在第二个学期会有一些实际系统的实施，以求获得更加完整的开发经验。其中，需要强调的是迭代开发。

对于强调面向对象开发的课程，我们推荐直接按照本书章节顺序进行讲授。

传统分析与设计课程

传统系统分析与设计课程提供了包括结构化分析、用户和系统界面设计、数据库设计、控制和安全以及实施和测试等全面的活动和任务。我们通常认为该项目采用定制开发，包括 Web 开发。本课程强调系统开发生命周期、项目管理、信息采集和管理报告。同样，一学期制课程通常局限于使学生完成用户界面的原型设计，而两学期制课程会在第二个学期通过一些实际系统的实施使学生有更完整的开发经验。

对于这门课程的设计，大纲中会合理地省略关于面向对象方法的细节。然而，面向对象的概念贯穿全书，因此学生仍可以不断熟悉它。

关于传统结构化方法的课程，我们推荐在教学大纲中省略第 5 章、第 10 章及第 11 章，同时可以适当补充一些关于系统分析员和传统需求的其他资料。

深入分析与项目管理

一些课程会涉及更有深度的面向对象系统分析方法，同时对结构化分析做了简明介绍，强调项目管理而不太关注面向对象设计。这些课程通常是研究生课程，并且通常认为设计和实施都包含在更加技术性的课程中。在一些案例中，软件包被认为更可能是解决方案而不是

定制开发的，所以定义需求和管理过程比设计行为更重要。

关于面向对象分析且深入讲解项目管理的课程，我们推荐在教学大纲中省略第 10~12 章，同时可以适当补充一些关于系统分析员、传统需求以及项目管理的其他资料。

相关支持[⊖]

本书提供支持教师在课堂上使用的教学工具，包括教师手册、答案、PowerPoint 演示文稿、图文件、在线测试以及在线章节等，可访问 login.cengage.com 查找和下载。

致谢

作为作者，我们得到了广大支持者的热情评论，在此表示由衷的感谢。在美国和加拿大，老师和学生认为我们的教材是目前最前沿、最灵活的。本书已被翻译成多种语言，在欧洲、澳大利亚、新西兰、印度和中国等地使用。我们由衷地感谢为本教材所有版本做出贡献的所有人。

第 6 版由编辑 Kate Mason 负责，他的工作包括招募开发编辑、与生产部门协商以加快写作和编辑进度、处理许多作者间的不确定性并调解冲突。开发编辑 Kent Williams 要将多种材料汇总并重新组织起来，有些材料看起来就像是疯狂的教授们随意放置的，因此这项任务是相当艰巨的。

最后，我们要衷心感谢为本书努力工作的评审者，从本书第 1 版到现在的第 6 版，他们自始至终都在提供建议。我们非常幸运有这么多知识渊博、观点鲜明的评审者。我们认真采纳了他们的建议，使得本书更加完善。所有版本的评审者包括：

Rob Anson, *Boise State University*
Marsha Baddeley, *Niagara College*
Teri Barnes, *DeVry Institute—Phoenix*
Robert Beatty, *University of Wisconsin—Milwaukee*
James Buck, *Gateway Technical College*
Anthony Cameron, *Fayetteville Technical Community College*
Genard Catalano, *Columbia College*
Paul H. Cheney, *University of Central Florida*
Kim Church, *Oklahoma State University*
Jung Choi, *Wright State University*
Jon D. Clark, *Colorado State University*
Mohammad Dadashzadeh, *Oakland University*
Lawrence E. Domine, *Milwaukee Area Technical College*
Gary Garrison, *Belmont University*
Cheryl Grimmett, *Wallace State Community College*
Jeff Hedrington, *University of Phoenix*
Janet Helwig, *Dominican University*
Susantha Herath, *St. Cloud State University*
Barbara Hewitt, *Texas A&M University*
Ellen D. Hoadley, *Loyola College in Maryland*
Jon Jasperson, *Texas A&M University*
Norman Jobes, *Conestoga College—Waterloo, Ontario*

⊖ 关于本书相关资源，有需要的读者可向圣智学习出版公司北京代表处申请，电话：010-8286 2096/95/97，电子邮件：kai.yao@Cengage.com 或 asia.infochina@Cengage.com。——编辑注

Gerald Karush, *Southern New Hampshire University*
Robert Keim, *Arizona State University*
Michael Kelly, *Community College of Rhode Island*
Rajiv Kishore, *The State University of New York—Buffalo*
Rebecca Koop, *Wright State University*
Hsiang-Jui Kung, *Georgia Southern University*
James E. LaBarre, *University of Wisconsin—Eau Claire*
Ingyu Lee, *Troy University*
Terrence Linkletter, *Central Washington University*
Tsun-Yin Law, *Seneca College*
David Little, *High Point University*
George M. Marakas, *Indiana University*
Roger McHaney, *Kansas State University*
Cindi A. Nadelman, *New England College*
Bruce Neubauer, *Pittsburgh State University*
Michael Nicholas, *Davenport University—Grand Rapids*
Mary Prescott, *University of South Florida*
Alex Ramirez, *Carleton University*
Eliot Rich, *The State University of New York—Albany*
Robert Saldarini, *Bergen Community College*
Laurie Schatzberg, *University of New Mexico*
Deborah Stockbridge, *Quincy College*
Jean Smith, *Technical College of the Lowcountry*
Peter Tarasewich, *Northeastern University*
Craig VanLengen, *Northern Arizona University*
Bruce Vanstone, *Bond University*
Haibo Wang, *Texas A&M University*
Terence M. Waterman, *Golden Gate University*

参与本书创作的所有人都希望，在这个不断变化的环境里，当你迎接分析与设计的挑战时能有最好的作为。

John Satzinger
Robert Jackson
Steve Burd

推荐阅读

软件工程：实践者的研究方法（原书第8版）

作者：Roger S. Pressman 等

ISBN：978-7-111-54897-3 定价：99.00元

软件工程：架构驱动的软件开发

作者：Richard F. Schmidt

ISBN：978-7-111-53314-6 定价：69.00元

人件（原书第3版）

作者：Tom DeMarco 等

ISBN：978-7-111-47436-4 定价：69.00元

设计原本——计算机科学巨匠Frederick P. Brooks的反思（经典珍藏）

作者：Frederick P. Brooks

ISBN：978-7-111-41626-5 定价：79.00元

推荐阅读

算法导论（原书第3版）

作者：Thomas H. Cormen 等

ISBN：978-7-111-40701-0 定价：128.00元

C程序设计语言（第2版·新版）

作者：Brian W. Kernighan 等

ISBN：978-7-111-12806-0 定价：30.00元

深入理解计算机系统（原书第3版）

作者：Randal E. Bryant 等

ISBN：978-7-111-54493-7 定价：139.00元

计算机组成与设计：硬件/软件接口（原书第5版）

作者：David Patterson 等

ISBN：978-7-111-50482-5 定价：99.00元

出版者的话
译者序
前言

第一部分 系统开发导论

第 1 章 从始至终——系统分析与设计概述 …… 2
1.1 软件开发以及系统分析与设计 …… 2
1.2 系统开发生命周期 …… 4
1.3 落基山运动用品（RMO）介绍 …… 4
1.4 迭代开发 …… 6
1.5 RMO 贸易展览系统的开发 …… 7
1.5.1 项目开始前的准备工作 …… 7
1.5.2 第一天的工作活动 …… 8
1.5.3 第二天的工作活动 …… 11
1.5.4 第三天的工作活动 …… 13
1.5.5 第四天的工作活动 …… 15
1.5.6 第五天的工作活动 …… 19
1.5.7 第六天的工作活动 …… 19
1.5.8 第一次迭代回顾 …… 21
1.6 后续内容导读 …… 21
1.6.1 第一部分：系统开发导论 …… 21
1.6.2 第二部分：系统分析活动 …… 22
1.6.3 第三部分：系统设计的要点 …… 22
1.6.4 第四部分：项目和项目管理 …… 22
1.6.5 第五部分：高级设计和部署概念 …… 22
本章小结 …… 23
复习题 …… 23

第二部分 系统分析活动

第 2 章 系统需求调查 …… 26
2.1 引言 …… 27
2.2 RMO 综合销售和市场营销系统项目 …… 27
2.2.1 现有的 RMO 信息系统与架构 …… 27
2.2.2 新综合销售和市场营销系统 …… 28
2.3 系统分析活动 …… 30
2.3.1 收集细节信息 …… 30
2.3.2 定义需求 …… 31
2.3.3 需求的优先级划分 …… 31
2.3.4 开发用户界面对话框 …… 31
2.3.5 与用户一起评估需求 …… 31
2.4 什么是需求 …… 32
2.5 模型和建模 …… 33
2.6 利益相关者 …… 35
2.7 信息收集技术 …… 38
2.7.1 与用户和其他利益相关者进行访谈 …… 38
2.7.2 分发和收集调查问卷 …… 41
2.7.3 检查输入、输出和流程 …… 42
2.7.4 观察和记录业务流程 …… 43
2.7.5 研究供应商的解决方案 …… 44
2.7.6 收集活跃的用户评论和建议 …… 44
2.8 用活动图记录工作流 …… 44
本章小结 …… 47
复习题 …… 48
问题和练习 …… 48
扩展资源 …… 49

第 3 章 用例 …… 50
3.1 引言 …… 51
3.2 用例和用户目标 …… 51
3.3 用例和事件分解 …… 52
3.3.1 事件分解技术 …… 53
3.3.2 事件类型 …… 54
3.3.3 定义事件 …… 55

3.3.4 使用事件分解技术 …… 57
3.4 用例和 CRUD …… 58
3.5 RMO 案例中的用例 …… 59
3.6 用例图 …… 60
3.6.1 用例、参与者和符号 …… 61
3.6.2 开发用例图 …… 63
本章小结 …… 64
复习题 …… 65
问题和练习 …… 66
扩展资源 …… 67

第 4 章 域建模 …… 68

4.1 引言 …… 68
4.2 问题域中的“事物” …… 69
4.2.1 头脑风暴法 …… 69
4.2.2 名词技术 …… 70
4.2.3 事物的属性 …… 71
4.2.4 事物间的关系 …… 72
4.3 实体－联系图 …… 73
4.4 域模型类图 …… 76
4.4.1 域模型类图符号 …… 76
4.4.2 有关对象类的更复杂的问题 …… 79
4.4.3 RMO 案例的域模型类图 …… 81
本章小结 …… 86
复习题 …… 86
问题和练习 …… 87
扩展资源 …… 88

第 5 章 需求模型的延伸 …… 89

5.1 引言 …… 90
5.2 用例描述 …… 90
5.2.1 简单的用例描述 …… 91
5.2.2 完全展开的用例描述 …… 91
5.3 用例活动图 …… 93
5.4 系统顺序图——确定输入和输出 …… 94
5.4.1 系统顺序图符号 …… 94
5.4.2 开发系统顺序图 …… 97
5.5 状态机图——确定对象行为 …… 100
5.5.1 复合状态和并发性 …… 101
5.5.2 开发状态机图的规则 …… 103
5.5.3 开发 RMO 状态机图 …… 104
5.6 需求模型的集成 …… 107
本章小结 …… 108
复习题 …… 108
问题和练习 …… 109
扩展资源 …… 111

第三部分 系统设计的要点

第 6 章 设计与设计活动的基本要素 …… 114

6.1 引言 …… 115
6.2 设计要素 …… 115
6.2.1 什么是系统设计 …… 116
6.2.2 设计的主要组件和层次 …… 116
6.3 系统设计的输入和输出 …… 118
6.4 设计活动 …… 120
6.4.1 设计环境 …… 121
6.4.2 设计应用程序结构和软件 …… 122
6.4.3 设计用户界面 …… 122
6.4.4 设计系统界面 …… 123
6.4.5 设计数据库 …… 124
6.4.6 设计安全和系统控制 …… 125
6.5 如何设计环境 …… 125
6.5.1 设计内部部署 …… 126
6.5.2 设计外部部署 …… 128
6.5.3 设计远程和分散的环境 …… 134
6.5.4 RMO 的企业技术结构 …… 135
本章小结 …… 136
复习题 …… 137
问题和练习 …… 137
扩展资源 …… 138

第 7 章 设计用户界面和系统界面 …… 139

7.1 引言 …… 140
7.2 用户界面和系统界面 …… 140
7.3 理解用户界面 …… 141
7.4 用户界面的设计概念 …… 144
7.4.1 提示性与可视性 …… 144
7.4.2 一致性 …… 145

7.4.3 快捷方式 …… 146
7.4.4 反馈 …… 146
7.4.5 完整的对话 …… 146
7.4.6 错误处理 …… 146
7.4.7 撤销动作 …… 147
7.4.8 减轻短期记忆负担 …… 147
7.5 从分析到用户界面设计的转换 …… 147
7.5.1 用例和菜单层次 …… 148
7.5.2 对话和故事板 …… 150
7.6 用户界面设计 …… 152
7.6.1 设计窗体和格式的指导原则 …… 152
7.6.2 网页浏览器用户界面的附加指导原则 …… 154
7.6.3 手持设备的附加指导原则 …… 157
7.7 确定系统界面 …… 158
7.8 设计系统输入 …… 160
7.8.1 自动化输入设备 …… 160
7.8.2 定义系统输入的细节 …… 160
7.9 设计系统输出 …… 161
本章小结 …… 167
复习题 …… 167
问题和练习 …… 168
扩展资源 …… 168

第四部分 项目和项目管理

第 8 章 系统开发方法 …… 170
8.1 引言 …… 171
8.2 系统开发生命周期 …… 171
8.2.1 系统开发生命周期的传统预测方法 …… 172
8.2.2 系统开发生命周期的新的自适应方法 …… 174
8.3 支持阶段 …… 176
8.4 方法、模型、工具和技术 …… 176
8.4.1 方法 …… 176
8.4.2 模型 …… 177
8.4.3 工具 …… 178
8.4.4 技术 …… 178
8.5 软件构造与建模的两种方法 …… 178
8.5.1 结构化方法 …… 179
8.5.2 面向对象方法 …… 182
8.6 敏捷开发 …… 185
8.6.1 敏捷开发的理论与价值 …… 185
8.6.2 敏捷建模原则 …… 186
本章小结 …… 188
复习题 …… 188
问题和练习 …… 189
扩展资源 …… 190

第 9 章 项目计划和项目管理 …… 191
9.1 引言 …… 192
9.2 项目管理原则 …… 192
9.2.1 项目管理的需求 …… 193
9.2.2 项目经理的角色 …… 193
9.2.3 项目管理和仪式 …… 194
9.2.4 项目管理知识体系（PMBOK） …… 195
9.2.5 敏捷项目管理（APM） …… 196
9.3 核心过程 1：确定问题并获得批准 …… 198
9.3.1 确定问题 …… 198
9.3.2 量化项目批准因素 …… 201
9.3.3 评估风险和可行性分析 …… 204
9.3.4 与客户一起评审并获得批准 …… 206
9.4 核心过程 2：计划和监控项目 …… 206
9.4.1 建立项目环境 …… 207
9.4.2 安排工作进度 …… 211
9.4.3 员工与资源分配 …… 214
9.4.4 评估工作过程 …… 215
9.4.5 监控过程与改正错误 …… 215
本章小结 …… 216
复习题 …… 217
问题和练习 …… 218
扩展资源 …… 220

第五部分 高级设计和部署概念

第 10 章 面向对象设计：设计原则 …… 222
10.1 引言 …… 223

10.2 面向对象设计：分析与实施的桥梁……223
10.2.1 面向对象程序概述……224
10.2.2 面向对象设计模型和过程……224
10.3 面向对象结构化设计……226
10.4 面向对象细节设计的基本原则……231
10.5 设计类和设计类图……235
10.5.1 设计类符号……235
10.5.2 设计类表示……236
10.5.3 开发初步的设计类图……238
10.6 用 CRC 卡进行细节设计……241
10.7 细节设计的基本原则……243
10.7.1 耦合……243
10.7.2 内聚……244
10.7.3 变量保护……245
10.7.4 间接……245
10.7.5 对象职责……245
本章小结……246
复习题……246
问题和练习……247
扩展资源……248

第 11 章 面向对象设计：用例实现……249

11.1 引言……250
11.2 多层系统的细节设计……250
11.3 用例实现和顺序图……252
11.3.1 理解顺序图……253
11.3.2 用例实现的设计流程……255
11.3.3 "创建顾客账户"用例的初步顺序图……256
11.3.4 "加入购物车"用例的初步顺序图……259
11.3.5 顺序图初步设计的指南和假设……263
11.3.6 开发多层设计……264
11.4 用协作图进行设计……268
11.5 更新和打包设计类图……269
11.5.1 包图——将主要部分结构化……270
11.5.2 三层设计的实现问题……271
11.6 设计模式……274
11.6.1 适配器……274
11.6.2 工厂……275
11.6.3 单例……276
本章小结……277
复习题……278
问题和练习……279
扩展资源……283

第 12 章 实现系统的可操作性……284

12.1 引言……285
12.2 测试……286
12.2.1 单元测试……287
12.2.2 集成测试……288
12.2.3 可用性测试……289
12.2.4 系统、性能和强化测试……290
12.3 部署活动……291
12.3.1 转换与初始化数据……291
12.3.2 培训用户……292
12.3.3 部署产品环境……295
12.4 计划与管理实施、测试和部署……296
12.4.1 开发顺序……296
12.4.2 源代码控制……300
12.4.3 打包、安装和部署组件……300
12.4.4 改动和版本控制……304
12.5 整体回看——再访 RMO……306
12.5.1 更新或代替？……306
12.5.2 最小化风险的阶段化部署……307
12.5.3 数据库开发和数据转换……307
12.5.4 开发顺序……308
12.5.5 文档和培训……308
本章小结……308
复习题……309
问题和练习……309
扩展资源……310

第一部分

Introduction to Systems Analysis and Design: An Agile, Iterative Approach, Sixth Edition

系统开发导论

第 1 章　从始至终——系统分析与设计概述

第 1 章

Introduction to Systems Analysis and Design: An Agile, Iterative Approach, Sixth Edition

从始至终——系统分析与设计概述

学习目标

阅读本章后，你应该具备的能力：

- 描述信息系统开发过程中系统分析与设计的目的。
- 描述迭代系统开发的特征。
- 解释系统开发生命周期的 6 个核心过程。
- 辨识计划项目过程中使用的重要文档。
- 辨识系统分析与系统设计中使用的重要图表。
- 解释系统开发中辨识用例的效用。
- 解释系统开发中辨识对象类的效用。

1.1 软件开发以及系统分析与设计

计算机在当代社会是很普及的，微芯片影响着我们生活的方方面面。我们生活的世界不仅普遍存在着计算，还到处存在着交流和联系。我们日常生活中相当大的一部分都要依靠计算机芯片、连接链路和应用软件。

在这个高科技社会中成长的你，能使用智能手机、便携式电脑、iPad、平板电脑、电子游戏设备等。你的手机提供日常（即使不是每小时）短信、推特、视频、快照、网络连接、游戏以及其他很多功能。你们当中的很多人已经开发出了属于自己的应用软件，或者你的朋友已经编写过笔记本电脑、智能手机、iPad 或者 Facebook 上的应用。你们中还有一些人已经开始了编程课程的学习，有些人已经知道怎样编写计算机应用软件。考虑到我们生活在一个遍布高科技的世界里，我们可能会问：什么是系统分析与设计？它为什么如此重要？新技术和新应用软件的开发要如何运用系统分析与设计？换句话说，系统分析与设计在高科技的解决方案和应用的开发中扮演什么角色？

首先，让我们明确两个重要定义。**计算机应用**是指在一个在计算机上执行的软件程序，且它能实现一些特定功能或者一系列相关功能。有时，计算机应用会简称为应用（app，例如 iPhone 应用或者 Facebook 应用）。**信息系统**是一些相互作用的部件的集合体，需要完成数据的收集、处理和存储（通常使用数据库）任务，最终提供所需的信息输出。尽管这两个定义有时是一致的，但应用通常只涉及计算机软件，而信息系统包括软件、数据库甚至还有相关的手工过程。计算机应用的一个例子是浏览器，可以通过浏览器连接到网络来玩游戏或者访问日历程序。图 1-1 显示了一个典型的移动数字设备。

为什么系统分析与设计在信息系统开发中这么重要？为了回答这个问题，让我们考虑一个类似的场景：创造一幢美丽建筑的艺术和科学。在这个场景中，有一个拥有想象力的购买者，还有一个构建出建筑的建筑者，以及一个作为购买者和建筑者之间桥梁的建筑师。建筑师帮助购买者发挥想象力，同时也要和建筑者就建筑规范进行沟通。在这个过程中，建筑师要运用各种工具在第一时间抓住购买者想象的内容，从而为建筑者提供指导——包括素描工

具、蓝图、扩展模型、详细规范甚至现场检查等。

图 1-1 一个典型的移动数字设备

就像一个建筑者不能没有计划就开始建造建筑物一样，编程者也不能一坐下来就直接编写代码。在编写代码和验证其能否满足购买者的需求之前，他们需要一些人（可能是他们自己）的帮助，这些人的作用就像建筑师一样——计划、捕获想象力、理解细节、细化需求。软件设计师需要理解并且抓住项目建立者想要的。通常我们把这个人称为系统分析员。如果你既是编程者也是分析员，可能无需记下需求分析也比较容易抓住相关细节。然而，在当今这个系统开发队伍分布在全球各地的时代，你可能仅仅负责编程的一部分，其余部分则会由分布在其他国家的队友完成。在分布式团队的情况下，通过书面文件来帮助你理解、捕获、解释以及详细说明软件应用是非常重要的。

简而言之，系统分析与设计（SA&D）为开发者提供工具和技术，这样开发者可以理解需求（业务需求）、捕捉想象力、定义解决方案、交流想象和解决方案、建立解决方案并在其他人建立解决方案时予以指导、确认解决方案满足需求并能在后面的启动方案中应用。

系统分析与设计包含全部技能、步骤、指导以及有助于实现系统编程的工具。系统分析与设计包括“软”技能，例如采访和与用户交流，同时也包括“硬”技能（更技术化），例如详细规范和设计方案。许多技术化的技能与创建模型有关，模型能够捕获规范或者定义解决方案。在本书中，你能学习到所有这些技能以及它们是怎样一起工作来开发信息系统的。

让我们用几个定义来总结一下。**系统分析**由一系列活动组成，这些活动能够使人理解并规范新系统能完成哪些功能。这里的关键词是“理解”和“规范”。系统分析远不止对问题的简单陈述。例如，一个顾客管理系统必须做到顾客跟踪、产品注册、监督保障、服务水平跟踪以及一些其他功能——这些功能都有无数的细节。系统分析详细描述了一个系统必须满足的需求或者必须解决的问题。

系统设计所包含的活动允许人们详细描述系统是如何解决需求的。这里的关键词是“解决”。换句话说，系统设计描述了系统“怎么样”工作。它详细地规范系统解决方案的所有组成部分，以及它们是怎样协同工作以提供预想解决方案的。

1.2 系统开发生命周期

对于一个新系统，最初的开发往往是作为一个项目来做的。这意味着要能够辨识、计划、组织及监督开发新系统需要的活动。我们把**项目**想象成一个计划任务，它有开始、结尾并产生了一些明确的结果。一些项目是非常正式的，然而还有一些是非常不正式的，以至于它们很勉强地被称为项目。

为了用分析、设计和其他开发活动来管理一个项目，需要一个项目管理框架来指导和整合项目团队的工作。**系统开发生命周期**（Systems Development Life Cycle，SDLC）确定了构建、实现以及维护信息系统所需的所有活动。通常，系统开发生命周期的所有活动包括：系统分析、系统设计、编程、测试和系统维护，以及成功地实现和部署新的信息系统所需的其他项目管理过程。

对于不同项目的不同需求，还有很多系统开发生命周期方法及其变体。然而，核心过程总是需要的，即使这些核心过程中还有不计其数的变体——每个过程怎样计划和执行以及过程怎样组合成为一个项目。下面是开发任何一个新的应用所需要的 6 个核心过程：

1. 确定问题或需求，并获得批准以向前推进。
2. 计划和监控项目——做什么、怎么做以及谁来做。
3. 发现和理解问题或者需求的细节。
4. 设计能解决问题或者满足需求的系统组件。
5. 建立、测试和整合系统组件。
6. 完成系统测试并部署解决方案。

有许多方法来实施系统开发生命周期这 6 个核心过程。**信息系统开发过程**常常是用于特定信息系统的实际方法。大多数要开发的信息系统是为了解决组织性的问题，这些问题通常非常复杂，因此计划和执行一个开发项目非常困难。实际上，许多项目到最后比初始设想要大得多——经常导致超预算并延期交付。在过去 10 年间，已开发出几个新的信息系统开发过程，这些过程增强了项目的成功率。其中一个更新颖和更高效的开发过程被称为**敏捷开发**。敏捷开发的基本原理是，团队成员和用户都不能完全理解新系统的问题和复杂性，因此项目的计划和执行必须对不曾预料到的问题负责。这种开发方法必须敏捷且灵活，必须允许、期待甚至拥抱开发过程中产生的变化和新需求。

可能理解这些概念最好的方法是看看其在一个完整例子中的实现方式，这就是本章的目标。这里，我们将会运用一个小型的信息系统应用来展示所有 6 种核心过程（在现实中和在课本中一样可行）。我们将会举例说明一种能将各种活动组织成一个实际的工作项目的方法；换句话说，我们会给大家展示信息系统开发过程的一个版本。通过将所有方法应用于一个非常简单的项目，你将会更加容易地学习和理解本书后面出现的复杂概念。我们的项目是落基山运动用品，它是户外运动服装零售商和制造商。

1.3 落基山运动用品（RMO）介绍

RMO 是一家大型零售公司，专门生产不同种类的户外运动服装和相关配饰。21 世纪初期，落基山和美国西部各州的运动用品市场在娱乐活动中出现巨大增长，随着人们户外运动兴趣的增长，冬夏运动服装市场迅速扩大。滑雪、单板滑雪、山地自行车运动、滑水、喷气式滑水车运动、水上速跑、慢跑、徒步旅行、ATV 自行车运动、野营、爬山、蹦极，人们对这些运动的兴趣在这些州中都有巨大的提升。人们需要为这些运动购买合适的运动服装，

因此，RMO 为了迎合市场需求而扩大了运动装的生产线，同时还增加了一条生产线，主要生产时尚的运动服和配饰来完善它的产品。

公司的发展创造了可喜的纪录，包括邮购、实体购物和网上购物。最初，RMO 是在犹他州帕克城等地区把衣服卖给当地的服装店。在 20 世纪 80 年代后期和 90 年代早期，它开始通过邮购和电话订购直接向顾客销售服装。1994 年，它开了第一家实体店，不久之后在西部迅速扩大到了 10 家零售店铺。去年，零售商店的收入达到了 6700 万美元，电话订购和邮购的收入达到了 1000 万美元，网购的销售额达到了 2 亿美元。虽然在美国东部一些地区和加拿大已经开拓了一部分市场，但大多数销售还是在西部。

图 1-2　RMO 冬季目录

RMO 主要生产自己的户外运动服装系列。然而，为了在它的零售店中提供更完整的户外服装系列，它也销售其他品牌的户外运动装。除此之外，其他种类的服装和配饰，如鞋类、皮革服装和特殊的运动装也能在零售店或者网上商城中看到。

图 1-2 展示了 RMO 邮购商品的目录示例。尽管只有少量商品通过邮购和电话实现销售，但收到目录仍然会鼓励顾客去网上购买商品，因此 RMO 才会继续制作和邮寄它的精简目录。

图 1-3 是一个典型的网上系统订购页面。

图 1-3　RMO 网上订购示例

贸易展览

为了保证生产线的现代化和流行化，RMO 的采购代理人会参加世界各地的服装展和面料展。RMO 的采购系统拥有良好的跟踪记录，可以预测哪个产品会销售得较好。此外，RMO 一直在寻找能拓展其生产线的新产品和配饰。

当采购代理人出席一个贸易展览时，他们会频繁地发现想要加到春季、夏季或者冬季待售品中的各种商品。过去，当 RMO 的购买者想要下订单时，他们会在贸易展览中和销售者交换联系信息，等到回到办公室后再通过电子邮件和电话进一步制定合同和购买订单。然而，为了迅速完成订单，RMO 现在已经启动了一个项目：开发一个系统来收集和跟进它的供应商信息以及新增加到销售规划中的待售品的信息。

1.4　迭代开发

迭代开发是一种系统开发的方法，可以这样说，系统以一种几乎有机的方式“成长”。首先开发核心组成部分，然后再把其他组成部分加进去。之所以称为“迭代”，是因为 6 种核心开发过程在一遍又一遍地重复以增加整个系统的额外功能。换句话说，有一个大项目，它由许多小项目组成，并且这个信息系统是逐渐成长起来的。

图 1-4 说明了怎样执行一个迭代敏捷项目。这是一个示例图。真实的项目可能会有很大不同。通过这个图你可以看到 6 个迭代。一个迭代就像是一个小型项目，因为它有完整的结果和严格的时间界限，通常会持续 2 ~ 4 周。在图的左边，你可以看到 6 个核心过程。图里面的圆形堆代表了在那个迭代过程中为核心过程付出的工作量。圆形堆的面积是一个近似指标，这个指标显示了一个特定核心过程在迭代过程中花费的工作量。例如，在图 1-4 中，迭代 1 主要集中于确定问题和计划项目。发现、设计、建立和测试也可能做了，但是占的比例就比较小。对于这个迭代，在部署系统方面什么也没有做。

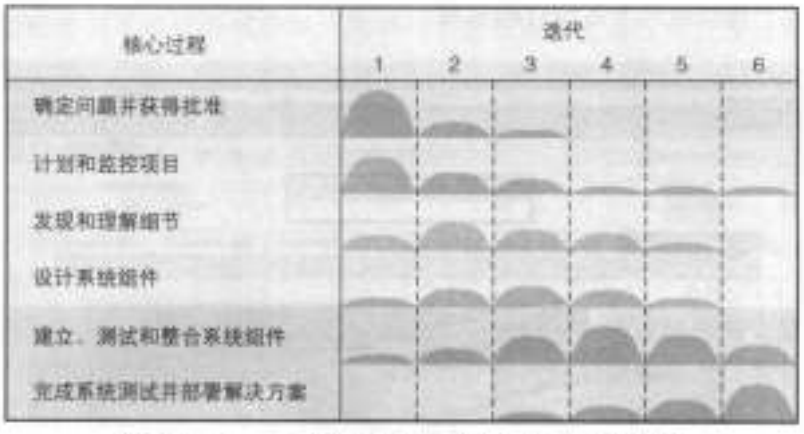


图 1-4　对一个典型项目进行迭代的 6 个核心过程

迭代开发有几个好处。第一，系统的一部分能很快被部署。如果存在可提供基础支持的核心功能，这些功能就能被部署到一个早期的迭代中。第二，取出一小部分首先开发，这样能在早期发现项目的困难问题。现今的许多系统庞大且复杂，记住和理解每个部分是不可能的。通过集中于一小部分，需求就减少了，而且容易抓住和解决需求。最后，用迭代方式开发系统使得整个开发过程更加灵活，而且能够处理贯穿整个项目的需求。

迭代开发的一个核心元素是选择能在2－4周完成的解决方案系统的一部分。在一个迭代期间，通常包含全部的核心开发过程，包括编程和系统级测试，因此这个结果是工作系统的一部分，即使它可能仅仅拥有最终被需要的功能的一部分。

1.5 RMO贸易展览系统的开发

对于示例项目“RMO贸易展览系统”的第1个迭代，我们将如此安排：目标是用6天时间完成这个迭代。然而，我们的主要目标是介绍这6个核心过程的概念和技术。因此，在一些实例中，相比在一个真实项目的第1个迭代中所做的，我们可能会涉及核心过程的更深层次。用所有必要的详细内容在仅仅6天之内完成一个完整的迭代是不切实际的，但是这也应该是一个不错的学习经历。6个系统开发生命周期的核心过程和项目的6天不会是一一对应的，但是我们的项目中包括系统开发生命周期的所有核心过程。

大多数新的应用要求一个项目拥有几个迭代。在第一个迭代中，通常有三个主要目标。

第一个目标是使项目获得批准。第二个目标是得到一个系统完整版本的清晰视图——体现所有的主要功能和数据需求。第三个目标是决定细节规范和开发这个系统一部分的解决方案（即分析、设计、建立和测试系统的一部分）。

在我们的项目中，我们会接触所有的这些目标。我们会给出一个系统可视化文档的示例，然后开发整个系统的一部分。我们已经限制了这个新系统的范围，因此我们可以在一个迭代中完成。需要注意的是，把项目划分为天以及每天的活动是任意的。划分和组织工作有很多种方法。下面的组织是完全可行的，但是它不是组织项目的唯一方法。

1.5.1 项目开始前的准备工作

在这个项目真正开始之前，RMO采购部门的领导会和系统分析师一起确定和记录具体业务需求，同时也定义一个具体的项目目标。RMO的管理层会评审重要的项目目标并审批预算。每个组织在一个项目开始之前都得审批预算。一些组织会通过正式的过程来审批项目；而有些组织则不采用正式的过程。通常，在项目开始前组织有两个目标是必须决定的：

- 确定问题，记录解决方案系统的目标。（核心过程1）
- 获得批准以开发这个项目。（核心过程1）

系统可视化文档

由于RMO有着很多新的项目，因此开发了系统可视化文档，用来识别对公司有益的功能和将会包括在系统中的功能。通常，这分为两步来做：开发一个初步的收益声明，然后增加详细花费和收益的估计。图1-5是针对这个项目的系统可视化文档。

就像前面描述的，落基山运动用品需要一个便携式的系统，采购代理人能在出席各种产品和服装面料展览时使用。这个系统需要满足两个主要需求。第一，它必须有能捕捉到供应商和产品信息的功能。第二，它必须能和总部的系统相互交流，因为这些贸易展览会在世界各地各种会场举办，所以各种各样的连通性设备都需要。

初步调查使用的各种设备，包括笔记本电脑、iPad和智能手机。虽然智能手机好像拥有最好的连接选择，但是小型的智能手机在观看照片的细节方面有难度；拥有先进技术的iPad和其他相似的便携式设备也是切实可行的选择。然而，鉴于智能手机和手写板的相似性，看起来开发一个应用是很可行的，这个应用会在两个设备中的一个执行。每个采购代理人可以选用他中意的设备。

系统可视化文档

RMO 贸易展览系统

问题描述

贸易展览已经成为新产品、新潮流和新面料的一个重要信息来源。除了很大的户外服装和面料供应商，还有许多其他小型的供应商。对于落基山运动用品来说，在这种贸易展览也在发展的同时，抓住有关这些供应商的信息是非常重要的。获得落基山运动用品计划购买的特定商品的信息也是很重要的。除此之外，如果在这个贸易展览中可以获得高质量的产品照片，那么对于网上产品页面的创新会有很大的促进。

建议开发和部署一个新系统，这样各领域的采购代理人能和总部更快地交流感兴趣的供应商和特定产品。这个系统应该在便携式设备上部署。

系统能力

这个新系统应该具备的性能：

- 收集和存储关于制造商 / 批发商（供应商）的信息。
- 收集和存储关于销售代表和其他核心人员（对每个供应商来说）的信息。
- 收集产品信息。
- 为产品照相（或上传产品的常用图片）。
- 能在没有连网的情况下独立工作。
- 能连接无线网（互联网）并传输数据。
- 能连接电话线并传递数据。

商业收益

期待这个新系统的部署能给落基山运动用品带来以下商业收益：

- 增加贸易展览出席者和总部之间的交流，从而提高采购订单决策的质量和速度。
- 维护供应商和核心人员的正确且实时的信息和图片，从而促进与供应商的快速交流。
- 维护新产品的正确且实时的信息和图片，从而促进目录和网页的开发。
- 加速新产品的采购和下单，从而能快速地掌握趋势和增加产品的实用性。

图 1-5 贸易展览系统可视化文档

在项目开始前的准备工作的最后，所有核心人员以及执行管理层的代表会组织一个会议。在会议中会决定是否继续这个项目并且预算需要的资金。

1.5.2 第一天的工作活动

落基山运动用品——供应商信息子系统

这个项目真正开始的第一天，实际上就是项目计划的第一天。通常，第一个工作活动就是让项目团队回顾系统可视化文档并核实初步的工作仍然有效。回顾项目的范围，熟悉要解决的问题，然后再计划剩余项目的迭代和工作活动。第二个系统开发生命周期核心过程是计划项目，包括商业分析和项目管理工作。所有这些话题都会在以后的章节中深入探讨。以下工作活动会在第一天完成：

- 决定需要的主要部分（功能性区域）。（核心过程 2）
- 定义迭代过程，将每个功能区域安排到一个迭代。（核心过程 2）
- 决定团队成员和相应职责。（核心过程 2）

计划整个项目和项目迭代

在一个项目计划中需要考虑无数细节。对于我们的项目，我们只会集中在本质上。在以后的章节中我们会更加精心地描述项目计划。

这个项目团队会与用户见面来回顾整个商业需求和新系统的目标。系统可视化文档作为这些讨论的开始部分。系统能力列表常常为决定整个项目计划提供基础信息。第一步是将这个系统分成几个子系统或组成部分。**子系统**就是整个系统的一部分。建立在系统能力列表的基础上，项目团队能识别以下这些功能子系统：

- 供应商信息子系统。
- 产品信息子系统。

供应商信息子系统会收集和维护制造商或者批发商和为他们工作的合同人员的信息。产品信息子系统会捕捉各种产品的信息，包括详细的描述和照片。

下一个步骤是识别哪个子系统会在相应的订单中被开发。我们会考虑许多议题，例如，各种任务之间的依赖关系、顺序与并行开发、项目团队的实用性和项目的紧要性。在我们的例子中，团队会通过供应商信息子系统这个首要迭代决定项目是否会在不断发展的潮流中前进。

计划首个迭代

每个迭代就像一个系统开发小型项目。随着范围缩小到在迭代中被开发的组成部分，核心过程的早期描述全部都能得到应用。迭代的计划过程由以下三个步骤组成：

- 识别任务要求的迭代。
- 组织和使这些任务按日程顺序排列。
- 识别需要的资源（尤其是人），安排人员负责相应的任务。

第一步是识别或者试着去识别所有单独的需要完成的任务。由于这些任务都已经被识别，因此它们会被编制和组织。有时候，这个被组织好的任务列表被称为工作分解结构。图1-6所示为用于这个迭代的工作分解结构。

工作分解结构

I. 发现和理解问题的各个方面的细节。
  1. 会见采购部门经理。～3小时
  2. 会见几个采购代理人。～4小时
  3. 识别和定义用例。～3小时
  4. 识别和定义信息需求。～2小时
  5. 开发用例的工作流程和描述。～6小时
II. 设计问题解决方案的组成部分
  1. 设计输入界面、输出界面和报告。～8小时
  2. 设计和建立数据库（属性、主键、索引）。～4小时
  3. 设计整体架构。～4小时
  4. 设计编程细节。～6小时
III. 建立组成部分和将所有细节整合到解决方案中
  1. 编码直到测试界面层的程序。～14小时
  2. 编码直到测试逻辑层的程序。～8小时
IV. 测试所有系统层的测试和部署解决方案
  1. 执行系统功能测试。～5小时
  2. 执行用户接受测试。～8小时

图1-6 工作分解结构示例

部分精力会被用于试着评估每项任务花费的时间。因为这个项目的时间范围是有限制的（只有6天），所以所有的评估都会精确到小时。这些评估不包括那些不在团队中的人员所增

加的时间。然而，对于在团队中的人员，这个评估包括初步工作的时间、讨论的时间以及回顾和检查工作分解结构的正确性和精确性的时间。

下个步骤是将这些任务组织成一个日程表。再一次，我们会非常正式地并且用一个复杂的项目调度工具或者按照我们认为必须要做的顺序来排列这些任务。建立日程表的一个重要部分是识别任务之间的依赖关系。例如，在我们识别出信息需求之前设计数据库是没有意义的。但是许多任务会并行完成。

单个迭代的好处是我们可以使日程安排非正式化，而且我们会一天天地调整工作来应对具体过程中突发的复杂情况。

对于我们的项目，我们不会建立一个完整的日程安排。你将会在以后的章节中学习到要怎么做。然而，为了组织 6 天的项目，我们已经从工作分解结构中获取任务并且将它们按顺序排在每一天，我们称为工作顺序草案，如图 1-7 所示。为了开发一个正式的日程表，项目领导会用这个图来安排人员去完成任务，同时也将任务放在一个绘制着日历日期的特定进度表上。

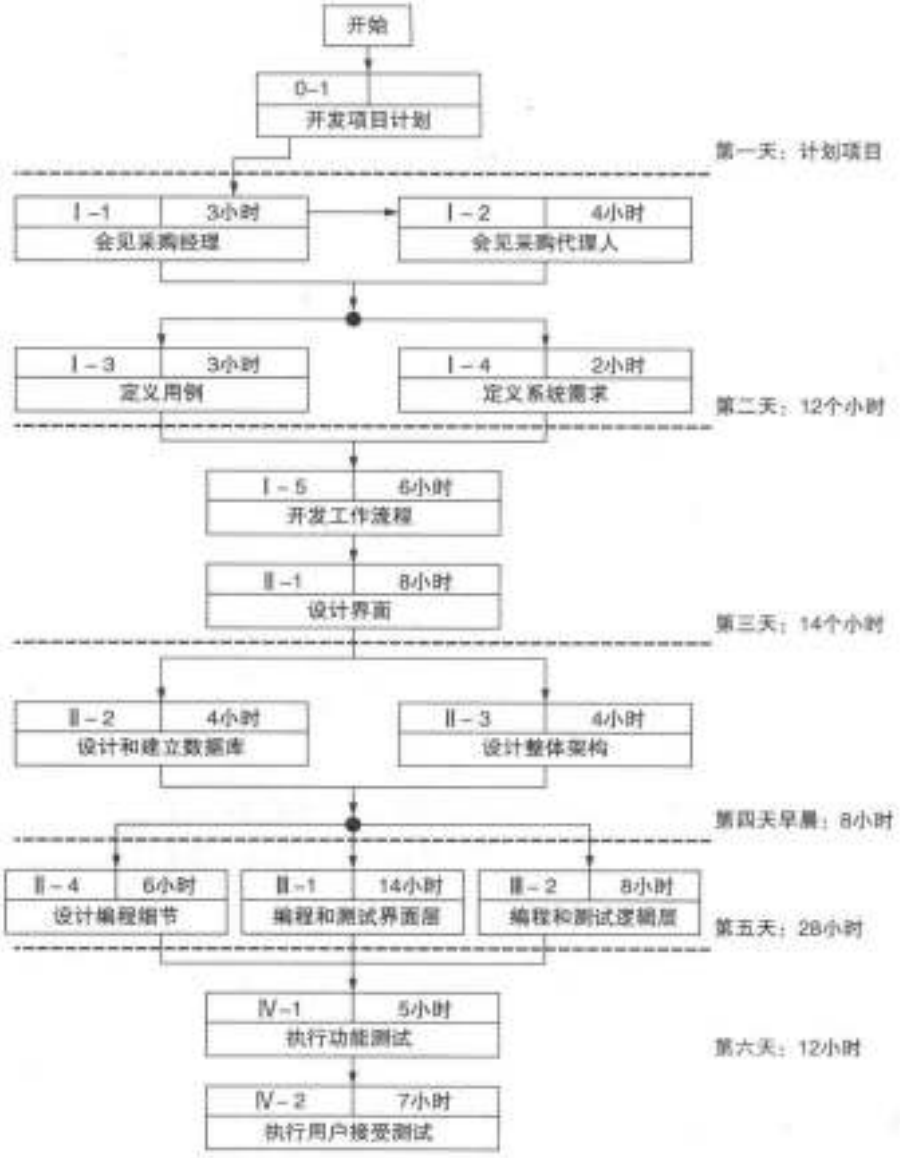


图 1-7　工作顺序草案

你应该注意到工作的顺序和工作的依赖关系已经用局部精度在这个图上显现出来了。例如，我们展示了编程是直到设计完成后才开始的。然而，实际上，在这两个工作之间可能有一些重叠部分。工作顺序草案的好处是三重的。第一，它帮助团队组织工作，在开始编程之前就有足够的时间来考虑决定性的设计主题。第二，它提供了一个衡量标准来看这个迭代是否在日程安排上。例如，如果会见采购代理人花费了一天或者多于一天的时间，那么团队会提早知道这个迭代将会花费比预期更多的时间。第三，如果项目会在这个日程安排上停留一段时间，那么项目的领导者可以看到编程需要更多的资源。因此，项目领导者能提早开始组织资源来帮助部分项目。很明显，甚至是简单的依赖关系图也能帮助项目经理计划和组织工作。

1.5.3 第二天的工作活动

第一天的工作活动包括计划和组织项目。第二天的工作活动则包括系统分析，这能帮助我们理解和记录需求。在第二天，我们会详细说明这些功能。这些工作活动包括：

- 做初步实情调查的工作来理解需求。（核心过程3）
- 开发一个初步用例列表和一个用例图。（核心过程3）
- 开发一个初步的类列表和一个类图。（核心过程3）

实情调查和用户参与

在项目开始之前，会先开发一个初步的功能板的定义。现在正是检查那些功能的特殊性和定义用户真正需要这个系统去做什么的时候。第一个步骤是识别能帮助定义这些细节的核心用户。显然，采购部门的经理会是第一拨要会见的人中的一个。他可能会指定一或两个聪明的采购代理人，这两个人可以在正在进行的基础部分与团队成员一起工作，开发说明书并核实系统是否按照要求运行。所有成功的项目依靠的都是大量的用户。在第2章，你会学习到更多关于辨识核心利益相关者的知识。

有很多种类的技术能确保实情调查的完整性和全面性。这些包括采访核心用户、观察现存的工作过程、回顾现存的文件和系统以及调查其他公司和其他系统。

识别用例

用例记录了一个简单的用户触发商业活动和系统对这个活动的回应。例如，让我们假设采购代理人出席贸易展览并且发现了一些新的轻型夹克衫，这类衣服与落基山运动用品的商品供应很相似。可能这个采购代理人必须做的第一个任务是调查出这个供应商以前是否与落基山运动用品合作过。因此，这个商业活动需要贸易展览系统能“查询供应商”。引导我们来运用这个系统的活动是很重要的，但是我们不能将它们作为商业活动来识别，直到贸易展览系统得到运用。因此，用例这个术语是指系统运行的一个例子或者情况。能帮助你识别用例的一个很好的方法是“采购代理人‘用’这个系统来‘查询供应商’”。

有些多功能的设备被用来识别用例，在本书中你会学习到。图1-8是整个贸易展览系统中的用例的一个初步列表。当这个项目团队在头脑风暴会议中会见采购代理人时，他们一起识别每个商业活动，在这个过程中采购代理人能运用这个系统。然而，因为第一个迭代只是集中于供应商信息子系统，所以项目团队也将只会集中注意力在列表上的前四个用例上。

识别对象类

对象类识别的是那些在真实世界中的信息，这些信息是这个系统需要知道和跟踪的。为了发现对象类，我们寻找系统应用或者捕捉的所有的对象或者信息。对象会以所有形式存在，从可触知的项目（就像是你能看到和触摸到的货物产品）到更加抽象而且不能触摸的概

念（像是一个命令），尽管不可触知但是确实存在的。

用例	描述
查询供应商	通过供应商的名称来寻找供应商的信息和联系方式
登记/更新供应商信息	登记（新）或者更新（现存的）供应商信息
查询联系人	通过联系人姓名来寻找联系人的信息
登记/更新联系人信息	登记（新）或者更新（现存的）联系人信息
查询产品信息	用描述或者供应商的名称来寻找产品信息
登记/更新产品信息	登记（新）或者更新（现存的）产品信息
上传产品图片	上传货物产品的图片

图 1-8　用例列表

对象类是在和采购代理人的讨论中被识别的，而这个讨论的过程是通过寻找能描述事物类别的名词。例如，代理人会经常讨论供应商、商品或者库存项目。识别用例和属性的更多细节会在以后的章节中出现。

图 1-9 说明了哪个名词已经被定为贸易展览系统的基础对象类。这些属性是那种能帮助定义和描述一个对象类的描述规范。

对象类	属性
供应商	供应商名称、地址、描述、注释
联系人	姓名、性别、地址、电话、电子邮件、职位、注释
产品	类别、名称、描述、注释
产品图片	ID、图片

图 1-9　对象类列表

除了只提供对象类的一个列表之外，系统分析员会经常开发一个类、属性和它们与另外的类之间的关系的视觉图。这个图被称为类图。图 1-10 举例说明了贸易展览系统的一个类图。

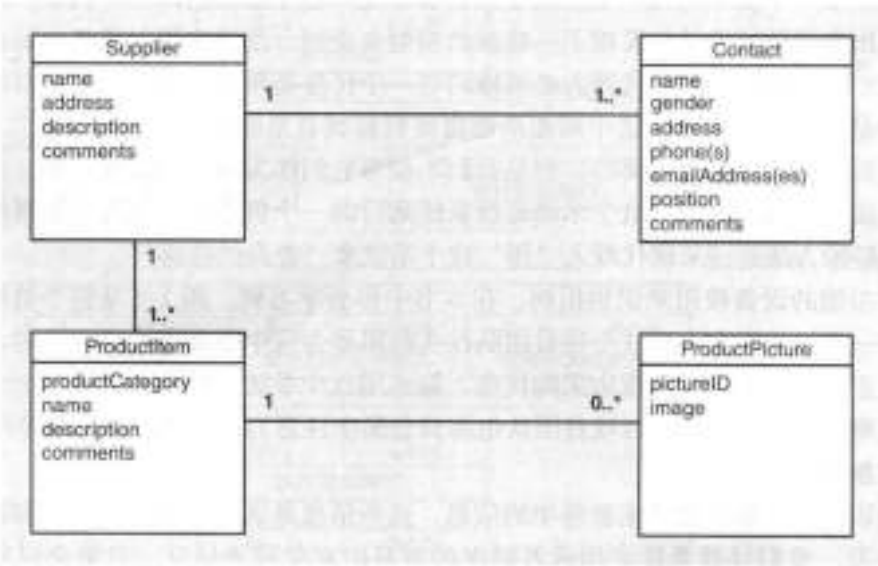


图 1-10　贸易展览系统的初始类图

每个盒子就是一个类，可视为一个特定的对象集合，这个集合对系统来说是很重要的。每个类的重要属性也包括在每个盒子中。这些代表了能被系统维护的每个对象的详细信息。可以看到一些类之间有线条。这代表了这些需要在系统中被捕捉的类之间的关系。例如，联系是指为一个特定的供应商工作的人。一个具体的例子就是 Bill Williams 是南太平洋运动服装公司的联系人。因此，系统需要为 Bill Williams 和南太平洋运动服装公司建立关联。关联线记录了这个需求。

类图是一个有力并且频繁运用的方式，用于理解和记录系统要求的信息。贸易展览系统是很简单的，它只有四个类，而且其中两个还是属于供应商信息子系统的。大多数现实生活中的系统是比较大型的，并且拥有很多类。

1.5.4 第三天的工作活动

第三天工作的目的是详细分析那些被选择在第一个迭代中实施的用例和类。在第三天中，我们仍旧执行系统分析的进程。我们仍会试着去理解对系统来说更详细的要求，包括以下这些内容：

- 深入地执行实情调查来理解细节内容。（核心过程 3）
- 理解和记录每个用例的详细工作流程。（核心过程 3）
- 用界面和报告定义用户体验。（核心过程 3、4）

用例能帮助项目团队组织工作。我们还要完成每个用例的下钻活动。就像早期提到的那样，这些用例是关于供应商信息子系统的：

- 查询供应商。
- 登记 / 更新供应商信息。
- 查询联系人信息。
- 登记 / 更新联系人信息。

项目团队会开发针对每个用例的工作流程，以更好地理解它是怎样工作的，并识别需要开发哪些界面或哪些报告。由于团队会获取更多的细节信息，因此一些最初的分析会被发现是不完整的或者是不正确的。这是做些探索的好时机——肯定比在代码写完后再做好很多。

图 1-11 展示了一个简单的用例图。它显示出了上述四种用例和将会初步执行那个功能的人。这个图表的意义和开发方法会在以后的章节中进行讨论。

用例描述和工作流图的开发

记录一个用例的细节有多种方法。其中一种在本书中将会学到的方法叫作用例描述。另一种方法是开发工作流图，他以用例的方法展示各个阶段的进程。各种方法的目的是记录用户和系统之间的交互作用（也就是，用户如何影响和使用系统来实施一个简单用例的特定任务）。

让我们对于用例制定一个工作流图。制定工作流图需要用到一种简单的图——活动图。图 1-12 阐明了“查询供应商”用例的工作流图。在图中椭圆代表任务，菱形代表决策点，箭头表示图标的序列流。序列代表各自要执行哪些任务。工作流图常常易于理解。

穿越中心线的箭头代表了系统和用户之间的相互作用。这些都是至关重要的，因为开发人员必须提供一个屏幕或网页页面来捕获或显示信息。穿过中心线的箭头确定了数据元素，它们是用户界面的一部分。

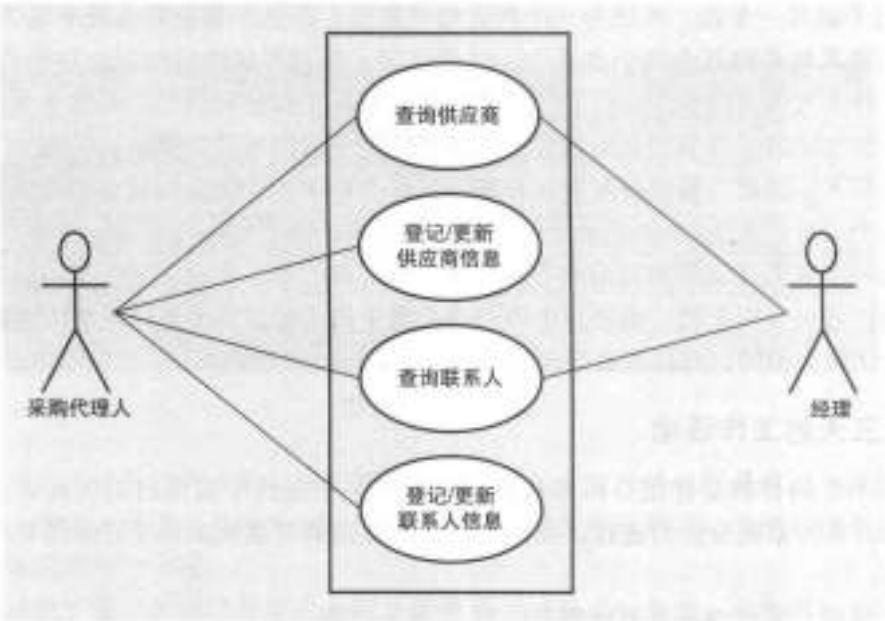


图 1-11 用例图

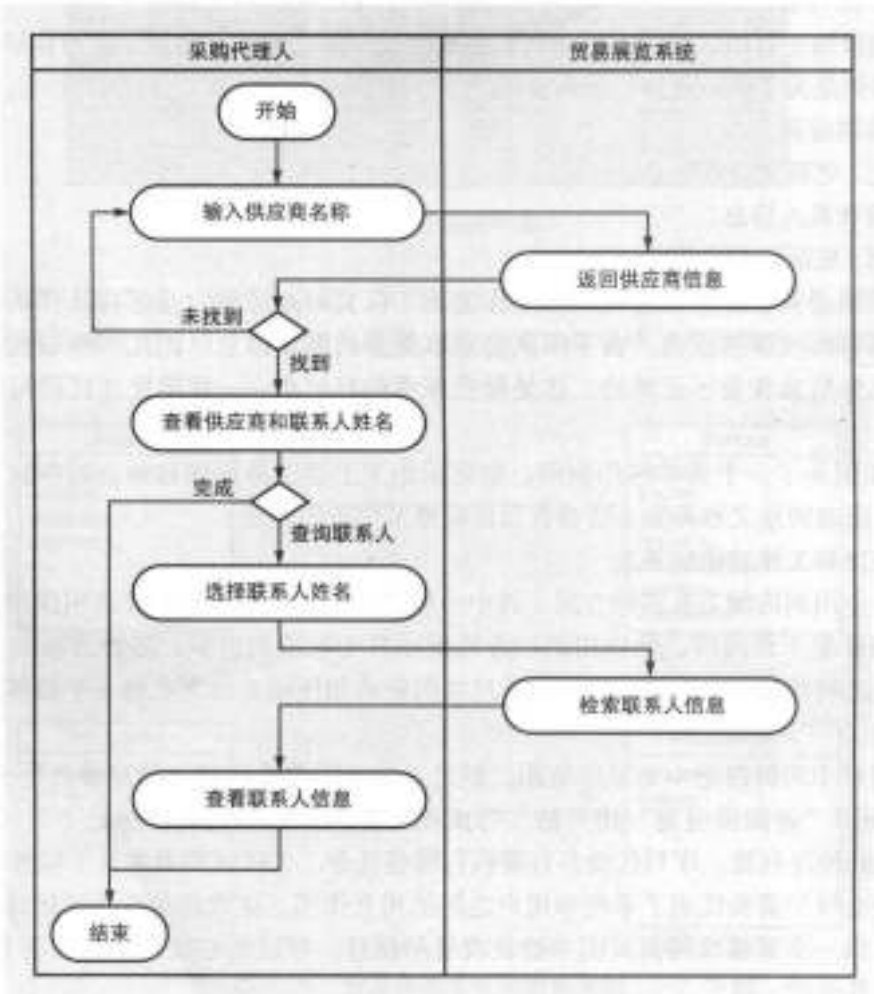


图 1-12 查询供应商用例的工作流图

根据图 1-12，我们看到顶部箭头表明输入该系统的供应商名称。因此，我们推断用户必须有一个在线表单用于输入供应商名称以进行最初的查找。另一个箭头表示必须有一个为供应商展示所有细节的形式，包括存在的联系人列表。同时用户也希望看到更多特定的细节，所以用户会要求关于特定的某个人的详细信息。因为用户要选择显示结果，因此我们必须设计表单，列表上的每个条目要么是一个链接，要么是一些选项。

定义屏幕布局

用户界面设计的任务是描述对于用户而言系统的外观和感受。因为用户界面是用户利用系统功能来完成工作的窗口，所以用户界面实际上就是系统。如果界面设计不佳，用户将无法充分利用系统，甚至觉得系统很差。另一方面，好的用户界面直观、易于使用、具有全方位的特征且便于导航，它能提供良好的信息并极大地提高系统的效用。

图 1-13 阐明了第一个屏幕的布局，用于“查询供应商”用例的工作流。屏幕的顶部为用户提供了输入供应商信息的位置，屏幕的底部显示结果。结果显示出来以后，搜索框仍将是可见的，且允许用户输入另一个搜索条目。结果中的每个条目将建立为一个链接，因此用户可以点击任何特定的供应商以获取更详细的信息。下钻技术在今天的系统中是常用的一种方法，使用户觉得更直观简单。

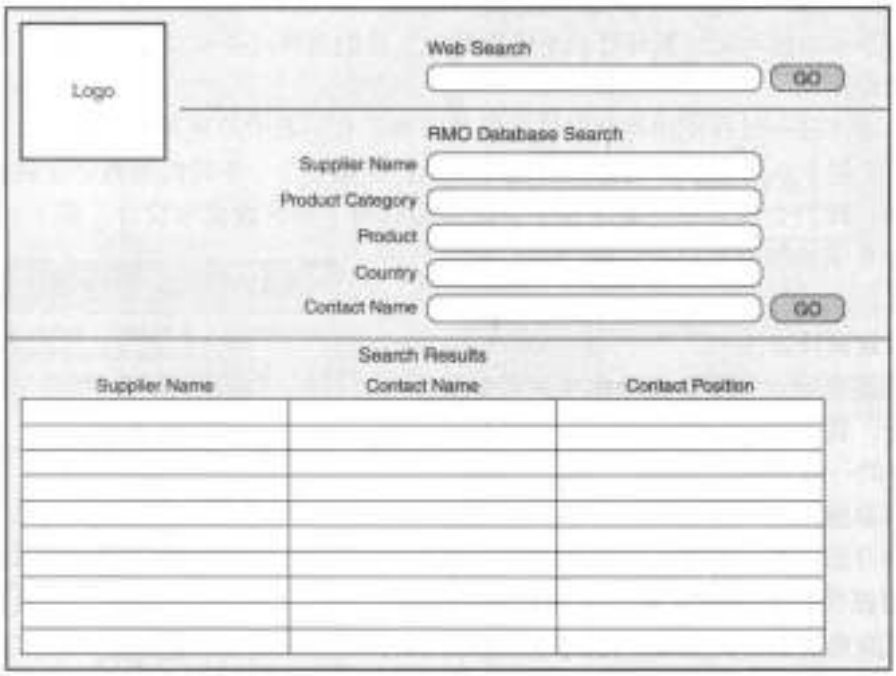


图 1-13　查询供应商用例的屏幕布局草图

搜索将在落基山运动用品数据库中进行，产生的结果信息包括姓名、地址和联系人信息。基于网络的搜索也是可能的。这允许采购代理人查询和查看供应商自己的网站，这是有益的，因为这样可以看到关于供应商的论坛和讨论。我们可以注意到这个离题的活动并没有被纳入工作流图 1-12，因为图 1-12 的目的是协助屏幕设计，而不是记录所有用户要做或将会做的事情。

1.5.5　第四天的工作活动

首要关注的第四天的活动是设计解决方案系统的各种组件。到目前为止，我们的大多数

工作是试图理解用户需求。在第四天，我们进行设计活动，这将直接关系到编程工作。在这个意义上，设计活动可以被认为是中间桥梁。在分析活动中，团队目标是理解用户需求。在编程活动中，目标是生产解决方案。因此，设计起了理解和构建间的桥梁作用。它提供了关于解决方案的构建大纲和编程大纲。系统设计往往涉及技术人员，也有少量用户参与。

设计可以是一个复杂的过程。在我们的小项目中，我们会使设计例子不仅仅局限于几个模型和技术。在本书中，你还将学习额外的设计技术。第四天的活动包括以下加点：

- 设计数据库结构（模式）。（核心过程 4）
- 设计系统的高阶结构。（核心过程 4）

数据库设计是一个相当简单的活动，使用类图作为输入并开发详细的数据库模式，可以直接实现数据库管理系统。表、关键字和索引标识以及属性类型等将在这一活动中确定。

设计高级系统结构和各个程序可以是一个复杂的过程。首先设计系统的总体结构，包括识别子系统及其与其他系统的连接。在每个子系统中，要制定关于各个程序的决策，比如用户界面程序、业务逻辑程序和数据库访问程序。之后，在最底层定义每个程序的登录部分，包括所需的程序函数和变量。

对于开发商将编写程序代码作为部分设计活动的开始，这一现象并不鲜见。在编写程序前完成大部分的结构设计是一个好方法。然而，当系统的较低层开始设计后，编码往往就开始了。但在落基山运动用品贸易展览系统项目中，我们会将这些活动单独罗列。

设计数据库

设计数据库这一过程使用类图提供的信息来确定表、表中的列和其他组件。有时，数据库设计将用于整个系统或子系统。在其他时候，它是用一个个零碎的用例建立起来的。为了使任务简单，我们会展示供应商信息子系统所需的两个类的数据库设计。图 1-14 展示了供应商信息子系统的数据库模式。两个表定义了供应商和联系人。

表名	属性
Supplier	SupplierID: integer {key} Name: string {index} Address1: string Address1: string City: string State-province: string Postal-code: string Country: string SupplierWebURL: string Comments: string
Contact	ContactID: integer {key} SupplierID: integer {foreign key} Name: string {index} Title: string WorkAddress1: string WorkAddress2: string WorkCity: string WorkState: string WorkPostal-code: string WorkCountry: string WorkPhone: string MobilePhone: string EmailAddress1: string EmailAddress2: string Comments: string

图 1-14　供应商信息子系统的数据库模式

高层系统设计方法

有一些基本设计原则会指导你来进行系统设计和程序设计，我们将在后面详细讲解这些原则。现在介绍设计的一般方法。

我们在系统设计中遇到的第一个问题是怎样开始以及从何开始。到目前为止，我们有三种类型的文档，它们提供的详细说明有助于回答这个问题。首先是用例以及用例相关文档，比如用例工作流图。其次是类图，类图会帮助我们识别系统所需的那些面向对象的类。（之前我们只是将类图作为数据库设计的基础，而这些类对于开发面向对象程序类也很重要。）最后，屏幕和报告提供了程序逻辑和显示逻辑的详细说明。

在投身设计工作之前，让我们简单讨论一下系统设计的目标以及期待的输出或结果。面向对象程序的结构被组织为一系列交互类，因此，为了编写程序，我们需要知道这些类是什么，还需要知道每个类的内在逻辑（比如函数）以及哪些类

之间存在交互。这就是系统设计的最终目标：定义类、类的内在方法以及类之间的交互。

我们从最高层开始执行设计过程，然后下钻至最低层，直到定义了每个类中的所有函数。细节设计是一个思考的过程，即考虑如何编程实现每个用例。后面我们将学习细节设计的相关技术。在第四天的工作活动中，我们只关注总体设计。

设计总体架构

图1-15展示了总体架构或是新系统的结构。尽管图中内容比较简单，但很多重要的决定在设计发展中已经被涵盖了。首先，我们决定将应用程序构建为一个基于浏览器的系统。另一个很受欢迎的方法是构建智能手机或平板电脑应用程序。浏览器系统能提供的连接速度和控制方式有时与智能手机或平板电脑应用程序不同，但它们用途更广泛，可以更容易地部署在不同的设备上，如笔记本电脑，而且不需要做任何修改。

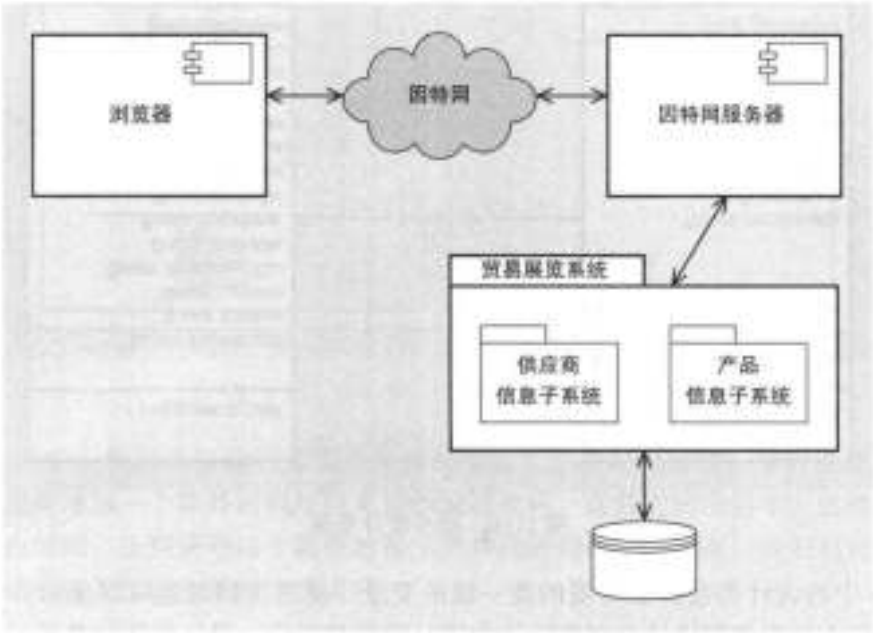


图1-15 贸易展览系统架构配置图

这些高层次的设计决策将决定系统的详细结构。一个基于浏览器的系统的结构和构建方式与运行在智能手机或平板电脑上的应用系统是不同的。

定义初步设计类图

鉴于贸易展览系统将使用面向对象的编程技术，因此一个重要的组件设计是开发一组系统所需要的对象类和功能。这一过程将十分详细，我们在此不进行解释，你将在本书后几章学到这些技术。

图1-16是一个贸易展览系统的初步设计类图。设计类图确定一个系统所必需的面向对象编程的类。这组设计类包含了问题域类、视图层类、独立的数据连接类以及工具类。在图1-16中，我们只展示了问题域类和视图层类。问题域类来自在分析过程中确定了的这些类，因此命名为问题（用户需求）域类。你会注意到这些与对应的数据库表很接近；事实上，在这一简单的项目中，这些类与数据库表大致相同。在更复杂的系统中，它们会相似但不会相同。然而，请记住程序类和数据库表是有区别的。

其他类需要图形用户界面（GUI）。在动态网络系统中，比如贸易展览系统，这些类从

浏览器接收输入，并以 HTML 文件格式在浏览器中显示。

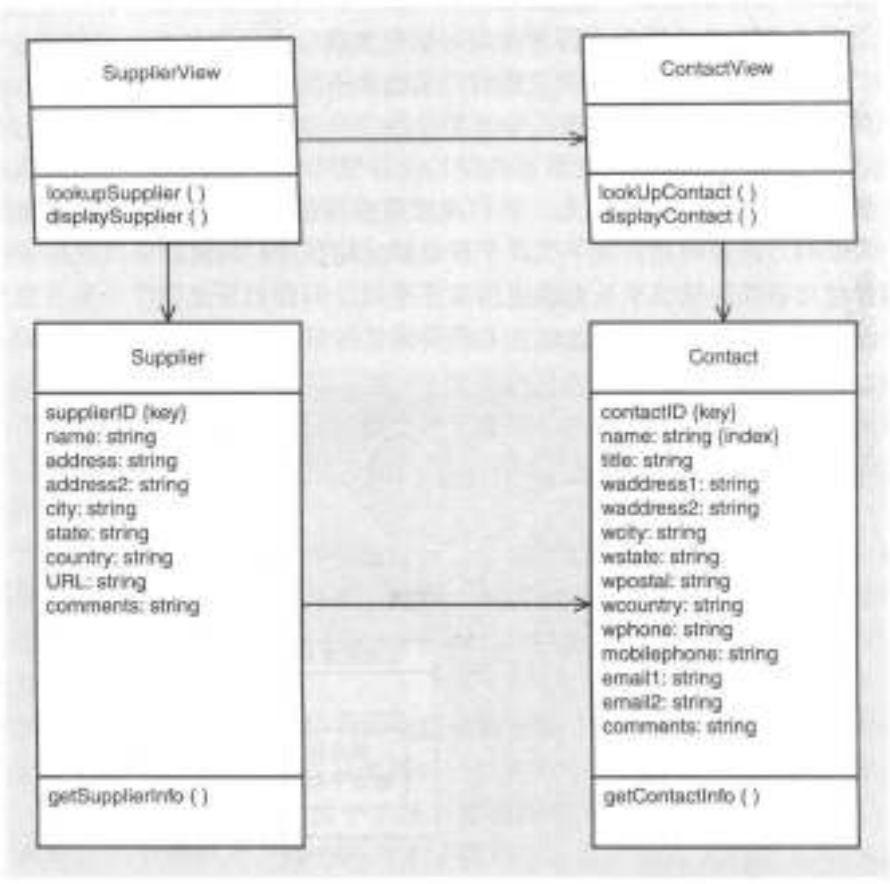


图 1-16 初步设计类图

图 1-16 中的设计类包括类所需的类一级的变量。这些类同时也可以显示一些重要方法的名称。这些方法在高层设计和详细设计中进行识别和指定。设计类图的最后一个元素是箭头，它表明其他类可以调用这个类的方法。

设计子系统架构

一旦有了总体结构和实现新系统的总体方法，我们就开始深入到子系统的设计中。图 1-17 阐明了供应商信息子系统的架构设计。注意这些子系统被深入划分成了两层：模型层和视图层。你将会在本书中学到更多关于多层次设计的知识。分层系统的优点之一是该系统的结构更容易建立和维护。举个例子，该系统将基于浏览器，但不同浏览器要求不同的技术。最好不要将这些复杂性与基本的程序功能混在一起。因此，它们被分在不同的层中。

图 1-17 表明视图层有两个 PHP 类，它们处理用户从浏览器输入的信息，并输出 HTML 格式文件。它还包含各种 JavaScript 函数，这将会在浏览器本身执行。模型层类将执行业务逻辑，并且会访问数据库。有时，数据层和业务逻辑层会被鲜明地区分开来。

管理项目

设计是需要许多阶段的复杂活动，从高层结构设计到低层细节程序设计。在我们的项目中，已将总体系统结构的设计与程序本身的详细设计相分离。然而，这些活动通常同时执行。基本的高层架构会先定义，但中层的以及低层的设计往往与编程同时进行。

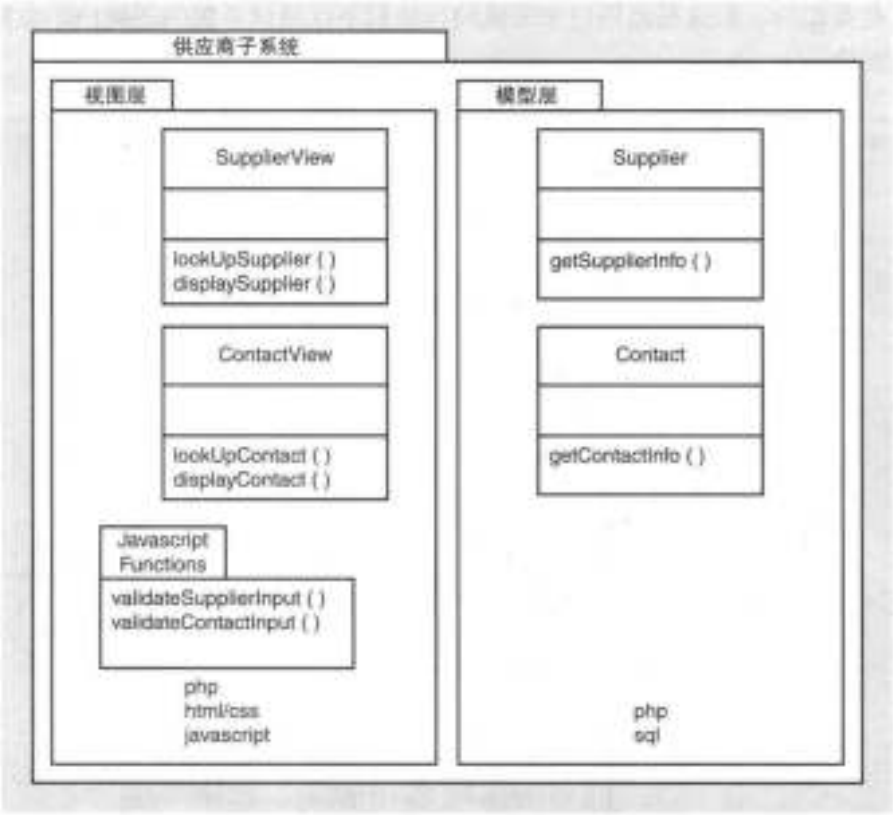


图 1-17　供应商子系统架构的设计图

在图 1-17 中，我们可以看到细节设计和编程确实是耗时的活动。项目经理必须决定是否扩展项目或是增加一个额外的程序员来协助编写代码。在我们的项目中，已经选择插入一个半天的自由时间，还要新增两个额外的程序员并且培训他们。当然，我们也可以直接开始第五天的活动来确保我们能及时完成项目。

1.5.6　第五天的工作活动

尽管详细的设计和编程往往在项目早期便已开始，但我们把它定义为独立一天的活动。有以下几个重要的原因。首先，我们要强调的是在没有获得关键信息并做出决策前就编程是错误的。新手程序员常常在未充分了解用户的需求或整体系统未被决定的情况下开始编程。但更好的方法是先去理解、设计，以及同时建立小块的系统。敏捷开发会为预期的变化做出准备和计划，并细化发生在详细设计和编程中的问题需求。

程序员编写代码的同时，他们也对所编的类和功能进行单独测试。本书并不着重于编程。然而，我们举了一些编程代码的例子，因此大家可以看到系统设计是如何与最终的程序代码联系的。图 1-18 是一个接收和处理供应商信息需求的类的样例。

1.5.7　第六天的工作活动

第六天活动的关键在于最后的测试，这被要求在系统开始部署前进行。需要进行许多类型的测试。在这个例子中，我们只提到两种类型的测试：整体系统功能测试和用户接受度测试。功能测试通常是所有用户功能的系统阶段测试，且常由质量保证团队完成。用户接受度

测试本质上是类似的，但这是由用户来完成的，他们不仅测试系统的正确性，也测试完成业务需求的完善性。

```
<?php
    class SupplierView
    {
        private Supplier $theSupplier;

        function __construct()
        {
            $this->theSupplier = new Supplier();
        }

        function lookupSupplier()
        {
          include('lookupSupplier.inc.html');
        }

        function displaySupplier()
        {
              include('displaySupplierTop.inc.html');
              extract($_REQUEST);    // get Form data
            //Call Supplier class to retrieve the data
            $results = $theSupplier->getSupplierInfo($supplier, $category,
                                                     $product, $country, $contact);

           foreach ($results as $resultItem){
           ?>
                    <tr>
                        <td style="border:1px solid black">
                              <?php echo $resultItem->supplierName?></td>
                        <td style="border:1px solid black">
                                <?php echo $resultItem->contactName?></td>
                        <td style="border:1px solid black">
                            <?php echo $resultItem->contactPosition?></td>
                    </tr>
            <?php }
            include('displaySupplierFoot.inc.html');
        }
    }
?>
```

图 1-18　SupplierView 类的代码

在第六天的活动中，每个测试对于需要执行的任务都有相似的序列。任务本身高度依赖于测试数据以及对于特定测试用例的方法。举个例子，测试可以是自动的，也可以手动进行测试。许多新的系统是伴随用户活动的交互式系统。有些测试工具包含自动化的过程，但它往往是一个相当复杂的任务。

图 1-19 是测试新系统的广义工作流图。在这个工作流图中，我们展示了不同阶段的不同测试任务。在现实中，它们往往被同时执行。然而，任何给出的测试例子都将会遵循这一流图。

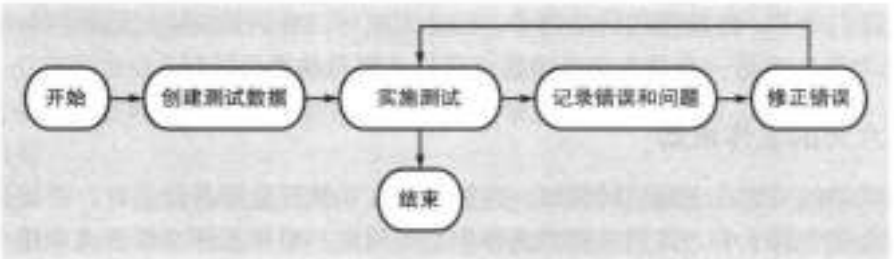


图 1-19　测试任务的广义工作流

1.5.8　第一次迭代回顾

图 1-20 是浏览器的页面截图，它用于在贸易展览系统中输入和查询供应商。

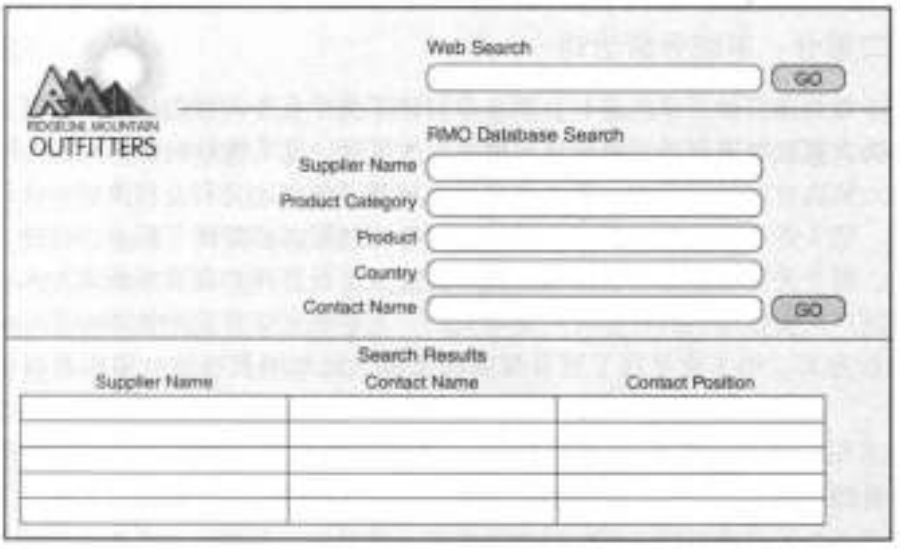


图 1-20　查询供应商用例的屏幕截图

如前所述，这是一个长期项目中的第一次（6 天）迭代。在整个项目中的使用敏捷技术和迭代使得定义并建立一个新系统变得更灵活。敏捷开发工作要求用户深入参与到新系统的开发中。在这六天的活动项目中，用户除了第四天和第五天都有深入的参与。

在开发新系统时的一个基本问题是，随着项目的展开，新的需求常常出现。这一现象是因为用户和项目团队学到了更多关于如何解决业务需求的知识。构建高敏捷、强交互的项目是处理这些新需求的方法，也就是常在整个项目中添加另一些迭代。

在当前迭代的最后一步，或许也是作为下一个迭代计划过程的一部分，都应该有一个证明当前迭代成功的审查过程。通过总结经验和引出问题，我们创造了一个不断改进和完善的环境。迭代项目往往会不断提高并在项目生命周期中变得更为有效。

1.6　后续内容导读

本书紧凑、合理且集中地展示对于信息系统开发人员所必不可少的主题。

1.6.1　第一部分：系统开发导论

第 1 章提供了关于典型软件开发项目的详细的例子。当然，为了使这一章保持一个合理的长度，我们忽略了许多细节。然而，这一章节还是包含了许多过程、技术以及图表。我们不要求你从这一简短的介绍中了解全部的元素。但是，对于开发系统必须有一个整体的概念。你可以时时回顾这一章来帮助自己了解整体架构。

第 1 章从简单地解释系统分析和系统设计的对象开始。许多上过编程课的学生认为开发一个软件和配置一个系统仅需要编程。这一章及本书的余下部分都在辩驳这一设想。

本书讨论了三个主要的课程领域：系统分析，系统设计，以及项目管理。同时也有一个关于系统实施、测试和调度的小主题，同样重要但并未深入讨论。此外，我们采取了和其他教材不同的方法。因为大家通过第 1 章已经对系统分析与设计有了基本的了解，所以我们能

直接呈现一些关于系统分析与设计的深入概念。在后面的章节中，我们会呈现项目管理的话题。这会使你在理解系统分析与设计元素后学习到一些项目管理概念。我们认为这会给你带来更多意义。

1.6.2 第二部分：系统分析活动

第 2 – 5 章详细讲解了系统分析。第 2 章讨论了关于业务问题的信息收集技术。开发正确的系统解决方案仅在问题被正确理解的情况下才可行。受系统影响的不同人（利益相关者）也同时被纳入解决方案的开发中。第 2 章同时也解释了如何定义利益相关者并介绍了模型和建模的概念。第 3 章和第 4 章呈现了在系统中以有用的形式捕捉细节需求的方法。在讨论到信息系统时，两个关键概念尤为有用：用例，它定义了最终用户需要系统做什么；数据实体或类，这是用户在实施他们的任务时所要用到的。无论在系统开发中使用的是何种方法，这两个概念都很重要。第 5 章呈现了更有深度的模型，比如用例描述、用例图表以及系统顺序图。

建模技术用于对用户需求进行深度分析，并允许分析员发展需求和规范。需要强调的是，系统分析的目的是彻底理解且详细说明用户的需要和需求。

1.6.3 第三部分：系统设计的要点

第 6 章和第 7 章介绍了系统设计的基本理念，并且定义和设计了用户经历。第 6 章提供了广泛且深入的系统设计重要原则。它不仅可作为一个广泛的设计原则的总览，也为后面章节详细介绍技术、任务和模型（实现设计）奠定了基础。

第 7 章介绍了用户界面和系统界面相关的设计原理。设计用户界面是分析与设计过程的结合。这与分析相关，因为它要求用户高度参与，而且要详细描述用户的活动和需求。另一方面，这是一个设计活动，因为它创造了驱动编程活动的特定最终组件。显示屏、报告以及其他用户互动组件必须被精确设计，这样才能作为最终系统的一部分而开始编程活动。系统交互是指一个信息系统和另一个信息系统在没有人工干预的情况下进行交流或交互。系统交互因为网络服务和云计算而变得越来越重要。

1.6.4 第四部分：项目和项目管理

通过这部分的学习，你将会对系统开发的所有元素有基本的了解。

第四部分将通过解释组织和管理项目的开发过程将所有理论集合在一起。第 8 章描述了现今环境下系统开发的不同方法，包括几个重要的系统开发生命周期模型。这是一个帮助你了解项目如何执行的重要章节。

第 9 章通过讲授项目管理的基础原理来延伸这些概念。每一个系统分析员都要组织、协调以及管理软件开发项目。此外，大家几乎都会成为团队领导及项目经理。第 9 章中呈现的原理对于想要事业成功的人是很有必要的。

1.6.5 第五部分：高级设计和部署概念

第五部分更深入地揭示了系统设计、数据库设计，以及其他关于有效且成功的系统开发和部署的重要问题。

第 10 章和第 11 章详细介绍了设计软件系统的模型、技巧及技术。在本章已经说过，系

统设计是一个复杂的活动，尤其是让它正确运行。这两个章节的目标是教会大家多样的技术——从简单到复杂，从而能有效地设计软件系统。

第 12 章描述了系统开发的最终要素：最终测试、部署、维护以及版本控制。

本章小结

本章提供了一个特定软件系统开发项目的快速概览，称为贸易展览系统。呈现了 6 个控制软件开发的核心过程。然后，我们在完成贸易展览系统的整个过程中通过各种活动支持了这 6 个核心过程的执行。6 个核心过程如下：

1. 确定问题或需求，并获得批准以向前推进。
2. 计划和监控项目——做什么、怎么做以及谁来做。
3. 发现和理解问题或者需求的细节。
4. 设计能解决问题或者满足需求的系统组件。
5. 建立、测试和整合系统组件。
6. 完成系统测试并部署解决方案。

为了促进学习并帮助记忆核心阶段及相关活动，我们将项目分成项目预备活动及其他 6 个分组，称其为*项目天*。需要注意的是，并没有奇特的或强制性的关于这个项目的组织方法。本书采用的这种方式只是简单地帮助你了解各种相关活动的核心过程。

复习题

1. 信息系统和计算机应用有什么不同？
2. 系统分析的目的是什么？为什么它很重要？
3. 系统分析和系统设计有何区别？
4. 什么是项目？
5. 软件系统开发的 6 个核心过程是什么？
6. 敏捷开发的意义？
7. 系统可视化文档的目的是什么？
8. 系统和子系统的区别是什么？
9. 工作分解结构的目的是什么？
10. 工作分解结构的组件有什么？表明了什么？
11. 用例或用例图表提供了什么信息？
12. 类图提供了什么信息？
13. 用例图和类图如何驱动系统开发过程？
14. 描述活动图的另一种方法是什么？它显示了什么？
15. 活动图是如何协助用户界面设计的？
16. 架构设计的目的是什么？
17. 设计类图提供了什么新的信息？（相比类图而言）
18. 系统测试有哪些阶段？
19. 用户接受度测试有什么目的？
20. 将项目分成单独迭代的优势是什么？
21. 迭代的目标和结果是什么？

第二部分

Introduction to Systems Analysis and Design: An Agile, Iterative Approach, Sixth Edition

系统分析活动

第 2 章　系统需求调查
第 3 章　用例
第 4 章　域建模
第 5 章　需求模型的延伸

系统需求调查

学习目标

阅读本章后，你应该具备的能力：

- 描述系统分析活动。
- 阐述功能和非功能系统需求的区别。
- 描述模型在系统分析中扮演的角色。
- 识别和了解不同类型的利益相关者以及他们对需求定义的贡献。
- 描述信息收集技术，决定每个技术的最佳应用场合。
- 开发活动图来构建工作流。

开篇案例　山地摩托运动

Amanda Lamy 是山地摩托运动公司（MVM）的总裁和股东，同时还是一个摩托车爱好者和商人。MVM 总部设在丹佛，它的分公司遍布在美国西部和加拿大。从 20 世纪 90 年代末开始，摩托车市场发展得非常迅速。Amanda 希望市场能在 21 世纪继续增长，尽管她同时也在担忧顾客占有率正在逐渐减少。

摩托车市场的人口统计学相对来说是一项有趣的研究。现在，大多数的消费者是超过 50 岁的男性专业人士或商业人士，而且部分或全部都退休了。他们拥有大量的可支配收入以及大量的空闲时间，倾向于从类似 Harley-Davidson、Honda、Ducati 及 Moto Guzi 这样的制造商处购买属于自己的昂贵摩托车。老年顾客们通常都会喜欢上网，但他们又不是社交媒体技术的主要用户。尽管他们中的许多人拥有智能手机，但他们还是倾向于使用录音、发邮件、发短信这样的基本功能。

30 岁以下的男性消费者倾向于购买运动型越野车，特别是喜欢选择 Suzuki 和 Kawasaki 这样的制造商。相对于年纪较大的顾客来说，他们会购买比较便宜的摩托车，而且更想从山地摩托车公司购买部件来组装他们自己的摩托车。30 岁以下的女性消费者倾向于购买小型摩托车和小型通勤摩托车。30 ~ 50 岁之间的消费者无论男女都没有特定的消费倾向。因特网技术、社交媒体、便携式计算设备（如智能手机和 iPad）在 50 岁以下的消费者中是很流行的，尤其是 30 岁以下的消费者。

Amanda 确信要想在摩托车市场取得长期成功，关键是在每个 MVM 分公司建立一个摩托车爱好者论坛，这个论坛能吸引不同类型的消费者。实际上，每个分公司在线上和线下方面都必须成为当地摩托车相关活动的信息中心。在线下方面，MVM 已经在自己的网站上增加了相关活动和新闻，在一些地方，他们所赞助的公路汽车赛和俱乐部设置了会客厅及小型咖啡屋，此外还有酒吧和餐厅，在这里顾客会感受到与摩托车相关的主题和氛围。

Amanda 关心的是年轻用户缺乏参与，她同时也体会到 MVM 在社交媒体方面和虚拟社区方面还很薄弱是一个不争的事实。她和大多数资深的高层员工还不知道怎样吸引年轻消费

者。他们缺乏这方面的知识和经验来创造出以现代技术为基础的网络虚拟社区。

MVM的信息主管正在着手开发一个适用于年轻消费者的虚拟社区项目计划。如果这个计划是用来开发一个传统的信息系统，那么她可能会使用一些标准的方法，如与内部用户与经理进行访谈、开发人员撰写说明书、形成脚本、布局界面、开发原型。除了极少数MVM的员工是虚拟社区用户，其他员工基本不能完全理解如何熟练地运用社交媒体和其他技术来构建虚拟社区。用传统方法来定义和修改需求看上去不是很适合这个项目。

2.1 引言

在第1章中，我们已经看到系统开发生命周期被落基山运动用品公司运用在一个称为贸易展览系统的小型信息系统中。那个系统的开发遵循了系统开发生命周期的6个核心过程：

1. 确定问题或需求，并获得批准以向前推进。
2. 计划和监控项目——做什么、怎么做以及谁来做。
3. 发现和理解问题或者需求的细节。
4. 设计能解决问题或者满足需求的系统组件。
5. 建立、测试和整合系统组件。
6. 完成系统测试并部署解决方案。

在本章，我们会开始扩大系统开发生命周期的范围和细节，以涵盖更宽泛的概念、工具和技术。更深层次的需求和细节会用来处理更大、更复杂的项目。这章的内容会集中在系统分析活动上（核心过程3），后面几章会继续跟进并详细讨论在系统分析活动中开发的模型。随后的章节会扩展对其他系统开发生命周期的核心过程的讨论。

2.2 RMO综合销售和市场营销系统项目

落基山运动用品（RMO）拥有一系列复杂的信息系统应用，这些应用的开发工作历经数年，主要用于支持公司事务的操作和管理。然而，在消费者期望、现代化技术能力和现有的RMO系统（包括销售支持和消费者互动）之间存在着不断增长的分歧。本节将回顾现有系统的创造过程，并介绍计划推出的综合销售与市场营销系统，这个系统将会更新和强化销售和市场营销工作。

2.2.1 现有的RMO信息系统与架构

RMO的信息系统部门是很有远见的。在过去的岁月里，这个部门配合着公司的战略计划已经为新技术和信息系统的发展和部署制定了一个五年计划。计划进程已经成为帮助组织一直保持拥有当前最新的趋势和技术能力的重要工具。然而，计划本身有着失败的风险。长期的IT和软件开发项目的一大复杂性是技术变化很快，而且不是朝期望的方向发展。例如，通过有效且灵活的手持设备和无处不在的无线网，使得在第1章中描述的贸易展览系统具备了可行性。仅仅几年时间，技术就达到了一个相当成熟的水平，这就为RMO创造了一个机会以优化这个重要的商业进程。

从RMO的发展历程来看，RMO的高成本效益得益于采用了新技术。过去的例子包括小型服务器和桌面计算机，专用的通信链路形成了一个内部局域网，这样基于网络的技术（例如面向顾客的网站和基于浏览器的用户界面）便可为内部系统所用。现在，RMO拥有一个

完全不同计算机的集合，横跨总公司、零售商、手机中心，订单/航运中心和仓库——所有这些都能通过局域网、广域网和虚拟私人网络连接起来。**技术架构**描述了计算机硬件、网络硬件和拓扑结构以及组织机构所使用的系统软件（比如操作系统和数据库管理系统）的集合。RMO 的技术架构是现代化的但不具有艺术感，这个架构仅仅是实现了高性价比技术的目标。

应用架构描述了软件资源是如何组织和构建设的，从而实现一个组织机构的信息系统。它描述了软件组织的模块和子系统，还包括支持技术（如编程语言与开发环境）、结构化应用（如面向服务体系结构）以及用户界面技术（如移动计算显示、触屏技术和声音识别）。

现今，主要的 RMO 系统的组成部分是：

- 供应链管理（SCM）——这个应用是 5 年前作为运用 Java 和 Oracle 技术的客户/服务应用而配置的。现在它被用来支持库存管理、采购和分配，不过功能的一体化还需要提高。新的贸易展览系统会和这个系统接口。
- 电话/邮件订购系统——12 年前通过运用 Visual Studio 和 Microsoft SQL Server 开发的一个规模适中的客户/服务应用，用来解决消费者对电话和邮购目录的需求。现在它被整合到了供应链管理中并且已经达到了饱和状态。
- 零售商店系统（RSS）——零售店被包装成一个销货店来处理。这个系统 8 年前有过升级，从夜间分批处理升级为实时库存信息更新。
- 顾客支持系统（CSS）——这个系统的最初配置是在 15 年前，作为一个以网站技术为基础的目录来支持顾客邮件和电话订单。四年后，它被更新为一个因特网商店，可支持顾客调查、购物车、订单追踪、运输、回馈订单和返回。

所有组织（包括 RMO）正面临一个艰难的挑战，那就是保持所有信息系统的先进性和有效性。因为开发资源是有限的，所以一个组织的技术、应用架构和信息系统将面临新旧系统并用的情况。陈旧的系统通常是使用过时的操作工具设计出来的，这种系统往往缺乏现代化技术和特征，一些采用这些系统的竞争者会试图提高它的有效性和竞争性。就像 RMO 现有的面向顾客系统这个例子一样，它有以下几点不足：

- 将电话、网站和零售店作为分开的系统而不是一个整体。
- 使用过时的以网站技术为基础的店面技术。
- 不支持现代化技术和顾客互动模块，包括移动计算机设备和社交网络。

RMO 计划重新开发一个系统而不是逐步更新这个现有的销售系统，如图 2-1 所示。

2.2.2 新综合销售和市场营销系统

综合销售和市场营销系统（CSMS）的目标是使顾客支持系统（CSS）的技术和功能更现代化，并且增加了更多以顾客为中心的功能。在技术方面，综合销售和市场营销系统将会吸收当前的网站标准，而且会以高宽带的顾客网络连接和高分辨率显示器为前提。更新技术后能产生高度的互动、丰富的图形和简化的界面。

系统功能的关键补充包括支持移动计算机设备、纳入顾客反馈意见和产品信息评论以及整合社交网络功能。不像顾客支持系统，综合销售和市场营销系统会以内部特别为每个平台设计且可下载的 App 来支持智能手机和平板电脑。通过 RMO 支持的论坛和博客以及 Facebook 和推特，顾客的回馈将会通过网上商店直接捕获。RMO 会在每个社交网络上开发一个完整的论坛，系统用户能够在论坛上分享买到的产品、建议、优惠券和会员卡。

新销售和市场营销系统有以下四个子系统：

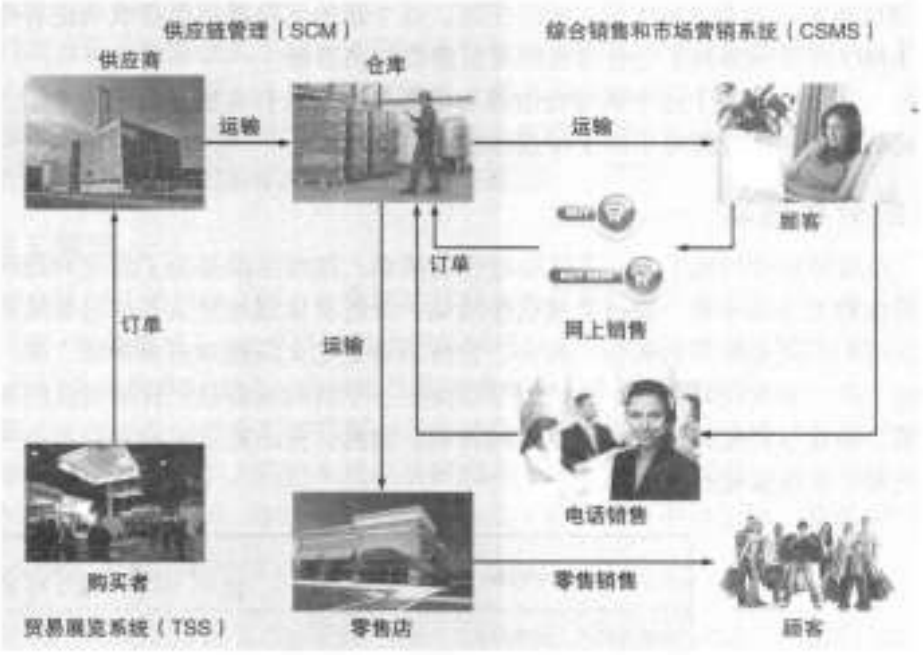


图 2-1　计划的 RMO 应用架构

- 销售子系统提供基础功能，如浏览网上目录、购买项目以及网上付款。然而，它有许多新的功能来协助购买者购物。这个系统会提供关于产品配件的具体建议，也会提供动画模拟的图片和视频来帮助消费者看到商品的多样性以及商品的配置。系统还会提供其他购买者需要的相关商品的信息给正在购买的顾客。排名和评价都将展示出来。最后，关键的社交网络组件允许购买者通过发信息和他们的朋友联网交流，询问他们对特定商品的意见。
- 订单实施子系统会执行所有运送物品的正常任务，并且允许消费者跟踪他们的订单及运输情况。除此以外，作为订单实施的一部分，消费者能评价特定商品和他们的整个购物经历。他们也能直接给 RMO 提出关于网站吸引程度和服务质量的意见。
- 顾客账户子系统提供能提升顾客购买体验的所有服务。消费者能看到和维护他们的账户信息。他们也能添加好友，这个好友可以是对同一件商品分享经历和意见的顾客。这个系统会跟踪详细的运送地址以及购物偏好信息。RMO 同时也建立了一个叫作“山地雄鹿”的程序，其中顾客可以累积积分，这些积分可以用来抵扣现金和购买商品。消费者可以自己使用积分，也可以把积分换给他们的家庭 / 朋友和其他人。这是个很好的机会，可以通过累积积分来获取昂贵的产品。
- 市场营销子系统主要是为 RMO 员工建立消费者的信息和服务。在这个子系统中，员工可以获取所有 RMO 提供的商品信息。这个子系统也需要供应链管理系统的支持来维护库存的精确数据和订单中商品的预期到达的日期。员工也能运用这个子系统的功能建立各种各样的促销套餐和季节性商品目录。RMO 正在测试一个新构思来提高消费者的满意度：和其他零售商建立搭档关系以便消费者能用 RMO 的产品或者搭档一个零售商产品来赚取双倍积分。这些积分可以用在 RMO 或者转移到和 RMO 搭档的零售店铺中。例如，因为 RMO 销售户外运动服装，所以它就可以和各种各样的运动商品店合作。通过这样的方式，消费者可以在运动装备店中购买运动装备，并且

获得的折扣可以在 RMO 买衣服时使用。这个新的风险项目能否成功还有待证明，但是 RMO 已经预感到了它会增加所有消费者的消费额。

在以后的章节中，关于这个新综合销售和市场营销系统会有更多的细节介绍。详细解释中也会描述必须包含在数据库中以支持这些功能的信息。

2.3 系统分析活动

图 2-2 左边的标注列出了第三个核心过程的活动，这些活动是为了发现和理解细节。这个核心过程也称为系统分析。通过完成这些活动，分析员详细地定义了信息系统需要完成什么才能提供组织所期望取得的利益。其实，分析活动是定义问题和需求的第二步，但却是更全面的过程。第一步仅仅是得到足够的细节以决定一个新的或者是正在被更新的系统是否能被授权。第二步是取得组织认可这个项目的权利。因此，付出更多时间和资源是为了能提供更加详细的对于系统能做什么的描述。

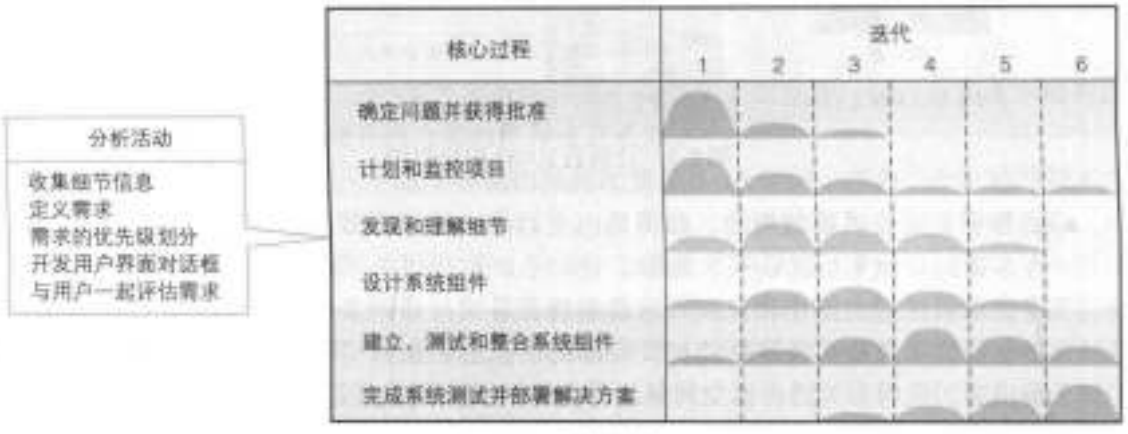


图 2-2 分析活动

尽管本章只将注意力放在分析活动上，但是要注意在系统开发生命周期中，分析通常是和设计、实施以及其他活动混合在一起的。例如，根据图 2-2，在所有项目的迭代中除了最后的迭代过程，分析活动在第二个次迭代中是最集中的，但是它们发生的程度是不一致的。这个模式通常是因为分析员会集中在系统的一部分，仅仅是为那个部分定义需求，同时设计和整合软件来满足这些需求。在这个迭代中的组织开发活动使开发过程分解成几个小阶段，并且使用户通过测试和观察一个功能系统来确认需求。下面几个部分清晰地描述了分析活动，而这章剩余的部分将扩充信息收集和定义系统需求的讨论。

2.3.1 收集细节信息

系统分析员从将会使用这个系统的人那里获取信息，要么是与他们访谈，要么是观察他们的工作。他们获取其他信息的途径是回顾计划文件和政策声明。分析员也会研究现有的系统，包括它们的参考资料。他们也会经常观察其他公司（特别是供应商）面临相似商业需求时的做法，从而获取额外的信息。通过识别和理解所有现在和将来的用户活动、工作发生的地点以及系统与其他系统（内部或外部）之间的借口，他们会试着去理解一个现有的这个系统。简而言之，分析员需要和几乎所有将会使用这个新系统或者已经开始使用相似系统的人进行交流，并且他们必须阅读几乎所有关于现有系统的可用信息。本章后面会讨论怎么样从

所有这些人中识别和获得信息。

刚入行的分析员经常会低估这份工作的工作量。分析员必须是该系统所在业务领域的专家。例如，如果你正在实现一个订单输入系统，那么你就必须在订单处理方式上（包括相关的会计程序）成为专家。如果你为银行工作，那么你就必须把自己看作是一个银行职员。最成功的分析员会成为他们的组织中主要业务的专家。

2.3.2　定义需求

分析员会使用从用户和文件中收集到的信息来为新系统定义需求。系统需求包括系统必须执行的功能（功能需求）、用户界面格式相关问题以及可靠性、性能和安全性的需求（非功能需求）。我们会在本章后面进一步讨论功能需求和非功能需求的差别。

定义需求不仅仅是记下事实和数据。分析员会创建模型来记录需求，与用户和其他人一起检查模型，并且改进和扩大模型来反映新的或者是需要更新的信息。建立和改进需求模型会占用分析员大部分的时间。本章和后面两章会在一定程度上解释怎样创建需求模型。

2.3.3　需求的优先级划分

一旦我们已经充分理解了系统需求，那么找出对系统最关键的需求就显得十分重要。有时，用户会建议增加额外的系统功能，这个功能是他的个人期望但不是必需的。然而，用户和分析员需要问问自己哪个功能是真正重要的，而哪个功能是重要的但又不是一定需要的。此外，能理解组织结构和用户工作的分析员将会有更强的洞察力来回答这些问题。

为什么要为用户需求的功能划分优先级？因为资源总是有限的，而且分析员必须时刻准备去证明系统的范围是合理的。因此，明确需求是很重要的。系统需求倾向于随着用户提出更多建议而不断扩充（这个现象称为*需求蔓延*），除非分析员已经仔细评估过优先级。需求优先级也能帮助我们定义项目迭代的数量、构成和顺序。优先级高的需求经常包含在项目早期的迭代中，因此分析员和用户有足够的机会去改进系统的那些部分。此外，一个有着较多高优先级需求的项目会有较多迭代。

2.3.4　开发用户界面对话框

当一个新系统取代一个做着相似工作的旧系统时，用户通常会很确定他们的需求和用户界面的形式。但是许多系统开发项目会通过使用自动化功能来代替以前的手工执行功能，或者通过整合那些没有被执行过的功能来开辟新的领域。不论是何种情况，用户往往会不确定系统需求中的大部分内容。像需求模型这样的用例、活动图和交互图可以在用户输入的基础上开发，但是对用户来说去解释和验证这样抽象的模型是很困难的。

相比之下，一个界面的用户验证工作是很简单且很可靠的，因为用户能看到和感知系统。对大多数用户来说，用户界面可以影响一切。因此，开发用户界面对话框是一个有力的工具，它可以引出和记录需求。分析员可以通过抽象模型来开发用户界面，例如故事板（第 7 章将详细介绍），或者可以在用户使用的实际输入 / 输出设备（例如计算机显示器、iPad 或者手机）上开发用户界面原型。原型界面可以在开发系统的一部分时作为需求和开端。换句话说，一个在早期迭代中开发的用户界面原型会在以后的迭代中被扩大成一个完整的系统功能部分。

2.3.5　与用户一起评估需求

理论上来说，与用户一起评估需求和需求文档化是同时进行的。但是在实践中，用户

除了开发一个新系统外还有其他的工作。因此，分析员的迭代过程通常这样进行：观察用户输入，独自建模需求后去用户那里确认并记录需要补充的地方，然后再次独自工作，融入用户的这些意见并精化模型。当“纸上”模型可能不合适，或者用户和分析员需要确认被选中的技术可能对系统产生的影响时，用户界面原型和系统的其他部分也同样需要开发出来。此外，如果系统将要引入新的需求或者进行技术更新，那么用户的参与会有助于在需求识别过程中形象化这些新技术的可用性。原型可以包含所有的需求。获取需求、建立模型和原型以及与用户一起评估的过程可能要重复很多遍，直到需求模型和原型完整且精确。

2.4　什么是需求

从前面的内容中可以了解到，需求和模型是分析活动中的关键。大多数分析员的时间贡献给了需求：收集关于需求的信息，用模型和原型确定、改进和扩充需求、划分需求的优先级以及产生和评估候选方案。但是为了完全理解这些活动，你必须回答一个基础问题：什么是需求？

系统需求是这个系统必须执行或者支持的所有活动和必须满足的约束条件。分析员通常会将系统需求分为两类：功能需求和非功能需求。**功能需求**是系统必须执行的活动（如系统将要投入的商业应用）。例如，如果你正在开发一个工资系统，那么需要的业务也许包括这样一些功能：电子支付转移、计算佣金数量、计算工资税、维护雇员的相关信息和向 IRS 报告税费减免。这些就是新系统应有的功能。确定和描述所有这些业务应用需要花费大量的时间和精力，因为功能列表及其关系非常复杂。

功能需求是根据公司进行业务交易的过程和业务规则确定的。有时，这些过程会详细记录在文档中从而易于确定和描述。下面是一个例子：所有的新雇员必须填写一张 W-4 表格以输入他们在工资系统中的预扣税信息。其他一些业务规则也许有点隐蔽而难以发现。以 RMO 为例：超过 200 美元而且重量少于 2 磅的订单的空运费用会降低 50%。发现类似这样的规则对于系统最终的设计是非常关键的。如果开始没有发现这些规则，那么过去使用这个系统的顾客可能会生气。在顾客开始抱怨后才修改系统会比一开始就建立规则更加困难。

非功能需求是这个系统的固有特征，它不同于系统必须执行或支持的活动。区别功能需求和非功能需求是非常不容易的。用架构来识别和分类需求就是一种方法。随着时间的推移，人们开发了很多这样的架构，现今最为广泛应用的方法称为 FURPS+（见图 2-3）。FURPS 是功能、可用性、可靠性、执行和安全的首字母缩略词的组合。FURPS 中的 F 相当于之前定义的功能需求。FURPS 的其余部分描述的是非功能需求。

需求	FURPS+	举例
功能需求	功能	业务规则和流程
非功能需求	可用性	用户界面、易用性
	可靠性	失误率、回收法
	性能	响应时间、生产力
	安全	访问控制、加密
	+ 设计约束条件	硬件和支持的软件
	实施	开发工具、协议
	接口	数据交换格式
	物理	尺寸、重量、能量消耗
	可支持性	安装和更新

图 2-3　FURPS+

- **可用性需求**描述了与用户相关的操作特征，比如用户界面、工作流程、在线帮助及文档。例如，当用户使用双指滑动或开合这样的手势时，智能手机 App 用户界面应该做出和其他 App 相似的反应。附加需求可能会包括菜单格式、色彩设计、组织标识的使用和双语支持。
- **可靠性需求**描述了系统的可靠性。比如系统出现服务损耗、不正当处理以及错误检测和恢复。
- **性能需求**描述了与工作方法相关的操作特征，比如生产能力和响应时间。例如，系统客户可能要求所有按钮按下的响应时间都是半秒，而服务器可能需要在同样的响应时间内支持 100 个并发的客户会话。
- **安全需求**描述了怎样进入被控制的应用，以及在存储和传送时数据是怎样被保护的。例如，应用也许是被密码所保护的，采用 1024 位本地存储的数据加密，以及在客户和服务器节点中使用 HTTP 来交流。

FURPS+ 是 FURPS 的衍生品，增加了额外的部分，包括设计约束条件以及实施、接口、物理和可支持性需求，所有这些增加的部分都用加号汇总。下面是每个部分的简短介绍。

- **设计约束条件**描述了硬件和软件必须遵循的约束。例如，一个手机应用可能要求使用安卓操作系统，当运行和操作的 CPU 频率在 1GHz 或更高时，会消耗至多 30MB 的闪存存储和至多 10MB 的系统内存。
- **实施需求**描述了约束条件，比如要求的编程语言和工具、文档分析法和层次细节，以及与不同组件之间的一份特定的通信协议。
- **接口需求**描述了系统之间的接口。比如，美国一家上市公司的财政报告系统必须产生出一份 XML 格式的数据，而这个格式是证券交易委员会可接收的。这个系统可能也直接提供数据给证券交易所和债券评级机构，或自动产生推特消息、RSS 反馈信息和 Facebook 更新。
- **物理需求**描述了硬件尺寸、重量、能量消耗和操作环境等特征。例如，一个支持战场通信的系统必须有以下要求：重量不超过 200g、不大于 $5cm^3$ 以及完全充电状态下能持续使用 48h 的 1200mW 锂离子电池。
- **可支持性需求**描述了一个系统是怎样安装、配置、监督和更新的。例如，一个安装在家用计算机上的游戏的需求可能会包括利用现有的硬件、错误报告和从支持服务器上下载的更新包进行自动配置以获取最佳性能。

由于需求内容多变，因此 FURPS+ 有一些灰色地带以及一些内容重叠的部分。例如，一个战场通信设备要既能浸泡在水里又能在 –20 ~ 50℃温度范围内操作，这是一个性能需求还是物理需求呢？不能使用超过 100MB 内存的约束是性能需求还是设计需求？要在工作站和服务器之间使用 1024 位加密的安全通信是性能设计需求还是实施需求？这些问题的答案并不重要。重要的是所有的需求在开发过程中必须尽早地被准确定义，并解决需求的不一致性。

2.5 模型和建模

建模是分析和设计的重要组成部分。分析员建立模型来描述系统需求并且使用这些模型来与用户、设计者进行交流。通过开发一个模型然后与用户一起进行评审，分析员便能理解用户的需求。如果用户发现错误或遗漏，那么在随后的设计和实施活动的中这些问题就不会

包含在模型中。图 2-4 总结了建立和使用模型的关键原因。

设计者建立高层次且详细的模型来描述系统组成部分和迭代。设计模型充当了评价备选设计方案的草稿纸，同时也是程序员、商家、其他利益相关者以及所有参与最终系统创建的人之间对于最终设计的一个交流方式。

- 从建模过程中学习
- 通过抽象降低复杂性
- 记住所有细节
- 与开发团队中的其他成员交流
- 与各种用户、股东交流
- 记录需为未来的维护和加强所做的事

图 2-4　建模的原因

模型是系统建立过程中的一些部分的代表。分析员或者设计师开发和使用的模型有很多种类（如图 2-5）。尽管本书强调的是模型和创建模型的技术，但很重要而且需要记住的一点是，在系统项目中所需的模型在数量和格式上是不同的。尤其是项目团队对于所要建立系统的类型非常有经验时，较小且较简单的系统项目不需要模型来显示系统的每个细节。有时候，关键的模型会在几个小时内以非正式的方式创造出来。尽管模型通常是用特定的软件工具来创建的，但是有用且重要的模型有时会在餐巾纸上或者在机场候机室里的信封的反面被快速地画出来！对于任何一项开发活动，迭代方法都可用来建立模型。模型的初稿会解决一些但不是所有的细节问题。下个迭代也许就能完成更多的细节或者纠正之前的错误。

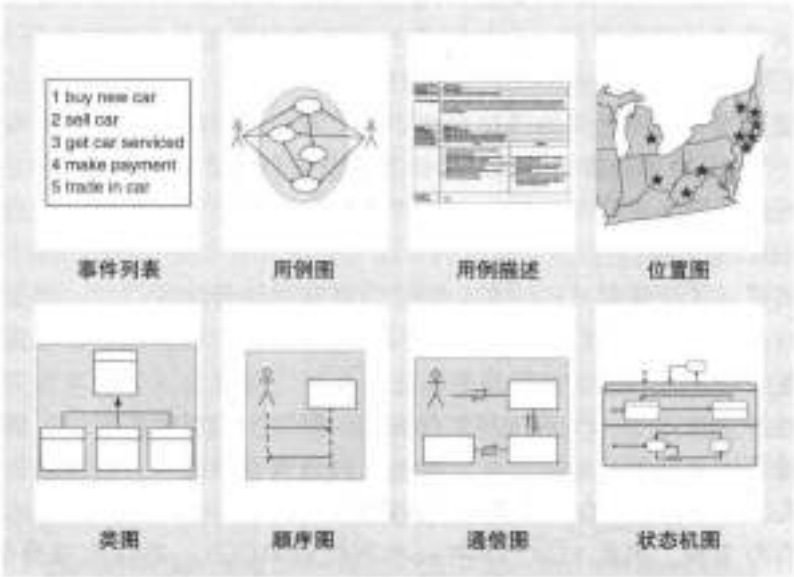


图 2-5　一些分析和设计模型

分析和设计模型可以分为以下三种基本类型。

- **文本模型**。分析员用文本模型作为备忘录、报告、叙述和列表来描述那些很详细且很难用其他方式表述的需求。图 2-5 中的事件列表就是文本模型的一个例子。叙述性描述通常是记录从利益相关者那里初步收集到的信息的最好方式，比如说在访谈中得到的信息。在很多情况下，叙述和其他文本模型会在以后转化成图形格式。
- **图形模型**。图形模型有助于理解那些很难用语言来描述的复杂关系。一句古老的谚语曾这样说过：一图胜千言。在系统开发过程中，一张仔细建立的图形模型可能比 100 万句话都有用！一些图形模型事实上看起来很像实际系统的一部分，例如，界面

设计或报表布局设计等。然而，在分析活动开发的图形模型代表了更多典型的抽象事物，如外部实体、处理过程、数据、对象、消息和连接等。

- **数学模型**。数学模型是描述系统技术方面的一个或多个公式。分析员通常用数学模型来代表科学与工程方面的功能需求，偶尔也会用数学模型描述会计和库存控制领域的业务系统需求。分析员和设计者也会用数学模型来描述需求和运行参数，比如网络流量或数据库查询的响应时间。

许多用在系统开发阶段的图形模型是根据**统一建模语言**（UML）中指定的符号所画的。图2-5中的用例图、类图、序列图、通信图和状态机图就是UML图形模型。UML是模型构建标准和由**对象管理组**（OMG）定义的符号标准的集合，其中OMG是开发系统标准的组织。通过使用UML，分析员和最终用户能描述和理解用在系统开发项目中的各种各样的特定图表。在UML存在之前，是没有任何标准的，因此图表可以被否决而且每个公司（每本书）所用的图表也是各种各样的。

我们在本章后面会通过详细考察一种类型的图形模型来扩大模型的讨论范围，这种图形模型就是工作流图。在后面的章节中，你会学习怎样开发许多其他类型的分析与设计模型。

2.6 利益相关者

系统功能需求信息的主要来源是新系统的各种利益相关者。**利益相关者**是对系统的成功运行感兴趣的人。根据系统的类型和范围，利益相关者可以是一个可小可大的、各式各样的组织。例如，当为一家美国上市公司实施一个综合性的会计系统时，利益相关者就包括记账员、会计师、经理、总经理、消费者、供应商、审计员、投资者、证券交易委员会以及美国国税局。每个利益相关者小组都以不同的方式与系统相互影响，而且每个成员对系统需求都有独一无二的看法。在收集到详细信息之前，分析员会确认每种对新系统感兴趣的利益相关者的类型，并且确保每个利益相关者小组的核心人物是业务专家。

有一个有用的方法能帮助我们确认所有感兴趣的利益相关者，就是考虑他们不同的两个特征：内部利益相关者与外部利益相关者，操作利益相关者与管理利益相关者（见图2-6）。**内部利益相关者**是那些在组织内部和系统相互影响的或者是对系统运行或成功有兴趣的人。你有可能想将内部利益相关者定义成组织的员工，但是一些组织（如非盈利组织和教育机构）有内部用户（例如志愿者和学生），不过他们并不是员工。**外部利益相关者**是那些在组织控制和影响范围之外的人，不过这个区别也是有些**模糊**的，就像是一个组织的战略伙伴（例如供应商和快递公司）也直接影响着内部系统。

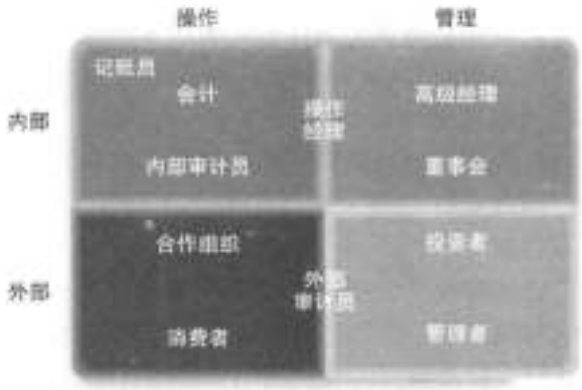


图2-6 上市公司综合会计系统的利益相关者

操作利益相关者是那些经常在工作或生活中与系统交互的人。例如记账员和会计与计费系统交互、工厂工人与生产调度系统交互、消费者与网上商店交互以及病人与保健网站、

Facebook 页面或推特新闻发送产生交互。操作层上的用户是需求信息的关键来源，尤其是它与用户界面及相关功能密切相关。**管理利益相关者**是那些不直接与系统交互但既能使用系统产生的信息又是在系统运行和成功上有投资兴趣或其他兴趣的人。例如组织的高级经理和董事会成员、管理机构以及税务机关。

在分析活动中考虑利益相关者是十分关键的，因为他们拥有的需求信息也许在组织中是不明显或者不广为人知的。此外，管理利益相关者强加的系统需求（尤其是外部需求）通常是有重要的法律效力和财政影响的。例如，要考虑到美国国税局规章对会计系统的潜在影响，或者是联邦和州政府的政策对社交网络系统的影响。

其他两种没有被描述到的利益相关者小组也是值得特别关注的。**客户**就是提供项目资金的人或组织。很多情况下，客户是高级经理。然而，客户也可以是一个独立的组织，比如由法人组成的董事会、母公司里的管理者或者是一个大学董事会。项目团队应包括重要利益相关者中的客户，因为团队在开发过程中必须为客户提供周期性的状态评论。指导或监督委员会的客户或是直接代表也会经常批准阶段项目和发放资金。

组织中的**技术支持**人员在任意系统中也是利益相关者。技术员包括建立和维护组织计算机环境的人。系统支持人员提供用户训练、故障诊断和相关服务。两个小组都会提供以下领域的指导：编程语言、计算机平台、网络接口和现有系统及它们支持的问题。任意一个新系统都必须符合这个组织现有的技术架构、应用架构和支持环境。因此，技术支持员工代表是重要的利益相关者。

RMO 的利益相关者

作为识别综合销售和市场营销系统（CSMS）的利益相关者的一个开端，开发一个顾客支持系统（CSS）的利益相关者列表是很有帮助的，包括：

- 电话 / 邮件销售订购人员。
- 仓库和运送人员。
- 维护在线目录信息的营销人员。
- 营销、销售、会计和财政经理。
- 高级管理员。
- 消费者。
- 外部运送人员（例如 UPS 和 FedEx）。

因为 CSMS 会接替 CSS 的现有功能，所以 CSMS 的利益相关者列表包括 CSS 列表中的所有利益相关者。然而，这里有一些细微的不同。例如，包含在 CSMS 中的社交网络功能和“山地雄鹿”计划扩大了谁是消费者这个概念。虽然过去的 CSS 是通过潜在消费者浏览网站时使用的，但是这个新系统可能会和更多外部利益相关者产生交互，包括现有消费者的亲朋好友以及流行的社交网站上的所有用户。实质上，利益相关者小组组成的“消费者”范围是更大、更不同的，也包括那些没有购买过 RMO 产品的人。同时确保这个不一样的组织在分析活动上的意见和需求是具有代表性的也是一个相当大的挑战。

与合作伙伴一起进行销售推广这个扩展功能创造了一个完整且全新的组织，包括合作伙伴内部的那些外部利益相关者。在这一点上，这个组织是否会包括操作利益相关者、管理利益相关者或两者都有以及那些利益相关者如何真正与系统交互依旧是不确定的。再次，确保那些利益相关者有足够的输入也将会是一个极大的挑战，尤其是利益相关者使用的那部分与

原有系统关联性不强的扩展功能。

RMO 是一个私营企业，John 和 Liz Blankens 是企业的拥有者，在他们下面还设有两个高级管理者职位。这样做是很重要，因为任何由许多股东持有的上市公司核心运营系统都要对其从系统到公司财务报告的信息流负责。RMO 是由外部的会计公司来审计的，通常是为了确保其与银行的贷款和其他私人金融业务。

作为拥有者和高级经理，John 和 Liz 也是首要客户，但是参与决策机构中的其他高级管理者一样也是重要客户。除此以外，现有的技术支持员工是核心利益相关者。图 2-7 以一个组织结构图的方式总结了内部管理利益相关者。

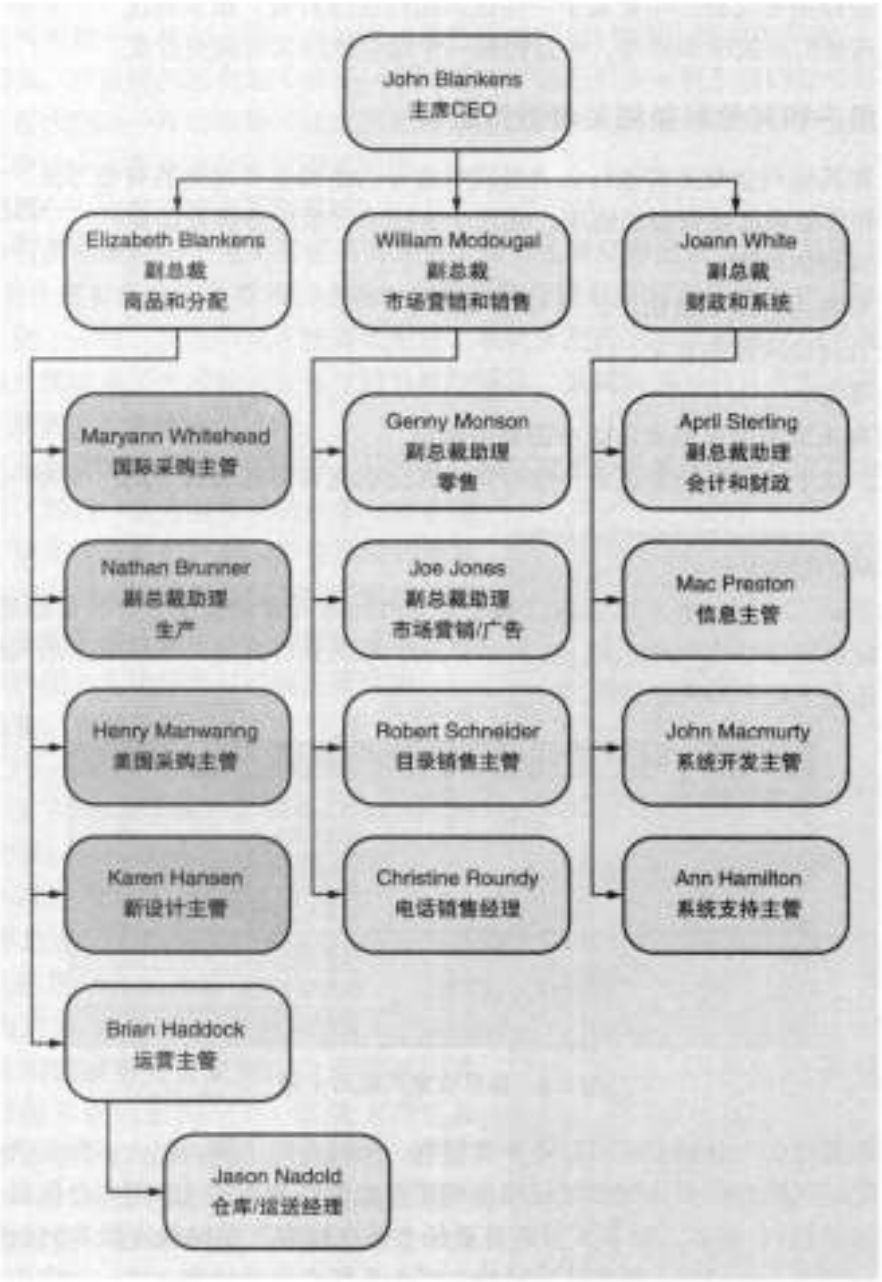


图 2-7　CSS 需求定义中涉及的 RMO 管理利益相关者

2.7 信息收集技术

收集详细需求信息的技术包括：

- 与用户和其他利益相关者进行访谈。
- 发放和收集调查问卷。
- 检查输入、输出和文档。
- 观察和记录业务流程。
- 研究供应商的解决方案。
- 收集活跃的用户评论和建议。

这些方法都是有效的，尽管其中一些比其他的更加有效。很多情况下，分析员会结合这些方法来提高他们的效率和效益，并且提供一个综合性的实情调查方法。

2.7.1 与用户和其他利益相关者进行访谈

与用户和其他利益相关者进行访谈是理解业务功能和业务规则的有效方法。然而，这也是最花时间和需要最昂贵资源的选项。在这个方法中，系统分析员需要：

- 准备详细的问题。
- 会见单独用户或小组用户。
- 获取且讨论问题的答案。
- 记录答案。
- 跟进在未来的会见或者访谈中需要的信息。

很显然，这个过程可能会花费一些时间，因此它通常需要和每个用户或用户小组进行多次谈话。

问题主题

无论在非正式会议、非正式访谈以及一部分的问卷或者调查中，分析员都会提出问题。但是分析员应该提出什么问题？图 2-8 展示了三个帮助分析员定义系统需求的问题主题。它同样展示了这些主题中问题的样例。

主题	对用户的问题
商业运营过程是什么？	你要做什么？
怎样进行商业运营？	你怎样做？
	你遵循什么样的步骤？
	你怎么做得与众不同？
进行商业运营时哪些信息是必需的？	你要使用什么信息？
	你需要使用什么样的输入？
	你需要产生什么样的输出？

图 2-8 信息收集问题的主题

业务流程是什么？ 分析员必须对业务流程有一个综合的了解。在大多数案例中，用户提供了当前系统的问题，分析员则必须仔细鉴别哪些功能是基本的（即哪些会保留，哪些会随着系统升级而消除）。例如，销售人员的首要任务是在顾客下单时检查顾客的信用记录。在新的系统中，销售人员可能不需要这项功能，因为系统会自动检索。这一功能依然是系统需求，但实施功能的方法和时间改变了。

业务流程如何运行？ 再次说明，聚焦点起始于当前系统但逐渐转向新系统。目标是决定新系统如何支持这一功能而不是如何马上支持功能。分析员通过新技术或方法来帮助用户了解更新、更有效的业务流程执行方法。

需要哪些信息呢？ 一些信息输出是正式的，一些是非正式的。当询问用户时，分析员应该专门询问异常情况。异常情况用以确定附加（非规定化）的信息需求。在这一主题和上一个主题中，细节是关键词。分析员必须理解根本的细节以开发正确的解决方案。

问题类型

问题可大致分为两类：

- **开放式问题**——比如“你如何使用这一功能？”这鼓励对方进行讨论和解释。
- **封闭式问题**——比如“你一天处理了几种形式？”这被用于特定的事实。

总的来说，开放式问题有助于展开一个讨论并且迅速引发未被发现的许多需求。需要注意的是，所有的问题一开始都是开放式问题。伴随着开放式问题的讨论常常会逐渐转成需要引出或具体确认一个业务流程的封闭式问题。

聚焦问题——当前系统还是新系统？

所有分析员面临的一个重大问题是要花多少精力去研究和记录当前的系统（如果已经存在）。过度关注现有系统会花费相当多的时间并可能导致最终只是使用新技术对系统做了一些简单的更新。这时，无论当前系统多么无效，系统开发者也只是重装存在的程序。另一方面，如果新系统继承了大多数或全部当前系统的需求，这样的话分析员会因为对现有系统分析不足而导致丢失重要需求的风险。

为了最大程度地降低两者的风险，分析员必须在发现新系统需求与审查当前业务功能之间取得平衡。得到一套完整且正确的系统需求很关键，但在如今快节奏的社会，没有足够的钱和时间来检查所有旧系统并记录所有程序缺陷。事实上，在如今的开发环境中，一个好的系统开发者最有价值的能力是能带来对问题的新看法。

访谈准备、实施及跟进

图2-9是一份示例清单，总结并覆盖了所有要点，这在准备、实施以及跟进访谈方面很重要。

准备访谈。 每一次成功的访谈都需要准备。准备访谈中最开始、最重要的一步是建立目标。换言之，你想通过这次访谈达到什么目的？写下目标，这能帮助你在脑海中留下很深的印象。第二步是决定哪些利益相关者将要加入这次访谈。当目标明确或事实清楚时，少量的被访者更好。在目标不明确的情况下，访谈大群体要更好，比如需要生成和评价新的想法时。然而，面对大群体时，从参与者中确保获得高质量的信息是有难度的。如果可能，在每次访问中至少有两个分析员，让他们事后比较记录来确保正确性。

访谈实施清单

之前
- 建立访谈的目标
- 决定访谈包含的正确用户
- 决定参加的项目团队成员
- 列出需要讨论的问题列表
- 检查相关的文件和材料
- 设置时间和地点
- 通知所有参与者访谈的目标、时间和地点

期间
- 准时抵达
- 寻找异常和错误情况
- 查探细节
- 全面记录
- 确认和记录未予答复的项目或者开放式问题

之后
- 精确、完整地检查记录
- 将信息转移到合适的模型和文件
- 感谢参与者
- 追踪未予答复的项目或者开放式问题

图2-9 用户访谈清单的样例

下一步是准备在访谈中使用的细节问题。写下特定问题的清单，并且根据之前收到的表格记录总结出要点。通常，你要准备一系列针对访谈对象的问题。开放式问题或者封闭式问题都是可以的。总的来说，开放式问题有助于讨论且鼓励用户解释业务流程和规则的所有细节。

最后一步是安排最终的访谈并将安排的步骤告知参与者。应该确定一个特定的时间和地点。如果可能，选择安静的环境，避免打扰。每个参与者要知道访谈的目标，并在适当的时间应该预览要用到的问题或材料。访谈可能会占用大量的时间，如果每个参与者事先知道要完成什么就会变得更有效率。

实施访谈。利益相关者访谈会议工作场所的流程是：事先计划，提早到达，确认房间可用且资源到位。限制访谈时间对分析员和利益相关者都是有益的。利益相关者有其他的责任，分析员一次只能获取一定的信息。几次较短的访谈会比一次性长时间的访谈更好。一系列访谈提供了充分从访谈材料中提取有用信息的机会，而且这些信息之后会变得更加清晰。

找机会询问“假设”类的问题。“如果这没达到？如果错失了信号？如果无法平衡？如果两个指令相同？”好的系统分析的本质是理解所有的“假设”情况。尽量确认所有异常状况，并做出回答。比起其他技巧，思考异常情况的能力会帮助你发现详细的业务规则。这是一项难以在书中教授的技能——实践出真知，你将会通过不断实践掌握这一技能。

除了寻找异常情况，分析员必须通过探测来确认已经完整了解了所有的程序和规则。作为系统分析员最困难的技巧之一是如何获得足够多的细节。对于程序是如何工作的，我们经常可以很容易地得出整体概览，但是别胆怯于询问细节问题，直到你完全了解它是如何运行的以及其他所需要的信息。不完全了解所有细节是不能完成有效的系统分析的。

记笔记是一个好方法。通常，录音记录让用户觉得紧张。然而，用户较认同记笔记因为这表示你认为你获得的信息是重要的。如果两个分析员在执行一次访谈，他们可以事后校对笔记。在你的笔记中标识和记录任何未回答的问题或未被解决的议题。好的笔记提供了案例分析模型的基础，同时也为下一次访谈中的会话做了基础准备。

图 2-10 是一次访谈会话的示例议程。显然，你无须精确遵循固定的议程。然而，正如图 2-9 展示的访谈清单，这能帮助你加强对于访谈中将要讨论的议题和事项的记忆。复印并使用它。如果你想找到自己的方式，也可以以喜欢的方式改变工作清单。

跟进访谈。跟进是每次访谈中的重要部分。第一项任务要吸收、理解以及记录获得的信息。通常，分析员根据业务流程建立的模型记录访谈细节，并写下关于非结构性需求的文本描述。这些任务应该在访谈后尽快完成，并请访谈的参与者验证所得出的结论。如果建模方法对于用户较复杂且不熟悉，那么分析员应该安排跟进会议来解释验证模型，正如本章节最后一部分所描述的。

访谈期间，你也许会问一些用户难以回答的“假设”问题。这些问题通常会是新系统中未曾考虑过的政策问题。这很重要，这些问题不能忘记或缺失。图 2-11 是 RMO 的跟进未解决或未完成问题的示例表。这一表格包含了用户或分析员提出的问题以及解决议题的责任与义务。如果几个团队同时工作，则可以产生一个集合清单。清单中可加入其他解释问题和数据解决方案的列。

最后，基于进一步细化或缺失的信息列一张新问题清单。这张清单将会为你的下一次访谈做好准备工作。

讨论和访谈议程

环境

访谈目标

　　决定销售委托费率的处理规则

日期、时间和地点

　　2012 年 4 月 21 日，上午 9 点，在 William McDougal 的办公室

用户参与者（姓名和职位）

　　William McDougal，市场销售副总裁，以及几个他的职员

访谈 / 讨论

1. 谁是合适的销售委托人？
2. 委托的基础是什么？支付多少费用？
3. 委托者如何返还收入？
4. 有特殊的激励措施吗？竞争措施呢？整个安排以时间为基础？
5. 委托者的规模是变动的吗？是定量的吗？
6. 什么是异常情况？

跟进

重要的决定或者问题的答案

　　参见附件中写明的委托政策

不能通过任务解决方案来解决的开放问题

　　参见开放问题列表中的项目 2 和项目 3

下一次会议的日期和时间或者跟进会议

　　2012 年 4 月 28 日，上午 9 点

图 2-10　访谈会话议程的样例

ID	问题标题	确定的日期	目标截止日期	项目责任人	用户联系人	说明
1	部分运输	6-12-2012	7-15-2012	Jim Williams	Jason Nadold	部分运输或者全部运输
2	返还和委托	7-01-2012	9-01-2012	Jim Williams	William McDougal	委托者从返还中获得补偿吗？
3	额外的委托	7-01-2012	8-01-2012	Mary Ellen Green	William McDougal	怎样解决委托者的特别促销？

图 2-11　未解决问题列表的样例

2.7.2　分发和收集调查问卷

问卷调查使分析员获得了大量从利益相关者处收集的问题。即使利益相关者是广泛分布的，他们还是可以通过需求问卷调查来帮助定义需求。问卷调查经常被用来初步洞察获得的利益相关者信息，这帮助我们决定哪个领域需要通过使用其他方法来进一步研究。

图 2-12 是一个展示了三类问题的问卷示例。第一部分是决定定量信息的封闭式问题。第二部分是由应答者回答他们对议题是否同意而组成的看法问题。这两个类型的问题都容易列表显示，以确定问题回答的平均数。第三部分是要求解释一个过程或问题。初步调查的问题有助于更深入的调查活动。

RMO 调查问卷

这张调查问卷将分发给电话销售部门的所有职员。如你所知，RMO 正在开发一个新的顾客支持系统，以提供更好购物体验和顾客服务。

调查问卷的目标是获取定义新系统需求的初步信息，跟踪讨论将会允许每个人详细说明系统需求。

第一部分。在典型的四小时轮班工作的基础上回答问题。

1. 你收到多少电话呼叫？ _____
2. 多少电话呼叫是为了订购产品？ _____
3. 多少电话呼叫是为了咨询 RMO 产品的资料，只是问问题吗？ _____
4. 估计有多少次当顾客打电话订购时缺货？ _____
5. 对于这些缺货订购，顾客决定退单的概率有多少？ _____
6. 有多少次一个顾客尝试从过期的目录中订货？ _____
7. 有多少次一个顾客在交流过程中取消订货？ _____
8. 有多少次一个订单因为顾客的不良记录而被拒绝？ _____

第二部分。在数字上画圈，1 – 7 表示你对同意或不同意一个申请的强烈程度。

问题	强烈同意						强烈不同意
与顾客交流时对产品进行更长的描述，这对我的工作更有益。	1	2	3	4	5	6	7
如果我有过去顾客的销售历史记录，这对我的工作更有益。	1	2	3	4	5	6	7
如果我有关于适合项目的配件的信息，这对我的工作更有益。	1	2	3	4	5	6	7
计算机的响应时间很长，使得我很难及时回答顾客的问题。	1	2	3	4	5	6	7

第三部分。请写下你的观点和评价。

_____

图 2-12 调查问卷样本

调查问卷不能有效地帮助你学习程序、流程图或技术。开放式问题（比如“你如何完成这个程序？”）最好用访谈法或观察法来回答。尽管调查问卷包含了很有限的开放式问题，但利益相关者通常不会响应那些他们认为包含很多开放式问题的调查问卷。

2.7.3 检查输入、输出和流程

关于输入、输出和流程有两种信息来源。一种是来自组织外部——全行业的专业组织和其他公司。从其他公司获得信息也许并不容易，但这是重要信息的潜在来源。有时，工业期刊和杂志会报道最新的研究成果。如果团队成员对最好的实践成果并不熟悉，那么可以说是项目团队的疏忽。

第二种输入、输出和流程的来源是组织内部的业务记录和流程描述。检查访谈记录和流程有两个目的。第一，这是初步理解流程的好途径。第二，已有的输入、输出和文档在访谈期间可作为视觉资料，在讨论时也可作为工作文档（见图 2-13）。讨论可以集中于特定的输

入或输出，集中于它的目标、贡献以及信息内容。讨论也需要包含特定的可初始化输入或生成输出的业务项目。几个不同的业务事件可能需要相同的形式，关于事件和业务流程的特定信息是很关键的。屏幕和表格必须填满真实信息，这对于确保分析员获得关于数据内容的正确理解是很有帮助的。

Ridgeline Mountain Outfitters—Customer Order Form

OUTFITTERS

Name and address of person placing order.
(Please verify your mailing address and make correction below.)
Order Date ___/___/___

Name
Address　Apt. No
City　State　Zip
Phone: Day (　)_____　Evening (　)_____

Gift Order or Ship To: (Use only if different from address at left.)
Name
Address　Apt. No
City　State　Zip
Gift ☐　Address for this Shipment Only ☐　Permanent Change of Address ☐
Gift Card Message
Delivery Phone (　)_____

Item No.	Description	Style	Color	Size	Sleeve Length	Qty	Monogram	Price Each	Total

MERCHANDISE TOTAL _____
Regular FedEx shipping $4.99 per U.S. delivery address _____
(Items are sent within 24 hours for delivery in 2 to 4 days)
Please add $4.99 per each additional U.S. delivery address _____
FedEx Standard Overnight Service _____
Any additional freight charges _____
International Shipping (see shipping information on back) _____

Method of Payment
Check/Money Order ☐　Gift Certificate(s) ☐　AMOUNT ENCLOSED $ _____
American Express ☐　MasterCard ☐　VISA ☐　Other ☐
Account Number
MO YR
___/___
Expiration Date
Signature

图 2-13　RMO 在访谈中使用邮件订单作为视觉资料

检查现有处理流程的文档有助于识别访谈中未提及的业务规则和记录。正规的流程分析文档也有助于发现业务流程中的差异和冗余。然而，流程记录时常不能更新，且通常包含错误。为了确保来自现有记录的假设和业务规则是正确的，分析员应该和用户一起检查。

2.7.4　观察和记录业务流程

第一手资料对于了解业务流程是很重要的。比起其他的活动，观察业务流程会帮助你了解业务的功能。然而，在观察现有的业务流程时，你必须留意新系统和业务流程间的联系。也就是说，当你通过观察当前的业务流程图来了解基础业务需求时，不要忘记流程经常会发生变化以达到更有效率的目的。不要局限于认为实现流程只有一种方法。

你可以通过很多方式来观察业务流程，例如，对办公室进行快速浏览或者自己亲身实践用户的工作。通过快速浏览能对办公室布局有整体的了解，还能了解计算机设备的使用和需求以及整个工作流的概貌。花几个小时观察用户是如何工作的，这将帮助你了解他们使用计算机系统的细节以及业务功能的实施。通过像用户一样接受训练和实际工作，你会发现学习新程序的困难之处，体会到系统易于使用的重要性，并理解现有的程序和信息来源为何会成为绊脚石和瓶颈。

以同样的仔细程度观察所有的流程是没有必要的。快速浏览对于一项进程是足够的，但

对于一个关键或难以理解的进程则需要更长时间的观察。要记住，你的目标是获得对于业务流程和规则的完整理解，这样你就能合理评估在哪里花时间才能获得对项目的了解。在访谈中，两个分析员在观察过程中齐心协力，效果通常会更好。

观察使用户紧张，所以你尽量做到不唐突。你可以以采取一些让用户放松的方式，比如跟用户一散步或者同时观察几个用户。理解并关心用户的需求及情感，这通常会产生积极的效果。

2.7.5 研究供应商的解决方案

公司期望新系统能处理的许多问题或许已被其他公司解决了。在大多数情况下，咨询公司可能会处理同样的问题，有时候，软件公司也可能已经为特定的业务开发了相应的系统软件包。利用现有的知识或解决方案可以避免犯高代价的错误并节省时间和金钱。

研究现有的解决方案有三方面的积极作用，但同时也有一个危险的方面。第一，研究这些解决方案通常会帮助用户思考如何更好地运行他们的业务功能。看别人如何解决问题并将他们的想法应用于自己公司的企业文化和结构中，这往往会为业务需求提供可行的替代方案。

第二，有些解决方案已经是一流的方案了。如果不研究这些方案，开发团队可能在系统形成概念前就创建一个已过时的系统。公司不仅要解决业务问题，而且要跟上当前竞争的发展趋势。

第三，购买一个解决方案比新建一个方案更便宜，风险性更小。如果解决方案满足公司需求且能直接买到，这将是一个更安全、更便捷、更省钱的路径。

探索现有方案的危险是，有时用户甚至是系统分析员可能想立刻购买某个方案。但如果过早购买一个方案，比如过早购买了系统软件，那么公司的需求则可能没有得到充分研究。很多公司急于购买系统，之后才发现这些系统仅能满足半数的需求。在充分定义需求且全面研究可行的选择方案之前不要匆忙做购买决定。

2.7.6 收集活跃的用户评论和建议

正如第 1 章和本章前面所讨论的，系统开发通常和分析、设计以及其他的分散在多个迭代中的活动同时进行。在每个迭代中，我们构建系统的某一部分并进行测试。在对这些功能进行迭代的过程中，用户和其他利益相关者执行初步测试。在之后的迭代中，他们还会测试和使用那些相同的功能。

用户在最初和后续测试中的反馈是一个有价值的需求信息的来源。访谈、讨论以及模型审查并不是获取准确需求的完美方法。“我一看便知”，这句话很适用于需求定义。在用户与实施这些需求的系统进行交互之前，用户通常不能完整或准确地陈述他们的需求。基于这些交互，用户能为提高和定义缺失的需求或为完善需求提出具体建议。

2.8 用活动图记录工作流

当你为业务流程收集信息时，你需要记录结果。一个有效的捕捉信息的方法是图表法。最后，你可能想用图表来描述新系统的工作流，但现在，让我们关注如何记录现有的业务流。

工作流是按序的处理流程，用于处理一个完整的商业事务或顾客请求。工作流可以是简

单或复杂的。复杂的工作流可能由数以百计的进程阶段组成，也可能包含不同组织部分的参与者。

活动图描述了多样的用户（或系统）活动、从事各项活动的人以及活动的顺序流。

图2-14展示了活动图中的基本符号。椭圆表示工作流中独立的活动。相关联的箭头表示各个活动间的顺序。黑圆圈指示了工作流图的开头和结尾。菱形是表示进程是否会跟在一个或另一个路径后的决定点。加粗实线是**同步条**，它分割了多个并行路径或重组并行路径。**泳道标题**呈现了执行活动的代理人。在工作流中，不同的代理人执行不同的工作流阶段是正常的，因此泳道标题把流图活动划分成了展现代理人执行哪些活动的不同小组。

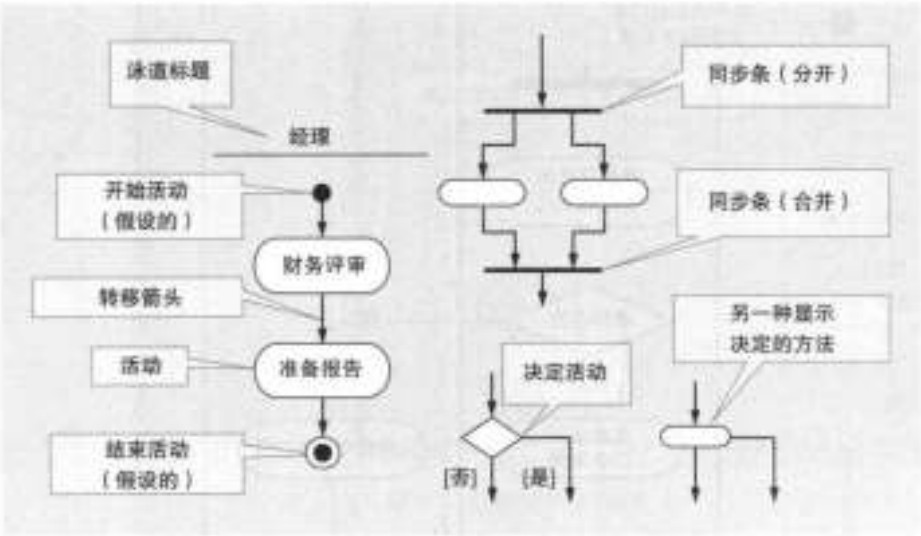


图2-14 活动图符号

图2-15是一个描述当前RMO CSMS系统实施订单流程的活动图。进程在顾客完成订单的付款过程时开始。图表描述了来回的信息流以及订单子系统、库存子系统、仓库和运输之间控制流程。图表是简化的，因为它省略了许多错误处理途径，包括如果手头有可用的货物库存能满足订单的一部分时应该怎么做。

图2-16阐明了另一个工作流图，它演示了一些新概念。这是一个私人订制的例子，需要特定的生产过程以满足顾客的需求。销售员将订单传给工程师，且图表中使用新符号来强调销售员和工程师间的传输文件。在工程师开发出规格说明后，两个并发的活动发生了：采购订单的材料，为自动铣床编写程序。这两个活动完全独立并可以同时发生。注意，一个同步条将路径分离成了两个并发的路径，另一个同步条则将它们连接起来。最后安排订单的生产进度。

用活动图记录工作流是简单易懂的。第一步是识别适于创建泳道的代理人。接着，按照各个步骤的工作流程，做出适当的关于活动的椭圆。用箭头连接活动的椭圆形来展示流图。这里有如下几个指导方针：

- 使用决策符号来呈现一种现状——选择一条路径，或另一条路径，但不是两条路都能选。作为一个速记符号，你可以合并一个活动（通过使用椭圆）和一个决定（通过使用菱形），使它们成为单一的有双箭头的椭圆，正如图2-14右边所示。这个符号呈现

了一个决策活动。无论在哪里有一个称作“核实”或者“检查”的活动，你都可能需要一个决定——一个“接受”路径或一个“反对”路径。

- 为并行路径使用同步条——两条路径都选取的情形。同步条包括开始和结束同步条。你可以使用同步条来呈现循环，比如“do while”循环程序。将同步条放置在循环的开头可描述成“每当”，将同步条放置在结尾可描述成“每次结束”。

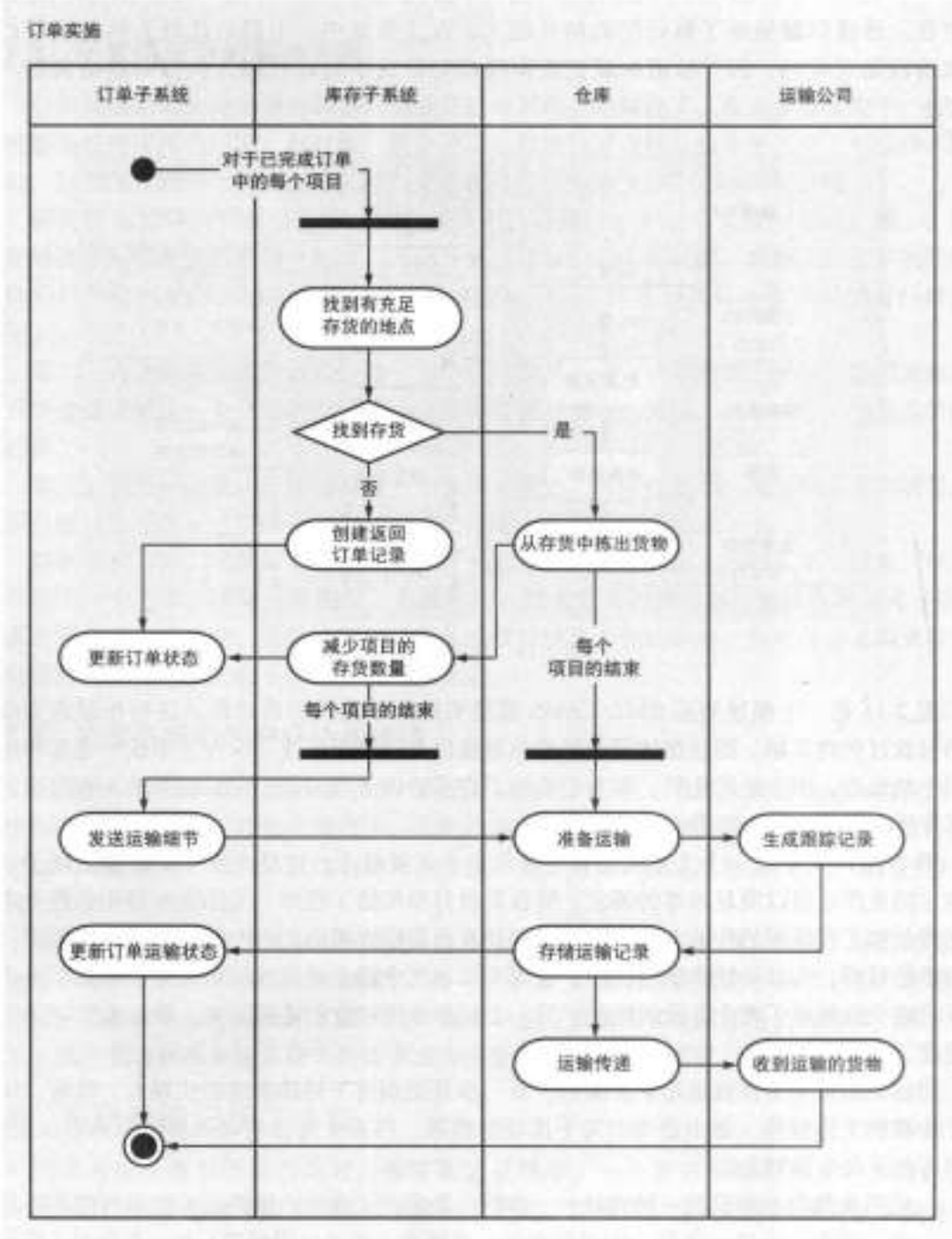


图 2-15 在线结账的简单活动图

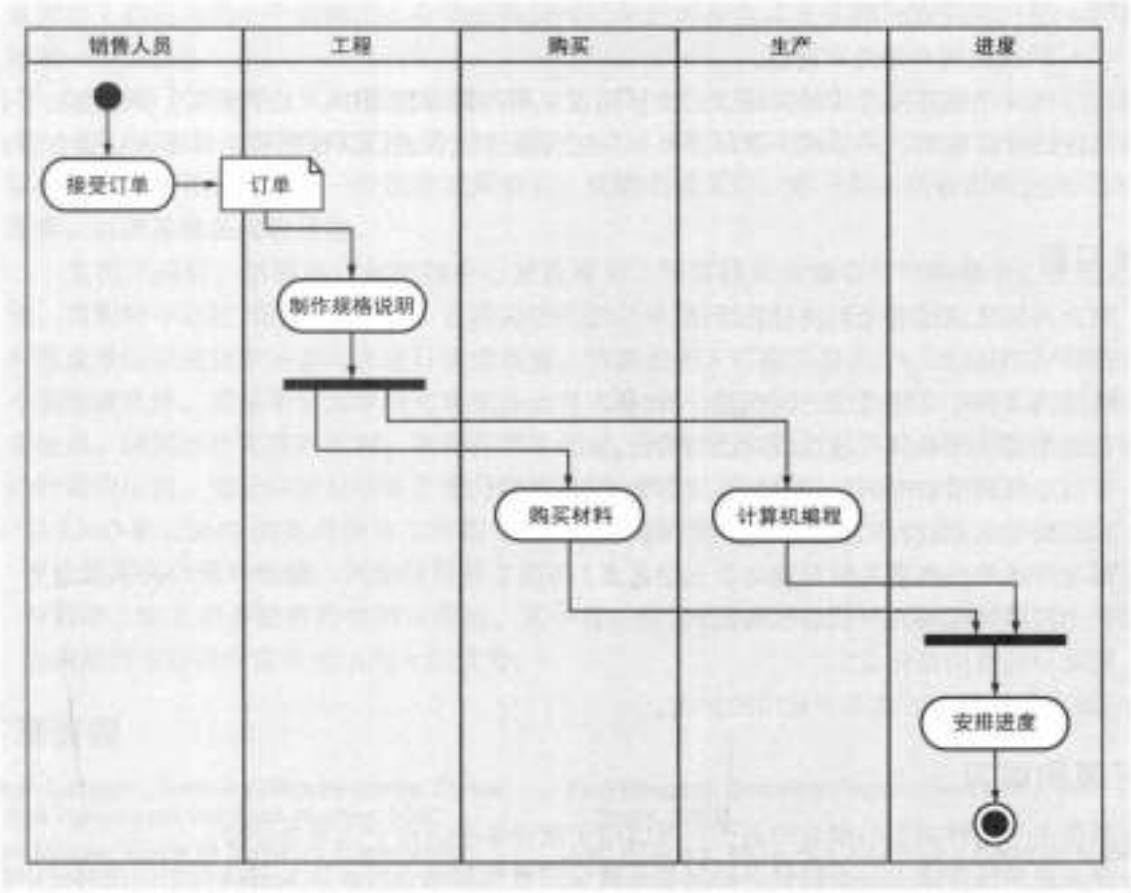


图 2-16 同时发生路径的活动图

本章小结

有五项系统分析的基本活动：

- 收集细节信息。
- 定义需求。
- 为需求划分优先级。
- 开发用户界面对话框。
- 与用户一起评估需求。

功能性需求解释了新系统必须支持的基本业务功能。非功能性需求包括系统关于技术、性能、可用性、可靠性和安全性的目的。

数学法、描述法和图解法都可用于记录需求，并可在与用户和其他利益相关者一起进行需求评估时作为辅助资料。利益相关者包括内部和外部的用户以及其他在系统中受益的人或组织。

分析员使用多种技术来收集需求信息，包括：

- 访谈。
- 调查问卷。
- 文档、输入和输出审查。

- 研究供应商的解决方案。
- 活跃的用户评论和建议。

工作流图是建模技术的关键，它常被用在早期的需求模型中。工作流图生动地建立了业务流程的阶段模型，并呈现了参与者是如何进行业务流程的。其他的模型和图表涵盖在后面的章节中。

复习题

1. 列出并简要描述五个系统分析活动。
2. 三种模型是什么？
3. 功能需求和非功能需求的区别是什么？
4. 描述准备、实施和跟进访谈对话的阶段。
5. 在信息集成活动中研究供应商解决方案的好处是什么？
6. 什么类型的利益相关者会被纳入实情调查中？
7. 描述开放项目列表并解释它为什么很重要。
8. 列出并简要描述 6 种信息收集技术。
9. 活动图的目的是什么？
10. 画出并解释在活动图中使用的标志。

问题和练习

1. 提供关于三种类型的模型的例子，可以用于设计车子、房子以及办公楼。
2. 在调查系统需求中一个最棘手的问题是确保完整性和综合性。你如何确保在访谈对话中获得了全部的正确信息？
3. 在调查中你会遇到“需求蔓延”的问题（即用户对特征和功能有额外的需求）。这是因为用户有许多未解决的问题，而且这可能是第一次有人在倾听他们的需求。你如何确保系统不会因增加不必要的功能而不断扩充规模？
4. 当两位不同的访谈者对于同一流程问题给出有冲突的答案时，你该怎么办？当其中一位是员工而另一位是部门经理时你又该怎么办？
5. 您已经被委派来解决关于开放项目列表的几个问题，你很难通过与用户的交流来制定决策。你如何说服用户采纳这一政策？
6. 在 RMO 的运行案例中，假设你已经和航运部门的经理安排了会面。你的目的是确定航运工作如何进行以及获知新系统将需要什么信息。列一张问题清单，你将用到开放式问题和封闭式问题。可以使用任何问题和技术来确保你能得到异常情况。
7. 根据以下叙述构建一个活动图。在构建模型时记录任何有争议或有疑问的事项，有时也可记录一些设想。

   公司采购部负责采购公司其他部门要求的货物。公司中最初创建采购部的人要求采购部门做到顾客至上。一旦采购部的工作人员接到任务，就要按要求和标准监控，直到任务完成。

   采购员工作要求如下：采购 1500 美元以下的产品时，填制采购单，之后将填好的采购单交给审核人员；采购 1500 美元以上的产品时，必须先让审核过的产品供应商进行投标，得到标书后，采购员选出合适的供应商并填制采购单，之后送交审核。

8. 根据以下叙述构建一个活动图。在构建模型时记录任何有争议或有疑问的事项，有时也可记录一些设想。

运输部门负责接收未处理的购物订单。当工作人员接到运输单时，应找出这些项目的未完成购物订单。之后将其复印多份并分别发放。一份送往购物中心，该部门将更新记录以标识购物订单完成。另一份送往结算中心，从而完成支付。第三份送往做出购物请求的顾客，以便其准备接收货物。

支付完成后，结算部门向购物中心发送通知。顾客收到货物后会向购物中心发送通知。当购物中心收到这些验证后，它将关闭购物订单并标记该项目已完成且已支付。

9. 对商业或组织流程中的参与者进行实情调查。被调查的人可能来自大学，或是来自周围的小型生意伙伴，或是来自大学里的学生办公室志愿者，或是来自医疗机构，或是来自志愿者组织。识别出已完成的流程，如保存学生记录、顾客记录或成员记录。将问题列出后再进行调查访谈。切记你的目的是要充分理解流程（例如，成为某一流程中的专家）。

10. 以 RMO 和 CSMS 的案例作为工作指导，列出所有可进行后续研究的流程。你可能还会考虑使用你以前的经验，例如销售商方面的 L.L.Bean、Lands' End 和 Amazon.com。打开网络，会有很多销售商的网页弹出，其中有一些潜在的能完成供应活动的供应商。列出满足要求的供应商并说出每个的优势。

扩展资源

Soren Lauesen, *Software Requirements: Styles and Techniques*. Addison-Wesley, 2002.

Stan Magee, *Guide to Software Engineering Standards and Specifications*. Artech House, 1997.

Suzanne Robertson and James Robertson, *Mastering the Requirements Process*, Second Edition. Addison-Wesley, 2006.

Karl Wiegers, *Software Requirements*. Microsoft Press, 2003.

Karl Wiegers, *More About Software Requirements: Thorny Issues and Practical Advice*. Microsoft Press, 2006.

Ralph Young, *The Requirements Engineering Handbook*. Artech House, 2003.

用　　例

学习目标

阅读本章后，你应该具备的能力：

- 阐述为什么确定用例是定义功能需求的关键。
- 描述确定用例的两种技术。
- 将用户目标技术运用于确定用例。
- 将事件分解技术运用于确定用例。
- 将 CRUD 技术运用于确认和细化用例列表。
- 描述用例图的符号和作用。
- 通过参与者和子系统画用例图。

开篇案例　Waiters on Call 餐饮送货系统

Waiters on Call 是在 2008 年由 Sue 和 Tom Bickford 开创的一项餐厅送餐服务。这对兄弟在上大学时就曾在餐厅打工，他们的梦想就是开一家属于自己的餐厅。但遗憾的是，最初的投资均以失败告终。Bickford 兄弟发现，许多餐厅提供外卖食物，还有一些餐厅（主要是匹萨饼店）会提供送货上门服务。然而，许多他们认识的人似乎希望送货上门服务中有更全面的食物选择。

Sue 和 Tom 认为电话订餐是最佳选择：一项不需要很高初始成本投资的餐饮项目。他们和全城许多知名的餐厅签订合同，接到顾客的订单然后负责将全部饭菜送货上门。当饭店准备好预订的食物后，餐厅会按批发价交给 Waiters on Call，饭菜送到后，顾客按零售价付款，并支付服务费和小费。Waiters on Call 刚开始时规模很小，仅包括两家餐厅和一个在就餐时间工作的司机。随着业务的快速扩大，Bickford 兄弟意识到他们需要一套专门的计算机系统来支持运营。他们雇用了一个顾问 Sam Wells 来帮助他们定义所需要系统的种类。

“在运营过程中有哪些因素促使你们想要一个计算机系统？” Sam 问道。“告诉我这些业务是如何进行的。”

“好，” Sam 回答道，“当一个顾客打电话订餐时，我需要记录信息并且通知相应的餐厅。我也需要决定派哪个司机去送餐，因此我需要司机们打电话通知我他们什么时间有空。有时候，顾客们会修改订单，因此我又需要处理初始订单并且通知餐厅修改。”

“好的，你是怎么样管理现金的？” Sam 问。

Tom 插话道：“司机从餐厅取饭时会从餐厅直接拿到账单的复印件。账单应该与我们的计算结果一致。司机送餐时收取相应的现金及额外的服务费。下班时，司机报账。我们会把他们收取的钱汇总，并且与我们的记录进行对比。所有司机交完账后，我们必须开张银行存款单，存入当天的总收入。每周末，我们会计算出应该付给每个餐厅的款项，把结算单与支票寄给他们。”

"你还想从这个系统中获取什么其他信息?" Sam 继续问。

"如果在每周末能统计出每个餐厅有多少订单、城里每个区有多少订单，以及诸如此类的信息就更好了。" Sue 补充道，这能帮助我们制定广告策略及与餐厅订合同，我们还需要每个月的财务统计报表。"

在 Sue 和 Tom 说话时，Sam 记下要点并且草拟了图表。然后，在经过一段时间的思考后，他总结了 Waiters on Call 的情况："在我看来，你们需要的系统负责在以下事件发生时进行一些处理工作：

- 顾客打电话订餐时，需要"记录订单"。
- 司机完成一次送餐时，需要"记录送餐完成"。
- 顾客打电话修改订单时，需要"更新订单"。
- 司机报告工作时，需要"签收"。
- 司机上交一天的收入，需要"协调收据"。

然后，你们需要系统在特定的时间点提供需要的信息，例如：

- 提供按日计算的存款单。
- 提供每周支付给餐厅的账单。
- 提供每周销售报表。
- 提供每月财务报表。

"我的设计正确的吗?"

Sue 和 Tom 一致同意，认为 Sam 所说的系统满足了他们的需求。他们确信已经找到了合适的顾问。

3.1 引言

第 2 章描述了系统开发中用到的系统分析活动，介绍了完成第一个分析活动所包含的许多任务和技术——收集系统、利益相关者和系统需求的相关信息。定义系统功能需求和非功能需求需要大量的信息。本章及第 4 章和第 5 章将介绍通过创建不同的模型来将功能需求整理成文档的一些技术。尽管分析阶段的所有活动实际上是平行进行的，但是这些模型是作为定义功能需求这一分析活动而创建的。

实际上，所有系统开发的新方法都始于需求模型的处理，而这个处理过程中就包含用例概念。**用例**是系统执行的一个活动，通常是为了响应用户的需求。在第 1 章，RMO 贸易展览系统这个示例已经列出了一系列用例，包括查询供应商、输入 / 更新信息以及上传产品信息。推荐两种定义用例的技术：用户目标技术与事件分解技术。另一种称为 CRUD 的技术也经常用于确认和增加用例列表。下面将会描述这些技术。

3.2 用例和用户目标

定义用例的一个方法是**用户目标技术**，这个技术要求用户描述他们使用新系统或者更新系统的目标。分析员首先会定义所有用户，其次会组织对每个用户的访谈。通过在一个时间段内聚焦于一类用户，分析员能够有条不紊地记录下定义用例的问题。

在访谈过程中，分析员会引导用户确定一些计算机帮助用户执行任务的具体方法。主要目标是为了发现一个系统怎样提升用户的执行效率。次要目标可能会包括将用户当前执行的任务流程化或者使用户能执行新的任务（那些新任务在当前系统中无法完成）。随着这些目

标的提出和描述，分析员从用户那里获取特定请求，并期望从目标系统获得回应，这些内容会被分析员记录下来作为用例。尽管用户是这些信息的首要来源，但通常需要分析员来引导他们以跳出常规的方式来思考他们当前的工作方式。

回顾一下在第 2 章中提到的 RMO 综合销售系统中不同的用户目标。在这个例子中，分析员需要与在运输部门工作的员工进行交谈来定义他们的特定目标。这些目标可能包括：运输货物、跟踪运输以及创建退货信息。当和市场营销部门的员工交流时，目标定义可能包括：增加 / 更新产品信息、增加 / 更新促销以及提供销售历史报告。当考虑到潜在顾客的这一目标时，分析员可能会要求一些人从顾客的角度来思考这个系统，让他们想象增值功能以及能使 RMO 具有吸引力的功能。可以形成焦点小组（focus group）来揭开潜在顾客的需求。潜在顾客的目标定义包括搜索商品、加入购物车以及查看用户评论和排名。图 3-1 列出了 CSMS 系统的几个潜在用户的用户目标。

用户	用户目标
潜在顾客	搜索商品 加入购物车 查看用户评论和排名
市场营销经理	增加 / 更新产品信息 增加 / 更新促销 提供销售历史报告
运输人员	运输商品 运输跟踪 创建退货信息

图 3-1　用“用户目标技术”识别用例

运用用户目标技术来确定用例包括以下步骤：

1. 定义新系统所有的潜在用户。
2. 根据潜在用户的功能角色（比如，购物、营销、销售）进行分类。
3. 通过组织的层次进一步将潜在用户分类（比如操作、管理、执行）。
4. 针对每种用户进行采访，并发现他们使用这个新系统时的一系列特定目标。以当前他们的目标为开端，然后让他们想象那些创新且能增加价值的功能。鼓励他们以动词加名词的格式设定目标，例如增加顾客、更新订单以及提供月底报告。
5. 创建按用户类型组织的主要用例列表。
6. 寻找相似用例名字的重复之处并解决矛盾。
7. 定义哪些不同种类的用户需要同样的用例。
8. 与每种用户和感兴趣的利益相关者一起评审已完成的列表。

3.3　用例和事件分解

定义用例最复杂的方法是事件分解技术。**事件分解技术**首先定义所有的业务事件，这些事件会引起信息系统的回应，同时每个事件都会产生一个用例。以业务事件为开端能帮助分析员在恰当的程度上定义每个用例。例如，第一个分析员可能会在表格中输入一个顾客姓名并将其定义为一个用例。第二个分析员可能会在增加一个新顾客的整个处理过程中定义一个用例。第三个分析员甚至可能会针对顾客一整天的工作定义一个用例，这其中包括增加新顾客、更新顾客记录、删除顾客、跟踪逾期付款的顾客或者联系以前的顾客。第一个例子范围太过狭隘以至于用处不大。第二个例子定义了一个完整的用户目标，它是合适的用例分析级别。第三个例子范围太宽，因此也用处不大。

用来确定用例的合适细节级别的是**基本业务流程**（EBP）。EBP 是由一个人在特定地点为了响应交易事件所执行的一项任务，它增加了可衡量的业务价值，保持了系统与数据的稳定和一致。图 3-1 中，RMO 的 CSMS 系统中会成为用例的用户目标的是搜索商品、加入购物车以及查看用户评论和排名，等等。这些用例对于基本业务流程来说是很好的例子。加入

购物车是为了回应业务事件“顾客想要购物”。有一个顾客正在加入购物车，同时对他来说还意味着能将具有可衡量价值的商品加到购物车中。当这个顾客停止增加商品并且移动到别的任务中时，那么这个系统就会保留当前的记录并且准备好调换到新任务中。

注意，每个EBP（以及每个用例）发生时会对一个业务事件做出回应。**事件**会在特定的时间地点发生，能够被描述出来同时也能被系统所记住。事件会驱动或者触发所有系统的处理过程，因此当你需要通过定义用例来定义系统需求时，将事件列出来并且进行分析是很有意义的。

3.3.1 事件分解技术

就像之前阐述的，事件分解技术聚焦于定义出系统必须对哪些事件做出响应，然后决定系统怎样响应（例如，系统用例）。当定义系统需求时，先调查清楚什么业务事件发生时需要系统做出响应。通过询问影响系统的事件，你会把注意力集中到外部环境，同时将系统看成一个黑盒子。最初的视角能帮助你主要关注系统的高层次，而不是系统内部的运作。这也使你的注意力集中在系统与外部人员以及与其他系统的接口上。

图3-2列出了一些对零售商店的赊购账务处理系统来说很重要的事件。功能需求基于六个事件分解为用例。顾客触发了三个事件：“顾客支付账单”“顾客赊账”“顾客变更地址”。系统会用三个用例做出响应：记录付款信息、处理赊账或者维护顾客数据。三个系统内部的事件在如下时刻被触发：“应发送月报清单”“应发送过期通知”“应生成周末汇总报表”。系统用如下用例做出响应：生成月报清单、发送过期通知以及生成汇总报表。按照事件描述系统能让赊购账系统集中于业务需求与基本业务流程。下一步是分配开发人员的工作：一个分析员集中于由人触发的事件，而另一个分析员集中处理内部触发的事件。系统以这样的方式被分解，这样使它能更容易被理解。结果是在一个由处于合适的分析级别的业务事件所触发的用例列表。

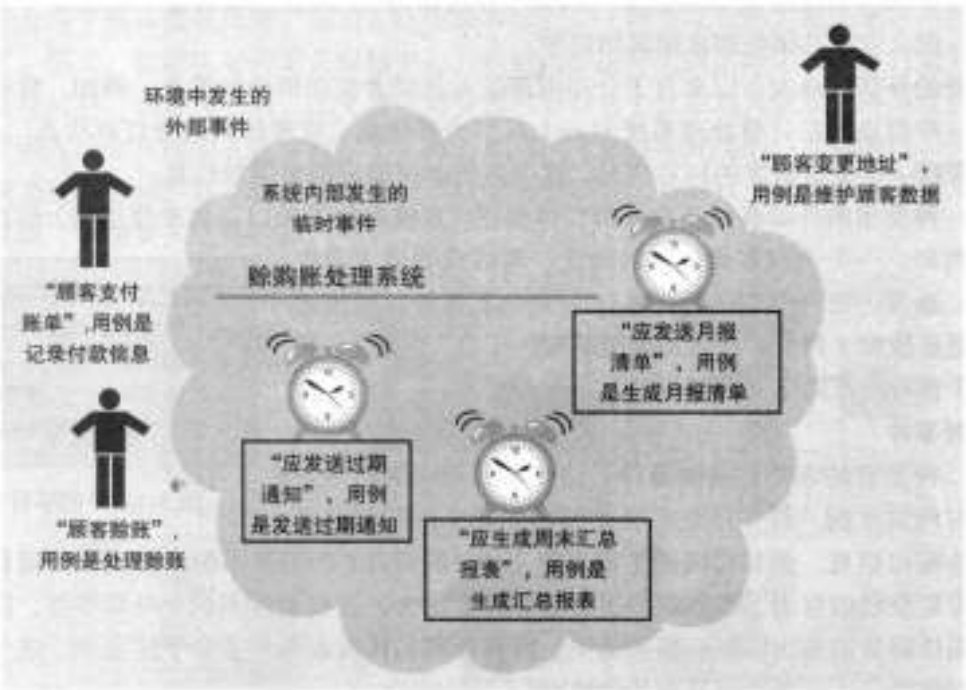


图3-2 得到用例的赊购账处理系统中的事件

早在20世纪80年代早期，实时操作系统中采用的现代结构分析就强调了事件这个概念

对于定义功能需求的重要性。实时操作要求系统立刻响应环境中发生的事件。早期的实时操作系统包括生产过程控制与航空导航系统。例如，在处理控制过程中，如果有一桶化学品满了，系统需要关闭进阀门。相关事件是“桶满了”，同时系统需要快速响应那个事件。在飞机导航系统中，如果飞机的海拔高度低于5000英尺[⊖]，那么这个系统需要开启低海拔警报。

现在大多数信息系统的交互性很强，因此它们可以被认为是实时操作系统。实际上，用户期望的是实时响应每件事。因此，现在通常采用事件分解方法来确定业务系统的用例。

3.3.2 事件类型

当使用事件分解技术来定义用例时要考虑三种事件类型：外部事件、临时事件与状态事件（也称为内部事件）。分析员在开始工作时会尝试着定义与列出尽可能多的这些事件，然后当他们与系统用户交流时再进行不断的细化。

外部事件

外部事件是指发生在系统之外的事件，通常是由外部实体或参与者触发的。外部实体（或**参与者**）是一个为系统提供或从系统接收数据的人或组织单位。为了定义关键的外部事件，分析员首先会尝试确定所有可能从系统中获取信息的外部实体。客户是外部实体的一个典型例子。顾客下订单时可能会想订购一个或者多个产品。对于RMO需要的订单处理系统来说，这是一个非常重要的事件。但是其他事件会和顾客联系在一起。有时，顾客会想要退换一个产品或者需要按发票支付订货的费用。像这样的外部事件就是分析员寻找的类型，因为分析员要定义系统需要完成哪些功能。这些外部事件会引出一些系统必须处理的重要事务。

当描述外部事件时，为这个事件命名是很重要的，这样外部代理才能清晰地定义。描述中也要包括外部实体需要进行的活动。因此，事件“顾客下订单”描述了外部实体（顾客）和顾客想要执行的能直接影响系统的活动（订购商品）。而且，如果这个系统是一个订单处理系统，那么它就必须处理该顾客的订单。

重要的外部代理也可以来自于公司内部的人员或者组织单位的需求，例如，管理部门请求得到一些信息。在订单处理系统中一个典型的事件是“管理部门检查订单状态”。也许管理者想跟踪一个关键顾客的订货情况，这个系统必须能定期提供该信息。

另一种类型的外部事件是在外部实体提供（系统存储下来以备将来使用的）信息时所触发的。例如，一个老顾客通知他的地址、电话或者雇主发生了变动。通常，每种类型的外部实体对应的事件可以被描述成处理更新数据，例如“顾客需要更新账户信息”。图3-3提供了一个清单来帮助定义外部事件。

要定义的外部事件包括：
- 外部实体想要触发一个事务
- 外部实体想要获得一些信息
- 数据发生变化，需要被更新
- 管理层想要获取某些信息

图3-3 外部事件清单

临时事件

第二种类型的事件是**临时事件**，这个事件是在到达某一个时刻时所发生的。许多信息系统按照事先定义的时间间隔产生一些输出信息，例如每两周生成工资支票（或每月）的工资系统。有时输出信息是管理者需要定期获取的报表，例如业绩报表或者异常报告。这些事件不同于外部事件，因为系统能自动提供需要的输出结果。换句话说，没有外部实体或者参与者会下达命令，这个系统在需要的时候能产生所需的信息或其他输出。

分析员是通过询问系统必须完成任务的最终期限来确定临时事件的。在截止日期前应生

⊖ 1英尺 = 0.3048米。——编辑注

成哪些输出信息？在截止日期前什么样的处理任务需要系统完成？分析员经常会通过定义当时系统需要产生的结果来确定这些事件。在一个工资系统中，临时事件可能称为“每两周生成一次工资单。”定义每月总结报告需求的这个事件可能命名为“生成每月销售汇总报表。”图3-4提供了一个检查列表用来定义临时事件。

临时事件不一定会在指定日期发生。它们可以在已经定义好的一段时间后再发生。例如，商品销售之后顾客会拿到一份账单。如果这份账单在15天之内没有付款，那么系统会发送逾期提醒。临时事件“应发送逾期提醒”会定义为在结账日期后的15天。

要定义的临时事件包括：
- 需要的内部输出结果
  - 管理层报表（总结或异常报表）
  - 操作报表（详细业务处理）
  - 综述、状态报表（包括工资单）
- 需要的外部输出结果
  - 结算单、状态报表、账单、备忘录

图3-4 临时事件清单

状态事件

第三种事件是**状态事件**，这个事件是当系统内部触发了需要处理的情况时所引发的事件。状态事件也称为内部事件。例如，一个产品的销售引起了库存记录的调整，若库存数量低于订货点，这时候就有必要重新订货。状态事件可能会命名为“到达订货点”。通常情况下，状态事件是作为外部事件的结果发生的。有时候，它们与临时事件相似，除了这个时间点不能定义之外。

3.3.3 定义事件

定义能影响系统的事件并不简单，但是有一些方法能帮助我们进行分析。

事件/前提条件和响应

有时候区分一个事件与造成这一事件的一系列前提条件是很难的。以一位顾客从一家零售商店中购买衬衫为例（见图3-5）。从顾客的角度来看，购买过程包括一系列的事件。第一个事件是这个顾客想要试穿。然后，这个顾客想要穿一件条纹衬衫。接下来，条纹衬衫正好卖完了。那么，他就决定开车去购物中心。然后他就决定去西尔斯（Sears）百货并且试穿了一件条纹衬衫。试穿完之后，他又决定离开西尔斯百货去沃尔玛再试穿一件条纹衬衫。最后，这个顾客决定在那里买这件衬衫了。分析员必须考虑诸如此类的一系列事件，然后确定直接影响系统的事件。在本例中，系统是不会受到影响的，直到这个顾客在商店里拿着这件衬衫并且说“我想要买这件衬衫”。

图3-5 造成影响系统的一个唯一事件的行为序列

在其他情况下，区分外部事件和系统响应也是不容易的。例如，当顾客购买衬衫时，系统需要信用卡号，然后顾客提供信用卡。提供信用卡这样的行为是一个事件吗？在本例中，它不是事件。这仅仅是处理初始交易时发生的交互行为。

决定发生的到底是事件，还是紧跟在事件之后发生的交互行为，采用的方法是看这是长时间的暂停还是间隔发生。（例如，系统交易在没有中断的情况下能完成吗？）再或者是系统暂停下来等待下一次的交易？这个顾客想要买这件衬衫后，处理过程会持续直到交易完成。交易从一开始就不会有明显的停止。交易完成后，系统会暂时终止，等待下一次交易的开始。早期定义的 EBP 概念就描述了这个情况，在一致的状态下保存系统和数据。

另一方面，当顾客通过商店信用卡账户购买衬衫时会发生独立的事件。当顾客在月底付款时，其处理过程是购买事件的一个交互行为吗？在本例中不是。系统会记录交易过程然后去做其他事。它不会终止所有进程来等待这次的付款。而是在此之后发生一个独立事件，该事件促使系统发给顾客一份账单。（这是个临时事件："该发送月底账单了"。）最后，另外一个外部事件发生了（"顾客支付账单"）。

事件序列：追踪事务处理的生命周期

定义事件时，追踪某一特定的外部实体或参与者而发生的一系列事件是很有用的。在落基山运动用品（RMO）新的 CSMS 系统这个例子中，分析员会思考由于增加一个新顾客引发的所有可能的事务（见图 3-6）。首先，顾客想要一份商品目录或者询问一些商品是否有货，这样会引起数据库中增加顾客名字和新地址。接下来，这个顾客想要下订单。也许他将来想要修改订单，例如纠正衬衫的尺寸或者购买另外一件衬衫。然后，这个顾客可能想要查看订单的现状来确认发货时间。也许他已经搬家了，为了方便将来的目录邮寄，现在想要修改之前登记的地址。最后，这个顾客想要退回一个商品。研究此类过程能帮助我们定义事件。

图 3-6　一位具体顾客引发许多事件的交易序列

技术依赖事件和系统控制

有时，分析员会关心那些对系统来说是很重要的但是与用户或交易无直接关联的事件。这样的事件通常包括设计选项或系统控制。分析过程中，分析员应该暂时忽略这些事件。然而，它们对于后期的设计来说是很重要的。

一些能影响设计的事件包括实际使用物理系统的外部事件，如登录。尽管对系统的最终操作来说是很重要的，但具体的实施细节应该推迟。在这一阶段，分析员应该只聚焦于功能

需求（例如系统需要完成的工作）。功能需求模型不需要说明系统是怎样实施的，因此这个模型应该省略实施细节。

大多数这类事件会包括**系统控制**，系统控制是为保证系统完整性而加入的防范和安全程序。例如，为了系统安全控制需要登录系统。其他控制可以保证数据库的完整性，例如每天备份数据。这两种控制对系统来说都非常重要，所以在设计阶段就会加入系统中。但在分析阶段花时间在这些控制上的结果只能是在需求模型中增加一些用户毫不关心的细节，用户相信系统开发中会考虑这些细节。

一个有助于确定哪个事件应该用于控制的方法是：假定技术是理想的。**理想技术假设**认为：只有在最佳条件下系统才需要做出响应，这样的事件才应该在分析阶段考虑进去（例如，设备不会损坏、处理和存储能力没有限制、用户操作完全遵照系统要求且没有误操作）。通过假定这个技术是理想的，分析员能排除诸如“数据库备份”这类事件，因为可以假定磁盘永远不会损坏。之后，在设计阶段，项目小组把这些控制加进来，因为此时技术假设显然已经不再是理想的了。图 3-7 列出了一些在设计阶段之前都可以忽略的事件。

图 3-7 设计阶段之前可以忽略的事件和功能

3.3.4 使用事件分解技术

总结，用事件分解技术确定用例包括以下步骤：

1. 考虑需要系统做出响应的外部事件，可以使用如图 3-3 所示的清单形式列出。
2. 对于每个外部事件，确定和命名系统要求的用例。
3. 考虑需要系统做出响应的临时事件，可以使用如图 3-4 所示的清单形式列出。
4. 对每个临时事件，确定和命名系统要求的用例，然后建立将会触发用例的时间点。
5. 状态事件是系统需要做出响应的事件，尤其是实时系统，在设备或内部状态改变时会触发用例。
6. 对每个状态事件，确定和命名系统要求的用例，然后定义状态改变。
7. 当事件和用例被定义之后，查看它们在假定技术是理想的状况下是否必要。不要把系统控制中的登录、登出、更改密码和备份或者重置数据库这些事件包括进去，因为这些事件被归为系统控制。

3.4 用例和 CRUD

另外一个用于确认和修改用例的重要技术是 **CRUD 技术**。"CRUD"是创建（Creat）、读取（Read）（或者报告（Report））、更新（Update）与删除（Delete）的首字母组合，它通常会在谈到数据库管理时介绍。分析员一开始会观察系统存储的数据类型，这些被建模为数据实体或域类，将会在第 4 章中描述。第 1 章介绍的 RMO 贸易展览系统中，数据类型包括供应商、合同、产品以及产品图片。在 RMO 的 CSMS 系统中，数据类型包括顾客、销售、库存、促销及运输。为了确认和修改用例，分析员会观察每种类型的数据并且核实已经确定的用例，像是创建数据、读取或报告数据、更新数据以及删除或存档数据。

当 CRUD 技术和用户目标技术交叉校验时，CRUD 技术是最有用的。用户会关注他们的首要目标，且更新或存档数据用例通常会被忽略。CRUD 技术能保证所有的可能性都被确定。有时，数据实体或域类会通过一系列的整合应用被共享。例如，RMO 拥有一个供应链管理应用，这个应用负责管理存货等级和增加产品。RMO 的 CSMS 系统不需要创建或删除产品，但是它需要检察和更新产品信息。通过定义其他应用来明白系统的范围是很重要的，这些应用是负责创建、更新或删除数据的。对于 RMO 顾客数据，图 3-8 展示了一个基于 CRUD 技术的潜在用例的例子。

数据实体 / 域类	CRUD	已检验的用例
顾客	创建	创建顾客账户
	读取 / 报告	查询顾客 生成顾客使用报告
	更新	处理账户调整 更新顾客账户
	删除	更新顾客账户（为了存档）

图 3-8 用 CRUD 技术检验用例

如图 3-8 所示，分析员没有盲目添加用例来创建、读取 / 报告、更新与删除一个顾客的实例。CRUD 技术最好用在已经定义好的用例上，并且作为交叉校验用来核实是否存在用于创建、读取、更新和删除的用例。

CRUD 技术的另外一个用法是总结所有用例和所有数据实体 / 域类来说明用例和数据之间的联系。在图 3-9 中，一些用例和数据实体 / 域类相对应，在用例数据表的单元格中填入 C、R、U、D。例如，用例"创建顾客账户"实际上就是创建顾客数据和账户数据，因此 C 就填入到在这两项交叉单元格中。用例"处理账户调整"读取关于销售的信息、读取关于顾客的信息、更新账户以及做出调整。

用例与实体 / 域类	顾客	账户	销售	调整
创建顾客账户	C	C		
查询顾客	R	R		
生成顾客使用报告	R	R	R	
处理账户调整	R	U	R	C
更新顾客账户	UD（存档）	UD（存档）		

图 3-9 CRUD 表显示的用例和对应的数据实体 / 域类

用 CRUD 技术确认和修改用例包括以下步骤：

1. 确定所有包含在新系统中的数据实体或域类。第 4 章会详细介绍。

2. 对于每种数据来说（数据实体或域类），要核实已确定的用例，包括创建一个新实例、更新现有的实例、读取或报告实例的价值以及删除（存储）实例。

3. 如果一个必需的用例被忽视了，那么需要添加一个新用例然后确定利益相关者。

4. 通过已整合的应用来保证能清晰地指出哪个应用是负责添加和维护数据的，哪个系统仅仅是使用数据的。

3.5 RMO 案例中的用例

RMO 的 CSMS 系统包括各种用例，其中许多用例和刚刚讨论的很像。分析员在开发新系统的过程中已经使用了三种技术来定义、确认以及修改用例。最初的系统（第 2 章中讨论过的）分为四个子系统：销售子系统、订单实施子系统、顾客账户子系统以及市场营销子系统。经过处理，分析员将每个子系统提供的报告合并到第五个子系统——报告子系统。在这个系统中，分析员应该通过子系统组织用例来帮助跟踪子系统负责的每个用例。分析员也要识别那些包含超过一种类型的用户的用例。

随着项目进程的推进，逐步完善用例表很重要。额外的用例将会添加进来，有一些可能需要再斟酌，有一些可能需要合并。用一句话快速描述每个用例的一些细节是很有帮助的。在开发者设计和实施用例时，这样清晰的描述通常会扩充为记录的更多细节（见第 5 章）。图 3-10 中列出了一些简短的用例描述。图 3-11 显示了 RMO 的 CSMS 系统和用户的最初用例列表。我们会发现很多用例都有不止一个用户参与。

用例	清晰的用例描述
创建顾客账户	用户 / 参与者输入新的顾客账户数据，并且系统安排了账户号码，创建了顾客记录且创建了账户记录
查询顾客	用户 / 参与者输入顾客账户号码，同时系统获取和显示顾客与账户数据
处理账户调整	用户 / 参与者输入订单号码，系统获取顾客和订单数据；参与者输入调整量，系统创建调整的事务记录

图 3-10 用例和简要描述

CSMS 销售子系统	
用例	用户 / 参与者
搜索商品	顾客、客服代表、商店销售代表
查看产品评论和排名	顾客、客服代表、商店销售代表
查看配套商品	顾客、客服代表、商店销售代表
加入购物车	顾客
清空购物车	顾客
结算购物车	顾客
加入储备车	顾客
清空储备车	顾客
转换储备车	顾客
创建电话销售	客服代表
创建商店销售	商店的销售代表

图 3-11 CSMS 子系统的用例和用户 / 参与者

CSMS 订单实施子系统	
用例	用户 / 参与者
运输商品	运输部
管理运输人	运输部
创建缺货	运输部
创建退货	运输部、顾客
查询订单现状	运输部、顾客、管理部
跟踪运输	运输部、顾客、市场部
产品排名和评论	顾客
提供建议	顾客
评审建议	管理层
运输商品	运输部
管理运输人	运输部
CSMS 顾客账户子系统	
用例	用户 / 参与者
创建 / 更新顾客账户	顾客、客服代表、商店销售代表
处理账户调整	管理层
发送信息	顾客
浏览信息	顾客
申请朋友连接	顾客
回复连接请求	顾客
发送 / 接收观点	顾客
查看“山地供鹿”积分	顾客
交换“山地供鹿”积分	顾客
CSMS 市场营销子系统	
用例	用户 / 参与者
添加 / 更新产品信息	经营部、市场部
添加 / 更新促销	市场部
添加 / 更新配件包装	经营部
添加 / 更新业务伙伴连接	市场部
CSMS 报告子系统	
用例	用户 / 参与者
生成日常事务总结报告	管理部
生成销售历史报告	管理部、市场部
生成销售趋势报告	市场部
生成顾客使用报告	市场部
生成运输记录报告	管理部、运输部
生成促销效果报告	市场部
生成业务伙伴活动报告	管理部、市场部

图 3-11 （续）

3.6 用例图

有时，创建更生动的图表来说明用例及其如何组织是很有用的。**用例图**是 UML 模型，

它用来说明用例及其与用户之间的关系。回想一下第2章中讲到的，UML是指在系统中使用的图和模型的一系列标准。在第1章中你看到的是用例图的一个例子。这个例子中的符号都是比较简单的。

3.6.1 用例、参与者和符号

大多数用例意味着使用系统的人，即我们之前提到过的用户。在UML图中，这个人称为参与者。参与者总是处于系统自动化边界的外部，但是又可能是系统手工部分的成员。有时，用例的参与者不是人，而是另一个系统或者接收系统服务的设备。

图3-12说明了一个用例图的基础部分。我们用一个简单的人物线条图来代表参与者。这个人物线条图有一个名字，它描述了人物扮演的角色。椭圆形代表用例本身，而在椭圆形内部就是用例的名称。参与者与用例之间的连线说明参与者包含在用例中。最后，**自动化边界**定义了应用的计算机部分与操作者之间边界，它会以长方形的形式包含在用例图中。参与者与用例的交流会贯穿整个自动化边界。图3-12中的例子就说明了作为运输员工的参与者与运输商品这个用例。

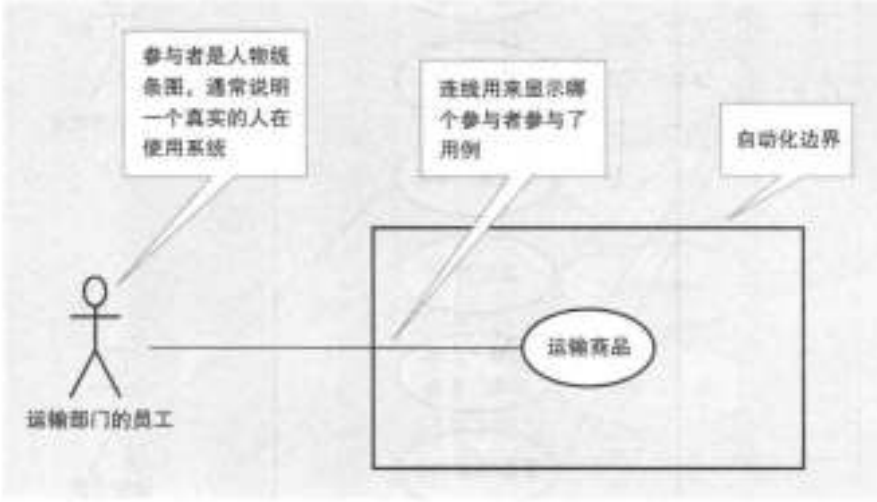


图3-12 有一个参与者的简单用例

用例图的例子

图3-13列出了一个关于RMO CSMS子系统的更完整用例图，这里选取的是顾客账户子系统。图3-11中的信息是为了在视觉上突出单个子系统的用例与参与者而新塑造单个用例图。这个图标在评审子系统用例与参与者的会议上是很有用的。这个例子中，顾客、客服代表与商店销售代表都可以直接进入系统。通过关系连线，我们可以知道每个参与者都能使用“创建/更新顾客账户”这个用例。顾客在网上查询时可能会用到，客服代表在与顾客进行电话沟通时也可能会用到，商店销售代表在商店和顾客接触时可能会用到。只有管理层的部分人员才能进行账户调整。其他包含的用例只针对顾客。

组织用例图有很多种方法，以便用户、利益相关者及项目团队成员之间进行交流。一种方法是列出所有特定参与者参与的用例（例如，从用户的角度来看）。这种方法通常会在需求定义阶段使用，因为系统分析员很可能会和一个特定参与者一起工作，同时还要定义出所有用户使用系统执行的功能。图3-14就阐述了这个观点，说明了在销售子系统中参与的所

有用例。图 3-15 列出了销售子系统中关于客服代表与商店销售代表的用例。分析员能扩展这种方法来列出所有属于特定部分的用例，不管是不是在这个子系统中，或者只列出对特定利益相关者来说很重要的所有用例。

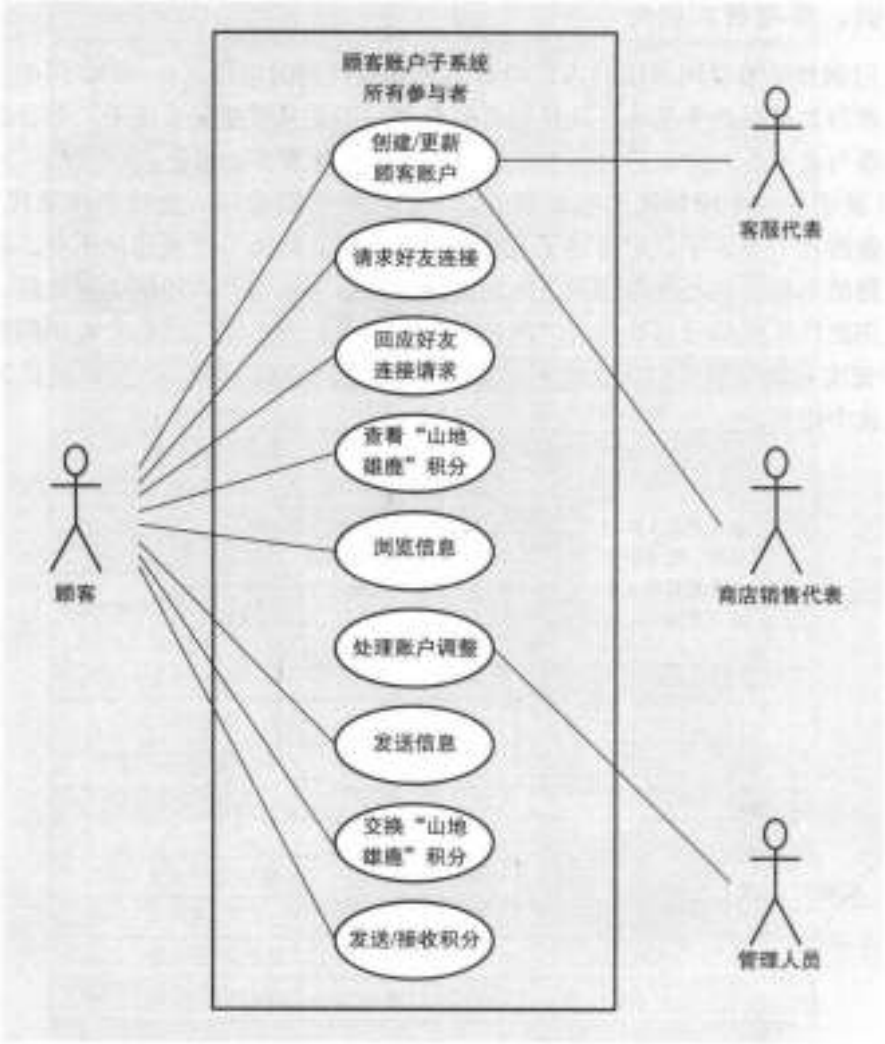


图 3-13　RMO 顾客账户子系统的用例图，显示所有参与者

包含关系

通常在用例图的开发过程中，经常会发生一个用例使用另一个用例的相关服务。例如，在销售子系统中使用了图 3-14 中列出的用例，顾客会搜索商品、查看产品评论和排名以及在加入购物车之前查看配套商品。然而，当加入购物车之后，顾客也会继续搜索商品、查看产品评论与配套商品。因此，一个用例可以使用或“包含”另外一个用例。图 3-16 中的用例图强调了这些用例的形态。加入购物车也包括搜索商品、查看产品评论和排名及查看配套商品。因此，顾客一开始就能看到评论并且实施加入购物车这个用例。这些用例之间的关系是由带有箭头的虚连线表示的，这些虚连线说明这个用例是被包含的。这种关系称为加入购物车包含搜索商品。有时，这种关系也叫作 << **包含** >> **关系**或者 << 使用 >> 关系。记住，“包含”这个词封闭在 << >> 中，这也是 UML 图中构造型的方法。它意味着一个用例与其

他用例之间的关系是构造型风格的 << 包含 >> 关系。

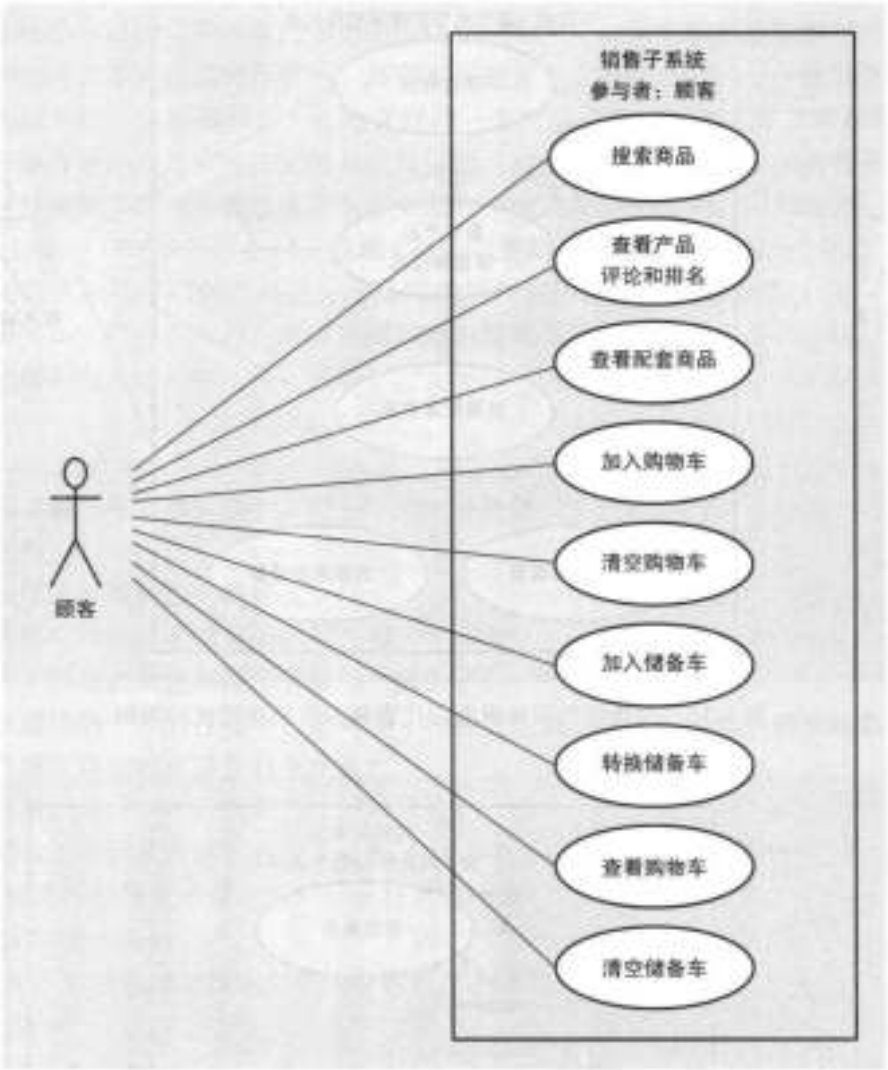


图 3-14 包含销售子系统所有顾客参与者的用例

3.6.2 开发用例图

分析员创建一系列的用例图来与用户、利益相关者、管理人员及团队成员进行交流。开发用例图的步骤是：

1. 确定所有的利益相关者和用户，这些人都能通过用例获取利益。

2. 决定每个利益相关者或用户需要在评审用例过程中做的事。通常情况下需要为每个子系统、每种类型的用户、具有 << 包含 >> 关系的用例以及特定利益相关者感兴趣的用例而开发用例图。

3. 对于每种潜在的交互需求，要选择用例与参与者来展示和画出用例图。有许多软件能用来画用例图。

4. 仔细命名每个用例图，然后记录这个用例图在什么时候以及怎样用于查看利益相关者和用户的用例。

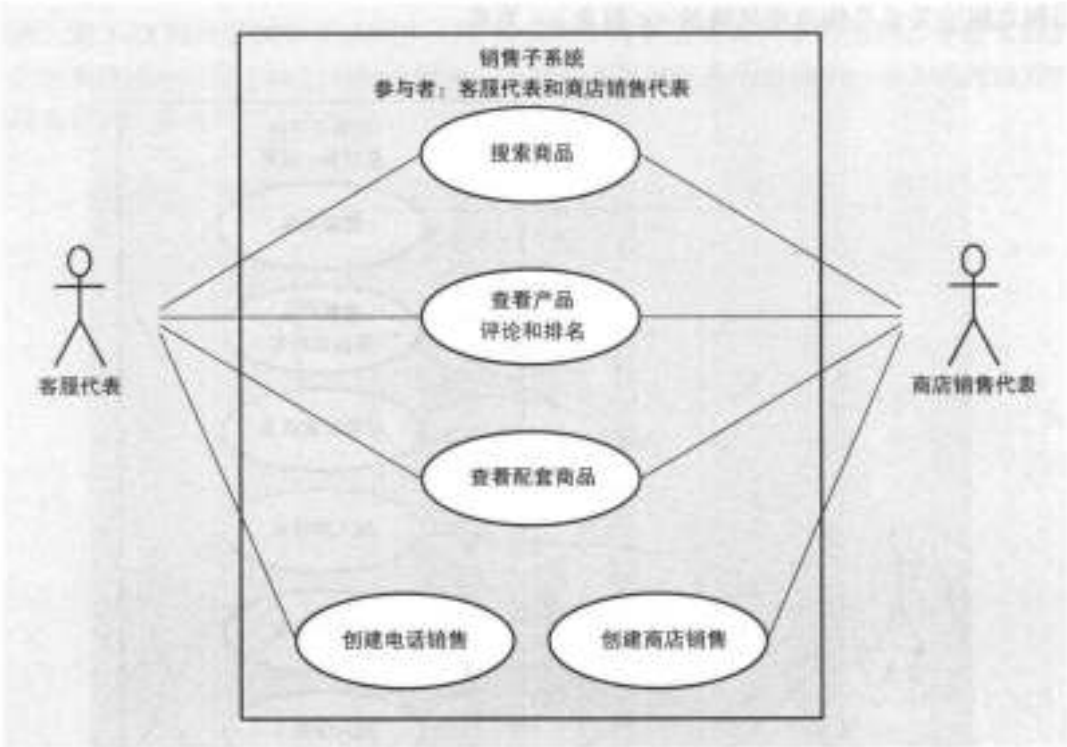


图 3-15 包含销售子系统客服代表和商店销售代表的用例

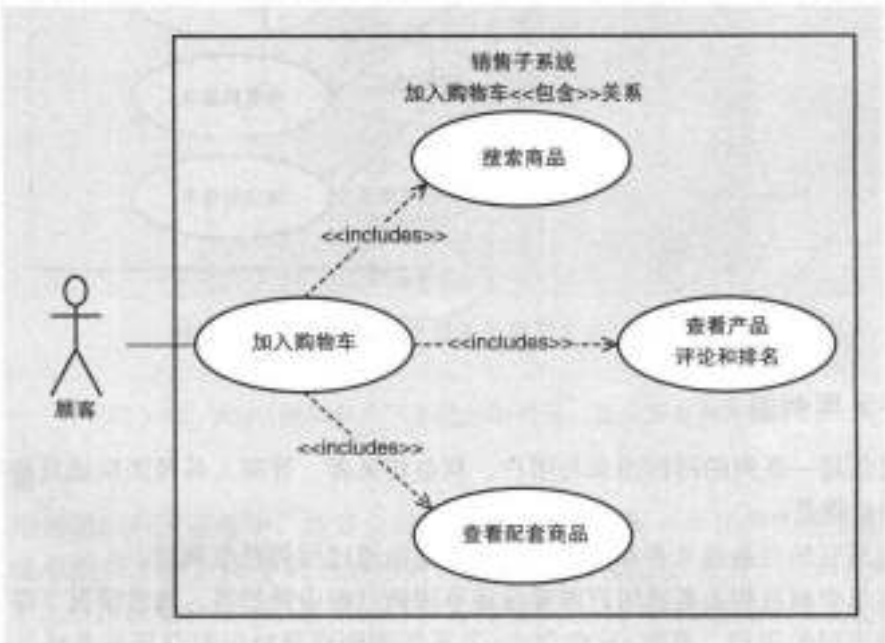


图 3-16 加入购物车 << 包含 >> 关系的用例图

本章小结

本章是讲述系统功能需求建模技术的三个章节中的第一章。建模早期的核心步骤是定义和列出用例，这个用例定义了系统的功能需求。用例能通过使用用户目标技术与事件分解

技术来确定。用户目标技术以确定系统最终用户的类型为开端，其中系统最终用户称为参与者。然后，会要求用户列出特定的用户目标，即他们在使用系统时需要用来支持工作的功能。事件分解技术以确定需要系统做出相应的事件为开端。这个事件是能描述的，发生在特定的时间和地点，并且是值得保留的。外部事件是发生在系统外部的——通常是由与系统交互的那个人触发的。临时事件发生在定义好的一个时间点上，例如工作日的最后或者月末。状态或内部事件是以内部系统变化为基础发生的。对于每个事件来说，用例都是确定和命名好的。事件分解技术能帮助确保在基本业务流程中确定每一个用例。用例的确认与细化是通过 CRUD 技术，"CRUD" 是创建、读取 / 报告、更新与删除的首字母的组合。

分析员定义的每个用例会通过一个简明的用例描述和确定参与者来记录。UML 用例图用于记录用例和它们的参与者。许多不同的用例图是基于各种利益相关者、用户及团队成员需要评审用例的需求绘制的。

复习题

1. 系统分析有哪六种活动？哪个活动在本章一开始就提到？
2. 什么是用例？
3. 用来确定用例的两种技术是什么？
4. 描述使用用户目标技术确定用例的过程。
5. 请举一些不同功能角色和操作等级用户的例子。
6. 在学生办理注册手续的过程中，请列举一些相关用例的名字。用动名词来命名。
7. 询问用户特定目标的首要目的是什么？
8. 有多少类型的用户使用系统时有相同的目标？
9. 描述使用事件分解技术确定用例的过程。
10. 为什么说事件分解技术要比用户目标技术更全面？
11. 什么是基本业务流程？
12. 阐述事件分解技术是怎样在分析的合理层次上确定用例的。
13. 什么是事件？
14. 事件有哪三种类型？
15. 定义一个外部事件，然后给出一个能应用到支票账户系统中的例子。
16. 定义一个临时事件，然后给出一个能应用到支票账户系统中的例子。
17. 什么是系统控制？为什么它们不属于用户功能需求这个部分中？
18. 什么是理想技术的假设？
19. 在一个典型的信息系统中请列举三个包括系统控制的事件例子。你所列举的事件应不是理想技术假设前提下的用例。
20. 组成 CRUD 组合的四个操作是什么？
21. 使用 CRUD 技术的主要目的是什么？
22. 什么是简短的用例描述？
23. 什么是 UML ？
24. UML 用例图的作用是什么？
25. 在 UML 中，参与者的另一个名称是什么？在用例图中它代表的是什么？
26. 用例图中的自动化边界是什么？它代表的是什么？

27. 用例图中用例能与多少参与者相关？
28. 在和最终用户评审用例时，为什么要系统分析员画出很多不同的用例图？
29. 两个用例之间的 << 包含 >> 关系是什么？

问题和练习

1. 查看图 3-3 中的外部事件列表，然后思考大学注册系统。列表中每个类型的事件对应的例子是什么？通过使用命名外部事件的规则为每个事件命名。
2. 查看图 3-4 中的临时事件列表。学生成绩报告是内部输出还是外部输出？班级教师的列表是内部输出还是外部输出？对于大学注册系统来说，还有什么其他的内部和外部输出？使用命名临时事件的规则，你要怎么命名这些触发了输出的事件？
3. 思考以下一系列的动作，这是由一个在银行的顾客做的。哪个动作是分析员应该为银行的事务处理系统定义的事件？
   （1）Kevin 在他生日时收到了祖母给他的支票。
   （2）Kevin 想要一辆汽车。
   （3）Kevin 决定存钱。
   （4）Kevin 去银行。
   （5）Kevin 在排队等候。
   （6）Kevin 在他的存款账户里存款。
   （7）Kevin 拿到了存款收据。
   （8）Kevin 要一本关于自动贷款的手册。
4. 思考理想技术假设，它说明了只有系统在理想条件下需要做出响应时用例才应该包含在分析活动中。在这种假设下，RMO 的 CSMS 系统中列出的一些用例是否可以去掉？阐述理由。为什么像登录系统和返回数据库这样的用例只在不定理想的状况下才需要？
5. 访问一些汽车制造商的网站，像 Honda，BMW，Toyota 和 Acura。许多这种网站都有用例，而且这种用例命名为生产汽车与定价。作为一个潜在顾客，你可以选择一辆汽车模型，选择它的外观和性能，并获取这辆车的价格和特征列表。写一份针对这个用例的清晰的用例描述（见图 3-10）。
6. 再一次访问一个汽车制造商的网站，假设你自己是一个潜在顾客，然后确定所有与你目标一致的用例。
7. 与图书管理员一起开一个会。开会过程中，要求图书管理员描述在图书馆中遇到的图书借阅系统需要响应的情景。列出这些外部事件。现在询问需要系统提供状态、注意事项、报告或其他输出的时间点或者截止期限。列出这些临时事件。图书管理员以这样的方式描述这个系统很常见吗？列出每个事件并且命名结果用例。
8. 再一次思考图书馆案例，询问学生在使用图书馆系统时的目标。同时也要询问一下图书馆的工作人员在使用系统时的目标。作为用例来命名这些目标，并且讨论学生用户是否应该与工作人员拥有不同的目标。
9. 访问一个餐厅或者大学食品服务部门，与服务者交谈（或者与一个从事食品服务的朋友交谈）。询问关于外部事件和临时事件的问题，就像第 7 题中那样做。在餐厅中的订单处理系统中有什么事件和结果用例？
10. 查看你所在大学的选课过程，然后与从事通知、登记和你所在专业的工作人员进行交谈。

思考贯穿整个学期的一系列活动。学生触发的事件有哪些？你自己的院系又触发了什么事件？导致信息流入学生的临时事件是什么？导致信息流入教师或专业的临时事件是什么？列出所有包含在系统中的这些事件和结果用例。

11. 图 3-11 是关于 RMO 的 CSMS 系统中的订单实施子系统。画一份用例图，显示参与者和所有用例。如果可以，就用像 Microsoft Visio 这样的画图软件来画图。
12. 再一次回到订单实施子系统，画一份用例图用于与运输部门开会讨论关于系统需求的用例。如果可以，就用像 Microsoft Visio 这样的画图软件来画图。
13. 图 3-11 是关于 RMO 的 CSMS 系统中的市场营销子系统。画一份用例图，显示所有参与者和用例。如果可以，就用像 Microsoft Visio 这样的画图软件来画图。
14. 图 3-11 是关于 RMO 的 CSMS 系统中的报告子系统。这些报告是通过询问用户关于临时事件以及需要系统提供有价值报告的重要时间点而定义的。现在大多数的实时系统中，要求参与者提供报告或其他输出。回忆一下，参与者是系统的一部分，是属于手工、不自动化的那部分。因此，这是“系统”能在某一个时间点负责提供输出的一个方式。在未来，更多的输出会以自动化的方式提供。如图 3-11 所示，画一份用例图，显示参与者和所有用例。如果可以，就用像 Microsoft Visio 这样的画图软件来画图。

扩展资源

Classic and more recent texts include:

Craig Larman, *Applying UML and Patterns* (3rd ed.). Prentice-Hall, 2005.

Grady Booch, Ivar Jacobson, and James Rumbaugh, *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

Ed Yourdon, *Modern Structured Analysis*. Prentice Hall, 1989.

Stephen McMenamin and John Palmer, *Essential Systems Analysis*. Prentice Hall, 1984.

域 建 模

学习目标

阅读本章后，你应该具备的能力：

- 解释如何用问题域中“事物”的概念来定义需求。
- 识别和分析系统中需要的数据实体和域类。
- 阅读、解释并创建实体－联系图。
- 阅读、解释并创建域模型类图。
- 理解 RMO 商店的销售和市场营销系统中的域模型类图。

开篇案例 Waiters on Call 餐饮送货系统（第二部分）

记得开发 Waiters on Call 系统时是与 Sam Wells 一起工作，讨论关于这个订餐送货系统的需求。Sue 和 Tom 想要的新系统是自动化的，而且能增加特色业务，包括提供顾客订单，以及提供由很多当地餐厅准备饭菜的送餐上门服务。Sam 做了很多工作来确定这个送餐系统需要的用例，这给 Bickford 兄弟留下了深刻印象。在确认用例的同时，Sam 还在坚持记录所有业务条款和 Bickford 兄弟用来描述其运营场景的概念。他也在跟进关于他们所回答的每天工作种类的问题。

Sam 说：“以你们告诉我的内容为基础，我假定你们需要用系统来存储以下事物种类的信息——餐厅、菜单项目、顾客及订单。我认为你们还需要存储以下事物种类的信息——司机、地址、路线及订单付款。”

Bickford 兄弟欣然接受了这个假定，同时还增加了这样一条信息：了解到达餐厅的路线及到达顾客的住所需要的时间也是很重要的。他们希望司机被安排的路线是根据两个地点之间的距离决定的。

Sam 同意了这个说法：“是的，我们需要决定系统怎么安排这些事情。你们能告诉我当司机接到几个餐厅的订单后什么时候出发吗？你们能告诉我一份订单中通常包括多少项目吗？你们能记录下接到订单的时间和派送时间吗？你们需要计划路线来先派送热菜吗？”

Bickford 兄弟更加确信他们已经找到了一位了解他们业务需求的分析员了。

4.1 引言

第 3 章的内容集中在确定用例来定义信息系统的功能需求。本章我们会重点讨论另外一个定义需求的核心概念：系统用户问题域中的事物。在讨论用例时，你一开始就学习到了数据实体或域类。在研究数据库管理系统时你也已经学习了这些，因为它们定义了用在数据库管理系统中的表的来源。几乎所有的系统开发方法都包括定义与建立数据实体或域类，这在定义功能需求这个分析活动中是重要的任务。

4.2 问题域中的“事物”

域类或数据实体是最终用户在工作时需要处理的，例如，产品、订单、发票及顾客。这些通常都被称为系统问题域中的“事物”。**问题域**是指包括在新系统范围中的用户业务的特定区域。新系统可处理和保存这些“事物”。例如，一些信息系统需要存储关于顾客和产品的信息，因此分析员确定有关这两个事物的信息是很重要的。通常，事物与那些和系统交互的人或利益相关者有联系。例如，顾客是下订单的，但是系统需要存储的是顾客的信息，因此顾客在这个问题域中也是一个事物。然而，事物有时候与人又有区别。例如，系统可能需要存储关于产品、订单及仓库的信息，但是这些就不是人。

有许多技术能定义问题域中的重要事物。其中两种技术会在本章中介绍：头脑风暴法和名词技术。

4.2.1 头脑风暴法

就用例来说，分析员会要求用户讨论他们惯常处理的事物的类型。分析员可以询问几种类型的事物来帮助用户定义用例。许多事物是有形的，因此很容易定义，但是有些事物是无形的。不同类型的事物对不同用户的重要性是不同的，因此把所有的用户包涵进来以帮助定义问题域中的事物是很重要的。**头脑风暴法**对于和用户一起工作来定义问题域中的事物是很有用的。

图 4-1 显示了一些要考虑的事物类型。有形事物通常是最明显的，比如一架飞机、一本书或者一辆车。在 RMO 的案例中，仓库里的产品和运输中的货物是重要的有形事物。信息系统中的另一种常见的事物类型是由人扮演的，就像一名员工、一名顾客、一名医生或者一名病人。顾客这个角色显然是 RMO 案例中很重要的事物。问题域中的很多事物都能符合多种类型。例如，一辆车是设备也是有形事物。不管怎么样，重要的是定义问题域中潜在的事物。

其他类型的事物包括组织单元，就像是一个部门、车间或工作小组。相似的还有一个地点或地址，像是仓库、店铺或分公司，这些也可能是系统中重要的事物。最后，一个事件或交互的信息也可以是事物，例如关于订单、服务电话、合同或者飞机航班的信息。销售、运输和退货也是 RMO 这个案例中重要的事件。有时，这些事件会被认为是事物之间的联系。例如，销售是顾客和库存商品之间的联系。最初分析员可能会简单列出所有事物，然后根据需求通过不同的分析与设计方法来做出调整。

分析员通过思考每个用例、与用户交谈及问问题来定义这些事物的类型。例如，对于每个用例来说，系统需要了解存储的信息将对什么类型的事物产生影响。图 4-1 列出的事物类型可以通过头脑风暴法来包含系统中每个事物的类型。当一个顾客想要从网站上购物时，系统需要存储关于顾客、选中的产品、销售细节的信息，例如日期、付款商品及被运输商品的地址。对于那个用例，分析员能定义有形事物（选中的商品）、扮演的角色（顾客）、事件或活动（销售）、地点 / 地址（仓库）及组织单元（运输）。

以下是使用头脑风暴法的步骤：

1. 定义用户和一系列用例。

2. 当实施用例时，和用户一起头脑风暴来定义包含在内的事物，即需要从系统中获取信息的事物。

3. 使用事物类型（目录）来系统地询问关于潜在事物的问题，例如，你存储的信息中有关于有形事物的吗？是否包括地址？你需要保存的信息有由人扮演的角色吗？

图 4-1 用于头脑风暴法的事物类型

4. 坚持与各种类型的用户和利益相关者工作，以此来扩大头脑风暴列表。

5. 合并所有的结果，排除所有的重复内容并编制一个初步列表。

4.2.2 名词技术

另外一种定义问题域中事物的有用程序称为**名词技术**。回想一下名词是指人、地点或事物。因此，确认名词能帮助你确定系统需要存储的信息。当谈论到系统时，一开始要做的是列出所有用户提到的名词。用来描述事件、用例与参与者的名词就是潜在事物。接下来，将关于现有系统信息的名词以及与利益相关者讨论系统问题域时提到的名词都添加到列表中。这份名词的列表会很长，因此需要不断改进。名词技术与头脑风暴法的区别是分析员不用思考很多，也不用与用户交谈很多，就可以列出所有的名词。基于利益相关者与用户的商议，仅在后期列表才会被修改并简化。

使用名词技术的步骤是：

1. 运用用例、参与者和其他系统信息（包括输入/输出）来确定所有名词。对于 RMO 的 CSMS 系统来说，名词包括：顾客、产品目录、销售、证明、交易、运送、银行、改变请求、总结报告、管理、业务报告、会计、延期订单、延期通知单、退货、退货确认、确认反馈、潜在顾客、市场营销、顾客账户、促销手段、收费调整、销售细节、销售规划以及顾客活动的报告。

2. 运用现有系统、当前的程序、报告或表格中的其他信息，添加所需信息的条款或目录。对于 RMO 的 CSMS 系统来说，这些可能包括更详细的信息，例如价格、尺寸、颜色、风格、季节、库存量、支付方式以及运送地址。其中的一些条款可能是附加的事物，另外一些可能是关于你已经确定的事物的一些具体信息（称为属性）。修改列表然后记录要探索的假设和重要问题。

3. 这份名词列表建立完成后，你需要进行修改。针对每个名词问以下这些问题能帮助你决定是否应该保留它：

- 这是系统需要了解的独特的事物吗？
- 这个事物是在我处理的系统范围之内吗？
- 系统是否需要记住两个以上的这种条款？

问以下这些问题能帮助你决定是否应该排除它：

- 这真的是我已经确认的其他事物的同义词吗？
- 这真的是从你已经定义的其他信息中产生的系统输出吗？
- 这真的只是我已经定义的导致记录其他信息的输入吗？

问以下这些问题能帮助你决定是否应该研究它：

- 这像是我已经确定的事物信息的更多特殊部分（称为属性）吗？
- 如果属性变了我会需要这些事物吗？

4. 创建所有已确认名词的主要列表，然后注意每个名词是否要保留、排除或是研究。

5. 与用户、利益相关者和团队成员一起评审，然后修改问题域中的事物列表。

图 4-2 列出了 RMO 的 CSMS 系统中的一些名词，并且每一个都有相关记录。和头脑风暴法相比，借助该表开发的这份初步的列表只是一个开端。还需要做更多的工作来修改列表并且在列表中定义更多关于每个条款的信息。

已确定的名词	作为事物存储的名词的相关记录
会计	我们了解却有哪些人，不需要存储
延期订单	特殊种类的订单？或者是订单状态？需要研究
延期订单的信息	其他信息产生的输出
银行	只有一个，不需要存储
目录	是的，需要记住，涉及不同的季节和年份，需要保留
目录活动报告	其他信息产生的输出，不需要存储
目录细节	与目录一样？或者是与目录中的产品条款一样？需要研究
改变请求	引起订单记录变更的输入
收费调整	引起交易的输入
颜色	关于产品条款的部分信息
证明	其他信息产生的输出，不需要存储
信用卡信息	订单的一部分？或者是顾客信息的一部分？需要研究
顾客	是的，包含许多需要的细节的关键事物，需要保留
顾客账户	如果包含 RMO 支付计划那么可能就需要，要研究
确认反馈	运送信息产生的输出，不存储
库存量	产品条款的一部分信息，需要研究
管理	我们了解都有哪些人，不需要存储
市场营销	我们了解都有哪些人，不需要存储
销售规划	我们了解都有哪些人，不需要存储

图 4-2 基于 RMO 的名词的部分事物列表

4.2.3 事物的属性

名词技术包括列出的所有出现在关于需求的讨论或文档中的名词。就像之前讨论的那样，许多名词实际上就是属性。大多数系统会存储且使用的是每个事物的一些具体信息，就像图 4-2 中的一些名词。这些特定的信息被称为**属性**。例如，顾客会有姓名、电话号码、信贷限额等等。每个细节都是一个属性。分析员需要明确每个系统需要存储的事物属性。一个属性可以用来唯一标识某事物，如雇员的社会保障号或某次购物的订单号。能唯一标识事物的属性被称为**标识符**或**关键字**。有时，标识符是已经存在的（如社会保障号、车牌或产品 ID）。有时，系统需要分配一个具体的标识符（如发票号、交易号）。

系统可能需要记录很多相似的属性。例如，一个顾客有好几个名字——首名、中间名和姓，还可能有一个绰号。**复合属性**是指包括了许多相关属性的属性。因此分析员可以选择一个复合属性来代表所有的名字，也许可以将这个复合属性命名为“顾客全名”。顾客也可以有很多电话——住宅电话、办公室电话、传真电话和移动电话。分析员可能会先描述最重要的属性，然后再不断扩展列表，因此属性列表可能会很长。图 4-3 所示为一个顾客属性及其相应的取值。

所有顾客都具有如下属性	每个顾客的每个属性都有一个值		
顾客 ID	101	102	103
名	John	Mary	Bill
姓	Smith	Jones	Casper
住宅电话	555-9182	423-1298	874-1297
单位电话	555-3425	423-3419	874-8546

图 4-3　属性和值

4.2.4　事物间的关系

对列表中的事物进行记录和改进后，分析员需要进一步研究和记录其他的信息。事物间的各种关系对系统非常重要。**关系**（association）是指某些事物间自然发生的关联，如顾客下订单或雇员在某一部门工作（见图 4-4）。“被下达”和“工作于”就是两个自然的发生在特定事物间的关系。信息系统需要存储雇员和部门的信息，但同样重要的是，系统也需要存储某些关系的信息，例如，John 在财务部工作，Mary 在市场部工作。同样，存储由 John Smith 发出的订购衬衫的 1043 号订单也是非常重要的。在数据库管理中，经常用**联系**（relationship）这一术语来替代关系中，而关系是在建立 UML 图时使用的术语。我们会在本书中使用关系是因为我们强调 UML 图及其术语。

图 4-4　事物之间自然出现的关系

事物间的关系是两方面的。例如，顾客下订单描述的是一方面的关系。类似地，订单由顾客下达描述的是另一方面的关系。理解关系的双向性是很重要的，因为有时候系统从一个方面记录关系比从另一个方面记录关系重要得多。例如，RMO 商店肯定需要知道顾客订购了什么商品，这样才能准备发货。但是，公司要知道所有订购了某一特殊商品的顾客名单这一需求起初可能并不明显。如果公司要给所有订购了残次品或需要收回商品的顾客发通知，这该怎么办呢？知道这个信息是十分重要的，但是操作层的用户可能无法马上认识到这一点。

根据每件事物的关联数目来理解每种关系的本质是非常重要的。例如，一个顾客可能下了许多不同的订单，但是一张订单只能被一个顾客下达。在数据库管理中，发生的关联数目被称为关系的**基数**。基数可以是一对一或一对多的。在面向对象的方法中，术语**重数**常常表示关联的数目，并且在 UML 模型中会被使用。重数是针对于关系的每一个方面提出的。图 4-5 所示为有关订单基数 / 重数的例子。

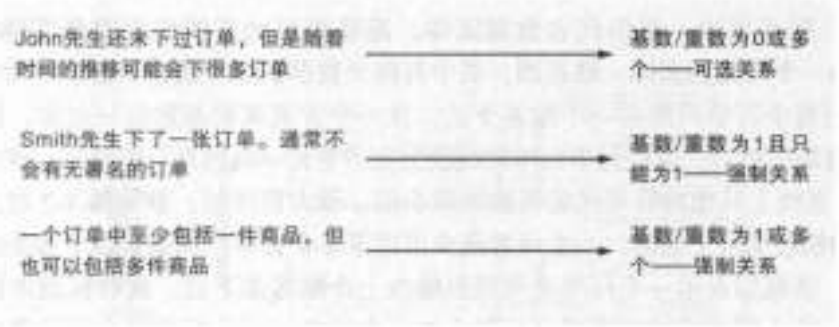


图 4-5 关系的重数 / 基数

有时候重要的不仅是描述基数，还要描述基数可能的取值范围（即基数的最小值和最大值）。例如，某个顾客可能从来没有下过订单。在这种情况下关系数为 0。或者该顾客下过一次订单，此时存在一个关系，最后顾客可能下了两张、三张甚至更多的订单。因此，顾客下订单这个关系的范围是 0、1 或多个，通常记为 0 或多个。0 是基数的最小值，“多个”是基数的最大值，被称为**基数限制**。

在某些情况下，至少需要一个关系（一个和可选关系相反的强制关系）。例如，只有当某顾客下了订单时，系统才开始记录该顾客的信息。因此，基数限制可解释为顾客下了一个或多个订单。

一对一的关系也可以改进成包括最小值和最大值的基数。例如，一个订单是由一个顾客下达的，如果没有顾客也不可能有订单。因此，一是最小的基数值，这就是强制关系。由于每张订单不可能对应多个顾客，因此一也是最大的基数值。有时，这种关系可以解释为一个订单必须且只能由一个顾客来下达。

这里描述的关系是两种不同类型事物间的关系——如顾客和订单。这种关系称为**二元关系**。有时关系是同一类型的两件不同事物之间的关系。例如，婚姻这个关系就是两个人之间的关系。这种类型的关系称为**一元关系**（有时也称为回归关系）。一元关系的另一个例子是组织体系，在这个体系中，一个单位要向另一个单位报告，如包装部门向发货部门报告，发货部门再向调度部门报告，调度部门再向市场部门报告。

关系还可以存在于三种不同类型的事物之间，这种关系称为**三元关系**。甚至还可以存在于多种不同类型的事物之间，此时这种关系称为 **n 元关系**。例如，某一张订单可能和某个顾客、某个销售代表之间有关联，这就是三元关系。

存储关系的信息和存储事物的信息同样重要。记录每个顾客的姓名、地址信息虽然很重要，但记录该顾客订购了哪些商品也同样重要（也许可能更重要）。

4.3 实体 - 联系图

传统的系统开发方法都把重点集中在新系统的数据存储需求上，并且使用术语**数据实体**

来表示系统需要存储信息这个事物。数据存储需求包括数据实体、数据实体的属性及它们之间的联系（在面向对象方法中称为“关系”）。传统分析员和数据库分析员经常使用的模型称为**实体－联系图**（ERD）。实体－联系图不是 UML 图，但是它经常被使用而且它与后面章节中提到的 UML 域模型类图很相似。

实体－联系图中符号的实例

在实体－联系图中，矩形代表数据实体，连接矩形的直线代表数据实体之间的联系。图 4-6 所示为一个简化的实体－联系图，图中有两个数据实体：顾客和订单。每个顾客可以下多个订单，但每个订单只能由一个顾客下达。从一个方面来看基数是一对多，而从另一个方面来看基数则是一对一。连接订单实体的线段上有一个像“乌鸦爪”的符号，该符号表示“多个订单”。关系线上其他的符号代表基数的最小值、最大值限制。参见图 4-7 对关系符号的解释。图 4-6 的模型实际上表示，一个顾客最少可以下 0 个订单，最多可以下多个订单。从另一个方面来看，该模型表示一个订单必须且只能由一个顾客来下达。这种标记方法表示了精确的系统细节。这个限制反映了管理部门制定的业务策略，而分析员则必须发现这些策略。分析员不能随意决定两个顾客不可以共享一个订单，但是管理部门却可以制定这样的策略。

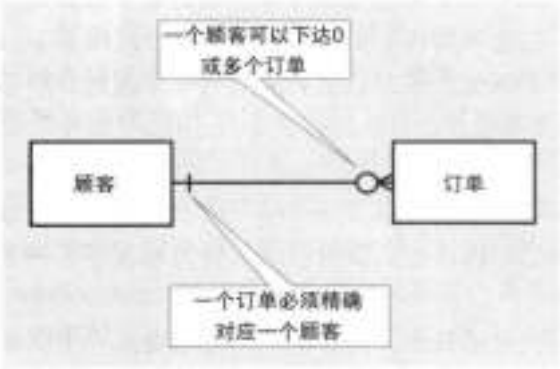


图 4-6　一个简单的实体－联系图

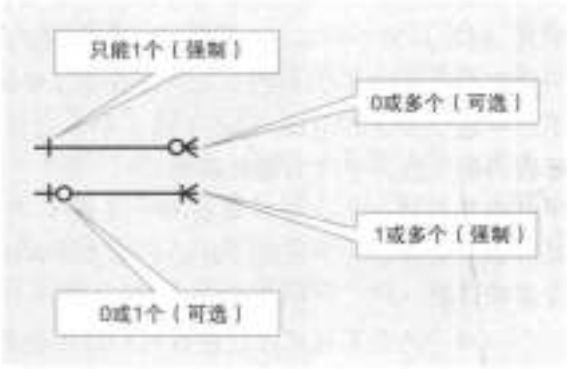


图 4-7　ERD 关系的基数符号

图 4-8 所示为一个扩展的模型，其中包括了订单中的具体商品（一个订单中包含一个或多个商品）。每个订单最少有一个商品，最多可以有很多个商品（不存在一个商品都没有的订单）。例如，一个订单可能包含了一件衬衫、一双皮鞋和一条皮带，每件商品都和该订单关联。这个例子也列出了每个数据实体的属性：每个顾客都有一个顾客 ID、姓名、地址和几个电话号码。每个订单有一个订单号以及订货日期等。订单中的每件商品有商品 ID、数量和价格。每个数据实体的属性都列在其名字的下面，而关键标识符属性列在第一列，通常后面会加上 PK 以表示是主键。

图 4-8　一个扩展的显示属性的 ERD

图 4-9 所示为实际数据在事务处理中呈现的形式。顾客 John 下了两个订单。第一个是在 2 月 4 日下的，订了两件衬衫和一条皮带。3 月 29 日他下了第二个订单，包括一双靴子和两双拖鞋。顾客 Mary 还没有下过订单。记住，一个顾客可以下 0 到多个订单。因此 Mary 和任何订单都没有关系。最后，Sara 在 3 月 30 日订了三双凉鞋。图 4-9 所示的这种情况有时会被称为语义网。**语义网**显示的是属于一个类或数据实体的具体对象及它们之间的关联。语义网对思考和确定实体－联系图中的实体与联系以及类图中的类与关系是很有用的（接下来会讨论）。

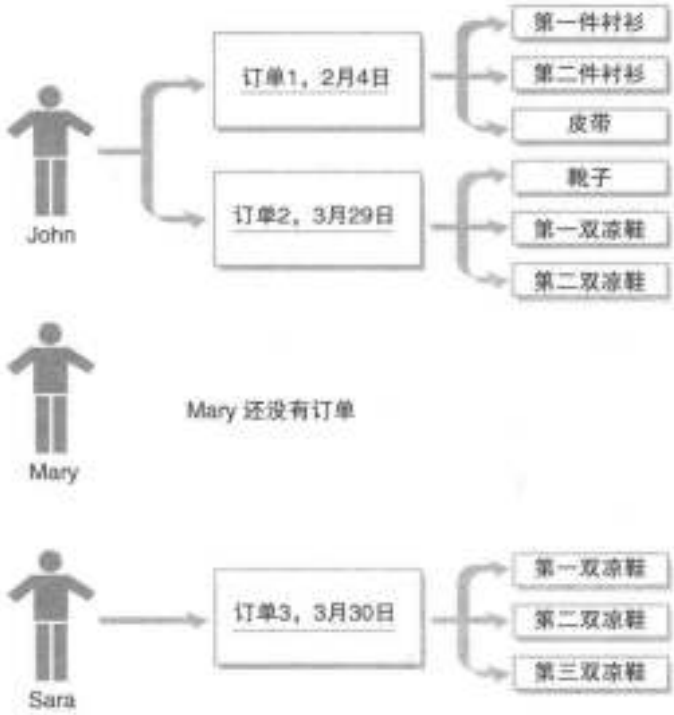


图 4-9 与扩展的 ERD 保持一致的顾客、订单和订单项的语义网

图 4-10 所示的是另一个例子，这是一个银行的实体－联系图，这个银行有很多支行。每个支行有一个或多个账户。每个顾客都有一个账户，这就会产生一笔或多笔交易。在银行这个例子中还有其他几个重要问题要考虑。首先，没有“银行”这一数据实体。因为实体－联系图所显示的数据存储需求都是这家银行的，且只有一家银行（即总行）。因此，在模型中没有必要包含银行这一项。这是应用到实体－联系图中的一个常用规则。如果这个系统是用于国家银行监管机构，那么银行将会成为一个重要的数据实体，因为在国家银行监管机构的管辖范围内有很多银行。

再一次观察基数。注意一个顾客必须至少有一个账户。这个基本原理用在这里就是银行不会添加一个顾客直到这个顾客添加一个账户。同时还要注意的是支行可以有 0 个账户。一个支行可能在其还没有开门营业前就被添加进来，因此支行没有账户是可能的。此外，也可能会有一些支行没有任何账户，例如大学或机场里面的服务柜台。一个基本的规则是开一个新的账户需要顾客的初始存款，这就是一笔交易。认清关于基数、重数以及最小和最大的基数限制的问题是非常重要的，这都需要与系统利益相关者进行商讨和评审。

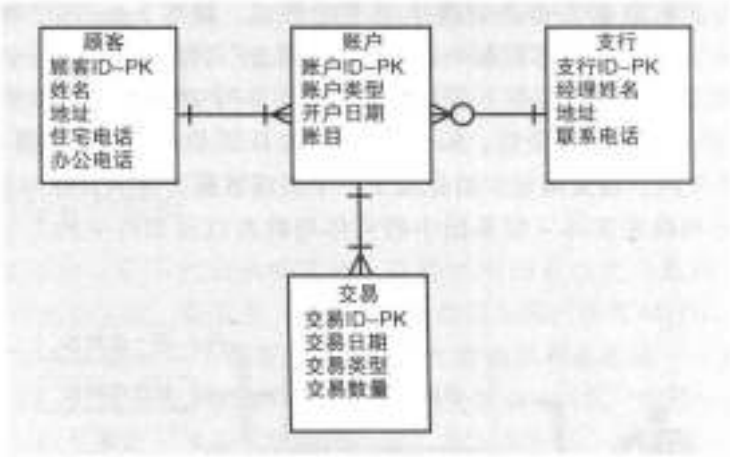


图 4-10 一个有许多分支的银行的 ERD

4.4 域模型类图

当前许多系统开发方法使用类而不是数据实体，同时在 UML 的基础上使用概念和符号来建立问题域中事物的模型。这些概念来自于面向对象的方法。**类**用于描述对象集合的一个目录或分类。每个对象属于一个类。因此，学生 Mary、Joe 和 Maria 属于学生这个类。描述问题域中事物的类被称为**域类**。域类有属性和关系。重数（在实体－联系图中称为基数）在类之间使用。定义需求时，ERD 或 UML 这两种建模方法是很相似的。

UML **类图**用于显示系统的对象类。一种用于显示用户问题域中事物的 UML 类图被称为**域模型类图**。另外一种 UML 类图被称为设计类图，这是在设计软件类时使用的。在第 10 章中，你会学习到设计类图。

在类图中，矩形代表类，连接矩形的线代表类之间的关系。图 4-11 所示为单个的域类：顾客。域类符号矩形有两部分。顶部写出类名，底部列出类的属性。以后你会学习到设计类图符号，其中矩形由三个部分组成，底部列出了类的方法；在问题域类图中可以不列出方法。

图 4-11 有名和属性的 UML 域类符号

类名和属性使用**驼峰符号**，使用这个记号使得词与词之间没有空格或下划线。类名以大写字母开头，属性以小写字母开头（如图 4-11 所示）。类图是为了显示类及类之间的关系。之前用来说明实体－联系图的例子将会利用 UML 域类图符号重新绘制，这样就能加以比较了。此外，在域模型类图中会说明有关类和关系的更复杂的问题。

4.4.1 域模型类图符号

图 4-12 所示为一个简化的域模型类图，由顾客、订单和订单商品三个类组成（就像是图 4-9 所示的 ERD 的例子）。每个类包括两个部分。类图的符号中，我们看到每个顾客可以下多个订单（从 0 到多个），而且每个订单只能由一个顾客来下达。为了解释得更清楚，“下达”和“由……组成”的关系也可以加入图中，如图 4-12 所示，但是这些细节是可选的。

从一方面来看重数是一对多的，而从另一个方面来看重数则是一对一的。重数符号，如图中订单类旁边直线上的星号所示，表示有很多订单。另一个关系表明一个订单包括一个或多个订单商品，一个订单商品只能关联一个订单。

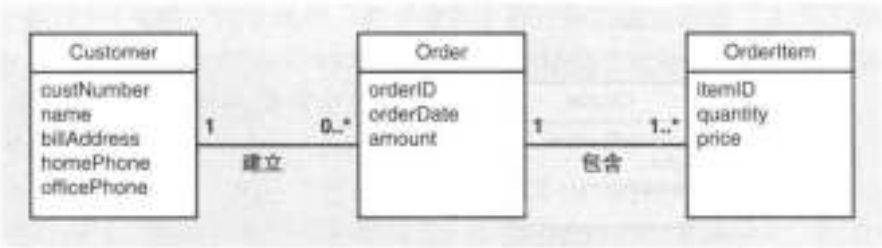


图 4-12 一个简单的域模型类图

图 4-13 总结了重数符号。

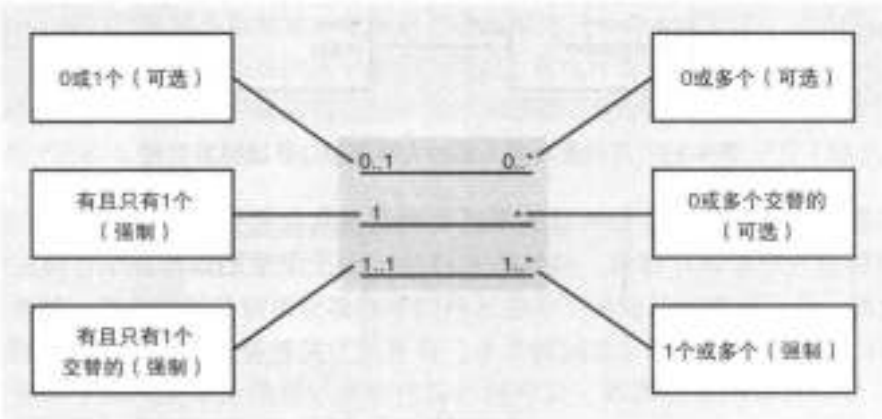


图 4-13 关系重数的 UML 符号

图 4-14 所示为域模型类图的另一个例子，这是之前讨论过的有很多分支银行的例子，不过之前是用 ERD 表示的。在这次的例子中，说明属性的 UML 符号是标识符或主键 {key}。

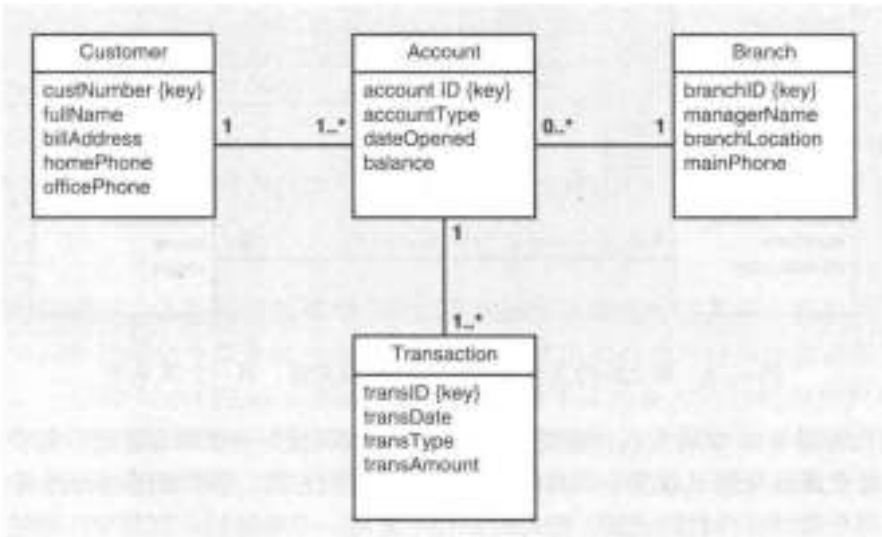


图 4-14 银行的域模型类图

图 4-15 所示为一个有多对多关系的域模型类图。大学里，一个课程可能分为多个课程部分，每个学生可以注册多个课程部分。每个课程部分也可以包括很多学生。因此，课程部分和学生之间是多对多的关系。在这种情况下，引入多对多关系是适当的，同时它们也要在模型中被表达出来。

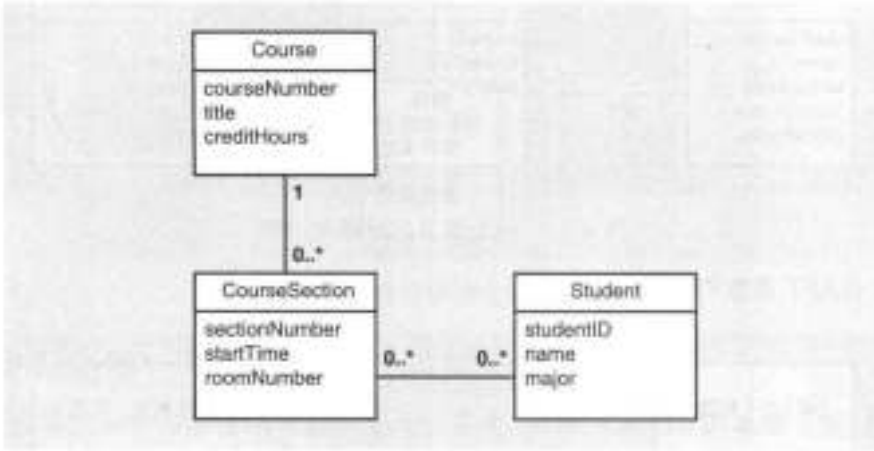


图 4-15　具有多对多关系的大学课程注册域模型类图

然而，在相近的分析中，分析员通常要发现包含附加数据的多对多关系，这个附加数据是很重要的而且是必须存储的。例如图 4-15 中，每个学生的课程成绩存储在哪里？这就是重要的数据，尽管模型中只说明了学生选择的课程部分而没有说明成绩。解决方法是添加一个域类来代表学生和课程部分之间的关系，这被称为**关系类**。关系类给出了被遗漏掉的属性。图 4-16 所示为扩展了的类图，其中包括名为课程注册的关系类，其中有学生成绩这一属性。一条虚线将关系类连接到课程部分和学生之间的关系线上。

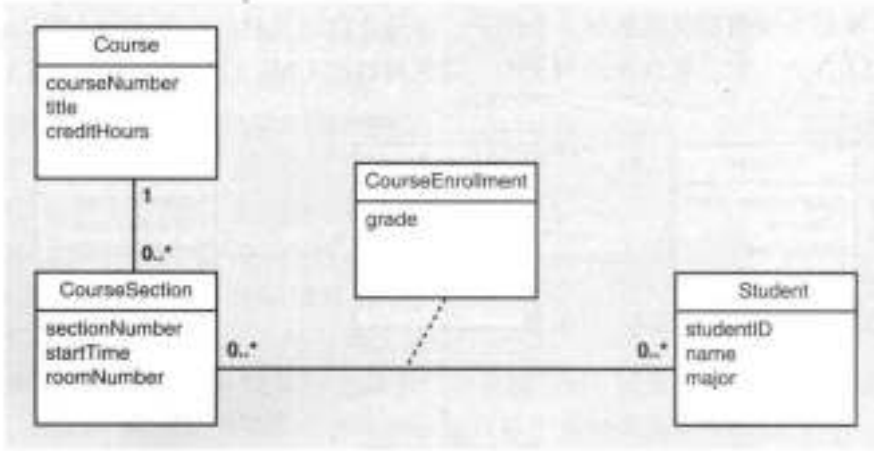


图 4-16　细化后的大学课程注册域模型类图，有一个关系类

从左向右看图 4-16 中的关系，该类图表示一个课程部分有许多课程注册，每个都对应各自的成绩，而每个课程注册又对应一个具体的学生。从右向左看，该类图表示一个学生对应很多课程注册，每个都对应各自的成绩，而每个课程又对应一个具体的课程部分。通过使用这个模型来实施的数据库能够生成成绩列表，列出所有学生每门课程对应的成绩及每个学生的成绩单。

4.4.2 有关对象类的更复杂的问题

先前，我们已经讨论过域类之间的关系了。在 UML 中，一个关系包含了很多类型的联系，因此在讨论 UML 图时需要比讨论 ERD 时更精确。例如，第 3 章中的用例图显示了用例之间的 << 包含 >> 关系。对于类图，在对象类中有三种类型的联系：关系（我们已经讨论过了）、泛化 / 特化联系以及整体 / 局部联系。这部分将介绍的是泛化 / 特化联系以及整体 / 局部联系，并说明它们在 UML 类图中是怎样表示的。

泛化 / 特化联系

泛化 / 特化联系是基于人们按照事物的异同来将其分类的思想建立的。泛化就是把相似类型的事物进行分组。例如，有很多种类的机动车：小汽车、卡车和坦克。所有的机动车都有某种共同的特点，因此机动车辆就是一个更泛化的类。特化就是把不同种类的事物进行分类。例如，某类小汽车中包括跑车、轿车和体育用车。这些小汽车在某些方面相似，而在其他方面却不同。因此，跑车就是小汽车中的一个特化类型。

泛化 / 特化联系用来把事物从最一般到最特殊的顺序进行建立或排列。如前面所介绍的那样，分类就是定义事物的类。在层次图中的每个类的上面也许有更一般的类，这个类称为**父类**。同时，每个类的下面也许有更具体的类，这个类称为**子类**。在图 4-17 中，一个小汽车有三个子类和一个父类（机动车辆）。UML 类图符号用一个指向父类的三角来表示泛化 / 特化层次图。

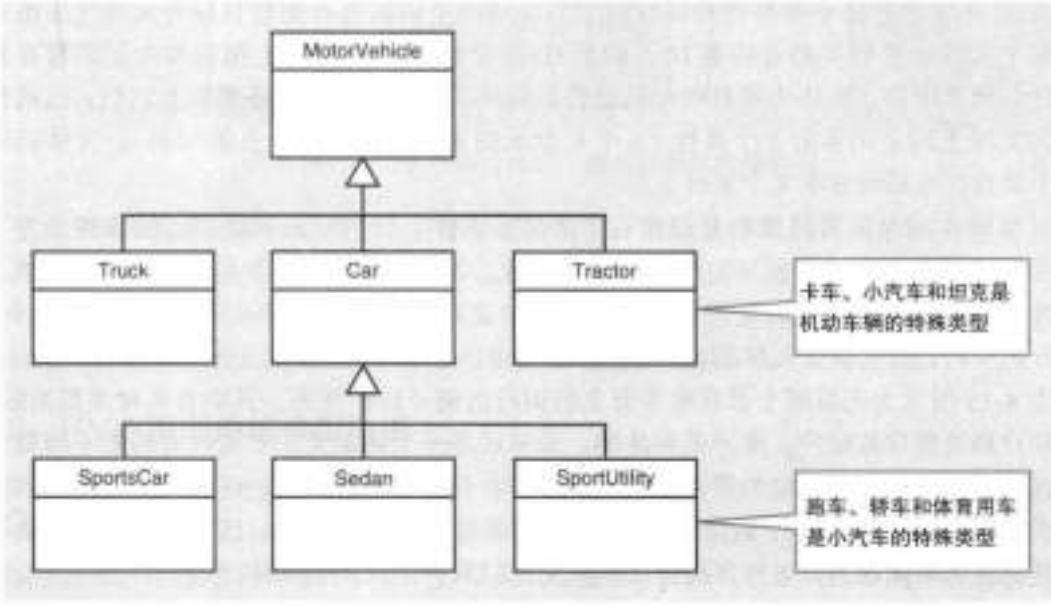


图 4-17 机动车类的泛化 / 特化联系

我们已经提到过，人们是使用泛化 / 特化层次图来理解现实世界的。换句话说，人们是通过把某些知识领域细化分类来学习的。一个知识丰富的银行家可以具体地讲解贷款和存款账户的种类。一个像 RMO 的 John Blankens 那样经验丰富的商人可以把各种户外运动和服装的种类说得清清楚楚。因此，当分析员询问用户的工作时，必须努力理解用户在工作中使用的知识，并把这些按照泛化 / 特化联系表示出来。从某种意义上来说，开发 RMO 新的销售和市场营销系统的动机就是因为 John 认识到 RMO 必须有一套新系统来处理各种特殊类型的销售（如网上销售、电话销售以及实体店销售）。图 4-18 所示为这些特殊的销售类型。

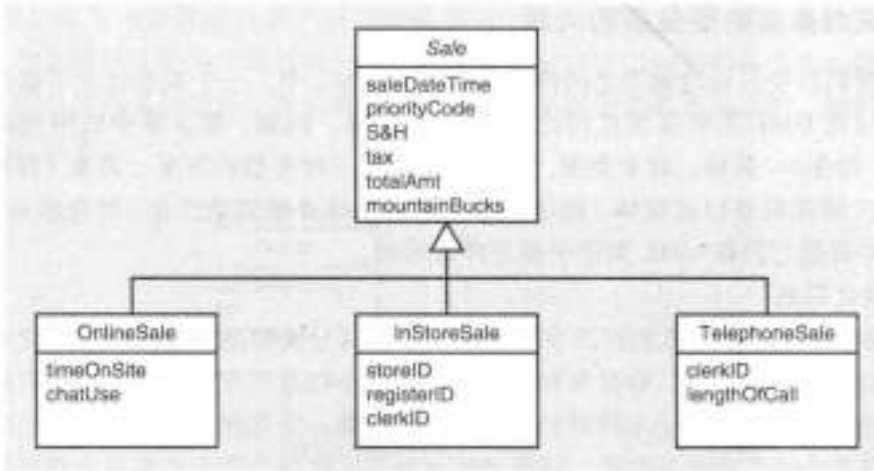


图 4-18 销售的泛化/特化联系（继承）

继承允许子类共享其父类所具有的特征。回到图 4-17，小汽车也是机动车辆，但是它有更具体的特征；跑车也是小汽车，不过多了些别的特征。从这一点来看，子类继承了父类的特征。在面向对象方法中，继承是一个关键的概念，这是由泛化/特化层次图决定的。有时，这种层次图也被称为继承联系。

在图 4-18 中，每个类都包括属性。销售类的每个成员都有销售日期时间和优先编码属性。每个实体店销售类都有店铺 ID、职员 ID 和登记 ID，但是网上销售和电话销售有其他的属性。网上销售、实体店销售和电话销售都继承了销售的属性，还要加上它们自己的特殊属性。实际上网上销售有 8 个属性（6 个是继承销售的，还有两个是附加的）。实体店销售有 9 个属性，电话销售有 8 个属性。

对于图 4-18，需要注意的是销售这个类名是斜体，因为它是抽象类。**抽象类**是为了使其子类能够继承它的属性、方法及联系的一个类。名为销售的这个类是没有真正对象的。每个销售必须是这些子类中的一个。**具体类**是有真正对象的类。有时，父类是抽象类；有时，由于分析员的目的它就是具体类。

图 4-19 所示为先前那个带有很多分支的银行的例子的扩展图，其中有两种类型的账户：存款账户和活期存款账户。账户是斜体的，说明这是一个抽象类。子类代表的是不同类型的账户，而不是包括账户类型的属性。每个子类都有自己的特殊属性，这些属性是不会运用到其他子类中的。存款账户有 4 个属性，而活期存款账户有 5 个属性。这里需要注意的是每个子类还能继承与顾客的关系，可随意选择一支行以及一个或多个交易。

整体/局部联系

人们认识事物信息的另一种方法是根据其各个部分来定义它们。例如，学习计算机系统可以使你认识到计算机是由不同部分组成的，这些不同部分是处理器、内存、键盘、磁盘存储器和显示器。键盘并不是计算机的一种特殊类型，而只是计算机的一个部分，同时它也是独立的事物。**整体/局部联系**用于表示一个类与包含在这类中的其他类之间的关系。

整体/局部联系有两种类型：聚合和组合。**聚合**代表了在聚合体（整体）与它的组件（局部）之间的整体/局部联系的一种类型，其中的局部是可以独立存在的。图 4-20 所示为计算机系统中聚合的概念，图中用菱形符号来表示聚合。**组合**代表了更强的整体/局部联系，其中的各个部分一旦关联就不能够独立存在。常用实心的菱形符号来表示组合。

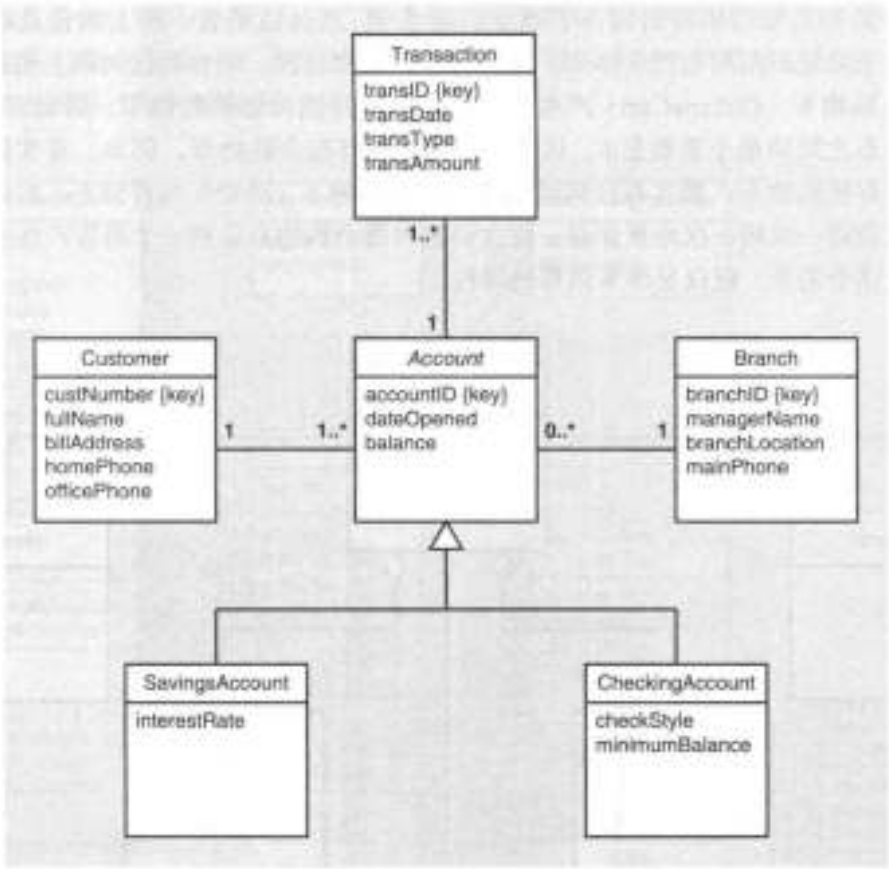


图 4-19 一个关于银行账户子类的扩展域模型类图

整体/局部联系（聚合或组合）主要使得分析员可以描述类之间关系的细微差别。对于任何关系，重数都能适用，例如计算机可以有一个或多个磁盘存储设备。

我们最近看到的 UML 类图例子是域模型类图。设计类图是类图的改进版，它用于说明新系统中的软件类。我们会在第 10 章学习如何将域模型类图转换成设计类图。

4.4.3 RMO 案例的域模型类图

RMO 的 CSMS 系统包含了许多域类、复杂的关系以及泛化/特化联系。信息系统的域模型类图会逐步发展为项目的成果；用例图会创建许多图表，而域模型类图只会创建一个。而且，域模型类图也不会像用例图那样，它不会只呈现出表象。开发和修改域模型类图的过程是分析员研究和学习问题域的过程。因此，域模型类图描述的信息是很详细并且有丰富特定含义的。

RMO 域模型类图是图 4-12 中显示的顾客、订单与订单商品的一个演变。大多数域类是从图 4-2 中的名词列表中被开发出来的。因为这个模型是非常复杂的，所以分析员可能一开始就会聚焦于一个子系统来降低复杂性。最后，所有的子系统能被组合一个域模型。

RMO 销售子系统

图 4-21 所示为 RMO 的 CSMS 系统中的销售子系统。销售子系统主要包括顾客、销售、销售商品、产品、促销和配套商品。这是一个不错的起点，但是还有附加的域类。此外，回忆一下关系，它们和类一样重要，因此这些必须得到确认。同时也还有特别的销售类型与购物车。

在图 4-21 中，每个顾客都能与一个或多个销售产生关系。注意，像上一节所讨论的那

样，有三种特别类型的销售出现在了继承的联系中（实体店销售、网上销售及电话销售）。因此，销售子系统的范围包括实体店、网上及电话销售过程。顾客的任何网上销售过程中都可以与在线购物车（OnLineCart）产生关系。有两种特别类型的购物车：活动车与存储车。顾客与购物车之间的最小重数是 0，这意味着可能没有包含购物车，例如，在实体店或电话销售中就不存在购物车。顾客在任何时候都能产生两种车（活动车与存储车）的最大值。存储车能通过会话一次接一次地被保存。每次销售和每个购物车会和一个顾客产生关系，因此子类能继承这个关系，仅仅是继承销售的属性。

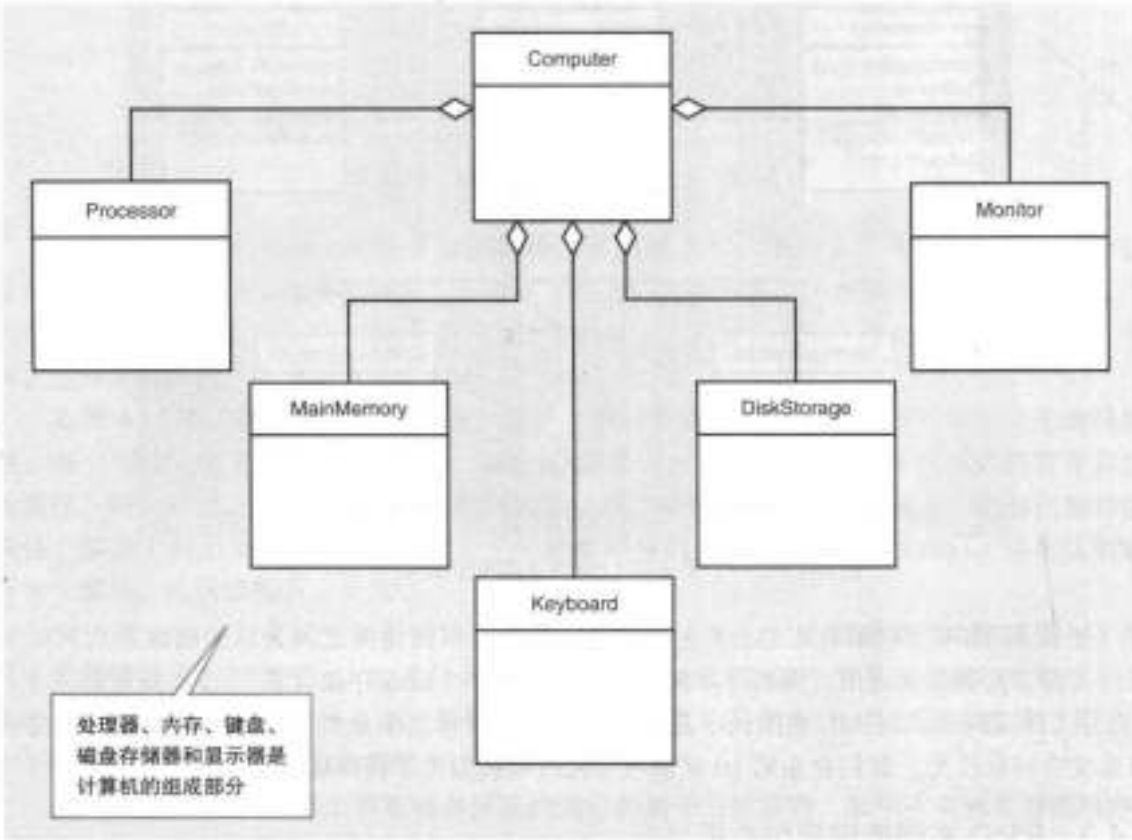


图 4-20　计算机及其组件之间的整体 / 局部图

一个独立销售与一个或多个销售商品产生关系。在线购物车中，它与一个或多个商品产生关系。在线销售时，当顾客对购物车中的商品进行结账时便产生了具体的销售。订单商品来自于每个购物车中的商品。最后，销售交易被创建且与销售产生关系。

销售可以有一个或多个销售商品，但每个商品是什么？每个销售商品与库存商品之间的关系解释了这个问题。每个特定的库存商品对应每个销售商品，这意味着一个特定尺寸和颜色的商品，例如一件衬衫或外套。对于某种尺寸和颜色的商品，库存中有其现存数量这一属性。因为有很多颜色和尺寸（每种都有自己的数量），所以每个库存商品就和产品有关系，它主要描述了这个商品（类别、描述、供应商、制造商及图片）。每个产品和很多库存有关系，并且每个库存和很多销售有关系。

一个产品可以在多个促销中出现，促销也可以包括许多产品，这样就产生了多对多的关系。关系类被添加并存储到每次促销中每个商品的价格信息中。每个产品可能会有很多关

系，配套商品也会应用到许多产品中。在这里，对于多对多关系没有定义关系类。需要注意的是这个关系可能会作为一元（递归）关系被建立。最后，每个产品可以有许多顾客评论，这些评论会在销售过程中被审查。

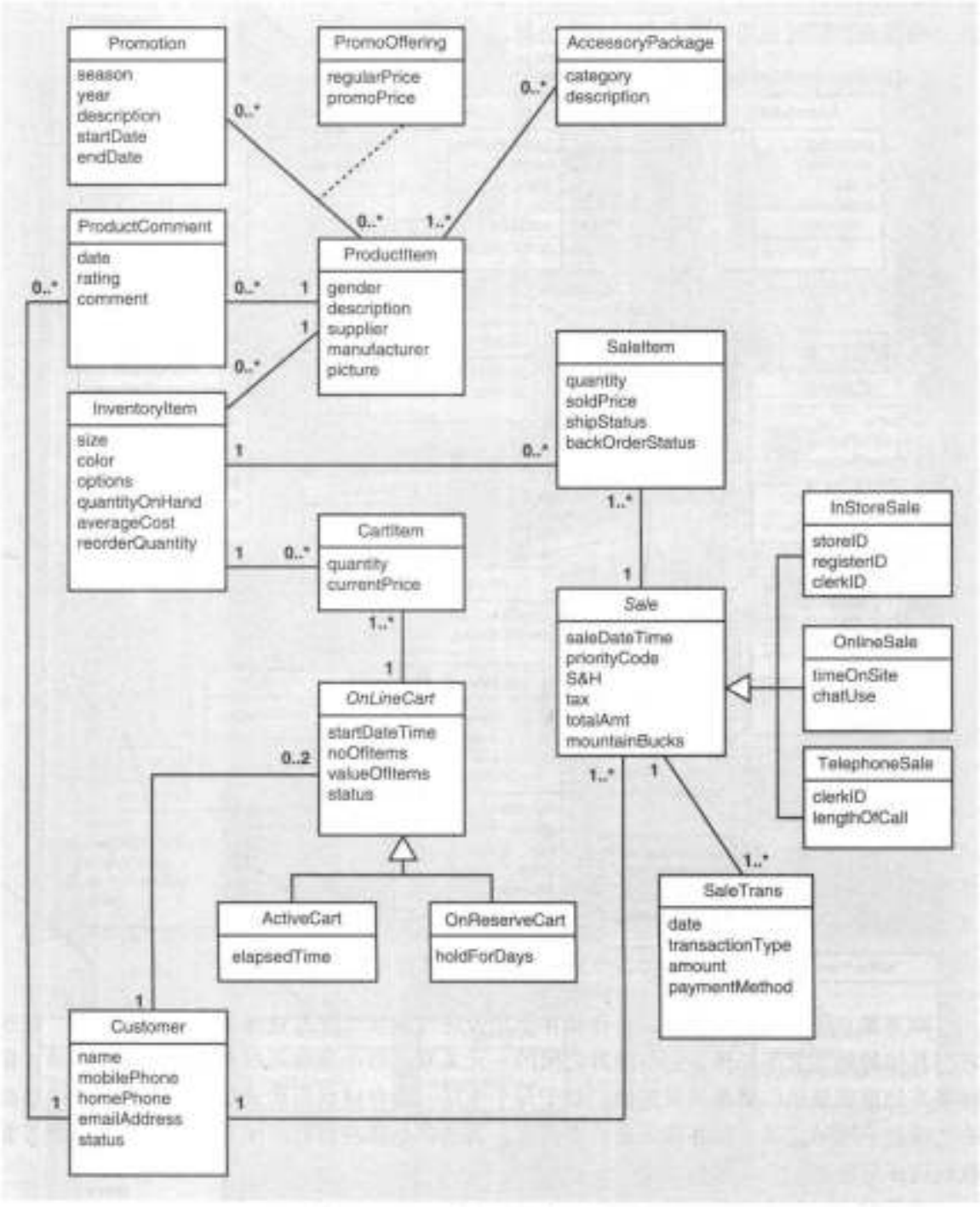


图 4-21 RMO 销售子系统域模型类图

RMO 顾客账户子系统

图 4-22 所示为顾客账户子系统域模型类图。有一些在销售子系统中的类也出现在了这

里。例如，顾客对两个子系统来说都是很重要的。销售和销售交易也同样如此。为了对顾客的账户进行核对并报告所有付款及退货，必须参考销售和销售交易信息。在几个子系统中重复运用域类并不意味着冗余。在复杂的域建模中，在将所有图表合并成整体之前，建立与分析独立的图表会比较简单。有时，项目团队根据子系统来分配工作，因此每个人会处理一个独立的图表，同时也要确定能和其他人协调。

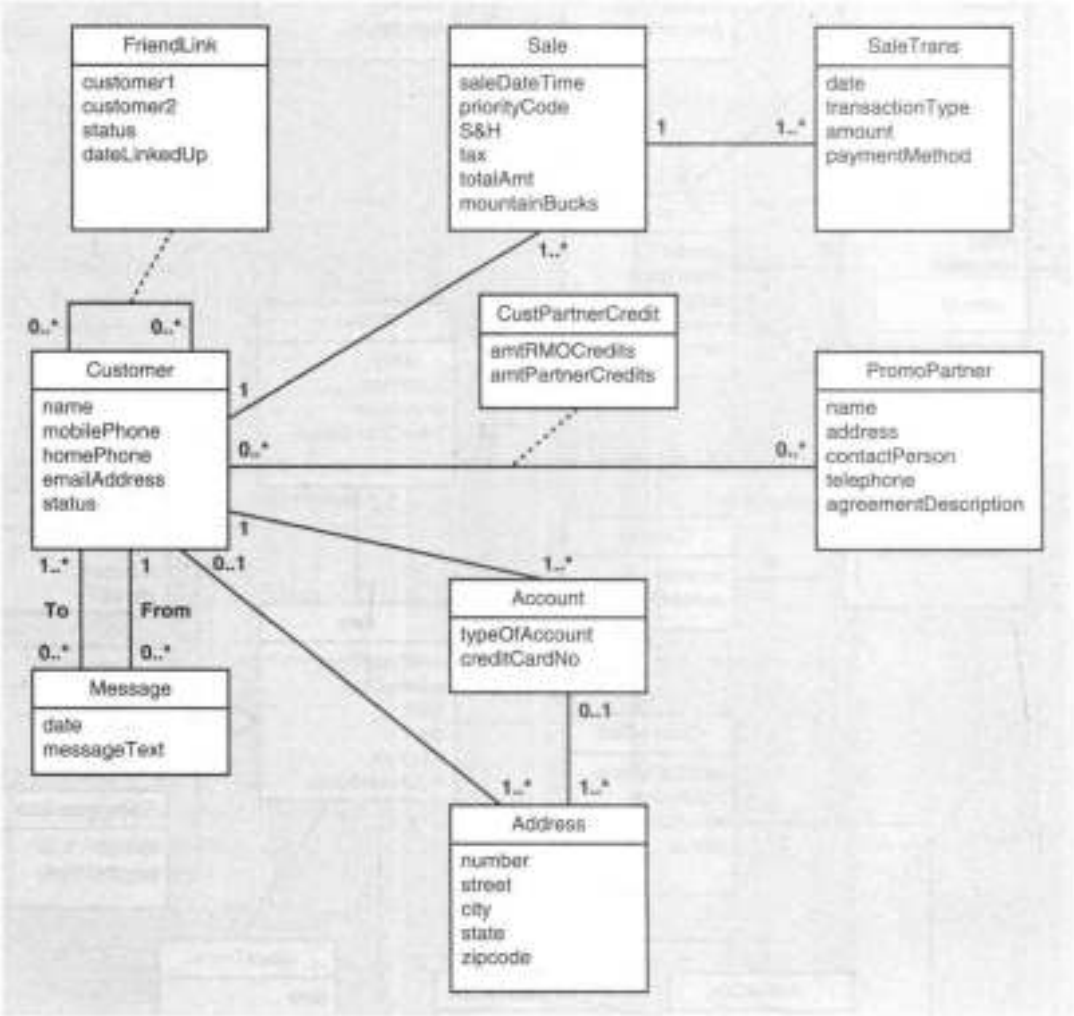


图 4-22　RMO 顾客账户子系统域模型类图

顾客账户子系统包括信息、合作伙伴信用及好友连接。好友连接类是一个关系类，但是它与其他的例子又不一样，它与顾客之间的一元关联。每个顾客可以和很多顾客有连接，在顾客类的顶部显示的就是关系连接。对于每个连接，会存储状态和成功连接的日期。信息类的处理是不同的。每个顾客能发送许多信息，每条信息能发给许多顾客。同样，每个顾客能收到许多信息。

完整的 RMO 域模型类图

RMO 的分析员会持续地建立每个子系统的模型。本章最后的练习部分会要求你创建其他子系统的图。图 4-23 所示为最终的 RMO 销售和市场营销系统的域模型类图。之前没有出现过的类包括运输者、运输、退货及建议。

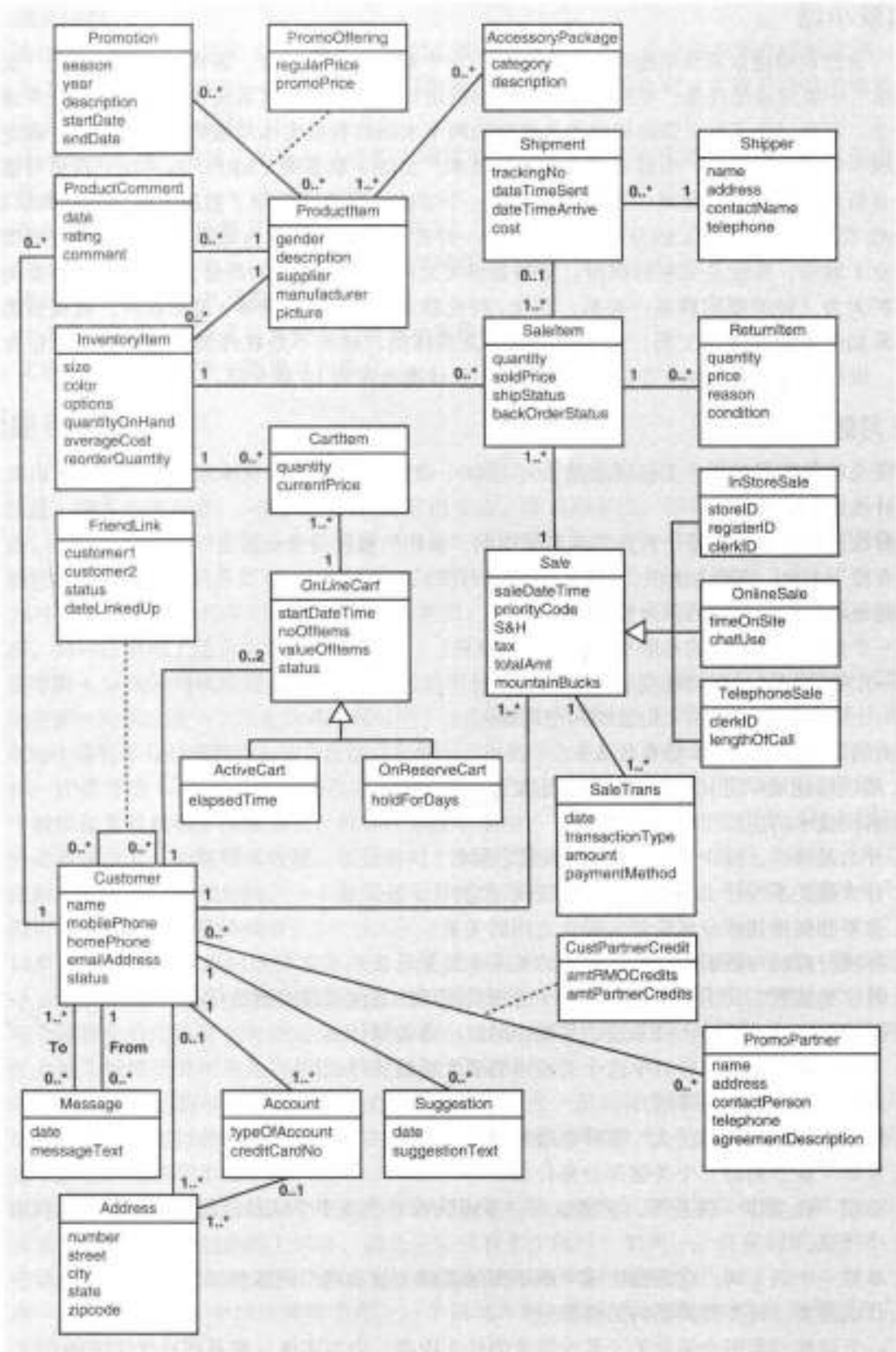


图 4-23 完整的 RMO CSMS 域模型类图

本章小结

这是说明建立系统功能需求模型技术的三个章节中的第二章，强调了在分析活动“定义需求”中要完成的任务。在用户工作环境中的用例和事物对所有系统开发方法来说是关键的概念。本章讨论了在工作环境中作为事物的两个术语的数据实体与域类。说明了两种确定问题域中的事物的技术：头脑风暴法和名词技术。实体－联系图（ERD）被传统分析员与数据库分析员用来建立问题域中事物的模型。一个实体－联系图显示了数据实体、属性和联系。UML 类图被使用 UML 的分析员用来达到一样的目标，这被称为域模型类图。域模型类图建立了域类、属性及关系的模型。重数是指类之间的关系连接的数量。UML 和域模型类图可扩大为三种类型的联系：关系、泛化/特化联系（继承）及整体/局部联系。域模型类图中附加的重要概念是父类、子类、抽象类及具体类。域类不是软件类，因此它们不包含方法。设计类图显示了包含方法的软件类图。设计类会在第 10 章介绍。

复习题

1. 定义功能需求的两个关键概念是什么（其中一个在第 3 章一个在本章）？
2. 什么是问题域？
3. 传统分析员与数据库分析员用在建模中的“事物”被称为什么？
4. 在使用 UML 的新方法中“事物”被称为什么？
5. 问题域中确定事物的两种技术是什么？
6. 一个餐厅问题域中的有形事物的例子有哪些？
7. 一个餐厅问题域中的地点或地址是什么？
8. 一个餐厅问题域中由人扮演的角色有哪些？
9. 头脑风暴法的主要步骤是什么？
10. 阐述问题域中定义的名词能帮助定义事物的原因。
11. 名词技术的主要步骤是什么？
12. 什么是属性、标识符（或主键）及复合属性？
13. 什么是关系？什么系统开发标准定义了它？
14. 你要如何描述或命名轮船与船长之间的关系？
15. 传统分析员与数据库分析员使用的关系术语是什么？
16. 什么是重数？传统分析员与数据库分析员使用的其他关系术语是什么？
17. 顾客下了 0 个或多个订单这个关系中的最小重数是什么？
18. 一个顾客只下了一份订单这个关系中的最大重数是什么？
19. 重数限制的例子有哪些？
20. 属性的三种类型是什么？哪种是最常用的？
21. 实体－联系图的三个关键部分是什么？
22. 草拟一张实体－联系图，它能显示一个团队有 0 个或多个队员以及每个队员有且仅有一个团队。
23. 草拟一个语义网，它能在实体－联系图的基础上显示两个团队和五个队员。
24. 什么是类、域类和类图的关键部分？
25. 一个域模型类图会显示关于系统需求的什么内容？它与实体－联系图有什么区别？
26. 使用驼峰符号为以下类列出合适的 UML 类名：毕业生、在校生、课程指导老师及最终

测试回馈。

27. 为以下属性列出合适的 UML 属性名：学生姓名、课程成绩、专业名及最终考试成绩。
28. 为 22 题中的例子画出一份简单的域模型类图，例子是一个团队有 0 个或多个队员而每个队员有且只有一个团队。
29. 什么是关系类？为这个团队与队员扩大域模型类图来显示在每次游戏中每个队员的游戏统计记录。
30. 在 UML 中，类图中的三种联系类型是什么？
31. 什么是泛化 / 特化联系？它说明了什么面向对象的术语？
32. 比较父类与子类。比较抽象类与具体类。
33. 什么是整体 / 局部联系？为什么它能显示重数？
34. 比较整体 / 局部联系中的聚合与组合。

问题和练习

1. 为以下例子画一张实体 – 联系图，包括最小基数与最大基数。这个系统存储两种事物的信息：汽车与拥有者。一辆汽车的属性有制造商、样式和年份。拥有者的属性有姓名与地址。假定一辆汽车只能被一个拥有者所有，一个拥有者能拥有许多辆车，但是拥有者也可能没有车（也许她只是卖车，但是你还是想要有她在这个系统中的记录）。
2. 为习题 1 中描述的汽车与拥有者画一张类图，但是还要包括子类跑车、小轿车和小型货车，同时还要加上适合它们的属性。
3. 思考图 4-16 所示的域模型类图，被修改过的图显示了课程注册这个关系类。这个模型允许学生在同一时间注册一个以上的课程部分吗？这个模型允许学生注册同一课程的几个部分及获取每个课程部分的成绩吗？这个模型会存储关于所有学生在所有课程部分中获得的成绩吗？
4. 再一次思考图 4-16 所示的域模型类图。添加以下内容并且列出你不得不做出的假设：一个教师经常教授很多课程部分，但是在某些学期中，一个教师又不会教很多。每个课程部分必须至少有一个教师来教授，但是有时，教师团队会教授课程的一部分。而且，为了确保所有课程部分是相似的，一个教师被安排成为课程协调者来监督这个课程，同时每个教师可以是许多课程的协调者。
5. 如果你在习题 4 中画的域模型类图显示了教师与课程部分之间多对多的关系，那么更进一步地观察这个关系可能会揭露存储一些附加信息的需求。这样的需求包括什么？（提示：每个课程部分的指导者有特定的工作时间吗？你会给每个课程部分的指导者一些评估吗？）扩大这个域模型类图来允许系统存储这样的附加信息。
6. 如果一个系统需要存储大学机房中关于计算机的信息，例如每个计算机的特征和地址。那么这个模型中包括的域类是什么？在这些类中的关系有哪些？每个类的属性有哪些？画出这个系统的域模型类图。
7. 思考图 4-21 所示的 CSMS 销售子系统的域模型类图。如果创建实体店销售，那么它应该有多少属性？如果创建网上销售，那么它应该有多少属性？如果一个现有的顾客下了一份电话订单，那么在此次交易中整体上有多少新对象被创建？解释原因。
8. 再一次思考图 4-21 中的域模型类图。一个活动车的对象有多少属性？存储用车能保存商品吗？解释原因。
9. 在 RMO 中，产品与库存并不一样。产品就像是由 Leather " R" 提供的一件男士皮夹克。

而库存是有着特定尺寸和颜色的皮夹克，比如一件中号的棕色皮夹克。如果 RMO 添加了一件新皮夹克到目录中，且库存中有六种尺寸和三种颜色，那么整体上需要添加多少对象？解释原因。

10. 思考图 4-24 所示的域模型类图，它包括的类有大学、专业及教师。
    a. 模型中显示的是哪种类型的 UML 联系？
    b. 教师类有多少属性？哪个属性是从其他类中继承的？
    c. 如果你添加一个大学、一个专业及四个教师的信息，那么要向系统中添加多少对象？
    d. 一个教师能同时在几个专业工作吗？解释原因。
    e. 一个教师能同时在两个专业工作吗？其中一个专业是在商学院，其他三个专业是在艺术与科技学院。解释原因。

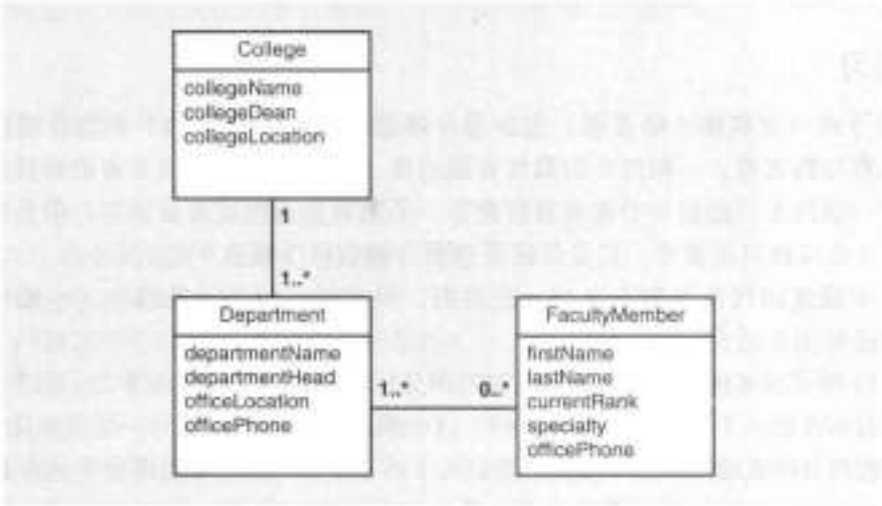


图 4-24　大学的类模型域图

11. 回忆一下关于你自己的大学的信息。通过使用域模型类图符号为以下信息来创建泛化 / 特化层次图：（1）教师的类型；（2）学生的类型；（3）课程的类型；（4）财政援助的类型；（5）住宿的类型。在每个用例中包括父类与子类的属性。
12. 思考在建立一辆汽车及其部件的模型时包含的类。然后画一张域模型类图来显示其中包含的整体 / 局部联系，包括重数。包含的是哪种类型的整体 / 局部联系？
13. 图 4-23 所示为完整的 RMO 的 CSMS 系统的域模型类图。以这张图以及第 3 章讨论的子系统为基础，画一张 CSMS 市场营销子系统的域模型类图。
14. 再一次以图 4-23 所示的完整 RMO 的 CSMS 系统的域模型类图为基础，画一张 CSMS 订单实施子系统的域模型类图。

扩展资源

Classic and more recent texts include the following:

Peter Rob and Carlos Coronel, *Database Systems: Design, Implementation, and Management*, (7th ed.). Course Technology, 2007.

Craig Larman, *Applying UML and Patterns* (3rd ed.). Prentice Hall, 2005.

Grady Booch, Ivar Jacobson, and James Rumbaugh, *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

需求模型的延伸

学习目标

阅读本章后，你应该具备的能力：

- 为正在开发的用例撰写完整的描述。
- 开发活动图来建立活动流的模型。
- 开发系统顺序图。
- 开发状态机图为对象行为建模。
- 解释如何结合用例描述与 UML 图一起定义功能需求。

开篇案例　无限电子公司：供应链一体化

无限电子公司是一家仓储式销售商，它从不同供应商处买入电子设备，然后再卖给遍及整个美国和加拿大的零售商。它在洛杉矶、休斯顿、巴尔的摩、亚特兰大、纽约、丹佛和明尼阿波利斯都有办事处和仓库。它的顾客既包括像 Target 这样的覆盖全国范围的大型零售商，同时也有中等规模的独立的电子商店。

许多大型零售商正致力于供应链一体化。信息系统过去只关注内部数据的处理，然而，如今的这些连锁零售商想要供应商成为完整的供应链系统的一部分。换句话说，系统需要在公司之间进行沟通，以使供应链更有效率。

为了保持作为批发商领导的地位，无限电子公司对其系统进行了调整，使之能够与供应商（电子设备的制造商）和顾客（零售商）进行连接。公司正在开发一个完整的新系统，而这个新系统是使用面向对象技术来提供这些连接的。面向对象技术通过使用预先定义好的组件和对象使系统与系统之间的接口连接变得容易，以此来加快开发过程。幸运的是，许多系统开发人员已经体验了面向对象的开发方法，同时他们也希望将这个技术和模型应用于系统开发项目中。

William Jones 正在给一批系统分析员讲解面向对象的开发方法，这些分析员正在接受这种方法的培训。

William 告诉他们："我们正在使用面向对象的原理开发大多数新系统。新系统的复杂性和它的交互功能使面向对象方法成为开发需求的自然之选。这与你过去的思维过程可能稍有不同，但是面向对象的模型能紧跟新的面向对象程序语言和框架。"

William 说这番话还只是热身。

他继续说："根据对象来考虑一个系统是非常有趣的。这也和你们在编程课上学到的面向对象的编程技术是一致的。开发用户界面时，你可能会首先学着去考虑对象。界面上的所有控件，如按钮、文本框和下拉框都是对象。每个对象都有自己一系列的触发事件，从而激活程序功能。"

其中一个分析员问："这要怎么应用到实际情况中呢？"

William 解释说："你们只要将这种思维过程拓展开来，要把订单、雇员这样的事物也都想象成对象。我们可以称之为问题域或业务对象，以便把它们和窗口、按钮这样的屏幕对象区分开。在分析过程中，我们要找到每个业务对象的所有触发事件和方法。"

"那我们要怎么做呢？"另一个分析员问道。

William 回答说："继续做实情调查活动并为每个用例建立一个更好的说明文档。用例中业务对象之间的交互方式决定了你是怎么确定起始活动的，我们把那些活动看成是对象之间的消息。最棘手的部分是你需要按照对象而不是仅按照过程来思考。有时，假设自己就是一个对象会非常有用。我会说：'我是一个订单对象。其他对象将会要求我有什么样的功能和服务呢？'一旦掌握了技巧，工作起来会得心应手，并且在开发图表时能很容易地看清楚系统需求是如何展现的。"

5.1 引言

系统开发中定义需求的主要目标是理解用户需求、理解业务过程如何运行以及理解系统如何支持那些业务过程。正如第 2 章指出的那样，系统开发者使用一系列模型来发现和理解新系统的需求。这种活动是系统开发过程中系统分析的一个关键部分。开发说明文档这个过程的第一步需要用到你在第 2 章学习的实情调查活动。实情调查活动也被称为*发现活动*，而且显然，发现必须是优于理解（即说明文档）的。

第 3、4 章介绍的模型关注系统功能需求的两个主要方面：包含在用户工作中的用例和事物。用例通过使用用户目标技术和事件分解技术来确定。UML 用例图用于展示用例和角色。信息系统也需要记录和存储包含在业务过程中的事物信息。在手工系统中，信息是记录在纸上并存储到档案柜中的。在自动化系统中，信息是存储在电子文档或数据库中的。系统的信息存储需求要么是用实体－联系图（ERD）进行记录的，要么是用 UML 域模型类图进行记录的。

在本章中学习的附加技术和模型能帮助你拓展需求模型，以此来展示系统中用例和域类的附加信息。用例描述、UML 活动图和 UML 系统顺序图的完整开发可用于显示更多关于用例的信息。接着要介绍的是 UML 状态机图，这些能帮助你展示更多关于域类的信息。要记住的是，在定义系统需求时，你也要做设计和实施工作，就像是在第 1 章中介绍的贸易展览应用程序。下一章开始介绍系统设计活动。

5.2 用例描述

用例和用例图的列表为系统提供了所有用例的概述。关于每个用例的详细信息是用**用例描述**来描述的。第 3 章介绍的是简单的用例描述。用例描述列出且描述了用例的处理细节。在 UML 中，人被称为参与者，正如用例图中显示的那样。参与者总是在系统自动化边界的外部，但可能是手工系统的一部分。通过那样的方式定义参与者——如那些和系统交互的人，我们可以更精确地定义自动化系统必须回应的交互活动。这种更紧密的聚焦可以定义自动化系统本身的具体要求——为了修改它们，我们会将事件表移到用例细节中。

另外一种考虑参与者的方式是将其作为一个角色。例如，在 RMO 例子中，用例"创建顾客账户"可能会包含一个客服代表，他在电话中与顾客交谈。如果这个顾客直接在网上添加或更新了信息，则顾客也可能是参与者。

为了创建一个全面、健壮并且能满足用户需求的系统，我们必须理解每个用例的详细步骤。在内部，为了完成业务过程，用例要包括完整的步骤顺序。通常，在单一用例中业务步骤会有几种变化。用例创建顾客账户会有一个独立的活动流，而这个活动是根据参与者调用的用例决定的。在客服代表在电话中更新信息这个过程中，每个活动流都是创建顾客账户这个用例的有效顺序。这些不同的活动流都称为**场景**，有时也称为**用例实例**。因此，场景是一个用例中唯一的一系列活动，而且也代表了通过用例的唯一途径。

5.2.1　简单的用例描述

根据分析员的需求，用例描述倾向于采用两种详细程度的描述：简单描述和完全展开描述。第 3 章中已经介绍了一些简单的用例描述（见图 5-1）。简单描述可以用在非常简单的用例中，尤其是当要开发的系统是小型且很好理解的应用程序时。简单的用例通常包含单一的场景和非常少的异常情况。“添加产品评论”或“发送信息”就是这样的例子。而“加入购物车”这样的用例就很复杂，它要用完全展开描述来描述。

用例	简单的用例描述
创建顾客账户	用户 / 参与者输入新顾客的账户数据，系统指定账户号，创建顾客记录和账户记录
查询顾客	用户 / 参与者输入顾客账户号，系统能检索并显示顾客和账户数据
处理账户变动	用户 / 参与者输入订单号，系统能检索顾客和订单数据；参与者输入调整数量，而系统根据调整情况创建交易记录

图 5-1　用例和简单的用例描述

5.2.2　完全展开的用例描述

完全展开描述是记录用例的最正式的方式。软件开发人员面临的主要困难之一是要不断加深对用户需求的理解。但是如果创建了一个完全展开用例描述，那么完全理解业务过程和系统支持它们的方式的可能性就加大了。图 5-2 就是创建顾客账户这个用例的一个完全展开用例描述。

图 5-2 也是记录其他用例和场景的完全展开描述的一个标准模板。第 1 行和第 2 行是用来确定所记录用例的内部用例和场景。在更大型或更正式的项目中，还可以为用例添加带有扩展名的唯一标识符，以标识特定的场景。有时，还会添加制作该表格的系统开发人员的姓名。

第 3 行确定了触发用例的事件。第 4 行是用例或场景的一个简单描述。分析员可能会复制之前建好的简单描述。第 5 行确定了参与者。第 6 行确定了其他用例和它们与该用例相关联的方式。这些对其他用例的交叉引用能帮助我们记录用户需求的各个方面。

利益相关者这行确定了相关的人员而不是特定的参与者。他们可以是那些没有真正调用用例但是对用例产生的结果感兴趣的用户。例如，在图 5-2 中，财务部门对精确地获取账单和信用卡信息很感兴趣。尽管没有人在市场部创建新顾客账户，但他们会对新顾客和创建市场推广做统计分析。因此，市场部人员对获取和存储在创建顾客账户用例中的数据非常感兴趣。销售部对创建一个容易使用且能吸引用户的界面来确保销售不会因为用户经验的缺乏而流失非常感兴趣。对系统开发人员来说，考虑所有的利益相关者是为了确保他们已经充分理解了所有的需求。

用例名称	创建顾客账户	
场景	创建在线顾客账户	
触发事件	新顾客想要在线建立账户	
简单描述	在线顾客通过输入基本信息创建顾客账户，然后填写一个或多个地址及信用卡或借记卡	
参与者	顾客	
相关用例	"购物车结账"可能会调用这个用例	
利益相关者	财务、市场、销售	
前提条件	顾客账户子系统必须可用 信用 / 借记授权服务必须可用	
后续条件	创建并保存顾客 创建并保存一个或多个地址 验证信用卡 / 借记卡信息 创建并保存账户 地址和账户必须与顾客关联	
活动流	参与者	系统
	1. 顾客表明想要创建顾客账户并且输入基本的顾客信息 2. 顾客输入一个或多个地址 3. 顾客输入信用卡 / 借记卡信息	1.1　系统创建一个新顾客 1.2　系统提示输入顾客地址 2.1　系统创建地址 2.2　系统提示输入信用卡 / 借记卡 3.1　系统创建账户 3.2　系统证实信用卡 / 借记卡信息的授权 3.3　系统关联顾客、地址和账户 3.4　系统返回有效的顾客账户细节
异常情况	1.1　基本的顾客数据不完整 1.2　地址无效 1.3　信用卡 / 借记信息无效	

图 5-2　创建顾客账户的完全展开用例描述

第 8 行和第 9 行提供了用例执行前后系统状态的临界信息，分别称为前提条件和后续条件。**前提条件**确定了在用例开始时系统的状态，包括什么对象必须存在、哪些信息可用，甚至用例开始之前参与者是什么样的状态。

后续条件确定了在用例完成时什么必须为真。最重要的是它们表明了通过用例创建了什么新对象或更新了什么新对象以及对象需要怎么关联。后续条件之所以那么重要有两个原因。首先，它们为说明测试用例的预期结果奠定了基础。例如，在创建顾客账户这个用例中，测试顾客记录、地址记录和账户记录是否已成功添加到数据库中是很重要的。其次，后续条件中的对象指明包含在用例中的对象，这对设计来说是很重要的。在第 10 章和第 11 章中你会看到用例的设计包括为对象确定和安排职责，而这些对象能协助完成用例。在这样的情况下，顾客、一个或多个地址以及账户对象能协作来创建一个新的顾客账户。

模板中的第 10 行描述了用例活动流的详细信息。在这个例子中采用了两列说明的形式，确定了参与者的执行步骤和系统需要的响应。项目编号能帮助确定步骤顺序。第 11 行描述了可选活动和异常情况。异常情况的编号也同样有助于将异常与活动流中的特定步骤联系在一起。

图 5-3 所示为用例"运输商品"的用例描述。这个描述的场景假定正在运送一份新订单

而不是运送先前订单中延期发货的物品。要注意的是，用例描述已经简化了手工作业的描述，并且手工作业是和运输商品相结合完成的。一些分析员记住了这些细节，但是其他人不会记住，因为他们重点关注的是与计算机应用程序的交互活动。在这个用例中，前提条件显示了用例执行之前必须存在什么样的现有对象。他们不能为顾客运送不包括在现有订单中的商品。后续条件再一次指出当说明测试用例的预期结果时要寻找什么，同时也显示了在设计过程需要合作的对象。

用例名称	运输商品	
场景	为新订单运输商品	
触发事件	运输部门被通知有新的送货订单	
简单描述	运输部门能检索销售细节，找到每个商品订单并标记为已发货，记录哪些商品订单暂时无法发货，以及进行发货工作	
参与者	运输部门的员工	
相关用例	无	
利益相关者	市场部、销售部、运输部、仓库经理	
前提条件	顾客和地址必须存在 订单必须存在 订单商品必须存在	
后续条件	创建运输情况并关联承运人 已发货的商品更新为“已发货”并与运输信息关联 未被运输的商品标记为延期订单 验证和提供运输标签	
活动流	参与者	系统
	1. 运输部需要销售订单及订单商品信息 2. 运输部门安排送货员 3. 对于每个可发货的订单，运输部标记这个订单已发货 4. 对于每个不可发货的商品，运输部门要记录这是延期订单 5. 运输部门要求运输标签提供包装尺寸和重量	1.1 系统查询销售并返回顾客、地址、销售和销售商品信息 2.1 系统创建运输信息并与送货员关联 3.1 系统更新销售商品 4.1 系统更新延期订单上的商品 5.1 系统产生运输标签 5.2 系统记录运输成本
异常情况	2.1 运输人如果不能到送货地点，那么就需选择其他人 3.1 如果订单商品被损坏了，那么就要获取新商品并更新商品数量 3.2 如果商品的条形码没有被扫描，那么运输部门必须手工输入条形码 5.1 如果打印标签没有被正确打印，那么这个标签必须手工解决	

图 5-3　运输商品的完全展开用例描述

5.3　用例活动图

记录用例的另一种方式是使用活动图。在第 2 章中，你已经学习了作为工作流图的一种形式的活动图。活动图是记录业务过程工作流，非常易于理解。活动图是一个标准的 UML 图，并且它们也是记录每个用例的活动流的一种有效技术。

图 5-4 是记录创建顾客账户这个用例的活动流的活动图。有时，活动图可以替代用例描述中活动流的部分，还有时活动图的创建是用来补充用例描述的。图中有两条泳道：一条是

顾客，还有一条是系统。顾客有三个活动，而系统有五个活动。

在用例的活动流很复杂的情况下，活动图就很有用。用例加入购物车是很复杂的，因为在添加商品到购物车时会调用三个其他的用例。例如，参与者可能要搜索一个产品，然后在添加商品到购物车之前会查看产品评论。一旦商品被添加，参与者可能会搜索并查看可用的配套商品，然后添加一个或多个到购物车中。图 5-5 所示的活动图是加入购物车用例的活动流。灰色的椭圆形是指在加入购物车时调用的其他用例。用例中的活动是在其他用例之间的。加入购物车用例包括选择选项和数量、添加到购物车、选择配套商品选项和数量以及添加到购物车。当活动图在上下文中显示时，丰富的用户体验意图变得更为明显。

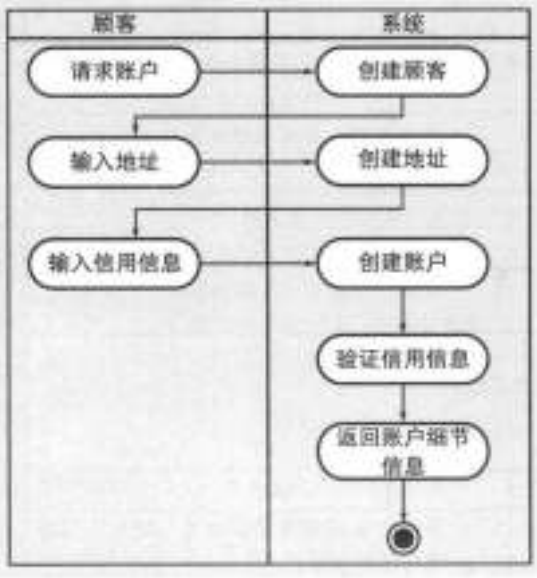


图 5-4　记录创建顾客账户这个用例的活动流的活动图

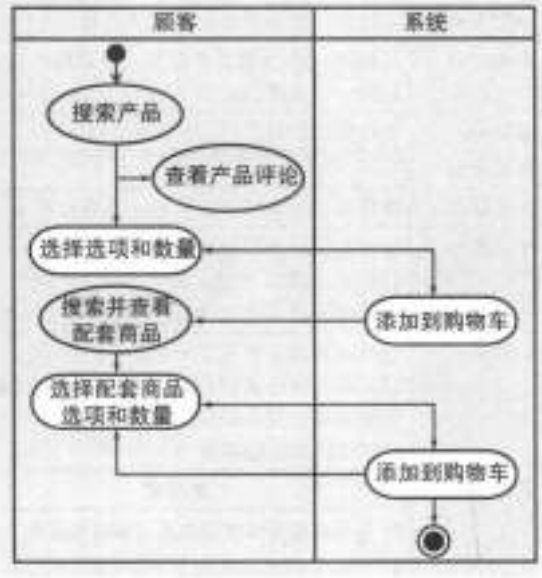


图 5-5　显示了丰富用户体验的加入购物车用例的活动流

5.4　系统顺序图——确定输入和输出

在面向对象方法中，信息流是通过参与者或内部对象来回发送信息形成的。**系统顺序图**（SSD）用于描述进出自动化系统的信息流。因此，系统顺序图记录输入和输出并确定了参与者和系统之间的交互。系统顺序图是**交互图**的一种。

5.4.1　系统顺序图符号

图 5-6 所示为一个普通的系统顺序图。与用例图一样，用人形符号代表和系统交互的参与者——人（或角色）。在用例图中，参与者“使用”系统，但是在系统顺序图中，参与者如何通过输入数据、获得输出数据来与系统进行交互才是重点。标记为“:System”的方框是代表整个自动化系统的一个对象。在系统顺序图和所有其他交互图中，分析员使用对象符号代替类符号。在对象符号中，方框指的是一个独立的对象，而不是所有类似对象的类。这个符号是由带下划线的对象名称和一个矩形框组成的。带下划线的对象名称前面的冒号是对象符号中的可选部分，但习惯上常常使用。在交互图中，消息通过独立的对象而不是类进行发送和接收。在系统顺序图中，只有代表整个系统的对象才被包括进来。

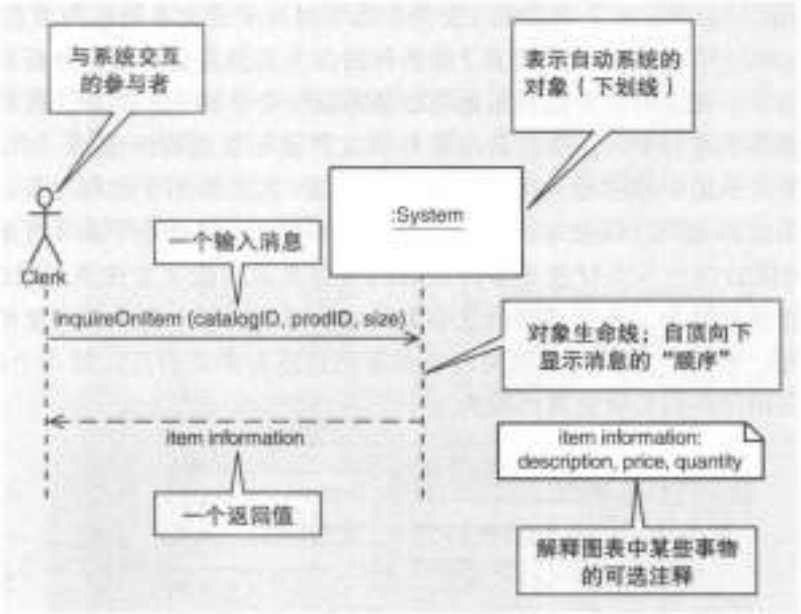


图 5-6　一个普通的系统顺序图

处在参与者和“:System”下面的竖直的虚线称为生命线。在用例中，**生命线**或**对象生命线**仅仅是这个对象（参与者或对象）的扩展。生命线之间的箭头代表了参与者发送的消息。每个箭头都有一个起点和一个终点。消息的起点就是发送它的参与者或对象，即箭头尾部的生命线所指示的。同样，消息的目标参与者或对象是由箭头所指向的生命线指示的。生命线的目的是指示参与者和对象发送和接收消息的顺序。在这个图中，消息的读取顺序是从顶部到底部的。

标有记号的消息用来描述消息的目的以及被发送的任何输入数据。消息的名称应该以动名词命名，以此来使目的明确。消息标记的语法有几个选项，图 5-6 显示了最简单的形式。注意，箭头用来代表消息和输入数据。但是消息在这里到底是什么意思？在顺序图中，消息是在目的对象上调用的一种活动，它更像是一条命令语句。注意，在图 5-6 中，输入的消息被称为 inquireOnItem。员工向系统发送请求（消息）来找到商品。与消息一起发送的输入数据被包括在圆括号内，在这个例子中，它是用来确定特定商品的数据。语法仅仅是带有用括号括起来的输入参数的消息名。这种语法形式用实线箭头表示。

返回值在格式和意义上有一些微小的差别。注意，箭头是虚线的。虚线箭头表明一个响应或应答，而且如图中所示，它通常紧跟在起始消息后面。标记的格式也是不同的。由于它是一个响应，所以只有响应中发送的数据才被标明。这里没有请求服务的消息，只有正在返回的数据。在本例中，一个有效的响应可以是所有返回消息的列表。例如，商品的描述、价格和数量。然而，用缩写形式表示同样令人满意。在这个例子中，返回消息被称为商品信息。同时需要另外的记录来显示详细内容。在图 5-6 中，这种另外的信息显示为注释。任何 UML 图中都可以添加注释，这是为了便于解释。商品信息的细节还可以记录在支持性说明中，甚至只是被顾客类中的属性进行引用。

通常，同样的消息可以被发送多次。例如，当参与者向一份订单中输入商品时，就要多次发送为一个订单增加商品的消息。图 5-7a 举例说明了显示这种重复操作的符号。消息和它的返回值放在一个较大的矩形框中，这被称为**循环框架**。这个框架的顶部小矩形框是控制大矩形框内部消息行为的描述性文本。所有商品的循环条件表明框内的消息重复多次或与多个实例关联。

图 5-7b 显示了一个备用符号。在这里，方括号和其中的文本被称为消息的**真 / 假条件**。真 / 假条件前面的星号（*）表明只要真 / 假条件的值为真消息就重复。分析员使用这种缩写符号有几种原因。首先，消息和返回数据可以显示在一个步骤中。注意，返回数据作为返回值标识在赋值操作符（:=）的左边。备用符号的这种表示仅表明该值是一个返回值。其次，真 / 假条件放置在消息中。注意，在这个例子中，真 / 假条件用于控制循环。真 / 假条件也用于评估任何形式的测试，以决定消息是否被发送。例如，考虑一下真 / 假条件［信用卡支付］。如果被测试的信用卡支付这个事件是真的，那么系统就要发送消息来验证信用卡号。最后，消息上的星号是为了表明这个消息是重复的。因此，对于简单的重复消息，备用符号可以变得更简短。然而，如果循环框架内有多条消息或有多重消息，每一个都有自己的真 / 假条件，那么使用循环框架就更清晰精确。

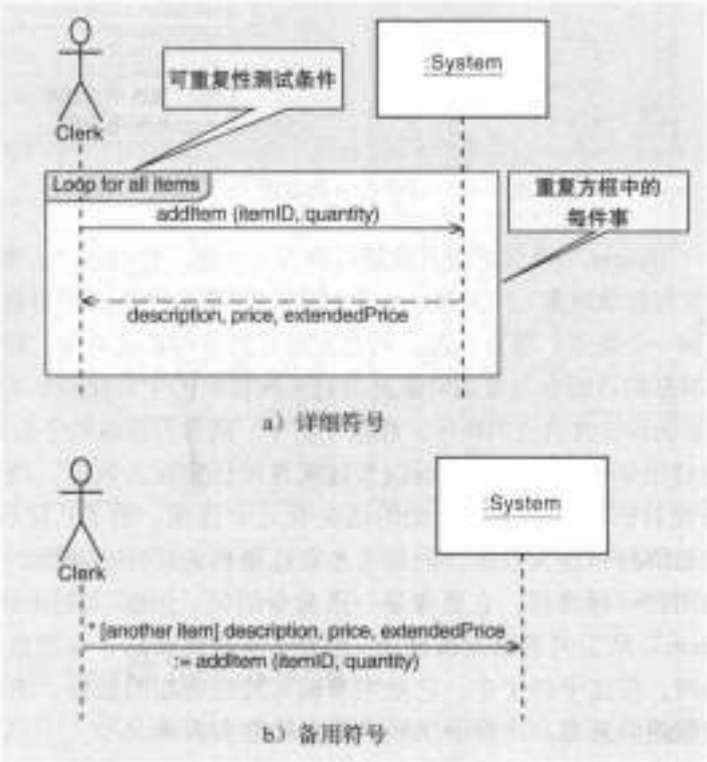


图 5-7　a 举例说明了显示这种重复操作的符号，b 显示了一个备用符号

一条消息的完整符号如下所示：

［真 / 假条件］返回值 := 消息名（参数列表）

消息的任何部分都可以被省略。简单来说，符号的组成说明如下：

- 星号（*）表示消息的重复或者循环。
- 方括号 [] 表示真 / 假条件。它只是对消息的检测。如果它的值为真，消息就发送，否则消息就不发送。
- 消息名就是对所需服务的描述。在虚线返回消息时，它被省略，而仅显示返回的数据参数。
- 参数列表（带有圆括号表示起始消息，没有圆括号表示返回消息）显示了消息传递的数据。

- 与消息处于同一线上（需要 :=）的返回值用于描述从目的对象返回到源对象的消息响应数据。

顺序图还使用两种另外的框架来描述处理逻辑，如图 5-8 所示。当一个消息或一系列消息是可选的或者是基于一些真 / 假条件时，那么就会使用图 5-8 中的**选择框架**。**备用框架**是和“如果 – 则 – 否则”逻辑一起使用的，如图 5-8b 所示。备用框架在这个图中表示如果一个条目应纳税，则添加销售税额，否则就要为销售添加一个免税代码。

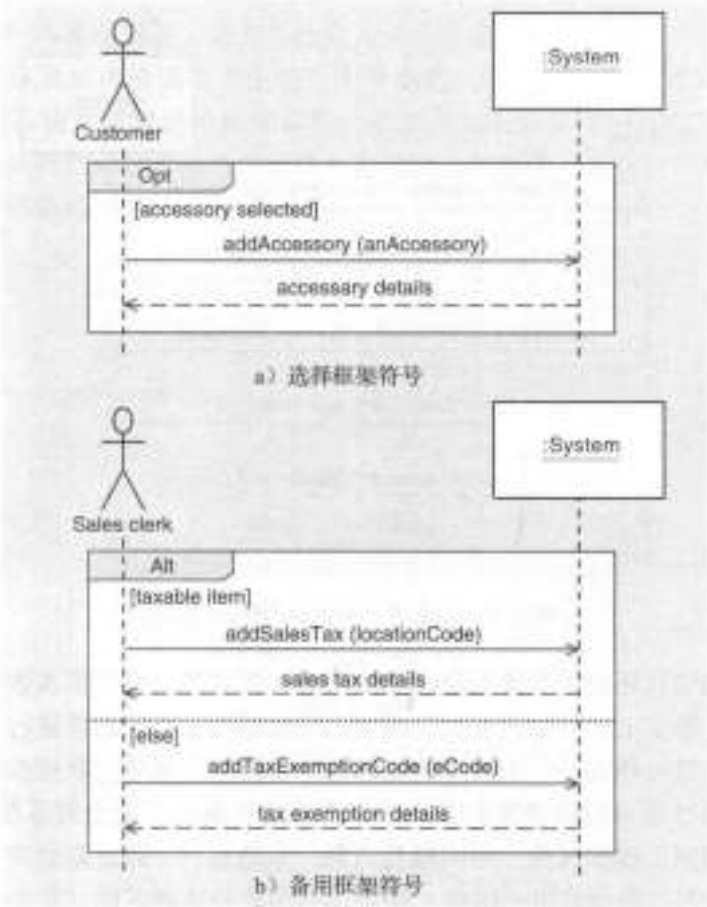


图 5-8　系统顺序图的选择框架和备用框架

5.4.2　开发系统顺序图

系统顺序图通常与用例描述联系在一起，以帮助记录用例中一个单独的用例或场景的详细信息。为了开发系统顺序图，你需要有用例的详细描述，可以是完全展开模式，也可以是活动图。这两个模型确定了用例中的活动流，但是它们并没有明确输入和输出。系统顺序图将提供输入和输出的详细说明。使用活动图的一个优点是很容易确定输入或输出在何时发生。输入和输出总是发生在活动图中的箭头从外部参与者到计算机系统这一阶段。

回顾图 5-4 中创建顾客账户的活动图。有两条泳道：顾客和计算机系统。在本例中，系统边界线是与顾客、计算机泳道之间的竖线一致的。

基于活动图的顺序图的开发可以分为以下四步。

1. **确定输入消息**。在图 5-4 中，有三个地方的工作流箭头穿过了顾客与系统之间的边界

线。在每一个工作流穿过自动化边界的地方都需要输入数据，因此同样需要消息。

2. 用先前介绍的消息符号来描述从外部参与者到系统的消息。在许多例子中，你都需要描述系统所需服务的消息名和需要传递的输入参数。图 5-9 所示为创建顾客账户用例的系统顺序图，说明了基于活动图的三个消息。注意，消息名能反映参与者向系统请求的服务：创建新顾客、输入地址和输入信用卡。其他名字也可以使用。例如，可以用创建地址来代替输入地址。重要的是消息名要能描述系统发出的服务请求，并且要以动名词格式命名。

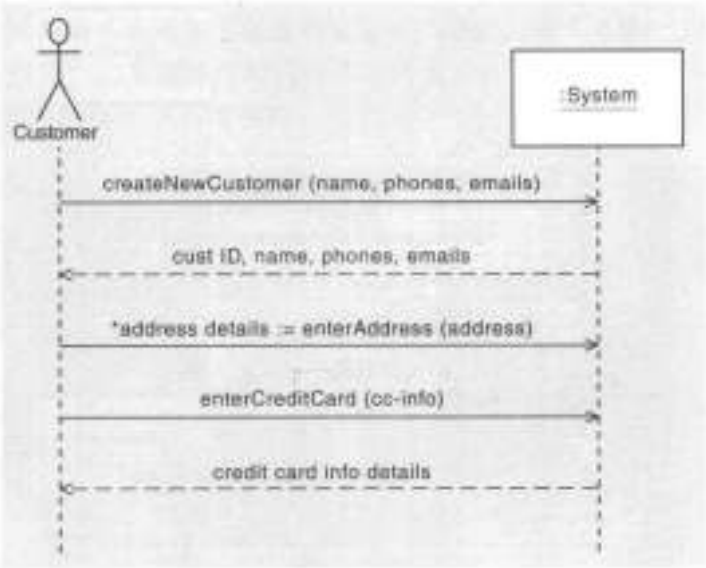


图 5-9 创建客户账户用例的顺序图

所需要的其他信息还包括每条信息的参数列表。要准确地确定哪些数据条目必须传递进来是比较困难的。事实上，开发人员会发现要确定数据参数通常需要进行数次迭代，才能找到一个正确而又完整的列表。一个重要的规则就是根据类图来确定数据参数。换句话说，类中的一些恰当的属性可以用作参数列表。仔细查看这些属性，加上对系统需求的充分理解，这些将有助于你找到正确的属性。刚刚提到的第一条消息——创建新顾客——参数包括顾客的基本信息，如姓名、电话和电子邮件。注意，当系统创建顾客时，它会安排一个新的顾客 ID，然后和其他顾客信息一起返回。

第二条消息——输入地址——参数需要确定完整的地址。通常，要包括街道地址、城市、州和邮政编码。这个系统顺序图简化了要作为参数显示地址的消息。

第三条消息——输入信用卡信息——基于活动图。这里的参数——信用卡信息——代表了账户编号、有效期限和验证码。

3. 在输入消息上确定并添加特定条件，包括迭代和真 / 假条件。本例中，对顾客需要的每个地址会重复发送输入地址消息。消息前会显示星号。

4. 确定并添加输出返回消息。记住，有两种方式可以显示返回消息：在消息上添加返回值或用虚线箭头表示独立的返回消息。活动图可以提供一些关于返回消息的线索，但是并没有一个标准的规则，当工作流中的转换箭头从系统出发到外部参与者的时候，不一定总会产生一个输出。图 5-4 中，从计算机系统泳道到顾客泳道有三个箭头。图 5-9 中，虚线上显示的是返回数据。注意，它们的名字中都有一个名词，这表明了返回的是什么。有时，没有输

出数据被返回。

记住，我们的目标是发现和理解，所以你应该与用户一起工作，以便准确定义工作流是如何运作的，以及什么样的信息需要作为输出被传递和提供。这是一个迭代过程，在这些图能准确反映用户需求之前，你可能需要多次修改。

让我们为运输商品用例开发一个系统顺序图，图 5-4 所示为这个用例的完全展开用例描述。注意，参与者在活动流中有五个被编号的步骤，因此图 5-10 所示的系统顺序图中有五条输入消息：获取下个订单、设置送货员、记录运输商品、记录延期订单以及获取运送标签。获取下个订单中不需要参数，因为系统会为下个要运送的商品返回信息。送货员是由参与者选择的，应该就是从表格或页面中的列表中选的，因此参数是送货员 ID。循环中有两条重复消息：记录运输商品和记录延期订单。在这个系统顺序图中，循环框架符号会被使用。最后，获取运送标签消息需要两个参数：包装大小和重量。系统使用相关信息，加上送货员和地址，就能提供运送标签并且记录成本。

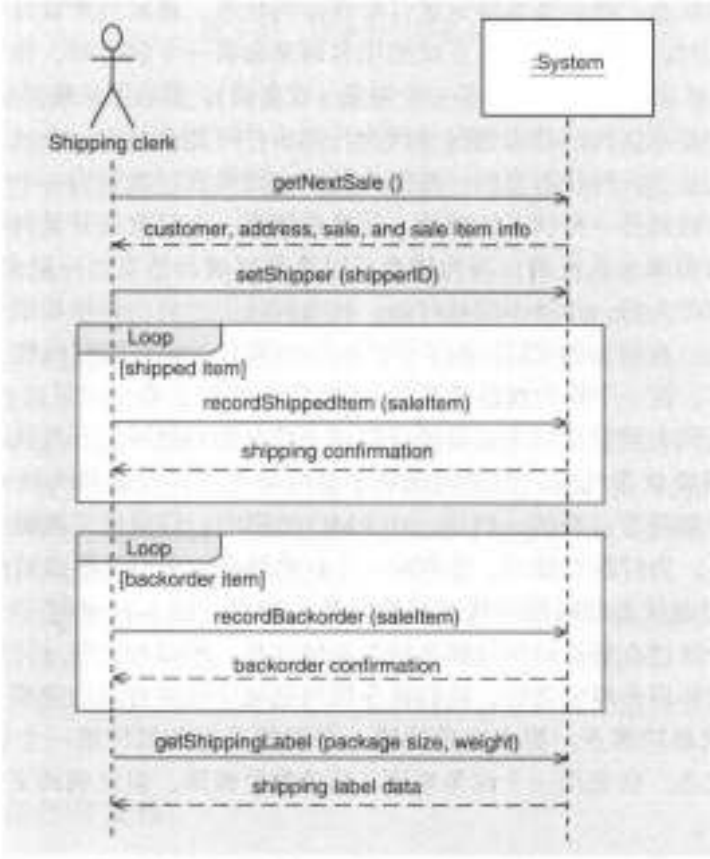


图 5-10　创建运输商品用例的顺序图

本章的开头部分已经解释了在面向对象方法中用于对新系统的处理方面进行描述的模型集。由文字叙述或活动图提供的用例描述给出了每个用例的内部步骤的详细信息。前提条件和后续条件描述有助于定义用例间的关联，也就是处理之前和之后什么必须存在。最后，系统顺序图描述了发生在用例中的输入和输出。这些模型一起为系统处理的需求提供了全面综合的描述，也为系统设计打下了基础。

我们已经很清楚地解释了用例，现在让我们关注下如何获得重要的对象状态信息。

5.5 状态机图——确定对象行为

有时对于一个计算机系统来说，维护问题域对象状态的信息是非常重要的。例如，顾客可能想要知道一个特定的订单是否已经发货，或经理想知道一个顾客的订单是否已经支付。因此，系统要能够跟踪顾客订单的状态。在定义需求时，分析员需要确定并记录哪些域对象需要状态检查，以及什么交易规则决定有效的状态条件。回顾 RMO 案例，一个交易规则的例子是顾客订单直到付款后才能发货。

现实世界对象的状态条件经常被称为对象的状态。准确地定义，对象的**状态**是在对象的生命周期中当其满足一些标准、执行一些行为或等待一个事件时所发生的状态情况。对于现实世界对象来说，我们将对象的状态和状态情况等同看待。

状态情况的命名规则有助于确定有效的状态。一个状态可能有一个简单状态情况的名字，如开启或正在修复。其他状态更加灵活，它们的名字由动名词或动词短语构成，如正在送货或正在工作。例如，一个特定的订单对象在顾客订购商品时生效。在创建之后，该对象处于增加新订单商品的状态，然后是等待运送订单商品的状态，最后当所有订单商品都运送完毕之后就是完成状态时。当你发现自己正试图用名词来命名一个状态时，你可能对状态或对象类有错误的理解。状态的名字本身不是一个对象（或名词），而是用来描述对象（或名词）的。

状态被描述为非永久性的状态情况是因为外部事件可能会打破一个状态或导致对象进入一个新状态。对象处于一种状态直到一些事件触发它转换或过渡到另一个状态。**转换**是指一个对象从一种状态转到另一种状态的活动。转换是使得一个对象离开某种状态而转为一种新状态的过程。状态是非永久性的，因为转换可以将其打破和结束。一般来讲，与状态相比，转换被认为是短期持久的，并且不能被打断。状态和状态之间的转换相结合为分析员获得业务规则提供了方法。在前面的 RMO 例子中，一个顾客订单在它能够转换为运输状态前必须首先经过付款状态。在一个称为**状态机图**的 UML 图中可以获取和记录这些信息。

我们可以为任何有着复杂行为或需要跟踪状态情况的问题域类开发状态机图。然而，并不是所有的类都需要状态机图。如果问题域中类的对象不存在必须为该对象控制程序的状态，则状态图就可能是非必要的。例如，在 RMO 类图中，订单类可能需要状态机图，而订单转换类则不需要。当付款完成时，会创建一个订单转换，不需要跟踪其他情况。

状态机图由代表状态的椭圆和代表转换的箭头构成。图 5-11 描述了一个简单的打印机的状态机图。因为通过有形条目学习状态机图更加容易，所以我们通过计算机硬件的几个例子开始学习。基础知识介绍完之后，我们将介绍问题域中软件对象的建模。状态机图的开始点是一个称为**伪状态**的黑点。黑点之后的第一个椭圆是打印机的第一个状态。在这个例子中，开始于关闭状态。状态由一个圆角矩形（几乎像是椭圆，但比椭圆更方一些）代表，状态名字处于椭圆内。

如图 5-11 所示，箭头离开关闭状态被称为*转换*。转换使得对象离开关闭状态转换为开启状态。当转换开始以后，它通过把对象带到一个称为**目标状态**的新状态中而完成。一个转换开始于一个从**初始状态**（先于转换的状态）指向目标状态的箭头，并且用一个字符串加以标记来描述转换的组成。

转换标签由以下三个部分的语法组成：

转换名称（参数，…）[判定条件]/行为描述

在图 5-11 中，转换名称是按下开启按钮。转换像一个被触发的触发器或发生的事件。名字应该反映触发事件的行为。在图 5-11 中，没有发送给打印机的参数。判定条件是［安

全盖关闭]。对于要发生的转换，判定条件必须为真。前向斜线将触发事件与行为或过程区分开来。**行为描述**表明在转换完成和对象达到目标状态之前必须经历的过程。在这个例子中，打印机在达到开启状态前将执行自检。

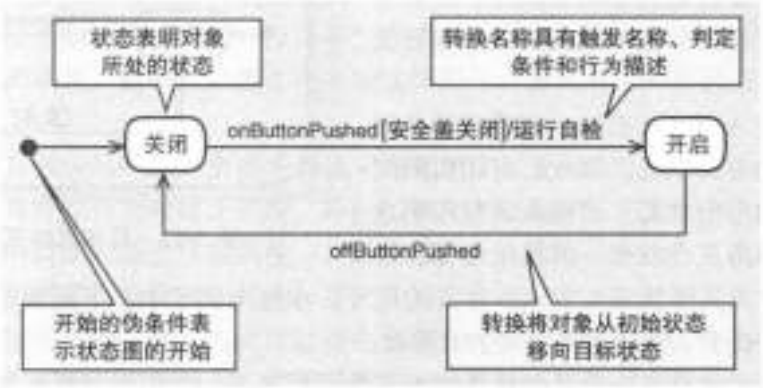


图 5-11　简单的打印机状态图

转换名称是触发转换和导致对象离开初始状态的消息事件的名字。注意，它的形式与系统顺序图中的消息非常类似。实际上，你会发现消息事件名称和转换名称使用几乎相同的语法。消息和转换之间还存在另一种关系，转换由到达对象的消息引起。转换名称的参数部分直接来自于消息参数。

判定条件是转换的限定条件或测试，它也是在转换发生之前必须满足的一个简单的真/假判定条件。对于一个转换，首先必须触发，然后判定条件必须为真。有时，一个转换只有一个判定条件而且没有触发事件。在那种情况下，触发是持续的，所以无论判定条件何时变为真，转换都会发生。

回想顺序图中也有类似的测试，称为真/假条件。这个真/假条件是位于消息发送端的测试，在消息可以被发送之前，这个真/假条件必须为真。相反，判定条件在消息的接收端。消息可能已经收到，但是只有当判定条件为真时才触发转换。这种测试、消息和转换的结合为定义复杂的行为提供了极大的灵活性。

行为表达式是在转换发生时执行的行为描述。换句话说，它描述将要执行的行为。转换标签中的三个组成部分（转换名称、判定条件和行为描述）中的任何一个都可以为空。如果转换名称或者判定条件为空，则自动为真。其中的任意一种都可能是复杂的，可以用 AND 和 OR 连接。

5.5.1　复合状态和并发性

在学习如何开发状态机图之前，我们需要介绍另一种状态：复合状态。在现实世界中，一个对象在同一时间处于多个状态是非常常见的。例如，当图 5-11 中的打印机处于开启状态时，也可能在做其他事情。有时在打印，有时空闲，当它第一次被打开时通常进行自我检查。所有这些情况都是在它开启的时候发生的，被认为是同时发生的状态。同一时间处于多个状态的情况称为**并发**或**并发状态**。表示并发的一种方式是同步线和并发路径，就像在活动图中完成的那样。因此，可以用同步线来分裂一个转换，这样一条路径到达开启状态，其他路径到达空闲状态、打印状态和自检状态。**路径**为一组有序的相互连接的状态和转换。

表示并发状态的另外一种方式是在其他高层状态中嵌套状态。这些高层状态称为**复合状态**。

复合状态表现更高层的抽象性，它包括嵌套的状态和转换路径。图 5-12 是图 5-11 的扩展，为打印机阐述了这一概念。打印机不仅处于开启状态，同时也处于空闲或工作状态。开启状态的圆角矩形被分为了两个部分。上面部分包括名称，下面部分包括嵌套的状态和转换路径。

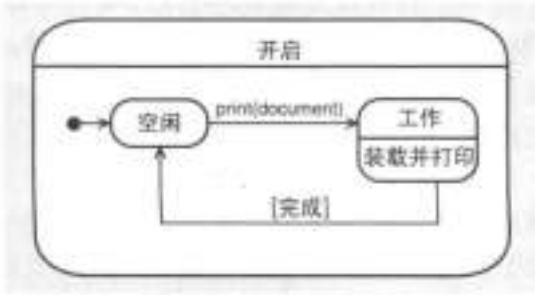


图 5-12　打印机的复合状态图

当打印机进入开启状态时，它将自动从嵌套黑点开始移向空闲状态。因此，打印机同时处于开启状态和空闲状态。当接收到打印消息时，打印机转换为工作状态，但是依然保持在开启状态。有时为工作状态也引入一些新的符号。在这个例子中，下面部分包括行为描述，也就是当打印机处于工作状态时所发生的活动。

我们可以通过允许在一个复合状态中有多条路径来进一步扩展负荷状态和并发的概念。也许一个对象有一组完整的活动并发的状态和转换（多路径）。为记录一个简单对象的并发多路径，我们画了一个复合状态，它的下面部分被分成多个部分，每一个部分对应一个行为并发路径。例如，假定打印机有一个输入口来放纸张。打印机工作时在空闲和工作两个状态之间循环。我们可能想要描述两条独立路径：一条描述纸张输入口的状态，另一条描述打印机的状态。第一条路径有空、满和不满三种状态。第二条路径包括空闲和工作两种状态。这两条路径是独立的，一个组成部分内状态间的转换和另一个组成部分内状态间的转换是完全独立的。

如前所述，有两种方法记录这种并发行为。首先，可以用一条由一条路径变为三条路径的并发线表示。第二，可以用复合状态。图 5-13 在图 5-12 的基础上扩展了打印机的例子。在这个例子中，在复合状态中有两个并发路径。上面的并发路径描绘打印机纸张输入口。这两条路径是完全独立的，并且打印机在每条路径里的状态和转换都是独立的。当按下关闭按钮时，打印机离开开启状态。很显然，当打印机离开开启状态时，它也就会离开这个嵌套状态的所有路径。打印机是否处于某状态或在转换中并没有关系。当按下关闭按钮时，所有活动结束，打印机退出开启状态。学习了状态机图的基本符号后，我们接下来介绍怎样开发状态机图。

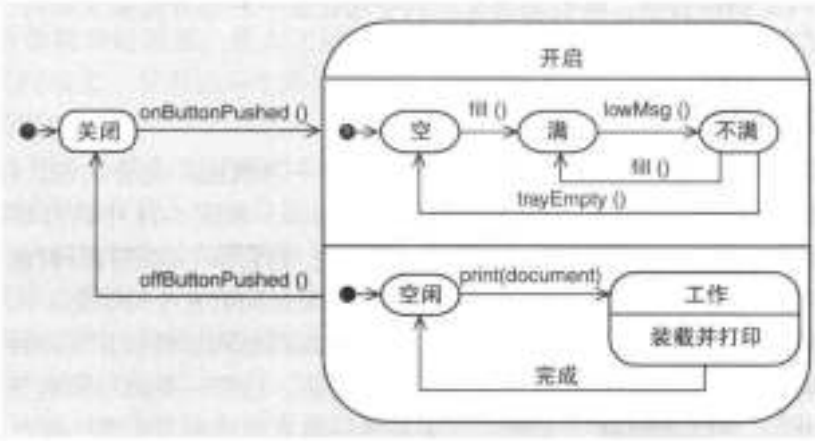


图 5-13　打印机在开启状态下的并发路径

5.5.2 开发状态机图的规则

状态图的开发需遵循一组规则。这些规则能帮助你为问题域类开发状态图。通常，开发状态机图的主要挑战是为对象确定正确的状态。把自己想象为对象本身会有所帮助。假设为顾客很容易，但是如果说“我是一份订单”或“我是一次运送”“我如何产生”“我处于什么状态”就相对困难些。然而，如果你能开始这样思考，那么对于开发状态机图将会有很大帮助。

对于新分析员来说另一个主要困难是确定和处理带有嵌套的复合状态。通常这个困难的主要原因是在思考并发行为时缺乏经验。最好的解决办法是记住开发状态机图是一个迭代行为——比开发其他类型的图更具复杂性。分析员很少一次就能得到一个正确的状态机图。他们通常画出状态机图并反复修改。另外，要记住当定义需求时，你只知道对象行为的大概情况。就像开发详细顺序图一样，在设计过程中你将有机会优化和修正重要的状态机图。

最后，不要忘记询问异常情况，尤其是看到验证和检查字样时。通常，有两种离开某状态的转换需要验证：一个是接受，一个是拒绝。

下面的步骤能帮助你开发状态机图。

1. **回顾类图并选出需要状态机图的类**。记住，只跟踪那些有多个对系统非常重要的状态的类。然后，从看起来有最简单的状态图的类开始，如后面要讨论的 RMO 的订单商品类。

2. **对于组中每一个选出的类，列出你能确定的所有状态**。在这一点上，就是简单的头脑风暴。如果你正在为一个团队工作，那就与整个小组进行集体讨论。记住这些状态必须能反映在软件中展现的现实世界对象的状态。有时，考虑物理对象、确定物理对象的状态、把这些合适的状态转换为相应的系统状态或状态情况，这样做是很有帮助的。考虑对象的生命周期也非常有帮助。它们在系统中如何产生？何时以及如何将它们从系统删除？有活动状态吗？有非活动状态吗？有等待状态吗？考虑对对象施加的活动或由对象完成的活动。通常，当这些活动发生时对象处于特定状态。

3. **通过确定导致对象离开特定状态的转换来开始建立状态机图片段**。例如，如果订单处于准备运输状态，那么像“开始运输”这样的转换将导致订单的那个状态。

4. **按正确顺序将这些状态－转换组合起来**。然后，将这些组合集合成大片段。在将这些片段集合成为“大路径”的过程中，很自然地会开始为对象寻找自然的生命周期。继续通过组合这些片段来构建更长的路径。

5. **回顾这些路径并找出独立且并发的路径**。当一个项目可以同时处于两种状态时，有两种可能。这两种状态可能处于独立的路径，就像打印机例子中的工作和满负荷。这发生在状态和路径独立时，且一个改变不影响另外一个。另外，一个状态可能是复合状态，因此这两个状态应该是嵌套的。为复合状态确定候选者的一个方法是确定它是否与其他几个状态并发，以及其他这些状态是否依赖于初始状态。例如，当打印机处于开启状态时有几个其他状态和路径可以发送，这些状态都依赖于打印机处于开启状态。

6. **查找其他的转换**。通常，在第一次迭代中，会错过一些可能的状态－转换－状态组合。确定它们的一个方法就是将状态配对，并询问状态之间是否有有效的转换。从两个方向检验这些转换。

7. **用合适的消息事件、判定条件和行为描述扩展每个转换**。包括每个状态都有合适的行为描述。很多工作可能在状态机图片段构建阶段已经完成。

8. **回顾并检查状态机图**。通过以下方法检查每一个状态机图。

a. 确保状态确实是类对象的状态。保证状态名确实描述了对象的状态而不是对象本身。

b. 从产生到被系统删除，跟踪对象的生命周期。确保所有可能的组合都已覆盖到，确保状态机图的所有路径都是正确的。

c. 确保状态机图覆盖了所有异常情况以及正常的行为流。

d. 再次检查并发行为（多路径）和可能的嵌套路径（复合状态）。

5.5.3　开发 RMO 状态机图

我们通过开发两个 RMO 的状态机图来实践这些步骤。步骤一是回顾域类图并选择可能需要被跟踪状态的类。在这个例子中，我们选择订单和订单商品类。我们假定顾客想知道他们订单的状态和订单中单独商品的状态。其他可开发状态机图的类有：库存商品，用来跟踪有货或缺货；运输，用来跟踪到达情况；顾客，用来跟踪活动和非活动的顾客。

开发订单商品的状态机图

开发订单商品的状态机图的第一步是确定可能涉及的状态条件。一些必要的状态是准备运输、订单延迟和运输。这时一个有趣的问题产生了：订单商品可以部分运输吗？换句话说，如果一个顾客订购了 10 件一样的商品，而库存只有 5 件，那么 RMO 要先运输 5 件而把剩余 5 件放在延迟订单中吗？要看清这个决定的几个分支。系统和数据库要设计成能够跟踪和监视详细信息以支持这种能力。RMO 的域类图表明一个订单商品可以与 0 个（还没运输）或 1 个（全部运输）运输相关。基于当前的这些描述，不允许定义订单商品的部分运输。

这是体现建模优势的另外一个例子。如果我们没有开发状态机图模型，这个问题可能永远也不会提出。开发详细模型和图是系统开发者所执行的重要活动之一。它使得分析员提出基础问题。有时，新的系统开发员认为建模是在浪费时间，尤其是对小系统而言。然而，在写程序之前，充分理解用户需求从长远角度来看是省时间的。

第二步是为每一个状态确定离开转换。图 5-14 是一个表，这个表列出了已经定义的所有状态，以及每个状态的离开转换。表中加入了一个额外的状态——新增加——它覆盖了当订单中加入新商品时发生的所有情况，但是订单还没有完成或付款，所以这个商品并没有准备运送。

状态	导致离开状态的转换
新增加	新增完成
准备运输	运输商品
订单延迟	商品到达
运输	没有定义离开转换

图 5-14　订单商品的所有状态，以及每个状态的离开转换

第三步是将成对的状态－转换组合为片段并按正确的顺序构建状态机图。图 5-15 所示为部分完成的状态机图。订单商品的对象从开始到结束的活动流都很明显。然而，至少有一个转换被忽视了。应该有一些路径允许进入订单延迟状态，所以我们认为第一次开发的状态机图还需要修改。我们将马上进行修正。

第四步是查找并发路径。在这个例子中，没有出现一个订单商品对象可以同时处于两个

状态的情况。当然，因为我们选择从简单状态图入手，这也正是我们所期待的。

第五步是查找其他转换。这一步是我们充实其他必要转换的过程。第一个就是从新增加到订单延迟的转换。接下来检查每一对状态，看看是否有其他可能的组合。特别地，查找反向转换。例如，订单商品可以从准备运输状态到订单延迟状态吗？当运送员发现仓库没有足够的货物而系统却显示有足够货物时，会发生这种情况。其他反向循环，如从运输状态到准备运输状态，或者从订单延迟状态到新增加状态都是没有意义的，也没有包括进来。

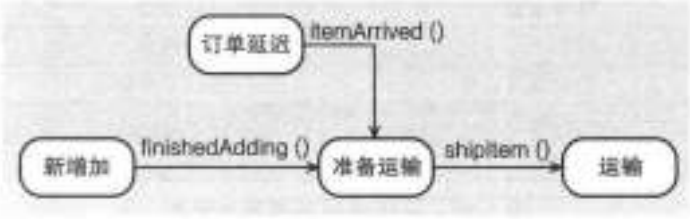


图 5-15　订单商品的部分完成的状态机图

第六步是用正确的名称、判定条件和行为描述完成所有的转换。增加了两个新的转换名称。第一个是从开始的黑点转换为新增加状态。这个转换导致创建了一个新订单商品，系统术语是实例化。它被赋予与消息进入系统相同的名称："增加商品"。最后一个转换是将订单商品从系统中移除。这个转换是从运输状态到最后的带圆圈的黑点，这个带圆圈的黑点就是最后的伪状态。假设当其从系统中删除时会存档备份，所以这个转换命名为"存档"。

为转换增加行为描述是为了表示由对象引起或对象执行的一些特殊行为。在这个例子中，只需要完成一个行为。当一个商品从准备运输到订单延迟状态时，系统要引发一个新的对供应商的购买订单来购买更多的商品。所以，在"标记延迟订单"转换中，行为描述表示为创建一个购买订单。图 5-16 所示为订单商品的最终状态机图。

图 5-16　订单商品对象的最终完成的状态机图

第七步是检查并测试状态机图，也就是质量评审阶段。总会有人想试图省略这一步，然而一个好的项目经理会保证系统分析员有时间来对他们的模型进行快速的质量评审。这样的项目走查（见第 2 章）是非常合适的。

开发订单状态机图

订单对象比订单商品对象稍微复杂一些。在这个例子中，你将会看到状态机图的一些其他特征，这些特征支持更复杂的对象。

图 5-17 所示为一个表，表中列出了定义的状态和离开转换，这些在第一次迭代中似乎是需要的。从上往下看，状态反映了订单的生命周期——例如状态条件。首先，一个订单产

生并准备增加商品——开启增加商品。RMO 的顾客表示他们想在 24 小时之内将订单保持在该状态以便增加更多的商品。当所有商品增加完之后，订单准备运输。接下来将运输并处于运输状态。这时，运输和等待延迟订单之间如何联系还不是很清楚。这个关系在状态机图的开发过程中会被挑选出来。最后，订单处于运输状态，付款完成之后就会关闭。

状态	离开转换
开启增加商品	订单完成
准备运输	开始运输
运输	运输完成
等待延迟订单	延迟订单到达
已运输	付款完成
关闭	存档

图 5-17　订单的状态和离开转换

第三步中，片段的建立和组合会生成初步的状态机图（见图 5-18）。状态机图由那些对于多数部分来说都是正确的片段构成。然而，我们注意到在等待延迟订单中存在一些问题。

图 5-18　订单的初步状态机图

在分析之后，我们认为运输状态和等待延迟订单状态是并发的。另外一个称为正在运输的状态也是需要的，在这个状态下负责运输的员工正在运输商品。表示订单生命周期的一种方法是当运输开始时将其置于运输状态，这也是它进入正在运输状态的时间点。订单可以在正在运输状态和等待延迟订单状态之间循环。离开复合状态只发生在正在运输状态，也处于运输这个复合状态中。很显然，离开内部状态后，订单也就会离开运输这个复合状态。

当我们执行完第四、五、六步之后，发现要增加新转换。需要从初始伪状态创建转换。此外，转换还必须显示商品何时增加、何时运输。通常，我们将这些循环活动置于那些离开某状态并回到同一状态的转换中。在该例中，这个转换称为“增加商品”。注意，它如何离开开启增加商品状态又回到该状态。图 5-19 所示为该完成水平下的状态图。

为问题域对象开发状态机图的好处就是帮助你获取和厘清业务规则。从状态机图中，我们可以看出当订单处于开启增加商品状态时，运输不能开始；当订单处于准备运输状态时，不能增加新的订单商品；订单不能考虑为运输状态直到所有商品都被运输。如果订单处于运输状态，我们就知道它要么处于工作状态要么处于等待延迟订单状态。

通常，仔细建模的好处就是能帮助我们获取对系统需求更好的理解。我们现在来看一个

大图，看看不同的模型是如何协助集合成一个整体的。

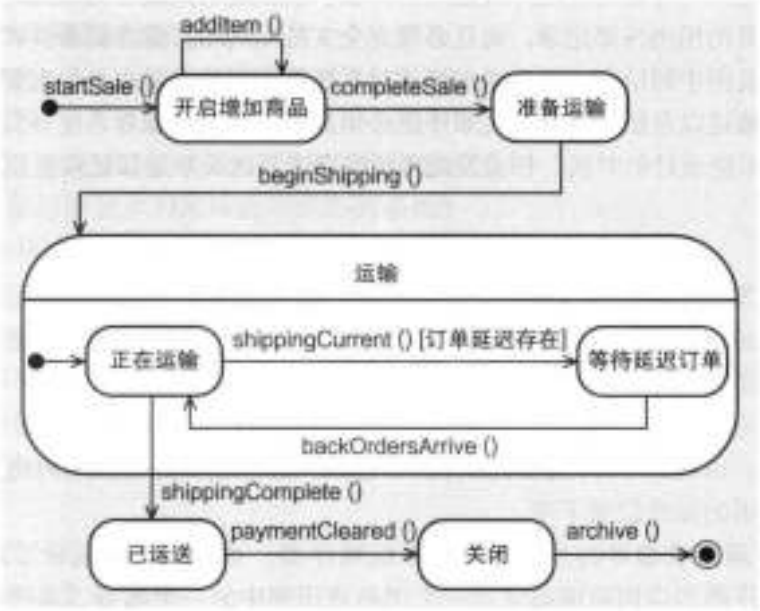


图 5-19 订单的完成水平下的状态机图

5.6 需求模型的集成

本章描述的图表使得分析员可以完整地描述系统需求。如果你正在使用瀑布系统开发生命周期方法开发一个系统，那么在继续设计之前，你需要开发一整套图来说明所有系统功能需求。然而，因为你在使用迭代方法，所以只需要构造一次特定的迭代所必需的图表就可以了。一个完整的用例图对于理解新系统的总体规模是很重要的。但是包含在用例描述、活动图和系统顺序图中的支持细节只需为特定迭代中的用例完成。

域模型类图是一个特殊的案例。域模型类图更像完整的用例图，对整个系统而言它应该尽可能完整，如第 4 章中 RMO 所示的一样。系统问题域类的数量为系统总体规模提供了一个额外的指示。许多类的修改和实际执行将等到以后的迭代中进行，但是域模型应该基本完成。域模型对于确定新系统所需要的所有域类是必要的。域模型还可以用于数据库的设计。

通览本章，你可以看到一幅图的构建要依靠另一幅图所提供的信息来完成。你也同样看到了新图的开发通常有助于精简和纠正先前的图。你应该也注意到了详细图的开发对于充分理解用户需求至关重要。图 5-20 说明了面向对象开发中需求模型间的主要关系。左边的用例图和其他图用来获取新系统的程序。类图和其他依赖图获取新系统的类信息。实心箭头表示主要的依赖关系，虚线箭头表示次要的依赖关系。这种依赖性通常从顶部流到底部，但是一些箭头是双向的，它在两个方向都产生影响。

图 5-20 面向对象需求模型之间的关系

注意，用例图和问题域类图是两个主要的模型，依靠这些主要模型，其他大图可以从中获取信息，因此应该尽可能完整地开发这两种图表。以叙述格式或者活动图的形式进行的详细描述能提供很重要的用例内部记录，而且必须完全支持用例图。像前提条件和后续条件这样的内部描述要使用类图中的信息。这些详细描述对系统顺序图的开发也是很重要的。所以，关于特定用例的详细描述以及活动图和系统顺序图必须具有一致性。随着系统开发的进行，特别是当你开始做详细系统设计的时候，你会发现理解模型之间的关系是保证模型质量的重要因素。

本章小结

信息系统开发的面向对象方法有一整套的图表和文本模型，一起用来记录用户的需求和定义系统需求。通过使用域模型类图和状态机图来建立问题域和用例图、用例描述或活动图和系统顺序图的模型，以此来建立用例的模型，从而说明这些需求。

用例的内部活动首先由内部的活动流描述。有可能有几个不同的内部流，代表同一个用例的不同场景。因此，一个用例可能有多个场景。这些细节要么以用例描述的形式记录下来，要么以活动图的形式记录下来。

提供用例处理需求细节的另一个图是系统顺序图。系统顺序图记录了系统的输入和输出。每个系统顺序图的应用范围通常是一个用例或用例中的一个场景。系统顺序图的组成部分有参与者（和用例中确定的参与者一样）和系统。系统被看成是一个黑盒子，因为不需要记录内部处理。代表输入的消息由参与者发送给系统。输出消息由系统返回给参与者。消息的顺序是自顶而下的。

在定义需求时，域模型类图会得到进一步修改。类图中代表的域对象也是要研究和建模的需求的一个方面。状态机图用来建立对象状态和用例发生的状态转换的模型。本章中讨论的所有模型都是内部相关联的，一个模型中的信息可以解释为其他模型的信息。

复习题

1. 那些模型更加详细地描述了用例？
2. 哪两种 UML 图是用于建立域类的？
3. 通过使用活动图能建立用例描述哪部分的模型？
4. 解释用例和场景的区别。举出一个有几个可能场景的用例。
5. 列出完全展开用例描述的部分或组件。
6. 比较前提条件和后续条件。
7. 比较前提条件和异常情况。
8. 比较用例中的业务过程和活动流。解释怎么使用活动图为两者建立模型。
9. 系统顺序图的目的是什么？系统顺序图中使用的符号有哪些？
10. 开发系统顺序图的步骤是什么？
11. 在参与者要求系统开始更新特定产品信息这个过程时，写出一份从参与者到系统的完整的系统顺序图消息。
12. 在用例的持续过程中，用来代表对象扩展的系统顺序图符号的名称是什么？
13. 在顺序图中显示返回值的两种方式是什么？
14. 在顺序图中显示重复的方式有哪两种？
15. 顺序图中会使用哪三种框架？

16. 顺序图中代表真 / 假条件的符号是什么？
17. 什么是消息参数？
18. 列出开发系统顺序图的主要步骤。
19. 什么是对象状态？
20. 什么是状态转换？
21. 在考虑需求时，状态和状态转换对理解哪种图很重要？
22. 哪种 UML 图是用来显示对象状态和状态转换的？
23. 列出组成转换描述的部分？哪个部分是可选的？
24. 什么是复合状态？它的作用是什么？
25. 术语“路径”是什么意思？
26. 判定条件的目的是什么？
27. 确定在本章中解释过的模型及它们之间的关系。

问题和练习

1. 根据下列描述进行操作：
   （1）为每个场景开发活动图。
   （2）为每个场景开发完全展开描述。

   高质量建筑供应系统有两类顾客：承包商和普通大众。针对不同人的销售方法是不同的。

   当承包商购买住宅时，销售人员把他们带到承包商专用柜台前。接待员向系统输入承包商的姓名。系统就会显示承包商的信息，包括他现在的信誉。接着接待员为承包商建立一个新的销售单据，然后接待员检索卖出的所有项目。系统自动发现每一项的价格并把它们累加到单据上。购买的最后，接待员指示销售结束。系统比较总金额和承包商现在的信用卡余额，如果信用卡余额大于总金额，则结束销售。系统为所有项目建立电子单据并且承包商的信用卡余额将减去销售的金额。一些承包商喜欢保留他们的购买记录，所以他们需要打印出详细单据。另外一些承包商则对打印单据不感兴趣。

   普通大众的消费情况会简单地输入收银机，然后当项目确定后打印一张收据。付款方式可能是现金、支票或信用卡。办事人员必须输入付款方式以确保收银机在交班时结算。当选择信用卡付款时，系统打印信用卡单据让客户签字。
2. 根据下列描述，针对一个汽车保险系统中将一辆新车加入一个已有保单中的用例，设计活动图或完整的开发描述。

   顾客打电话给保险公司的员工，并提供他的保单号。员工输入这个信息，系统显示基本的保单。然后这个员工检查信息，以确保保险费通用以及保单有效。

   顾客给出要添加汽车的品牌、模型、年份和车辆识别代号（VIN）。员工输入这些信息，系统验证这些数据是否有效。然后，顾客选择期望的保额类型及每种类型的数量，员工输入这些信息，系统逐一记录并根据保单限制验证所请求的数量。输入所有的保额后，系统验证保额总数，包括保单上的其他汽车。最后，客户必须确定所有的驾驶员及他们驾驶汽车的时间比例。如果有一个新驾驶员加入，则调用另一个用例“增加新驾驶员”。

   整个过程的最后，系统更新保单，计算新的保险费，打印新的保单说明，邮寄给保单所有人。
3. 给出前面的汽车保险系统中的类和关系列表如下，针对“将一辆新车加入一个已有保单

中”这个用例，列出其前提条件和后续条件。

系统中的类：

- 保单
- 被保人
- 保险车辆
- 保险总额
- 标准保险总额（按税率分类列出标准保险总额的价格）
- 标准车辆（列出已有的各类车辆）

系统中的关系：

- 保单有被保人（一对多）
- 保单有保险车辆（一对多）
- 车辆有保险总额（一对多）
- 保险总额是标准保险总额的一种
- 车辆是标准车辆的一种

4. 根据问题 1 的描述和活动图设计一个系统顺序图。
5. 根据问题 2 的描述或活动图设计一个系统顺序图。
6. 回顾图 5-21 所示的移动电话状态图，然后回答下列问题。（注意，这种电话具有不同于普通电话的特殊特点，请基于状态图回答。）

（1）打开电话会发生什么？

（2）电话打开后会进入什么状态？

（3）关闭电话的三种途径是什么？

（4）在活动（通话）过程中能关闭电话吗？

（5）电话如何才能到达活动（通话）状态？

（6）当有人在通话时可以接通电话吗？

（7）当有人通话时手机可以进入充电状态吗？请解释哪些活动是允许的，哪些是不允许的。

（8）哪些状态是与其他状态相并存的，列出一个两行表显示并存的状态。

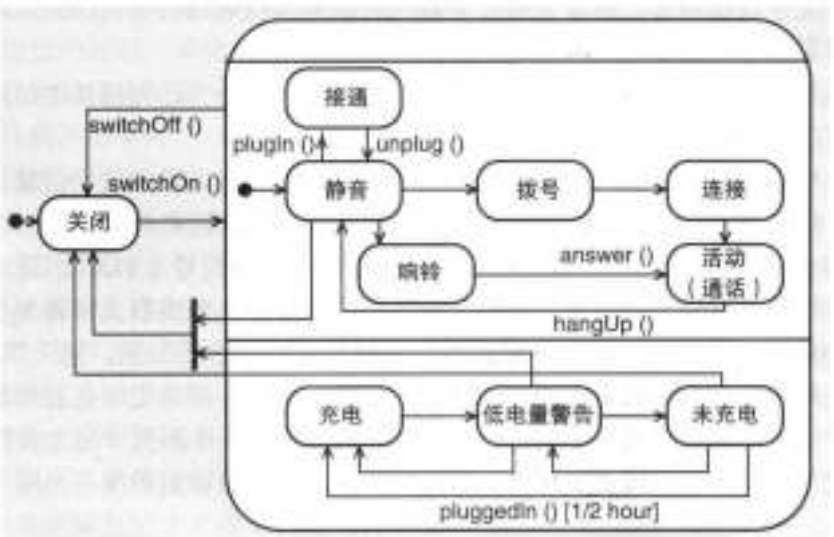


图 5-21　移动电话状态机图

7. 基于以下对包裹运送公司的描述，首先确定所有的状态和存在的转换，然后开发状态机图。

当从顾客处接收包裹后运送首次被确认。当包裹处于系统中时，它就被认为是活跃的并在运送中。每当它到达一个检查点，如到达中间目的地，将进行检索并创建一个记录来表明检查的时间和地点。当包裹装上运送卡车的时候状态发生变化，它仍是活跃的，但是它也被认为处于运送等待状态。当包裹运送完成后，状态会再次改变。

有时，运送的目的地在公司服务范围之外。在这些情况下，公司要与其他运送服务公司联合工作。当包裹移交到另一个运送商时，包裹标记为移交。在这种情况下，新运送商将记录跟踪号码（如果提供的话）。公司要求新运送商在包裹送达之后提供状态改变通知。

不幸的是，有时包裹会丢失。在这种情况下，包裹在两周内会保持活动状态，但是会打上暂时丢失标签。如果两周内包裹还没找到就认为丢失了。这时候，客户可以启动丢失包裹程序来挽回损失。

8. 定位一家在你附近的软件开发公司。这个公司是咨询公司或者有许多信息系统专员，它有比较严格的软件开发方法。通过一次会面来确定他们使用的开发方法。许多公司仍然在使用面向对象方法和传统结构化技术相结合的方法。在另一些公司中，一些项目使用传统方法，而另一些使用面向对象方法。找出这个公司使用哪种建模方法来为需求说明服务。比较本章所学到的技术和你的发现。

扩展资源

Grady Booch, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

E. Reed Doke, J. W. Satzinger, and S. R. Williams, *Object-Oriented Application Development Using Java*. Course Technology, 2002.

Hans-Erik Eriksson, Magnus Penker, Brian Lyons, and David Fado, *UML 2 Toolkit*. John Wiley & Sons, 2004.

Martin Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (3rd edition). Addison-Wesley, 2004.

Philippe Kruchten, *The Rational Unified Process: An Introduction* (3rd edition). Addison-Wesley, 2005.

Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process* (3rd edition). Prentice Hall, 2005.

Object Management Group, *UML 2.0 Superstructu Specification*, 2004.

第三部分

Introduction to Systems Analysis and Design: An Agile, Iterative Approach, Sixth Edition

系统设计的要点

第 6 章　设计与设计活动的基本要素

第 7 章　设计用户界面和系统界面

设计与设计活动的基本要素

学习目标

阅读本章后，你应该具备的能力：

- 描述系统分析与系统设计之间的区别。
- 解释每个主要的设计活动。
- 描述主要的硬件和网络环境的选择。
- 描述不同的托管服务。

开篇案例　Wysotronics 公司的技术决策

当 James Schultz 与他的员工走向会议室准备开会时，他就在思考自己的这份新工作。Schultz 成为一家中型供应商的副总裁和信息主管已经有一年时间了，他负责管理向几个大型电子公司（如三星和索基）供应电子元器件。James 所在的公司（Wysotronics 公司）从事这项生意已经有许多年了，但是公司内部的计算机系统最近出现了一些问题。James 负责修复这些问题。

在开始工作后不久，James 发现系统的好几个部分是能正常运作的，但是基础设施却因为没有连接其他计算机和网络而比较混乱。就企业而言，要有财务系统和人力资源系统，这两者都是桌面客户端 / 服务器系统，托管于本地网络计算机上，属于财务部门。

工程部门有自己的数据库和网络计算机，用于几个拥有密集计算需求的复杂工程系统。工程师本地的桌面系统配备了最新的设备和软件，服务器也是大容量的，它有很大的数据仓库来存储所有的工程文档和图片。

市场营销和销售也有基于他们自己网络服务器的系统，这个系统也是能连网的。销售人员与制造业和装配厂密切配合，以此来确保及时发货，同时他们也要经常拜访 Wysotronics 公司的客户。这是他们为确保客户对进度安排、发货情况和质量满意而进行的工作，并且在拜访客户的途中，他们也希望能连接销售和产品数据库。但是，他们使用的服务器不是很稳定，而且会不断产生问题。

也许最大的问题是供应链管理系统。Wysotronics 公司有一个很大的制造工厂、一个装配工厂和几个需要连接到库存与供应链系统的供应商。当前的底层结构没有足够的能力来给这些设备和供应商提供最新信息。

今天的会议是计划和部署公司系统总体底层结构的众多会议中的一次。正当他走进房间时，William Hendricks 就和 James 打招呼，William 将要做一个总结过去决策和展望未来方向的报告。

James 说："你好，Bill。今天你会为我们提出一些新建议吗？你的研究中有惊奇的发现吗？"

Bill 回答说："没有惊奇的发现。但是你会很高兴地知道我们的研究证实了你最近做出

的相关决策。我们正在为公司提供比以往更好的服务，并且我们所花费的成本相比以往来说是最少的。我只是对关于怎么调整底层结构来提供更好的服务有几个建议。"

很显然，Bill 对他的研究成果很有信心。

James 问道："在我们开始之前，你能跟我说下节省成本的主要想法吗？"

"当然！正如你所知道的，我们决定为所有的供应链和产品需求创建一个使用因特网的虚拟专用网络。我们动用所有的计算机来支持系统成为一个托管平台。我们仍会有服务器，但是我们已经与托管公司签订了服务协议来管理所有的操作系统、连接和网络维护。这就能让我们集中精力于软件本身而不用担心要使用高价的私人环境或连接。而且我们不用为了一个更大的数据中心额外投资一幢建筑物。此外，服务水平也是惊人的。因为工作方式的转换，我们有几乎 100% 的工作时间。我们工厂里的员工都很高兴能在任何时候从他们的供应商那里查到库存量和运输数据。"

James 说："太棒了！这是个好消息！你要对我们的市场营销和销售系统提出什么建议？"

Bill 回答说："就像你了解的那样，这是一个以 Web 为基础的系统。它没有产品系统的安全需求，但是它需要能广泛地利用。我们的研究表明可以通过能提供云计算的托管公司来部署系统。我们可以选择与本地供应商继续合作，也可以选择使用过去合作过的其他公司。我认为其他公司将会给我们一些价格上的让步并且仍然能提供好的服务。"

"听起来很棒！" James 说，"我很有兴趣听到更多细节。我能假定你也已经制定了一些迁移计划来推进系统吗？"

"当然可以。我在这方面已经开始了相关工作。而且我认为你会满意最终成果的。"

6.1 引言

前面章节描述了与发现和理解用户需求的主要部分有关的活动和决策——换句话说，就是分析活动。本章将重点讨论解决方案系统。在分析阶段，主要讨论理解系统是做什么的（如需求），然而在设计阶段，主要讨论解决方案（如说明系统如何构建以及新系统的结构组件会是什么）。

开发新手经常会问的一个问题是："在一个真实的项目中，何时实施这些任务？"但是，这个问题没有单一的回答。许多项目一开始要做的就是制定一些设计决策，尤其是在公司已经有了一个强大的技术结构时，关于部署环境的决策就很重要。对于其他项目，新系统可能是组织新推动力的结果，因此这些决策是完全开放的。然而，对项目团队来说，在项目早期就开始考虑这些重要问题然后在需求被定义后制定初步决策是很正常的情况。本章及下一章讨论面向解决方案的设计主题，然而，你不应该直到理解问题之后才尝试提出解决方案。

这是讨论设计的几个章节中的第一章。本章将简要地描述所有设计活动并详细讨论第一个活动（部署环境）。后面的章节将探索其他的设计活动，并且详细解释用于系统设计的各种模型和技术。

6.2 设计要素

在第 1 章，定义的系统设计是一些活动，这些活动能使项目团队详细描述解决需求的系统。很显然，系统的很多方面都需要设计。任何复杂的人工制品的设计都需要详细的设计文档。例如，要考虑建立邮轮或商用飞机必需的所有组件。对于商用飞机，设计范围包括基本

形状、飞机主要子系统的大小，如机械系统、液压系统和电子系统，一直到最微小的细节，如机翼的形状、座位的大小和形状、座舱显示的位置以及控制设备，甚至还有外部计量装置。不得有任何疏漏。没有多年的经验和广博的知识基础将无法设计现今的商用飞机。

如今正在设计和构建的软件应用程序同样是很复杂的，同时也只有在底层结构到位、有大量的知识基础和一系列开发工具的情况下才有可能做到。即使有了所有的可用工具，设计并构建软件应用程序仍是一个困难且复杂的过程。

本节首先探究设计的一些不同方面和层次。接下来，我们会从计算机应用程序系统的更高层次视角来观察设计中必须要包含的事物。在本节的后面，我们会确定由设计过程提供的文档和产品的类型。最后，我们回顾一下什么信息和文档能作为设计的输入。

6.2.1 什么是系统设计

系统设计实际上就是一个桥梁过程。系统分析的主要目的是完整地理解组织的信息需要或需求，同时也记录好一系列说明中的那些需求。软件构建的目的是建立能满足那些需求的系统。系统设计就是能把我们从需求带向解决方案的桥梁。系统设计的目的是定义、组织和构建最终解决方案系统的组件，并且这些组件能作为构建的蓝图。思考系统设计的另一种方式是鉴于分析告诉我们解决方案需要做什么，因而设计描述了如何部署和构建系统。

6.2.2 设计的主要组件和层次

如今，信息系统是在一定范围的物理设备中部署的——从单独的计算机和小型移动数码设备到局域网的计算机再到大型的分布式联网计算机。后面会讨论这些设备和网络部署中的几个。

设计需求根据目标环境会有很大区别。一些应用程序从未连接到因特网，一些定期检索特定信息的连接，还有一些要求不断的连接。例如，你可能拥有一台笔记本电脑，而有几个应用程序只能在你这台电脑上运行，例如电子表格程序、文字处理程序、税务申报程序或者音乐播放程序。当然，鉴于当今世界的连通性，甚至是这些小型程序都可能会不断检查更新的组件。然而，这些类型的程序不需要连接到因特网或者网络来达到它们的基本目的。

你也可能拥有一个能运行单机（不用联网）应用程序的移动数码设备或智能手机。也许你从网上下载了一个应用程序，但是一旦下载完成，它就能自己执行。这些类型的程序的设计通常都处于中等难度。

还有一些是要在分布式网络环境下运行的那些应用程序。这个环境要么是本地私人网络，如中型企业环境；要么是分布式全球网络。几乎所有大型企业都有只能被员工使用的这种类型的内部系统。例如，一个大型的全球企业可能有一个人力资源系统，这个系统是由世界各地许多公司办公室中的人力资源人员来访问的。

另外一种普遍存在的信息系统类型是以 Web 为基础的系统，这个系统是在一个服务器（或多个服务器）上部署的并且能用网页完全访问。其中一些类型的系统是小型的而且很简单，而有些其他系统则是非常复杂的，需要复杂的数据库连接、复杂的连通性以及多个外部系统界面。一个简单系统的例子就是在小型服务器上运行的一个私人博客。更复杂的系统就是有全面的商品信息和顾客销售的系统，例如 RMO 的 CSMS 系统。

图 6-1 为**网络图**，它说明了当今信息系统的一个常见部署。网络图是显示应用程序如何通过网络和计算机而部署的一种模型。图 6-1 描述的系统是以网络为基础的系统，同时它也

能通过因特网连接。我们用这一综合系统来确定必须要设计和构建的各种系统组件。

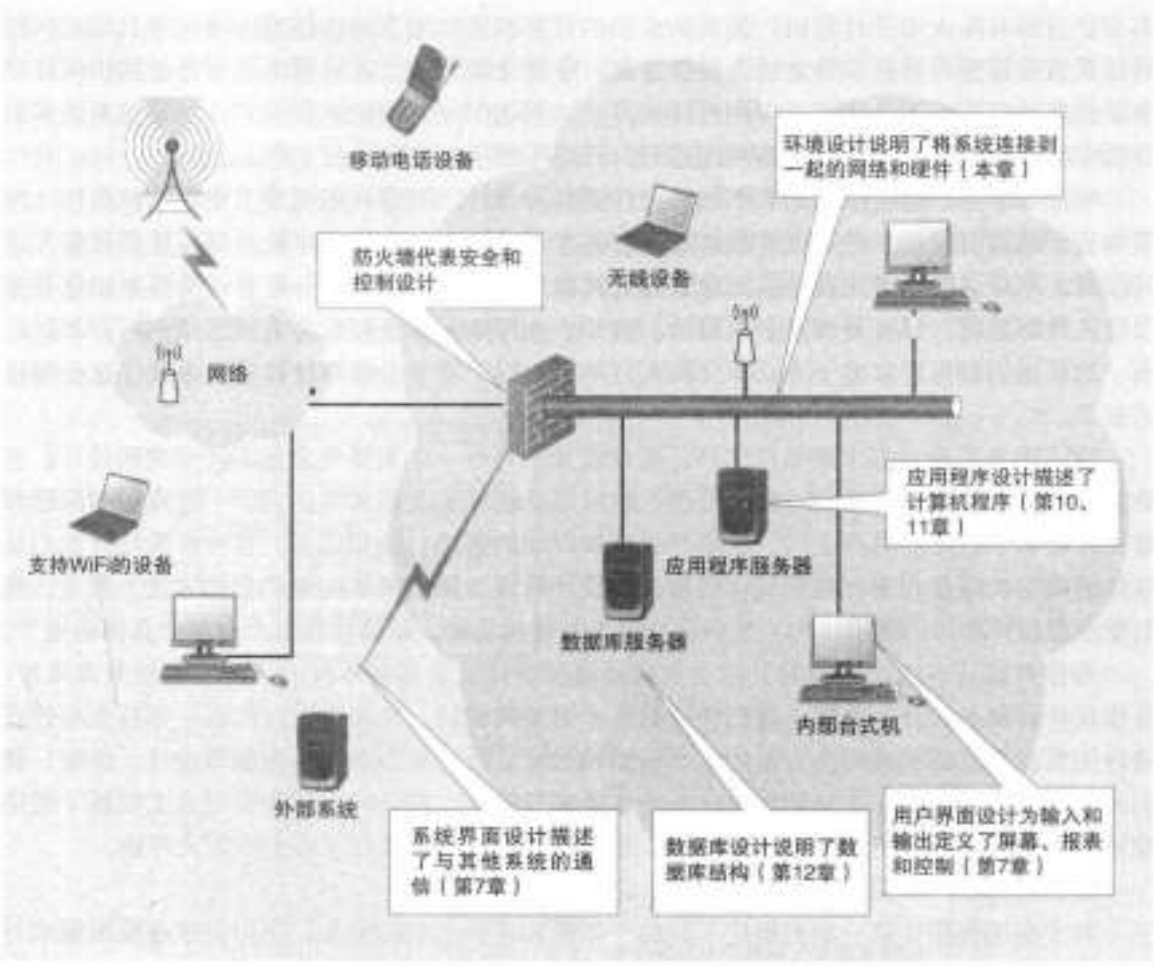


图 6-1　每个设计组件的含义

对于整个系统而言，分析员首先要确定整个应用程序的部署环境。这就需要定义硬件与软件环境。图 6-1 显示的硬件环境包括计算机、网络、防火墙等。软件环境包括用什么操作系统、用什么数据库管理系统和用哪种类型的网络协议。

在很多情况下，应用程序信息系统的底层结构已经存在，并且新系统必须遵循现存的部署。例如，大多数公司都会有现存的计算机、网络和通信设备的底层结构。因此，系统设计的一些部分可能不再需要，因为新的应用程序会融入现有的环境中。然而，即使有一个现存的底层结构，新的应用程序还是需要及时更新以满足新需求。

为了执行设计，分析员首先需要把整个系统分成几个主要的组件，因为同时设计出整个信息系统太复杂了。图 6-1 中的图标指的是硬件部分，在硬件内部的是软件部分。图中右边的大云团代表整个系统，云团内的各种图标表示系统的各个部分，这些部分必须要一起工作才能使系统运行起来。信息系统的专家们必须确保为用户开发了一个完整的解决方案。如果他们没有提出一个集成化的、完整的解决方案，那么他们就没能成功地完成工作。换句话说，设计必须包括整个底层结构以及各种组件。

图 6-1 所示的底层结构是当今计算机环境中很常见的一种。服务器包括软件应用程序和

数据库。通常，本地台式机能在本地网络上访问相应的应用程序——有线的或者无线的。本地网络是通过装有防火墙的计算机与因特网隔离的。能连接到因特网和其他地方的应用程序都要经过装有防火墙的计算机。装有防火墙的计算机通常能直接连接到因特网并且通过因特网的无线连接为计算机和移动设备提供连接。在防火墙外部，连接到电话设备也能访问移动电话设备的应用。不是所有的应用程序都需要这种访问因特网的扩展方式，但是越来越多的公司允许他们的员工在需要因特网连接到内部应用程序时采用远程工作方式。

在图 6-1 中，标注框确定了不同的组件需要的设计。就像在前面章节中提到的那样，网络和底层结构需要偶尔进行设计或扩展。存在于服务器中的应用程序软件需要伴随数据库进行设计。用户界面也必须设计，无论它是台式机的屏幕还是网页。一些系统需要自动化界面以接入外部系统，那些界面也必须设计。例如，内部购买系统可能会直接连接到厂商和供应商，这样他们就可以以电子方式提交购买订单了。以下章节会解释设计信息系统中这些组件的细节。

除了定义有设计需求的组件之外，系统设计中另外一个重要概念是不同层次的设计。在分析阶段，我们首先要确定问题的范围，然后用系统视觉文档来描述它。一旦我们对问题和重要议题有了清晰的概括之后，就要开始理解详细的需求。换句话说，在分析阶段，我们从总体的高层次信息到更详细的具体信息。在设计阶段，我们采取一个相似的方法。首先，我们要试着设计整体的底层结构，然后设计每个组件的功能，最后设计每个组件中具体的细节。

当你开始从事这个行业时，你会发现最高层设计有着许多不同的名称，包括架构设计、总体设计和概念设计。这里，我们使用的是术语**架构设计**。在架构设计阶段，你首先应该在进行细节设计之前明确解决方案的整个框架和形式。设计细节通常称为**细节设计**。这时，我们并不需要严格区分什么是架构设计、什么是细节设计，并且确定哪些模型或文档属于架构设计或细节设计也并不重要。重要的是，要认识到设计应该以自下而上的方式开始。

让我们回顾一下图 6-1 中确定的每个设计组件的含义。

对于应用程序，第一步是确定不同的子系统和它们之间的关系。应用程序也要部署成与网络、数据库和用户界面组件一致。应用程序早期设计中的部分是自动化系统边界。系统边界确定了哪些功能要包含在自动化系统中，哪些功能是手动程序。

对于数据库部分，第一步是确定要使用的数据库类型和数据库管理系统。表、数据字段和索引的一些细节可以确定下来，但是最终的设计决策还取决于应用程序的体系结构和细节。

对于用户界面，第一步是基于主要的输入和输出来确定用户对话框的通用形式和结构。项目团队也要描述用户界面要素与应用软件、硬件设备之间的关系。在当今世界，这可以是相当复杂的，因为许多不同类型的设备可能需要连接到系统。图 6-1 说明了内部计算机（有线与无线的）、连接到因特网的计算设备和通过电话协议连接的智能手机。用户界面要足够灵活，这样才能预料到要使用的所有不同类型的连接。在决定好整体连接和通信协议之后，要开发详细的屏幕布局和报表格式。

6.3　系统设计的输入和输出

在前面章节描述的分析活动中，我们将建立文档和模型。对于面向对象分析来说，我们使用事件表并建立一些其他模型，如类图、用例图、用例描述、活动图、系统顺序图和状态机图。无论是哪种方法，设计活动的输入都是由那些在前期阶段创建好的文档和模型组成的。

在迭代项目中（迭代项目已经在前面的章节中提到过了，并且将在第 8 章中详细解释它），分析与设计活动通常会同时进行。然而，任何迭代的第一个聚焦点是确定和说明需求（即分析），然后再决定解决方案（即设计）。

在分析阶段，分析员也要建立模型来代表现实世界，同时也便于理解所期望的业务过程以及在这些过程中所用到的信息。大体上来说，分析包括分解——把一个具有复杂信息的综合问题分解成易于理解的若干个小问题。然后分析员通过建立需求模型来对问题领域的知识进行组织、构造并编制文档。分析与建模需要大量的用户参与来解释需求并验证模型是否正确。

设计也是一个建模活动。分析员要转换在分析阶段收集到的信息——需求模型——以使其变成代表解决方案系统的模型。设计阶段所涉及的技术问题较多，因此它不要求太多的用户参与，但是要求更多其他系统专家们的参与。图 6-2 所示为从分析到设计的流程并突出了分析与设计阶段的主要目标。

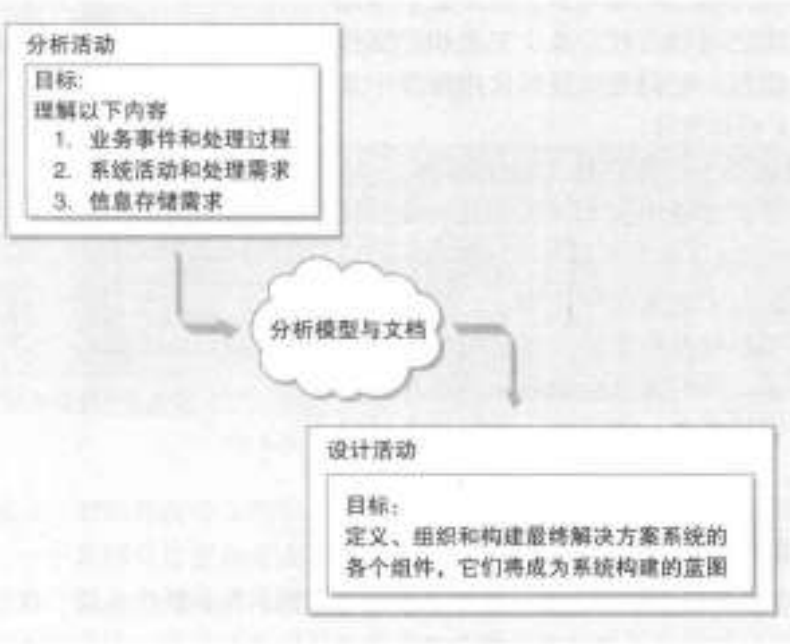


图 6-2　分析目标和设计目标

设计包括了对系统解决方案的描述、组织和构造。设计活动的输出是一系列实现这些目标的图和文档。这些图就是解决方案系统的各个方面的模型及其相应文档。

一个项目的正式程度也能影响设计。正式的项目通常需要开发良好的设计文档，这些文档通常都是在正规会议中评审的。非正式项目中的开发人员经常会用记事本和铅笔创建设计，然后一旦程序被编码就会扔掉这个设计。这种非正式的设计（在很多敏捷项目中会使用）仅仅是达到目的的方法，这实际上也就是程序代码。然而，即使外部人员没有看到设计文档，设计过程也必须继续跟进。程序员在没有仔细考虑过的情况下直接跳到编程环节的情况——这通常称为牛仔编码——必将得到错误、补丁和拙劣结构的程序。

图 6-3 确定了用在分析与设计中的主要模型。尽管它们也要和传统开发共享许多特征，但是这些是为面向对象开发提供的。注意，几个分析模型都会有相对应的设计模型，例如，类图对应的是设计类图。就像前面提到过的那样，从分析阶段过渡到设计阶段并没有严格的界限，并且当模型非常相似时这个方法将更加有帮助。以类图为例，分析员建立的模型要能确定类，

属性和关系，而设计类图添加了更多信息，如属性类型的信息、主键、主页和类方法或功能，它建立并扩展面向分析类图提供的信息。

你应该熟悉图 6-3 中所示的分析模型。在后面的章节中，你将会学习如何创建图中下半部分所示的设计模型。它们记录了最终系统的整个结构。设计类图以一种有助于数据库设计和应用程序设计的方式描述了类。顺序图是对系统顺序图的拓展并记录了类中控制和执行的流程。数据库模式要记录应用程序的数据库。对于大多数当前的应用层程序来说，要使用相关的数据库连接。用户界面模型为计算机屏幕和在线或纸质报表提供布局。通信图与顺序图很相似，因为它们都记录程序代码中类之间的迭代。系统安全和控制模型不是正规的模型，它们是确定新应用程序中重要控制特征的文档和符号。

在前面的章节中，你已经学习了分析活动，而且也已经具备了在使用技术和工具时所需的技能，这对创建如图 6-3 中上半部分所示的分析模型是有必要的。在一个真实的项目中，一旦系统分析员开始明白用户业务需求并通过使用分析模型来记录那些需求，他们就要开始设计活动并将分析模型拓展为设计模型。接下来，我们就会讨论那些设计活动。

图 6-3　分析和设计模型

6.4　设计活动

图 6-4 确定了与以下核心过程（第 1 章中讨论过的）有关联的活动：设计能解决问题或满足需求的系统组件。每个设计活动要与图 6-1 中确定的组件之一的设计相符合。本节会提供对每个设计活动的介绍。本书的后面会给出对具体概念和技能的深入解释和指导。设计环境将作为本章讨论的最后一部分，并且其他主题会按顺序在章节中介绍。

图 6-4　设计活动

系统设计是建立模型的尝试阶段，就像系统分析那样。在做出设计决策后，尤其是在详细的层次，它们是通过模型的建立而导出和记录的。正如之前说明的那样，这些模型可能不是很正规，但是它们是设计的本质。例如，在数据库设计中，在我们开始用 SQL 状态建立表之前，我们要确定需要的表和每个表的字段。在软件设计中，我们要决定哪些类是核心类，哪些是工具类以及每个类将包含的职责（方法）。用户界面设计通常需要故事板或其他 visual 模型来做出有效的工作流决策。所以这些设计任务都是建立模型的任务。

系统设计包括详细说明系统是如何使用专门的技术而工作的。一些设计细节在进行系统分析的时候就可以开发出来，但是我们需要了解更多的细节。此外，所有其他组件都会对最终解决方案中的每个组件产生很大影响。因此，系统设计活动通常是同时进行的。例如，数据库设计在软件设计中会频繁使用，甚至还会影响用户界面设计。同样，应用程序架构决定着如何部署网络的不同方案。当使用系统开发生命周期的迭代方法时，主要的设计决策是在第一个或第二个迭代中做出的。然而，许多设计好地组件在后期的迭代中还会再重新访问。为了能更好地理解这些设计活动，你可以用一个问题来总结每个活动。事实上，系统开发者通常会问他们自己这些问题，以此来帮助自己集中在每个设计活动的目标上。图 6-5 所示为这些问题。

设计活动	关键问题
设计环境	我们是否已经详细说明了软件执行的环境和所有不同的选项？
设计应用程序结构和软件	我们是否已经详细说明了软件所有的要素及每个用例是如何执行的？
设计系统界面	我们是否已经详细说明了系统会如何与组织内外部的所有其他系统相互通信？
设计用户界面	我们是否已经详细说明了用户是如何与系统交互以实施其所有任务的（用例）？
设计数据库	我们是否已经详细说明了所有信息存储需求，包括所有的模型要素？
设计系统控制和安全	我们是否已经详细说明了所有的要素以确保系统和数据是安全的且处于被保护状态？

图 6-5　设计活动和关键问题

每个活动都会在最终的设计文档中开发一个特定部分。就像一座建筑的设计图有几份不同的文档一样，一个系统设计包也是由一系列专门用来详细说明整个系统的文档构成的。此外，与设计图在描述同一个实际建筑时必须一致、完整一样，不同的系统设计文档也必须相互结合以为整个系统提供一套全面的说明。例如，如果一个分析员在设计用户界面时没有咨询过数据库设计者，那么这个分析员可能会用错误的字段或错误的字段类型和长度来建立界面。内部的一致性对于系统建模和设计是一个有效的强制要素。在下文中，我们会简要讨论一下这些设计活动，以便能更好地理解其包含的内容。在后面的章节中，你将会为每个活动训练必要的技能。

6.4.1　设计环境

环境是支持正在开发的软件应用程序所需的所有技术。例如，在 RMO 的 CSMS 系统的开发过程中，我们集中在功能需求和非功能需求上。然而，系统会依赖一系列的计算机服务器、台式机、笔记本电脑和一些其他的计算机设备。每个计算机设备都会有操作系统、通信功能、不同的输入 / 输出功能等。其他的软件通常叫作中间件，它们用于帮助这些不同的计算机设备集成为一个全面的解决方案。所有这些支持系统——软件和硬件都被认为是技术结构的一部分，这是第 2 章讨论过的。为了使新的 CSMS 能被成功地开发与部署，必须要精确定义完整的环境。因此，开发新系统的第一步是定义环境。

每个软件应用都必须在一些技术环境中执行。这个环境包括应用程序部署过程中所需的计算机和其他硬件，以及像服务器计算机、台式机、笔记本电脑、防火墙、路由器、电缆、光纤和无线接入点这样的事物。一些应用程序就是简单的单机应用，也就是只能在简单的计算机、笔记本电脑或移动计算机设备上执行。其他应用程序则以服务器为基础并且利用应用程序服务器和数据库服务器，也许还有一些内容传递网络，通过本地浏览器与用户在他们的计算机上连接所有的应用程序功能。还有一些应用程序是复杂的分布式应用程序，其中应用程序本身和数据同时在不同的计算机上执行。另外还有一些应用程序可能是为远程计算机设备部署的，如智能手机或远程监控设备。当今的计算机环境已经成为连接世界的技术，许多环境都能在不同的协议上操作，而且并不是完全兼容的。设计环境的一大部分是确定和定义所有需要的计算机设备的类型。这就包括确定所有集成计算机硬件所需的位置和通信协议。

技术环境不仅包括硬件。另外一种重要的组件是由操作系统、通信协议和系统以及其他支持软件（即中间件）组成的。例如，一个以 Web 为基础的系统的部署要包含服务器操作系统以及用户计算机上的操作系统。服务器也可能有其他系统，如 Web 服务器、数据库管理系统、编程语言服务器、图像和图形处理器或其他特殊的软件。支持软件环境的设计在软件系统不能直接相互通信时会变得更加复杂。这些不兼容性必须得以解决。

6.4.2　设计应用程序结构和软件

在设计应用程序结构时，要包括新系统结构和配置的决策以及计算机软件本身的设计。在设计过程中的第一步是将软件分解成子系统。同时也要做出关于数据库底层结构和多层结构设计的决策，多层结构设计是从业务逻辑和数据库处理过程中分离出来的。技术结构会影响很多类似的决策。例如，哪些子系统需要安装在设备的哪个部分？根据重要性、反应时间需求或隐私和安全问题，会把子系统放在不同的服务器计算机上。

需要处理的其他类型的需求能影响技术结构和应用程序结构。例如，用户只在工作的时候才能在他们的台式机上访问到新系统还是他们能够通过因特网连接在家办公？允许远程无线电设备连接到系统是有必要的吗？新系统必须能处理哪种类型的事务（用例）以及能处理多少事务量？这些类型的应用程序决策会影响应用程序结构、环境和其他硬件需求。设计应用程序结构通常是自上而下的处理过程，首先是定义整体架构，然后是不同组件的细节设计。

应用程序设计的其他部分是在一个细节层次上设计应用软件。细节设计首先是一个建立模型的活动。创建模型不仅启动了设计过程，还提供编写代码所需的文档。这些模型包括活动图、顺序图、设计类图和其他物理模型。对于传统方法而言，要开发的是像数据流图这样的模型。例如，对于面向对象设计，主要模型之一是设计类图，它确定了类及其属性和方法。图 6-6 是 RMO 的 CSMS 系统的部分设计类图。第 10 章和第 11 章会解释如何设计应用程序结构和软件的细节。

6.4.3　设计用户界面

分析员要记住，对于系统的用户来说，系统界面就是这个系统。用户界面不仅是屏幕，它是用户在使用系统时所能接触到的一切——无论是从概念上、感知上还是实际上都是这样。因此，用户界面不仅仅是系统的附属品。

新技术已经使用户界面产生了许多新需求。例如，用户只会用大屏幕的计算机还是也会使用屏幕小的 PDA 和其他远程设备？其他设备会用来输入文本、口令、图片和图像等信息

吗？这些用户界面要素和需求都需要在整个开发过程中考虑。

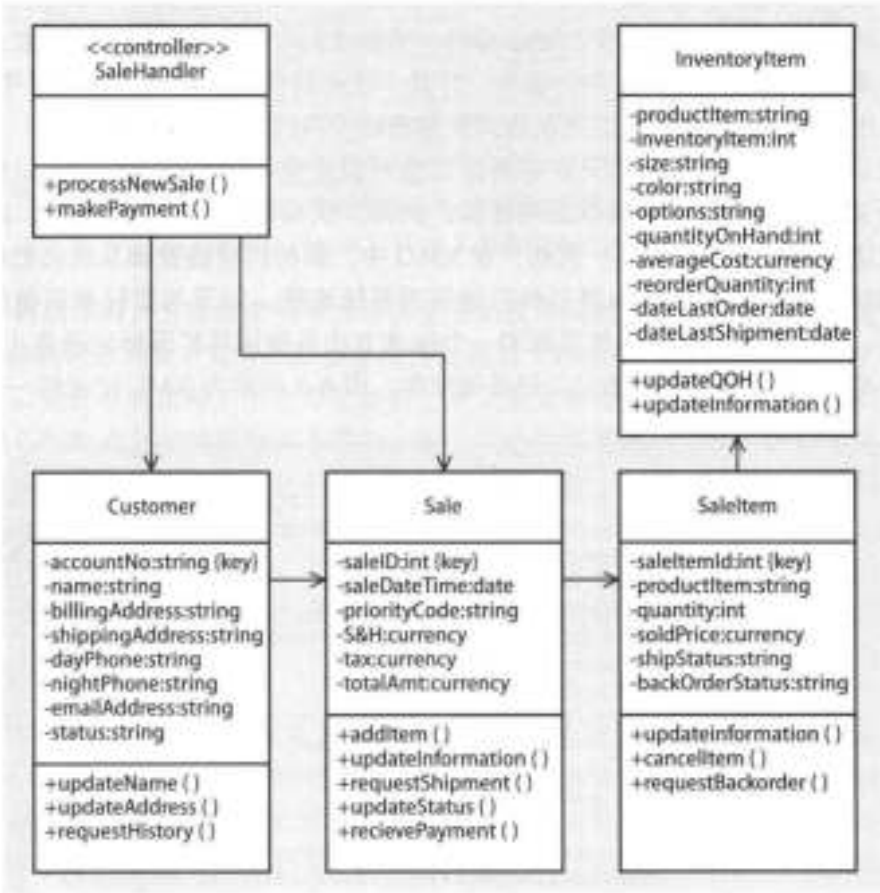


图 6-6　RMO 的 CSMS 系统的部分设计类图

在台式机、笔记本电脑和平板电脑上，界面是指有窗口、对话框和鼠标交互的一个图形用户界面。这些甚至包括声音、视频和声控命令。在移动设备上，触摸屏、屏幕键盘、声控命令和响应、传送和定位输入 / 输出是用户界面的标准组成部分。随着信息系统变得越来越互动和更易接近，用户界面也变成了整个系统中更大、更重要的部分。

设计用户界面可以被看作一个分析与设计活动。它有一些基本的分析要素，因为开发者必须要理解用户需求和用户如何完成他们的工作。用户界面不仅要有正确的信息，还必须是高效和美观的。用户界面设计也是设计活动，它需要创造力，并要符合严格的技术要求。许多类型的模型和工具用来执行用户界面设计，包括实物模型、故事板、图形布局和使用屏幕建模工具的原型。在当今这个互连的世界中，设计用户界面的主要困难之一是台式机屏幕和智能手机显示的不同通常会使同一个应用程序可能具有多种用户界面。第 7 章描述了许多用于有效实施用户界面设计的工具和技术。

6.4.4　设计系统界面

在当今相互连接的计算机环境中没有凭空存在的系统。一个新的信息系统可能会影响和利用许多其他信息系统。有时，一个信息系统提供的信息以后会被别的系统使用；有时，当系统运行时它们之间会不断地交换信息。使系统之间能够共享信息的组件就是系统界面，我

们对每个系统界面都要进行详细的设计。

这些界面的形式都有很大的不同。在某些情况下，一份文档会从一个系统发送到另一个系统。在其他情况下，实时数据交换是必需的，系统之间存在日常数据传输。在另外一些情况下，一个系统需要另一个系统中的服务，并且一个函数调用通过一个应用程序界面执行。交换的形式也可以是不同的，从二进制格式到加密格式再到基于文本的形式。

从系统设计的一开始，分析员就必须保证所有的系统可以在一起良好地运作。在某些情况下，新系统需要和组织外的系统相连接，例如，供应商站点或顾客家中。渐渐地，组织通过组织边界将系统连在一起。例如，在 RMO 中，新的供应链管理系统会把信息传送给它的主要供应商。新的 CSMS 也要与供应链管理系统连接，以及与银行和其他信用验证组织实时连接。定义基于文本的系统界面的一个标准方法是使用可扩展标记语言（XML）。与 HTML 相类似，XML 使用标签来定义记录的结构。图 6-7 所示为 XML 记录的一个例子。

```
<inventoryRecord>
            <productItem>WS39448-7</productItem>
            <inventoryItem>48763920</inventoryItem>
            <itemCharacteristics>
                <size>large</size>
                <color>blue</color>
                <options>withzippers</options>
            </itemCharacteristics>
            <orderRules>
                <quantityOnHand>54</quantityOnHand>
                <averageCost>38.27</averageCost>
                <reorderQuantity>25</reorderQuantity>
            </orderRules>
            <dates>
                <dateLastOrder>06042012</dateLaseOrder>
                <dateLastShipment>08072012</dateLastShipment>
            </dates>
</inventoryRecord>
```

图 6-7　使用 XML 的系统界面的例子

一些系统界面能连接内部组织化系统，因此分析员可以获得一些关于其他系统的信息。在 RMO 内部，供应链管理系统会和贸易展览系统进行实时通信。销售子系统为了了解某个商品是否有存货，就必须与供应链仓库数据库连接起来。

系统界面可能变得很复杂，尤其是在有着许多可用技术的今天。系统界面设计会在第 7 章详细讨论。

6.4.5　设计数据库

每个计算机信息系统的一个完整部分就是信息本身和它的基本数据库。数据模型（域模型）会在系统分析的早期创建，然后用来创建数据库的实施模型。通常，首先要决定的是数据库结构。有时，数据库是传统计算机文件的一个集合。更常见的是一个包含几十张甚至上百张表的关系数据库。有时，文件和关系数据库在同一个系统中一起使用。另外需要做出的决策是数据库要集中起来还是分散开。数据库内部属性也要设计，包括表、属性和连接。图 6-8 所示为在 MySQL 中库存商品的一个 RMO 数据库表定义的例子。

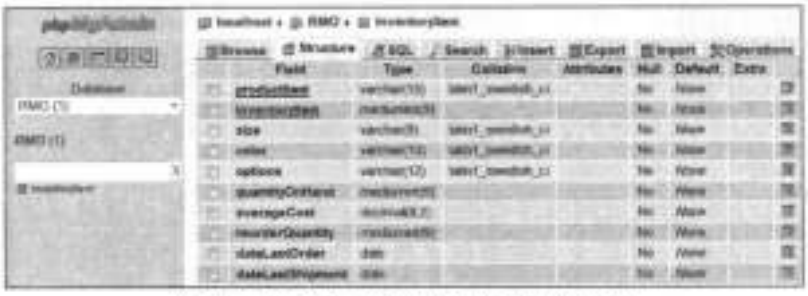


图 6-8　MySQL 中定义的数据库表的例子

在设计数据库时，分析员必须考虑许多重要的技术问题。在系统分析阶段定义的许多技术需求（与功能需求相对）要考虑数据库的性能需求（如响应时间）。许多设计工作都需要进行性能优化以确保系统实际上工作得足够快。对信息完整性来说很重要的一方面，就是安全和加密问题必须解决且设计到解决方案中。鉴于当今世界普遍的连接，数据库可能在世界不同的地方需要复制或分割开来。在不同的数据库管理系统中有多个数据库也是很常见的。这些数据库可能会分散在多个数据库服务器上，甚至可能位于完全不同的站点中。这些高度技术性的问题通常需要数据库设计、安全、执行和物理配置方面专家的专业技能。数据库设计的最后一个关键方面是确保新数据库能与现有数据库恰当结合。

6.4.6　设计安全和系统控制

最后的设计活动包括确保系统有足够的安全措施来保护组织的资产——这里的安全措施就是指系统控制。这项活动列在最后并不是因为和其他活动相比它不重要。相反，在当今这个文化环境中，外部人员能对系统及其数据造成严重损坏，因此设计系统控制是重要的活动。安全和系统控制的设计应包括在所有其他设计活动中：用户界面、系统界面、应用程序结构、数据库和网络设计。

用户界面限制了授权用户对系统的访问。系统界面控制确保其他系统不会对本系统造成损害。应用程序控制用来确保准确记录事务并且系统执行的其他所有工作都能正确完成。数据库控制确保防止在未经授权的情况下访问数据，以及防止由于软件或硬件故障而造成的意外数据丢失。最后，网络控制用来保证网络间的通信得到保护，并且其重要性日益增加。在现有技术的基础上，所有这些控制都应该设计到系统中。常常邀请专家来做一些控制方面的工作，所有的系统控制需要全面地测试。

6.5　如何设计环境

设计活动列表中的第一个活动是设计环境。将这个活动列在最前面是因为它渗透在所有其他设计决策中。例如，相对于一个复杂互联的分布式系统，单个单机的桌面系统需要完全不同的设计决策，如软件、用户界面、系统界面、数据库。尽管关于环境的全部细节设计决策可能无法在项目的一开始就完成，但一些重大决策问题需要被解决。

现今因为设备类型及软件应用程序配置的激增，软件系统的部署有着难以置信的变化。因此，组织和讨论关于设计环境这样的问题没有捷径。本节讨论的问题是关于软件部署中三个主要的行业发展趋势：在组织中完整部署的软件系统，仅仅为外部使用建立的软件系统（在该例子中，是通过因特网在万维网上部署的），以及以分布式方式远程部署的软件系统（为内部和外部使用）。

6.5.1　设计内部部署

内部部署的软件系统有两种类型：单机系统和内部网络系统。即使这两种系统的内部环境有一些共同之处，但是每种系统都有它在设计阶段必须考虑的唯一需求。

单机软件系统

在单个没有经过因特网或网络连接的计算机设备上执行的任何软件系统都是单机系统。公司仍会开发单机系统，但是大多数单机系统是单独开发的，然后再被卖给或交付给公司或其他个人。例如，微软 Office 套件和苹果 iWork 套件都主要是在单独的计算机上执行的，而且都允许创建 word 文档、电子表格、演示文稿。同样，许多人使用像 QuickBooks 或 H&R Block at Home 这样的单机软件——这些程序的获取要么通过 CD 里的程序包安装，要么通过下载的安装文件安装。单机系统的另一种类型是许多人在他们的笔记本电脑或台式机上下载下来玩的游戏。

单机系统的设计问题通常是很简单的。这些系统通常在没有访问数据库的情况下读出数据并且将数据写入文档。单机系统最大的问题是它们经常需要在不同设备中部署。例如，一个税收程序可能需要在 Windows 系统下可以运行的版本、在 UNIX 系统下可以运行的版本以及使用 Mac 操作系统下可以运行的版本。其他单机程序要能在这三种配置环境下运行，而且还要能在移动设备（包括平板电脑和智能手机）上运行。每种环境都需要操作系统界面和用户界面功能有细微不同的版本。通常，我们会设计和建立不同版本的应用程序，这样最适合某一特定设备的那个版本就会被部署。

基于网络的内部系统

基于网络的内部系统是指由组织自建或购买的专用系统。这并不是意味着除了公司员工外，它能被该地理区域内的任何人使用。图 6-9 所示为网络图，它说明了这种系统一种可能的硬件部署。通常，这样的硬件环境称为**局域网**（LAN）——其中电缆和硬件被限制在单个地点的计算机网络，如一幢建筑物。

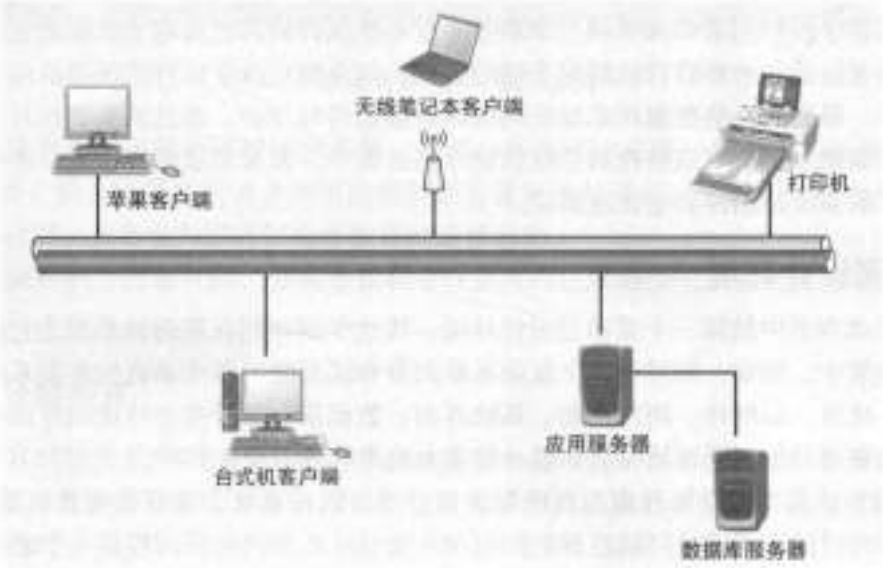


图 6-9　内部网络系统的网络图

这样的配置描述的是内部网络系统一个简单的**客户端 / 服务器结构**。客户端 / 服务器结

构和单个计算机结构的不同之处是，在客户端/服务器中的每个计算机必须连接到服务器。用户工作时使用的计算机称为**客户端计算机**，而主计算机称为**服务器计算机**。后者提供功能和数据，这是客户端计算机会接收到的。这两种系统可以在一个客户端/服务器结构中部署：

- 桌面应用系统。
- 基于浏览器的应用系统。

桌面系统最简单的版本是在客户端计算机上执行的一个计算机程序。在那样的情况下甚至可能不需要服务器计算机。然而，许多桌面系统连接到服务器计算机来检索和更新数据库中的数据。即使是再复杂的桌面系统也是由在客户端和服务器计算机之间一起通信的计算机程序组成的。这种系统的优势是演示（即用户界面）和功能可以自定义成用户的确切要求。这些类型的系统的例子包括图形系统或工程系统，其中的处理过程和演示需求都是十分严格和集中的。

内部网络系统的另一种类型是基于浏览器的。在一个基于浏览器的系统中，用户计算机（即客户端）的屏幕和报表的演示是通过因特网浏览器处理的，如IE、火狐、Chrome或Safari。在这个配置中，大多数处理过程和大量的计算都是由服务器完成的，然后再作为**超文本标记语言**（HTML）页面传递给客户端计算机。这就为服务器计算机增加了大量工作量，因为它不仅要给所有客户提供数据，还要为所有客户做所有处理工作。因此，通常会购买高速计算机来提供必要的计算能力。另一个劣势是用户界面屏幕和报表演示必须符合浏览器的功能。通常，这不是主要问题，但是有时它是受限的。使用基于浏览器设计的一个优点是系统能很容易地扩展到本地局域网之外，而且还可以经由因特网进行部署。这些类型的系统使用同种传输协议：**传输控制协议/互联网协议**（TCP/IP）。

三层客户端/服务器结构

软件设计的一个有效方法是将用户界面程序和业务逻辑程序分开，将业务逻辑程序和数据库访问程序分开。这种设计应用软件的方法称为**三层结构**。三层结构用在所有类型的系统中，包括桌面应用程序和基于浏览器的应用程序。三层结构将应用软件分为三层：

- 用户界面或**视图层**，负责接收用户输入，并将处理结果格式化输出。
- **业务逻辑**或**域层**，负责实施业务处理过程的规则和程序。
- **数据层**，负责管理存储的数据，这些数据通常存储在一个或多个数据库中。

图6-10概括地说明了这三层是如何一起工作来响应处理过程或信息的用户需求。这是一种建立软件的有效方法，因为程序员可以更简单地集中他们的注意力来解决问题。更新和加强系统不同部分也变得更简单了。例如，用户界面可以在对业务逻辑程序产生最小影响的情况下进行改变。

图6-10 完整的三层结构

客户端/服务器结构的优点之一是它能简单地进行支持（事实上，鼓励）通过使用应用程序三层结构来开发软件。图6-11说明了用三层结构开发的内部部署系统，同时也显示了这三层是如何在三个分开的计算平台上配置的。

视图层位于所有的客户端计算机中，而且它也是应用服务器计算机的一部分。HTML是通过客户端计算机上的浏览器显示的。格式化HTML的视图层类位于应用服务器上。数据

层由数据服务器和有必要访问数据的应用服务器中的任何一个应用程序组成。业务逻辑层位于应用服务器计算机上，包括所有处理业务规则的逻辑。

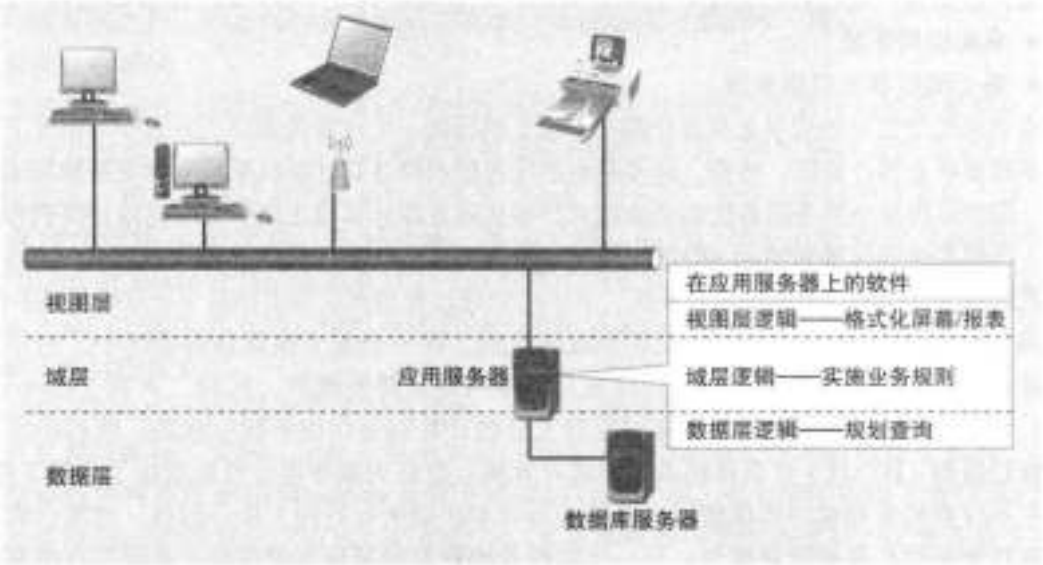


图 6-11　三层结构的内部部署

使用三层结构的一个主要好处是它与生俱来的灵活性。各层之间总是通过请求与响应的交互方式，这使得层与层之间相对独立。其他层在何处实现、在哪台计算机上采用何种操作系统并不重要。它们之间唯一需要彼此一致的是响应和请求的通用语言和一个有足够通信容量的可靠网络环境。

多个层可以在同一台计算机上执行，每个层也可以由独立的计算机来操作。复杂的层可以由两到三台计算机来实现。通过将层的功能分配给多台计算机或者在冗余计算机之间实现负载均衡可以提高系统处理能力。在出现故障时，服务器负载可以从一台计算机转移到另一台计算机上，这种冗余将增强系统的可靠性。总之，三层结构为现代企业提供了部署和重新部署信息处理资源的灵活性以响应不断变化的情况。我们将会在第 10 章和第 11 章讨论三层设计的软件方面。

6.5.2　设计外部部署

新软件应用最大最快的成长舞台是在互联网上只为外部使用的系统部署。宽带连接性能和网络设备上巨大的发展为在线业务的创建提供了难以置信的机会。如今，大多数实体商店已经扩展了他们的业务模型，包括商品和服务的在线购买。巨大的增长也发生在基于家庭和其他只通过互联网做生意的小企业。在大多数情况下，支持这些业务活动的软件应用是仅为外部使用而建立的。换句话说，对于在内部工作的员工来说，是不需要使用这些系统的。这些系统由那些不属于托管组织的顾客使用。与外部部署系统环境相关的重要问题包括：

- 为互联网部署进行的配置。
- 为互联网部署做出的托管选择。
- 带有互联网部署的客户设备的多样性。

为互联网部署进行的配置

图 6-12 所示为一个简单的互联网部署配置。注意，这和三层结构的最后一部分非常相

似。事实上，大多数部署好的互联网应用程序使用的都是三层结构。后端（即应用服务器和数据库服务器）如同内部部署好的客户服务器系统那样提供了同样的功能。视图层结构有一些相同之处，但由于互联网的多变性和不安全，也有一些更复杂的需求。视图层由 HTML 页面（通过浏览器显示）组成。它也包括那些位于服务器上和负责格式化动态 HTML 的程序或类。

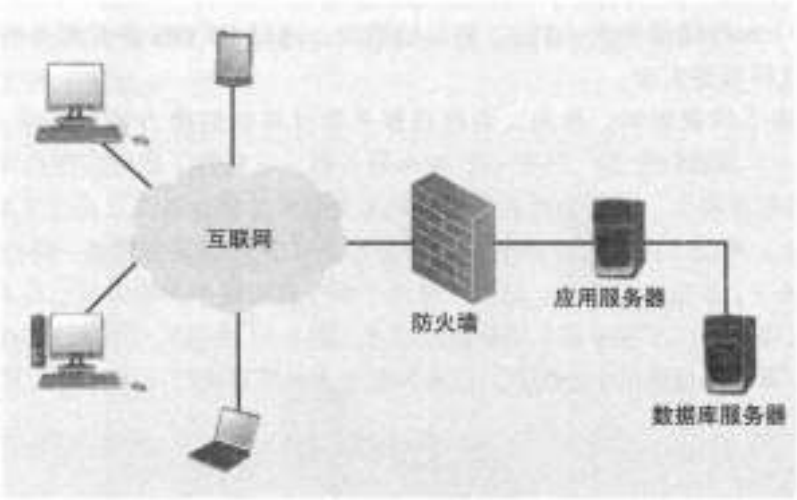


图 6-12　软件应用的 Internet 部署

互联网和网络技术为由外部顾客和组织员工使用的信息系统的实施提供了一个具有吸引力的选择。例如，考虑一下从公司供应商那边购买商品的一位 RMO 顾客的数据输入和数据访问需求。购买者在路上的时间一年中就有几个月——通常每次都要几个星期。因此，旅行购买者需要一些能与 RMO 的供应链管理系统进行远程交互的方法来记录购买协议和查询库存状态。

通过网络来实现应用程序相比传统的客户端 / 服务器应用有很多优点，包括：

- 可访问性——鉴于 Web 浏览器和互联网连接几乎无处不在，基于 Web 的应用程序对大量的潜在用户来说都是可访问的（包括顾客、供应商和非当地的员工）。
- 通信成本低——作为互联网骨干网的高容量广域网由政府投资建立。对最终用户来说，在骨干网上的通信是免费的。互联网之间的连接可以用相对较低的成本从私人互联网服务供应商那里购买。
- 广泛的实现标准——Web 标准已经众所周知，许多计算专业人员也已在使用中得到锻炼。
- 当然，通过互联网和 Web 技术的应用传送也有其不利的方面，包括：
- 安全性——对于安全漏洞，Web 服务器是一个明确的目标，因为 Web 标准是公开且众所周知的。网络大范围的互相连接、互联网的使用和 Web 标准创造了一个可被全球黑客访问的服务器。这可能是必须通过应用程序的外部部署解决的最严重问题。家庭系统必须提供保护，包括数据和他们传送到互联网的数据。
- 吞吐量——在高负荷发生时，吞吐量和响应时间可能受到极大的影响。这个配置必须不仅支持日常平均用户量还要能支持高峰时段用户量。这是不可预测的且可能变化很大。
- 变化的标准——Web 标准变化很快。客户端软件没几个月要更新一次。广泛使用的应用程序的开发者也进退两难：使用最新的标准来增加功能，或者使用旧标准来确保与旧的用户软件保持较大的兼容性。

对 RMO 来说，通过互联网实施顾客订单应用程序甚至是一个 RMO 购买者访问的家庭系统最主要的缺点是安全性和吞吐量。如果购买者可以通过 Web 访问系统，那么其他用户也可以。有很多方式可以控制对系统敏感部分的访问，包括用户账户和密码。但是安全漏洞的风险仍然存在。传输中数据的保护是很重要的，因为“监听”软件可以在传输中发现用户 ID、密码和敏感数据。

传输中的数据保护是通过**超文本传输协议安全**（HTTPS）实现的，HTTPS 是超文本传输协议（HTTP）和**传输层安全**（TLS）协议的结合。通过 HTTPS 协议服务的网页是以加密格式传输的，这样做更安全。

性能会被多个因素影响。首先，当然是服务器计算机的能力和它必须支持的通信量。图 6-13 所示为一个简单的配置，只有一个 Web 服务器，它托管了软件应用程序和数据库服务器。然而，随着容量增大，性能的增加是通过利用更强大的服务器以及通过增加更多服务器来实现的。从概念上来说，这个配置和图 6-12 中显示的互联网部署配置是一样的：一个 Web 应用程序服务器和一个数据库服务器。然而，每两个服务器形成多个服务器，后者在输入端有设备，用于分发页面请求以及服务器上的数据库请求。图 6-13 说明了一个使用多个应用服务器和多个数据库服务器的典型数据中心配置。这样的配置为计算环境的设计增加了另一层复杂性。

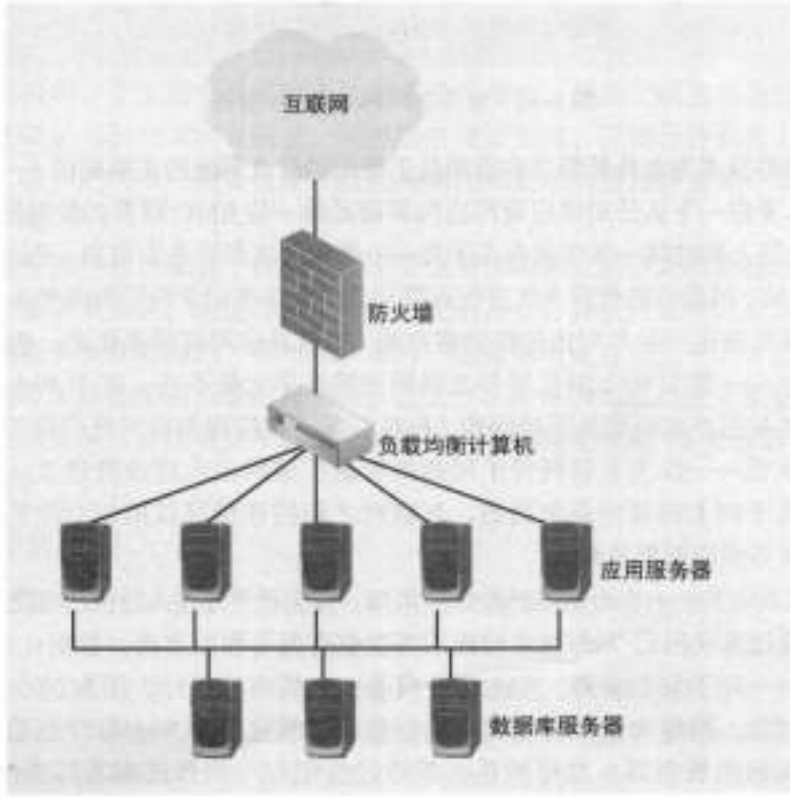


图 6-13　多层服务器结构

许多支持大容量的公司也会建立服务器区，而服务器区是由在全国或是全世界范围中被定位的多个数据中心组成的。每个数据中心库中有很多用负载均衡的硬件连接在一起的服务器。这样做给在访问应用这个请求被发送到正确的数据中心时增添了更多复杂性，这个请求通常发送到最近的数据中心，但也不是一直这样。这时就需要复杂的同步算法和软件来保持

数据当前在不同的数据中心。

吞吐量也能通过使用**内容分发网络**（CDN）来增加。这需要另外一组能用于传递像图片或视频这样静态内容的计算机。例如，当 RMO 的顾客从目录中请求一个页面时，应用软件决定所有的信息要返回到这个页面上。这个页面是基于从数据库中使用数据这个请求而动态创建的。然而，许多需要返回的图片是静态图片，很少会发生变化。换句话说，它们不会动态地变化。RMO 可以使用一个 CDN 服务器来传递所有要使用的图片和视频，而不是占用宽带来进出数据中心防火墙和负载均衡器。图 6-14 就是这样配置的一个例子，有着多个服务器和内容传递的独立地址。

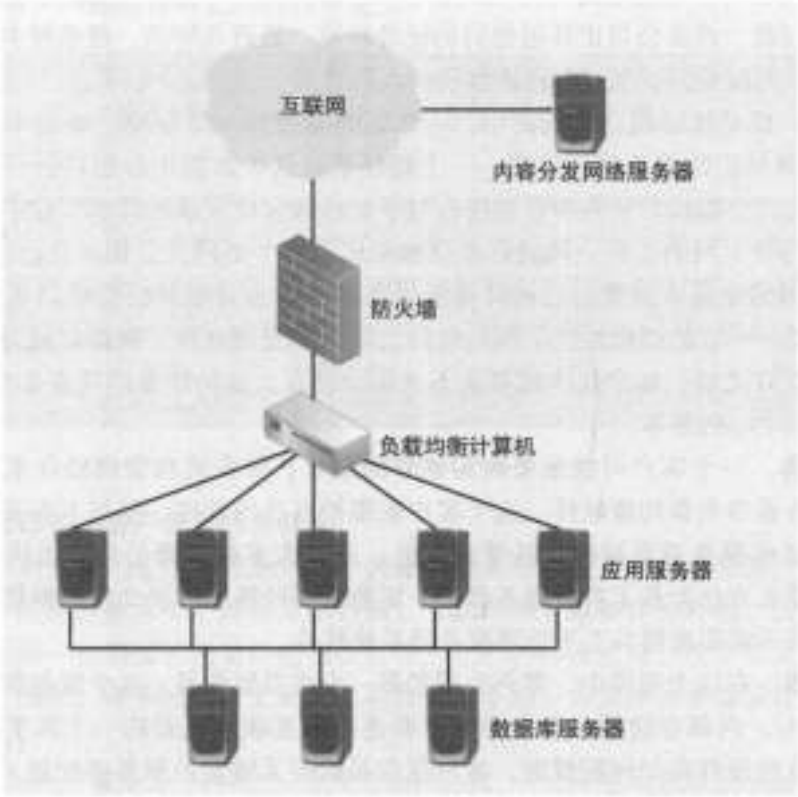


图 6-14　使用内容分发网络的多层服务器结构

最后，性能会受到 RMO 用户的网络连接点、连接与应用服务器之间可用的互联网能力的限制。不可靠或超载的本地互联网连接能导致这个应用程序无法使用。RMO 没有控制用户连接的能力。

为互联网部署做出的托管选择

仅为外部使用开发的软件应用开拓了许多托管选择。**托管**指的是代表应用软件和数据所属的人来运行和维护一个计算机系统。在部署外部使用的任何系统，尤其是顾客或其他外部组织使用的系统的过程中，必须要考虑几个关键问题，它们包括：

- 可靠性——硬件环境必须完全可靠，因为顾客和其他外部组织通常几乎无法容忍不可用的系统。这就要求镜像计算机、硬盘驱动器和数据库记录。备份和恢复必须被很好地建立。
- 安全性——硬件和软件系统必须是安全的。当前的法律条例对于财政和医疗数据要求

有很高的安全性。相应的处罚也是很严重的。

- 物理设备——为了确保可靠性和安全性，就需要特殊的房间甚至特殊的建筑物。此外，互联网连接时常需要多条路径连接到互联网骨干网。电源也必须是安全的，这通常意味着现场要有备用发电机。同时空调设备和备用设备也必须是充足的，以此来确保一个恒定的物理环境。
- 员工——为了确保可靠性和安全性，也需要有能工作7天×24小时的高素质的技术员工。
- 增长——外部系统通常会随着业务的扩展而快速增长，增加服务器数量也需要能响应这个总量的增加。当应用服务器和数据库服务器数量增加时，就会产生对更复杂的负载均衡的需求。增长也能使物理设备超出原来的数量，使多个数据中心成为必要。

由于这些问题，许多公司正外包他们的硬件环境。最近几年来，越来越多的公司需要这种托管应用程序的服务。下面就会讨论常用的几种选择。

场地出租。在场地出租这个安排中，一个公司会提供一个安全的数据中心让其他公司（如客户）来放置他们的服务器计算机。一个特征就是这个数据中心还包括一个带有安全锁和被保护的站点，它能满足所有的管理机构对于财政和医疗记录的需求。这个站点也有多个高性能的互联网骨干网的连接。同时它也会被集成为多个电网并且拥有自己的应急发电机。一个客户可以租借空间来放置自己的计算机服务器或者也可以从托管提供者那里租借计算机。管理服务器——它的操作系统、网络软件、数据库管理软件、数据备份等——会远程完成。在初步建立好之后，客户很少或者从不去那个站点。这种服务的优点是它不会有物理、安全、复杂数据中心的成本。

托管式服务。一个客户可能想要购买额外的服务，如安装和管理操作系统、网络服务器、数据库服务器和负载均衡软件。这个客户能维护自己的软件，但是不需要雇佣员工来管理操作环境。这些服务通常被称为托管式服务，并且大多数托管公司会提供这种服务。通常，客户公司要么自己配备了对应服务器的一定数量的计算机，要么就会租借。这个服务的优点是客户公司不需要雇佣员工来管理服务器系统软件。

虚拟服务器。在这个安排中，客户公司租借一个虚拟服务器，这个服务器已经被配置成一个有确切CPU、内部存储器、硬盘存储器和连接到互联网宽带的一个真实服务器。客户不知道这个计算机硬件是如何配置的，客户仅仅是购买了特定的服务器配置（通常是以月或年租借的）。这个客户公司可以购买带有或没有托管式服务的虚拟服务器。提供服务器的公司使用特殊的系统软件来为那个客户公司配置一个虚拟世界。公司为带有很少容量且不需要整个计算机的计算能力的应用程序使用这种类型的服务。这种服务的价格范围是从每月几美元到每月一两百美元，主要取决于虚拟服务器的大小和能力。这种服务器（除了那些已经被列出来的）的优点是客户公司一开始可以使用小的服务器，然后根据需求增加容量更大的。通常，客户购买了一个虚拟服务器，然后再逐步增加容量，根据需求要么增加一个更大的虚拟服务器，要么增加额外的虚拟服务器。

云计算。云计算有两个原则。首先，客户要有能力购买计算容量，就像是一个人能买水或电的能力。换句话说，客户只要购买需要的和要使用的。其次，客户不会担忧这样的环境问题：这个计算容量是如何或在哪里提供的，就像是一个人不需要担心电是如何产生的。有了云计算，客户公司就要购买在短时间内增量很小的计算容量（带有相关存储器、硬盘存储器和宽带）。客户公司还要说明需求环境，如带有Apache Web服务器的一个Unix操作系统，但是不要和操作环境有联系。换句话说，客户的应用软件运行“在云中”。当发生增长时，

云会自动提供更多容量。根据推测，这个安排能为客户公司节约成本，因为它不需要购买不必要的容量。现在，这是应用软件和操作环境分离的极限。

所有托管公司的一个主要卖点是其设备和网络连接的可靠性。大多数这些服务的合同包括一个**服务水平协议**（SLA），这个协议保证了系统可用性处在一个特定层次。网站（如一个特定的软件应用）的容量和活动使可用性变得非常重要。例如，如果亚马逊的系统在一天的高峰时间失效一分钟，那么它会损失多少收入？在没有性能损失时，SLA 能保证 99% 的时间是可用的，这样的情况是很常见的。一些提供者甚至能保证 100% 的可用性。提供者有能力做出这样的保证，因为他们有带有多个冗余层和备份层的多个服务器区。

图 6-15 列出了托管选择和它们的各种能力。

托管选择				
服务器选择	场地出租	托管式服务	虚拟服务器	云计算
托管服务提供建立和应用结构	是	是	是	是
客户有计算机	是	也许	不是	不是
客户管理计算机配置	是	不是	可能	不是
可扩展性	客户增加更多计算机	客户增加更多计算机	客户购买更大或更多虚拟服务器	客户增加增量小的计算能力
维护	客户提供	托管提供	托管提供	托管提供
备份和恢复	客户提供	托管提供	可用	可用

图 6-15　各种能力的贡献

带有互联网部署的客户设备的多样性

为外部使用而部署的应用程序的另一个关键问题是客户设备的巨大范围。问题是各种各样的设备都有不同的屏幕大小、屏幕显示特质、互联网浏览器和操作环境。被用来查看网页的设备通常会提供一些类型的浏览器应用程序作为它们的标准软件，这个事实能稍微改善一下这样的问题。然而，在不同设备上的浏览器性能也不同。为这些浏览器设计和实施用户界面总是一个挑战。

客户设备通过大小被分为三种类型：全尺寸的计算机、中型平板电脑以及小型移动计算设备。

全尺寸设备包括台式机和笔记本。尽管带有 24 英寸或 28 英寸屏幕的显示器也很常见，但是这些电脑通常有 15 英寸和 17 英寸屏幕两种。此外，这些全尺寸设备提供分辨率的层次，分辨率允许它们在更高层次的细节上进行显示。

中型平板设备有更多标准显示尺寸。尽管有一些的屏幕是 12 英寸的，但大多数设备有一个大约 10 英寸的屏幕。大多数平板要么是以横向模式显示，要么是以竖向模式来显示，这方面网页设计者会进行考虑。它们的分辨率通常较低，而且在这些小屏幕上很难查看细节。

最近，移动计算设备的数量正不断增长。不仅是以横向模式或竖向模式显示的这些设备，而且还有一类屏幕和分辨率更宽的设备。如果用户想要接收尽可能好的画面，那么设计用户界面是一个非常大的挑战。屏幕尺寸小的设备在能显示的细节上有很大限制，但是这能通过设备的缩放功能进行适当的弥补。

建立两或三个独立视图层是很常见的，因此一个软件应用可以在三种类型的设备上查看。事实上，大多数新的互联网软件应用都至少有两个独立的视图层来适应这些设备的差异。在某些情况下，视图层之间的不同不过是 HTML 被格式化的方式。在其他情况下，完

全不同的屏幕是通过不同类型的设备显示的。

6.5.3　设计远程和分散的环境

远程和分散的环境有内部环境、外部环境和基于 Web 环境的特征。就内部配置来说，一个有远程且分散环境的软件应用通常是被企业员工使用的内部系统。就外部且基于 Web 的部署来说，员工是不会被限制在一个单一的地点的，事实上，他们可以在全世界范围内使用。从历史角度来看，许多公司已经建立了他们自己的 WAN 来为员工服务。然而，建立与维护这些独立通信网络的费用对大多数公司来说已经变得太高了。如今，大多数的系统是通过使用互联网建立的，且被称为**虚拟专用网络**（VPN）。VPN 是一个建立在像互联网这种公共网络之上的网络，它为私人群体提供了安全且能被控制连接的网络。

通过虚拟专用网络进行的远程部署

先前，我们描述了一个在路上且需要连接到总公司系统的 RMO 购买者。如果这个购买者只需要连接到几个页面，那么用 HTTPS 协议建立的安全的 TCP/IP 连接就足够了。然而，如果这个购买者需要连接到总公司内部的其他安全系统，那么使用 VPN 可能会是一个更好的解决方案。有了 VPN，购买者可以连接到总公司的服务器，就像那些在总公司办公楼里的员工那样。

图 6-16 所示为一个 VPN 通过互联网正在使用 TCP/IP 协议。正如图中说明的那样，在远程计算机和总公司服务器之间有一个安全“管道”。这说明了 VPN 正使用互联网协议，但是却带有更高的安全性和控制性。为了实施这种类型的 VPN，特殊软件被用来建立一个安全的连接并且为所有数据传输加密。只有带有真正软件和密钥的计算机才能连接到 VPN 网络。

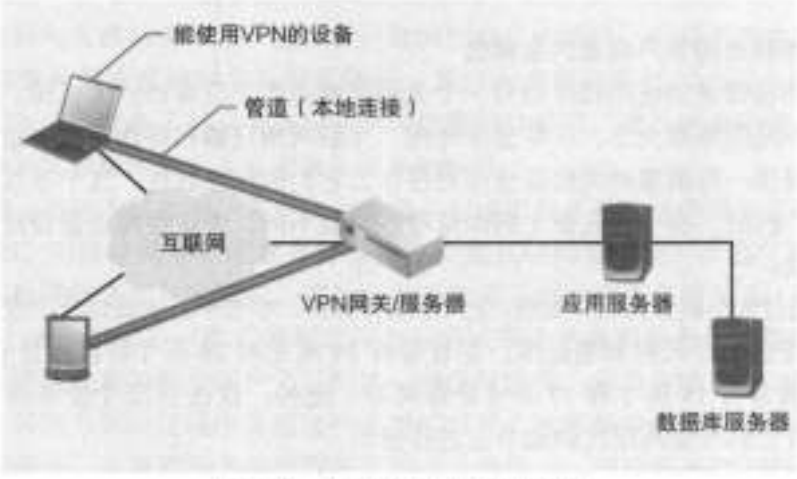


图 6-16　VPN 使用 TCP/IP 协议

这种配置在当两台远程计算机需要直接与对方交流时会发生变异。它们可以使用总公司服务器来帮助两者之间的连接，但是在它们连接上以后，它们就建立了**端对端连接**，这个连接能在没有其他辅助设备的情况下持续这次通信。一个端对端连接能直接在远程计算机之间连接，而且不需要连接到主计算机。那样的配置可以直接通过远程计算机之间的一个管道被代替。该配置的优点是这个连接的速度可以处理高容量的转换或即时响应，这对在线聊天是很有利的。

对购买者来说实施远程连接还有另一种方式，那就是构建一个使用 Web 浏览器界面的应用程序。这个应用程序在一个 Web 服务器上执行，使用 HTML 协议与 Web 浏览器进行通信，并且能用互联网连接访问任何一台计算机。购买者可以在他们自己的笔记本电脑上使用

Web 浏览器，然后通过本地的互联网服务提供者来连接到这个应用程序。他们也能用互联网连接访问任何计算机上的应用程序（如一台在供应商办公室的计算机、酒店商务套房中的计算机或者像 FedEx Kinko 这样的复制中心里的计算机）。

所有 VPN 重要的一面是通信连接总是会被加密，以此来维护安全。因为 VPN 的目标是允许同一组织中人员之间的私人通信，所以这个通信是被加密的。VPN 服务器和软件不仅使用 HTTP 安全协议，他们还使用包括额外授权、更安全的加密技术以及更高层次的传输监控。

客户设备的多样性

之前我们讨论了在一个范围很广的设备中部署软件的困难之处。软件应用的远程部署甚至要求更复杂的渲染要求。通常，为远距离的员工部署应用程序需要专门的设备——例如，快递服务在交付包裹时会传输顾客签名到总公司。其他类型的监控设备会有数据捕获和数据通信需求，这对一个特定的软件应用来说是独一无二的。

6.5.4　RMO 的企业技术结构

RMO 主要的办公室由在帕克城的企业总部、一个大零售商店、一个制造厂和一个大配送仓库组成。帕克城是创建公司和建立第一家零售商店的地方。在许多方面，盐湖城是 RMO 日常运营的中心。主要的数据中心位于帕克城中的一幢独立建筑物内。有两个配送中心：一个在俄勒冈州的波特兰，另一个在新墨西哥州的阿尔伯克基。另外的制造厂在华盛顿的西雅图。图 6-17 所示为 10 个零售商店在地图上的位置。

图 6-17　RMO 的零售店、防火墙分配中心和制造基地

伴随着在帕克城的主要数据中心，RMO 在每个办公室、仓库、制造厂和零售商店内都有内部局域网（LAN）。此外，配送中心和制造厂都有 VPN，能用这些设备连接到帕克城的数据中心和企业办公室。所有的零售商店都能用 VPN 连接到中心站点，这个 VPN 是与零售应用程序相连接的。在每幢楼的内部，局域网被配置成能提供公共连接。图 6-18 提供了描述当前技术配置的网络图。

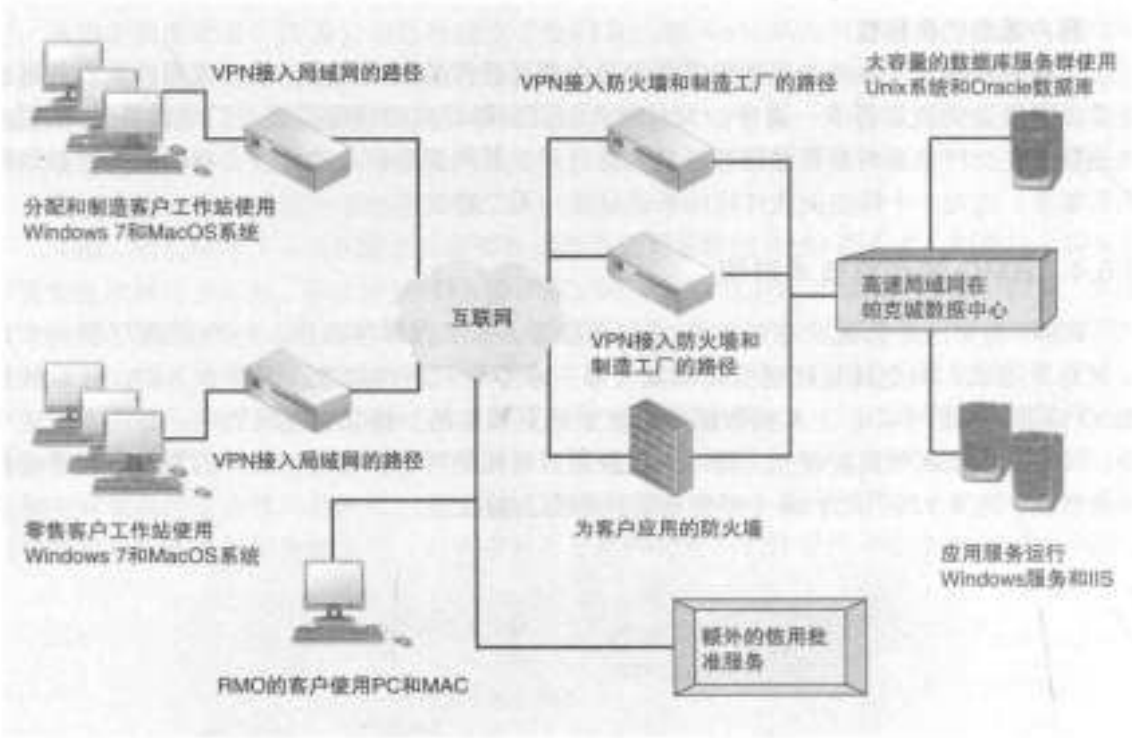


图 6-18　RMO 的最新技术结构

新 CSMS 系统项目的一部分将包括对使用大容量提供者来托管新系统的可行性加以评价。尽管没有一个选择被排除，但早期的分析说明了利用带有托管式服务的虚拟服务器是最佳选择。在犹他州和加利福尼亚州的托管公司似乎是最具吸引力的选择。接下来还要做更深一步的分析，并且对两到三个最具发展潜力的托管公司进行更为深入的调查和站点访问，这已经安排在项目的时间表中。

本章小结

系统设计就是组织和构造系统组件来允许新系统构造的过程（如编程）。新系统的设计由那些与新系统各种组件设计有关的活动组成。组件包括部署环境、应用程序结构和软件、用户界面、系统界面、数据库和与系统安全有关的系统控制。

设计活动的输入由在设计阶段建立的模型组成。设计阶段的输出由描述新系统结构和各种编程组件的详细逻辑关系的一系列图或模型组成。

设计应用程序结构可以分为结构化设计和细节设计。细节设计通常指软件程序的设计。结构化设计使应用程序适合配置环境，包括硬件、软件和网络。现代应用程序软件通常配置在分布式计算机环境中，并且通常按照客户－服务器结构——通常是三层结构来进行组织。

在当今这个广泛连接的计算环境中，软件应用的设计必须考虑客户计算机特征和服务器环境。客户计算环境的范围从简单的桌面系统到平板电脑再到非常小的移动设备。在设计的服务器方面有很多选择——从内部环境到场地出租或云计算。潜在客户设备和服务器计算机的特定需求影响了新系统的最终设计和操作。

复习题

1. 系统设计的主要目标是什么？
2. 系统分析与系统设计的区别是什么？
3. 列出为一个新的软件应用设计的主要要素。
4. 列出用在系统分析中的模型。
5. 列出用在系统设计中的模型。
6. 用户界面设计和系统界面设计的区别是什么？
7. 使用迭代开发系统的项目中，系统设计在哪个迭代开始？解释原因。
8. 结构化设计和细节设计的区别是什么？
9. 设计安全性和控制影响了其他什么设计要素？
10. 描述数据库设计需要什么？
11. 什么是局域网？在部署新系统时什么时候会用到它？
12. 什么是三层结构？
13. 描述三层结构中每一层的内容。
14. 列出在客户 - 服务器结构中客户设备的类型。
15. HTTP 和 HTTPS 的区别是什么？
16. 在软件的使用超过互联网时，要考虑的主要问题是哪两个？
17. 描述影响互联网系统吞吐量的主要因素。
18. 列出考虑外部托管公司的五个重要问题。
19. 云计算和虚拟服务器的区别是什么？
20. 为什么公司要使用场地出租设施？
21. 描述在为多个客户设计时需要考虑的问题。
22. 什么是 VPN？为什么公司要使用 VPN？

问题和练习

1. 一个金融公司有运行在不同办公室的桌面应用程序，这都是由集中式应用银行的两台计算机支持的。此外，还有一个集中数据库，它需要三个服务器。画出代表这个需求的网络图。
2. 一个销售组织拥有一个基于互联网的顾客支持系统，它需要支持每种类型的客户设备。这个服务器配置是一个常用分层的应用程序服务器和数据库服务器。画出代表这个需求的网络图。
3. 一个中型工程公司有三个独立的工程办公室。在每个办公室中，本地的局域网支持所有在那间办公室的工程师。由于办公室之间的合作需求，所有计算机要能查看和更新三个办公室内的任何数据。换句话说，每个局域网内的数据存储服务器要能访问所有的计算机，无论它们在哪里。画出支持这个配置的网络图。

4. 一个小型新兴企业拥有一个基于 Web 的顾客销售系统，它是通过使用 PHP 和 JavaScript 而编写的。这个公司正决定是否要在自己的服务器上托管这个系统，正在与一个托管公司商量用虚拟服务器托管，或者使用亚马逊的云计算。期望的容量在一开始较低，同时预测一个正在成长的模式是很困难的，尽管它们有快速增长的潜力。决定这个公司要选择哪种方式。通过给出基于这个新兴企业的每个解决方案的优点和缺点来支持决策。
5. 描述 HTTPS 和 VPN 之间的区别。哪种计算和网络情况更适合 HTTPS？哪种计算和网络情况更适合 VPN？
6. 找到四个独立的托管提供者，并且比较他们所提供的内容（包括价格）。将你的回答制作成表格，以此来显示调查成果。
7. 比较五个流行的且能使用互联网的智能手机的屏幕尺寸、分辨率和其他重要的显示特征。你会选择哪个作为最好的？解释原因。
8. 调查关于支持一个必须分布在多个服务器中的大型数据库的问题。写出一个需要被定址的问题列表，以及分布式和分区的数据库的可选解决方案的列表：（a）所有服务器处于同一数据中心；（b）服务器处于不同的数据中心。

扩展资源

Frederick P. Brooks, *The Design of Design: Essays from a Computer Scientist*, Addison-Wesley, 2010.

Priscilla Oppenheimer, *Top-Down Network Design* (3rd ed.), Cisco, 2010.

Doug Kaye, *Strategies for Web Hosting and Managed Services*, Wiley, 2001.

设计用户界面和系统界面

学习目标

阅读本章后，你应该具备的能力：

- 描述用户界面与系统界面的区别。
- 描述人机交互（HCI）领域的历史发展。
- 讨论可视性和提示性如何影响可用性。
- 描述能应用到所有类型的用户界面中的指导原则和针对于网页和移动应用程序的其他指导原则。
- 创建故事脚本来显示用在图中的表格的顺序。
- 讨论信息系统中发现的系统界面的例子。
- 基于应用程序定义系统输入与输出。
- 为接收者设计打印及在屏幕上显示的报表格式。

开篇案例　Aviation Electronics 的界面设计

Bob Crain 近来一直很赞赏安装在 Aviation Electronics（AE）制造支持系统上的用户界面。Bob 是 AE 在中西部的制造厂经理，他所在的制造厂提供在商用飞机上使用的航空设备。这些航空设备具备为飞行机组人员提供指导和控制的功能，而且提供飞行员在驾驶商用飞机时所需要的最新安全保障功能。

制造支持系统被用在制造过程的各个方面，包括产品规划、购买、零配件库存、质量控制、成品库存和配送。Bob 在几年的时间里全程参与了该系统的开发工作，包括初期的规划和开发。该系统体现了他在制造方面所知道的几乎全部内容。开发这个系统的信息系统团队完全依靠 Bob 的专业知识。Bob 对制造过程可谓轻车熟路。

最终的用户界面尤其令 Bob 满意。他坚持要求开发团队在开始就考虑整个用户体验过程。他不想要一个切蛋糕式的事务处理系统。他希望构造出的系统能够像制造过程中的一个合作伙伴，让人感觉到它真正地适用于用户所做的工作。

第一位项目经理不重视用户界面的设计。当 Bob 问他为什么用户界面在早期迭代中不是关键部分时，他回答说："在我们做完账户控制之后，会增加用户界面。"在 Bob 一再强调撤换项目经理后，信息系统部门委派了 Sara Robinson 来领导这个项目小组。

Sara 持有截然不同的见解，她刚刚接手工作就积极了解影响制造过程的事件以及用户需要系统提供哪些支持。虽然她有一组系统分析员从一开始就着手账户交易细节的开发，但她仍然一直强调从用户的观点出发来设计系统。Bob 和 Sara 组织了有用户参加讨论系统设计方案的会议，甚至请用户到现场模拟与系统进行对话。

在另外的会议上，Sara 提出了界面设计的草图，并要求用户将自己希望看到的信息和想要用到的选项补充到草图中。这些会议产生了许多新的想法。例如，许多用户并不是整天

坐在办公桌前，他们需要更大的和更多的图形显示信息以便能在房间的其他地方也能看到。许多用户需要参考多种显示内容，并且希望能够同时看到这些信息。使用制造过程中的图形仿真是完成这些功能最好的方法。经过用户的充实，界面草图真正反映了生产过程的实际流程，并且项目团队使用这些草图定义了系统界面的大部分。Sara 和她的团队坚持每月都与用户见面，拿出更多的设计方案，征求更多的建议和意见。

当系统最终完成并安装时，由于许多用户一直参与了系统的设计工作，因此他们已经知道如何使用系统。Bob 了解系统能做的全部工作，但他只使用他自己这一部分。他坐在办公桌前，只要单击“查看当前过程”按钮，制造支持系统就会给他整个上午工作情况的介绍。

7.1　引言

信息系统是与人和其他系统进行交互。因为很少有系统是自动化操作或独立操作的，所以设计系统、用户与环境之间的界面（输入与输出）是重要的系统任务。界面设计不良可能会导致系统操作的不完善或不符合目标。例如，一个用户界面不佳的人力资源系统可能会降低系统效率并成为数据输入错误的来源。一个用户界面不佳的面向顾客的系统可能会导致顾客转移他们的业务。正如面向用户界面那样，其他自动化系统设计不佳的界面会成为发生错误或效率低的来源。因此，系统界面的设计是一个系统开发项目的重要部分。

系统输入和输出是任何系统开发项目中要早期关注的问题。项目规划列出了分析员在定义系统范围时要确定的关键输入和输出。在分析阶段，分析员要在早期就和利益相关者讨论输入和输出，以此来确定会影响系统和依赖系统提供信息的用户和参与者。在分析阶段提供的需求模型也要强调输入与输出。例如，用例描述定义了发生在用例中的输入和输出。在系统顺序图中会把输入和输出作为消息和返回值来进一步定义。

7.2　用户界面和系统界面

无论是系统界面还是用户界面，为每个事件划分输入和输出都是系统设计阶段中的一个关键步骤。**系统界面**只需要很少的人工输入和输出。它们可能是由像扫描仪这种特定输入设备自动捕获的输入数据，或者是来自其他系统的电子消息，或者是来自其他系统的已捕获事务。如果是向其他系统发送消息或信息这样的输出，往往会被认为是系统界面（如一个运输公司接到的通知），或者是在没有很多人工干预的情况下为外部代理人提供报告、报表或文档这样的输出也会被认为是系统界面（如月末信用卡报表邮寄到了持卡人）。

用户界面包含了用户直接干预的输入和输出。用户界面可以是针对内部用户的，同时也可以是针对外部用户的。它们的不同取决于界面目的、用户特征以及一个特定接口设备的特征。例如，尽管所有用户界面都是为最大的易用性而设计，还有其他考虑（如使用效率），但对一个内部用户更重要的是：通过培训后，他能够使用为一个特定硬件设备而优化设计的特殊界面（如键盘、鼠标及高分辨率显示器）。相比之下，差别很大的用户界面可能是为智能手机这样的输入 / 输出设备的面向顾客系统而设计的。

在大多数系统开发项目中，分析员会将系统界面设计和用户界面设计分开，因为这两种设计需要的是不同的专业知识和技术。但是对于任何系统组件的设计而言，都需要大量的系统工作。

7.3　理解用户界面

许多人认为用户界面是在开发过程临近结束时才开发并增加到系统上的，但是用户界面是非常重要的。它是最终用户使用系统时所接触到的全部内容，无论是从物理意义、感知意义还是从概念意义上来讲，都是如此（如图 7-1 所示）。

图 7-1　以用户为中心的设计

从用户视角来看，用户界面就是整个系统。在界面后面的程序、脚本、数据库和硬件都是不相干的。能体现出用户界面这一视角的设计技术统称为**以用户为中心的设计**，它强调以下三个重要原则：

- 及早关注用户及其工作。
- 评价系统以确保其可用性。
- 使用迭代开发方法。

及早关注用户及其工作与本书中的系统分析方法是一致的。面向用户的分析与设计任务要尽可能早地执行，并且通常要比其他任务更优先执行。例如，像确定利益相关者和采访这样的面向用户分析任务要在项目的早期就开始做。用户界面在早期的迭代中就要被设计，而且与用户相关的决策会影响其他设计决策和任务。

对用户及其工作的关注程度比指定任务和优先级这个问题更急迫。它体现了理解用户的各种尝试以及回答以下问题：他们要了解什么？他们如何学习？他们喜欢如何工作？他们的动机是什么？面向用户的集中程度与在开发的系统的类型是不一样的。例如，如果系统是一个预包装的直接面向最终用户的桌面应用系统，那么系统就要集中关注用户和他们的偏好。

以用户为中心的设计方法的第二个原则是评价设计以确保其可用性。**可用性**是指学习和使用一个系统的容易程度。确保系统可用性并不容易，有太多类型的用户，而他们又有不同的偏好和技能。对于某个人来说非常易学的设计特点有可能对于另一个人来说却非常难。如果一个系统要面对各种各样的最终用户，那么系统设计者要如何确保该界面所有用户用起来都能得心应手？例如，如果界面设计得太易变，某些最终用户可能会感到无所适从。另一方

面，如果界面设计得过于死板，那么某些用户将会觉得备受挫折。

我们在方便用户学习和使用方面通常都会有不一致的看法。例如，拥有多个表格、许多对话框、广泛提示信息和指导信息的基于菜单的应用程序往往是易于学习的，实际上，它们都是自解释型的。这种易于学习的界面适用于那些最终用户并不经常使用的系统。但是如果内部用户整天都要使用这个系统，那么设计的重点要放在界面的快速切换和灵活性方面，应该包括快捷键、热键、语音命令和大信息量屏幕等方面的设计。这里的第二种界面是不容易学会的，但是一旦学会就显得非常好用。内部用户（在管理人员的支持下）为了变成高效率的用户会愿意花更多的时间学会这个系统。

开发人员会利用许多技术来评价界面设计以确保可用性。以用户为中心的设计需要测试用户界面的各个方面。一些可用性测试技术要收集进行统计分析的客观数据，用于与设计时的数据进行比较。有些技术收集那些关于用户感觉和态度方面的主观数据。为了评价用户态度，开发人员要进行正式调查、重点小组会议、设计走查、书面评价、专家评价、正式的实验室试验和非正式的观察。

以用户为中心的设计方法的第三个原则是使用迭代开发方法——那就是做一些分析，然后做一些设计，再接着做一些实施，然后重复以上过程。每次迭代之后，项目团队会对当前系统进行评价。迭代开发在每次迭代中不断返回到用户需求并在每次迭代后评价系统。这样的方式一直以用户为焦点，就像及早关注用户及其工作这个原则一样，总之这个原则通过贯穿于整本书中的系统开发方法而被反映，尤其体现在分析与设计任务中。

人机交互的隐喻

广泛使用的形象的面向用户界面是在20世纪80年代由Apple Macintosh在大众市场推出的，并且在20世纪90年代与各种版本的微软Windows一样变得普遍存在。为了使计算机更容易使用和学习，早期面向用户界面的设计者采用了**隐喻**。这是用户界面特征与用户熟悉的物理实体之间的类比。隐喻仍然会被广泛运用在用户界面设计中，如表7-1所示。

表7-1　用户界面设计中的隐喻

隐喻	描述	例子
直接操纵	操纵显示器上的物理对象（图片）或能代表它们的对象（图标）	用户拖动一个文件夹图标到一个回收站或垃圾桶这个图片中可以删除一整个集合的文档
桌面隐喻	在边线中间和工具栏图标集中的地方用大量的空置工作区来组织视觉显示的不同区域	在计算机启动时，Windows用户会看到桌面，上面有时钟、日历、记事本、收件箱和即时贴的图标（在电子版本的不同位置之间跳转）
文档隐喻	像页面或表格那样显示文档中的数据	用户要为自己购买的产品填写表格，制造商的网站上会找出并以Adobe Acrobat文档的形式显示产品手册，这份手册中包含一个内容的超链接列表和能连接到相关文档的嵌入式连接
对话隐喻	用户和计算机通过使用文本、声音或工具（如带标签的按钮）来参与交流或对话并完成任务	用户单击标记为“troubleshoot”的按钮，说明打印机不能工作了，计算机会在显示器上显示问题，并且用户要通过输入答复或从打印列表中选择回复来进行响应

图7-2是计算机正在使用说明**直接操纵**、**桌面隐喻**和**文档隐喻**的窗体的屏幕截图。所有的显示内容与物理桌面看上去很相似。常用工具的图标和图片位于左右两边。这些图标可以直接用鼠标或另外的定点设备来操纵。在屏幕中心的活动窗体是看上去与摆在桌面上的真实

纸质页面相似的文档，而且每一页后面都附有一张标准说明。

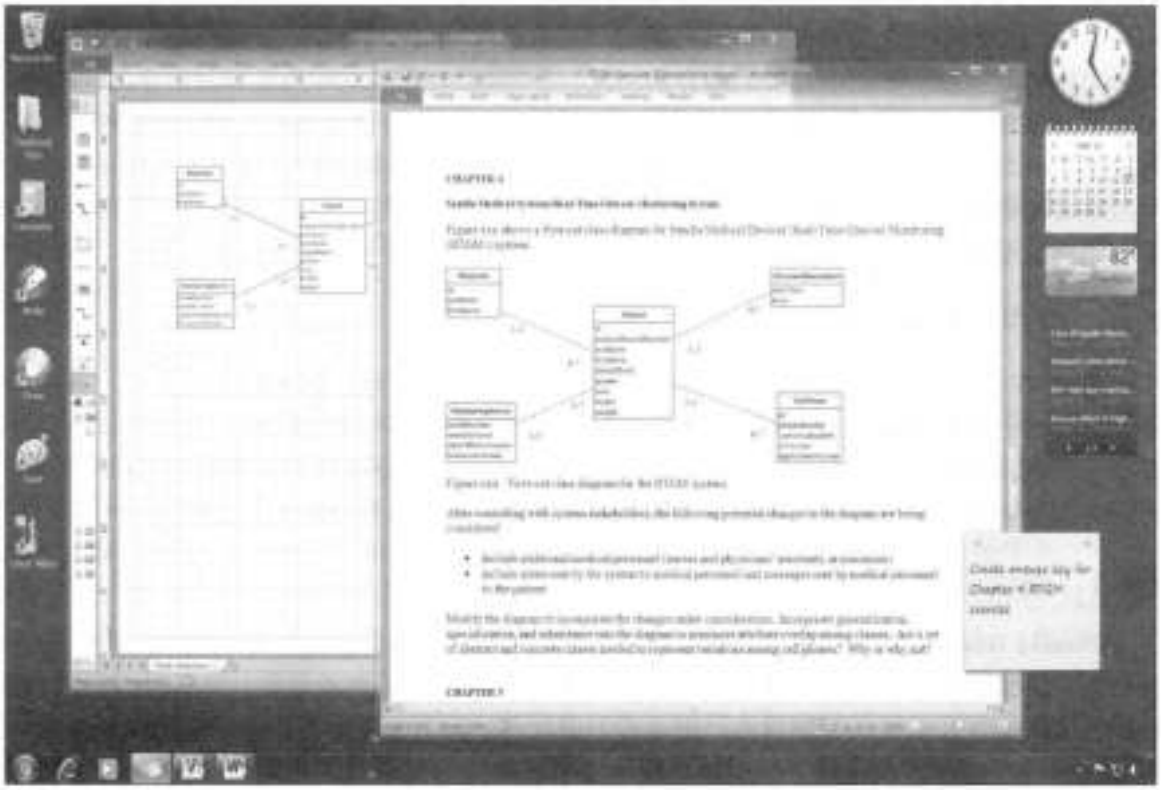


图 7-2　典型的电脑显示中的直接操作、桌面隐喻和文档隐喻

直接操纵、桌面隐喻和文档隐喻强调了与用户交互的被显示的对象。**对话隐喻**强调了发生在用户与计算机之间的通信。两个人之间的交流或对话中，一个人能听到并回复另一个人的问题和评论，这样信息就会按顺序交换。对话隐喻是思考人机交互的另一种方式，因为计算机要“听到”并且“回复”用户的问题或评论，然后用户也能“听到”并“回复”计算机的问题与评论。图 7-3 所示为用户与计算机之间的一个概念性对话。

图 7-3　对话隐喻表示用户与计算机之间的交互

在用户界面中可以用不同的方式来实施对话隐喻。一个直接的方法是在声音交流渠道中使用语音生成和语音识别，就像我们经常见到的大公司的客户呼叫号码支持。电脑化的声音

会询问一系列的问题、为每个问题列出答案以及回复答案。另一种对话隐喻的实施方式是使用用户通过文本和回复显示的问题或指令以及使用计算机通过文本显示的反馈问题。为了使用户输入的需求最小化，针对计算机问题的回复要被缩小到一系列特定可能性中，用户通过鼠标点击或触摸显示界面来选择最适合的回复。

无论实施的具体形式是什么，说明用户与计算机之间的对话是用户界面设计者使用的最有力的工具之一。尽管写出来的和说出来的语言在世界各地是不同的，但交流与对话是最基础且最普遍的人际技能。将用户与计算机之间的交互作为一个对话来建立模型，使得用户能够将语言与早期磨炼出的技术合并起来。

7.4　用户界面的设计概念

许多 IT 方面的调查者和参与者已经发布了能为用户界面设计提供指导的文章、书籍和网站。尽管一些指导随着用户界面技术的变化在变化，但是许多指导仍然是通用的，已经存在很长一段时间且形成了特定技术。我们先回顾一下在这部分中提过的一些通用指导原则，然后再接下来讨论用户界面开发，特定的界面类型有特定的指导原则。

7.4.1　提示性与可视性

Donald Norman 是**人机交互**（HCI）的主要研究员，人机交互是一个研究关于计算机系统的用户界面、面向人类的输入 / 输出技术和用户界面逻辑方面的领域。Norman 提出了两项确保人机交互友好性的关键原则：提示性和可视性。这两项原则都会应用在用户界面控件中，这是用户操作以执行任务的用户界面的要素。控制的例子包括菜单、按钮、下拉列表、滚动条和文本输入框。

提示性指的是一个特定控件的外观能体现其功能——也就是控件的使用目的。例如，一个看上去像方向盘的控件说明该控件是能转动的。提示性也能通过用户在另外一个环境中熟悉的用户界面控件来实现。例如，图 7-4 所示的媒体播放器控件图标在 20 世纪 70 年代首次被广泛运用在录音带和录像带中，并且不断地被用在像 DVD 和便携式音乐播放器这样的设备中。它们被广泛地合并在计算机界面中，因为有很多用户已经很熟悉它们了。

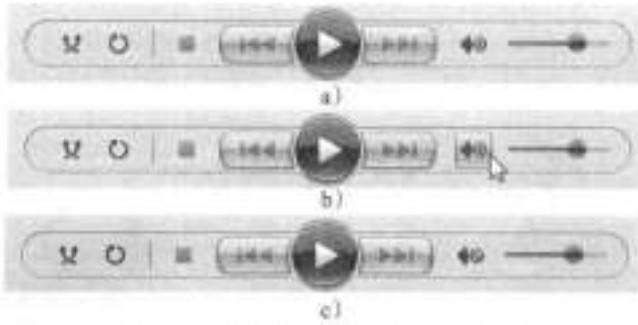


图 7-4　媒体播放器的可见性和控制度

可视性是指控件是可见的，因此用户才能知道它是可用的，它也指控件能产生及时的反馈来指示响应。例如，图 7-4a 中所示的静音键在用户移动鼠标将它关掉之后会改变它本身的外观，如图 7-4b 所示。当用户点击这个按钮时，它会改变它本身的外观，如图 7-4c 所示。

通用的平台能很容易地实现可视性和提示性的设计目标。如 iPad、运行安卓系统的手机或者是运行 Windows 操作系统的个人电脑。这样的平台拥有被良好定义的用户界面设计指导原则以及很多能被应用软件再利用的用户界面特征和功能的信息。当设计者从这些库中合并了用户界面对象和类型时，他就能在那些平台上的其他应用程序相似的用户界面中挖掘到用户经验。

网页用户界面设计就没有那么标准化，因为网页浏览器是有意要与平台无关的。设计者可以从各种各样的用户界面库中选择，每个都有自己的用户界面对象和类型。对提示性和可视性的关注在网页界面设计中是非常重要的，因为没有特定的标准能提供一个用户熟悉的已存在的框架。

7.4.2　一致性

用户界面的设计要注重功能和外观一致性。信息在窗体上的组织方式、菜单项的名称及其排列方式、图标的大小和形状以及任务的执行次序都应该是贯穿系统始末的。为什么要这样做呢？因为人都是有习惯的。我们学会一种做事方式之后就很难改变。当我们在操作一个计算机应用程序时，许多要采取的动作都以自动方式执行，我们不必考虑正在做什么。

图 7-5 所示为 Microsoft Word 的界面，它说明了应用程序之间多方面的一致性，这些应用程序在 Windows 操作系统下以及在 Microsoft Office 套件中的各个组件下运行。出现在许多 Windows 应用程序窗体左上角和右上角的图标都是标准化的，因此用户知道去哪里找到它们，并且看一眼就能知道这些图标的作用是什么。同样，右边的滚动条、放大或缩小滑块以及右下角的重置大小在很多 Windows 应用程序中都是同一个标准。Word、Powerpoint、Publisher 和其他 Office 组件中的顶部菜单和工具栏都是很相似的。有经验的用户会因为这些相似性从而更快地学习其他内容。

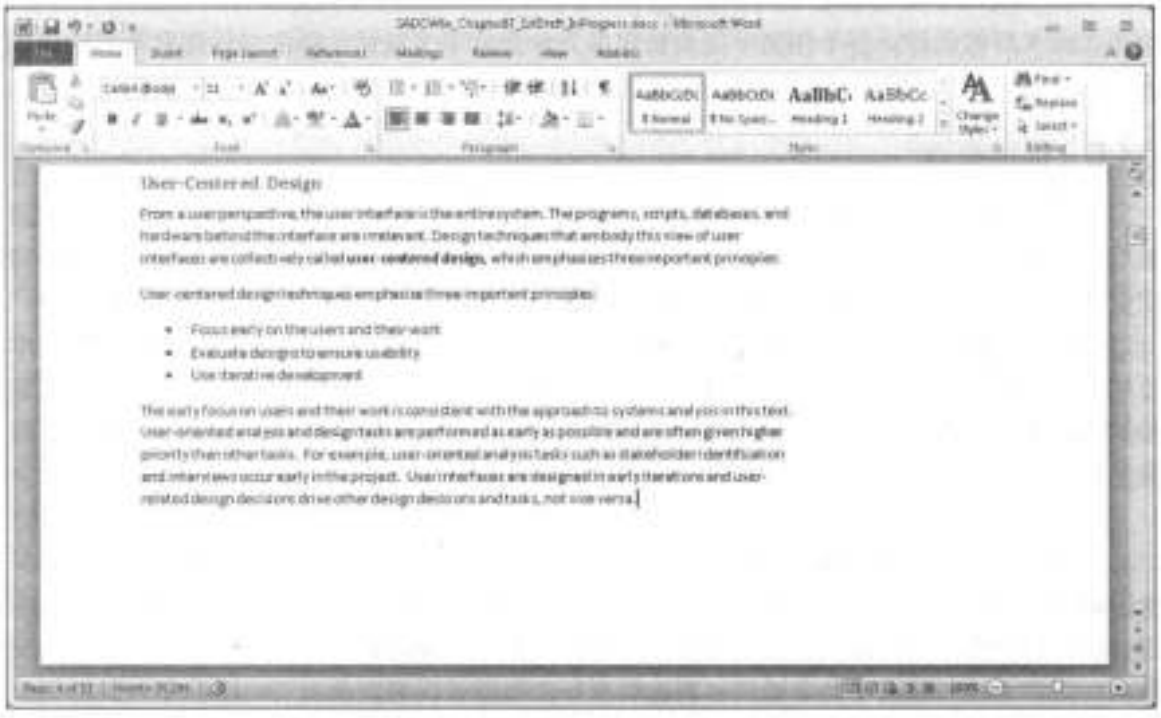


图 7-5　Word 的界面特征被用在大量的 Windows 应用上

7.4.3 快捷方式

为初学者设计的用户界面和对话对于有经验的用户来说是一种烦恼和障碍，因为这会降低他们的工作效率。不断重复或长时间处理同一个应用程序的用户会想要为经常使用的功能创建快捷方式，这就最大限度地减少了为完成任务而进行按键、点击鼠标和菜单选项的次数。语音命令以及快捷键这样的例子，如 Windows 键盘中复制的快捷方式是 Ctrl+C、粘贴是 Ctrl+V。应用程序设计者在应用时要使用标准的快捷方式或者创建自定义快捷方式。

7.4.4 反馈

用户采取的每个动作都会引起计算机做出某些类型的反馈，这样用户才能知道这个动作已被确认。在用户界面中，反馈可以有多种形式，包括：

- 声音响应，如当按下键时发出的点击声音和当屏幕上的按钮被点击时发出的声音。
- 可见的反馈，如图 7-4b 和图 7-4c 中所示的图标变化，或者是下载一个大文件时出现的进度条。

反馈为用户提供了某种意义上的保证，以及系统能正确做出响应和运行的感觉。缺少反馈会让用户疑惑这个命令或输入是否已被确认，或者这个系统是否出了故障。当延迟了一到两秒后，用户可能会重复点击控件或重新输入信息，这样会引起处理过程出错和用户受挫。

7.4.5 完整的对话

系统的每一次对话都应该有明确的次序——开始、中间处理过程和结束。任何定义完好的任务都有开始、中间处理和结束三部分，因此计算机上用户的任务也有相同的操作感觉。如果任务的开始和结束都不明确，那么用户可能会失去方向。此外，用户通常会一心一意地专注于某一任务，因此当明确该任务已完成时，用户就会清理思路并准备下个任务。

如果系统需求最初被定义为系统要响应的事件，那么每一事件触发某一特定处理过程，即预先定义好的活动。每个用例可能会被定义为一个或多个对话，每次对话都有步骤的流程和被定义好的交互过程。事件分解为实现完整的对话提供了平台。

7.4.6 错误处理

用户出错后要浪费时间提交错误并改正。一个好的用户界面设计要预先列出常见错误，然后帮助用户避免产生错误。一个方式是限制可用选项以及允许用户只能在对话框的某一特定位置选择有效项。前面也讨论过，充足的反馈信息也有助于减少错误。

当产生错误时，用户界面需要相应机制来探测。在本章后面的部分会讨论的有效项技术可用于捕捉错误，但是系统还必须帮助用户改正错误。当系统发现错误之后，错误消息应该特别说明出了什么错误，并且解释如何改正。下面这个错误是在用户输入一个新顾客信息后发生的：

输入的顾客信息无效，请重试。

这条消息并没有解释哪里出错或者下一步要做什么。而且，在消息显示后，系统如果清除输入窗体内容并重新显示的话，结果会怎么样？用户将需要重新输入先前输入的相关内容，而且对于做错了什么还是一头雾水。这是因为对错误不做解释，而且清除了已被输入的数据，用户无法判断出了什么错误。表达得更好一些的错误消息是：

输入的出生日期无效。请检查并在日期输入框中输入合适范围内的出生日期数值。

系统也要使正确的动作都合理化。例如，如果用户输入一项无效的顾客 ID，系统要提示用户发生了错误，并把先前输入的该顾客 ID 号显示在文本框中以供编辑。这样，用户能看到错误并编辑修改，而不必完全重新输入。系统也会建议使用基于过去经验的有效值或用户已经输入的其他信息。

7.4.7　撤销动作

用户需要能感觉到他们可以检查选项，并且可以毫不费力地取消或撤销相应的动作。试验是用户学习使用系统的一种方法。这也是防止出错的方法，如果用户发现自己出错就可以取消该动作。在棋子游戏中，只有参加游戏者的手指离开棋盘，相应的移动才算结束，用户在屏幕上的用鼠标拖放对象也是同理。此外，设计者应该要确保在所有对话框中包含取消按钮，允许用户在任一步骤上都可以回退。最后，当用户删除某些内容（文件、记录或事务）时，系统要询问用户来确认动作和有可能会延迟实施操作的情况。

允许撤销动作的一个关键问题是构建对话和一致的系统动作。设计新手和初级编程员通常会假定用户对话框架和相应系统动作的顺序是完全一致的。例如，复杂的事务可能需要多个独立的步骤，每个步骤从用户那里接收数据，另外一些步骤修改系统存储的数据。尽管用户对话应该要反映这个结构，但内部编程不需要处理接收到的数据。相反，它可能还要收集对话进行时的数据并建立一个内部“要去做”的列表。当用户完成对话中的最后一步时，系统可以立刻为最终对话完成处理过程。如果用户决定在对话的最后取消这个序列，那么没有内部变化可以撤销。此外，用户界面的执行也会被提升，因为在步骤与步骤之间的潜在处理过程延迟的情况也越来越少了。

7.4.8　减轻短期记忆负担

人有很多限制，短期记忆是最大限制之一。心理学家证明人在同一时间只能记住 7 条信息。用户界面设计者要避免让用户要记住在人机交互中一个接一个的窗体或者一个接一个的对话框的所有内容。如果用户不得不停下并询问“文件名是什么、顾客 ID 是什么、产品描述信息是什么”，那么就是系统设计给用户制造了太多的记忆负担。记忆限制也应用在一个复杂的处理过程的步骤中。这个界面应该帮助用户通过视觉提示和其他帮助来跟踪一个复杂的处理过程。

7.5　从分析到用户界面设计的转换

当确定并记录好用例时，用户界面设计的基础就已经制定了，如第 3 章所描述的那样。需要用户直接交互的用例是对话的开始，同时相应的用例图、活动图和系统顺序图是初步的对话文档。交互用例可能需要用户输入选择和数据到系统中（如制作网上订单时），或者要生成输出来响应用户请求（如跟踪运送情况时）。在设计阶段中，交互用例的对话会通过开发菜单、表格和其他用户界面要素来进一步修改。

对话和用户界面设计按照自上而下或自下而上的流行方法来进行。在自上而下这个方法中，菜单（一组相关的用例、对话和用户界面）会首先定义，接着就是制定每个交互用例的详细描述和相关用户界面要素的开发。在自下而上这个方法中，首先要开发的是交互用例，然后相关对话和用户界面是同时开发的。菜单是在项目后期添加的，也就是在相关完整的且被实现的用户界面被完成后。没有一个方法是完美的，对于一个具体项目来说总会有一个或

两个方法是合适的。

7.5.1　用例和菜单层次

菜单是一种组织用户界面中大量的相关用例或对话的方式。在除了最小的系统之外的所有系统中，向用户呈现的菜单数量应当是每个屏幕容易处理的数量。菜单也用于组织相关的功能以便用户可以更容易的定位，还用于为复杂的事件分解准确排列相关界面。图 7-6 所示为两种不同类型的菜单。在图 7-6a 中，鼠标所在的地方是网页顶部菜单中商店条目部分。单击 Shop 会显示第二个包含另外一个菜单的网页。图 7-6b 显示了一个更复杂的菜单设计，它有三个菜单层。

a）

b）

图 7-6　两种不同的菜单类型

设计者要如何决定哪个用例和用户界面包含在哪个菜单中，哪个菜单是需要的，以及需要多少菜单层次？这些决定首先会通过用例或菜单选项的数量以及人类识别的限制来考虑。菜单通常包含 5 ~ 10 个选项，以避免用户难以承受（如图 7-6a）。有了仔细的设计过程，更多选择被提供了，尤其是针对有经验的用户来说（如图 7-6b）。对于一个典型的业务系统来说，将交互用例的总数除以 5 可初步提供一个估计的菜单数量，包含所有交互用例和允许的额外菜单项，如设置选项或偏好。如果菜单数量大于 10，那么包含与其他菜单的连接的高层次菜单是需要的。例如，图 7-6b 中的顶部菜单选项，如文件、开始和审阅，每个与第二层次的连接都设计为一个工具栏。在工具栏中的一些选择是连接到更低级菜单中的，如图中显示的 Markup 菜单。

有着常见参与者和事件分解的用例，或者是为一个特定域类实施 CRUD 动作的用例，都是组织成一个简单菜单或相关菜单组织的候选。例如，表 7-2 所示的 RMO 的 CSMS 系统的用例。通过参与者和子系统而初步组织的这些用例是菜单设计的一个很好开端。

表 7-2　RMO 的参与者和子系统用例的集合

子系统	用例	用户 / 参与者
销售	搜索商品	顾客、客服代表、店铺销售代表
销售	查看产品评价和排名	顾客、客服代表、店铺销售代表
销售	查看配套商品组合	顾客、客服代表、店铺销售代表
销售	加入购物车	顾客
销售	清空购物车	顾客
销售	检查购物车	顾客
销售	加入备用车	顾客
销售	清空备用车	顾客
销售	转换备用车	顾客
销售	创建电话销售	客服代表
销售	创建店铺销售	店铺销售代表
订单实施	运输商品	运输部
订单实施	管理运输人员	运输部
订单实施	创建延期订单	运输部
订单实施	创建退货商品	运输部、顾客
订单实施	查看订单状态	运输部、顾客、管理部
订单实施	跟踪运输情况	运输部、顾客、市场部
订单实施	排名及评价产品	顾客
订单实施	提供建议	顾客

表 7-3 所示以表 7-2 中的用例而设计的四个菜单。每个菜单从一个子系统中为顾客或内部销售代表收集用例。菜单选项的数量范围是 4 – 7，不会超载任何一个菜单，可以一次显示多个菜单级别。每个菜单选项都要创建对话设计。在对话设计产生之后，设计者可能要再定义菜单选项或结构。事实上，设计者通常会在用户界面设计时发现遗失或不完整的用例，这会导致要暂时返回到分析活动中来完成分解。

表 7-3　RMO 的 CSMS 系统通过相似的功能和用户集合成的初步菜单

菜单描述	菜单选项（用例）	目标用户
购物车功能（初始或备用）	搜索商品 查看产品评价和排名 查看配套商品组合 转换车（从初始到备用，反之亦然） 加入购物车 清空购物车 检查购物车	顾客
销售创建	搜索商品 查看产品评价和排名 查看配套商品组合 创建销售	客服代表、店铺销售代表
订单运输	运输商品 管理运输人员 创建延期订单 创建退货商品 查看订单状态 跟踪运输情况	客服代表、店铺销售代表

（续）

菜单描述	菜单选项（用例）	目标用户
顾客订单控制	查看订单状态 跟踪运输情况 排名及评价产品 提供建议	顾客

菜单包括的选项通常包含那些不在事件列表中的活动或用例。许多选项与系统控制有关，如账户维护或数据库备份与恢复，这些都会在后面进行讨论。其他被添加的条目包括帮助连接以及连接到其他菜单或子系统。

7.5.2　对话和故事板

在确定了所有用户需求的对话之后，设计者必须记录下这些对话。许多选项是存在的，但目标还不存在事实标准。一种方法是列出对话中的关键步骤并附带每一步骤中用户和计算机交互内容的文字描述。书写交互步骤的形式可以依据第 2 章中描述的活动图和第 4 章中的用例描述形式，也可以更加自由。

假设用户和系统是作为两个人以交流的方式进行对话，那么设计者也可以通过写出他们是如何交互的来记录对话设计。有时，设计者可以直接从对话框中抓取一系列的屏幕画面作为更具可视性的设计。想一想 RMO 的 CSMS 系统中检查购物车这个用例。系统和用户之间的对话可能是这样的：

系统：您想做什么？

用户：我想检查一下。

系统：好的。您的电子邮件或者账户 ID 是什么？

用户：我的电子邮件是 nwells22@gmail.com。

系统：好的。您是住在 Silicon 大道 1122 号的 Nancy Wells，对吗？

用户：是的。

系统：您购物车中所有的商品都是有存货的，并且今天就可发货了。因为您的订单小计超过了 100 美元，所以用 UPS 地面运输方式（3 ~ 5 天到货）是免运送费的。其他购物选项包括下一天到货（35 美元）、2 天到货（20 美元）以及 UPS 包裹邮寄（11.70 美元，5 ~ 7 天到货）。您要选择哪种运送方式？

用户：免费的 UPS 地面运送。

系统：记录中您的地址是 Silicon 大道 1122 号，您想要运送到这个地址吗？

用户：不。

系统：那您想要订单运送到哪个地址？

用户：John Wells，Jefferson Street NE，Albuquerque，NM87123。

系统：好的，总费用是 125.56 美元（117.90 美元的商品小计加上 7.66 美元的销售税）。我在您登记的信用卡上扣除这笔费用了（尾号是 0899 的一张 Visa 卡），行吗？

用户：好的。

系统：您的支付已被批准，同时您的订单正准备发货。一封确认邮件会发送给您，再过一会儿等订单发送之后，会再给您发送带有运送跟踪号码的另外一封邮件。还需要别的帮助吗？

用户：没有了。

这样的场景有很多种可能实现的方式。在决定一个实施之前，分析员通常使用一种**故事板**的技术——即在对话中显示一系列的屏幕概略图。这些概略图不必包含体现基本设计概念的诸多细节。设计者可以用 VB 这样的 Visual 编程工具来实现故事板，但是使用图形工具包绘制出的简单概略图能帮助设计者始终集中在基本设计思路中，并且避免将设计的重心偏向一个具体应用程序开发工具的能力。

图 7-7 所示为基于检查购物车这个用例的对话的故事板。屏幕格式是很简单的，尽管有充足的细节显示所有被提出和由用户输入的信息。用户和设计者可以审查这个故事板来确定遗失或无关的信息，以及讨论最终实施过程中不同的选项，而最终实施过程是基于在大屏幕上显示的网页、传统窗体对话或一个移动设备上 APP 的用户界面。

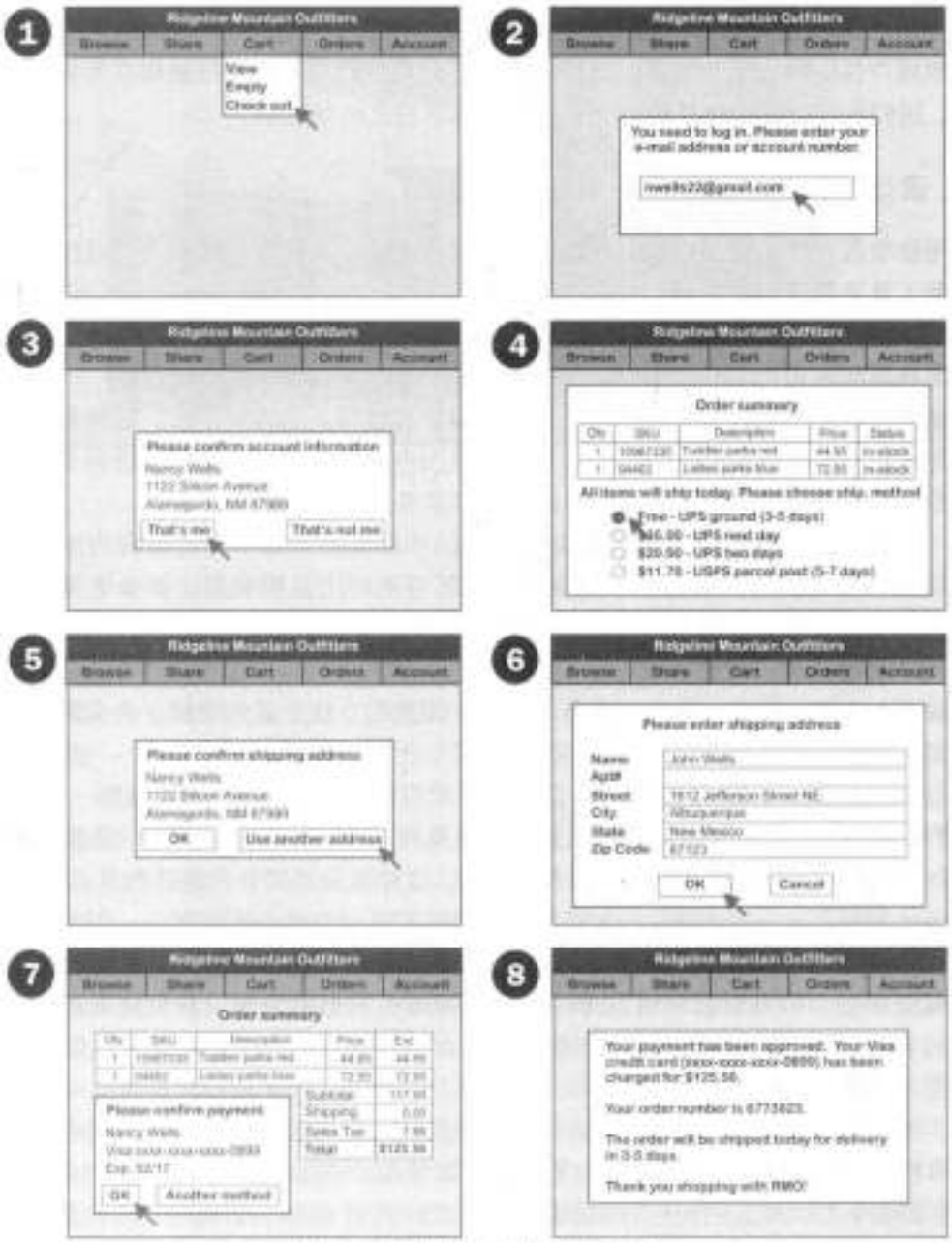


图 7-7　检查购物车对话的故事板

7.6　用户界面设计

随着计算设备的激增，可用设备和技术的用户界面也在更快、更多样化地增长。从用在台式机上形状较大的平板显示器到用在平板电脑和手机上的小型显示器。显示器可以附带简单声音，如点击声、哔声、音乐或语音。用户输入可以通过语音、触摸屏、键盘和鼠标或者是数字成像被捕捉。

随着用户界面技术范围的增长，创建多个用户界面的需求也在增长。例如，电子商务应用程序针对台式机有一个用户界面，对笔记本电脑而言有一个大的显示器，对手机来说是一个小型显示器，有时还要加上中型显示器（如 iPad）。在相似设备中的用户界面也可能是不同的。例如，目标是手机的用户界面可能是运行在手机中网页浏览器上的版本，或者也可能是基于 iPhone 和安卓系统手机的不同版本的定制 APP。

虽然用户界面的大小和能力的范围很宽泛，但是它们的一些特征被用在几乎所有的计算设备上。我们先开始讨论常见的特征，然后探究它们之间的区别。

7.6.1　设计窗体和格式的指导原则

在用故事板或其他技术确定了对话、菜单和格式后，系统开发人员可以通过使用一种可用的原型工具来构造用户界面。在界面设计阶段要考虑的主要问题包括界面布局和格式化、数据键控和输入以及导航控制。

界面布局和格式化

高质量的界面都是有精良布局的，包括能很容易地确定和理解字段。确保界面精良布局的最佳方法之一是设计不同的替代原型，然后让用户测试。用户会让你知道哪个特征是有帮助的，哪个是分散注意力的。在设计界面时，你要考虑以下内容：

- 一致性——系统中所有格式都要有一样的外观和感官效果。功能键的连贯使用、快捷键、控制按钮、颜色和布局能使系统显得更好用且更专业。如果是为支持操作系统的界面做设计（如 Windows、iPhone 或者是安卓），要遵循已有的指导原则来提高 APP 和格式的一致性。
- 标签和标题——标签也应该是易于识别和阅读的。位于界面顶部、含义清晰的标题有助于将混淆格式使用的可能性降到最低。
- 分布与组织——相关字段通常会被放在相邻位置，并且能被组织进一个文本框中。Tab 顺序（光标的移动或者输入点）应该遵循用户常用的阅读顺序（从左到右，在美国和欧洲是从上到下）。还要留出空白区，这样能易于区分和阅读内容。
- 字体和颜色——字体及大小的变化能帮助用户区分窗体的不同部分，但是对于大屏幕来说少量的字体及颜色变化就够了，对小屏幕就要尽可能少。太多变化可能会分散视觉注意力以及引起眼睛疲劳。颜色也是遵循相似的原则。避免使用过多颜色，同时要确保颜色的互补。此外，要小心混合蓝色和黄色、红色和绿色，以免造成色盲用户阅读上的困难。

图 7-8 所示为顾客访问 RMO 网站时显示的主页。格式包括靠近顶部的两个菜单栏，这两个菜单栏组织了网页相同部分的相关功能。如果用户点击进入 Shop for Clothing 这个条目，会立刻在下方以相近的配色方案和字体显示出一个子菜单。如果用户点击进入 Shop for Gear 这个条目，Shop for Clothing 条目是以蓝色为背景，字体变成白色，后面的子菜单条

目则以相反的方式显示。菜单条目的标签排列稀疏，并且使用易读字体。除了 logo 和图片，这个页面使用了一小部分互补的颜色。标题被放在靠近图片顶部的地方，并且在图片背景和其他页面元素中被突显了出来。

图 7-8　RMO 主页

数据输入

用户界面中常用的数据输入控件的几个类型包括：

- **文本框**——接收由键盘输入或从语音输入中被识别的文本的矩形框，如图 7-9 中的 Product ID（产品 ID）。
- **列表框**——包含预先定义好的数据值列表的文本框，如图 7-9 中的 Size（大小）。
- **组合框**——包含预先定义好的可接受输入列表的文本框，但是在列表中没有包含预期值时，能允许用户输入一个新的数值。
- **单选按钮**——一组选择，但是用户只能选择其中一项，一旦用户选择某个选项后系统将自动关闭其他所有按钮，如图 7-7 中屏幕 4 里的运输方法。
- **复选框**——与单选按钮相似，但是用户可以选择多个选项。

这些数据输入控件是为 Apple Macintosh 开发的，之后被 Windows 和其他操作系统所采用。早期的网页浏览器标准已经限制了数据输入控件的支持，但是现在的版本能支持先前描述的控件和很多其他控件。iPhone 和安卓界面支持的是相似的控件。

图 7-9 中的表格可以被 RMO 的员工用来查找产品信息或修改目录中的信息。要注意的是如何用标题和标签来使表格易读。表格的自然流是自上而下的，与相关字段在一起。导航和关闭按钮能很容易找到，但不是数据输入活动的方式。这个表格使用标准的 Microsoft Windows 控件，包括文本框、列表框和组合框。尽管在图中不可见，但这个表格还包括为用户完善的特征，这些特征包括标准 Windows 键盘快捷方式、自上而下和从左到右的 Tab 顺

序、基于数据库查找，以及部分已输入文本补丁的一些字段的自动完成。

图 7-9　通过微软的数据输入控制查询 RMO 产品的详细界面

导航与支持控件

标准 Windows 界面提供几种用于导航和操纵窗口的控件。对于微软公司系列的应用程序，这些控件包括窗体右上角的最小化、最大化和关闭按钮，水平和垂直滚动条等（图 7-5 是具体例子）。为了维护应用程序之间的一致性，要求尽可能地使用内置的或标准化的导航控件。

7.6.2　网页浏览器用户界面的附加指导原则

大多数用户界面设计人员首先要学习开发基于 Web 的界面，这个界面是在网页浏览器中操作的，如 IE 浏览器、Mozilla 浏览器、谷歌浏览器或是 Safari 浏览器。随着 Web 技术和标准的成熟化，基于浏览器的界面（如图 7-11 所示）与那些使用操作系统支持库的界面（如图 7-9 所示）在能力上的区别都消失了。在许多方面，基于浏览器的界面已经变得更有力了。尽管如此，在设计网页和基于浏览器的形式时还是要考虑某些区别。

一致性

一致性在网站中是特别关键的，因为大多数网站包含大量的页面，这些页面为很多不同目的和目标的大众服务。例如，一个典型的企业网站提供电子商务功能（如在线订购）、信息投资者、目录与公共联系信息以及像说明书和手册这样的产品信息。其实，一个企业网站是通向为许多不同用户和任务服务的系统集合的大门。尽管有各种各样的用户和任务，但是网站作为一个整体是一个简单的系统，这个系统既能支持简单的外观和感觉，也能为企业这个整体设计一个一致的、吸引人的和令人满意的图片。

大多数企业花费大量的资源来开发和维护他们的网页并确保它们之间的一致性。因此，网站特定部分的用户界面设计者必须遵循企业的总体设计规范。**层叠样式表**（CSS）是网页编码的标准，同时它们也使网站设计者能够区分那些看上去总是相同的页面部分，以及那些随着任务或受众不同而产生区别的部分。它们也能限制“不同”页面部分中的选项，包括工

具栏、菜单、字体、颜色和背景图片的布局与外观。

图 7-10 和图 7-11 显示了 RMO 网站的附加页面，这个页面是为了显示顾客搜索商品和完成订单的。菜单、轮廓周围的详细内容以及选项的字体和颜色都是通过 CSS 设置的。当用户选择菜单条目或点击链接或控件时，所示的页面会再利用这些要素来确保一致的外观和用户交互。

图 7-10　RMO 的产品详述网页界面

图 7-11　RMO 的购物车网页界面

性能考虑

一般的网站和特殊的基于浏览器的形式对应用程序设计、用户计算设备和托管站点的服务

器之间的网络连接质量是很敏感的。当用户点击超链接或与超链接有相同作用的控件时，浏览器会发送用户输入的信息（无论是什么）到网站服务器，并且伴随着对新页面的请求。贯穿多个网络的信息会通过服务器接收并处理，然后会通过网络将回复发送回来（即显示一个新页面）。点击超链接和被请求页面的显示之间的延迟取决于要传输数据的总量、用户计算设备的显示与网络连接速度、网络携带信息的性能以及为了那个网络性能而竞争的其他用户与应用程序的数量。

在用户计算设备与服务器中传输的信息数量中会产生一次交换，并且它还要花费时间来更新页面，传输越多信息，延迟的时间越长。在因特网中的交换对通信来说是非常重要的，尽管当用户的台式机和服务器在共享高速连接时，它在企业网络中也是一个重要问题。

信息数量与包含在网页和基于 Web 的应用程序执行过程的其他数据之间也会发生交换。带有大量信息内容或嵌入式编程的页面可以避免或推迟更新。例如，一个包含空白订单格式的页面可能会很小。但是如果浏览器必须与服务器交互来验证每个用户的输入，那就会需要许多页面的更新。如果页面包含的格式也包含嵌入式编程，那么许多服务器的交互和页面更新就能被避免。然而，如果用户计算设备性能不够优秀，那么页面的初步下载就需要很长时间，因为要下载更多的内容和验证程序（如一个相对便宜的手机）。

基于 Web 的用户界面设计必须执行一个小心的平衡动作，当用户从一个页面移动到另一个页面时，在页面中提供嵌入式“智能”来避免更新，而不是过载的页面内容，以此来避免长时间的延迟。深入的测试是确保已经找到合适平衡的最优方法。例如，在图 7-12 中，RMO 网页包括菜单条目列表（如 Shop for Clothing）和子菜单（如 Women's Apparel）。当用户点到菜单条目时，子菜单内容会自动出现，不会有页面更新的需要。将这个与图 7-6a 中的菜单类型进行对比，在图 7-6a 的菜单中，用户必须点击一个菜单条目，然后等待下个层次菜单下载完成，并显示到一个新页面上。RMO 的页面由于它的嵌入式菜单内容和相关程序，所以要花费很长时间去下载，但是它能避免页面更新和在显示子菜单时的相关延迟。

图 7-12　RMO 的主页，有三级菜单

图片、视频和声音

基于 Web 的界面通常喜欢将它们固有的能力与文本、图片和声音相结合。有力且引人注目的界面可以被构造，并且它们在面向顾客系统中是特别重要的。然而，声音与图片的大量使用加剧了先前讨论过的性能问题，并且也会导致兼容性问题的产生。高性能对视频和高分辨率静态图像来说最有意义，但它要消耗大量的网络容量。例如，图 7-8 中的背景图在一台笔记本或台式机上为了高质量的显示，将需要 100kb 的已下载数据。对于手机的小屏幕就会用小一点的图片来代替。

兼容性问题因为声音和视频而上升，因为有它们有太多的编码方式。大多数网页浏览器依靠附加组件（有时被称为插件）来播放声音、音乐和视频。不幸的是，所有的插件不会和所有的浏览器兼容，尤其是版本老的浏览器。因此，网站设计者必须小心地选择使用的格式和插件。在很多情况下，设计者必须为不同插件创建不同页面，为当前插件编程负责的程序来查询适用的浏览器，这样才可以下载正确的页面。

残疾用户

所有用户界面的设计者必须对残疾用户的特殊需求很敏感。因为网页在如今现代化的世界中是基本资源，为了确保那些有视觉障碍或行动不便的人也能使用网页，人们已经开发了相关标准。视觉障碍用户要通过语音合成软件来与系统交互，而这个语音软件是要检测网页上的文本内容然后再读出来。行动不便的用户通常使用语音识别软件来浏览网页中的内容，执行那些在正常情况下使用键盘、鼠标或触屏完成的任务。这两种软件都是被称为**辅助技术**的通用软件类中的例子。

万维网联盟（W3C）是为网页的许多方面设置标准的一个组织，包括辅助技术的兼容性。在 2010 年 6 月，它发布了用户代理易用性指导原则（UAAG）的工作草案 2.0 版本。尽管这份草案在撰写本书时还没有通过最终批准，但还是有许多组织使用了该指导原则或那些早期拟定的标准来指导他们基于 Web 的用户界面设计。

7.6.3　手持设备的附加指导原则

为手持设备设计网页和基于 APP 的用户界面的额外设计挑战包括：

- 小屏幕。
- 小键盘及触摸屏。
- 有限的网络容量。
- APP 设计指导原则和工具箱。

在 2012 年，一个手机的尺寸大约是 3.5 英寸 ×2.25 英寸、480×320 像素。小屏幕为显示内容提供了有限空间。因此，设计者必须减少用户界面的内容来确保可读性，同时也要避免充斥屏幕。图 7-13 所示为一个手机版 RMO 网页的例子。与先前的大屏幕版相比较，移动版的页面删除了许多内容（包括所有的图片），只显示一个按比例缩小的商标图片。剩下的文字内容被缩小了，为了确保可读性，将特别注意力放在了对比和布局上。

图 7-13　RMO 的手机网页布局

2012 年，大多数手机都包含手机网络与无限网的连通性。大多数当前的手机网络都被描述为 3G 网络，这种网络的初始设计是针对声音通信，之后加入了数据通信。3G 网络

的吞吐量通常不会多于无限网络吞吐量的 1/10。在 2012 年，美国开始部署 4G 网络。4G 网络增加了 WiFi 的数据吞吐量，尽管许多用户为连接到那种带宽而互相竞争。

因为手机数据吞吐量少于其他计算设备，所以之前描述过的性能问题已经变成了越来越重大的设计约束。页面大小必须被限制，以此来获取可接受的下载和页面更新率。高分辨率的图像只在必要时会使用，并且带宽消耗视频是完全被禁止的。对于 RMO 手机版网站，背景图是避免使用的，高分辨率的图片只会在顾客想要查看产品细节时才使用。

一些组织部署为顾客开发的 APP，即用户可以安装在他们的移动设备上的应用程序。那些 APP 是在手机操作系统上运行的，如 iPhone 操作系统、iPad 操作系统或者是 Google 安卓操作系统。每个操作系统都为用户界面开发者提供了一个工具箱和一系列的开发指导原则，这些能确保 APP 之间的最大兼容性。无论何时，用户界面开发者都应该使用这些工具箱和指导原则。

7.7 确定系统界面

用户界面包括了需要系统用户直接参与的输入与输出。但是还有许多其他系统界面，处理输入、与系统进行实时交互和用最少的用户干预分配输出。我们将系统界面广义地定义为不需任何用户干预或用户干预很少的输入和输出。为人们设计的输出界面通常包含以下术语：账单、报表、打印表格和流向其他自动化系统的电子输出。同时也包含自动化输入和来自于非用户界面设备的输入。例如，来自自动扫描仪、条形阅读器、光字符识别设备以及作为系统界面组成部分的其他计算机系统的输入。

信息系统完整的输入和输出如图 7-14 所示，描述如下：

- 来自于其他系统的输入和流向其他系统的输出——这些是与其他信息系统接触的直接接口，通常被格式化为网络消息。电子数据交换（EDI）和许多基于网络的系统可以通过消息与其他系统实现集成。例如，在 RMO 公司的集成供应链管理和客户支持系统中，来自供应商的物品的到达会触发客户的延期商品的运输操作。
- 高自动化的输入与输出——这些是通过设备（扫描仪）捕获的，或者是通过人来开始一个不会产生进一步干预的处理过程形成的。例如，仓库中的商品在通过传动带移动时，会通过一个条形码扫描仪，这个扫描仪能记录它的位置。此外，月末报表能通过高自动化系统进行打印和邮寄，这个系统能将报表放在信封中、应用邮资、通过邮政编码分类然后将他们交付给邮政局。
- 来自于外部数据库的输入与输出——这些是指提供输入给系统或从系统中接收输出。EDI 消息更常用，但是与其他系统数据库的直接交互可能更有效。例如，RMO 的采购系统可以直接将产品订单放入供应商数据库。

EDI 的主要挑战是确定交易格式。例如，通用汽车公司是最早应用 EDI 的公司，它有数千个供应商，当然会带来数千笔交易。更复杂的是，每个供应商可能通过 EDI 同数十个或上百个顾客相联系，而这些客户仍有可能通过 EDI 相连。所以即使一笔单一类型的交易也可能需要一打或更多的规定的交易格式。建立并维护一个 EDI 系统需要很高的代价也就不足为奇了。即便如此，EDI 系统仍然比采用文件交易格式更有效且更有影响，因为后者必须被打印和重复输入。

现代化的 EDI 消息被格式化为**可扩展标识语言**（XML）。XML 是对 HTML 的扩展，它也在文字信息中嵌入了自定义数据结构。因此，包含数据字段的事务可以以 XML 代码的方式发送，以定义数据字段的含义。许多较新的系统正使用这种技术提供一个系统到系统的公共接口。图 7-15 说明了一个可以用来在系统之间传送客户信息的简单的 XML 事务。数据被

XML 标记所围绕，如 <name>、</name>，它定义了开始、结束和中间的文本。

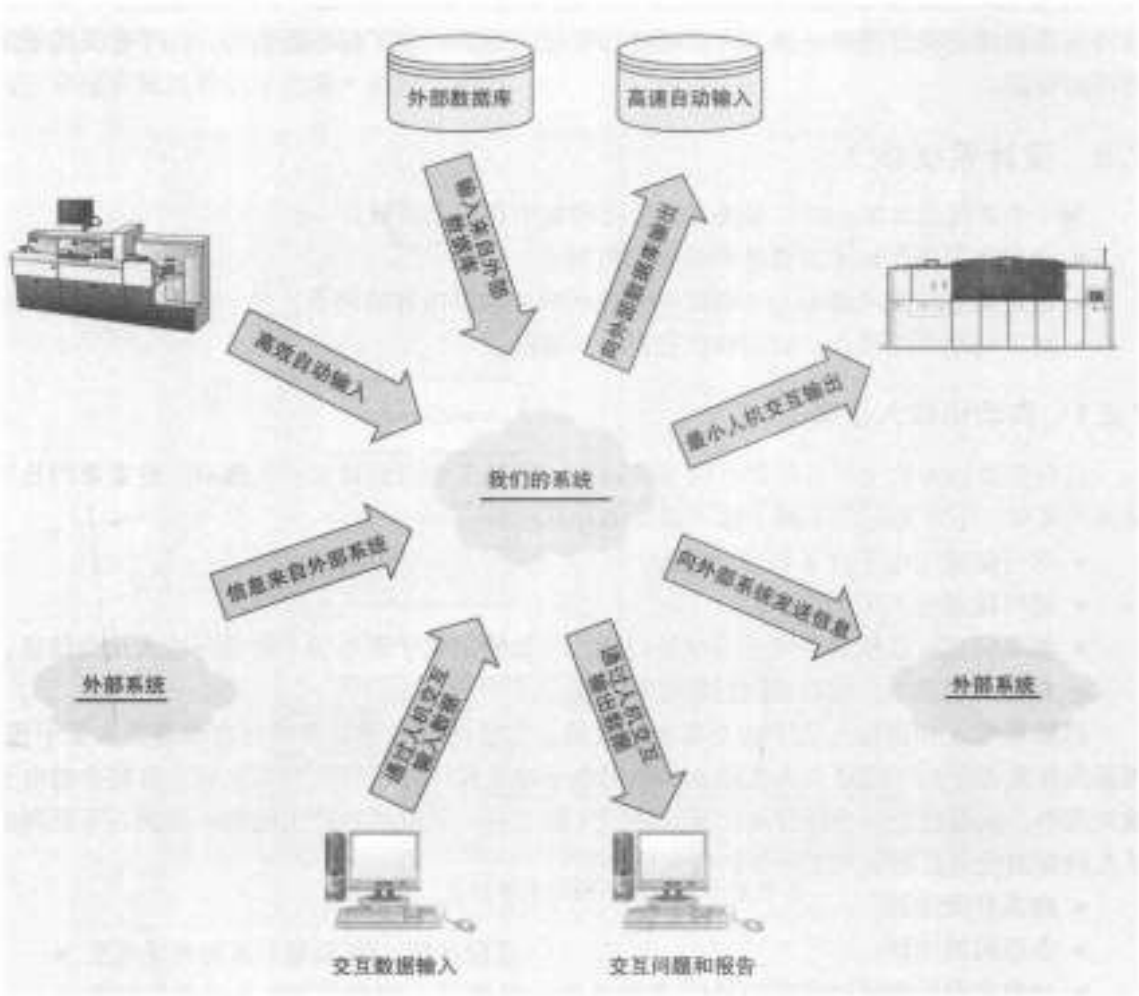


图 7-14　整个信息系统的输入和输出

```
<customer record>
      <accountNumber>RM010989</accountNumber>
      <name>William Jones</name>
      <billingAddress>
            <street>120 Roundabout Road</street>
            <city>Los Angeles</city>
            <state>CA</state>
            <zip>98115</zip></billingAddress>
      <shippingAddress>
            <street>120 Roundabout Road</street>
            <city>Los Angeles</city>
            <state>CA</state>
            <zip>98115</zip></shippingAddress>
      <dayPhone>215.767.2334</dayPhone>
      <nightPhone>215.899.8763</nightPhone>
</customer record>
```

图 7-15　客户信息的 XML 格式化语句

XML 被称为可扩展语言是因为用户可以定义任何他们想要使用的标记。对于基于 XML 的 EDI 来说，系统必须能辨识出标记，但是建立完一系列代码之后，交易就包括许多不同的格式且仍然能被辨识和处理。许多行业和专业组织都成立了标准委员会，用于定义为 EDI 使用的标记。

7.8 设计系统输入

为一个系统设计输入时，系统开发者必须集中在三个区域：

- 确定将要用作输入的设备和采用的机制。
- 确定所有的系统输入，并拟定一个包括所有数据内容的列表。
- 对于每个系统输入，确定哪些控制是必需的。

7.8.1 自动化输入设备

任何数据输入的主要目的都是向系统输入或更新无差错的数据。在这里，最重要的是不要出现差错。下面是有助于减少输入错误的几点经验：

- 尽可能使用电子设备和自动输入。
- 尽可能避免人工干涉。
- 如果信息可以从某个电子表单处得到，那么使用电子表单而不要重新输入这些信息。
- 在输入信息时，对数据进行验证和更正。

自动化输入和避免人工干扰是紧密相关的，尽管使用电子设备不能自动避免人工干预。当系统开发者仔细考虑避免人工输入而使用电子输入媒体时，得到的系统就含有较少的电子输入表单，从而避免一个最普遍的输入错误来源之一：通过用户产生的键入错误。下面列出了几种普遍使用且避免人工干预的输入设备：

- 磁条片阅读器。
- 条形码阅读器。
- 光电字符识别阅读器和扫描仪。
- 无线电频率识别标签。
- 触摸屏设备。
- 电子笔和书写板。
- 数字化仪，如数码相机和数字音频设备。
- 语音识别软件。

下一个减少错误的原则是尽可能重复使用计算机已有的信息。例如，思考一下没有要检查的行李的顾客的自动化航线签入过程。顾客刷信用卡或驾驶证，然后系统查询内部数据库或外部数据库来确定顾客并检索保存的信息。已保存的信息会显示给顾客进行确认。因为检索到的信息基本上都是正确的，所以数据输入的任务降为磁卡扫描和在触屏上点击按钮，消除手工数据输入和相关的错误率。如果显示的数据是不正确的，最终减小误差的原则是让顾客直接输入正确数据。

7.8.2 定义系统输入的细节

分析员用来确定用户和系统输入的基本方法是在分析活动中开发的文档中搜索贯穿系统边界的信息流。分析员检查系统顺序图来确定每个活动或用例中引入和流出的消息，同时利

用设计类图来确定和描述数据内容。

图7-16是面向对象版本的工资单系统中的一个部分系统顺序图。不同的用例相互结合来巩固简单图中的主要输入。贯穿系统边界的消息确定了输入——系统输入和用户界面输入。穿过系统边界的三个输入是：

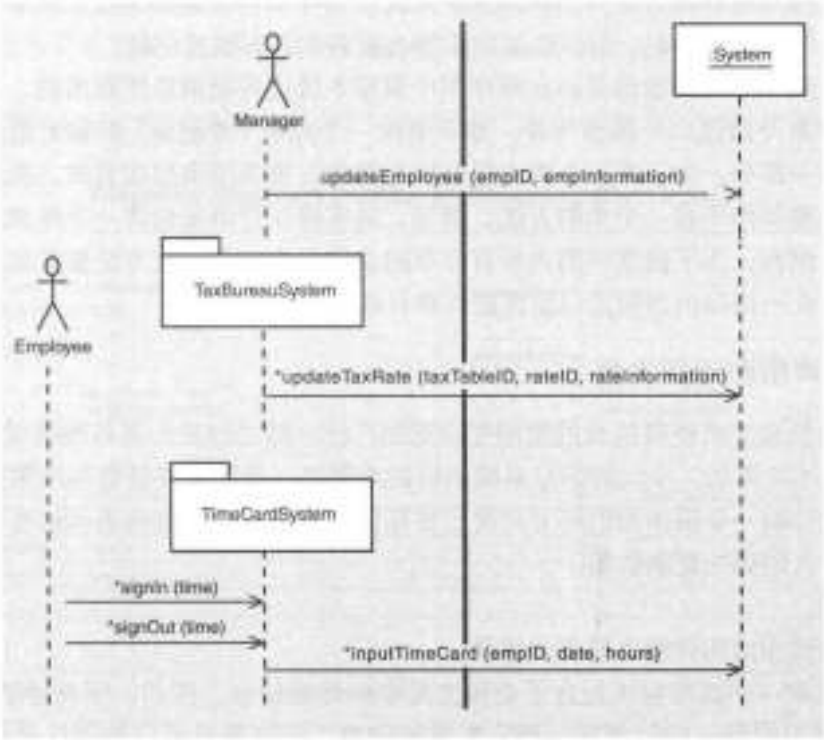


图7-16 支付系统用例的部分系统顺序图

- 更新雇员信息（雇员ID、雇员信息）。
- 更新税率信息（税率表ID、税率ID、税率信息）。
- 输入时间卡信息（雇员ID、日期、时间）。

第一个输入是用户界面的一部分。另外两个输入来自外部系统，并且不需要用户参与。来自税务局的信息要么以实时消息的形式发送，要么以CD或其他电子设备上的输入文件的形式发送。时间卡信息可以以各种方式进入系统，可以通过电子读卡器直接读取时间卡的方式，也可以通过子系统输入的方式，如员工在下班打卡的时候，时间信息便被录入。后面两个输入消息需要准确定义，包括传输方法、内容和格式。这里要注意的是顺序图提供了用户与系统输入的一个详细观点来支持用例和类似的业务事件。

7.9 设计系统输出

系统输出的主要目的是在正确的时间和地点为正确的人提供相关信息。在这个活动中的任务集中在四个方面：

- 确定每个系统的输出类型。
- 为应用设计所要求的特定输出制作一份列表。
- 提供必要的控制来保护输出信息。
- 设计输出的布局并为其建立原型。

前两个任务的目标是评价不同的输出信息候选方法，并为每个需要的输出设计最合适的方法。在分析阶段中，必要的系统输出通常作为系统需求建模的一部分被确定下来。在设计阶段，主要任务是使那些输出的产品与在应用程序结构化设计中被确定的方法相互协调。

第三个任务是确保设计者为组织和保护系统已经评估了信息的价值。通常，组织机构对输入和系统访问实现了控制，但往往忽略了输出报表也含有敏感信息。

正如系统输入那样，输出是由在顺序图中贯穿系统边界的消息所指出的，系统边界是指从内部系统对象开始流向外部参与者。基于单独一个对象（或记录）的输出消息通常是那个对象类方法的一部分。为了在一个类中报告所有对象，要使用类层次方法。类层次方法是指处理对象的完整类而不是一个类的方法。例如，顾客确认订单是包含一个单独订单对象信息的输出消息。然而，为了提供一周内所有订单的总结报表，类层次方法要看到在订单类中所有的订单，并在一周以内为每个订单发送订单日期的输出信息。

设计报表、声明和返回文档

现代化的信息系统使得信息的使用变得更加广泛，随之而来的是各种类型的报表（书面的和电子的）迅速增加。今天的信息系统面临的难题之一是为了支持管理决策而要组织海量信息。输出设计的一个最困难的环节是决定要提供哪些信息以及如何表示这些信息，从而避免产生大量令人困惑的复杂数据。

报表类型

信息系统常用的四种输出报表类型是：

- **详细报表**——这些报表包含了业务交易中的详细信息。例如，所有过期账目的列表，这份列表的每一行包含某一特定账单的信息。公司职员可以使用这张列表来查看哪些账目已经过期，并想好收回那些账目的措施。
- **汇总报表**——这种报表通常用来对阶段性活动进行总结。这种报表的一个例子是对每天或每周的所有销售交易进行汇总，从而计算出销售总金额。管理人员通常使用这些报表来跟踪部门的业绩。
- **异常报表**——这种报表是在交易或操作结果超出其正常范围之后生成详细或总结信息。当业务正常进行时，不会生成异常列表。例如，工厂会列出一份不合格质量控制测试大于 0.2% 的零件报表。
- **执行报告**——组织中的高层管理人员通常会使用这种报表来评估组织的整体健康状况和运行情况。因此，这些报表包含从公司内部活动中得到的汇总信息。因此，它们也能显示在全行业范围内平均水平业绩的比较。使用这些报表，管理人员可以评估自己公司的竞争力和薄弱环节。

内部输出与外部输出的对比

打印输出可以分为内部输出和外部输出。**内部输出**是为了组织或单位内部使用而生成的，我们在前面讨论过的报表就属于这种类型。**外部输出**是为了组织外部人员的使用而生成的，包括声明、通知及其他文档。因为它们都是为外部人士生成的官方业务文档，所以需要使用高质量的图形和色彩。例如，银行每月的结算单、最新通知、订单确认信息、包装纸（提供给 RMO 顾客的）以及保险单等法律文件。有一些外部文档被称为**返回文档**，因为提供给用户的文档中包含了可撕下来的一部分，而这部分用于系统以后的输入，如包含将与支票

一起返回的付款存根的账单。所有这些打印的输出都必须仔细地设计。如今，高速的彩色激光打印机使得生成各种各类型的报表和其他输出形式成为可能。

图 7-17 给出了一个外部输出的详细报表例子。当顾客通过网络下订单时，系统可以把订单打印出来以便进一步确认。当然用户也可以通过浏览器的打印功能打印出网页，但这样做太浪费时间，因为这将打印网页上所有的图形和链接。处理图 7-11 中基于 Web 的显示，只向用户显示与订单有关的、格式良好的确认信息会显得更加友好。

Ridgeline Mountain Outfitters—Shopping Cart Order

Customer Name: Fred Weeting
Customer Number: 6747222

Order Number: 4673064
Today's Date: May 18, 2013

Shipping Address:
936 N Swivel Street
Hillville, Ohio 59222

Billing Address:
936 N Swivel Street
Hillville, Ohio 59222

Qty	Product ID	Description	Size	Color	Price	Extended Price
1	458238WL	Jordan Men's Jumpman Team J	12	White/ Light Blue	$110.00	$110.00
1	347827OP	Woolrich Men's Backpacker Shirt	XL	Oatmeal Plaid	$41.00	$41.00
2	875942SSH	Nike D.R.I. – Fit Shirt	M	Black	$30.00	$60.00
1	5858642OR	Puma Hiking Shorts	L	Tan	$15.00	$15.00
					Subtotal	$236.00
					Shipping	$6.00
					Tax	$11.25
					Total	$253.73

Shipping Information:
Shipping Method: Normal 7-10 day
Shipping Company: UPS
Tracking Number: To be sent via email
Email Address: Fred1953@aol.com

Payment Information:
American Express ☐ MasterCard ☐ VISA ☒ Discover ☐
Account Number
X X X X – X X X X – X X X X – 6 7 8 4
MO YR
Expiration Date 05 / 15

Thank you for your order. It is a pleasure to serve you.
Check back next week for new weekly specials!

图 7-17　RMO 购物车订单报表

图 7-18 是基于存货记录的一个内部输出的例子。这份报表包括明细和汇总部分，尽管这个图没有显示出汇总部分。控制变值是一个可将明细部分分成多组的数据项。在这个例子中，控制变值是产品号，即报表上的 ID 号。在输入记录过程中，任何时间遇到新的 ID 号，报告都将重新从新的控制值开始。明细部分列出了来自数据库的交易记录，汇总部分提供了总计信息和摘要信息。报表根据产品排序和显示。然而，每个产品都列出了库存项，以此来显示现有的库存数量。

外部输出可以包括复杂的多页文档。众所周知的一个例子是你收到的关于汽车保险结算的一套报表和声明。这个声明通常是一个多页文档，包括详细的汽车保险信息和费用、汇总页、返回的奖励支付卡和每辆汽车的保险卡。另一个例子是为每个雇员定制的多页信息的雇员福利报表。有时候，这些文档以带有高亮显示部分或徽标的彩印形式来输出。图 7-19 是雇员福利册生存保护页当中的一页，上面的文字都是一样的，只有数字会因人而异。

Ridgeline Mountain Outfitters — Products and Items

ID	Season	Category	Supplier	Unit Price	Special Price	Discontinued
RMO12587	Spr/Fall	Mens C	8201	$39.00	$34.95	No

Description Outdoor Nylon Jacket with Lining

Size	Color	Style	Units in Stock	Reorder Level	Units on Order
Small	Blue		691	150	
	Green		723	150	
	Red		569	150	
	Yellow		827	150	
Medium	Blue		722	150	
	Green		756	150	
	Red		698	150	
	Yellow		590	150	
Large	Blue		1289	150	
	Green		1455	150	
	Red		1329	150	
	Yellow		1370	150	
Xlarge	Blue		1498	150	
	Green		1248	150	
	Red		1266	150	
	Yellow		1322	150	

ID	Season	Category	Supplier	Unit Price	Special Price	Discontinued
RMO28497	All	Footwe	7993	$49.95	$44.89	No

Description Hiking Walkers with Patterned Tread Durable Uppers

Size	Color	Style	Units in Stock	Reorder Level	Units on Order
7	Brown		389	100	
	Tan		422	100	
8	Brown		597	100	
	Tan		521	100	
9	Brown		633	100	
	Tan		654	100	
10	Brown		836	100	
	Tan		954	100	
11	Brown		862	100	
	Tan		792	100	
12	Brown		754	100	
	Tan		788	100	
13	Brown		830	100	
	Tan		921	100	

图 7-18 RMO 存货报表

电子报表

组织机构使用各种类型的报表，每一种服务于不同的目的，并且每一种都有各自的优点和缺点。电子报表在组织和显示信息上具有灵活性。在一些实例中，屏幕输出模拟打印报表的格式，只不过以电子形式显示。然而，电子报表也可以有多种形式来显示信息。一些报表有明细和汇总部分，一些报表将数据与图像一起显示，其他一些报表是以黑体和高亮显示，还有一些可以动态改变组织结构及汇总部分，另外一些包含连接相关信息的热点链接。电子报表的一个有利之处是它的动态性，它可以在一个特定的环境中为满足特定需求而改变。事实上，许多系统提供了特殊的报表能力，可以使用户悠闲地设计自己的报表。例如，电子报表可以链接到进一步的信息。所谓的**下钻**技术能允许用户激活报表上的"热点链接"功能，它能告诉系统显示下一层报表，提供更详细的信息。例如，图 7-20 是月末销售汇总报表，这份报表按照产品的目录和季节提供了销售汇总情况。然而，如果用户单击任何季节的热键，都会弹出带有详细销售数据的报表。

Survivor Protection

In the event of your death while working for a participating employer, your designated beneficiaries could receive:

Lump Sum Benefits

$50,000	Basic Life Insurance
$230,000	Supplemental Life Insurance
$148,677	Thrift Plan
$31,686	Tax Sheltered Annuity (TSA) Plan
$255	Social Security for your eligible dependents
$460,618	Total*

You have not elected Universal Life Insurance. If you would like more information on this plan, please call 1-800-555-7772.

*Refer to page 7 for additional information on the amount of coverage needed to provide ongoing replacement income.

Accidental Death Benefits

If your death is due to an accident, your designated beneficiaries will receive the above benefits plus:

$100,000	24-Hour Accidental Death and Dismemberment Insurance
$100,000	Occupational Accidental Death and Dismemberment Insurance, if the accident is work related

Monthly Death Benefits

If you die before receiving the Master Retirement Plan benefits and you are vested and have a surviving spouse, your spouse may be eligible for a Qualified Pre-Retirement Survivor Annuity.

In addition, your family may be eligible for the following estimated monthly benefits from Social Security, not to exceed a maximum of $2,591 based on:

$1,110	for each child under age 18
$1,110	for a spouse with children under age 16; or
$1,058	for a spouse age 60 or older

图 7-19　雇员效益报表样本

这种热点链接的另一个不同之处是它可以让用户将一个报表中的信息与另一个报表中的信息相互关联。大多数经常浏览 Internet 网页的人对这种热点链接都非常熟悉。在电子报表中，热点链接能提供一些相关和扩展的主要信息。同样的功能在业务报表上也非常有用。例如，用来链接某个特定行业内主要公司的年度结算。

电子报表的另一个动态特性是可以从不同的角度来观察数据。例如，按照地区、销售经理、生产线、时间段来查看销售数据或者与上一季度的数据进行比较，这些可能是很有益的。如果需要，电子报表可以生成不同的视图，而不是打印出所有报表。有时候，冗长的或复杂的报表会包括很多热点链接，这些链接可以连接到报表内容的不同部分。一些报表生成程序提供了电子报表能力，它包括所有网页上提供的功能，包括框架、热点链接、图形甚至动画。

图形和多媒体显示

数据的图形显示是信息时代最大的优势。允许数据以图形和图表的方式表示的工具，使得打印的和电子格式的信息报表更加友好。由于信息报表使商务人士能够预测趋势和变化，所以它们被越来越多地用于战略决策的制定。另外，如今的系统通常维护的是海量数据，远远超出了人们可以想象的程度。使用这些数据的唯一有效方法是对它们进行总结并以图形表格的形式表示出来。图 7-21 举例说明了柱状图和饼状图，它们是表示汇总数据最常用的两种方法。

Monthly Sales Summary

Year 2013 Month January

Category	Season Code	Web Sales	Telephone Sales	Mail Sales	Total Sales
Footwear	All	$ 289,323	$ 1,347,878	$ 540,883	$ 2,178,084
Men's Clothing	Spring	$ 1,768,454	$ 2,879,243	$ 437,874	$ 4,691,484
	Summer	213,938	387,121	123,590	724,649
	Fall	142,823	129,873	112,234	384,930
	Winter	2,960,489	6,453,896	675,290	10,109,675
	All	939,729	4,897,235	349,234	7,086,196
Totals			747,968	$ 1,698,222	$ 23,391,023
Women's Clothing	Spring				985,610
	Summer				
	Fall				
	Winter				
	All				
Totals					

Monthly Sales Detail

Year 2013 Month January Category Men's Clothing Season Winter

Product ID	Product Description	Web Sales	Telephone Sales	Mail Sales	Total Sales
RMO12967	Winter Parka	$ 1,490,245	$ 3,226,948	$ 337,640	$ 5,054,833
RMO13788	Fur-Lined Gloves	149,022	322,695	33,765	505,482
RMO23788	Wool Sweater	596,097	1,290,775	135,058	2,021,930
RMO12980	Long Underwear	298,050	645,339	68,556	1,003,005
RMO32998	Fleece-Lined Jacket	447,075	1,258,079	100,271	1,805,425
Total		$ 2,980,489	$ 6,743,836	$ 675,290	$ 10,394,615

图 7-20 RMO 汇总报表及详细的细节报表

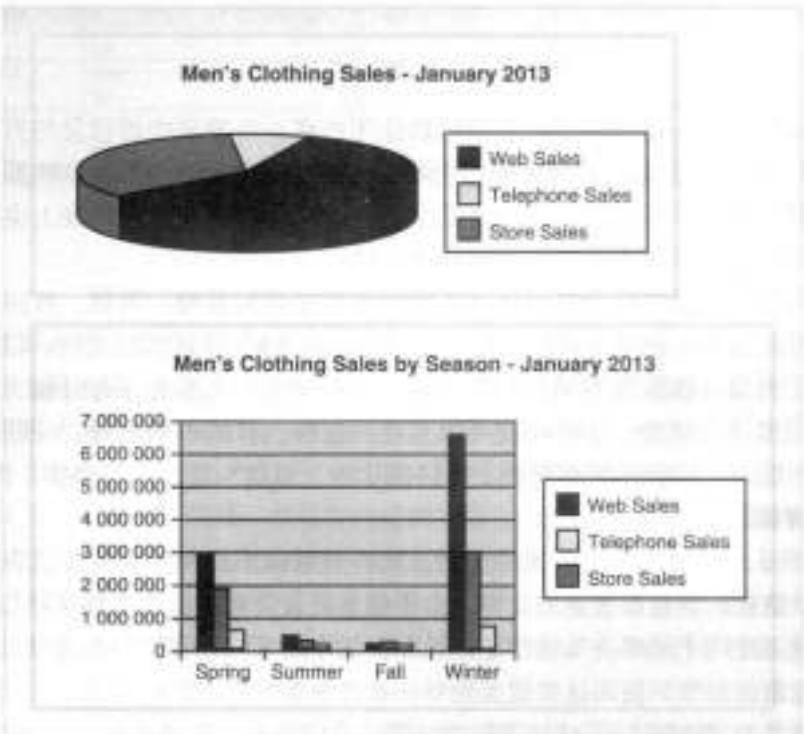


图 7-21 饼图和柱形图报表样本

近几年，随着多媒体工具性能的发展，多媒体输出也成为可能。如今，在屏幕上以图形和动画形式呈现信息并且突出部分伴有声音描述已经成为可能。将视觉和声音输出结合起来是表示信息的一种强有力手段（当然，视频游戏将包含视觉、声音和嗅觉输出的虚拟现实技术推向前沿）。

随着系统输出设计的进行，对各种表示方法进行评价将是有益的。在系统中，报表包可以被集成到系统中来提供一个全范围的报表定制方法。开发者应仔细分析每一个输出报表，确定输出的目标，选择最适合信息及其使用的报表格式。

本章小结

输入和输出可以分为系统界面和用户界面。从物理意义、感官意义和概念意义上来说，用户界面都是用户开始使用系统时所获得的全部内容。描述用户界面的方式有很多种，包含桌面隐喻、文档隐喻和对话隐喻。对话设计的初步工作是确定基于活动或用例的对话。故事板显示了顺序的屏幕概略图，它可以表达设计要求，以便与用户一起评审设计，或者利用 Visual Basic 等工具创建对话原型。面向对象的方法提供 UML 模型来编制对话设计的文档，包括顺序图、活动图和类图。

对话中使用的每个窗体和表格都需要设计，并且要遵循有关布局、选择输入控件、导航和帮助的指导原则。这些原则应用于标准窗体和以网络为基础的系统浏览器窗体的设计。网站的对话设计几乎等同于其他对话，不同的是，用户需要更多信息且要求有更好的灵活性。附加的 Web 设计指导原则用于设计计算机媒体、设计整个网站及设计用户网站。此外，因为网站反映了客户眼中的公司形象，所以图形设计者和市场专业人员应该参与设计过程。

设计系统输入时，开发者要确定输入设备和所有系统输入，并列出每个输入的数据内容。为了开发系统输入列表，设计者要使用顺序图和设计类图。设计系统输出过程的步骤与输入设计的步骤是一样的。

复习题

1. 为什么界面设计通常指的是对话设计？
2. 对用户来说组成用户界面的系统的三个方面是什么？
3. 用户界面的物理、感官和概念方面有哪些具体的例子？
4. 用于描述人机交互的三种隐喻是什么？
5. 屏幕上的桌面是描述人机交互的三种隐喻中的哪一种隐喻的例子？
6. 哪种类型的文档能允许用户点击连接之后跳到文档的另一个部分？
7. 列出并简单描述能应用到所有类型的用户显示和输入设备中的界面布局的四个指导原则和格式。
8. 哪种技术能显示一个对话框中的一系列草图？
9. 哪种 UML 图可以用于显示对话中的界面对象在参与者和问题域类之间是如何插入的？
10. 哪些输入控制可以用于从列表中选择条目？
11. 哪两种类型的输入控制是成对出现的？
12. 顾客在线购物时直接访问网站的流行模式方式是什么？
13. XML 指的是什么？解释 XML 与 HTML 的相同之处。同时也讨论一下两者的区别。
14. 你是如何通过使用 UML 和面向对象方法来确定系统界面的数据字段的？

15. 输出屏幕设计和输出报表设计要考虑的事项中有哪些不同？
16. 下钻的意义何在？试举例说明在报表设计中如何使用下钻方法。
17. 信息过载的危险何在？你能想到的避免信息过载的解决方案是什么？

问题和练习

1. 想一想你用过的所有软件，那些软件存在着易学和易用的冲突？
2. 访问一些网站并明确用于导航和输入的所有控件。这些控件都显面易见吗？讨论一下控件可视性和提示性的差别。
3. 设计人机界面的一个公共准则是改变机器以适应人而不是改变人来适应机器。在你的日常生活中是不是还存在着有待进一步改善的机器或系统？当前的 Windows PC 和 Apple Mac 在可用性上是不是已经尽善尽美？如果不是，哪些措施有助于进一步完善？万维网的可用性是不是就是这样？如果不是，哪些措施有助于进一步完善？我们是不是已看到可用性方面的突破，或者已经做出了许多重大进步？
4. 从 Google 上下载并安装 App Inventor。使用它去开发一个原型界面，如图 7-7 中实施的故事板那样。
5. 评价你所在大学的课程注册系统。列出用户与系统对话的基本步骤。从易学与易用观点来看，系统存在哪些问题？系统不灵活在什么地方？哪些方面是需要信息而不是可用性？处理的任务中是否转移了太多所要提供的信息？
6. 评价你所在大学图书馆的在线目录系统。写出能显示用户与系统之间交互的对话。再进行修改。创建一个故事板来显示你的设计的外观与感觉。
7. 查找一个用户能直接订购的网站。通过浏览一些产品描述并记录对话与网页的设计。你对其比较欣赏和不欣赏的地方是什么？评价这个基于可视性和提示性的网站。这个网站在页面更新数量和页面更新延迟之间取得最佳平衡了吗？如果你使用不同的计算设备、不同的网络或者是在同一天不同的时间连接系统，你会改变回答吗？

扩展资源

Randolph G. Bias and Deborah J. Mayhew, *Cost-Justifying Usability: An Update for the Internet Age* (2nd ed). Morgan Kaufmann, 2005.

Paul C. Brown, *Implementing SOA: Total Architecture in Practice*. Addison-Wesley, 2008.

Patrick Carey, *New Perspectives on Creating Web Pages with HTML, XHTML, and XML* (3rd ed.). Cengage Learning, 2010.

Donald Norman, *The Design of Everyday Things*. Basic Books, 2002.

Janice Redish, *Letting Go of the Words: Writing Web Content that Works*. Morgan Kaufmann, 2007.

Ben Shneiderman, Catherine Plaisant, Maxine Cohen, and Steven Jacobs, *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (5th ed.). Addison Wesley, 2009.

Joel Sklar, *Principles of Web Design* (5th ed.). Cengage Learning, 2012.

第四部分

项目和项目管理

第 8 章　系统开发方法
第 9 章　项目计划和项目管理

系统开发方法

学习目标

阅读本章后，你应该具备的能力：

- 比较一个预测性和一个自适应的系统开发生命周期之间的基础假设与使用方法。
- 描述信息系统支持的关键活动和任务。
- 阐述系统开发方法中包括哪些内容——系统开发生命周期以及模型、工具和技术。
- 描述用于软件设计和建模的两种完整方法：结构化方法和面向对象方法。
- 描述敏捷开发的核心特征。

开篇案例　Ajax Corporation、Consolidate Concepts 和 Pinnacle Manufacturing 的开发方法

Kim、Mary 和 Bob 是三个即将毕业的大学生，他们正在讨论最近来校园招聘 CIS 专业学生的各公司面试的情况。他们一致认为，虽然一开始他们觉得有点不知所措，但通过面试学到了很多东西。

Kim 说："刚开始我不能确信我是否知道他们在谈论什么。"在面试中，她在数据建模方面的知识给 Ajax Corporation 的面试人员留下了深刻的印象。第二轮面试时，她参观了 Ajax Corporation 本部的数据中心，同时，面试人员花了大量时间向她介绍公司的系统开发方法。

Kim 继续说："他们说要我忘记在学校所学的东西，这引起了我的注意。"

Ajax Corporation 已经从一个小咨询公司购买了一套完整的开发方法——IM One。大多数员工认为该方法很好。Ajax Corporation 的老员工认为 IM One 是独一无二的，他们为此感到骄傲。

Kim 又说："然后，他们就开始向我讲述 SDLC，迭代、业务事件、数据流图、实体－联系图以及诸如此类的事物。"她发现 IM One 方法中的许多关键概念都来自系统开发结构化方法中经常使用的模型和技术。

"我明白你的意思。"Mary 说。Mary 是一个非常有天分的程序员，几乎熟悉每一种新的可供利用的编程语言。"Consolidate Concepts 公司中 Booch、Rumbaugh 和 Jacobson 三名员工一直重复使用着 OMG、UML 和 UP 的技术开发系统。但随后我发现他们是在用面向对象的方法开发系统，知道我了解 Java 和 VB.NET 后他们很高兴。当我了解了他们使用的所有术语后，就不会有什么问题了。"

参加了 Pinnacle Manufacturing 面试的 Bob 有一个不同的故事要告诉我们。他说："一些人说分析与设计不再是大任务，但是我认为懂得这些将会节省我在校的很多时间。"

Pinnacle 拥有支持制造及存货控制的小型系统开发小组。Bob 还说："少量的文档及不完整的项目计划。然后他们给我看了一些放在办公桌上的书籍，看起来他们好像是读了大量有关分析与设计的内容。我还发现他们使用敏捷开发和敏捷建模技术，并且他们也关注他们

的小项目需要的最佳实施方式。这证明他们在建立原型时是通过观察风险及编写用户材料来组织不同工作的。我在老板的白板上看到一些类图及序列图的草图，这让我感到非常舒服。"

Kim、Mary 和 Bob 一致认为在这样的工作环境中有许多东西需要学习，但同时许多用来描述关键概念和技术的不同方式他们在学校已经学过了。他们一直重视学习 CIS 课程中的基础知识，并且懂得系统开发中所要用到的许多方法，这一点令他们觉得很高兴。

8.1 引言

Kim、Mary 和 Bob 的经历证明，开发信息系统的方法有很多，同时也非常复杂。项目经理依靠各种各样的辅助工具来帮助他们完成每一步处理过程。在本书中，你已经学习了整体的系统开发过程。在第 1 章，你学到了系统开发生命周期。那种特别的系统开发生命周期包括六个核心过程和多重迭代。本章会更加详细地讨论系统开发生命周期，包括系统开发生命周期的许多变体。此外，一个信息系统包括配置之后的大量支持工作，因此系统开发生命周期中的支持阶段也是需要讨论的。

开发信息系统的整个处理过程需要的不仅仅是系统开发生命周期。系统开发方法包括完成每个核心处理过程活动的更多具体指导，而这些指导是通过使用具体的模型、工具和技术来获取的。本章也会回顾两种主要的定义信息系统技术和用于业务系统的软件开发的方法：传统方法和面向对象方法。传统方法是指结构化软件开发方法，通常按阶段化和模块化进行描述，它使用结构化分析、结构化设计和结构化编程。面向对象方法是指面向对象软件开发，描述了作为一系列交互对象的软件。它使用的是对象类图、顺序图、状态图及面向对象编程这样的模型。最后，敏捷开发是作为一种理论被介绍的，它可以用于指导一个开发项目。它聚焦于的技术和方法能鼓励更多用户参与，而且在更灵活的项目中允许带有变化的需求。

8.2 系统开发生命周期

第 1 章说明了怎样通过建立一个信息系统并且使用分析与设计模型来解决业务问题。为使这一解决业务问题的工作富有成效，必须有组织且目标明确。分析员通过把这一工作组织成项目来实现目标。就像在第 1 章中定义的，项目是一个有始有终有计划的任务，它能得到被很好定义的结果或产品。信息系统开发这个术语是指一个有计划、能产生新信息系统的任务。一些系统开发项目很大，需要许多人进行数千小时的工作，并且可能会持续几年时间。在第 2 章介绍的 RMO 实例研究中，正在开发的系统是一种基于计算机的中等规模的信息系统，需要一个持续时间不超过一年的中等规模的项目。许多开发项目比较小，持续时间也就是一两个月。

要使系统开发项目取得成功，那么这个项目必须要被计划和组织。这个计划必须包括以合适顺序组织的一系列活动。不然的话，有些活动会被遗漏或者有些工作会重复多次。最终结果当然是产生一个高质量的信息系统，衡量目标就是它是可靠、强大而高效的。第 1 章介绍的系统开发生命周期（SDLC）是信息系统开发项目取得成功的基本概念。

系统开发生命周期提供了一种方式，使得我们可以将新系统开发作为先进的处理过程来思考，更像是一个活的实体。我们可以扩大这个观念，然后将这个信息系统看作本身有生命的系统来观察；事实上，我们经常会提到系统的生命周期。在一个信息系统的生命周期中，首先是构思，其次是开发项目部分中的设计、构建及部署，最后是形成成品并用于支持业

务。然而，就算是在其被广泛使用的过程中，系统仍然是动态的、活的实体，需要通过更小的项目进行更新、修改和维护。

一个系统生命中有很多个项目，首先是开发初始系统，之后是对初始系统进行升级。在本章中——事实上是在本书的大部分内容中——我们将把重点放在初始开发项目而不是支持项目。换句话说，我们关心的是开发系统及第一时间配置。

在如今多种多样的开发环境下，有很多开发系统的方法，而且这些方法是以不同的系统开发生命周期为基础的。尽管很难找到包含所有方法的单一、综合的分类系统，但我们可以使用一种有用的技术，那就是根据方法是否具有预测性和适应性来对系统开发生命周期进行分类（见图 8-1）。

图 8-1　不同项目选择不同的系统开发生命周期

系统开发生命周期的预测方法是一个可以预先计划、组织开发项目并可以根据计划对新的信息系统进行开发的方法。对构造系统来说，预测系统开发生命周期易懂、易定义。例如，一家公司希望将老的网络化客户－服务系统转变为新的以 Web 为基础的系统，同时也包括智能手机 App。在这类项目中，要求员工清楚地知道需求，而且不再需要添加新的程序。因此，这种项目通常需要仔细计划，系统可根据说明书进行构造。

当系统需求或用户需求没有被很好地理解时，就会使用**系统开发生命周期的适应方法**。这种情况下，项目不能被完整地计划。在初步开发工作之后还需确定系统的一些需求。开发人员仍可以制定解决方案，但要求必须灵活，而且在项目发生进展的时候可以进行调整。第 1 章中的贸易展览系统就是使用了这种方法。

事实上，任何一个项目或者说大部分项目都既包含预测方法又包含自适应方法。这就是图 8-1 显示的它们在预测方法和自适应方法之间如何过渡以取得平衡，而不是互斥。预测方法产生于 20 世纪 70 － 90 年代，更为传统。很多新的适应方法是随着面向对象技术和 Web 开发而发展起来的，创始于 20 世纪 90 年代后期并一直发展到 21 世纪。我们先看一些偏重于预测的方法，然后再看一些新的适应方法。

8.2.1　系统开发生命周期的传统预测方法

一个新信息系统的开发需要一系列不同的但相互关联的活动。在预测方法中，首先要有一组能确定问题且获得批准的活动来开发新系统，这被称为*项目启动*。第二组活动被称为*项目计划*，包括计划、组织和计划项目。这些活动计划了项目的整个结构。第三组——*分析*——集中于发现和理解问题或需求的细节。用在这里的目的是理解系统在支持业务过程中必须做的事情。第四组——*设计*——集中于配置和建立新系统的组件。这些活动使用先前定义的需求来为新系统开发程序结构和算法。第五组——*实施*——包括编程和测试系统。第六组——*部署*——包括安装和操作系统。

这六组活动——项目启动、项目计划、分析、设计、实施和部署——有时是指系统开发项目的**阶段**，同时它们也会提供管理项目的框架。另一个阶段称为**支持阶段**，包括在部署好之后对系统的升级和维护。虽然支持阶段是整个系统开发生命周期的一部分，但总是被排除在初始开发项目之外。图 8-2 所示为传统预测性的系统开发生命周期的六个阶段加支持阶段。

图 8-2　传统的信息系统开发步骤

最具预测性的系统开发生命周期方法（例如，在预测 / 适应天平最左端的）被称为**瀑布模型**，是指项目从一个阶段到另一个阶段连续进行。如图 8-3 所示，这个模型指出这几个阶段是有序实施和完成的。首先，开发一个详细的计划，然后全面确定需求，接着设计系统直到最后的算法，之后编程、测试及安装。一旦经该方法进入到下个阶段，则不可再返回到上个阶段。事实上，在开发项目的每一步中，瀑布模型都需要严格的计划和最终决策。可以想到，一个纯粹的瀑布模型是不能很好地完成工作的。作为开发人员，不可能在完成一个阶段时不出现任何错误，或者遗留下一些重要的部分以待后期添加。然而，即使我们不使用纯瀑布模型，它也可以为我们的开发奠定基础。无论开发系统与否，我们仍需要包括启动、计划、分析、设计、实施和部署活动。

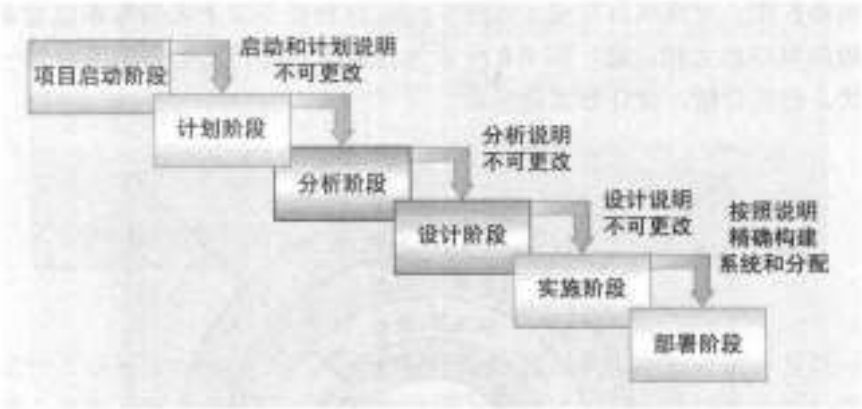


图 8-3　瀑布法系统开发生命周期

在预测 / 适应天平的右端是改进的瀑布模型。这些仍然是需要预测的，也就是说，还需要开发一个全面的计划。但是我们发现项目的每个阶段都是互相重叠、影响和依赖的。在开始设计之前必须做一些分析，但在设计中我们会发现在需求方面需要更多的细节，或者一些需求是原先我们没有遇见过的。图 8-4 展示了这些活动是如何重叠的。

另外一个造成阶段重叠的原因是效率。在团队成员分析需求的同时，他们可能会考虑并设计各种表格或报表。为了帮助理解用户的需求，他们可能要设计最终系统的一部分。但是他们做早期设计时，经常要删减部分内容，而其他的一直保留到最后的系统中。此外，计算机系统的许多内容是相互依赖的，这就要求分析员既要做分析又要做一些设计。

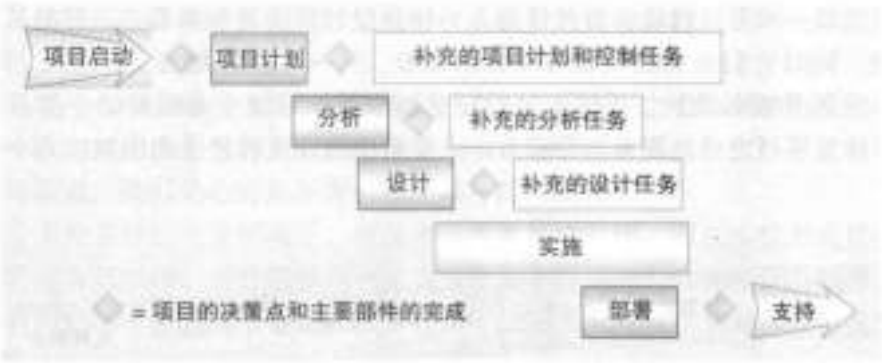


图 8-4　迭代法系统开发生命周期

8.2.2　系统开发生命周期的新的自适应方法

在自适应方法中，包括计划和模型在内的项目活动的开发方法可以根据项目的进展进行调整。有很多种方式能描述自适应系统开发生命周期。在所有方式中都包括第 1 章介绍的迭代。迭代可用来创建一系列能处理一小部分应用的小型项目，而不是带着一些重叠进行有序的分析、设计和实施阶段。这些小部分中的一个是在一个简单的迭代中被分析、设计、建立和测试的，然后以这个成果为基础，下个迭代继续分析、设计、建立和测试下个小部分。使用迭代，项目能适应各种变化。同时，系统的一部分也能很早被用在用户评估和反馈中，这样做能帮助确保这个应用是符合用户需求的。

远离天平右端的是**螺旋模型**。它更偏向于自适应性方法，可视为以每个迭代成果为基础来自适应项目的最早概念之一。这种模型用一个螺旋来描述生命周期，从中心开始，一遍又一遍地反复向外扩张，直至项目完成（见图 8-5）。这种模型看上去与瀑布模型非常不同，这使得项目管理的风格也大相径庭。图 8-6 所示为自适应系统开发生命周期的另一种表现，显示了几个迭代，包括分析、设计和实施活动。

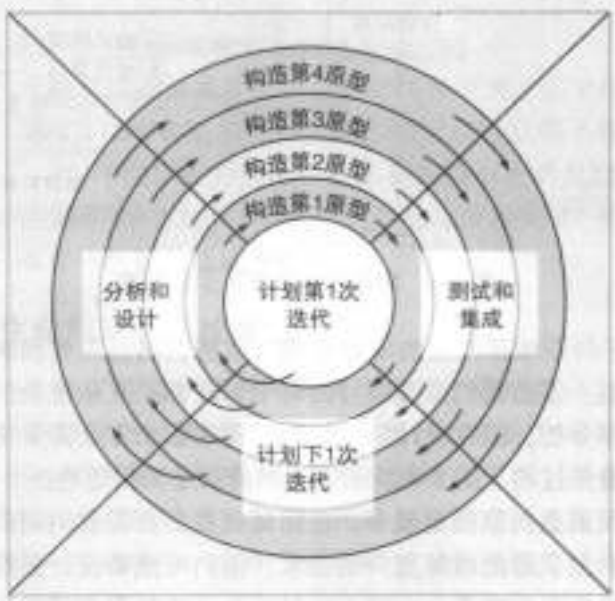


图 8-5　螺旋形生命周期模型

图8-6说明了开发活动中的多重迭代。这两个特点也可以表现为一个表的行和列。表格的行是开发活动，表格的列是迭代。你一开始看到这个概念是在图1-4中，在这里被替代为图8-7。第1章中定义的核心过程会贯穿在整本书中，图8-6所示的就是开发活动的扩展图。表中的列当然就代表项目的多重迭代。事实上，如果我们将这个自适应、迭代生命周期中的核心过程与图8-3的瀑布生命周期中的阶段进行比较，我们会发现两者之间相近的一致性。这两个生命周期首要的区别是瀑布法企图通过一个简单的过程做完所有的计划、所有的分析、所有的设计等。迭代法是自适应的方法，因为有着每个迭代的分析、设计和实施，才能创造出方法来适应项目中不断变化的需求。本书中出现的自适应方法是简单化的，一个更正式化的迭代方法称为统一过程（UP）。

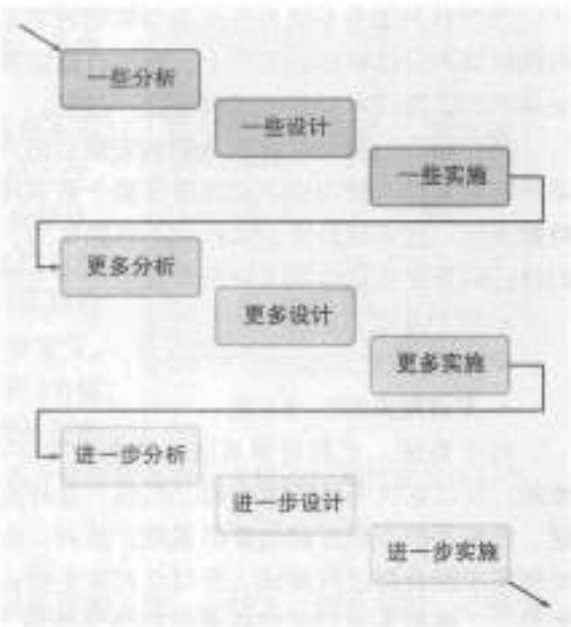


图8-6 迭代交叉生命周期阶段

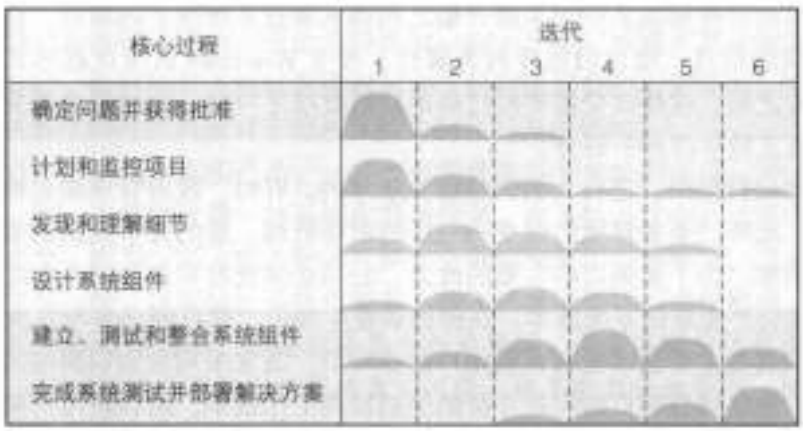


图8-7 对一个典型的系统进行迭代的六个核心过程

一个与迭代系统开发生命周期相关的概念称为**增量开发**。增量开发总是以一个迭代生命周期为基础的。其基本概念是系统在很小的增量上被建立的。一个增量是用一个简单的迭代开发的，或者可能包括2～3个迭代。当每个增量都完成时，整个系统就和增量整合在一起了。事实上，系统是以一种有机方式“增长”的。这个方法的优势是能使用户更快地获得系统的一部分，所以这个业务才能尽快开始盈利。

另一个基于迭代方法的相关概念源自于**行走骷髅**的想法。行走骷髅，按照字面上的意思是它能提供一个新系统从前到后完整的实施过程的功能“骨架”。行走骷髅是在项目早期的几次迭代中被开发出来的。后期的迭代会用更多的功能和能力来充实这个骨架。很显然这个方法在项目开发中能尽早使用户获取工作软件。这两种方法在项目开发过程中由于有大量用户测试和反馈，因此会给项目团队带来额外优势——这也是迭代项目如何自适应的另一个例子。

8.3　支持阶段

预测性的瀑布系统开发生命周期明确地包括一个支持阶段，但是自适应迭代系统开发生命周期就不是这样的。实际上，新的自适应系统开发生命周期认为支持阶段就是一个完整独立的项目，方法本身就已经包含这一过程。

支持阶段的目标是在系统初始安装后的几年里保持系统的有效运行。支持阶段起始于新系统安装并投入使用后，持续贯穿整个新系统的有效使用周期。大多数业务系统期望系统要持续多年。在支持阶段，为扩大系统的能力，可能会执行一些升级或加强，这将需要用到他们自己的开发项目。在支持阶段发生三个主要的活动：

- 维护系统。
- 加强系统。
- 支持用户。

每个系统，尤其是新系统，常常会有一些不能正确运行的组件。软件开发是复杂而困难的，所以它决不能随意出错，当然，良好组织并精心执行的系统开发的目标是提供一个稳定、完整并能提供正确结果的系统。然而，由于软件的复杂性，以及不可能对每一种可能的处理需求组合都进行测试，所以总有发生错误的时候。此外，业务需求和用户需求随时会发生变化。维护系统的关键任务包括修复错误（也称为修复补丁）和为处理需求稍微做调整两部分。通常，会安排一个系统支持小组负责维护系统。

多数刚工作的程序员在其职业生涯开始之初都从事过系统维护的项目。任务通常会包括改变报告中提供的信息、增加数据库列表属性、改变 Windows 或浏览器形式的设计。在这些工作分配下去之前，这些改变需要经过申请且被批准才可以，因此每个改变的请求批准过程往往也是系统支持阶段的一部分。

在系统正常运行期间，进行大的调整也是正常的。有时，政府管理需要维护新的数据或提供新的信息。此外，业务环境的改变——新的市场机遇、新的竞争或新的系统构造也需要对系统做大的调整。为了实施这些主要的改进，公司必须批准和启动一个升级开发项目。一个升级项目经常会产生新的系统版本。在你的职业生涯中，你可能会参与几次升级项目。

支持阶段的其他主要活动是对系统用户提供帮助。通常采用由知识渊博的技术人员组成**帮助台**的方法，快速回答用户的问题并帮助他们提高工作效率。培训新用户和维护目前的文档是这一活动的重要工作。作为一个新的系统分析员，你可能会去指导培训或充当帮助台的职员来获取用户问题和需求。多数刚工作的程序分析员在开始他们的职业生涯前，往往会花费一些时间在帮助台上工作。

8.4　方法、模型、工具和技术

除了系统开发生命周期之外，系统开发者使用各种辅助工具来帮助他们完成完整的活动和任务。这些辅助工具包括方法、模型、工具和技术。下面各节将对其进行逐一讨论。

8.4.1　方法

为完成系统开发生命周期的每一步，**系统开发方法**可提供一些指导方法。例如，在一种方法中，会使用像图这种具体的模型来描述和说明需求。与这些模型相关的是项目团队用于完成工作的技术。技术的一个例子是你在第 2 章学到的关于组织对于用户采访的指导原则。最后，每个项目团队会使用一系列基于计算机的工具来建立模型、记录信息和书写编码。这

些工具被看作全部方法中的一部分而在项目中使用。图 8-8 说明了技术、模型和工具相互支持，并提供了一个全面且一体化的方法。一些方法是由公司内部的系统专业人士根据自身经验而开发的。其他的一些方法则是从咨询公司或供应商处购买的。

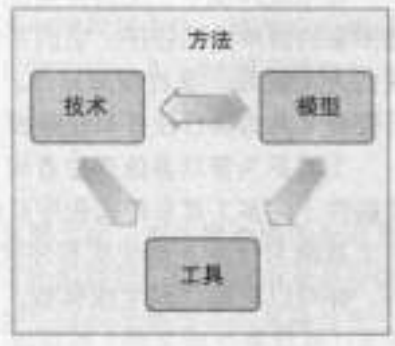


图 8-8 方法的构成

一些方法（自己开发的或者购买的）拥有大量的书写文档，这些文档定义了开发人员在项目中需要做的工作，包括文档应该写成什么样子、管理报告应该包括哪些部分。其他的一些方法就不会这么正式了，通常在一份文档中包含了应该完成的所有工作的大致描述。有时，公司采用的方法不仅是非正式的，而且是临时的且大多数没有被定义，但是这样的自由选择正变得越来越少。大多数 IT 部门的管理偏向于采用灵活的方法，这样它才能适用于不同类型的项目和系统。组织使用的方法取决于系统开发项目的方法应该具有什么样的预测性或自适应性。

8.4.2 模型

任何时候人们都需要记录或同现实世界中的某些事物交流一些信息，因此创建一个模型是非常有用的。就像第 2 章讨论的那样，模型是现实世界中某些重要方面的表示。有时，我们使用术语抽象来表示模型，因为从现实世界中抽象出的某些方面对我们特别重要。例如，考虑一个飞机模型。在谈到飞机的空气动力特性时，有一个能三维展示其全面形状的小模型是非常有用的。有时，能够展示飞机机翼横截面细节的制图是我们所需要的。在另外的情况下，也许有关飞机数学特性的列表对于理解飞机如何飞行是必要的。

一些模型在外形上类似于真实产品，一些模型是重要细节的绘图表示，另一些模型则是抽象的数学符号。每一种模型强调一种不同类型的信息。在飞机设计中，飞机工程师使用大量的不同类型的模型。成为一名飞机工程师需要学会创建和使用各种模型，这一点对于信息系统开发人员来说也一样，尽管信息系统模型并不像飞机模型那样标准或精确。现在，信息系统开发人员也正不断取得进步，但是这还是个新兴领域，许多高级分析员都是自学成才的。更重要的是，信息系统并不像飞机那样真实可感，你不能真正地看到、抓到或感觉到它。因此，信息系统模型显得更加无形。

系统开发中使用的模型包括输入、输出、过程、数据、对象、对象之间的相互作用、位置、网络和设备以及其他事物的表示。大多数模型是图形模型，包括使用公认的符号和惯例画一张表示图。这些通常被称为图和表，以及最近你在本书学到的 UML 图的例子。本书大部分描述的是如何理解和创建描述信息系统的各种模型。

另一种重要的模型是项目计划模型，如甘特图和净现值，这两种图都会在第 9 章说明。这些模型是对系统开发项目自身的表示，其中突出显示了项目任务及其他的考虑。与项目管理有关的另一个模型是一张显示了分派给项目所有者的图表。图 8-9 列出了系统开发中使用的一些模型。

系统组件的一些模型
- 流程图
- 数据流图（DFD）
- 实体－关系图（ERD）
- 结构图
- 用例图
- 类图
- 顺序图

用于管理系统开发过程的一些模型
- 甘特图
- 组织层次图
- 财务分析模型－净现值，回收期

图 8-9 一些用在系统开发中的方法

8.4.3 工具

在系统环境中，工具是帮助生成项目中所需模型或其他组件的软件支持。工具也许是创建图表的简单绘图程序。它们可包括存储项目信息的应用程序，如数据定义、用例描述及其他的人工制品。项目管理软件，如微软的 Project 是用于生成模型的工具的另一个例子。项目管理工具为项目任务和任务相关性创建了一个模型。

工具是为帮助系统开发者而专门设计的。程序员应该熟悉**集成开发环境**（IDE），这个环境提供了许多工具来帮助程序员编程，例如灵巧的编辑器、上下文相关帮助和调试工具。有些工具能为开发人员生成程序代码。有些工具则可以通过反向工程执行文件来获取程序代码，并可以根据代码生成模型，即使在开发人员将文档丢失（或没有生成文档）的情况下也能推断出程序的用途。系统分析员可用的 Visual 建模工具能够帮助他们创建和核实重要系统模型。这些工具被用来画出如类图和活动图这样的图表。其他的 Visual 建模工具能帮助设计数据库，甚至是生成程序代码。图 8-10 列出了在系统开发过程中使用的工具类型。

项目管理应用程序
制图 / 图形应用程序
字处理器 / 文本编辑器
Visual 建模工具
集成开发环境（IDE）
数据库管理应用程序
反向工程工具
代码生成工具

图 8-10 一些用在系统开发中的工具

8.4.4 技术

在第 2 章中你学到了几种收集信息的技术。在第 3 ~ 5 章中你学到了几种定义功能需求的技术。在第 7 章中你学到了用户界面设计技术。在系统开发中，**技术**是指导原则的组合，能帮助分析员完成系统开发活动或任务。它通常为创建模型提供逐步指导，或者为从系统用户处收集信息提供更一般的建议。常见的实例包括数据建模技术、软件测试技术、用户面谈技术和关系数据库设计技术。

有时一项技术可能适用于整个生命周期，能帮助你创建一些模型和其他的文档资料。现代结构化分析技术（我们会在以后讨论）就是一个例子。图 8-11 列出了在系统开发中经常使用的一些技术。

战略计划技术
项目管理技术
用户面谈技术
数据建模技术
关系型数据库设计技术
结构化编程技术
软件测试技术
处理过程建模技术
域建模技术
用例建模技术
面向对象编程技术
结构化设计技术
用户界面设计技术

图 8-11 一些用在系统开发中的技术

那么，怎么把方法、模型、工具和技术组合在一起呢？方法包括一组用来完成系统开发生命周期每一阶段活动的技术。这些活动包括完成各种模型及其他文档资料和交付资料。与其他行业一样，系统开发人员使用软件工具来帮助他们完成这些活动。

8.5 软件构造与建模的两种方法

系统开发可以使用多种不同的方法，而这种多样性会使系统开发者感到困惑不解。有时，每个公司看起来都有自己的开发方法。事实上，甚至是同一家公司的不同开发小组或个人也都有可能有自己的系统开发方法。

然而，系统开发中有很多通用的概念。事实上，所有开发小组都会使用系统开发生命周期的一些变体，这些变体同样包括项目启动、项目计划、分析、设计、实施、部署和支持阶

段。此外，几乎每个开发小组都使用模型、工具和技术，这些模型、工具和技术组成了一个完整的系统开发方法。

所有的系统开发者都应该熟悉两种非常普通的软件构造与建模的方法，因为它们构成了几乎所有方法的基础，这两种方法是：结构化方法和面向对象方法。本节回顾了这两种方法的主要特点，并给出了这两种方法的发展历史。

8.5.1　结构化方法

先前在本章中，我们讨论了系统开发生命周期的传统预测方法。那些概念集中在开发项目本身的阶段和活动中。这部分内容将集中于包括分析与设计的模型，以及被用来开发软件的编程结构。这个软件构造方法称为结构化系统开发。有时，这两个概念——系统开发生命周期的传统预测方法与结构化软件构造方法——可能引起混淆，因为它们都作为传统方法。我们也会在本书中更准确地使用术语，但是你也应该意识到在普通行业中不会这么精确。

结构化系统开发

结构化分析、结构化设计和结构化编程是组成**结构化方法**的三种技术。有时把这三种技术一起称为结构化分析与设计技术（SADT）。20 世纪 60 年代开发的结构化编程技术是人们首次试图通过指导来改善计算机编程的质量。在第一次编程中，你肯定要学习有关结构化编程的基本原理。结构化设计技术发展于 20 世纪 70 年代，这种技术使得把分散的过程组合为更加复杂的信息系统成为可能。结构化分析技术发展于 20 世纪 80 年代早期，使得开发人员在设计程序之前就对计算机系统的需求了解得非常清楚。

结构化编程。高质量的程序不仅在程序每次运行时都能产生正确的输出结果，而且使得其他程序员以后可以非常容易地阅读和修改程序，因为程序总是需要不断修改。**结构化编程**具有一个开始和一个结束，并且程序执行过程中的每一步都是由三种程序结构组成的：

- 顺序程序语句。
- 选择是由这一组语句来执行程序，还是由另一组语句来执行程序。
- 循环语句。

图 8-12 所示为结构化编程的三种结构。

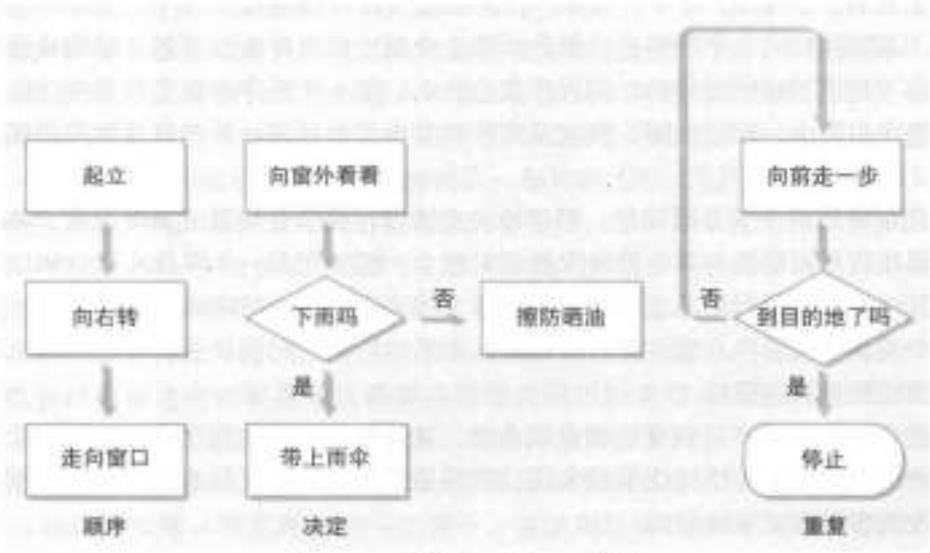


图 8-12　结构化编程的三种结构

与结构化编程有关的一个概念是自上而下程序设计。**自上而下程序设计**把复杂的程序分解成程序模块的层次图（如图 8-13 所示）。在需要的时候，层次图顶层的模块通过“调用”底层模块来控制程序执行。有时，这些模块是同一程序的一个部分。例如，在 COBOL 中，一个主程序段通过使用“Perform”关键字来调用另一段程序。在 Visual BASIC 中，事件过程中的语句可以调用一个通用过程。程序员使用结构化编程的规则（一个开始、一个结束和顺序、选择与循环结构）来编写每个程序模块（程序段或过程）。

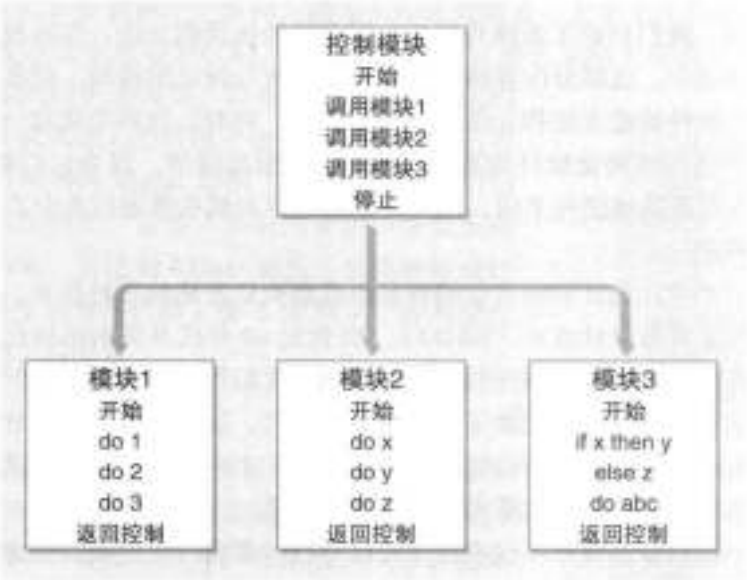


图 8-13　自上而下或模块化程序设计

有时，我们开发多个单独的程序，然后把这些程序组合为“系统”一起运行。这些程序的每一个都使用自上而下的程序设计和结构化编程规则，但这些程序本身可以被组织成一个层次图，就像自上而下的程序设计一样。一个程序可以调用其他的程序。当层次图包括多个程序时，这样一种安排有时称为模块化编程。

结构化设计。20 世纪 70 年代以来，信息系统变得日益复杂，每一个系统都包含许多不同的功能。系统执行的每个功能也许都是由数十个独立的程序所组成的。**结构化设计**技术是用来为确定下列事物提供指导的，即程序集是什么、每一个程序应该实现哪些功能，以及如何把这些程序组织成一张层次图。模块及模块的安排可以使用一种叫做**结构图**的模型来图形化地表示（见图 8-14）。

结构化设计的两个主要原则是：程序模块应该设计成耦合松散或高度内聚。耦合松散意味着每个模块应尽可能地与其他模块保持相对独立，这使得每一个模块在设计和以后修改时不会干扰其他模块的运行。高度内聚意味着每个模块实现一个清晰的任务。如此一来，很容易理解每个模块实现哪些功能，并且可以确保如果以后需要对模块进行修改，那么不会由于意外的原因而影响其他模块。

结构化设计定义了不同程度的耦合和内聚，并且提供一种在程序真正编写之前对设计质量进行评价的方法。对于结构化编程来说，质量是根据以后出现新的需求时，以程序设计被理解和修改的容易程度来确定的。

结构化设计方法假设设计者知道系统需要做什么：主系统的功能有哪些，需求数据有哪

些，以及需要的输出结果是什么。设计系统显然不仅仅是设计程序模块的组织结构。因此，结构化设计只是帮助设计者完成整个系统设计生命周期阶段的一部分，认识到这一点非常重要。

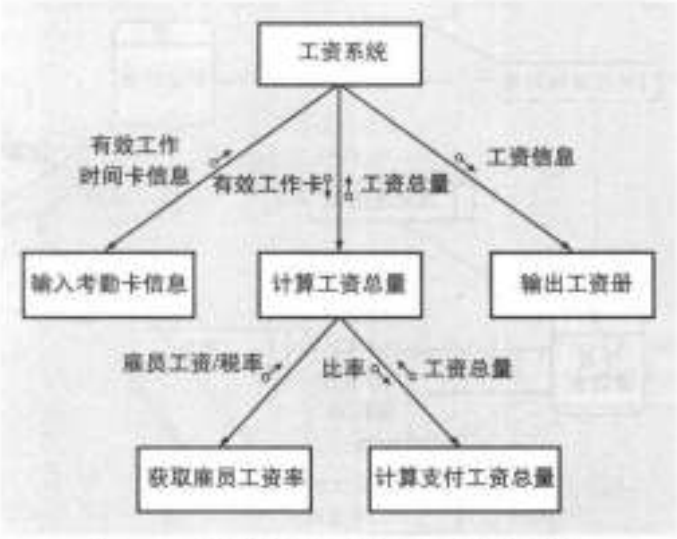


图 8-14　使用结构化设计技术生成的结构图

到了 20 世纪 80 年代，文件和数据库设计技术也被用于结构化设计。更新的结构化设计技术假设在系统中使用数据库管理系统，并设计程序模块来和数据交互。此外，由于越来越多的非技术人员开始涉及信息系统，因此用户界面的设计技术也相应地得到了发展。例如，交互系统中的菜单决定了调用层次图中的哪一个程序。因此，用户界面设计的一个关键部分是与结构化设计一起完成的。

结构化分析。因为结构化设计技术要求系统设计人员熟知系统应该实现哪些功能，因此定义系统需求的技术获得了发展。系统需求详细定义了系统必需的功能，但并没有规定实现这些功能的具体技术。通过推迟确定实现系统功能的具体技术，开发人员能够把他们的注意力集中在需要系统做什么而不是如何做方面。如果事先不能充分而又清楚地计算出这些需求，那么设计人员不可能知道需要设计什么样的系统。

结构化分析技术帮助开发人员定义系统需要做什么（处理需求），系统需要存储和使用哪些数据（数据需求），需要什么样的输入和输出，以及如何把这些功能结合在一起完成任务。在结构化分析中使用的表示系统需求的主要图形模型是**数据流图**（DFD），它表示了系统的输入、处理、存储和输出，以及它们如何在一起协调工作（见图 8-15）。

通过确认引起系统以某种方式做出响应的所有事件，结构化分析的最新变体定义了系统的处理需求。例如，在一个订单录入系统中，如果顾客订购了一件商品，那么订单录入系统必须处理一张新的订单（一种主要的系统活动）。每一个事件引起不同的系统活动。系统分析员获得这些活动中的每一种，并且生成相应的数据流图来显示包括输入、输出在内的处理细节。

所需数据的模型也可以根据系统需要存储信息的事物类型来创建（数据实体）。例如，为了处理一张新的订单，系统需要知道顾客、所需商品及订单的细节。在第 4 章你学到的这种模型被称为实体－联系图（ERD）。实体－联系图的数据实体对应于数据流图中的数据存储。图 8-16 所示为实体－联系图中的一个例子。在使用结构化分析、结构化设计及结构化编程方法进行系统开发时的顺序如图 8-17 所示。

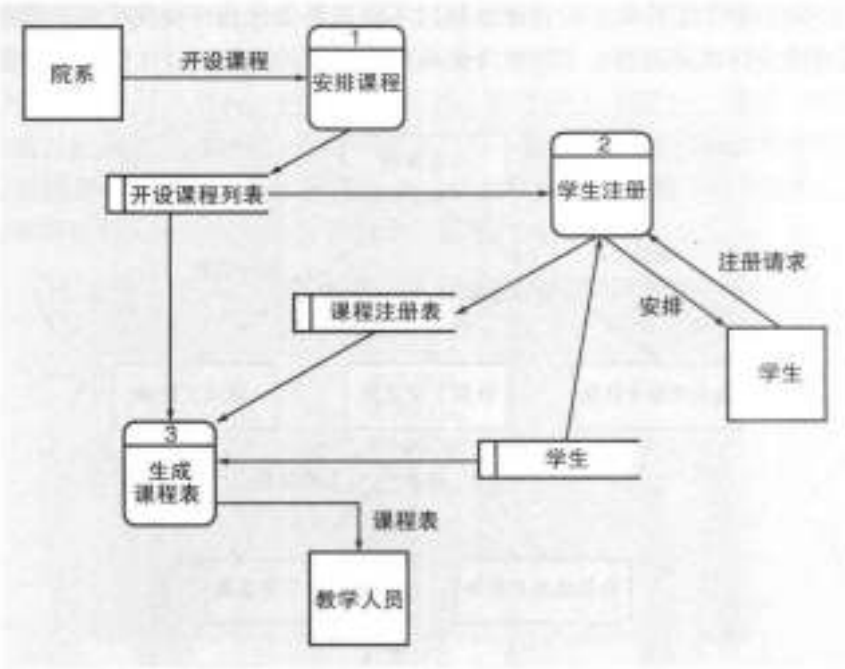


图 8-15 使用结构化分析技术生成的数据流图（DFD）

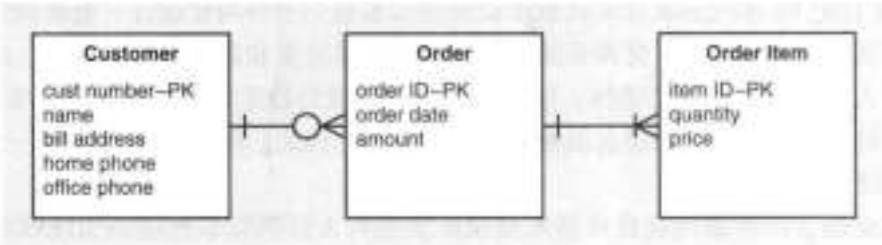


图 8-16 用结构化分析技术生成的实体－联系图（ERD）

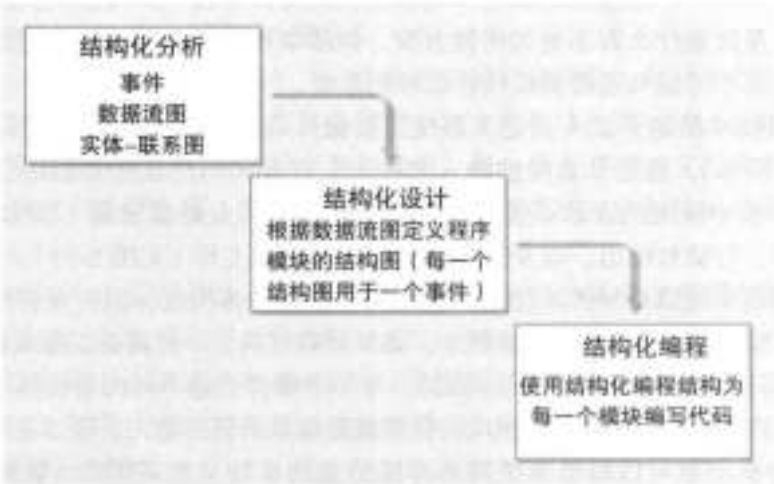


图 8-17 结构化分析如何指导结构化设计和结构化编程

8.5.2 面向对象方法

一种完全不同的信息系统开发方法——**面向对象方法**——把信息系统看作一起工作来完成某

项任务的相互作用的对象集合（如图 8-18 所示）。在面向对象方法中，既没有过程和程序，也没有数据实体和文件，系统只是由对象组成。**对象**是一个在计算机系统中能对消息做出响应的事物。这种对计算机系统完全不同的看法要求使用一种不同的方法来进行系统分析、系统设计和编程。

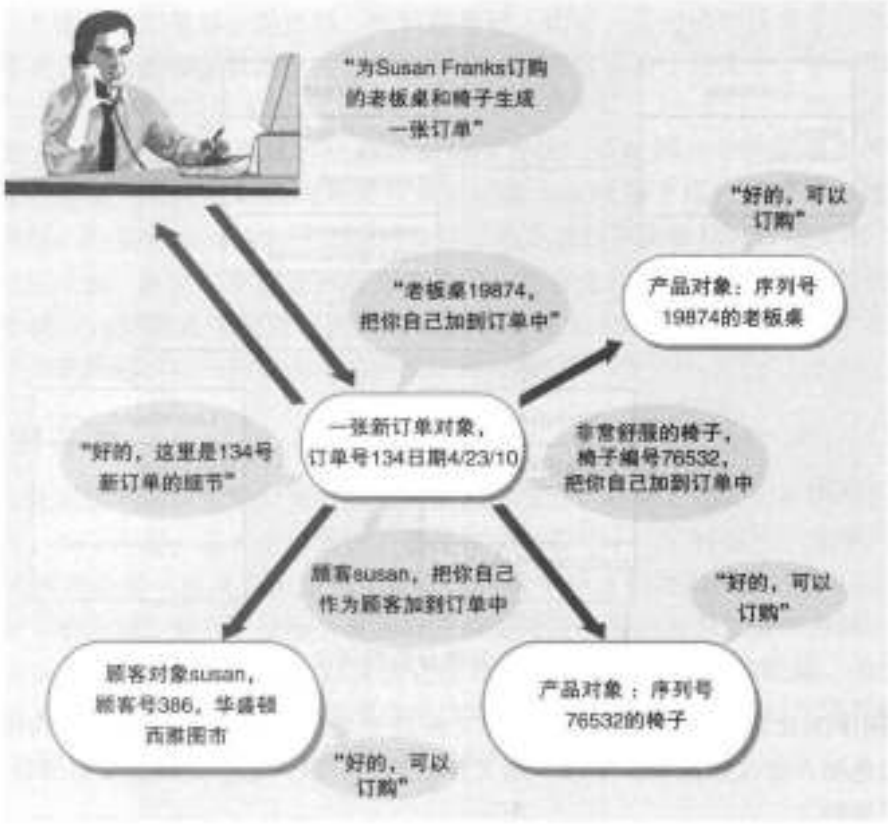


图 8-18　面向对象的系统开发方法（从图中的用户顺时针查看）

面向对象方法开始于 20 世纪 60 年代挪威 Simula 编程语言的开发。Simula 语言用于创建包括像轮船、浮标和峡湾中的潮汐等"对象"的计算机模拟。编写模拟轮船运动的过程化程序是非常困难的。但一种新的编程方法简化了这个问题。在 20 世纪 70 年代，人们开发了 SmallTalk 语言用于解决创建包含诸如下拉式菜单、按钮、复选框和对话框等"对象"的图形用户界面（GUI）的问题。最近的面向对象语言包括 C++、Java 和 C#。这些语言集中于编写系统中所需对象的类型定义，结果一个系统的所有部分都可以看成是对象，而不仅仅是图形用户界面。

考虑到面向对象方法把信息系统看作相互作用的对象集合，因此**面向对象分析**（OOA）意味着定义系统中所有的对象类型，并显示对象之间是如何通过相互作用来完成任务的。**面向对象设计**（OOD）意味着定义系统中的人和设备进行通信所必需的其他对象类型，同时它还显示了对象之间是如何通过交互来完成任务的，它也修改了每种类型对象的定义，这样能用一种具体的语言或环境来实现它。**面向对象编程**（OOP）意味着用一种编程语言书写语句来定义每一类对象的行为。

对象是事物的一种类型。它可以是一个顾客或者一个雇员，也可以是一个按钮或菜单。确定对象类型意味着对事物进行分类。一些诸如顾客这样的事物，是既存在于系统之外又存在于系统之内的。存在于系统之外的就是真实顾客，存在于系统之内的就是顾客在计算机的内部表

示。一个分类或“类”表示相似对象的集合，因此，面向对象的开发方法使用UML类图（第4章中介绍过的）来表示系统中所有对象的类型（如图8-19所示）。对于每个类来说，也许还有更具体的子类。例如，储蓄账户和支票账户是账户的两个具体类型（账户类的两个子类）。类似地，下拉式菜单和弹出式菜单是菜单的两个特殊类型。子菜单体现或继承在其之上的类的特性。

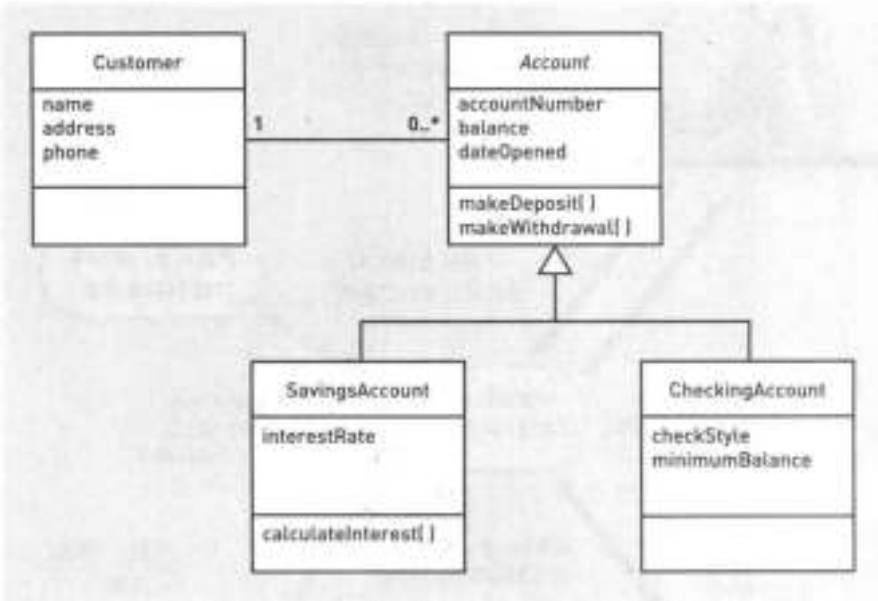


图8-19　在面向对象分析期间生成的类图

UML顺序图显示了在实施任务时对象之间的交互或合作。图8-20所示为用例实现图，包括代表角色的人物线条图和5个在一起工作的其他对象，它们是通过发送消息来完成取消订单这个用例的。

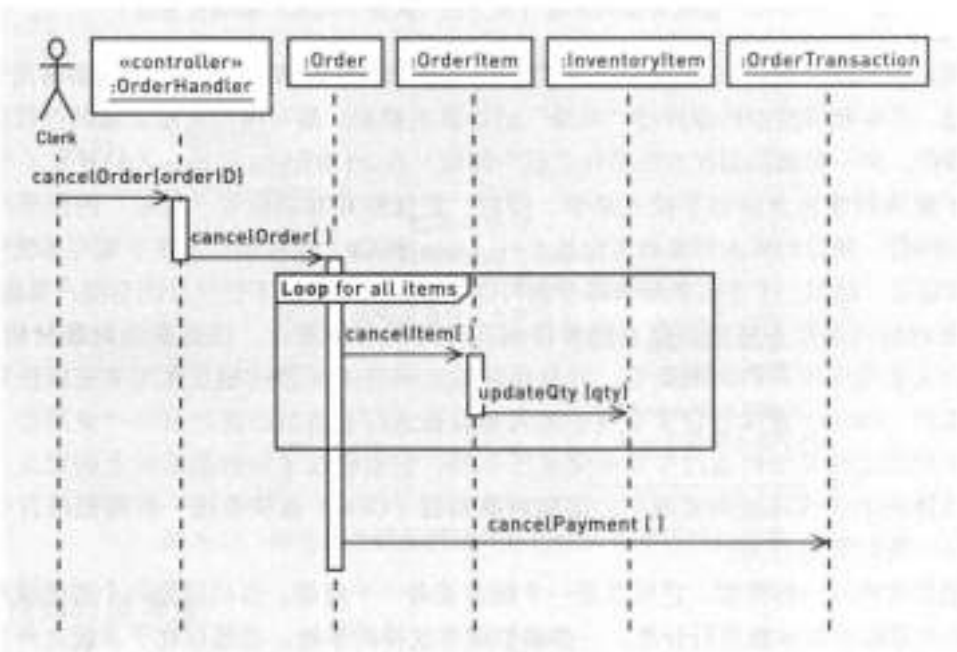


图8-20　显示一个用例目标迭代的UML顺序图

面向对象方法有几个主要的优点，其中包括自然性和复用性。对人而言，面向对象方法是自然的或者直观的，因为我们倾向于按照可感知的对象来思考世界。在面向过程编程语言中随处可见的复杂过程是非常不自然的。同时，由于面向对象方法包括对象的类，并且组织中的许多系统使用同样的对象，因此无论何时需要，这些类都可以一次次重复使用。例如，几乎所有的系统都使用菜单、对话框、窗口和按钮，而同一公司中的许多系统也采用可重复使用的顾客类、产品类和发货清单类。因此，我们不再需要为了创建一个新对象而"重复发明车轮"。

如今开发的许多系统将传统方法和面向对象方法结合使用。一些集成开发环境（IDE）也在同一工具中使用了传统和面向对象技术。例如，OOP用于用户界面，而过程化编程则用于其他部分。许多Web应用程序是通过使用结构化编程和模块化设计建立的。例如，PHP是支持所有技术的。在同一个信息系统开发部门，甚至也有许多系统项目在分析与设计阶段只用传统方法，而其他系统则只用面向对象方法。每个人都应该知道这两种方法的基本概念，但是在大学课程中或许只强调了其中的一种方法。

8.6 敏捷开发

快速变化的市场迫使业务对新的机遇做出新的反应。有时，这些机会出现在实施其他业务的计划中。为了生存，业务必须敏捷，即使在项目进行了一半时也能够快速改变方向。**敏捷开发**是一种理论和一系列在未知且快速变化环境中开发信息系统的纲要，它能被任意一种系统开发方法使用。通常，敏捷开发能补充系统开发生命周期的自适应方法，并且能支持它。但重点在于采用自适应方法，以及使它在开发活动和任务时更加敏捷。关于敏捷开发，敏捷建模是关于怎样建模的理论，即哪些地方要求正式、详细，哪些地方可以概要、简略。在本书中你将学习怎样使建立的所有模型都能和敏捷开发一起使用。

8.6.1 敏捷开发的理论与价值

"敏捷软件开发宣言"确定了四个基本价值，它们代表了敏捷开发的核心理论：

- 快速响应凌驾于原有计划的价值。
- 个体和交流凌驾于程序和工具的价值。
- 执行软件凌驾于综合文档的价值。
- 顾客合作凌驾于合同协商的价值。

系统开发中的人员，不管是团队成员、用户还是其他利益相关者，对一个确定的敏捷开发项目都需要接受这些优先权。采用敏捷方法并不一定是容易的。经理和执行人员通常很难接受这种不严格的观点，他们反而想要在更大程度上控制开发团队，并且让他们交出更详细的计划和时间表。然而，敏捷理论采用的是相反的方法，它在项目的时间安排上更加灵活，并且让项目团队以项目的进度来计划和执行他们的工作。

在敏捷运动中，一些业界领导者创立了**混序**这个术语来描述敏捷项目。混序来源于两个单词：混乱和有序。列表中的前两个价值——快速响应凌驾于原有计划，个体和交流凌驾于程序和工具——是产生混乱的原因，但是软件开发本来就有未知和不可预测的因素，所以当然会有一些混乱。敏捷理论认为这样的不可预测性，可以用不断增加的灵活性来处理，并且通过信任项目团队来开发项目问题的解决方案。当不可预测的需求上升时，过分依赖于计划和预先定义的处理过程会使问题恶化。开发者需要接受这些混乱，但是也需要使用其他有益

于将有序和结构添加到项目中的敏捷建模与开发技术。第 9 章会涵盖更多敏捷项目管理技术的内容。

敏捷开发的另一个重要方面是顾客必须参与到项目团队中。他们不仅是和开发团队坐下为一些开发任务来讨论开发规范，然后做他们自己的事情。他们要成为技术团队的一部分。因为软件开发贯穿于整个项目，顾客要一直参与到确定需求与测试组件中。

从历史的角度来看，特别是带有预测性的项目，许多系统开发计划会企图努力维持固定价格。这对于内部组织和外部开发团队来说都是正确的。然而，敏捷开发的方法是这样一种系统开发项目，包含更多合作共赢的成果。因此，合同也应该采取一种完全不同的方式。固定的价格和固定的交付没有任何意义。针对敏捷项目的合同为顾客提供了其他类型的选择。活动的时间安排方法、系统组件的分配，以及项目提早结束的方法使得客户可以保持对项目的控制，但是这样做使用的是与固定的投标合同不同的方法。

对于敏捷开发来说，模型与建模是很关键的，因此我们接下来解释敏捷建模。在建模的原则与实践方面阐述了许多核心价值。

8.6.2　敏捷建模原则

本书中的大部分内容是在讲述创建模型的技术。你的第一印象可能是敏捷方法意味着很少的建模或是不用建模。**敏捷建模**（AM）不是更少的建模，而是为实现正确的目标在恰当的水平建立恰当的模型。AM 并不是命令人们去建立哪个模型或者指明这些模型的正式程度。相反，它是帮助开发者沿着他们的模型轨迹进行开发——使用模型而不是把建模作为一种目标。AM 的基本原理说明了开发者开发软件时应该具有的态度。图 8-21 概括了敏捷建模的原则。下面我们将讨论这些原则。

敏捷建模原则

- 将软件开发作为你的首要目标
- 将下一步的成果作为你的第二个目标
- 最小化建模活动——少而精
- 拥护变化和增量改变
- 目的模型
- 建立多种模型
- 建立高质量的模型并快速得到反馈
- 重点放在内容上而不是表达上
- 公开交流的方式，相互学习
- 熟悉模型并知道如何使用模型
- 适应特定的项目需求

图 8-21　敏捷建模原则

将软件开发作为你的首要目标

软件开发项目的首要目标是开发出高质量的软件。工作进展的主要衡量方法是软件能够工作，而不是系统需求或说明书的中间模型。建模只是一种达到目的的方法，而不是目标本身。任何活动，如果不能证明能直接对软件产品的最终目标有贡献，则应该受到质疑和避免。

将下一步的成果作为你的第二个目标

只把目光放在软件能工作上可能会被自己击败，因此开发者必须考虑两个重要的对象。首先，需求模型对开发设计模型是有必要的。因此，不要认为不能用于写代码的模型就是没有必要的。有时，在编写最终代码之前的一些中间步骤是必需的。其次，尽管高质量的软件是首要目标，但代码的长期使用也很重要。因此，一些模型对系统支持维护和增强也是有必要的。当然，代码是最好的文档，但是一些结构设计很难从代码中看出来。认真对待其他附属品对高质量软件的长期使用是有必要的。

最小化建模活动——少而精

只创建那些必要的模型。通过这些工作就足够了。这个原则不是进行草率工作和不充分分析的借口。创建的模型应该清晰、正确和完整。但是不要创建不必要的模型，同时要保证

每个模型尽可能简单。通常，最简单的方案就是最好的解决方案。复杂的方法往往难以理解和维护。当然，简单并不是不完整的理由。

拥护变化和增量改变

因为敏捷建模的潜在理论是开发者必须灵活并对变化做出快速反应，所以一个好的敏捷开发者要自愿接受甚至拥护变化。把变化看作正常的而不是异常。观察变化并且把变化集成为一个模型。接受变化的最佳方法是增量式开发。做很小的一步并且把问题分解成几个小问题。增量地改变模型，然后证明其合法性并确保其正确。不要试图完成一个大版本的所有事情。

目的模型

我们前面说过建模的两个理由是：理解你建的是什么，以及系统解决方案各重要部分之间的通信。确保你建的模型支持这些理由。有时，开发者通过以下几点声明来证明其建模的正确性：（1）开发方法决定了模型的开发；（2）有人想要模型，即使这些人不知道它为什么重要；（3）一个模型可以替代面对面的讨论。为你开发的每个模型确定一个理由和一个观众。然后，开发的模型要足够详细来满足理由和观众的要求。顺便说一句，这个观众可能就是你自己。

建立多种模型

UML 等多种建模方法可以表达问题的不同方面。为了成功——理解问题或交流解决方案——你要对问题域或要求的解决方案的不同方面建模。不要去开发所有模型，确保模型最小化，但是要开发足够多的模型来确保表达所有的问题。

建立高质量的模型并快速得到反馈

没有人喜欢草率的工作。草率的工作来自于错误的思想和错误的解释。模型中避免错误的一种方法是快速得到反馈。反馈来源于用户，也来源于技术团队成员。其他人也可能有很好的观察力和不同的方式来看问题与鉴别解决方案。

重点放在内容上而不是表达上

有时，项目团队可以获得一个复杂的可视化建模工具。这些可能是很有用的，但是有时它们会分散开发者的精力，因为开发者会花费很多时间在制作漂亮的流图上。使用工具是明智的。同时需要建立一些模型用于交流或签合同。有时，用工具建立模型是富有成效的，因为可以预计它发生变化的频率较高，所以使用工具建立模型通常比用手来画更有效率。在其他情况下，手绘图可能就足够了。有时，开发者在会议室的白板上建立模型，同时也会为处理好的细节进行拍照以保存记录。

公开交流的方式，相互学习

所有自适应方法都强调团队工作。不要把你的模型保护起来。其他团队成员会有很好的建议。没有人需要掌握问题或模型的每个方面。

熟悉模型并知道如何使用模型

成为一个敏捷开发者并不意味着你没有技术。你要有更多的技术去知道模型的强项或弱项，包括什么时候和怎么样使用模型。专家级模型开发者应采用前面提到的简易、高质量和多种模型开发的原则。

适应特定的项目需求

因为每个项目都存在于唯一的环境，所以项目各不相同，包括不同的用户、利益相关者和团队成员，要求不同的开发环境和配置平台。让模型和建模技术适应业务和项目的需求。

有时，模型简单而富有信息。对于其他的项目，可能要求更加正式而复杂的模型。敏捷建模者要有能力适应每个项目。

本章小结

系统开发项目是围绕着系统开发生命周期（SDLC）而组织的。一些系统开发生命周期是基于项目的预测方法，其他系统开发生命周期是基于自适应的方法。系统开发生命周期的预测方法包括按顺序完成的阶段或有些重叠部分。传统的系统开发生命周期阶段包括项目启动、项目计划、分析、设计、实施、配置和支持。当需求或技术不确定并且很难提前计划项目的每件事时，就使用自适应系统开发生命周期。自适应系统开发生命周期使用多重迭代，允许应用程序一小部分的分析、设计和实施被完成及评估。本书中用到的系统开发生命周期是自适应系统开发生命周期的一个例子，并且这六个核心过程与系统开发生命周期的传统预测方法中的阶段都符合。

所有的开发项目都使用系统开发生命周期来管理项目，但是开发系统的方法还有更多。模型、技术和工具构成了系统开发方法学，而系统开发方法学为完成系统开发生命周期的每个活动提供指导。大多数的系统开发方法基于两种软件构造与建模方法之一：传统方法和面向对象方法。传统方法使用的模型和技术有用例、数据流图、实体－联系图、结构化图表、结构化分析技术、结构化设计技术和结构化编程技术。面向对象方法是将合作完成任务的交互对象的集合看作软件。像用例、类图、顺序图、包图、状态机图、面向对象分析、面向对象设计和面向对象编程这样的模型和技术都会使用到。

敏捷开发是系统开发中的主导趋势，它能帮助系统开发项目对变化做出响应。变化凌驾于原有计划、个体凌驾于程序和工具、执行软件凌驾于合同协商这些价值都是理论。敏捷建模描述了使项目保持敏捷的原则。

复习题

1. 什么是项目？
2. 信息系统开发项目规模的范围是什么？
3. 什么是系统开发生命周期？
4. 一个项目应用系统开发生命周期的预测方法有什么特点？
5. 一个项目应用系统开发生命周期的自适应方法有什么特点？
6. SDLC 的预测方法有哪七个阶段？
7. 支持阶段的目标是什么？
8. 阐述系统开发生命周期的瀑布模型在项目中怎样控制发生的变化。
9. 阐述 SDLC 预测方法中有重叠阶段的优势。
10. SDLC 的自适应方法中包括哪些组织性的概念？
11. 在 SDLC 的自适应方法中首先考虑的是什么？进行简述。
12. 在 SDLC 的自适应方法中，阐述每个迭代期间发生了什么。
13. 本书中使用的自适应方法的 SDLC 的含义是什么？
14. 本书中使用的 SDLC 有哪些核心过程？哪些 SDLC 的传统预测方法中的阶段与每个处理过程相对应？
15. 通过几个迭代完成和配置应用程序的一部分，然后再通过另外几个迭代完成和配置应用

程序的其他部分，包含这两个过程的迭代方法是什么？
16. 为什么 SDLC 的自适应方法中不包括支持阶段？
17. 支持阶段的三个活动是什么？
18. 被用来回答用户问题和帮助他们提高效率的流行支持技术是什么？
19. 什么是系统开发方法学？
20. 包含在一个方法中的模型有哪些？
21. 包含在一个方法中的技术有哪些？
22. 包含在一个方法中的工具有哪些？
23. 软件构造与建模的两种方法是什么？
24. 传统方法的基本特征是什么？
25. 面向对象方法的基本特征是什么？
26. 三种主要的结构化技术是什么？
27. 通过结构化方法创建的图有哪三种？
28. 主要的面向对象技术是什么？
29. 什么是敏捷开发？
30. 敏捷开发者中反映的价值是哪四个？
31. 什么是敏捷建模（AM）？
32. 敏捷建模的 11 个原则是什么？

问题和练习

1. 写一篇一页以内的文章，对比 SDLC 的传统预测方法中分析、设计和实施阶段基本目标的不同之处。
2. 描述一个有三个子系统的系统项目，并讨论该项目如何使用三次迭代。
3. 事实上，既然迭代过程几乎应用于所有的开发项目，为什么像瀑布一样顺序地介绍分析、设计阶段及活动还是有意义的？
4. 列出一些设计师设计的用来显示他们所设计房屋的不同方面的模型。解释一下为什么要使用多个模型。
5. 汽车设计师使用哪些模型来表示一辆汽车的不同方面？
6. 画出你家里的房间布局，并写下对于你的房间布局的描述。这些都是你房间的布局模型吗？哪个更准确、更详细、更容易使不熟悉你房间的人了解？
7. 描述一项用来帮助你完成“准时上课”活动的技术。与这项技术一起使用的是哪些工具？
8. 描述一项用来确保分配的工作准时完成的技术。与这项技术一起使用的是哪些工具？
9. 你使用的其他一些帮助你完成生活中活动的技术是什么？
10. 至少有两种系统开发方法、两种软件构造与建模方法以及一系列的技术和模型。讨论以下方法多样性的原因：这个领域非常新；技术变化快；不同组织有不同的需求；存在许多不同类型的系统；开发系统的人在背景知识上存在巨大差异。
11. 收集一些在校园里招募信息系统毕业生的公司信息。试着找到关于这些公司开发系统的方法的任何信息。他们描述 SDLC 了吗？他们提到集成化环境或者 Visual 建模工具了吗？访问该公司的网站查看更多的信息。
12. 访问一些主要的信息系统咨询公司的网站。试着找到他们用于开发系统的方法。他们描

述了 SDLC 吗？提到任何工具、模型或技术了吗？

扩展资源

Craig Larman, *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley, 2004.

Scott W. Ambler, *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. Wiley Publishing, 2002.

D. E. Avison and G. Fitzgerald, *Information Systems Development: Methodologies, Techniques, and Tools* (3rd ed.). McGraw-Hill, 2003.

Tom DeMarco, *Structured Analysis and System Specification*. Prentice Hall, 1978.

Ivar Jacobson, Grady Booch, and James Rumbaugh, *The Unified Software Development Process*. Addison-Wesley, 1999.

Steve McConnell, *Rapid Development*. Microsoft Press, 1996.

Meilir Page-Jones, *The Practical Guide to Structured Systems Design* (2nd ed.). Prentice Hall, 1988.

John Satzinger, Robert Jackson, and Stephen Burd, *Object-Oriented Analysis and Design with the Unified Process*. Course Technology, 2005.

项目计划和项目管理

学习目标

阅读本章后，你应该具备的能力：

- 描述能导致软件开发项目成功或失败的因素。
- 描述项目经理的职责。
- 描述项目管理知识体系（PMBOK）中的知识领域。
- 描述项目管理知识领域中的敏捷开发。
- 解释要获取项目批准的活动（核心过程 1）。
- 解释要计划与监控项目的活动（核心过程 2）。

开篇案例　蓝天共有基金：新的开发方法

蓝天共有基金的财务副总裁 Jim Williams 首先发言了，他对公司的信息技术指导 Gary Johnson 说："关于这个新的开发方法，有一部分是我觉得满意的，但是还有一部分让我有点担忧。"

"通过几次迭代来优化系统的这个想法我觉得很有意义。用户们总是很难了解他们真正需要新系统做什么以及哪个是最合适公司的。因此，如果他们能在早期就接触到系统的话，他们就能开始接受测试并且尝试发现这个系统是否是最符合需求的。"

"让我想一下，我是否已经理解了整个过程，你的开发团队与我的投资顾问会确定几个系统需要支持的核心过程，然后你的团队再进行设计并建立系统来支持那些核心过程。你要在六周内完成这个小型项目。然后，通过其他几个小型项目你要继续添加更多功能直到系统变得完整，并且功能完善。这样可以吗？"

Jim 对于这个新的系统开发方法变得越来越热情了。

Gary 说："是的，那就是基本的想法。你们的用户需要理解这个系统开始的几个版本可能是不完美的，也不是完全稳定的。但是这些早期的版本会要他们处理一些事情以进行试验。我们也需要在他们接受测试后获得反馈，因此这个系统才会在不断试验中变得完善。"

Jim 又说："我也意识到了这一点。用户不喜欢一开始就考虑他们需要系统做的每件事。他们喜欢尝试新东西。就像我之前说的，我喜欢这个方法。然而，我不喜欢的部分是这个方法对于你来说很难给我一个确切的时间表及项目成本。这让我很担忧。过去，我们会使用两个主要工具来监控一个项目的进度。你认为现在我们就完全不用时间表了吗？你想要开放性的预算吗？"

Gary 回答道："结果不会像一开始听起来那样糟糕。这个开发方法是一个'自适应'方法，我的意思是因为这个系统是不断成长的，所以项目的结尾会更具开放性。项目经理仍然能创建时间表并且评估项目的成本，但是她不会因为几次迭代就能确定和锁定所有需要的功

能。由于系统的范围在最初几次迭代后将会继续改进，所以会存在‘范围蔓延’这个风险。这也是自适应方法中最大的风险之一。我们应该频繁地与项目经理交流，以此来确保范围是在控制之内的，并且这个项目不会脱离控制。”

Jim 说：“好的。你已经说服我去尝试这个新方法了。但是，让我们先把这个项目作为试用项目，然后再观察它的效果。如果成功了，那么我们会考虑在其他项目上使用这个迭代方法。”Jim 和 Gary 都同意将这次试用作为开始的最好方式。Gary 随后就会见了项目经理并且启动了这个项目。

9.1 引言

第 8 章介绍了系统开发生命周期以及组织软件开发活动的各种选择。到现在为止，你可能要问自己以下几个问题：

- 所有这些活动是怎么协调的？
- 我怎样才能知道最先做的任务是哪个？
- 对于不同的团队和团队成员要怎样安排工作？
- 我怎样才能知道首先应该开发系统的哪个部分？

项目计划与项目管理的目的是给所有这些活动（有时看上去不相关）制定一些规则。就像你在本章会学到的，任何给定项目的成功主要依靠技术与那些项目管理者的能力。你会学到不仅仅是项目经理的项目管理技术，而是所有项目团队成员对项目做出了贡献并由此获得了成功。

本章首先会讨论项目管理的需求以及与此有关的原则。其余部分讨论了和系统开发过程中前两个核心过程有关系的详细活动，这些都是首要的项目管理过程。本章的目的是教会你怎么计划、组织和指导系统开发项目。

9.2 项目管理原则

许多人可能已经用 HTML 建立了一个网页，或者为自己或朋友编写了一个计算机程序。在那些情况下，这只是你的工作，你不会过多地担心怎么组织你的工作或怎么管理项目。然而，一旦两个或更多的开发者在一起工作，那么这个工作必须被分配与组织，从而每个开发者都有特定的工作。不管这个项目使用的是预测方法还是自适应方法，这样做都是正确的。前一章中已介绍，所选择的方法勾画出了一套复杂的活动和任务，必须认真管理。失败的组织方式通常会浪费时间和精力并导致混乱，甚至还可能导致项目的失败。

即使每个项目团队会指派一个人作为项目经理，他拥有团队功能的首要职责，但是其他成员还是要对项目管理做出自己的贡献。RMO 的 CSMS 系统的项目经理是 Barbara Halifax，但是还有一个资深系统分析员帮助她实施每一步。项目开始以后，所有团队成员都被包含在了管理项目中。

我们在之前章节中曾介绍过，项目是一个有始有终的计划任务，它能产生一个既定的结果，但是通常被时间表和资源所约束。信息系统的开发要符合它的定义。此外，有很多人和任务要组织和协调，而这通常是一个非常复杂的项目。不论它的目标是什么，每个项目都是唯一的。生产的是不同的产品，不同的活动需要不同的时间表，使用的也是不同的资源。这种唯一性使得信息系统项目很难控制。

9.2.1　项目管理的需求

研究表明大多数 IT 项目开发成功与否取决于以下三个标准：准时完成、在预算内完成以及有效地符合原始问题定义表达的需求。自 1994 年以来，知名的 Standish Group 发表了一份年度 CHAOS 报告，这份报告里提供了上一年度 IT 开发项目的成果。Standish Group 将项目按以下三种方式分类：（1）成功的项目，这样的项目在符合用户功能需求的同时，还准时且在预算内完成了；（2）受到挑战的项目，这样的项目存在推迟完成、超过预算和范围减少的状况；（3）失败的项目，这样的项目是被终止的或系统的成果从来没被使用过的。每年的数量是有差异的，最近几年有细微的提升。2009 年的结果是：32% 是成功的，44% 是受到挑战的，24% 是失败的（见图 9-1）。大量的金钱花费在项目上，但是这个项目却不符合他们的目标。

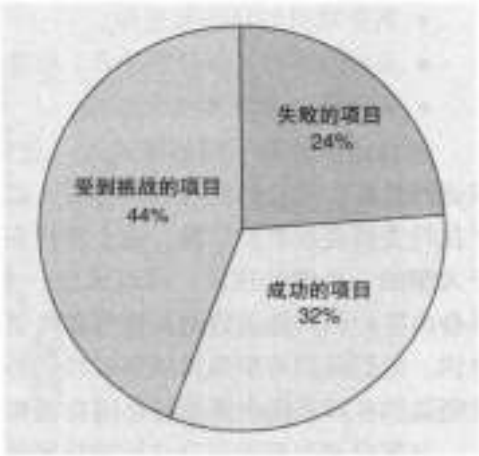


图 9-1　Standish Group 报告的项目完成结果

Standish Group 的报告不仅仅显示了 IT 项目的成功率或失败率，它同时也确定了每个项目失败或成功的理由。以下是一些项目失败的理由：

- 没有定义项目管理实践。
- 不善的 IT 管理与过程。
- 不充分的执行项目支持。
- 没有经验的项目经理。
- 不清晰的需求和项目目标。
- 不足的用户参与。

项目失败的首要原因是缺乏高层管理者的参与和管理技巧，这点是非常重要的。其他主要的原因是缺乏用户群体的参与。换句话说，项目不会因为缺乏编程技术或热情的开发者而趋于失败。

为了能使一个 IT 项目取得成功，强有力的 IT 管理与业务方针是必要的。在所有成功的项目中其他主要的元素是健全的项目管理程序及经验丰富且有能力的项目经理。事实上，好的项目经理总是会确保他们已经从企业高层和用户群体参与新系统的需求中接收到了清晰的指令。

9.2.2　项目经理的角色

项目管理是组织和指导其他人按照先前确定的进度和预算实现计划结果。在项目的初始阶段，首先要制定计划，这个计划详细说明了会发生的活动，然后要有可交付的成果以及所需的资源。因此，项目管理也可以定义为用来计划项目然后检测和控制项目的过程。

对于 IT 领域的人来说，现有的职业之一就是成为一个项目经理。由于较短的时间框架、分散式的项目团队（包括国外的和跨文化的团队）、快速变化的科技以及更复杂的需求使得项目变得更加复杂，因此高素质的项目经理非常受欢迎并且待遇很好。许多大学也添加了项目管理课程来应对行业的需求。对于有能力胜任项目经理的人来说，他们对这门课程具有强烈的需求。就像你的职业生涯一样，你应该开发自己的管理技能。你甚至可以在项目管理协会（PMI）中变得活跃，而这个协会对项目经理来说是最知名的职业组织。

整体来看，项目经理能有效地处理内部事物（管理成员和资源）和外部事物（处理人际关系）。从内部来看，项目经理是作为项目团队和所有活动的指导者和控制者。项目经理通

过建立团队结构来完成工作。下面列出了几项内部职责：

- 制定项目进度表。
- 招募和培训团队成员。
- 协调团队和团队成员的工作。
- 评价项目风险。
- 监测和控制项目成果和里程碑。

从外部来看，项目经理是项目的主要联系人。他必须把团队推广到外面去交流团队成员的需求。主要的外部职责包括：

- 报告项目的状态和进展。
- 直接与客户（项目赞助者）和其他利益相关者一起工作。
- 确定资源需求和获取资源。

项目经理要和不同团体的人一起工作。首先就是**客户**（例如顾客），他们会为新系统开发进行投资。项目的批准和资金投入都来自于**客户**。对于内部开发人员来说，客户可以是一个执行委员会或者副总裁。这个客户在批准和监督项目的同时还要审查资金的使用状况。对于大型的、关键的项目，可以成立一个**监督委员会**（有时也称为筹划指导委员会）。这个委员会由客户和其他关键的高级管理人员组成，这些高级管理人员具有该组织战略方向的远见卓识，他们强烈希望项目能够成功。另外一方面，**用户**是实际使用新系统的那些人。用户通常能提供在新系统中需要的详细功能和操作信息。

与客户和监督委员会之间的交流是项目经理外部职责的重要部分。同样，和组长、组员和分包商一起工作是项目经理内部职责的重要部分。项目经理必须确保所有内部和外部的交流都是正常流动的。图 9-2 所示为包含在项目开发中的各个组织的人群。

9.2.3　项目管理和仪式

另外一种对项目管理有重要影响的维度是正式程度，有时称为**仪式**，这对于一个给定的项目来说是必要的。仪式是对生成文档的数量、说明书的可追溯性及项目中决定过程的仪式性的衡量。小型项目通常会进行很低端的仪式。会议可能发生在走廊或饮水机附近。写下来的文档、正式的说明书及详细的模型都被保持在最小量。开发者和用户通常每天都会在一起工作以定义需求和开发系统。大型的且更复杂的项目通常会执行高质量的仪式。一般会提前确定好会议的召开时间、出席会议的具体参与者、议程、备忘录及后续进程。说明书会用丰富的图表和文档正式记录下来，而且经常会通过开发者和用户之间正式的评审会议进行验证。

项目的仪式与方法是否是预测或自适应是不一样的。然而，即使方法和仪式是不同的，大型的预测项目通常有高质量的仪式，并且有许多会议和文档。不幸的是，执行文档可能会增加项目的长度，有时还会导致成本超支。像快速应用程序开发（RAD）这样的技术通过使用少量的正式手续来帮助管理大型的预测项目。这个方法需要少量的文档和状态评审会议。当然，许多小项目通常用很少的仪式进行管理。

自适应的项目在管理方式上可以是更正式化的或者非正式的。统一过程就是相当正式的，并且带有高质量的仪式。每次迭代都要有具体的成果被精确定义，比如说明书、图表、原型和可交付的成果。然而，自适应的迭代方法也使它本身变得被非正式化地管理了。迭代方法继承的特征，以及它“刚好准时”的项目计划，能很容易适应更少的文档、更少的解释图表以及更少的正式状态报告。在几个章节都讨论过的敏捷开发就是一个典型的少仪式的方法。

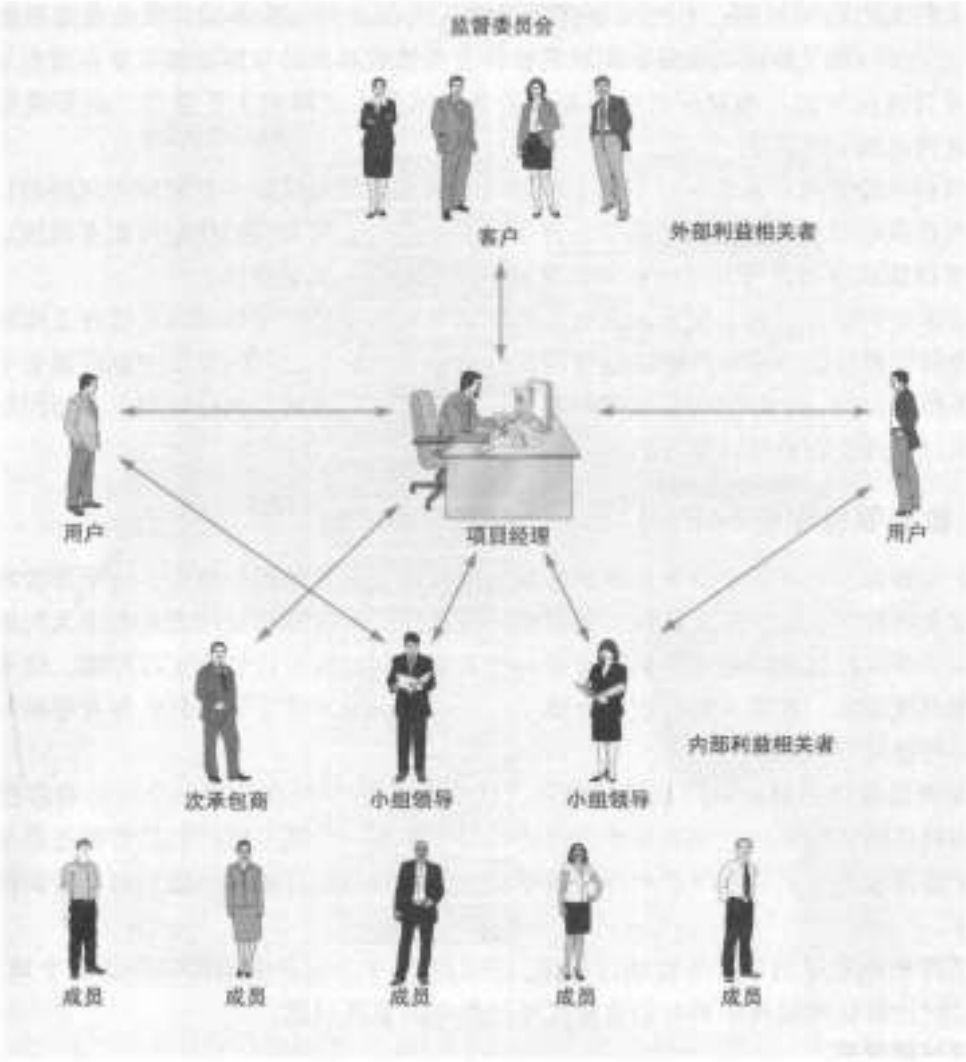


图 9-2　系统开发项目中的参与者

9.2.4　项目管理知识体系（PMBOK）

项目管理协会（PMI）是推进项目管理的专业组织，主要分布在美国，也分布在世界各地。此外，其他国家的专业组织也在推进项目管理。PMI 有一个权威且严格的认证程序，许多公司会鼓励他们的项目经理去获取 PMI 认证。

作为其中的一部分任务，PMI 已经定义了一个项目管理知识体系。这个知识体系叫作**项目管理知识体系**（PMBOK），是每个项目经理都应该了解的且已被广泛接受的信息基础。PMBOK 由九个知识领域组成：

- 项目规模管理：定义和控制需要被包含在系统中的功能及项目要做的工作的范围。
- 项目时间管理：创建一个所有项目任务的详细进度表，然后定义好里程碑来监控项目的进程。
- 项目成本管理：计算初始的成本 / 收益分析，以及之后的更新和作为项目进程监控的支出费用。
- 项目质量管理：为确保质量建立一个全面的计划，包括项目每个阶段的质量控制活动。

- 项目人力资源管理：补充和雇佣项目团队成员，也包括培训、激励成员和团队的建立，举行相关的活动来保证团队的愉快并营造有活力的氛围。
- 项目通信管理：确定所有的利益相关者和每个人之间的主要通信，也要建立所有的通信机制和进度表。
- 项目风险管理：确定和评审整个项目所有潜在的失败风险并制定减少风险的计划。
- 项目获取管理：制定提案需求、评价投标、签订合同和监控供应商服务性能。
- 项目集成管理：将其他所有知识领域集成为一个无缝的整体。

随着事业的进步，你会记录从其他人那里观察到的项目管理技能以及你在工作经历中学习到的项目管理技能，这样的做法是很明智的。一开始就在之前的章节中就讲到系统分析员需要很多技能。而一个好的项目经理则需要知道如何制定计划、执行计划、参与问题和做出调整。项目管理技能是可以学习的。

9.2.5 敏捷项目管理（APM）

在上一章你已经学习了开发系统的敏捷方法及敏捷开发的四个价值，这个方法相较于计划和被定义的程序更倾向于灵活性。很显然，这些价值对管理项目的方式有很大的影响。然而，有一个令人担忧的问题是它们隐含着一个无法控制且没有计划的工作环境，这可能会使项目管理出现混乱。在第 8 章，我们介绍了术语混序，混序在保持项目整体有序控制的基础上允许一些混乱发生。

敏捷项目管理仍然是一门新兴的学科，IT 行业还需学习在保持一个项目有序控制的前提下，如何在团队的灵活性和混乱性之间达到最佳平衡。尤其是敏捷项目管理还是平衡这两个互相矛盾需求的方式：在维护对项目进度表、预算和可交付成果的控制时，要如何保持敏捷和灵活。

为了帮助你更好地理解敏捷项目管理，我们接下来会讨论 PMBOK 中的五个知识领域，并且也会讨论在使用敏捷原则时包含在实施过程中的重要问题。

敏捷规模管理

规模管理是指新系统的规模和项目的规模。在传统的预测方法中，项目经理和团队会在项目的初始阶段也就是计划阶段就企图定义系统和项目的规模。不幸的是，对于大多数新系统来说，有太多的未知问题，比如规模基本上从来没被实际定义过。敏捷理论接受规模没有被很好理解这个事实，也接受在项目过程中存在很多需求的变化、更新和修改这个事实。然而，不被控制的规模可能会导致项目不能完成，即使这是个敏捷项目。项目经理必须制定一个过程和机制来控制项目的规模。这要怎么做呢？

让我们假设计划迭代的主要成果之一是一个决定，这个决定是要制定一张关于新系统需要支持的业务需求的优先级列表。图 9-3 代表的就是这份列表，顶端的是有较高优先级的条目，底部则是优先级较低的。这些需求通过使用几个准则来区分优先级，包括业务、风险、复杂性、大小和对其他依赖程度的重要性。在大多数的项目中，会结合使用这些准则来优先化需求。图 9-3 也说明了这个项目团队已制定预先的安排，以将这些需求加到迭代中。在新需求被定义时，它们就是优先的，并将会被加入栈中及安排到迭代中。

控制规模是由客户通过项目团队和用户提供的输入来做出决定的。在迭代项目中，一个可交付的成果通常是在每个迭代的最后才产生的。因为系统在整个项目中不断成长，并且最高优先权的需求会首先被实施，这样客户就能在他感觉到系统已经足够完整到可以迎合业务

需求时暂停项目。大多数项目通常需要一、两个或更多迭代来完成最后的集成和测试，以此来确保系统能有更多的容量，而且还能符合所有以安全为目的的“硬化”需求。

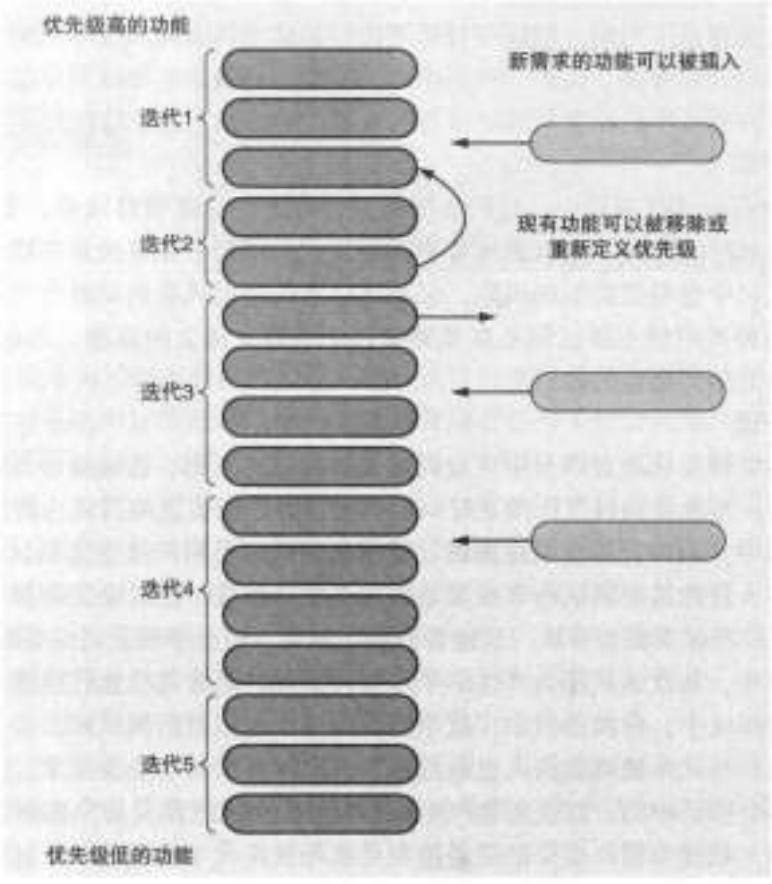


图 9-3　改变需求的范围管理

敏捷时间管理

传统时间管理要担心的首要问题是安排任务：创建进度表、根据进度表安排工作以及在进度表的基础上监控项目的进程。在预测项目中，进程表在初始计划阶段就被建立，然后会被加到项目进度计划系统，例如 Microsoft Project。

在敏捷项目中，由于需求总是在变化，所以创建维护一个重要的项目进度表是很难的。初始计划阶段的努力会包括一系列需求的开始和将项目分到迭代中，也就是把需求的初步安排加到迭代中。然而，迭代的数量和安排会随着发现的新需求和优先级别的变化而变化，这是在预期之中的。

迭代通常会持续 2 – 4 周，一个更详细的进度表会被创建。敏捷理论包括仅仅针对小型工作项目的思想，在这个小型工作项目中执行任务的时间几乎是同时的（例如，在一个迭代中），同时也会建立一个重要的进度表。此外，项目团队（不是项目经理或团队领导者）将会安排自己的工作进度。因此，对于敏捷项目，每个迭代都会作为首要任务来计划。这样任务就确定了，工作努力程度的评估也建立了，项目团队成员的工作也安排好了。因为一个项目中有很多迭代，所以项目团队要获取很多的实践经验并且能快速地精通评估和安排工作进度。

敏捷成本管理

客户这类利益相关者问出“新系统开发要多久或成本是多少”这样的问题是很正常的。但这些问题是很难回答的。对于预测方法，项目经理会给出评估，但是就像我们之前说过的那样，这些答案通常是不对的。敏捷项目经理能轻易地承认时间和成本很难评估，尤其是需求在整个项目中会经常改变。因此，评估项目的成本在项目的生命周期中没有控制成本那么重要。项目经理控制成本的职责对敏捷项目的重要性就像是它之于传统预测项目。

敏捷风险管理

在大多数自适应迭代项目中，包括敏捷项目，要密切注意项目风险，尤其是技术风险。迭代项目通常是风险驱动的，这就意味着早期的迭代具体集中在解决最关键的项目风险。尽管在一个预测项目中也包括类似的风险，但是将特定的降低风险活动融合到项目进度表中是很困难的。这两种类型的主要区别是在预测项目中会建立独立的原型，而在自适应项目中，新系统中风险高的部分会首先被创建。

敏捷质量管理

通常，质量管理必须处理项目中可交付成果的质量。然而，在敏捷项目中，我们也会考虑到过程的质量。怎样使项目工作得更好？内部程序是如何促进项目成功的？

在预测项目中，最终任务由系统测试、集成化测试以及用户接受度测试组成。然而，在项目的最后安排大量的这些测试会导致很难做出必要的改变，而且会花费很多时间。还有一个选择是用最少的测试来配置系统，这能帮助制定预算，但会导致公司的系统产生许多问题。

在敏捷项目中，每次迭代都会产生一个可交付成果。通常每次迭代也能将一个新部分整合进正在成长的系统中。每次迭代中，这个新部分是由他们自己测试的，而且会和系统的其他部分进行整合。测试系统功能的人也包括用户，这样可以符合业务需求。因此，测试和质量控制贯穿在整个项目中的，而且通常产生一个很好的测试过的且更坚固的系统。

另外一种作为敏捷项目一部分的质量控制是在每次迭代中的最后一个过程性评价。换句话说，项目团队会做一个自我评价来估计到底做得怎么样，以及能够多做些什么来提高下一次迭代。

9.3　核心过程 1：确定问题并获得批准

本书中的自适应系统开发生命周期包括六个核心过程。第 2 章概述了核心过程 3（“发现和理解细节”）的活动，第 6 章概述了核心过程 4（“设计系统组件”）的活动。在本章，我们会讨论核心过程 1 和 2 的活动。

核心过程 1 可能是项目取得成功最关键的过程。正如我们在 Standish 报告中提到的，建立运行支持、清晰的商业案例及有效的计划这样的事情对项目的成功是很关键的。这些重要的因素在核心过程 1 中会被确定并解决。图 9-4 强调了与核心过程 1 有关系的四个活动。

9.3.1　确定问题

信息系统开发项目的创立是因为各种各样的原因，包括为了回应机会、为了解决问题或为了回应外部指令。

大多数公司都在不断寻找能增加他们的市场份额或开拓新市场的方式。他们创建机会的一种方式是制定战略计划，分为短期和长期。在很多情况下，计划是确定新项目的最理想的方式。随着战略计划的开发，项目得以确定、优先化和安排。

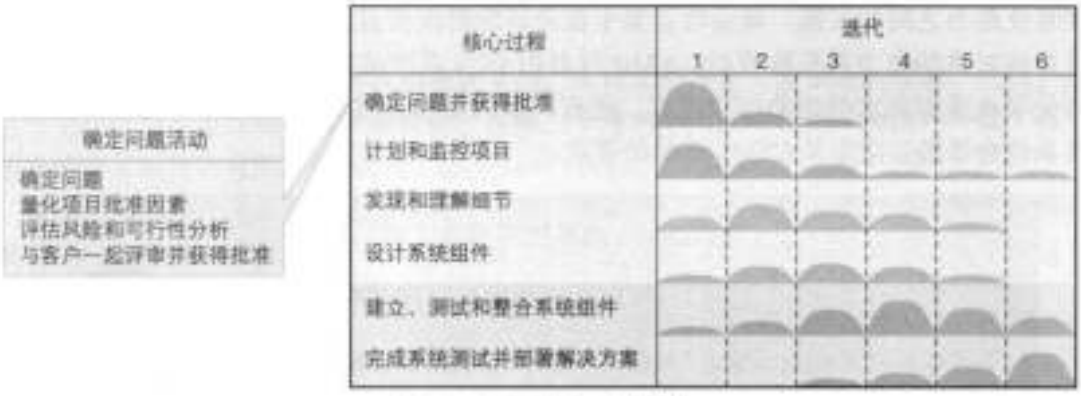


图 9-4　核心过程 1 的活动

项目的创建也是为了解决现有的业务问题。这样的项目是作为战略计划的一部分被创建的，但是它们通常是由中层管理者申请的，这些管理者想要关注公司操作中的一些困难。有时，这些需求是非常关键的以至于它们引起了战略计划团队的注意并且被集成到整个业务战略中。在其他时候，有些直接需求不能耽搁，比如一个新销售的委任进度表或者是评估生产力需要的新报告。在这样的情况下，业务功能的经理会申请创建一个独立的开发项目。

最后，项目的创建被用来回应外部指令。通用版本若需适应立法变更，就需要收集和报告新信息，例如税收法及劳动法的变化。立法变更也能扩大或影响组织在市场中提供的服务和产品的范围。电信行业的监管变化为有线电视和电话公司的竞争打开了大门，而这些公司也正在争夺机会，如提供基础服务、互联网接入以及个性化的娱乐。

确定及仔细定义问题是一个成功项目的关键活动。目标是确保新系统能真正地符合业务需求，同时能精确地定义业务问题及决定新系统的范围。这个活动定义了你想达到的目标。如果这个目标是定义不清晰的，那么所有接下来的活动也会不清楚。例如，系统要做出一个“能跟踪销售人员任命”的请求。不了解这个请求的背景环境的话，这个系统的建立仅仅是记录这些任命，而忽略了税收报告、内部外部销售人员、延期的任命、复杂关系、共享任命等它们之间的复杂性。因此，即使所有的说明在初始阶段没有被定义，但是要定义足够的需求来理解大多数解决方案的实施。

定义问题的一个有效方式是创建**系统可视化文档**，这个在第 1 章就介绍过了。这个文档有三个组件：问题描述、商业收益和系统能力。

创建系统可视化文档的首要任务是回顾建立项目时的业务需求。如果项目是作为战略计划的一部分被开发的，那么这个计划文档就需要被查看。如果这项目是起源于部门需求的，那么需要向关键用户请教，以此来帮助项目团队了解业务需求。从这个任务中得知，需要建立清晰的问题描述。随着这些需求的确定，项目团队也要创建一份期望的商业收益的详细列表。商业收益列表包含组织期望从新系统中获得的成果。商业收益通常是按照具体结果被描述的，这里的具体结果是指通过减少成本或增加收入而改变财政状况。

随着商业收益的确定，项目团队也要确定新系统的具体能力来支持这些收益的实现。这个任务的目标是按照信息系统的需求来定义问题的范围。这个范围状态是由**系统能力**列表定义的，也是为了帮助确定新系统的大小和复杂性及需要的项目。

开发团队的成员是和用户、客户一起工作的，将问题描述、商业收益和系统能力这三个组件结合成一个系统可视化文档。图 9-5 代表了 RMO 的系统可视化文档。要注意商业收

益和系统能力之间的区别。商业收益集中在公司的财政收益上，系统能力集中在系统本身，通过系统提供的能力来获取收益。RMO 现有的 CCS 系统是在严格的截止日期下被建立的，同时公司也意识到它的生命相当短暂。还有许多关于 Web 市场的东西要学习，但是现有的 CSS 系统会帮助公司定义 CSMS 系统的需求。

CSMS 的系统可视化文档

问题描述

自从 CSS 系统建立以来，在线销售和市场营销经历了翻天覆地的变化。顾客变得越来越有经验，他们已习惯了商品目录系统及销售系统。这些系统使用起来很方便，并且能够提供许多服务，如一键下单、延期付款跟踪、简易查询和同类商品比较。此外，研究还显示，当基本销售功能与社交媒体营销工具相结合时，销售业绩会大幅增长。因此，开发新的 CSMS 不仅是为了应对当下的市场竞争，更是为了使 RMO 加速融入今天的全球化社交媒体和移动计算。如果 RMO 不及早开启这项计划，那么它必将错失重重机会。

系统能力

这篇文档从宏观上确定了我们需要的系统能力，之后的文档将说明需求的细节。具体能力包括：

- 购物车功能。
  - 支持高度自动化的销售（如一键下单等）。
  - 推荐相关商品，比较同类商品。
  - 允许顾客进行评价和评分。
  - 具备“好友”网络功能。
- 订单实施功能。
  - 支持打包订单或拆分订单的运输和跟踪。
  - 支持待代发货订单和跟踪。
  - 允许顾客进行评价和反馈。
- 顾客账户和账单功能。
  - 提供个性化的顾客账户。
  - 支持电子账单和多种电子支付方式。
  - 计算“积分”并允许顾客换取和分享“积分”。
- 用于促销和特价的营销功能。
  - 提供灵活的促销和销售。
  - 从供应商到顾客直接计算并跟踪“积分”。
  - 与社交媒体交互，发布广告或组织营销活动。
  - 支持移动设备参与社交媒体营销及销售。

商业收益

通过与顾客的连接，这些功能提升了顾客体验，因此其主要商业利益是提高销售额。具体利益包括：

- 提高顾客的购买规模。
- 提高顾客的购买频率。
- 提高顾客的满意度。
- 提高顾客及其好友的商品评论数量。
- 通过推荐和社交媒体营销吸引新的顾客。
- 通过推荐和服务建立顾客忠诚度。
- 使商品更快到达用户手中。
- 减少运输延迟和储运损耗。

图 9-5 RMO CSMS 系统的可视化文档

9.3.2　量化项目批准因素

第一个活动产生了一个高层次的概述文档，这份文档确定了新系统的需求。然而，仅仅是这份文档可能不足以接受批准和资助。在第二个活动期间，项目团队和用户一起工作，以更精确地定义项目的范围和影响。

我们的目标是产生足够的正当理由，这样才能获取资金，项目才能启动。有时，需求太大或太明显以至于项目批准几乎是自动化的。在其他情况下，准备一份完整的成本收益分析是很有必要的。必须考虑以下这些准则以获取项目批准：

- 项目完成的预计时间。
- 项目和系统的预计成本。
- 新系统配置的预期收益。

这些是很难评估的。在传统预测的系统开发方法中，经常是用大量的细节来做出评估。然而，这类评估通常是不符合实际结果的。当然，问题是在使用新系统时，项目团队会冒险使用未知的需求和技术。用更多的自适应方法能使利益相关者意识到有些需求是未知的而且监控和控制成本、范围和进度表比尝试做评估更重要。

项目完成的预计时间

在核心过程 2（"计划与监控项目"）中，会创建一个更详细的项目进度表。在项目初始阶段，我们对项目的有限了解通常不足以创建出进度表。但是不管怎样，还是需要估计项目完成的日期，即使这是最难做的事情之一。

有时，业务上的限制决定了项目的完成。例如，新的立法需求可能会影响项目日程。这种机会之窗为特定时间内完成项目提供了强大的动力。这些因素都要在项目审批和项目计划过程中加以考虑。

估计项目完成日期的材料是一些范围文件以及需求列表中所要开发的工作量。如前所述，这很难做出任何程度的准确估计。在这个项目早期，对团队大小和大体时间做总量估算通常是最好的，且可以实现。对于预测方法，需求列表可以作为估算定义和开发一个特定功能所需工作量的起点。对于预测方法，同样的信息可以被用来评估在各种各样的子系统中需要迭代的次数及团队的大小和数量。

图 9-6 所示为 RMO 时间评估文档的一个例子。

新 CSMS 项目的时间估计			
子系统	功能需求	迭代需求	估计时间
销售子系统 *	15	5	20 周
订单实施子系统 *	12	5	20 周
顾客账户子系统 **	10	4	15 周
市场营销子系统 **	6	3	13 周
报告子系统 **	7	4	12 周
总开发时间（2 个团队）			40 周
最终强化度和接受度测试		2	8 周
总的项目时间			48 周

图 9-6　CSMS 项目的项目完成时间估计

注：* 安排给 Tiger 团队

** 安排给 Cougar 团队

对于RMO，时间评估的创建需要一天的时间。由于项目还没有被批准或获得资金，因此项目经理和系统分析员都没有被安排到项目中。然而，项目经理被安排去获取批准，还有两个系统分析员被安排去帮助他们。这三个有经验的技术人员与RMO不同部门中的用户会见四个小时。这些会议的目的是建立一份每个部门所有需求的全面列表。在会议结束之后，这些人会再一次见面以将需求列表组织进各部分中，这些部分是为软件开发的各个迭代所安排的。

新的开发项目的指导者也做出了一个假设，那就是会为项目再分配两个四人的子团队。如图9-6所示，这个项目的预计时间是48周。

项目和系统的预计成本

图9-7所示为开发新CSMS系统的预计成本。到现在为止，项目预算中最大的成本是项目团队的工资。其他成本元素包括新电脑的成本、用户、办公室、设备及项目团队联合培训、项目团队为做站点访问的旅行费用以及软件许可证。就像你看到的那样，这些评估就已经达到了150万美元。

CSMS的开发成本小结	
花费目录	价格
薪水/工资（包括收益成本）（1个PM，8个分析员，1个支持）	$936 000.00
设备/装置	$308 000.00
培训	$78 000.00
设施	$57 000.00
实用程序	$97 000.00
出差/其他	$87 000.00
许可证	$18 000.00
总价	$1 581 000.00

图9-7　CSMS的开发成本小结

在项目投入运行后，每年就有正常的运行费用，如图9-8所示。花费成本最多的是为主机服务提供一些设备，以及网络的连接和服务器管理服务。所预计的这些成本是以RMO使用主机服务来提供设备、网络连接和服务器管理为基础的。项目团队估计那些花费大概是一个月13 000美元，这对15个非常大型的受管理的服务器来说已经足够了。根据数据交换量，这样的估算绰绰有余。其他成本主要是全职程序员和两个客服平台职员的成本。

CSMS估计的年运行费用小结	
常规费用	价格
连接/主机代管	$156 000.00
编程	$75 000.00
帮助台	$90 000.00
总价	$321 000.00

图9-8　CSMS估计的年运行费用小结

新系统配置的预期收益

系统可视化文档确定了新系统的预期商业收益。在这个任务中，我们会分析那些商业收益并且提供它们对于组织的价值评估。这个价值会成为总决策准则的一部分。很显然这些

与存款或收入有关系的金钱数量必须由客户来估计。预测商业收益的价值不是项目经理的工作。然而，项目经理能帮助客户确定潜在收益的目录。增加的收入或成本下降收益的典型内容包括：

- 用新服务、产品或地理位置开拓新市场。
- 增加现有市场的市场份额。
- 增强对现有顾客的交叉销售能力。
- 通过自动化功能减少员工或增加效率。
- 减少操作费用，例如“紧急出货”的运输费用。
- 通过自动化编辑或验证降低错误率。
- 减少坏账或不良信贷损失。
- 通过更严格的控制来减少库存或商品的损失。
- 更快速地收集应收票据（应收账款）。

RMO 的项目团队会和销售与市场营销副总裁一起工作来确定收益范围并且为每个范围评估价值。投资的大小和持续的费用将需要 RMO 董事会批准贷款。董事会想知道新系统的收益是什么以及投资的回报是什么。RMO 的一大困难是决定如何为收益赋值。一个典型的问题就是：“考虑到这个系统需要在市场上保持竞争力，我们要为所有的销售活动赋值吗？或者我们仅仅为那些期望能从营销和更高销量中获得的增长销售赋值？”如果因为 RMO 在市场上的竞争力减弱而导致销售下降，那么就可能用到总销售额的历史记录。然而，如果现有系统已经足以维护一个良好的客户基础数据，那么就应该用到销售增长。这些问题的决策通常是用户需要的，而不是项目经理。在这种情况下，RMO 的销售与市场营销副总裁会决定使用更保守的评估。图 9-9 总结了他所做的评估。

CSMS 的年收益估计	
收益或成本节约	价格
拿回 / 预防失去的销售	$200 000.00
为现有顾客增加销售	$300 000.00
销售给新顾客	$350 000.00**
为处理提升效率	$50 000.00
由于主机代替减少数据中心和设备的成本	$146 000.00
总价	$1 046 000.00

图 9-9　CSMS 的年收益估计

注：** 加 8% 的年增长

许多组织喜欢将被估计的成本和预期的收益作比较，以此来计算收益是否超出成本。这个过程称为**成本 / 收益分析**。公司会使用多种方法测定新系统的总收益。一个通用的方法是确定新系统的**净现值**（NPV）。净现值的两个概念是：（1）用如今的美元（现值）计算所有的收益和成本；（2）把收益和成本结合起来给出一个净现值。未来收益和成本的流量是同时得到的，以后每年用一定的因子进行折算。贴现因子是用于把未来的价值变换成现在价值的比率。

图 9-10 所示为 RMO 的新销售与市场营销系统净现值计算的副本。为一个给定的投资计算净现值有各种各样的方法。在这个例子中，第 0 年代表了开发阶段优先于系统配置。每

年的年收益位于第 1 行。开发成本是第 2 行。第 3 行是每年的花费。第 4 行是前三行之和，表示的是净收益和净成本。第五行是贴现因子，给出了 6% 的折扣率。第 6 行表示的是第 4、第 5 行的产品及按照如今美元计算的现值（如净现值）。第 7 行显示了每年净现值的累计总量。

	A	B	C	D	E	F	G	H
1				RMO的CSMS成本/收益分析				
2		目录	第0年	第1年	第2年	第3年	第4年	第5年
3	1	收益值		$1,046,000	$1,074,000	$1,104,240	$1,136,899	$1,172,171
4	2	开发成本	-$1,581,000					
5	3	年花费		-$321,000	-$321,000	-$321,000	-$321,000	-$321,000
6	4	净收益/成本	-$1,581,000	$725,000	$753,000	$783,240	$815,899	$851,171
7	5	贴现因子	1.0000	0.9434	0.8900	0.8396	0.7921	0.7473
8	6	净现值	-$1,581,000	$683,965	$670,170	$657,608	$646,274	$636,080
9	7	累积的净现值	-$1,581,000	-$897,035	-$226,865	$430,743	$1,077,017	$1,713,097
10	8	投资回报期	2年+	226865 / (226865+430743) = .35			或　2年+128天 (.35*365)	

图 9-10　CSMS 的 5 年成本 / 收益分析

在图 9-10 中，第 7 行的数量最终从负数变为了正数。这个情况发生的时间点称为**盈亏平衡点**。在出现盈亏平衡点之前的这段时间的长度称为**回报期**。回报期会发生在累积总量变为正数的那一年中。为了计算它，首先找出这个例子中前年累积价值是负数的时候，也就是第 2 年。然后加上下一年累积价值变为正数所花费的天数（这个例子中是第 3 年）。计算方法是将第 2 年的最终价值的绝对值除以第 2 年和第 3 年最终价值的绝对值的和——这个例子中是 226 865/（226 865+430 743）。在这里，这样的计算说明了累积价值在这一年过去 35% 后会变成正数。365 天乘以 0.35 得到的 128 天就是第 3 年的天数。许多公司需要新系统在两到三年中有一个回报期。

以前的成本 / 收益计算是依赖组织对成本和收益量化的能力。如果用美元可以估算收益或成本，组织就把它看作**有形收益**或成本。然而，在许多情况下，组织不能通过衡量成本和收益的一部分来决定价值。不要忽视确定项目幕后的重要性。可能会有一些覆盖了所有其他可行性分析的政治原因在支持或反对项目。如果没有估算或衡量的可靠方法，那么它就被认为是**无形收益**。在一些情况下，无形收益的重要性远大于有形收益，至少从客户的角度来说是这样的，即使美元数不能预示一个好投资，但是客户还是会继续开发系统。

无形收益的例子包括：

- 提高服务水平（不能用美元衡量的方法）。
- 提高顾客满意度（不能用美元衡量）。
- 生存。
- 需要开发内部专门技术（例如，用新技术的试点计划）。

有形成本的例子包括：

- 降低员工士气。
- 影响生产力（组织不能估算的）。
- 失去顾客或销售（在一些未知的时间段）。

9.3.3　评估风险和可行性分析

项目风险与可行性分析即验证项目是否已经启动并成功完成。由于每个项目都是独一无二的努力成果，所以每个项目都有唯一的影响其潜在成功的挑战。

这个活动的目标是确定及评估影响项目成功的潜在风险，并且采取措施来消除或至少减弱这些风险。它们应该在项目申请批准的过程中被确定，这样所有的利益相关者都能意识到潜在的失败。项目团队也能建立计划和程序来确保那些风险不会影响项目的成功。一般来说，在确定项目的可行性时，项目团队会自己安排这些任务：

- 确定组织的风险和可行性。
- 评价技术的风险和可行性。
- 评价资源的风险和可行性。
- 确定进度安排的风险和可行性。

确定组织的风险和可行性

每个公司都有自己的文化，并且任何一个新系统都必须适应那种文化。常常有这样一种情况，即新系统可能与原有的标准有极大的不同，从而造成系统不能成功使用。涉及可行性分析的分析员应该评价组织和文化的问题，进而确定潜在新系统的风险。这样的问题包括：

- 大量的计算机恐惧症。
- 部分员工或管理者的失落感。
- 因为新系统导致政治和组织权利的潜在变化。
- 工作职责变化的恐惧。
- 因为不断提高的自动化程度导致害怕失去工作。
- 撤销持续长时间的工作过程。

列举出所有现有的潜在组织和文化风险是不可能的。项目管理团队需要对组织中的反对意见非常敏感，以此来确定和解决这些风险。

在确定风险之后，项目管理团队可以采用积极的措施来应对它们。例如，项目团队可以组织额外的训练来教授新的程序并且不强化计算机技能。参加开发新系统中高级用户能增加用户的热情和参与度。

评价技术的风险和可行性

一般来说，新系统会为公司带来新技术，有时新系统扩展了技术的最新水平。其他项目使用的是现有技术，但是也会把它结合到新的、未测试的结构中。如果一个外部供应商在某一确定领域提供能力，那么客户组织通常会把这个供应商看作那个领域的专家，这样他们会负担所需技术水平太复杂的风险。

项目管理团队需要十分谨慎地评定所达成的技术需求和有用的专门技术人才。当这些风险被确定后，解决方案通常是公正而简明的。技术风险的解决方案包括提供额外的培训、雇佣顾问或雇佣更多有经验的员工。在某种情况下，项目的作用域和方法需要变化以改善技术风险。重要的是，现实的评估将较早地确定技术风险并有可能采取正确的措施。

评价资源的风险和可行性

项目管理团队也必须评估项目资源的可行性。主要的资源由团队成员组成。开发项目需要系统分析员、系统技术员和用户的参与。一种风险是项目组得不到所需要的人。另外一种风险是项目团队的成员没有项目所需的技能。在项目团队行使职责之后，就不断有成员离开团队的风险。如果其他特别的项目在组织内部执行或者合格的团队成员被其他组织雇佣时，这种威胁会在组织内部发生。尽管项目经理通常不愿意考虑这些可能性，但有技能的人供不应求，有时必定会有人离开项目组。

一个成功项目所需的其他资源包括足够的计算机资源、物理设备和维护人员。这些资源基本都可以得到，但是若在这些资源有用时耽误了，则会影响进度。

确定进度安排的风险和可行性

一个项目进度表的制定总会包含高风险。每个进度表需要许多假设和用不充分的项目信息所做的估算。例如，对新系统的需求（以及作用域）不十分了解。同时，所需时间要进行调研，而且最后的需求也要被评估。团队成员的可用性和能力也都是不了解的。

制定项目进度时会发生的另一个风险就是领导要求新系统必须在一定时间内配置好。有时，设定最终期限是因为有一个很重要的业务需求，例如 RMO 项目中要求及时完成 CSS 以便在假期中进行网上购物。同样，一些大学往往会要求在某些关键日期前完成新系统。例如，如果一个学校的新注册系统在入学季前没有完成，那么有可能必须延至下个学年。类似这样的例子说明，进度安排的可行性是最重要的可行性因素，我们必须考虑。

如果最终期限可以随意定，那么制定进度表的意图是表明我们能够建立一个进度表。遗憾的是，这通常意味着灾难。项目团队应该在没有需要完成日期的预想打算下创建进度表。完成进度表之后，要做相应的比较来确定时间是否一致。如果不一致，项目团队要采取正确的措施，如减少项目的作用域来增加项目按时完成的可能性。

定义项目进度表中里程碑和迭代的一个目的是允许项目经理评估项目进度表中正在出现的失误的风险。如果团队开始错过里程碑，那么项目经理要尽早采取正确的措施。可以创建并执行临时计划来降低风险进一步扩大的可能性。

9.3.4　与客户一起评审并获得批准

就像之前提到的那样，RMO 项目的经费需要董事会的批准。然而，在将项目介绍交给到董事会之前，RMO 的执行委员会需要理解并同意开发这个项目。这个项目会对公司所有的区域产生主要影响。像销售和市场营销这样的部门会受到直接影响。他们不得不分配员工和资源来帮助定义需求、开发测试案例以及在项目完成后测试新系统。换句话说，这个部门里的员工在接下来的 12 周或更长时间里将会有额外的工作内容。甚至那些没有被直接影响的部门也需要支持这沉重的开发活动，这也许会使他们的预算变紧。在任何事件中，获得整个公司的批准和支持总是好的策略。这个过程的开端是向 RMO 的高层做介绍。通常，会要求项目经理做项目的介绍或者至少在现场回答问题。

执行委员会批准这个项目之后，由董事会继续裁定。经过董事会批准后，IT 部门才开始为项目安排全部资源。此时，为全公司准备备忘录或者组织开会来标记这次主要活动的开端是一个很好的想法。如果整个公司都知道所有的管理层都支持项目并且要求合作，那么这个项目会进行得更加顺利。

9.4　核心过程 2：计划和监控项目

这个核心过程是在整个项目中持续发生的。在项目批准后就会立刻进行主要计划。正在进行的计划和项目监控会持续在所有项目迭代中。每个迭代不仅要像它开始被计划的那样，而且还要一直监控项目的进展，同时正确的行为也是需要的。图 9-11 通过在每个迭代中“所费精力曲线”的高度，说明了计划与监控活动必须是每个项目迭代的一个不可缺少的部分。与这个核心过程有关系的具体活动也在图 9-11 中列出了。我们接下来会单独讨论每个活动。

图 9-11　核心过程 2 的活动

9.4.1　建立项目环境

在本书中，我们已经讨论过不同类型的项目，例如预测和自适应项目，以及在这些项目中使用的工具、技术和方法。我们也已经讨论了仪式、项目报告、利益相关者、用户参与和项目团队工作环境。所有这些元素在项目开始进行时都要被放到相应的位置。一些决定的做出是以这个组织的标准政策和程序为基础的。其他的则会在批准过程中再决定。不管怎样，项目经理必须确保项目的参数和工作环境已经敲定，这样项目工作就可以在没有障碍和延迟的情况下继续进行。在项目开始进行时要解决重要的项目结构问题。例如，采用哪种类型的通信过程来保持团队和外部利益相关者的实时联系？此外，项目团队的成员都需要计算机和集成开发环境以及其他工具来工作。当然，关于项目团队如何满足用户需求、如何编写代码以及如何提交要验收的代码，这样的具体程序都需要被最终敲定。我们会讨论三个重要的考虑因素：

- 记录与通信——内部 / 外部。
- 工作环境——支持 / 设备 / 工具。
- 过程与程序。

记录与通信——内部 / 外部

项目经理与项目团队成员要参与所有类型的会议，在会议中做出决定和获取信息。哪些是重要的信息以及怎么记录信息都需要在具体的项目程序中陈述。其他关于信息的关键问题是：什么信息、怎样用、使用频率、给谁用。项目经理在一个新项目上的首要任务之一是为如何处理项目信息建立规范和指导。

IT 项目的一个关键成功因素是获得组织管理层和其他主要利益相关者的支持。一个好的项目经理会理解这一需求并且构造他的项目，这样他能用更合适的细节来和每个利益相关者进行沟通。图 9-2 确定了项目中各种各样的利益相关者和参与者。其中一些利益相关者会完整参与到项目中。其他利益相关者则参与不多，只接收周期状态报告。客户利益相关者（即支付项目花费的人）需要一直注意项目的状态及发生的任何困难或延迟。利益相关者分析能帮助我们确定那些对项目有兴趣的人，而且也定义了他们想要的相关项目信息。一般来说，我们把这个分析作为项目信息的外部报告。

维护项目信息可以通过电子方式来实现。进度信息可以上传到网站上，这样每个人都可以看到。另外一种类型的项目跟踪工具有时也称为项目仪表板，它允许上传所有类型的项目信息并且可以通过网页浏览器查看。图 9-12 是一个项目仪表板系统的例子，它允许对项目信息做出简单的评价。电子数据表、电子邮件、时事通信和列表服务器都能提供维护、收集

和分配信息的方式。一旦电子系统被建立，它们经常会进行自我管理。

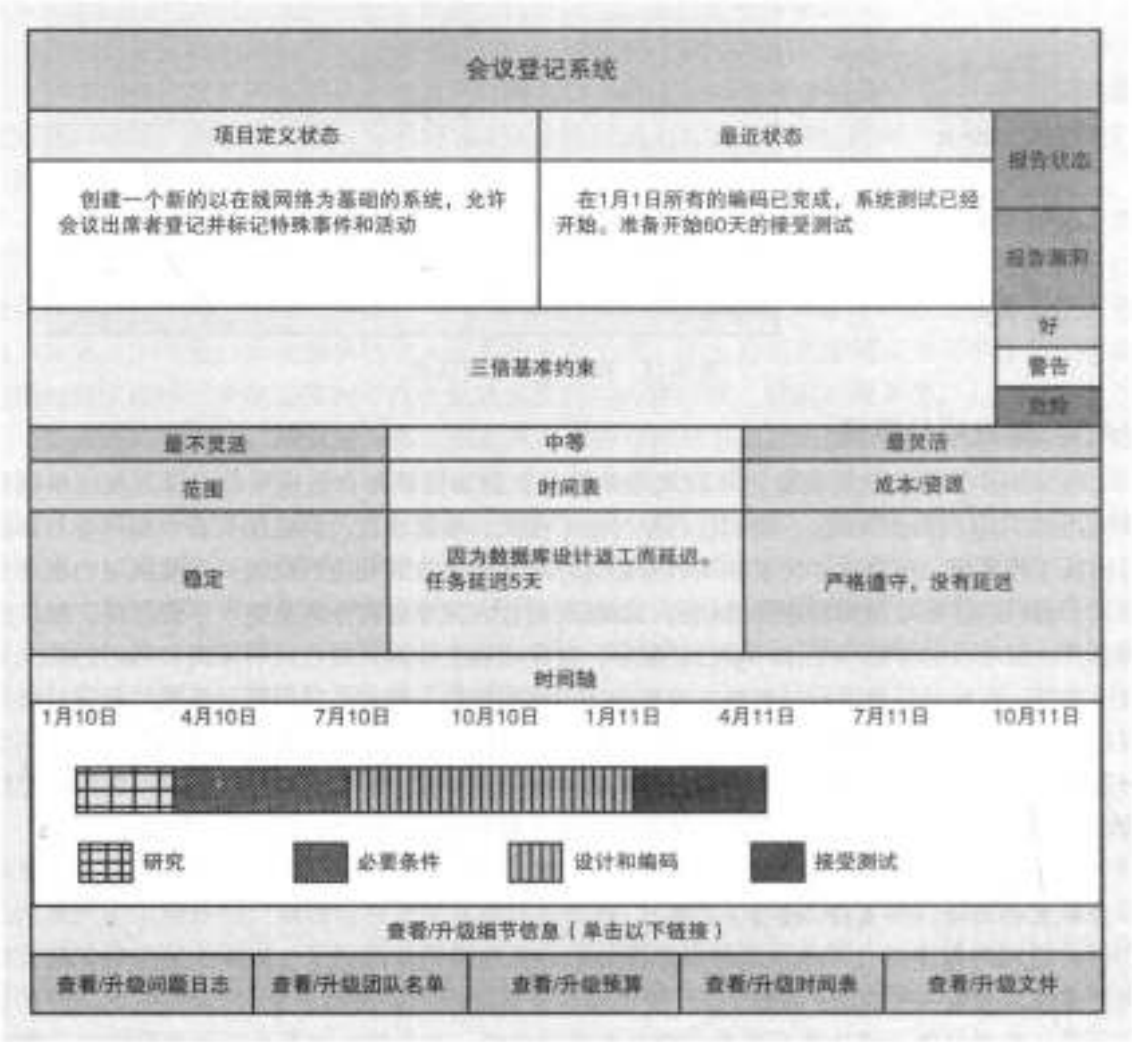


图 9-12　显示仪表板项目信息和状态的例子

项目团队的成员也需要制定他们互相之间通信和记录项目决定的机制。这是不同类型的信息——系统开发时的信息。例如，在分析活动时，项目团队会通过使用各种各样的方法（如写下用例描述）来记录用户会议的结果。在设计阶段，信息也需要被记录并且被分配到合适的团队成员中。在测试阶段，当发现错误时，要把错误记录下来并且安排程序员来解决。最后，整个记录和通信需求通常被位于世界各地的项目团队的成员（以及用户）变得更具有决定性。图 9-13 列出了一些需要获取及维护的信息。图中的数据资源库通常包含许多不同类型的数据结构和存储技术——从维基百科到数据库到问题跟踪系统。

关于记录和通信有一个警告。在传统预测项目中，倾向于创建大量的文档。你在之前的章节中已经学过，使用敏捷原理的自适应项目强调代码大于文档。一个新手项目经理可能会解释，这意味着文档是不必要的。然而，即使是敏捷方法，为了以后进行确认，基础用户的定义也需要被记录下来。对于程序员来说，在编程期间不得不回顾笔记和模型来记住一个独立需求的真正细节并做出决定，这是很正常的。一个有经验的项目经理知道文档的合适数量，这样对于项目并不是超负荷的开销，而且关键的决定已经被记录下来了。

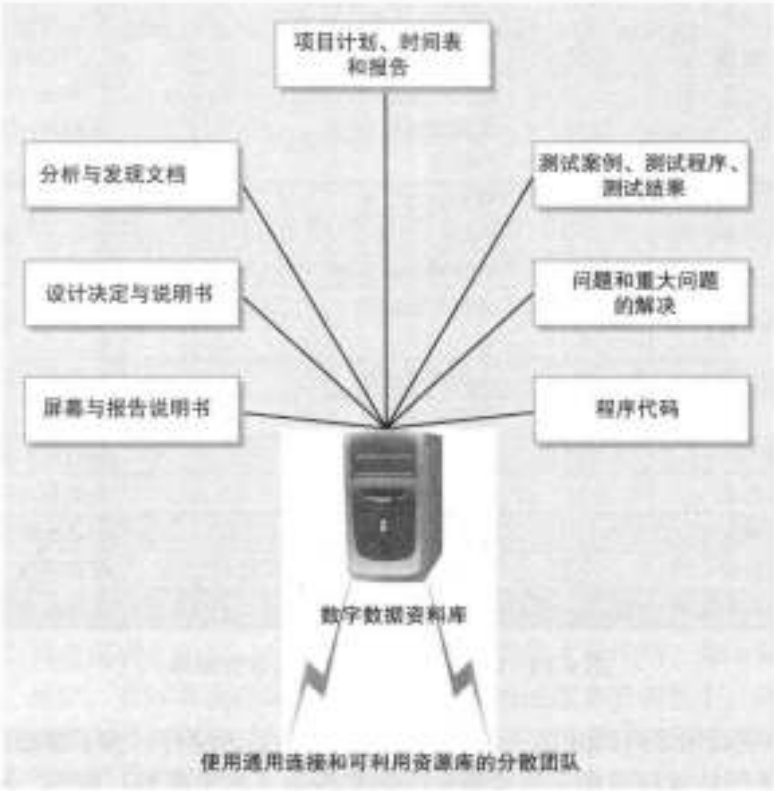


图 9-13　数据资源库中存储的系统信息

很明显，必须做好全面的记录和通信计划。幸运的是，在如今这个相互联系的世界里，有很多工具可以使用，所以外部和内部的通信很容易做到。因为有着太多的电子工具，所以所有的项目信息都要在网上可以查到并且开放给所有利益相关者。事实上，由于维基百科的使用，现在允许很多项目团队甚至用户来辅助记录和更新关键项目信息，这样的情况是很常见的。

CSMS 团队想要以数字格式来维护项目信息，并且开放给所有利益相关者，包括团队成员、用户、客户及监督委员会成员。RMO 是一个由理论指导的非常开放式的商店，其中的理论是信息要被广泛分配。图 9-14 所示为 CSMS 项目团队用来通信和获取信息的所有工具。核心团队成员先前就已经在多个敏捷项目中工作过，所以他们已经学到了文档需要有一个正确的平衡状态，不是太多，而是要足够跟踪关键决定和需求。Barbara Halifax 项目经理想要确保工具是到位的，这样才能在谨慎做事时很容易地记录信息。

用户文档（如样品发票）会被扫描且放到一个文档资源库中。用户功能定义会被记录在一个论坛系统中。当关键问题被讨论且需要被记住时，团队成员和用户可以使用论坛来更新它。样品屏幕和报表布局要么被草拟出来，要么用 Visio 或 Keynote 画出来。徒手草图经常会经扫描然后存档。大多数设计决定和说明要符合编程代码而且不会被记录下来。然而，有一些决定是全球化的，而且能在维基百科中搜索到。

每天，项目团队都会开展一个“动员”会议，也是一个短暂的协调会议。大多数团队成员都生活在帕克城中心，但是还有一些人是从其他地方调过来而被安排到团队中的。有时，团队成员会访问用户站点，因此那些人无法调配，而且还有一些人是在盐湖城的办公室工作的。因此，日常会议被定为视频会议，每个人使用自己的网络摄像头和个人电脑。这个会议通常只持续 15 分钟。

电子数字资源库		
获取的数据	电子工具	谁可以更新 / 查看
用户定义和功能 用户文档	软件论坛 文档服务器 扫描仪	分析员、用户 / 所有
屏幕和报告布局	网页设计工具 Visio PowerPoint/Keynote	分析员、用户 / 所有
设计说明书和图	维基百科软件 Visio	分析员 / 所有
重大问题和未解决的问题	问题跟踪软件	分析员、用户 / 所有
程序代码	源码配置管理（SVN）	分析员 / 分析员
项目进度表	MS Project	分析员 / 所有
项目状态和信息	软件论坛	分析员、用户 / 所有
日常团队协调会议	笔记本电脑视频会议	项目团队
分散团队的交流	IM 聊天和视频	项目团队
项目更新的新闻邮件	博客软件	项目经理 / 所有

图 9-14　CSMS 信息的电子数字资源库

最后，有一些讨论是两周发送一次关于项目进度的新闻邮件。为了激励员工的热情和支持，Barbara 认为保持对项目的了解对整个公司来说是非常重要的。然而，相对于打印版的新闻邮件，她选择了以博客的形式来做它。所有用户都会被邀请使用 RSS 订阅，以此来保持对项目进度的了解。

工作环境

尽管工作环境更多地是与项目团队的工作过程有关，然而项目经理必须确保能适当允许项目团队更有效地工作。工作环境有五种主要的组件：

- 个人电脑或工作站。
- 个人开发软件和工具。
- 带有资源库、沙盒和通信工具的开发服务器。
- 办公室空间、会议室和设备，包括打印机、扫描仪和投影仪。
- 支持员工。

最重要的当然是计算机设备和团队将会需要的其他硬件。很显然，每个开发者都需要自己的计算机配置，这个配置可能由多个计算机或者显示器组成。其他重要的硬件包括开发服务器、打印机和内部开发网络。如果团队被分散了，此时就很有必要用摄像机和投影仪来管理被分散团队的会议。伴随着硬件的使用，一定要有可用的资源来管理开发服务器之类的事物。

和硬件有关的就是计算机软件和其他工具。软件工具可以变得相当复杂——从独立的集成化开发环境工具到建模软件再到代码库软件。有着自己的环境和软件的开发服务器必须也要经过配置和部署。这个服务器可能是作为一个虚拟服务器或是一个独立的计算机被建立的。应用程序包括代码库、问题跟踪应用程序、测试系统和项目仪表板。

伴随着硬件和软件的使用，要为每个开发人员提供一个工作配置，要有登录允许、沙盒环境、资源库连接等。最后两个组件是办公室空间和其他可能需要的设备。这包括与会议室的连接、演示设备甚至是运输车辆。最后，当足够的支持人员可以照顾到那些伴随着项目的

大量细节时，团队成员的工作效率总能增强。

过程与程序

一系列主要的最终决定要结合项目的内部过程和程序来做出。先前，我们讨论了一个项目的正式程度。较大的项目需要更加复杂的报告过程和会议进度。当很多人参与其中时，活动之间的协作变得很关键。程序包括：

- 报告与文档——要做什么？怎么做？谁来做？
- 编程——个人或双人编程？工作怎么安排？由谁来安排？
- 测试——程序员测试或用户测试？怎么标记准备要测试的项目？
- 可交付成果——是什么？什么时候要怎样交给用户？它们怎样被接收？

代码和版本控制——怎样控制代码以阻止冲突？如何用新系统协调修复 bug？什么时候以及怎样才会产生可交付成果？

9.4.2 安排工作进度

安排工作进度对任何大小或类型的项目都是有必要的。然而，根据项目的类型，使用的技术可以相差很大。对于预言性的、被高度控制的项目来说，通常会建立一个详细的、完整的且包含整个项目的进度表。而且，只有这些类型的进度表才有作用，因为要建立的软件必须被很好地理解。然而，有时甚至在那些有详细且全面的进度表的项目中，随着项目进展中事物的变化，有些事物需要被调节。另一方面，小的敏捷项目有时甚至没有项目进度表，只是团队成员负责安排他们自己的工作。合作是通过交谈以保证双方都能了解每个人的工作内容是什么来完成的。这也是混序的意思。

为现今的许多项目安排工作总是处于两个极端。大型项目可能有多个独立的开发团队来开发各种子系统。即使团队之间的工作是很独立的，但合作也还是需要的。自适应项目也会预见到初始进度任务的额外需求和变化。

对于自适应类型的项目，创建项目进度表是贯穿项目生命周期的。在初始计划阶段中，用例或用户商店的初始列表是为每个子系统开发的。用例会被分配且暂时安排到迭代中。我们把它称为**项目迭代进度表**。随着每个迭代的开始，我们会创建要被完成的任务和工作的详细进度表。你在第1章中看到了创建迭代进度表的例子。让我们把它称为**详细的工作进度表**，这意味着它在一个迭代中安排了工作。有时，在每个迭代中，通常是在一个迭代完成并且在下个迭代开始之前，项目经理和团队领导者的助理以及关键用户会一起评审和修改项目迭代进度表。在这个过程中，发生的变化和任何新的需求都是具有优先权的，并且会被放到进度表里。

创建项目迭代进度表必须考虑到解决方案系统的总大小和配置以及可用在项目上的团队数量。需求的独立列表是由子系统做的，项目迭代进度表也可以由子系统来完成。一些任务（如设计数据库）可贯穿所有子系统，而且需要被独立安排或包含在每个子系统列表中。图 9-15 所示为 CSMS 销售子系统的一份样品的项目迭代进度表。正如你看到的那样，每个迭代的长度在四周之内是相当稳定的。所有被确定的且代表需求的任务已经被安排到迭代中了。在这种情况下，我们确定了五种迭代。

为一个简单的迭代创建一份详细的工作进度表包括三个步骤：

- 创建工作分解结构。
- 估计工作量和确定依赖关系。
- 使用甘特图创建进度表。

CSMS 销售子系统的项目迭代进度表		
迭代	估计时间	安排给迭代的用例
1	4 周	1. 查找商品 2. 查看具体的描述 3. 查看特效图（3D） 4. 比较商品特征
2	4 周	5. 查看评论和排名 6. 搜索朋友的评论和排名 7. 查看配套商品（图片） 8. 保存商品 + 配套商品组成“套餐”
3	5 周	9. 添加商品（或套餐）到购物车 10. 从购物车中移除商品（或套餐） 11. 添加商品（或套餐）到备用车 12. 从备用车中移除商品（或套餐）
4	4 周	13. 检查购物车动态信息 14. 创建和处理商店销售 15. 创建和处理电话销售
5	3 周	16. 完成、最终测试、强化站点、优化数据库等
总时间	20 周	

图 9-15　CSMS 销售子系统的项目迭代进度表

工作分解结构（WBS）是所有项目需要的独立活动和任务的列表。创建 WBS 有两种常用方法：通过可交付成果或通过时间轴。第一种方法确定了在一个给定的迭代中需要完成的所有可交付成果。然后，WBS 会确定每个任务，这对创建每个可交付成果是必要的。第二种方法是通过正常顺序的活动进行工作的，这些活动是最终可交付成果所需要的。在敏捷项目中工作过的有经验的开发者能理解需要创建一个详细的可交付成果的步骤和任务。当然，这取决于特定功能和包括的可交付成果，每次迭代都会有细微的不同。

图 9-16 是销售子系统第一次迭代的 WBS 示意图。根据核心过程计划、分析、设计和建立，任务会被划分开来。在图中，每个任务也会有一个对需要时间的评估。有时，会提供两种评估：需要的工作量及预期持续时间。所需的工作量会以工作日给出，而持续时间是对失效日历时间的衡量。当然，这些都取决于在具体任务上工作的人数。图 9-16 中只显示了持续时间，然而，估算的时间是假定了项目团队有四个人。

在创建 WBS 时，新分析员通常会问：“独立的任务需要多详细？”几个指导原理能帮助回答这个问题：

- 当任务完成时要有一个能辨认的方式。
- 任务的目的应该是很清晰的，这样才能评估所需努力的程度。
- 作为软件开发的通用规则，工作量应对应 1 – 5 天的工作时间。

为一次简单的迭代开发一份详细工作进度表的第二步是区分每个任务和所需工作量之间的相关性。与任务有关的最常用的方式是考虑它们的完成顺序，即就是：当一个任务完成时，下一个任务就开始。这称为结束 – 开始关系。与任务相关的其他方式包括开始 – 开始关系，这个关系是指任务必须同时开始；还有完成 – 完成关系，这个关系是指任务必须同时完成。所需的工作量应该是指完成这个任务所需的真正工作量。随着 WBS 中任务的确定，依赖关系和工作量的评估应该由将要真正做这项工作的开发者来完成。

销售子系统工作分解结构的第一次迭代

Ⅰ．项目计划

创建 UBS，建立进度表，然后计划工作——1/2 天

Ⅱ．分析任务

会见销售部——1 天

会见市场部——1 天

定义需求信息和数据元素（与 Cougar 团队共享）——1 天

建立用户活动模型——1 天

Ⅲ．设计任务

设计数据库模式（与 Cougar 一起工作）——1 天

设计屏幕布局和交叉步骤——2 天

确定程序类和方法——1 天

Ⅳ．建立任务

建立数据库（与 Cougar 团队合作）——1/2 天

编写程序代码——4 天

整合 3D 图像的代码——2 天

建立测试数据——2 天

建立用户"模拟现场环境"——1/2 天

执行用户模型度测试——2 天

发布模型版本——1/2 天

执行团队反思——1/2 天

注意：

在本次迭代中的用例需要数据和数据库模式，其中数据库模式在建立目录中也是需要的。建立目录要在市场营销子系统的第个迭代中完成。这两个迭代是并行完成的，而且两个团队（Tiger 团队和 Cougar 团队）合作且共享信息。

这是个四人团队。工作部分是：（1）所有人一起进行交叉调查和设计任务；（2）编程和测试任务是分组进行的。

在本次迭代中会开发四个用例：

1. 搜索商品。
2. 查看商品的具体信息。
3. 查看商品的 3D 图。
4. 比较多个商品的特征和价格。

图 9-16　第一次迭代的工作分解结构

开发详细工作进度表的第三步是创建迭代进度表。在图 1-7 中，我们提供了一张首次迭代的任务图，这张图包含贸易展览系统，它们的顺序及估算完成任务的时间。这张图实际上就是一张简化了的 PERT/CER 图。呈现进度表的其他形式是条形图，它显示的活动是条状的而且是在一条水平的时间线上，这被称为**甘特图**。创建甘特图的一个常用工具是 Microsoft Project。新版 MS Project 是能连接网络的，而且还提供了一个强大的工具，不仅能创建进度表，还能通过使用 HTML 协议分离组织中的进度表信息，这样才能在浏览器中看到。使用 MS Project 等工具的好处是项目经理能很容易地更新进程并且能使信息被广泛利用。

图 9-17 所示为将 RMO 的 CSMS 项目中的迭代进度通过表格所形成的一幅条状图。在图中，任务分解结构中的任务被列在了任务名那一栏，而持续时间则被列在了持续时间那一栏。前置栏确定了任务之间的依赖关系。就像你看到的那样，除了第一个任务，每个任务都

至少有一个前置任务，而且除了最后一个任务，其他每个任务都是一个或其他更多任务的前置。记录依赖关系有很多种方式。最常见的方式是在另外一个任务开始之前显示正在发生任务的完成（FS）。另一种常见的方式是结束－结束（FF），这种方式中的任务必须是同时结束的；还有开始－开始（SS），这种方式中的任务必须同时开始。任何依赖关系都会有滞后时间，如图 9-17 中显示的第 11 行。最后一栏记录了要被安排到每项任务中的资源。在这个例子中，Tiger 团队被分成两组，每组有两个人，即 TT1 和 TT2。

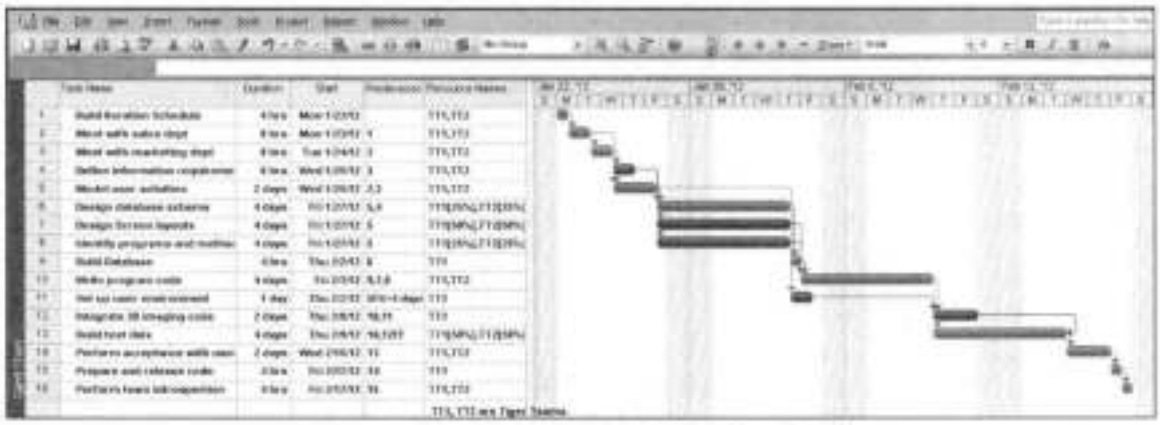


图 9-17　购物车子系统迭代时间表的第一次迭代

图 9-17 中的条形说明了附加了日历的每个任务的持续时间。浅色条是进度表中的关键路径。**关键路径**是指那些必须在进度表上停留的任务。如果任何一个关键路径任务能引起延期，那么整个项目都会延迟。深色条是那些不在关键路径上的任务。很显然，项目经理会相当密切地监控关键路径任务。

9.4.3　员工与资源分配

在敏捷项目中，各种团队都是自己组织的。他们自己决定要怎样一起工作及怎样安排。然而，确定项目需要的专业和把那些人安排到项目中的工作都是由项目经理处理的。这包括找到有拥有合适技能的人才，然后在整个项目中组织和管理他们。员工活动由五个任务组成：

- 创建项目的资源计划。
- 确定和要求具体的技术员工。
- 确定和要求具体的用户员工。
- 组织项目团队到工作组中。
- 进行初步培训和团队建立练习。

以在项目进度表中被确定的任务为基础，项目经理能创建一份详细的资源计划。事实上，进度表和资源需求通常会同时创建。在创建计划时，项目经理会意识到：（1）所需资源通常无法立即得到利用；（2）一个人要熟识项目是需要一段时间的。在创建完计划后，项目经理就能确定具体的人和要求，然后确认团队的成员。

在小型项目中，可能所有项目团队成员都在一起工作。然而，一个有多于四个或五个成员的项目团队通常会被分成小的工作小组。每个小组会有一个小组领导，他会协调任务并将其安排到小组中。项目经理则负责分组并且安排小组领导。

最后，要进行培训和团队建立的练习。当类似于新数据库或新编程语言这样的新技术被使用时，项目培训会作为一个整体来完成。其他情况下，不熟悉使用的工具和技术的团队成员可能会需要单独的培训。团队需要为技术人员和用户进行合适的培训。若成员们以前没有在一起工作过，则团队建立的练习是非常重要的。用户和技术人员的集成在创建有效团队和工作小组时是一个重要的考虑因素。

9.4.4 评估工作过程

尽管评价项目团队的工作执行得如何这一工作有时会在预测项目中做，但它还不是一个常用的实践方式。然而在迭代项目中，许多公司要求在“迭代的最后”审查团队一起执行工作的情况。迭代项目的一个优势就是同一团队通常会为了一系列的迭代而一起工作。在每次迭代过后，团队成员能评价他们在一起工作的情况如何，以及他们要怎样才能提升作为一个团队的效率和执行状况。在敏捷项目中，这被称为**回顾展**。下面是团队可能想问的几种问题：

- 我们的通信程序是合适的吗？要如何提升它们？
- 我们和用户的工作关系是有效的吗？
- 我们达到目标了吗？为什么或为什么不？
- 什么事情是进行得尤其顺利的？我们怎么确保它能保持下去？
- 障碍或问题的区域是什么？我们要怎么消除它们？

9.4.5 监控过程与改正错误

理论上，执行和控制项目计划听起来很容易，但事实上是相当复杂的。为了能执行任意一个项目，你需要制定不同的项目计划。一个团队建立与执行项目计划的不同取决于项目结构是以预测方法还是自适应方法为基础的。在预测方法中，项目计划是非常大型且复杂的。自适应方法则没有那么可怕，因为每个迭代都会有一个详细的项目计划。由于工作的每一部分都是比较小的，而且通常得到了很好的理解，因此这些计划倾向于变得更小且更不复杂。

图 9-18 是高层次的流程图，它说明了监控与控制项目的基础过程。第一个盒子——安排工作给个人或团队——代表的是本身就很复杂的任务，因为事实是团队是由有着各种技术层次和经验的人组成的。

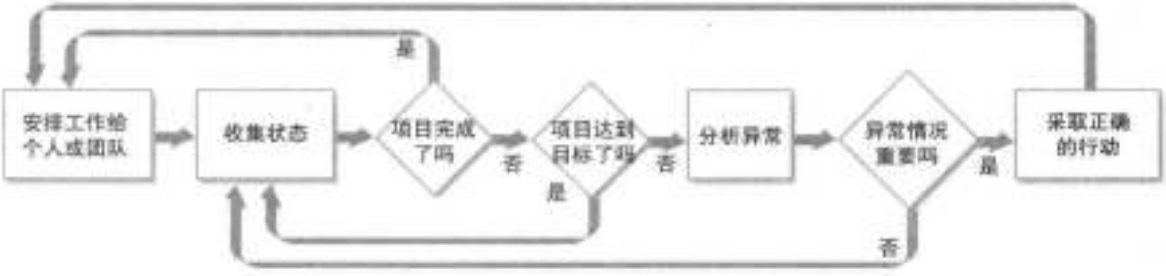


图 9-18 监控和控制项目执行的过程

第二个盒子的任务——收集状态——是不复杂的。在收集状态信息时，你应该坚持正确

的指导。首先，提供状态信息对所有团队成员来说应该都是一个标准过程。其次，应该收集且公布的信息是所有人都能看到的。状态信息是能在里程碑中被完整或不完整报告的。

第三个盒子的任务——分析异常情况——需要项目经理尝试发现为什么项目没有达到目标及延迟对总项目的影响到底有多大。

第四个盒子的任务——采取正确的行动——可以是很复杂的。有经验的项目经理会准备好一整套他们需要使用的工具来试着纠正错误。有时，改正就像重新安排团队成员那样简单，或者可能仅仅需要加班几个小时就能完成。其他情况下，可能要重新安排任务。在更严峻的状况下，整个进度表可能都要重做，或者还需要为团队招募更多成员。改正措施的目的是让项目回到熟知且能被预测的进度中。

每个开发项目——不管它是用预测方法还是自适应方法——都会有许多需要回答的问题和需要做的决定。在很多情况下，这些问题都会被快速解决，而且项目会进行得很快。然而，在其他情况下，一个问题的回答或者一个开放式问题的解决方案需要额外的调查。例如，销售委员会的一系列规则可能包括：委员会是何时、如何计算的，当产生退货情况时委员会会发生什么，什么时候付款给委员会，委员会是怎样安排各种活动来促进高利润商品及普通商品的销售的，等等。如果管理层还在考虑如何制定这些规则，那么你需要跟踪这些问题直到它们得到解决。

项目中开放式问题和风险的监控和控制通常不会比建立各种各样的跟踪日志更复杂。这些日志能用简单的电子表格来创建，并且可以上传到项目的网站或中心资源库上。让所有团队成员都能利用这些日志是一个很好的想法。图 9-19 所示为跟踪日志的一个例子。列标题会根据你使用的日志类型而变化。图 9-19 显示了需要在确定日期内被解决的问题以及负责解决这些问题的人。

	A	B	C	D	E	F	G	H	I
1	问题日志	问题时间	问题描述	优先级	问题影响	负责人	处理完成的目标时间	再次解决的描述	实际处理完成时间
2		1/18/2012	促销委员会的结构未被定义	紧急	数据库结构需要修改	William Henry	2/1/2102		
3									
4									
5									

图 9-19　一个简单的问题跟踪日志

本章小结

本章的主要内容集中在与计划和管理系统开发项目相关的原理和活动上。它包括三个主题：（1）项目管理的原理；（2）为了能使项目启动并且获取批准所进行的活动；（3）计划项目及监控其进程的活动。

项目管理是组织与指导其他人取得一个计划好的成果。从历史的角度来看，软件项目不会有一个完整的历史记录。强有力的项目管理被看作提升软件开发项目成功率的一个因素。其他因素（如 SDLC 的自适应方法）也有利于项目的成功。

本章首先讨论了许多和项目管理有关的重要技能、技术和概念。项目管理知识体系（PMBOK）为学习项目管理提供了大量的理论基础。敏捷项目管理需要像 PMBOK 这样的基础概念和技能，尽管有许多特定的技术会很难。

本章接下来集中于核心过程 1 的特定活动。这样做的目的是确定业务需求和让项目启动。这些活动包括：

- 确定问题。
- 量化项目批准因素。
- 评估风险和可行性分析。
- 与客户一起评审并获得批准。

本章最后集中于那些能计划项目、安排项目进度且开启项目的活动。这些活动包括：

- 建立项目环境。
- 安排工作进度。
- 员工与资源分配。
- 评估工作过程。
- 监控过程与改正错误。

复习题

1. 列出项目失败的六个主要原因。
2. 列出有利于项目成功的六个关键因素。
3. 定义项目管理。
4. 列出项目经理的五项内部职责。
5. 客户与用户之间的区别是什么？
6. 一种有机的方法是什么意思？
7. “仪式”的重要性是什么？
8. 列出 PMBOK 的九个领域。
9. 敏捷项目管理是指什么？
10. 范围管理是如何完成敏捷项目开发管理的？
11. 核心过程 1 的四个活动是什么？
12. 项目被启动的三个原因是什么？
13. 系统能力与商业收益的区别是什么？
14. 当批准一个项目时通常要考虑什么因素？
15. 列出在批准项目时需要考虑的 10 种收益。
16. 解释净现值（NPV）是怎么计算的。
17. 有形收益和无形收益的区别是什么？
18. 在估计组织灵活性时要考虑哪些因素？
19. 核心过程 2 有哪五个活动？
20. 列出在项目中要获取的七种类型的信息。
21. 项目迭代进度表和具体的工作进度表的区别是什么？
22. 工作分解结构用于做什么？
23. 迭代审查和追溯的收益是什么？

问题和练习

1. 阅读以下描述，然后制定一张公司可能从新系统中获得的预期商业收益的列表。

Especially for You Jewelers 是大学城的一家小珠宝公司。在过去的两年里，它的业务有了极大的发展。然而，它的财政业绩却与发展不同步。现在的系统是部分手动、部分自动，不能有效地跟踪账户收据，而且公司也很难确定为什么它的成本那么高。此外，它频繁地实行特价来吸引顾客，但却不知道特价是否有利可图或者这份收益是否带来了其他相关销售。Especially for You 想在现有系统中增加重复交易，因此需要开发一个顾客数据库。它也想安装一个直接销售和财务处理系统来帮助解决这些问题。

2. 阅读以下描述，然后为这家公司制定一张系统能力的列表。

Especially for You Jewelers 新的直接销售和财务处理系统是这家珠宝公司未来发展和成功的重要因素。直接销售部分需要跟进每次销售过程，并且能够连接到库存系统以提供每天的利润和亏损报表。顾客数据库需要提供购买历史记录，帮助管理人员为现有顾客准备特别邮购和特别销售。每个顾客的详细贷方余额和过期账户会帮助解决应收账款高度平衡的问题。特别注意许可证和信贷历史报表将帮助管理人员减少应收账款。

3. 以问题 1 和问题 2 为基础，为 Especially for You Jewelers 创建一份系统可视化文档。

4. 以下面叙述的内容为基础，创建工作分解结构（WBS）。它应该涵盖各方面的步骤——从项目的开始到结束，即从所有雇员搬进他们的新办公室开始。用以下栏标题来设计列表的解决方法：任务 ID、任务描述、估计工作量、前置任务 ID。对于你的解决方案，要遵循以下准则：

- 包含依赖关系。
- 包含工作量估计。
- 有 30 ~ 40 个详细任务。
- 涵盖 2 ~ 6 周的时间。

你是一家小公司的雇员，其设施已不再适用。这是一个 Web 开发和托管公司，因此你要有技术网络管理员、开发者和几个处理市场营销和销售的人。应该需要 10 个员工。

公司的董事长已经购买了一幢附近的单层建筑，而且公司正要搬进去。这个建筑需要一些内部改造来使它适用我们的工作。董事长要求你负责此次搬迁。你的安排是：（1）让大楼做好准备；（2）安排搬迁；（3）实施搬迁。

这幢大楼基本上已经完成了，因此剩下的工作不会太难（不需要任何建设——仅仅是一些翻新）。这幢大楼有几个办公室以及需要设置隔间的一个大区域。

你和董事长正步行穿过这幢大楼，同时她也在告告诉你她的想法。

她说："我们要使用原有的办公室，而且我们还需要为来参观的顾客准备一个接待处。后面角落里的办公室应该可以安置我们的计算机服务器。沿着东面的墙依次可以是销售人员的办公室。我们缺少一些办公室，所以要在大房间里为我们的初级开发人员安排几个隔间。当然，我们的电脑还需要能与系统相连接，而且我认为以太网会比无线网络更快速。同时我们所有人都要有电话。让我们计划一下在周四、周五和周六这样的长周末进行搬迁。当然，我们必须小心，不要关闭客户端，我们的主机要继续运行。你要为员工的搬迁建立一份进度表，同时还要创建一份指导，这样他们才知道要为搬迁做什么准备。谢谢。"

5. 用 MS Project 创建问题 4 中的 WBS。首先，创建人物、依赖关系和持续时间。然后使用 MS Project 写一段关于你的经历的感悟。

6. 创建一份六年净现值的电子数据表格，类似于图 9-10 中的表格。使用图 9-20 中所示的收益、成本和贴现因子的数据。系统的开发成本是 225 000 美元。

年　数	年收益	年运行成本	6% 的折扣因素
1	$55 000	$5000	0.9524
2	$60 000	$5000	0.9070
3	$70 000	$5500	0.8638
4	$75 000	$5500	0.8227
5	$80 000	$7000	0.7835
6	$80 000	$8000	0.7462

图 9-20　计算 NPV 的收益、成本和贴现因子

7. 使用 MS Project 创建以图 9-21 所示的表为基础的甘特图。输入任务、依赖关系和持续时间。打印出 PERT 图（网络图）和甘特图。

任务 ID	描述	持续时间（天）	前置任务
1	从国际交流办公室获取表格	1	无
2	填写并发送国外大学的申请表	3	1
3	接收国外大学的批准通知	21	2
4	申请奖学金	3	2
5	检查奖学金申请的注意事项	30	4
6	安排资金	5	3、5
7	为入住宿舍做相应的安排	25	6
8	获取护照和需要的签证	35	6
9	向大学发送预先登记表格	2	8
10	做旅行的准备	1	7、9
11	决定服装要求和购物	10	10
12	打包行李并且做好离开的最后安排	3	11
13	旅行	1	12
14	搬进宿舍	1	13
15	课程的最后注册和其他大学的书面工作	2	14
16	开始上课	1	15

图 9-21　去国外留学的 WBS 任务列表

图 9-21 所示为一份任务列表，是一个学生想要通过进入国外大学读书来获得国际经历的过程。假设所有前置任务必须在成功任务之前完成（最简单的版本）。而且，要插入几个概述任务，如应用程序任务、准备任务、旅行任务和到达任务。你要确定能声明你的假设。

8. 州立大学想要实施一个更好的系统来跟踪所有它本身具有且需要维护的计算机环境。大学购买了大量的计算机和软件，这些计算机和软件分布在校园各地并且被学院、职工、部门和大学所使用。现在，大学拥有很少的关于设备的记录，关于已经购买的维护或软件基本没有记录。用例的列表已经定义好，它会作为开发这个系统的开端。

用以下用例的列表来创建一份项目迭代进度表。你应该试着安排这些用例，这样相似

的内容便可以一起开发。而且，最重要的用例应该首先开发。陈述你的假设，然后解释你做出这一解决方案的原因。注意：为简洁起见，我们会使用单词“计算机”来指代计算机环境的任意一种类型，如台式机、笔记本电脑、服务器、打印机、监控器、投影仪、无线网络连接点等。

1）购买一台计算机。
2）销售一台计算机。
3）启动计算机相关服务。
4）停用计算机（过剩服务）。
5）为个人安排一台计算机。
6）记录一台计算机的地理位置。
7）修理一台计算机（内部）。
8）回收一台需要维修的计算机。
9）确定准备更换的计算机。
10）通过各种选项来搜索一台特定计算机。
11）购买软件许可证。
12）更新软件许可证。
13）在计算机上安装软件。
14）删除计算机上的软件。
15）记录计算机的使用证明。
16）购买计算机的使用证明。
17）通过各种选项搜索多台计算机。
18）通过各种选项搜索计算机上的软件。
19）为一个部门或一所大学安排计算机。

扩展资源

Scott W. Ambler, *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. John Wiley and Sons, 2002.

Charles G. Cobb, *Making Sense of Agile Project Management: Balancing Control and Agility*. John Wiley and Sons, 2011.

Jim Highsmith, *Agile Project Management: Creating Innovative Products*. John Wiley and Sons, 2009.

Gopal K. Kapur, *Project Management for Information, Technology, Business, and Certification*. Prentice-Hall, 2005.

Craig Larman and Bas Vodde, *Scaling Lean and Agine Development: Thinking and Organizational Tools for Large-Scale Scrum*. Addison-Wesley, 2009.

Jack R. Meredith and Samuel J. Mantel Jr., *Project Management: A Managerial Approach* (6th ed.). John Wiley and Sons, 2004.

Project Management Institute, *A Guide to the Project Management Body of Knowledge* (4th ed.). Project Management Institute, 2008.

Kathy Schwalbe, *Information Technology Project Management*, (6th ed.). Course Technology, 2009.

Robert K. Wysocki, *Effective Project Management: Traditional, Agile, Extreme*. John Wiley and Sons, 2009.

第五部分

Introduction to Systems Analysis and Design: An Agile, Iterative Approach, Sixth Edition

高级设计和部署概念

第 10 章　面向对象设计：设计原则

第 11 章　面向对象设计：用例实现

第 12 章　实现系统的可操作性

面向对象设计：设计原则

学习目标

阅读本章后，你应该具备的能力：

- 解释面向对象设计的目的和目标。
- 开发 UML 组件图。
- 开发设计类图。
- 使用 CRC 卡来定义类的职责和合作。
- 解释某些面向对象设计的基本原则。

开篇案例　NEW Capital Bank：第一部分

Bill Santora 是 New Capital Bank 的项目经理，负责开发一个集成客户账户系统，他刚刚会见过审查委员会的委员们，并与他们完成了新系统初步设计的技术审查。初步的设计关注四个核心用例，它们将作为系统最基础的部分在第一次开发迭代中实施。

New Capital Bank 使用面向对象技术已经有一段时间了，但开始采用新型的敏捷方法就比较晚。Bill 曾使用面向对象技术开发过一些系统，比如早期曾用过统一建模语言（UML）来开发系统。然而，这个开发项目是他第一次遇到的完全采用敏捷开发的大型项目。

当 Bill 正在收集演示材料时，他的上司 Mary Garcia 与他聊了起来。

她说："你的技术评审做得很好。委员们只提出了几处要修改的部分。虽然我现在没有完全明白这个新方法，但是我还是能理解这些核心功能是如何运作的。我还是很难相信你能用接下来的两个星期时间实施这四个过程。"

Bill 笑着说："等等，这个系统不是就这样给用户使用的。完成这四个核心功能的编码和运行并不意味着项目的结束，这个项目还需要一年时间来完成。"

Mary 说："我知道了。但是两个月后我们可以做出点东西就很好。不仅我会对项目有信心，用户也更意看到事情有所进展。"

"没错。别忘了我开始提出这个基于迭代方法的规划时是多么艰难。为后期的迭代制定项目进度是比较困难的，所以我花了很多时间让每个人都相信这个项目的风险不是很大。因为每次迭代只有四个星期的时间，所以在开始阶段就要展示一些东西。你不知道，设计通过评审后，我的压力减轻了很多。整个团队做了很多工作确保设计的可靠性，并且我们都觉得很有信心。" Bob 回答说。

Mary 说："很好，采用渐增式的方法很有意义，而且看上去很有用。我尤其喜欢你展示的图。你所展示的支持每个用例的三层结构化设计做得非常好。即使我自己还不太明白先进的面向对象技术，但是我还是能理解这个面向对象技术是如何与结构相适应的。我想当你证明用同样的基本设计既可以为我们的内部银行出纳员又可以为 Web 端的用户设计系统时，大家会拍手为你叫好的。祝你成功！"

Bill 欣然接受了 Mary 的祝贺。

他问："设计类图怎么样？您不觉得设计类图使得类和方法看上去更加明白吗？我们在团队中讨论的时候都是用它们进行交流的。它们确实能帮助程序员写出好的、可靠的代码。"

Mary 问："顺便问一下，你们安排再和用户进行复审了吗？"

Bill 回答说："还没有。结构化设计大多数是技术性的材料，并且我们还没完全准备好会见用户。用户会通过证实我们对信息可用性的理解来帮助我们，但是我们所做的大多数事情对他们来说很难理解。"

Mary 说："我很期待看见第一阶段的成果。在项目的后续开发过程中可以测试这些核心功能是很有意义的。让我再次祝贺你！"然后，他们一边说着一边一起去吃午饭了。

10.1　引言

我们在第 3、4、5 章中已经学习了如何通过开发功能需求模型来完成面向对象分析。分析是由两部分组成的：发现和理解。理解是提取从被采访用户那里得到的信息，并且构造相互关联且广泛的模型的过程。模型建立是理解用户需求及这些需求是如何影响目标系统的一个必要部分。然而要记住，分析模型的目标不是描述新系统，而是用正确的术语来理解需求。

本章和下一章集中于如何在需求模型的基础上开发面向对象设计模型，这些模型之后会被程序员用来编写系统代码。本章关注的是两个层次的设计（曾在第 6 章介绍过），分别是：结构化设计——通常把它称作高层设计；细节设计——每个用例的设计是特定的。本章开始的内容是在为新系统开发整个结构化架构时所需的模型和过程。要讨论的主要 UML 图是组件图，这种图是基于你在第 6 章学习过的关于开发环境的知识。

本章后面的部分中，你会开始学习细节设计的过程。我们会首先解释设计类图，这是问题域类图的扩展，增加了设计信息。接下来，我们会将解释类的 CRC 卡，作为教授以用例为中心和面向对象设计的细节的开端。

本章的结尾部分是对好的面向对象设计原则的重要讨论。通过本章和下一章，我们不仅会讨论面向对象设计的基础，也会讨论基本教学原则，这样你建立的系统就能被很好地构造和维护。设计原则会为正确的系统设计提供坚实的基础。

10.2　面向对象设计：分析与实施的桥梁

那么什么是面向对象设计呢？它是一个建立一系列面向对象设计模型的过程，程序员利用这些模型对系统进行编码和测试。系统设计是用户需求和新系统程序之间的桥梁。面向对象方法的一个优势是设计模型通常仅仅是需求模型的扩展。很显然，扩充现有的模型比创建全新的设计模型简单得多。然而，创建设计模型而不进行编程是很好的练习。建立者如果没有蓝图，是怎样也不会建造出比狗窝或小屋更大的东西的。同样，系统开发者如果没有设计模型是不会开发出大型系统的。

敏捷的自适应开发方法的一个原则是只有在模型是有意义且有必要时才会创建模型。有时，新手开发人员会误解这是告诉他们不需要开发设计模型。设计模型不会被正式归为综合性的文档和图表，但是它们是必需的。在没有设计的情况下开发系统相当于在没有大纲的情况下写调查报告。你可以做下然后就开始写，然而，如果你想要的报告是紧密结合的、完整的而且是综合性的，那么你首先要写一份大纲。在没有大纲的情况下试图写出一份复杂的报

告，那么结果多半是混乱的、难以理解和没有关键点的，并且分数会很低！大纲可以是草草地记在报告上，但是思考和写下来的过程使得写作者能确保它是紧密结合的。系统设计提供了同样类型的框架。

关于自适应方法的一个重点是需求、设计和编程会在迭代中并行完成。因此，一组完整的设计文档不会同时被开发。一个普通用例或几个用例的需求可能会先被开发，然后为那个用例开发设计文档。为了实时地跟进解决方案的设计，可以完成编程工作。一些人称之为"及时"系统设计。

10.2.1　面向对象程序概述

让我们快速回顾一下面向对象程序是如何工作的。然后，我们会讨论设计模型及其构造方式，以此来支持面向对象编程（OOP）。

面向对象程序由一组合作完成一个结果的程序对象组成。每个程序对象都有程序逻辑和一些必要的属性，这些逻辑和属性都被封装在一个单元中。这些对象之间通过互相传递消息等共同工作的方式来支持主要程序的功能。

图 10-1 描述了面向对象程序是如何工作的。程序包括一个窗口对象，该窗口显示了输入学生 ID 和其他信息的表格。输入学生 ID 之后，窗口对象会发送一个消息（2 号）给学生类，告诉它在程序中创建一个新学生对象（实例），然后去数据库获取学生信息并且将它放入对象中（消息 3）。接下来，学生对象会发送信息给窗口对象，从而在屏幕上显示该信息。然后这个学生再输入更新数据或个人信息（消息 4），并且另一序列的消息会被发送到程序中来更新学生对象及数据库中的学生信息。

图 10-1　面向对象事件驱动程序流

面向对象系统由一系列计算对象组成。每个对象都封装了它自身的数据和逻辑程序。分析员通过定义一个类来定义程序逻辑的结构和数据字段。只有当程序开始执行时，对象才能存在。这个过程我们称之为类的**实例化**——也就是基于类定义所提供的模板生成的对象实例。

图 10-1 所示为简单的程序执行过程中的三个对象。每个对象也代表了三层结构的一个架构。这三个对象不需要存在于同样的机器中。事实上，在一个多层结构中，对象的三个类一般会存在于三个不同的机器中。在第 6 章中你已经学习了关于多层结构的知识。

10.2.2　面向对象设计模型和过程

面向对象设计的目标是确定和说明那些必须要一起工作来实施每个用例的所有对象。正如图 10-1 所示，这些对象包括用户界面对象、问题域对象和数据访问对象。除了简单地确

定对象，你还需要说明对象中的细节方法和属性，这样程序员就可以理解对象是如何合作来执行用例的。

图 10-2 说明了能直接用于开发相应的设计模型的需求模型。左边一栏的需求模型是在分析阶段开发的，而右边一栏的设计模型是我们在设计过程中开发的。你可以从指向它们的箭头数量推断出，交互图是用在细节设计中的核心图。它们可以是顺序图（图中所示的），也可以是协作图。

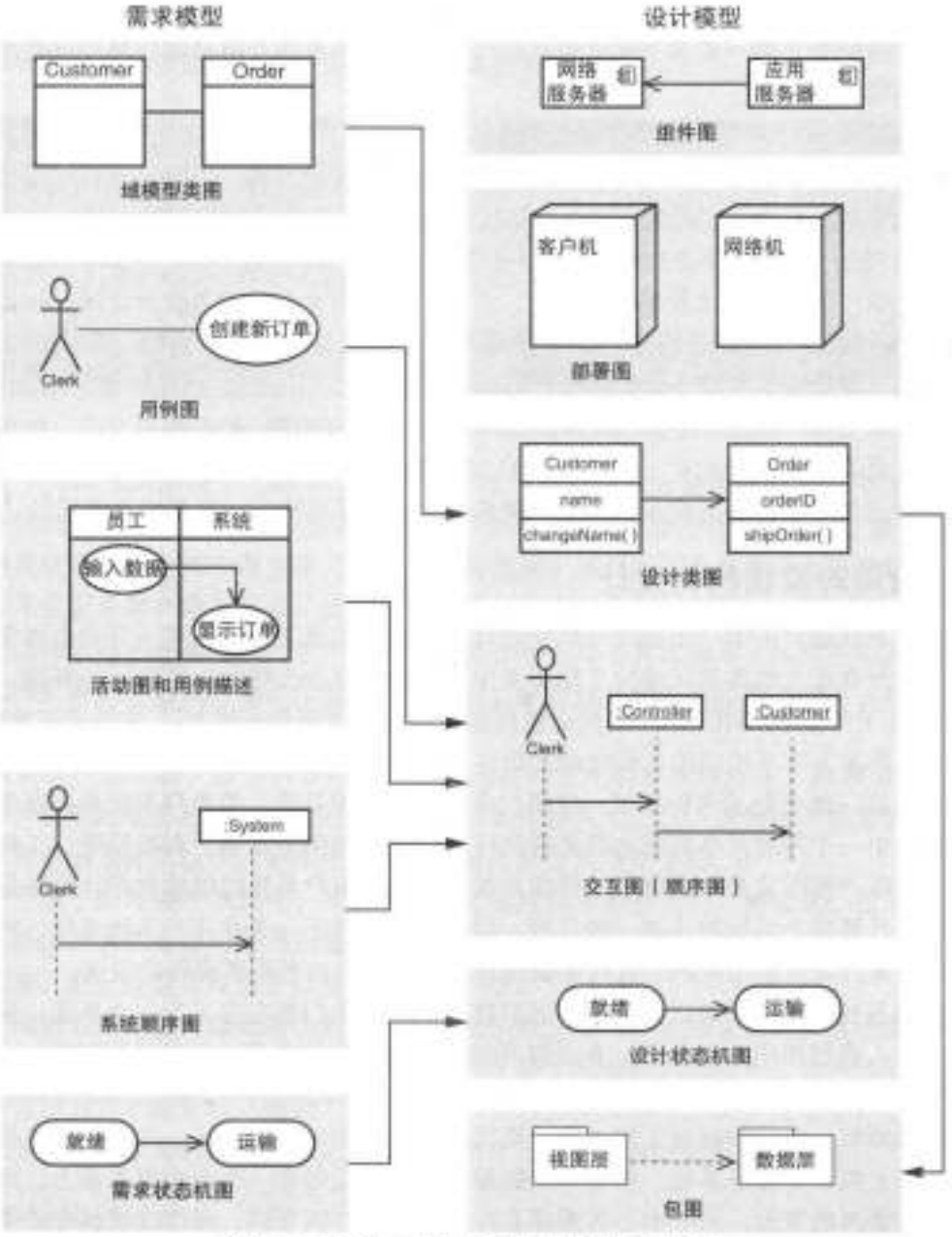


图 10-2　设计模型及其代表性的输入需求模型

此时，你应该很熟悉需求了，但是让我们花一分钟回顾一下相关知识。域模型类图确定了所有的类或“事物”，这些在问题域中都是很重要的。用例图确定了系统需要支持的基本业务流程——换句话说，也就是用户想要使用系统来实施过程目标的所有方式。活动图和

用例描述记录了每个用例的内部工作流程。活动图显示了实施一个普通用例的必要步骤。系统顺序图和活动图密切相关，它还会显示在用例步骤中用户和系统之间来回传递的消息或数据。最后，状态机图跟踪了一个特别类的所有状态环境需求。它们也显示了控制一个状态到另一个状态的变化的业务规则。

图 10-2 的右侧所示为设计模型。结构化设计是系统设计过程中的早期步骤之一，因为它能提供新系统的全景和整体结构。右边一栏的顶部是组件图和部署图。你在第 6 章已经学过了一点关于部署环境的知识，但是没有使用官方 UML 符号。本章会解释如何绘制 UML 组件图。你也能学习到一点关于多层软件设计的知识。本章还会解释如何使用这些图来记录软件系统的结构化设计。

下面关注图 10-2 右边的下半部分，我们会看到设计类图（DCD），这是域建模类图的扩展。它们用于记录软件类的设计元素，你会在本章的后面学习到。回想一下在域建模类图中的类不是软件类，但是在 DCD 中的类就是软件类。域类是设计软件类的基础。接下来是交互图，它可以是 UML 顺序图也可以是 UML 协作图。在第 5 章，你已经学习了如何建立系统顺序图。顺序图的设计版本是更详细的，且会用于实现大多数细节设计活动。你将会在第 11 章学习如何创建交互图。程序员也会使用状态机图来开发细节类方法，你在第 5 章已经学习到这些。设计版本与在分析阶段使用的版本很相似。

最后还有 UML 包图，这是一种将所有设计元素都组织在一起的简单方式。包图可以被用来组织任意类型的设计元素，但是在第 11 章我们会使用它们来组织设计类。

我们通过思考新解决方案系统的整体架构——也就是结构——来开始系统设计。

10.3 面向对象结构化设计

通常，系统设计的第一步就是结构化设计。大部分情况下，开发者一开始会思考如何开发系统，以及在需求收集和记录的早期步骤中整体结构是什么样的。在项目的开端，我们说"这是一个基于 Web 的系统"或"这只会被用在我们的网络和台式机上"，这是很正常的情况。那些评论是解决方案系统结构化设计的开始。

软件系统一般会分成两种类型：单用户系统和企业级系统。**单用户系统**是用在单独的台式机上或是由一个没有共享资源的服务器执行的。典型的例子是电子表格程序、工程画图程序、简单的账户程序或是一个电子邮件客户端程序。单用户系统的结构化设计通常是很简单的，在单个计算机上运行的系统一般只有一层。然而，对于一个单用户系统来说，将系统作为多层程序来开发是很明智的，这样做能使各个层之间的边界得到很好的定义。

企业级系统这个术语可以是许多不同事物的意思。我们将它定义为一个系统，这个系统可以在多个人或组织中共享组件。企业级系统总会使用多层的客户端服务器结构。你在第 6 章已经学习了关于客户端服务器和三层结构的知识。这种结构的一个典型例子是内部网络客户端服务器环境，在这种环境下客户端计算机包含视图层和域层程序，并且数据连接层是在中心服务器上的。企业级系统、数据库和数据连接的特征是在一个中心服务器上，因为它在组织中是可共享的资源。由于中心数据库是在企业中可以共享的，所以它被放在一个中心服务器上，这个服务器是所有应用程序的用户都可共享的。因为本地客户端计算机通常是强大的工作站，所以视图层和域逻辑在本地便可执行。

企业级系统的定义是一个宽泛的概念。系统的两个主要类别要符合关于系统设计的这个定义：（1）基于网络的客户端服务器，（2）基于因特网的系统。

实施企业级系统的这两种方法有许多相似的属性。两者都需要网络，都有中心服务器，以及在客户端设备上都有视图层。然而，一些基本的不同点也存在于这两种方法的设计和实施阶段。主要的区别在于视图层是如何与域层和数据连接层交互的。作为开发者，我们必须能够区别这两种系统，因为我们必须考虑重要的设计问题。

图 10-3 确定了影响系统结构化设计的三个基本区别：状态、客户端部署和服务器部署。状态的概念与客户端视图层和服务器域层之间连接的永久性有关。如果这个连接是永久性的，比如在一个客户端 / 服务器系统中，则变量值可以被来回传送，并且系统中的每个组件都可以被记住。视图层能直接连接到域层中的数据字段。

设计问题	客户端 / 服务器网络系统	因特网系统
状态	“状态性”或基于状态的系统——例如，客户端 / 服务器连接是长期的	无状态的系统——例如，客户端 / 服务器连接不是长期的，并且没有内在记忆
客户端部署	被编程的屏幕和表格都能直接显示。域层通常是在客户端上或是处于客户端和服务器的分界处	屏幕和表格只能通过浏览器显示。它们必须遵循浏览器技术
服务器部署	应用程序或数据服务器能直接连接到客户端层	客户端层不能通过网页服务器直接连接到应用程序服务器

图 10-3　客户端 / 服务器系统和因特网系统之间的区别

在一个无状态的系统中（如因特网），客户端视图层与服务器域层没有一个永久性的连接。因此因特网被设计成当一个客户端经过一个输入在浏览器中的 URL 地址请求屏幕时，服务器会发送合适的文档和两次断开连接。换句话说，客户端不会知道服务器的状态，并且服务器也不会记住客户端的状态。这种过渡连接使得实现购物车中的销售这样的功能变得困难。为了增加无状态环境的永久性，网页设计者已经开发了其他技术，如信息记录程序、会话变量和可扩展语言数据传输。作为系统设计者，在设计互联网企业级系统时，你必须考虑这些额外的组件。

考虑到客户端部署，基于网络的系统客户端包括视图层类和域层类。屏幕中的格式化、显示和事件处理过程都能直接通过视图层和域层程序逻辑来控制。这些电子屏幕的设计和编程过程都具有很大的灵活性。视图层类和域层类可以直接相互交流。

在一个基于因特网的系统中，所有信息都是通过浏览器显示的。格式化、显示和事件处理过程都必须符合正在使用的浏览器的性能。特殊的技术和工具，如脚本语言、小型应用程序和样式表，都已经被开发出来以提高网络的性能。

基于网络系统的服务器部署由数据连接层类组成，有时还包括域层类。这些类通过相互之间公共方法的直接通信和连接进行合作。在一个基于因特网的系统中，客户端层中所有的通信必须通过 HTTP 服务器。通信不是直接的，而且方法和程序逻辑不能直接通过传输的参数调用。连接域层逻辑的间接技术更加复杂，而且在设计系统时需要额外关注。

组件图和结构化设计

组件图确定了逻辑、可再用性和便携式的系统组件，这些组件定义了系统结构。组件图的必要元素是带有接口的组件元素。

组件是指一个可执行的模块或程序，它由编译成的一个简单实体的所有类组成。它有定义完好的接口或是公共方法，可以通过其他程序或外部设备连接。所有这些能连接到外部世

界的公共方法组都被称为**应用程序接口**（API）。图 10-4 所示为组件及其接口的 UML 符号。在一个简单的组件中没有必要列出所有的接口，只有那些和图相关的接口才会被列出来。表示组件的方式有两种：作为一般类或一个特定实例。应用在这种情况下的相同规则也能被应用在类和对象符号中：一般类使用组件类的名称，并且实例名称会加下划线，组件名会写在图形里面。

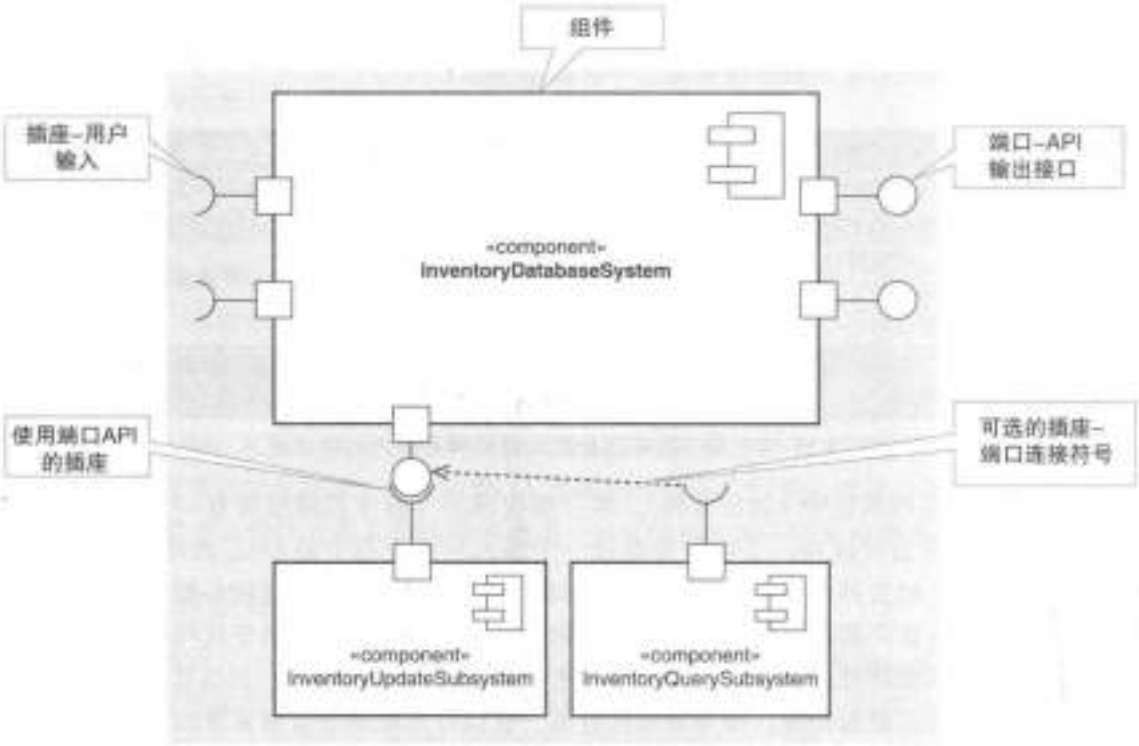


图 10-4　UML 组件图的表示

图 10-4 中顶部的矩形是代表组件的符号，包括它的接口。右上角的图标是带有两个从左边延伸过来的塞子的小矩形，代表这是可移动的、可执行的组件，而且是可重复使用且可插入的。

这个图也显示了两种类型的接口：一种是输出端口，一种是输入插座。输出端口与编程接口很相似，它定义了可以被其他组件用来连接组件功能的方法名（如 API 的一部分）。输入插座代表组件从其他组件需要的服务。要注意球状和插座符号，它们连在一起，这样一个组件的输入能精确地符合另一个组件的输出。

图的底部显示了端口接口和插座是如何被用在一个组件图中的。顶部的库存数据库系统组件显示了连接到世界的接口，正如通过端口 API 接口和它底部边缘显示的那样。库存更新子系统组件使用的接口能连接到库存数据库系统的方法。在这张图中，我们通过一个虚线箭头显示库存查询子系统组件连接到同样的接口。

在设计例子中，我们会显示如何为一个基于 Web 的系统做多层设计，而且会说明各种网页的位置。换句话说，我们想要用一些符号来显示网页属于哪里以及在哪里被部署。由于 UML 符号对于窗口而言没有标准，所以我们会拓展符号来把它包括进来。UML 为构造一个符号和拓展语言制定了规则。图 10-5 所示为一个网页符号。

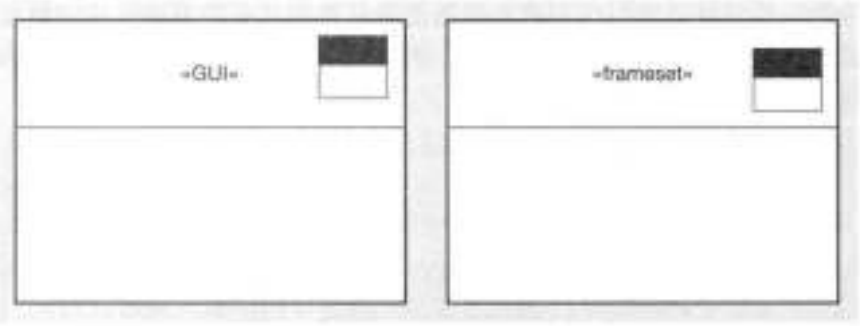


图 10-5　GUI 窗口和网页的扩展标记

图中很简单地显示了构造型的类符号，如 <<GUI>> 和 <<frameset>>，右上角还有一个小窗口图标。这个符号会作为台式机系统窗口或是网页系统框架集。框架集是高层次对象，它能让条目通过浏览器显示。我们会使用一个框架集符号和定型符号来说明一个网页。你可以将一个框架集看作能显示框架或一组框架的浏览器窗口。

接下来，我们会使用组件和窗口符号来做一些简单的因特网系统的结构化设计。

因特网系统的两层结构化设计

许多大学都有网页开发课程。大多数这些课程都会分成以下两类之一：网站设计或网站编程语言。它们对于你作为一个系统开发者的教育来说都是很重要的且很有益的。

网页编程课程教授你各种编程语言和在网页中插入程序流程的方式。你会学习 JavaScript、VBScript，PHP 和 ASP（活动服务器页面）。你可以学到如何使用先进的数据库工具（如 Cold Fusion）从你的页面连接到数据库。你也可以学习浏览器和服务器如何一起工作来服务页面，这个页面是有足够的程序流程来支持业务应用程序的。这个课程的深入版本甚至还可以教你 Java 或 .NET 环境，这样你就可以部署一个完整的应用程序。

我们没有打算用这短短的一部分内容来代替那门课程。相反，我们介绍的是这些基于 Web 的系统结构，并提供了一些你在其他课程中所能运用的指导原则。在第 6 章，我们解释了作为一个有效的、开发强健的、易于维护的系统的三层设计。但是，设计者如何在一个基于 Web 的结构中实施三层设计？如果一个组织想要为两种企业系统使用同样的问题域逻辑——客户端 / 服务器系统和基于因特网的系统，那么这个问题是非常重要的。

图 10-6 所示为一个简单的、通用的因特网结构。记住，我们在此刻做的是逻辑设计而不是关注物理计算机部署。在之后学习了更多关于部署图的知识后，我们将会讨论计算机部署的内容。当然，由于在两个组件之间有云计算机，所以我们可以很自然地假设它们是处于不同的物理位置中的。

图 10-6 包括 4 个可识别的组件。浏览器是一个可执行的组件，它的目的是格式化、显示和执行活动代码，如 JavaScript 或 ActiveX Control。因特网服务器是一个可执行的组件，它的目的是检索页面和调用其他组件。图 10-6 中的图还显示了另外两种组件的例子：CGI 的可执行程序和应用程序服务器。

为了简洁易懂，端口和插口在这个图中被省略了。这些组件之间的接口是行业标准，而且唯一的端口 / 插口不需要在这里强调。然而，每个双向箭头都代表两队端口 / 插口。

正如第 6 章中指出的那样，许多简单的业务系统可以作为两层系统来设计。这些系统主要从用户那里获取信息和更新数据库，不需要复杂的域层逻辑。在那些实例中，域层和数据

连接层通常是相互结合的。域层中的业务逻辑通常只与格式化数据以及决定更新哪个数据库表相关。许多业务应用程序都属于这种类型。例如，一个简单的通讯录系统可以很简单地设计成两层系统。

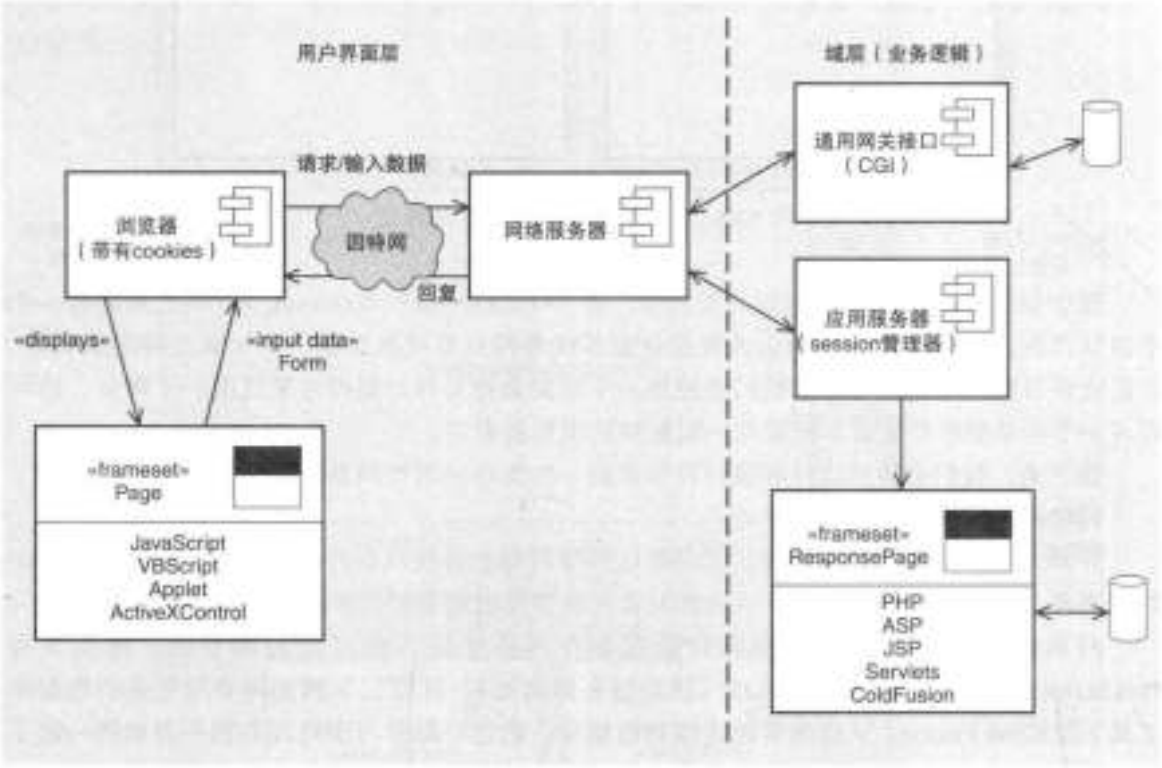


图 10-6 组件图显示的两层互联网结构

CGI 是处理输入数据的原始方式。CGI 目录包含被编译的程序，这些程序能从服务器中接收输入数据。CGI 目录中的程序可以用任意一种编译语言编写，如 C++。这种技术是有效的，而且通常能快速响应并快速处理。唯一的缺点是这些程序的编写是非常复杂且困难的。它们处理输入数据，连接任意所需的数据库，以及在 HTML 中格式化一个响应页面，正如图中响应页面所表明的那样。

输入数据的其他潜在提示是直接针对有着嵌入式程序代码的网页地址而言的。延伸到 ResponsePage 可显示嵌套在页面中程序代码，其类型包括 ASP、PHP、JSP、CFM 等。作为语言处理者的应用程序服务器会被调用，它将处理何种嵌套代码取决于扩展的类型。嵌套代码通过应用程序服务器处理代码，包括阅读和编写数据库。与浏览器的 cookies 一起工作的应用程序服务器可以管理用户对话。建立会话变量是为了在多个页面请求和产生时维护用户信息。应用程序服务器也能格式化基于 HTML 状态和代码的 ResponsePage，并且将它转发回因特网服务器。

尽管我们举例的是这种两层结构的设计，但用户接口类通常包含业务逻辑和数据连接。由于网页服务器的结构，处理输入表格的程序（被定义为面向对象类）也会输出 HTML 代码，而这些代码会被发送回客户端服务器。例如，在基于 Java 的系统上，Java 小服务程序接收在网页上输入的数据，处理数据和格式化输出 HTML 页面。为了处理数据，小服务程序通常包括任何所需的业务逻辑和数据处理逻辑。这与 .NET 环境是很相似的。对于每个网

页格式来说，都会有用 VB、C#、J# 或一些相似语言编写的一个后置代码。另一个常用的语言组合是 PHP，这是在服务器上执行的，还有 JavaScript，这是在浏览器上执行的。后置代码对象接收了网页中的数据，并格式化了输出 HTML 页面。因此，这个结构是一层的还是两层的就不是很清晰。然而，我们称这个结构为两层，是为了强调它是动态的以及 HTML 响应页面的建立是动态的。

这个结构对于不复杂的两层应用程序来说是很好用的，例如当响应页面的大多数 HTML 已经编写好时。嵌套代码执行像验证数据和存储进数据库这样的功能。要注意业务逻辑是最小的，所以将它和数据连接逻辑混合仍然能提供可维护的解决方案。

开发基于 Web 系统的几个集成化开发环境（IDE）是以两层或三层结构为基础的，这被称为模型视图控制器（MVC）。你会第 11 章学习更多关于设计模式和 MVC 的知识。最基本的概念是基于 Web 的系统可以在多个视图层中被开发，也就是用户接口、模型（业务逻辑和数据库访问）以及控制器（提供视图和模型之间的连接）。

正如第 2 章中所提到的，RMO 的 CSMS 系统需要支持一个网页用户接口和一个内部桌面接口。重要的是，业务逻辑和数据库访问的后端要连接用户接口。很显然，设计团队必须详细说明结构化设计，以确保程序员实施一个能支持用户接口的系统。

一旦结构化设计被决定，就是时候去下钻到一个抽象的底层层次。换句话说，你不能再把逻辑组件当作黑盒子，而应该开始看内部。每个组件都是可执行的程序，并且由类组成。所以，应用程序设计的下个步骤是为每个用例开始定义设计类的交互，在这个层次上的类通常被称为细节设计。

10.4　面向对象细节设计的基本原则

既然我们已经学习了结构化设计，那么现在我们可以开始学习细节设计了。回顾图 10-2，继续看右边那一栏，接下来要讨论的两个图是设计类图和交互图（顺序图和协作图）。这两种图表是细节设计中最重要的图。

面向对象细节设计的目标是确定和说明所有对象，这些对象必须要在一起工作来实现每个用例。正如图 10-1 所示，有用户接口对象、问题域对象和数据连接对象。你可以这样认为，细节设计的主要职责是确定和描述每层中的每组对象，以及确定和描述在这些对象之间被发送的交互情况或消息。

在面向对象设计中最重要的模型是顺序图，或者是它的“近亲”协作图。在第 5 章，你学习了开发系统顺序图（SSD）来为用例建立输入和输出需求的模型。完整系统顺序图是用于设计阶段的，并且是交互图的一种。协作图也是交互图的一种。在设计阶段，开发人员通过修改单独的 :System 对象来包括所有的交互用户接口、问题域和数据库访问对象。换句话说，它们透视了 :System 内部对象，以此来观察系统内部正在发生什么。我们在第 11 章会花费大量的时间学习如何开发这些详细的顺序图。图 10-7 所示为一个基于图 10-1 的简单顺序图，它更新了学生信息。顺序图和系统顺序图使用一样的符号，而系统顺序图你已经在第 5 章学习过了。

在本章中，你还将学到另外一种重要设计模型——设计类图。它的主要目的是记录和描述构建新系统需要的类。它描述的是类、属性名称、属性、方法名和属性之间的编程、导航所需的一组面向对象类。设计类图是利用详细顺序图得到的最终设计的一个总结，并且在编码过程中可以直接使用。图 10-8 所示为在分析阶段开发的初步域模型以及相应类的设计类

图。设计类图的底部有一个新组件，它说明了这个类的方法特征。同时，属性也增强了。我们将在下一节中介绍图中使用的各个符号的细节。细节设计是将域模型转换成设计类模型的过程。

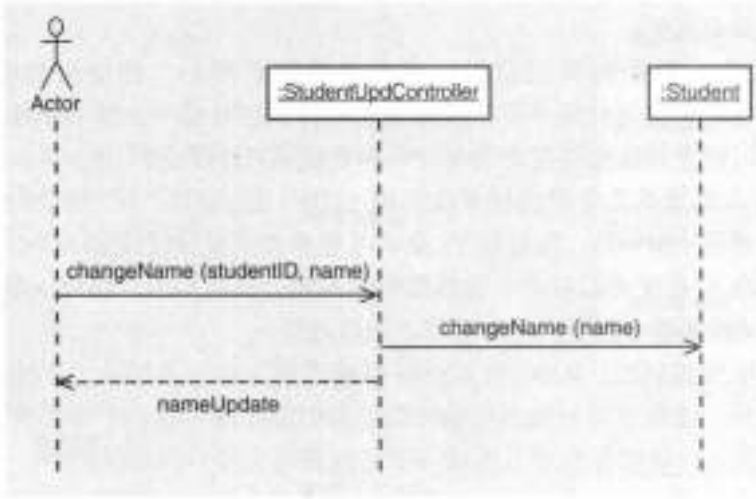


图 10-7　更新学生姓名的顺序图

作为一个面向对象系统的设计者，你必须提供足够的信息，这样程序员可以编写初步的类的定义，包括方法代码。例如，设计类说明有助于定义属性和方法。图 10-9a 所示是用 Java 语言实现的 Student 类的部分代码。图 10-9b 所示是 Visual Basic .NET 实现的部分代码。再次回顾图 10-8，我们可以从中看出设计类是如何为图 10-9 的代码提供输入的。注意，类名、属性和方法名来源于设计类的符号。

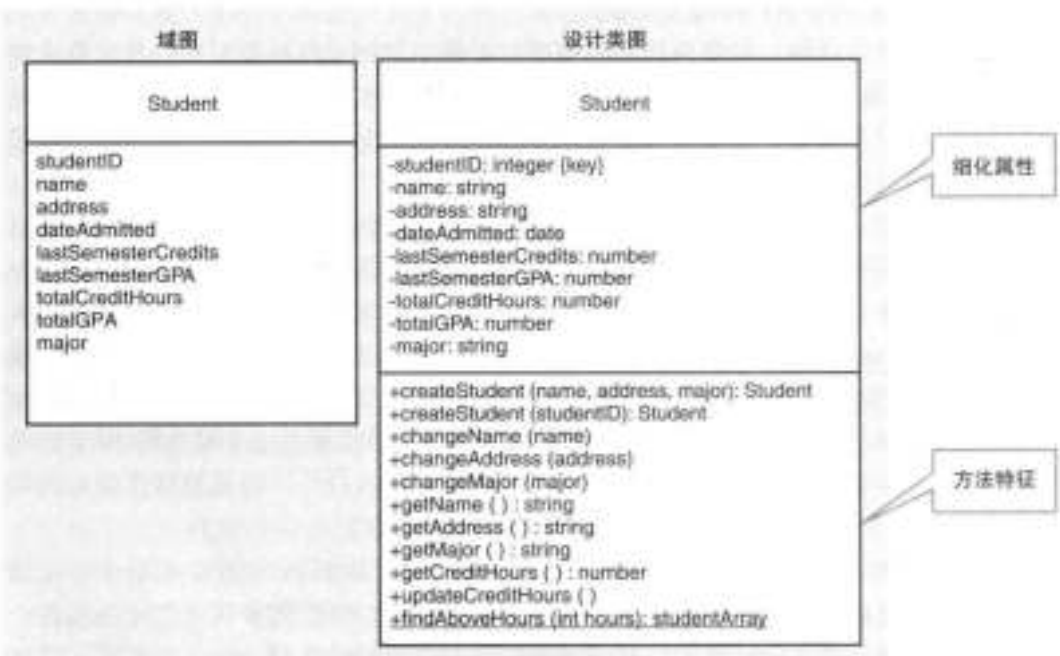


图 10-8　域类和设计类的例子——学生类

```
public class Student
{
        //attributes
        private int studentID;
        private String firstName;
        private String lastName;
        private String street;
        private String city;
        private String state;
        private String zipCode;
        private Date dateAdmitted;
        private float numberCredits;
        private String lastActiveSemester;
        private float lastActiveSemesterGPA;
        private float gradePointAverage;
        private String major;

        //constructors
        public Student (String inFirstName, String inLastName, String inStreet,
                String inCity, String inState, String inZip, Date inDate)
        {
                firstName = inFirstName;
                lastName = inLastName;
                ...
        }
        public Student (int inStudentID)
        {
                //read database to get values
        }

        //get and set methods
        public String getFullName ( )
        {
                return firstName + " " + lastName;
        }
        public void setFirstName (String inFirstName)
        {
                firstName = inFirstName;
        }
        public float getGPA ( )
        {
                return gradePointAverage;
        }
        //and so on

        //processing methods
        public void updateGPA ( )
        {
                //access course records and update lastActiveSemester and
                //to-date credits and GPA
        }
}
```

a) 用 Java 为学生类定义的例子

图 10-9

```
Public Class Student

    'attributes
    Private studentID As Integer
    Private firstName As String
    Private lastName As String
    Private street As String
    Private city As String
    Private state As String
    Private zipCode As String
    Private dateAdmitted As Date
    Private numberCredits As Single
    Private lastActiveSemester As String
    Private lastActiveSemesterGPA As Single
    Private gradePointAverage As Single
    Private major As String

    'constructor methods
    Public Sub New(ByVal inFirstName As String, ByVal inLastName As String,
          ByVal inStreet As String, ByVal inCity As String, ByVal inState As String,
          ByVal inZip As String, ByVal inDate As Date)
       firstName = inFirstName
       lastName = inLastName
        ...
    End Sub

    Public Sub New(ByVal inStudentID)
        'read database to get values
    End Sub

    'get and set accessor methods
    Public Function GetFullName() As String
        Dim info As String
        info = firstName & " " & lastName
        Return info
    End Function

    Public Property firstName()
        Get
           Return firstName
        End Get
        Set(ByVal Value)
        firstName = Value
        End Set
    End Property

    Public ReadOnly Property GPA()
        Get
          Return gradePointAverage
        End Get
    End Property

    'Processing Methods
    Public Function UpdateGPA()
       'read the database and update last semester
       'and to date credits and GPA
    End Function

End Class
```

b) 用 VB 为学生类定义的例子

图 10-9 （续）

面向对象设计过程

面向对象设计是模型驱动和用例驱动的。如图 10-2 所示，整个设计过程将需求模型作为输入，并生成设计模型作为输出。显然，我们需要一个过程来组织这项活动，并且围绕用例展开。换言之，我们通过用例来开发设计模型用例。例如，每个用例都相应开发了一个顺序图。一旦一组顺序图设计完成，整个用例组的设计类图也就相应完成了。我们可以将这个过程划分为 4 步，如图 10-10 所示，同时它也显示了章节中每个要讨论的步骤。

设计步骤	章节
1. 开发初步设计类图，显示导航可见性	10
2. 使用 CRC 卡为用例决定类的职责和合作	10
3. 为每个用例开发详细的顺序图 1）开发初步顺序图 2）开发多层顺序图	11
4. 通过使用 CRC 卡或顺序图，添加方法特征和导航信息，以此来更新设计类图	11
5. 将解决方案划分成适当的包	11

图 10-10　面向对象的细节设计步骤

首先，为设计类图创建一个初步的模型。该模型中必须包括一些基本信息，如属性名等，以便进一步开发顺序图。这一步是第二步和第三步的基础。

第二步通常是由开发人员为每个用例开发一组 CRC 卡。这些有助于为系统提供所需内部步骤的整体理解，以此来支持用例。CRC 卡为确定所有包含在特定用例之间的对象和职责提供了简单的方法。CRC 活动的成果被设置成很多卡，这些卡有助于开发顺序图，如果用例非常简单，那么这些卡片可以用来为用例编程。我们会在后面更详细地解释 CRC 卡。

10.5　设计类和设计类图

如图 10-2 所示，设计类图和详细顺序图需要协同工作。设计类图的第一次迭代是基于域模型和软件设计准则的。初步设计类图用于辅助开发顺序图。由于设计决策是在顺序图的开发过程中制定的，所以其结果又可以用来完善设计类图。

域建模类图揭示了问题域类和它们之间的联系。由于分析是一个发现的过程，所以分析员一般不大关心属性或方法的细节。然而，在面向对象编程中，类的属性必须被声明成 public 或 private，每一个属性必须定义类型，比如 character 或 numeric。在细节设计中，详细定义这些细节很重要，详细定义传递给方法的参数和方法的返回值也很重要。有的时候，开发人员还要定义每个方法的逻辑。我们通过集成来自顺序图和其他模型的信息来完成设计类图。

开发者建立设计类图时，还要在以前域模型的基础上增加很多类。回顾图 10-1，输入窗口对象和数据库访问对象就是很好的例子。系统中的类可以划分为几类，如用户接口类等。有时候，设计者也可以以子系统的方式来划分类。下面我们来介绍设计类图中使用的符号和在第一次迭代中用到的设计准则。

10.5.1　设计类符号

UML 没有说明设计类表示法和域模型表示法的区别。然而，二者是存在区别的，主要

是因为设计模型和域模型的目标不同。域模型展现的是用户工作环境下的事物以及它们之间的联系。类不是特指软件类。但是我们一旦开始创建设计类图，就要定义软件类。因为在设计过程中要定义很多不同类型的设计类，UML 使用了一种特殊的表示法，叫作构造型，它允许设计者为每一个类指明一个专门的类型。**构造型**将模型元素以特定的类型分类，通过说明我们想要强调的特征来扩展模型元素的基本定义。构造型的表示是将类型的名称放到类似书名号的符号中，像 <<control>> 这样。

有四种类型的设计类被看作标准的设计类：实体类、控制类、边界类和数据访问类。图 10-11 展示了这四种构造型的表示法。

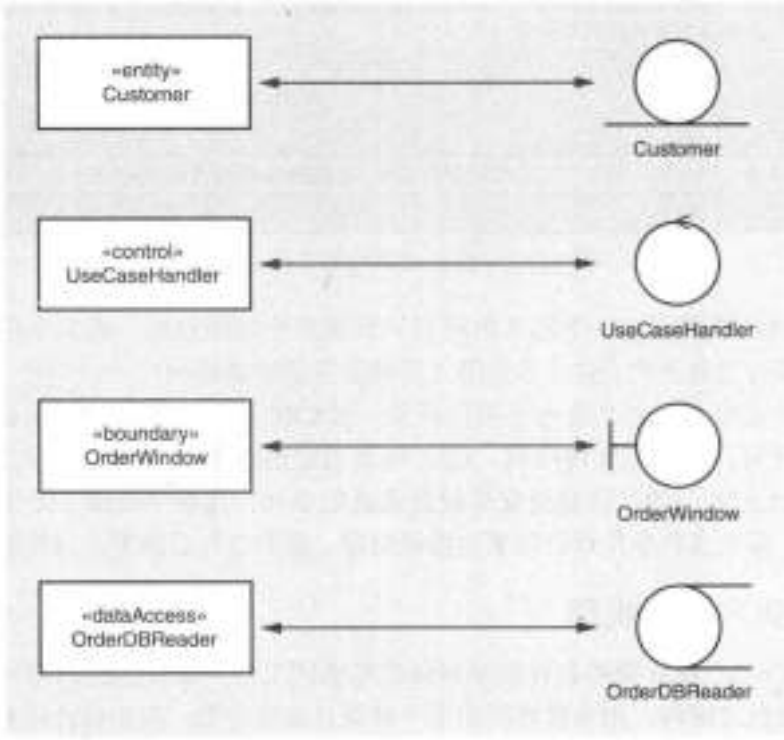


图 10-11 UML 设计模型中的标准构造型

实体类是问题域类的设计标识符。通常，它们还是持久类。**持久类**在程序结束后仍然存在。很明显，实现方式是将它们写入文件或数据库中。

边界类或**视图类**是存在于系统的自动化边界上的类。在桌面系统中，这些类可以是窗口类和所有与用户接口有关的其他类。

控制类是在边界类和实体类之间起协调作用的类。换句话说，它负责从边界类对象获取信息，然后发送给正确的实体类对象，就像是域层和视图层之间的一个交换机。

数据访问类是从数据库中获取信息或向数据库发送信息的类。它不是在实体类方法中包含数据库访问逻辑（包括 SQL 语句），而是设计一个专门用于访问数据库的数据访问层。

10.5.2 设计类表示

图 10-12 所示为在图 10-8 中首次出现的设计类符号的详细信息。名字部分包括类名和

构造型信息。底部的两个部分是关于属性和方法的更详细的信息。

分析员用来定义属性的格式如下：

- 可见性——**可见性**表示其他对象是否能直接访问该属性（“+”表示可见，“-”表示不可见）。
- 属性名称。
- 类型表达式。
- 初始值。
- 原型（在花括号内），比如｛关键字｝。

图 10-12　定义设计类的符号

第三部分包含了方法特征信息。方法特征显示了需要调用这个方法的所有信息。它给出了需要发送的消息的格式，包括如下内容：

- 方法可见性。
- 方法名称。
- 方法参数列表（输入参数）。
- 类型表达式（方法返回参数的类型）。

域模型列表包含了所有在分析活动中发现的属性。设计类图包含了关于属性类型、初始值和原型的更多信息。它也可以包含用来声明的构造型，如图 10-8 所示，Student 设计类图的第三个组件包含了该类的方法特征。UML 是一种通用的面向对象的表示方法，并不局限于某一种语言。所以，这种表示法与程序设计中方法的表示不完全一样。

在图 10-8 中，用下划线标识的 findAboveHours(int hours):StudentArray 方法是一个比较特殊的方法。在面向对象方法中，类是创建个体对象和实例的模板。大多数方法适用于类的实例。然而，分析员常常需要检查类的所有实例。这种类型的方法被称为**类方法**，它们用下划线表示。

图 10-13 是带有属性和方法的设计类的例子，它显示了设计类如何继承工作。在第 4 章，你学习了关于泛化 / 特化层次图。在问题域模型中，泛化 / 特化层次图在设计模型和编程语言中成为了继承。三个子类的每一个都继承了父类（Sale）的所有属性和方法。因此，每个子类都有 SaleID、SaleDate 等。在这个例子中，每个子类也都有附加的属性，这些属性对它自己特定的类来说是唯一的。每个子类也都有一个带有下划线的唯一属性，如 noOfPhoneSale。有下划线的属性是**类属性**，而且有与类方法一样的特征。类属性是静止的变量，并且包含与所有同种类型实例化对象一样的值。

不仅方法和属性是通过子类来继承的，关系也是这样继承的。在图 10-13 中，Sale 对象只能与一个顾客关联。每个子类继承了同样的关系，并且只能与一个顾客关联。最后，注意 Sale 类的标题是斜体。斜体类名表示它是一个**抽象类**——不能被实例化的类。换句话说，就是不会有 Sale 类的对象。系统中的所有订单将作为必须被实例化的三个子类之一。系统的每个订单可以是 PhoneSale、InternetSale 或 StoreSale。三个子类之一被称为**具体类**，因为它能被实例化，换言之，可以创建对象。抽象类的目的通过图说明了，它为每个子类需要的所有属性和方法提供了一个中心存储区域。这个例子证明了重用面向对象编程的一种实施方式。抽象类中的属性和方法允许再使用每个子类，仅需要编写一次。

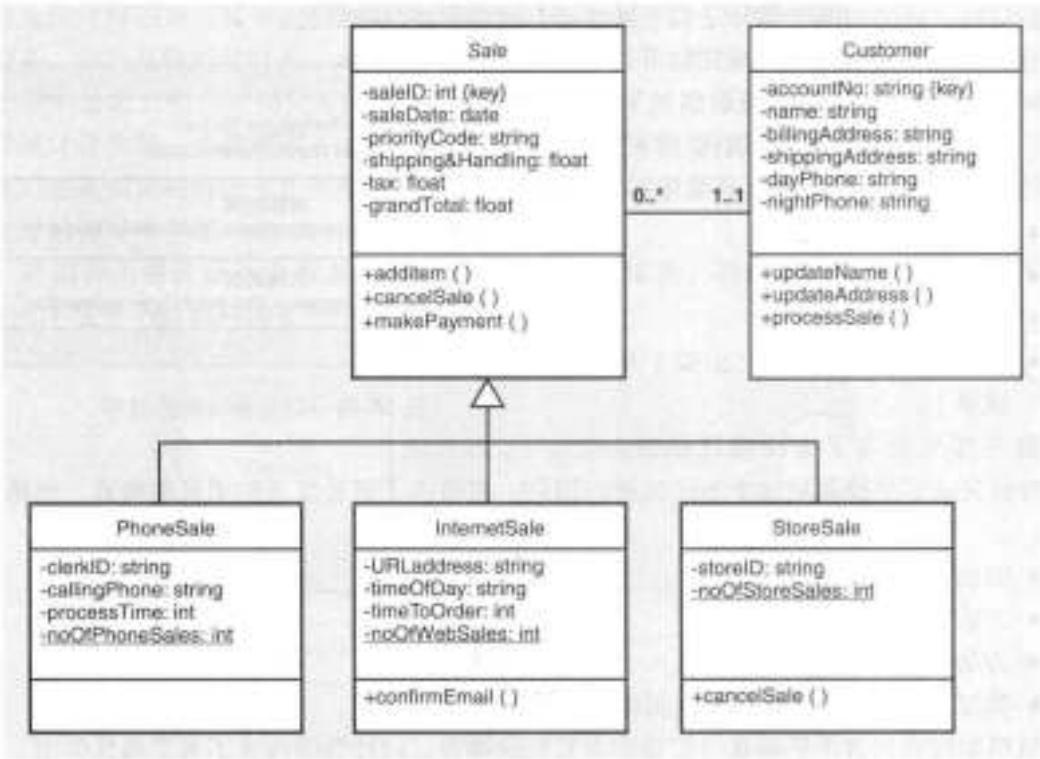


图 10-13 销售类（抽象）和显示继承关系的三个具体子类

10.5.3 开发初步的设计类图

为了说明如何开始设计过程，我们会开发一个基于域模型的初步设计类图。图 10-14 是在第 4 章中为 RMO 设计的域模型类图的片段。

可以通过扩展域模型类图的方法来构造初步设计类图。它需要两个步骤：（1）给属性添加类型和初始值信息，（2）添加导航可见性。正如前面所提到的，面向对象设计是用例驱动的。所以，我们先选择一个用例，然后只关注这个用例所涉及的类。

属性细化

属性细化很简单。设计者往往凭借自己的经验来确定属性的类型。在大多数情况下，所有属性是不可见的，并且前面带有“-”。我们也需要增加一个新部分来描述该类的方法特征。

导航可见性

就像前面提到的，面向对象系统是相互作用的对象的集合。顺序图记录对象之间的交互。然而，如果一个对象通过发消息的方式和另一个对象进行交互，那么第一个对象对于第二个对象来说必须是可见的。这里我们所说的**导航可见性**，就是指一个对象能够看见另一个对象并与其进行交互的能力。在设计中，我们会用到两种导航可见性：属性导航可见性和参数导航可见性。属性导航可见性是指一个类具有一个引用其他对象的属性，这种可见性是通过属性引用实现的。参数导航可见性是指一个对象作为参数被传递给另一个对象。参数通常是通过一个方法调用来传递的。有时，开发人员直接把导航可见性简称为导航或可见性，但是我们推荐使用导航可见性，以便与属性、方法的 public 和 private 可见性区分开来。

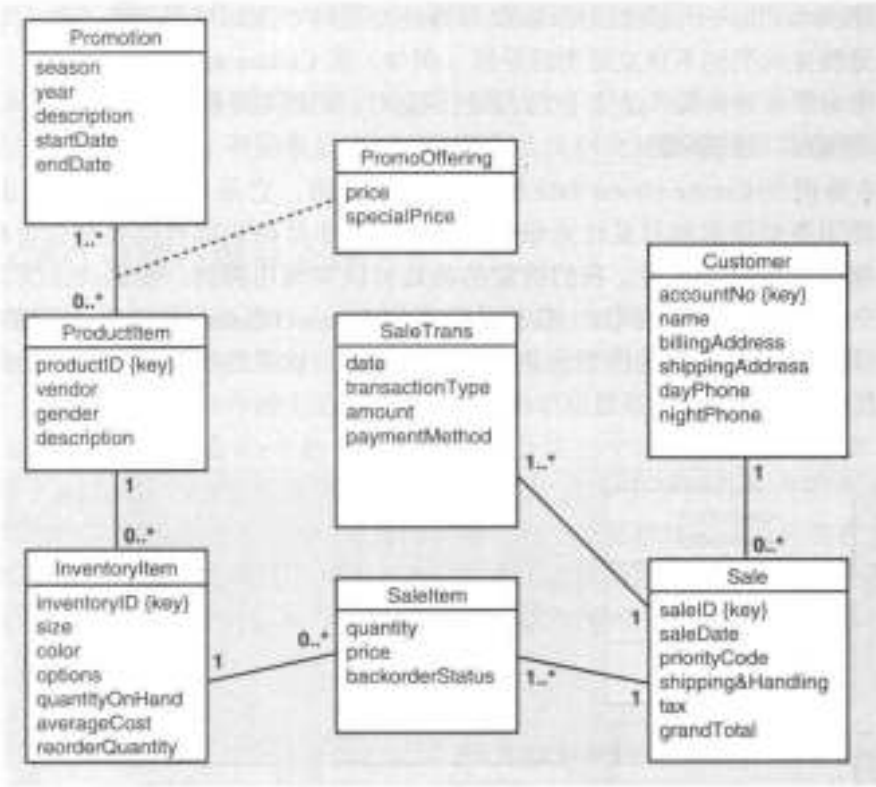


图 10-14　部分 RMO 销售子系统域建模类图

图 10-15 所示的是 Customer 类和 Sale 类之间的单向导航可见性。注意，在 Customer 类中有一个叫 mySale 的变量。该变量指向某个 Sale 实例，导航箭头表示一个 Sale 对象必须对 Customer 类可见。在这个例子中，我们添加了 mySale 属性来强调这个概念。

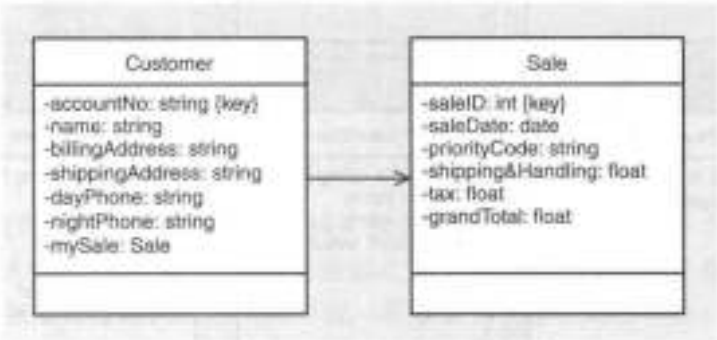


图 10-15　Customer 和 Sale 之间的属性导航可见性

现在，让我们考虑为 RMO 设计类图添加导航可见性。别忘了，我们目前开发的只是初步设计类图，所以在随后的设计过程中，我们可能还需要对图中的导航箭头做适当的修改。在构建导航可见性时，需要解决的基本问题是：哪些类需要访问其他类以及能访问哪些类？下面是一些指导原则。

- 一对多关系说明了上级 / 下级关系通常是从上级到下级的导航，例如从 Sale 到 SaleItem。有时候，这些关系组成了导航链的各个层次，例如从 Promotion 到 ProductItem 再到 InventoryItem。

- 在强制关系里，一个类的对象在没有另一个类的对象的情况下是不能存在的，通常是从更独立的类到不独立的类的导航，例如，从 Customer 到 Sale。
- 当一个对象需要获取来自另一个对象的信息时，必须有导航箭头指向对象本身或父类。
- 导航箭头可以是双向的。

图 10-16 是用例 Create phone sale 的初步设计类图，它是基于上述两个步骤所构建的。第一步是用类型信息和可见性来细化属性。第二步是确定哪些类要被包含在内，哪些类需要对其他类有导航可见性。我们确定的类是可以实现用例的。我们决定其他类是否必要时，要基于所需要的信息。例如，价格信息是在 PromoOffering 类中的，而描述信息是在 ProductItem 类中的。关于可见性要记住的一件事情是，这里的类是编程类，而不是数据库类。所以，我们不用考虑在关系数据库中的外键。

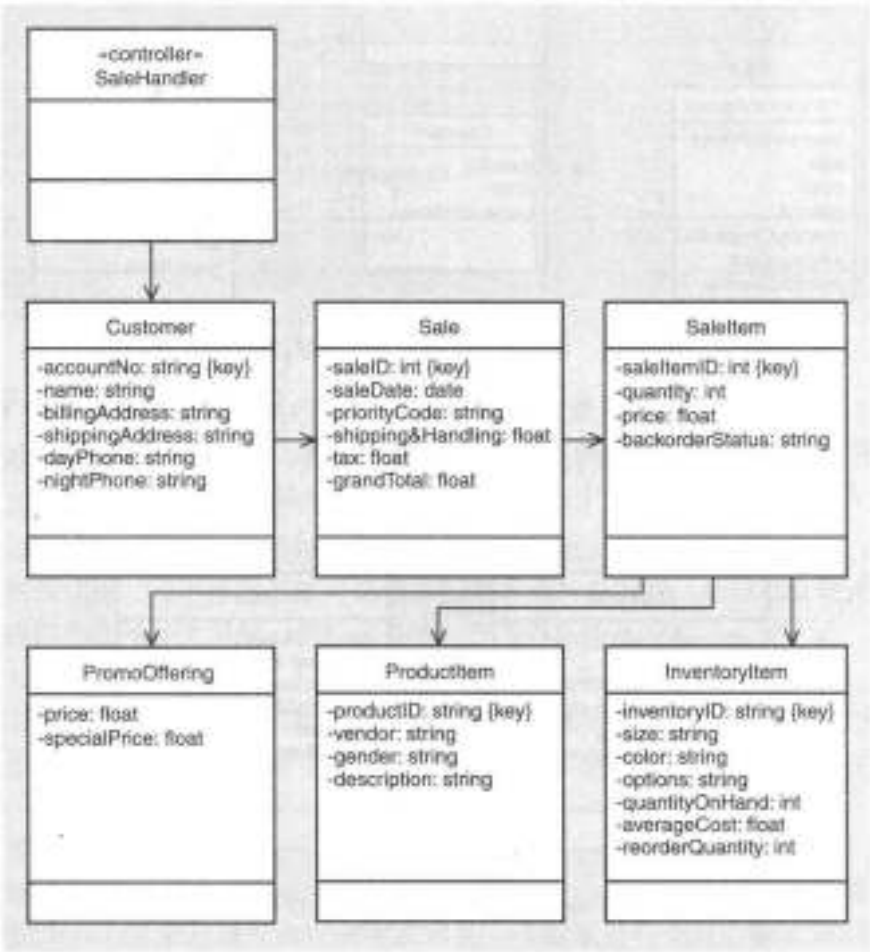


图 10-16　用例 Create phone sale 的初步设计类图

图 10-16 中多了一个设计类——SaleHandler，这是构造型充当控制类的角色。就像之前提到的，控制类或是用例控制器就是一个能在用例处理过程中有帮助的工具类。注意，它的可见性处于可见性层次中的顶部。

这里要强调三点：第一，因为设计的细节是依靠用例完成的，所以我们需要确保顺序图支持并实现最初定义的导航；第二，导航箭头要随着设计过程进行相应的修改，以保持

和设计细节的一致；第三，要基于为用例创建顺序图时制定的设计决策为每个类添加方法特征。

作为在开发顺序图之前开始的步骤，许多开发人员喜欢在头脑风暴过程中使用 CRC 卡来帮助确定包含在每个用例中类的集合。下一节我们会解释交互图是如何帮助面向对象细节设计的。

10.6　用 CRC 卡进行细节设计

CRC 卡是一种头脑风暴技术，它在面向对象开发者中非常流行。在这里，"CRC"是类、职责和合作这几个词语的首字母组合。开发人员在设计阶段使用这种技术来帮助确定类的职责和为一个特定类相互合作的类的集合。

CRC 卡是简单的 3×5 或 4×6 的卡片，用线条分成三个区域：类名、职责和合作类。图 10-17 说明了 RMO 的 CSMS 系统中 CRC 卡的两面。这个卡片的部分内容已经填好。卡片的顶部是类名。左边部分列出了类中对象的职责，包括类要维护的信息和类在支持某些用例时所采取的措施。右边部分列出了与这个类一起合作支持特定用例的其他类。括号内的信息是从合作类返回信息到主类中。在卡片的背面，可以选择列出在一个特定用例中需要的重要属性。

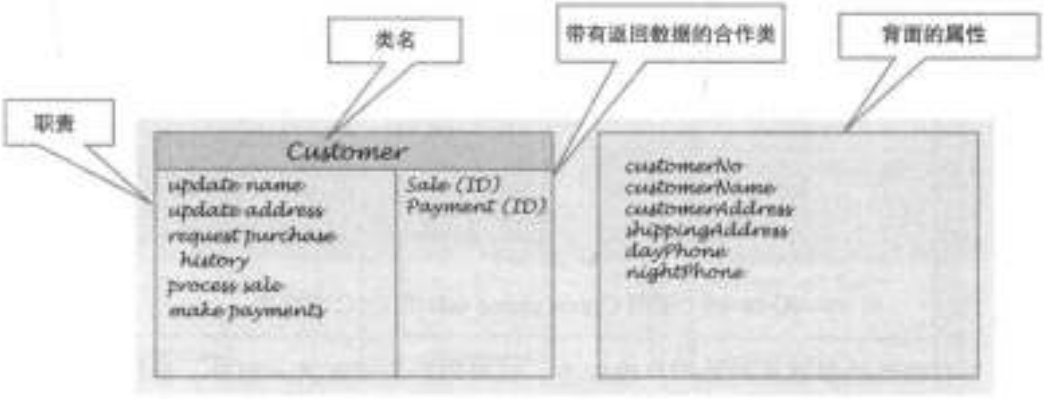


图 10-17　CRC 卡的例子

开发 CRC 模型的处理过程通常会在头脑风暴会议中完成。使用 CRC 卡的设计会议从开始时就已经包含了大量的信息。在开始设计会议之前，每个团队成员都要有一份域模型类图的复印件。当然，也需要有用例图或用例列表。其他细节信息，如活动图、系统顺序图和用例描述，都要和空白的 CRC 格式卡片一起提供。对于你要设计的每个用例，以下这些步骤要反复完成。

- 选择用例——因为这个过程是为了设计或辨识一个简单的用例，所以一开始要做的是用一组没有用过的 CRC 卡。因为我们正在做多层设计，所以将一张卡片当作用例控制器卡片。
- 确定负责这个用例的问题域类——这个对象会接收到用例控制器发送的第一个消息。使用在分析阶段开发的域模型，选择一个类负责。只关注问题域类。在卡片的左边写下对象的职责。例如，Customer 对象可能是负责产生新销售的，所以职责就是 Create phone Sale。

- 确定必须要与主要对象类合作完成用例的其他类——其他类将会需要信息。对于创建销售这个例子来说，我们需要 Sale 类卡片和 SaleItem 卡片，价格信息可能来源于 PromOffering 类，InventoryItem 类会被用来检查存货。在主要问题域卡片上列出这些类。确定好其他类之后，在卡片上写下它们的职责。此外，在所有卡片的背面写下每个对象类的相关信息或属性。

在这个过程的最后，你会拥有一组 CRC 卡，它们能合作支持用例。这个过程可以用几个其他活动来强化。首先，CRC 卡可以放在桌上以便执行或调用。换句话说，调用订单可以在这时候决定。例如，Customer 对象创建了一个 Sale 对象，Sale 对象创建了 SaleItem 对象，然后 SaleItem 对象访问了 ProductItem 和 InventoryItem 对象来获取信息。图 10-18 所示为用例 Create phone sale 的一组 CRC 卡。

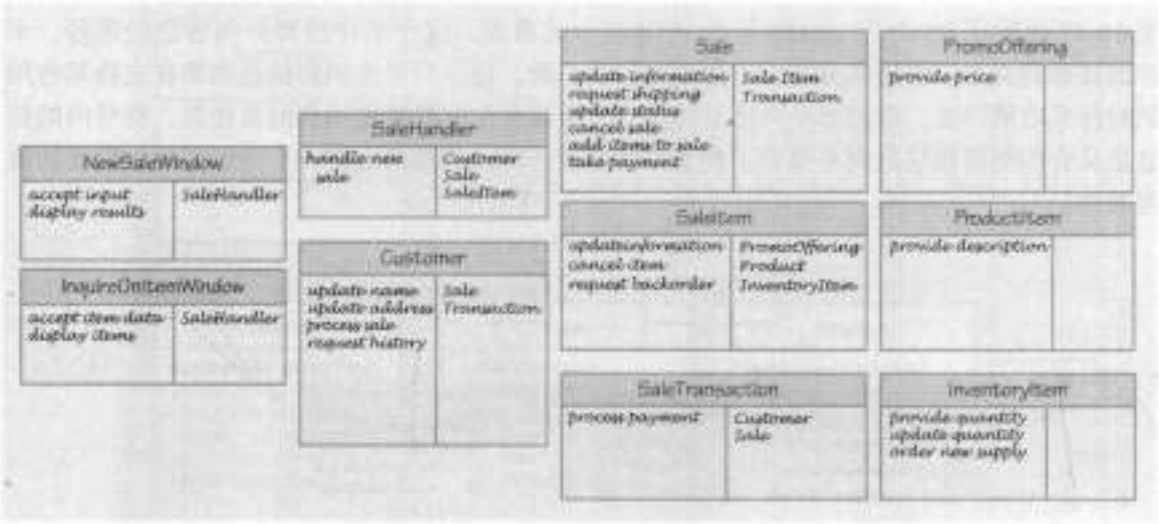


图 10-18　用例 Create phone sale 的 CRC 卡模型

另一个有帮助的步骤是增加用户接口类。如果用户是团队的一部分，如果用户接口需求的一些准备工作已经完成，那么为用例所需的所有用户接口窗口类添加 CRC 卡是很有效的。通过增加用户接口类，所有的输入和输出表格可以包含在设计阶段中。很显然，这就变成了更加复杂的设计。

其他任何工具类也可以被添加到解决方案中。例如，对于一个三层设计来说，数据访问对象会成为解决方案的一部分。每个持久类都会有一个数据访问类来读和写数据库。

用例设计的最后，还剩下两个其他的重要任务。因为 CRC 卡只有单独用例的数据，所以信息可以被转换成设计类图。然后设计类图可以变成新系统中每个类的所有信息所在的中心存储库。

第二个任务是将一组 CRC 卡用橡皮筋绑在一起，以便下个步骤中的使用。如果用例是很简单的，那么 CRC 卡由程序员携带，然后实施用例。

为了完成这个例子，让我们回到 DCD，基于在头脑风暴会议中创建的设计信息将它更新。图 10-19 所示为一个更新了的 DCD，添加了几个方法并更新了可见性。我们首先注意的是添加了一个新类。很明显，我们在初步设计类图中忽视了 OrderTransaction 类。我们也注意到 OrderHandler 类需要对 Sale 类具有可见性，以此来处理支付过程。比较在 CRC 卡上

确定的职责以及每个类中描述的方法名，注意其密切关系。

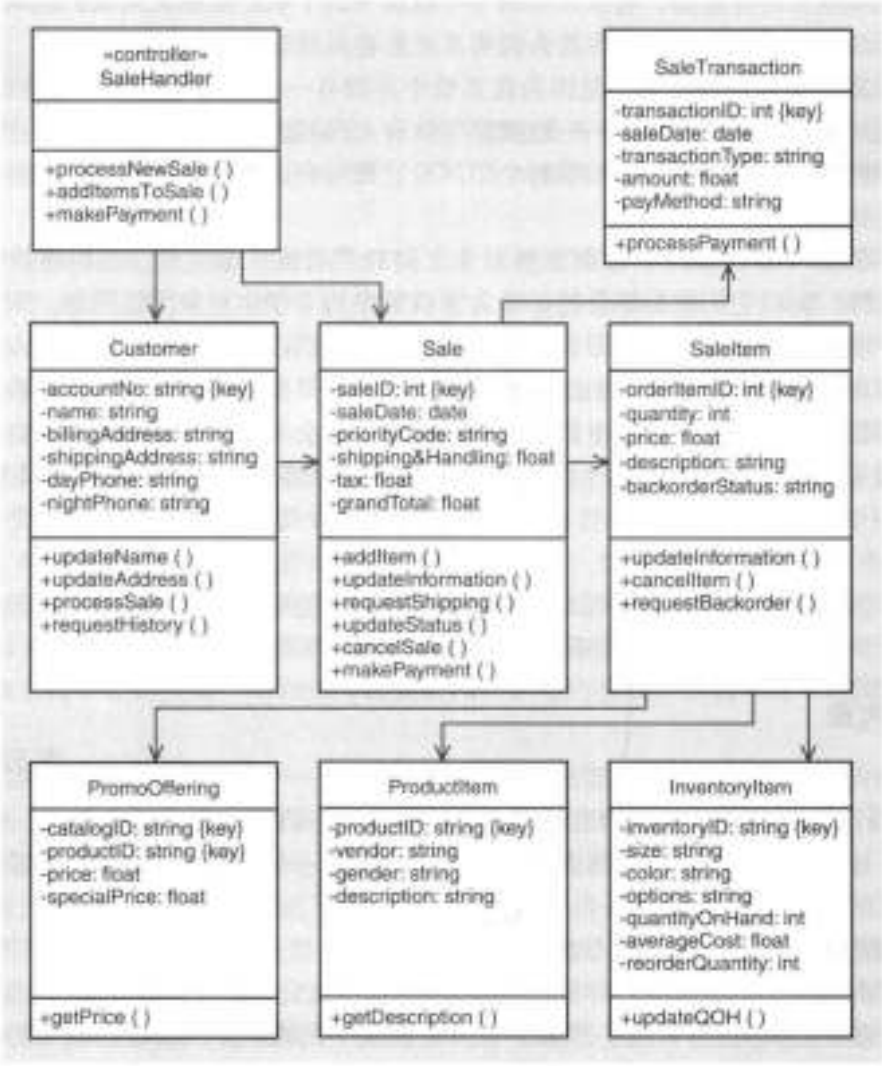


图 10-19　用例 Create phone sale 更新之后的设计类图

通常，当程序员开始使用 CRC 卡时，他们会列出类的许多不同职责。例如，开发人员可能会说 SaleItem 类应该有价格。实际上，PromoOffering 类会提供价格，然后 SaleItem 会使用。换言之，它和方法很相似，通常是帮助考虑职责的——请求做某些事而不是发生事情时随机地采取措施。

10.7　细节设计的基本原则

既然你已经了解了一个面向对象程序是如何运行的，也学习了设计类的表示方法，让我们来回顾一些指导设计决策的基本原则。在本章中，我们讨论面向对象设计的步骤时已经使用了这些原则，因为它们对面向对象设计的每一部分都很重要。

10.7.1　耦合

在前面的例子中，Customer 对 Sale 有导航可见性，也可以说 Customer 和 Sale 是耦合

的，或者是有联系的。**耦合**是对设计类图中类与类之间连接关系紧密程度的定性度量。一种比较容易理解耦合的方法是，看设计类图中导航箭头的个数。对系统来说，弱耦合比强耦合好。换句话说，较少的导航可见性箭头说明系统更容易理解和维护。

我们说耦合是定性的度量，是因为在系统中并没有一个特定的数字来定量地说明耦合。设计者必须对耦合有一定的感觉——也就是当耦合太强或太弱时能认识到。耦合是作为一个设计过程用例被评价的。通常，如果每个用例设计都有合适的耦合级别，那么整个系统也会有比较合适的耦合。

我们再回顾一下图 10-1，仔细观察对象之间的消息流。很明显，互相通信的对象必须有导航可见性，所以它们就是耦合的。输入窗口对象为了学生对象发送消息，则它对学生对象必须有导航可见性，所以输入窗口对象和学生对象是耦合的。但是注意，输入窗口对象并没有和数据库对象连接，所以它们之间不是耦合的。如果我们设计一个系统使输入窗口对象也可以访问数据库对象，那么这个用例的整体耦合度将会增加——也就是说，会有更多的连接。这样做是好还是坏呢？在这个简单的例子中，这也许不会成为问题。但是对一个有 10 个或更多用例的系统，无规则的连接和导航可见性会导致很强的耦合，使系统分析变得很复杂。

那么，为什么强耦合度不好呢？主要的原因是一个类的变化会波及整个系统。因此，有经验的分析员会尽量简化耦合，并降低对新系统设计的影响。

10.7.2　内聚

内聚指的是一个类中各种功能的一致性。**内聚**是对一个类的功能一致性的定性度量。比如在图 10-1 中，我们希望学生类有这样的方法或功能，即能够输入学生信息，如学生的学号或名字。这代表了单一功能和高内聚类。但是如果同一个对象也有分配教室或分配教授的方法呢？类的内聚度就会被减弱。

低内聚的类有几个方面的负效应。第一，它们难于维护。因为它们有很多不同的功能，所以对系统内的变化非常敏感，容易产生连锁反应。第二，很难再次使用这些类。它们有很多不同的（通常是无关的）功能。因此，在其他环境下的再次使用是没有意义的。比如，一个有按键功能的按钮经常被再次使用。但是，有按键功能和用户登录功能的按钮类却很少被重用。最后一个缺点是没有内聚的类难于理解。通常，它们的功能相互交叉，逻辑非常复杂。

虽然内聚无法用一个度量系统来衡量，但是我们可以把类的内聚性分成极低、低、中等以及高内聚 4 种类型。我们最需要的是高内聚。一个极低内聚的类可以是这样的，它在不同的功能领域都有服务的任务，比如一个类既可以访问网络又可以访问数据库，这两种活动是完全不同的，而且是为了达到不同的目的，因此我们把它们放在一个类里就会造成低内聚。

举个低内聚的例子。这个类可能在相关领域有不同的任务，比如说一个类完成了数据库中所有表的数据访问功能。然而，更好的方法是用不同的类来访问用户信息、命令信息和库信息。虽然功能是一样的——这两种方法都访问数据库——但是传入和导出的数据类型是不一样的。所以，与整个数据库连接的类就不像只与用户表连接的类那样易于再次使用。

举个中等内聚的例子。这个类有联系密切的相关任务，比如一个维护信息和客户账号信息的类。可以定义两个高内聚的类：一个类记录客户信息，比如名字和地址；另一个或一组类记录客户账号，比如余额、支出、信用卡信息和所有的财务活动。如果客户信息和账号信

息是受限的，它们就被组合到一个中等内聚的类中。在系统设计中，中等内聚或高度内聚的类都是可以接受的。

10.7.3　变量保护

设计中一条基本的准则是**变量保护**——它的思想是系统中不会变化的那部分应该和那些将会变化的部分隔离开来。在设计系统时，你应该试着将系统中比较稳定的部分和需要改变的部分分开。

变量保护是驱动多层设计模型的原则。设计者应该将所有用户接口逻辑和业务逻辑混合在同一个类中。事实上，在早期的面向对象、事件驱动系统中，如那些用早期 VB 和 PowerBuilder 版本建立的系统，业务逻辑会包含在视图层类中——通常会以窗口输入的格式。许多 Web 应用程序也结合了 HTML 和业务逻辑。这种设计的问题是当接口需要被更新时，所有的业务逻辑都要被重写。一个更好的方法是将用户接口逻辑和业务逻辑分离。然后，用户接口可以在不影响业务逻辑的情况下被重写。换句话说，业务逻辑——为了变得更稳定——在用户接口中是变量保护的。

此外，如果更新业务逻辑需要添加新类和新方法呢？如果用户接口类与业务类是紧紧地联系的，那么在用户接口类中变化的影响是连带的。然而，因为用户接口可以发送所有的输入消息到用例控制器类，所以业务逻辑中方法或类的改变是与控制器类分开的。你会发现变量保护影响了每个设计决策，所以你应该观察和认识在所有设计活动中这个原则的应用。

10.7.4　间接

间接是通过放置一个充当连接的中间件来分离两个类或其他系统组件的原则。换句话说，间接不是从 A 直接到 B，而是首先通过一个中间者 C。或者用消息术语，不是从 A 发送消息到 B，而是让 A 先发消息给 C，然后 C 再将消息传送给 B。

尽管变量保护有很多方法，但是经常使用的是间接。插入一个中间对象使得系统中任何变量与这个中间对象都是可以分离的。间接同样是一条有助于系统安全的设计准则。许多公司在内网和外网之间设有防火墙和代理服务器，代理服务器可以接收和发送消息。代理服务器就像一个真正的服务器，它接收 E-mail 和 HTML 页面请求等消息。但是它并不是一个真正的服务器，它只是接收消息并将它们分发给客户，这样就可以设置安全控制以便保护系统。

10.7.5　对象职责

面向对象开发的一个基本准则是**对象职责**的思想，即由对象负责实施系统过程。这些职责可以分为两类：认识和行动。换句话说，一个对象应该知道什么，以及一个对象应该做或者激起什么。

“认识”包括这样的职责，如认识自己的数据，认识其他一起协作执行用例的类。很明显，一个类应该认识自己的数据，了解存在什么样的属性，以及怎样维持这些属性的信息，还应该认识到哪里去找需要的信息。比如，在一个对象初始化的时候，没有被作为参数传输的数据可能也是需要的。一个对象应该认识到，也就是说，一个对象应该具有对向它提供信息的对象的导航可见性。比如，在图 10-8 中，学生类的第一个构造器不接收学生 ID 值作为参数。学生类负责根据它所知道的某些原则创建一个新的学生 ID 值。

"行动"包括一个对象所做的支持用例执行的所有活动。一些活动接收和发送信息。另一些对象负责实例化或创建完成用例所需的新的对象。类之间要相互协作来完成用例的实施，一些类负责统筹协作。例如用例 Create phone sale，Sale 类负责创建 SaleItem 对象。另一个类，如 InventoryItem，只负责提供它自己的信息。

本章小结

系统开发人员主要的创造性活动是编写能解决业务问题的计算机软件。最近，本书关注的是两个主要的活动：理解用户需求（业务问题）和解决并设想解决方案系统。本章关注的是如何部署和开发解决方案系统——也就是设计系统。系统设计是一座桥梁，它将业务需求转换为程序员能用来编写解决方案系统的术语。

结构化设计是部署新系统的第一步。它的目的是确定新系统中不同组件的结构和部署。组件图显示了新系统中不同的可执行组件以及它们是如何与其他组件相联系的。许多新系统是企业级系统，因为它们被用在组织的各个位置。它们也能共享资源，如公共的信息数据库。

一旦结构化设计确定了，就要开始细节设计。细节设计的目的是决定单个类的对象和方法，以此来支持用例。细节设计的过程是用例驱动的，因为它是为每个用例而做的。

细节设计的过程可以分成两个主要部分：开发设计类图（DCD）和开发交互类的集合及通过顺序图所得到的每个用例的方法。DCD 通常用两个步骤来开发。初步设计类图是基于域模型类图而创建的，但是它会经过修改和拓展成为顺序图。描述对象如何合作的主要方法是使用 CRC 卡。对于简单的用例，一组 CRC 卡对编写代码来说已经足够了。对于更复杂的用例来说，CRC 卡是用作开发顺序图的开端。

我们建议开始一个更正规的系统设计而不是一开始就编写代码的原因在于，最终系统是更加强健的和可维护的。将设计看作严格的活动来做能建立出更好的系统。一些基本原则应该在系统开发过程中被考虑，两个关键思想是耦合和内聚。在一个好的系统中，类之间的是低耦合的，每个类是高内聚的。另外一个重要的原则是变量保护，意思是系统的一些部分要被保护，而不是和另一些不稳定的、会随时变化的部分紧紧相连。成为一个优秀的开发人员需要学习和遵循好的设计原则。

复习题

1. 用你自己的话描述面向对象程序是如何运行的。
2. 什么是实例化？
3. 列出用在面向对象系统设计中的模型。
4. 模型结构化设计中用到的 UML 图是什么？
5. 解释域类与设计类的区别。
6. 什么是企业级系统？为什么它在设计阶段是重要的？
7. 客户端 / 服务器网络系统和因特网系统的区别有哪些？
8. 什么是 API？为什么它很重要？
9. 被用来确定组件接口的符号是什么？
10. 问题域类和设计类在表示上的区别是什么？
11. 用你自己的话描述面向对象细节设计的步骤。

12. 用例驱动的设计是什么意思，什么是用例实现？
13. 什么是持久类、实体类、边界类、控制类和数据访问类？
14. 什么是类方法和类属性？
15. 什么是属性可见性和方法可见性？设计类图中显示的是哪种类型的可见性？
16. 什么是方法特征？
17. 比较抽象类和具体类，并给出相应的例子。
18. 描述导航可见性。为什么它在细节设计中是重要的？
19. 列出一些典型的环境，在这些环境中发生的是直接的导航可见性。
20. CRC 卡中维护的是什么信息？
21. CRC 卡设计会议的目的是什么？
22. 比较耦合和内聚的思想。
23. 什么是变量保护？为什么它在细节设计中是重要的？
24. 对象职责是什么意思？为什么它在细节设计中是重要的？

问题和练习

1. 给出以下系统描述，为内部桌面网络系统（如不需要因特网）开发组件图。

   BESD 系统是由人力资源部首先使用的，而且包含机密信息。因此，它要作为一个内部系统而开发，没有任何因特网元素。系统的数据库是人力资源员工数据库（HRED），这个数据库是通过公司内几个其他系统共享的。

   系统设计观点中的屏幕有两种类型：简单的查询屏幕和复杂的查询 / 更新屏幕。简单的查询屏幕只访问数据，不需要业务逻辑。复杂的屏幕通常是做基于复杂业务规则的很复杂的计算。这些程序通常能访问公司内其他数据库的数据表。

   数据库总是会存在于一个中心数据库服务器上。应用程序本身会被安装在每个允许访问的台式机上，并且它会通过访问的用户控制屏幕和程序功能。

2. 为下面的 Facebook 应用程序的描述开发组件图。

   Facebook 平台是为所有 Facebook 用户开发的应用程序。编写一个新程序，允许 Facebook 用户给朋友发送礼物和问候卡（这些都是真实的礼物和卡片，不是电子图片）。这个程序是在 Facebook 内部服务器上运行的，拥有自己的信息数据库，包括已经发送了的礼物和卡片的列表。礼物和卡片的实体店必须位于不同的服务器，因为它是一般互联网销售店面的一部分。这个店面维持了要销售的库存产品的数据库信息，并且收集信用卡支付信息。

3. 本章中，我们开发了初步设计类图、一组 CRC 卡和 Create phone sale 用例的最终设计类图。为用例 Look up item availability 创建同样的三个图。
4. 找到一家公司，它是使用 CRC 卡来做面向对象设计的。你所在大学的信息系统单元通常是使用面向对象技术的。参与 CRC 设计头脑风暴会议。采访一些开发人员，了解他们对 CRC 卡使用效率的感觉。查询在会议之后需要什么文档及如何使用。
5. 找到一家公司，它拥有企业级系统（如果你正在为这家公司工作，那么就查看一下他们使用的系统是什么）。分析系统，然后开发组件图和部署图。
6. 找一个使用 Java 开发的系统。如果可能，找到一个有互联网用户接口和基于网络的用户接口。它是多层（三层或两层）的吗？你能确定视图层类、域层类和数据访问层类吗？

7. 找到一个使用 Visual Studio .NET（VB 或 C#）开发的系统。如果可能，找到一个有互联网用户接口和基于网络的用户接口。它是多层（三层或两层）的吗？业务逻辑在哪里？你能确定视图层类、域层类和数据访问层类吗？
8. 选择一种你熟悉的面向对象编程语言。找到支持该语言的程序设计集成开发环境。测试一下从现有代码产生的 UML 类图的逆向工程的处理能力。评价一下效果如何，模型使用得轻松吗？它能输入 UML 图，并产生类定义草图吗？写个报告，讲讲有关它的工作过程以及它能生成什么样的 UML 模型。
9. 绘制能显示以下信息的 UML 设计类。

   类名是 Boat，它是实体类。所有三个属性是 private 的字符串，且初始值是空。属性 boat 标识符有主键“key”。其他属性是 boat 的制造商和模型。还有一个整数型的类属性，包括所有被实例化的 boat 对象的总量。Boat 方法包括创建一个新实例、更新制造商、更新模型以及获取 boat 标识符、制造商和模型年份。有一个类方法，它能获取所有 boat 的数量。

扩展资源

Grady Booch, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

Grady Booch, et al., *Object-Oriented Analysis and Design with Applications* (3rd edition). Addison-Wesley, 2007.

E. Reed Doke, J. W. Satzinger, and S. R. Williams, *Object-Oriented Application Development Using Java*. Course Technology, 2002.

E. Reed Doke, J. W. Satzinger, and S. R. Williams, *Object-Oriented Application Development Using Microsoft Visual Basic .NET*. Course Technology, 2003.

Hans-Erik Eriksson, Magnus Penker, Brian Lyons, and David Fado, *UML 2 Toolkit*. John Wiley and Sons, 2004.

Martin Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (3rd edition). Addison-Wesley, 2004.

Ivar Jacobson, Grady Booch, and James Rumbaugh, *The Unified Software Development Process*. Addison-Wesley, 1999.

Philippe Kruchten, *The Rational Unified Process, An Introduction*. Addison-Wesley, 2000.

Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process* (3rd edition). Prentice Hall, 2004.

Jeffrey Putz, *Maximizing ASP.NET Real World, Object-Oriented Development*. Addison-Wesley, 2005.

James Rumbaugh, Ivar Jacobson, and Grady Booch, *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

面向对象设计：用例实现

学习目标

阅读本章后，你应该具备的能力：

- 解释设计中不同种类的目标和层次。
- 为实现用例开发顺序图。
- 为细节设计开发协作图。
- 开发更新的设计类图。
- 开发多层子系统包。
- 解释设计模式并识别不同的特殊模式。

开篇案例　NEW Capital Bank：第二部分

NEW Capital Bank 的集成顾客账户系统项目已经进行了 2 个月。尽管由于团队仍处于学习迭代开发项目的进程中而会有少许障碍，第一次迭代开发依然运行得很顺利。项目领导 Bill Santora 正在与 Charlie Hensen——其中一个团队的领导——讨论一些系统技术的细节，从而为迭代回顾做准备。

"关于细节设计你的团队有何感受？" Bill 问 Charlie——这位早期正式评审设计的成员之一。"我知道一些程序设计人员想从为用户开发的用例描述中直接编写代码。他们并不乐意花时间设计。这还是个问题吗？"

"它运行得已经非常好了，" Charlie 说，"你知道，刚开始我保持怀疑，认为这浪费了很多时间。然而，这让我们一起工作得更好，因为我们知道其他团队成员在做些什么。我同样认为这样的系统更具稳定性。我们都是用同样的方法，也共享了大量的课程。当然，我们不用浪费大量时间去画一些草图。就使用而言，我们用一些快速图纸记录设计。

"你对于我们方法的优缺点有何见解？" Bill 问。"或者你认为在下一个迭代中有什么更好的改进方法？""我确实欣赏首先使用 CRC 卡来做一个初步设计的方法。"Charlie 回答。"通过一些用户来确认我们的合作是正确的，这是一个很好的方法。关于简单的用例，我们可以和用户一起来展示用户界面。在 CRC 卡和用户界面规格之间，我们应该有足够的程序，特别是现在我们已经确立的基本结构。接着，对于更复杂的用例，我们可以继续去做，并通过顺序图做一个详细的设计。关于顺序图的优点是它们足够详细，可以为初级程序员传递我们的设计想法，这使他们在团队贡献中效率更高。"

"所以，你会更改我们的方法或者你认为这样的方法对吗？" Bill 问，他依然在寻找改进程序的方法。

"现在它运行得很好。" Charlie 说。"我喜欢它的一点是我们有共同的 DCD，每个人都能访问和检查。当你准备在一个类中插入一些代码以检查类的已有功能时，这才是它的真正用处。中心存储库是一个伟大的工具，它让我们所有的代码和图变得形式化。我疑惑的是是

否有方法能够更多地使用这一工具。此外，我认为我们可以一起将这一方法运用到另一个迭代中去，并看看是否存在需要改进的地方。"

11.1 引言

第10章解释了适用于多层系统的设计理念和模型以及结构设计。章节后面的部分介绍了相关联的面向对象细节设计的理念。你同时也学到了如何通过CRC卡和设计类图来标识用哪些类的协作来执行用例。可以通过由这两个步骤产生的设计信息来对简单的用例进行反复编译。

这一章将更深入和系统地讲解面向对象的细节设计。细节设计可以在多个层面解决，对于初学者来说，可以定义一个比较简单但完整的过程。这一章注重基于顺序图和设计模式的用例来实现基础原理。一旦精通这两个主题，你就是一名面向对象的设计者。可以借阅一些关于设计模式和设计方法的优秀书籍。

用于扩展详细设计过程的方法称作**用例实现**。在用例实现中，每个用例都被独立使用来定义所有相互合作的类。作为这一过程的一部分，任何其他实用工具或支持类都会被标识。在此过程中需要注意定义类，从而维护多层结构设计的完整性。正如通过一个又一个的用例来定义类的细节，设计图的更新也是必要的。

这一章的最后简短地介绍了设计模式。正如任何工程准则一样，某些程序经尝试后已经成为行之有效的解决方法。即使面向对象的开发是一个相对年轻的工程学科，但它提供了标准的方法来设计用例，带来了稳定且构建良好的解决方案。你将学到一些标准的设计或模式。

11.2 多层系统的细节设计

第10章中关于CRC卡的讨论介绍了理想的合作对象，可以执行各种用例。然而，使用CRC卡的设计会议关注问题域类，而对影响细节设计的多层问题关注甚少。这一章将深入描述所有多层系统的细节设计。

回顾第10章中的图10-1，有三个代表着系统三个层次的对象。每个对象有一个确定的职能。输入窗口这个对象负责的主要是格式化并在屏幕上展示学生信息。它同时负责接收输入数据，不管是学生信息还是更改的信息，并将之转发到系统。对象也可能处理一些输入数据的编辑。这一对象从何而来？这一对象的作用和方法是什么？确认和定义窗口对象是应用设计和用户界面设计的一部分。

学生对象代表了中间层，或是对于用例而言的业务逻辑层。CRC设计部分将帮助你设计这一层对象的结构。然而，你也许会注意到CRC卡相当非正式，尤其是尝试从对象职能中确定类的方法时。CRC卡提供了在定义方法特征中的小方向与适当的输入和输出参数。本章形式化了准确识别方法和定义方法特征的过程。

再回到图10-1，数据库访问对象表示多层设计中的第三层。它负责连接数据库，读取学生信息，并发送给学生对象。它同时负责必要时把学生信息传送回数据库。这一对象并不是来自问题域类，它是由设计师创造的实用程序对象。

当你回顾细节系统设计时，应该考虑到几个问题。首先，这些对象如何在内存中创建？例如，学生对象如何以及何时创建？数据库访问对象又该怎样？其他的问题包括：其他对象将是必需的吗？什么对象表示认证？各个对象的生命周期是什么？也许学生对象应该在更新后离开，但数据库访问对象又该如何？

模式和用例控制器

模式也称作是模板，重复运用在每天的生活中。主厨运用食谱，这是模式的另一种说法，它将各种成分结合到一道美味的菜肴中。裁缝运用模板为精美的套装剪裁面料。工程师运用标准组件将它们结合到建立部署或设置模式中，来建造大楼、音响系统以及大量其他产品。模式被创造用来以解决问题。随着时间的推移和大量尝试，钻研特定问题的人开发了一套解决问题的方法。解决方法一般足够反复应用。经历过长时间的洗礼，解决方法被记录和公布，最终被接受为标准。

标准设计模板因为能加速面向对象的设计工作而在软件开发者中变得流行。模板的正式名称叫做**设计模式**。在 1996 年，由 Erich Gamma、Richard Helm、Ralph Johnson 和 John Vlissides 合著的《Elements of Reusable Object-Oriented Software》一书问世后，设计模式在面向对象的设计技术中被广泛接受。这四位作者也被称为“四人帮”（Gang of Four，GOF）。学习过更多的设计模式后，你将常会看到参考 GOF 模式的特定设计模式。在他们的书中，作者定义了 23 个基本的设计模式。如今，大量的模式都被定义了——从底层程序模式到中层结构模式再到高层企业模式。两个重要的企业平台——Java 和 .NET——有一系列的企业模式，这在很多著作中都有描述。

第 10 章介绍了用例控制器的概念。我们先构建用例控制器的概念，再解释它作为设计模式的重要性。对于任何特定用例，消息来源都是从外部角色到一个窗口类（也就是电子输入表单），再到一个问题域类。系统设计中涉及的一个问题是这个问题域类应该接收输入消息以减少耦合，保持高度聚合的域类，以及维护用户界面和域层之间的独立性。设计师通常定义中介类充当缓冲区之间的用户界面和域类，我们称这些类为**用例控制器**。例如，购物车的用例中也许会有称作购物车处理程序的控制器。

图 11-1 提供了用例控制器模式的规格。这一规格有五个主要元素：

- 模式名称
- 需要解决方案的问题
- 模式的解决方案或解释
- 模式样例
- 模式的优点和结果

你应该阅读这一规范来理解控制器模式的重要原理，了解它解决了什么问题、如何工作以及优点。这一相同的模板会和其他设计模式一起用于后续章节中。

用例控制器相当于一个开关，它将接收的消息路由给相应的域类。实际上，用例控制器是外部世界和内部系统的一个协调者。假如输入窗口对象要向一些对象发送消息，该怎么办？若没有使用用例控制器，输入窗口就需要引用所有的问题域对象。这样的话，输入窗口对象与系统之间的耦合度可能会很高，模块间将有很多连接。因此，用一个用例控制器来处理所有的输入消息能降低用户交互对象与问题域对象之间的耦合度。用例控制器有其自身的业务逻辑，这些业务逻辑控制用例的信息流。这样，域层设计类就能够重点关注属于自己的明确功能，从而提高内聚度。

在后面章节的所有例子中，我们会为每个用例定义一个控制器，这是一个普遍的做法，并且在许多开发环境中（如 Java Struts 等）会为每个用例定义一个控制器。当然，这就产生了许多制品。如有 100 个用例，那么就有 100 个控制器人造对象。为了减少控制器的数目，

开发人员通常将功能相仿的几个控制器合并为一个。不管采用哪种方法，只要设计合理，都会得到一个好的解决方案。

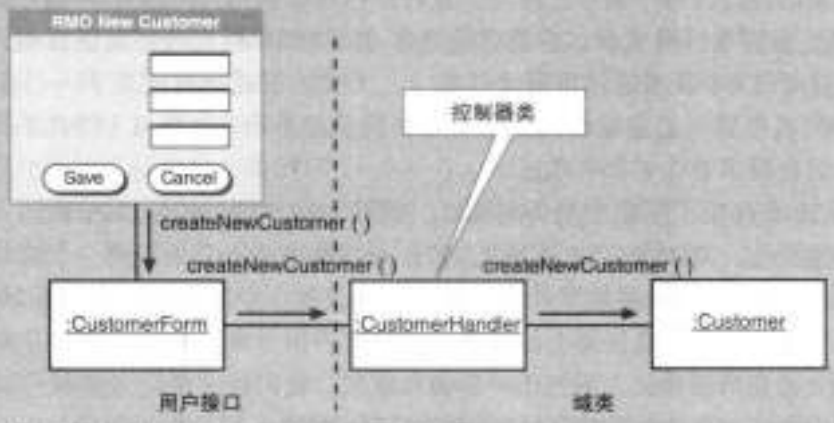
名称	控制器
问题	域类负责处理用例。然而，有许多的域类，哪一个是负责接收输入消息的？ 用户界面类如果有所有域类的可见性，那么它将变得很复杂。用户界面和域类的耦合度要如何降低？
解决方案	分配职责以接收发送到类的输入消息，这个类接收所有输入消息并充当交换机，然后转发到正确的域类。有几种方式来完成这一解决方案： 1）通过单一的类来表示整个系统。 2）通过每个用例的类或创建一群用例来演示用例处理过程。
样例	RMO 顾客账户子系统从 :CustomerForm 窗口接收输入消息。 这些输入消息传送给 :CustomerHandler，它相当于一个开关，将接收的消息路由给正确的域类。  其他的控制器模式样例将会被每个 RMO 用例所使用。
优点和结果	视图层和域层的耦合度减小了。 控制器提供了间接的层。 控制器和许多域类紧密耦合。 如果不注意，控制器会变得不合逻辑，并带有许多不相关的功能。 如果不注意，商务逻辑将会嵌入控制器类中。

图 11-1　控制器模板的模板规格

用例控制器是人在设计系统时创造的，是一种完全人工构造的类。有时，这些类被称为人工制品或人工制品对象。在深入解释设计时，你将会看到我们需要创建各种各样的人工构造的服务类——这些类用于执行用例但不基于任何域模型类。

这个过程将在后续段落以及第 10 章中阐明。这一过程平衡了耦合、间接、类、对象职责和变量保护，从而精确了系统设计的过程。读完这一章后，你将会了解第 10 章中描述的设计原理的重要性。

11.3　用例实现和顺序图

开发交互图是面向对象设计的核心。用例的实现确定了哪些类通过发送消息与其他类进行协作，用例的实现是在交互图的开发过程中完成的。在设计时，要开发两种交互图：**顺序图**和**协作图**。本节将展示如何用顺序图进行设计，下一节将简要说明用于系统设计的协作图。

自适应项目使用迭代和敏捷模型技术来使设计形式简化。对于这些类型的项目，相同

的开发者有时会做分析、设计和编程工作。而那些实例设计图常常被描绘于白板或者便笺中，并在编程完成后被丢弃。分布式团队的分析师和设计师在一个位置而程序员在另一个位置——通常不在一起。因此，在这种情况下，设计图有助于在整个团队中交流设计决定。

在这一章（可能是你的家庭作业），图表将会使用 MS Visio 开发。然而，在真实的项目中手绘图能更方便地在全队成员中扫描和传送。

后续部分详细解释了用例实现需要的阶段和技术。在第一部分，我们提供了部分顺序图来介绍术语以及顺序图的组成。接着我们通过创建顾客账户和加入购物车的例子，使用问题域类来阐明用例实现的过程。这些例子解释了组织和结构化问题域类到用例解决场景的核心过程。最后的样例解释了如何增加数据访问层类和视图层类。每个层都以使用相同用例的详细样本来阐明。

11.3.1 理解顺序图

你第一次听到顺序图是在第 5 章中学习如何开发系统顺序图（SSD）。现在，你应该适应了阅读、解释和开发一个 SSD。记住，SSD 是用来为某个单一的用例或场景记录系统中的输入和输出消息的。系统本身被看作一个叫 :System 的对象。系统的输入就是参与者传递给系统的信息，输出通常是回复消息。图 11-2 通过展示创建顾客账户这一用例的部分样例来回顾系统顺序图的元素。正如图中所示，每条消息都有源头和终点。

记住，如第 5 章所讨论的输入消息的排列是：

* *[true/false condition] return-value := message-name (parameter-list)*

用例细节设计的出发点往往是它的 SSD。SSD 只有两条生命线——一条是参与者，一条是系统。SSD 中最重要的是参与者和系统间消息的顺序。也许会有单一或大量的消息输入。输入消息也许会有数据参数，也许没有。也许会有 Loop 结构、Alt 结构、Opt 结构来重复消息输入和消息输出。Loop 结构用环形记录了一系列的消息。Alt 结构类似于 if-then-else 语句或选择语句，它允许不同的一系列消息的产生。Opt 结构可以选择调用一组消息。我们将在这一章中看到样例。

在图 11-2 中三条输入和输出消息描述了添加顾客这一活动所需要的顺序。第一条消息传送了顾客姓名、电话号码和邮箱的基础消息。相同的消息以顾客 ID 返回。返回的消息发生于系统在数据库中产生了一个新记录之后。接下来的两个输入消息简单地向账户增加了更多的消息：顾客地址消息和信用卡消息。返回消息在本质上是顾客需要确认的输入数据的重新格式化。额外的复杂情况将被增加到真实的环境中，比如验证和改进输入的消息。

图 11-3 是用例的两层细节设计。插图编号表示与两个层相联系的类。这一用例有一个视图层对象 :CustomerForm。注意，来自外部参与者的输入消息总是归类到视图层的对象。设计过程的目的是采集各个输入消息，并决定哪个系统必须反馈消息。对于第一条消息——创造新用户，系统简单地使用了 :CustomerForm 屏幕来接收输入值，并可以进行编辑。基于设计中的这一点，我们不必担心要求编辑输入值。顺序图的基本目的是定义合作的类，以及它们必须传送什么消息给其他类。

在 :CustomerForm 对象接收了创建新顾客消息后，它传送消息给 :CustomerHandler 对象，接着消息传送给顾客类，让它实例化一个新的顾客对象。这一消息被直接传送到对象矩形框中，这是可选择的，首选符号表示一条调用构造函数的消息。在矩形框中的标签

aC:Customer 表示这一矩形框显示了一个有 aC 变量名的顾客对象。应注意返回给控制器的创建消息对象的引用，这使顾客对象有可见性。后面发送其他消息时将会用到这一引用。

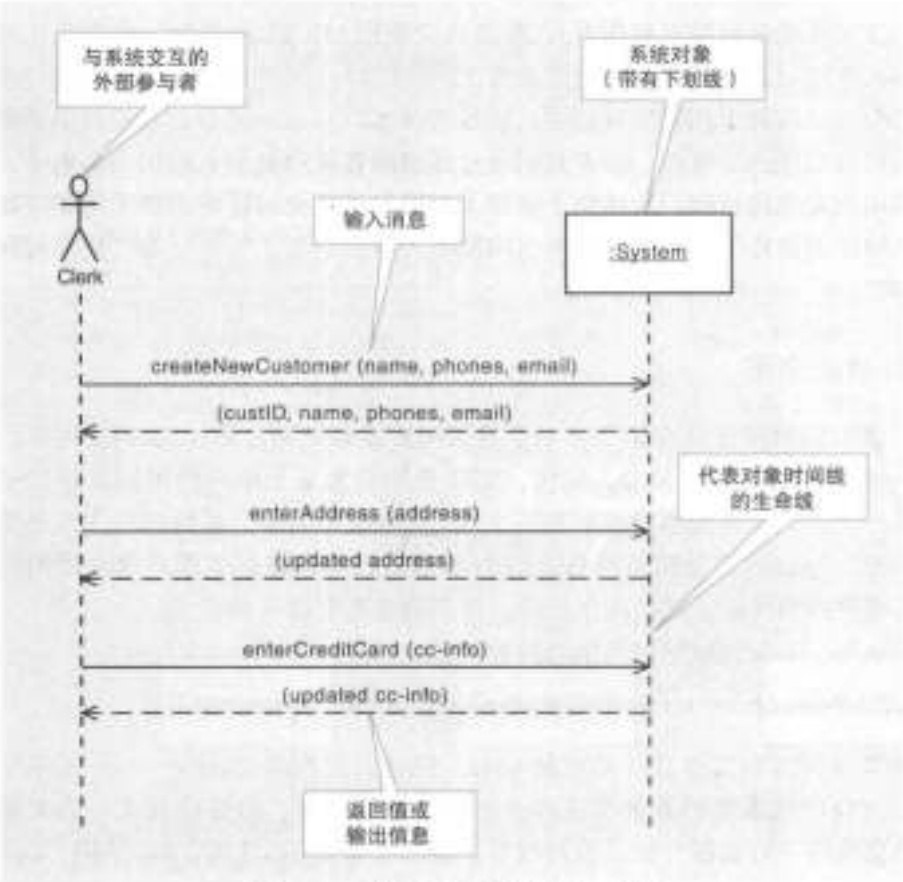


图 11-2　创建顾客账户用例的顺序图

当顾客对象被创建后，它负责将自身保存至数据库。因为这是一个相当简单的用例，所以我们将它限制于一个两层设计中。换句话说，顾客对象包括顾客逻辑和将自身写入数据库的数据访问语句。

一旦顾客对象被创建并保存到数据库，控制器收到一个指向对象的指针，它可以使用其为新创建的顾客对象“:Ac”访问 custID 字段以及任何其他数据。一旦数据返回到控制器，接着将返回至 :CustomerForm 屏幕，对外部参与者可见。或者，我们可以从 :Customer 屏幕到外部实体中添加虚线来表示用户看着屏幕。有两种记录返回消息的方法：要么作为返回值的函数，要么作为一个虚线返回特定值。

在这一点中一件显而易见的事是：当一条消息从初始对象传递到目的对象时，在编程中，原始对象正在调用目的对象的方法。因此，通过定义不同外部对象的消息，我们最终确定对象的方法。由消息传递的数据对应输入参数的方法，消息中返回的数据是方法的值。因此，一旦细节设计过程的用例实现，就能提取一系列的类和被要求的方法来完成编程。这些方法成为设计类图中的部分文档。

图 11-3 包括了一个称为活动生命线的新注释，它由垂直的矩形框所表示。因为一条消息在目标对象上调用一个方法，所以一条有价值的消息就可能是那个方法执行的持续期（即

方法处于活动状态的时间）。活动生命线代表了一个对象处于活动执行状态的时间。这就是为什么输入消息在垂直矩形框的顶部而返回消息在底部。顾客对象有到达对象底部的函数构造方法。活动状态一直持续到所有数据保存，甚至到其他方法被调用。

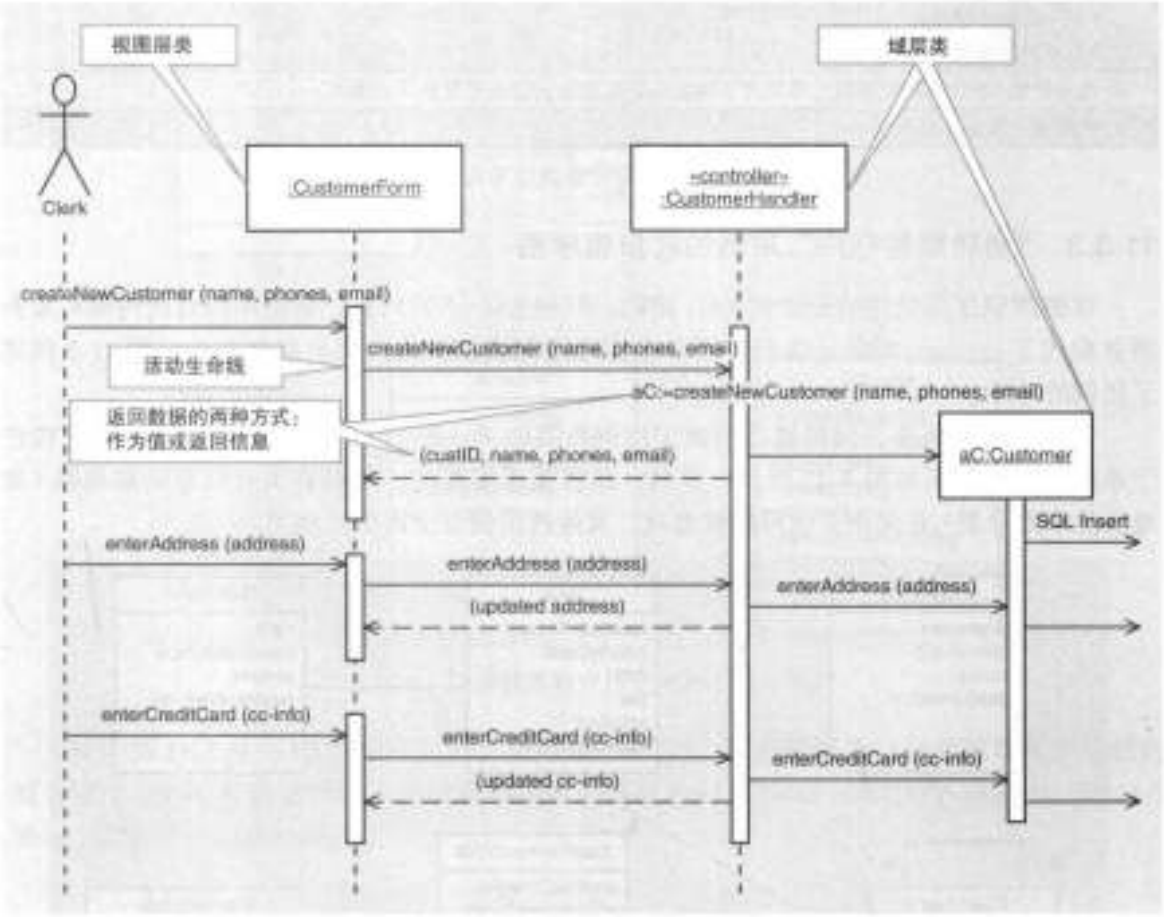


图 11-3　创建顾客账户用例的部分顺序图

11.3.2　用例实现的设计流程

在进入样例前，让我们回顾一下最终成果以及需要的步骤。正如第 10 章所述，细节设计的目的是确认新系统需要的类以及每个类的方法。因此，结果是一个综合阐述了属性和方法指定特征的设计类图（DCD），DCD 可以被一个大的图表或几个小的子系统图表所模式化。其他的结果不是细节顺序图就是用例或者场景的 CRC 卡。这两个模型是编写程序代码的基本输入。

图 11-4 与图 10-10 相同，罗列了细节设计的步骤以及每个讨论步骤所在的章节。我们在这一章节中继续跟随这一步骤。在第 10 章中，你已经学习了初步的 DCD 和 CRC 卡。在这一部分，你会学习如何开发细节顺序图和以传统消息更新初步 DCD。正如你在第 10 章所学，为了开发初步消息图，要逐个评估每个输入消息，以决定其他内部消息和需要全过程输入请求的问题域类。这致使我们开发初步的顺序图，它只包含问题域类。一旦问题域类的处理过程已知晓，数据访问层、视图层类和消息就被添加入图表。最终，DCD 通过用例实现过程中产生的细节的方法特征被更新。在接下来的两个部分中，我们实现了两个用例：创建顾客账户和加入购物车。

设计步骤	章节
1. 开发初步设计类图，显示导航可见性	10
2. 使用 CRC 卡为用例决定类的职责和合作	10
3. 为每个用例开发详细的顺序图 1) 开发初步顺序图 2) 开发多层顺序图	11
4. 通过使用 CRC 卡或顺序图，添加方法特征和导航信息，以此来更新设计类图	11
5. 将解决方案划分成适当的包	11

图 11-4　面向对象的细节设计步骤

11.3.3　"创建顾客账户"用例的初步顺序图

详细的顺序图使用的元素和 SSD 相同。但与 SSD 不同的是，系统中所有的内部对象和消息取代了 :System 对象。我们需要确定实现用例或场景的内部对象和消息。图 11-2 阐述了用例的 SSD。

下一个步骤是查看问题域类和确定用例所需的类。图 11-5 是 RMO 的 CSMS 顾客账户子系统的类图，正如图 4-22 所示。显然，顾客类是需要的。与顾客类有联系的其他类（如地址类和账号类）在用例中也可能被创建，其他的用例也会创建其他类。

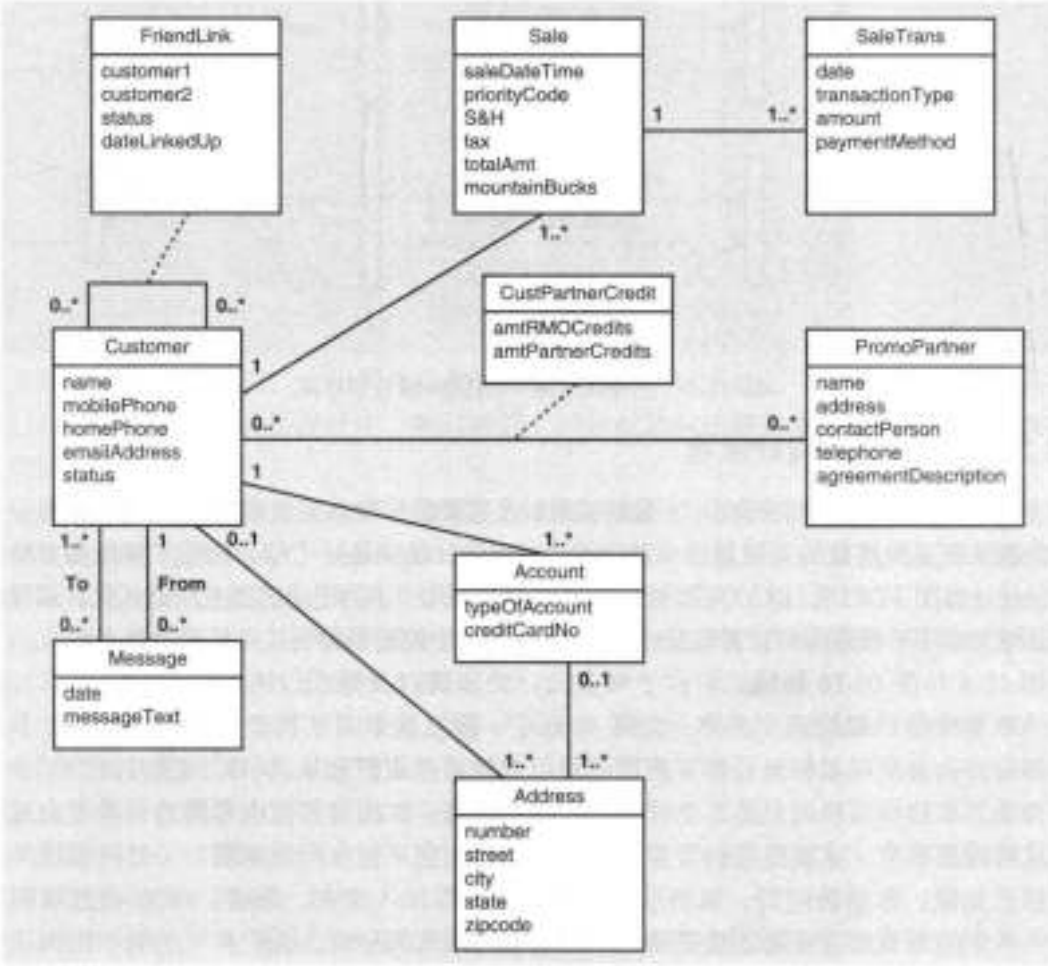


图 11-5　创建顾客账户用例的部分类图

创建初步 DCD，我们将阐述属性与类型消息。接着，我们将为导航可见性制定逻辑决策。使用第 10 章的指导方针，我们决定用户类将对其他两个类有可见性，分别是账户和地址。图 11-6 提供了用例的初步 DCD。

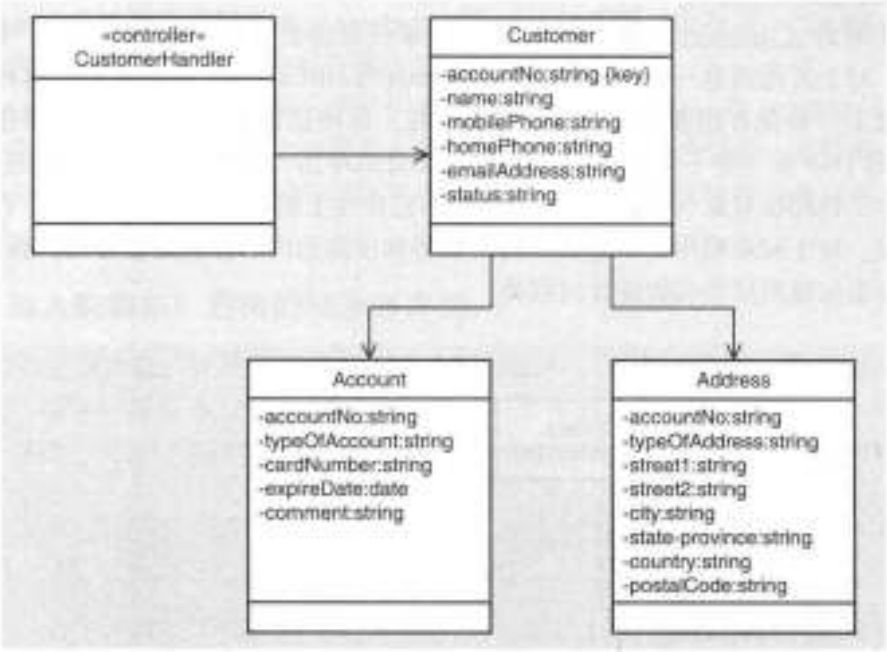


图 11-6　创建顾客账户用例的初步设计类图

基于图 11-2 这一 SSD 的用例和图 11-6 这一 DCD，我们继续进行创建顾客账户用例的细节设计。扩展 SSD 的第一步是将问题域对象以及 SSD 中的输入消息放入图表中。图 11-7 展示了细节设计的第一步。

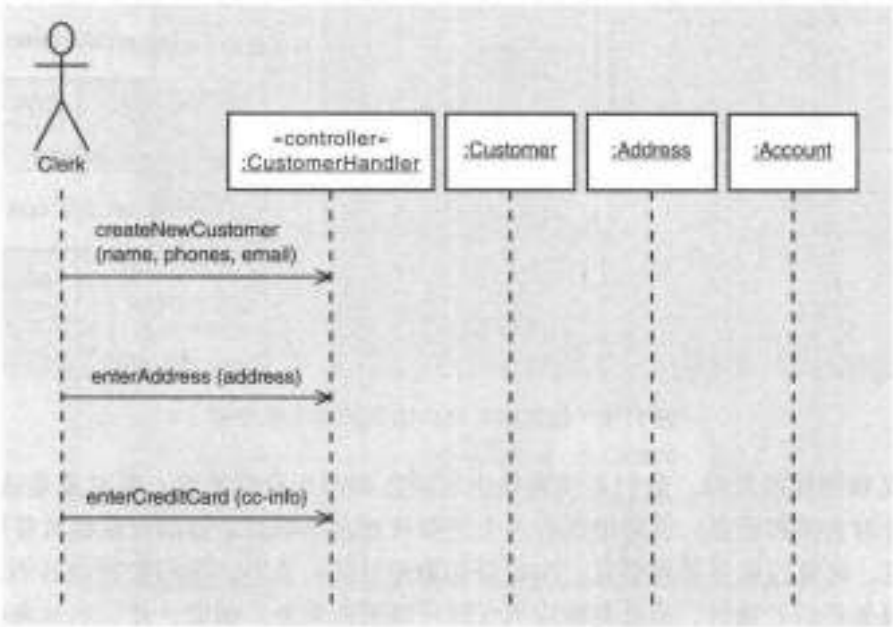


图 11-7　创建顾客账户用例包含的对象和输入信息

下一步骤需要确定对象之间的内部消息，以及确定每个消息的源头和目的地，从而收集到所有需要的消息。明确哪些消息是必要的、哪些对象会被激发，要基于前面提到的设计准则：耦合度、内聚度、职责和控制器。图 11-8 显示了“创建顾客账户”用例的完整的初步顺序图。:CustomerHandler 控制器接收输入消息，搜索正确的订单对象，并将创建新用户消息转发到正确的 :Customer 对象。:Customer 对象负责将自身存储至基于新用户输入消息的数据库中，对于其他消息——enterAddress 和 enterCreditCard，它也负责创建这些新的对象。在某种意义上，若没有顾客对象，它们则不存在。这些新创建的对象接着会保存至数据库。图 11-8 和图 11-3 有一些不同。图 11-3 仅表示部分顺序图，因为它并不定义全部的域类。然而，它包含了视图层对象 :CustomerForm，增加它是为了阐明更完整的顺序图。图 11-8 仅仅着重于域类。对于初步顺序图，定义所有的域类和所需的内部消息是重要的。再接下去的步骤中，将会添加视图层类和数据访问层类。

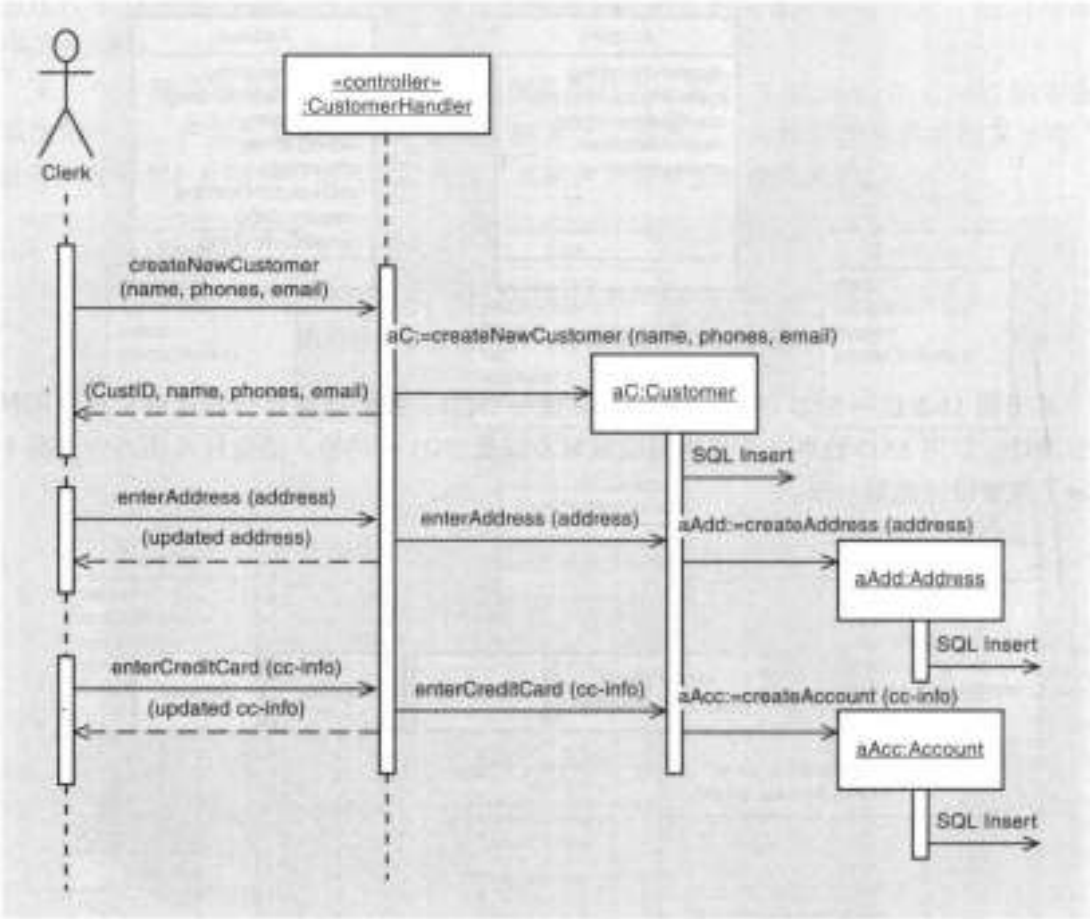


图 11-8 创建顾客账户用例的初步顺序图

在定义和创建消息时，我们必须首先决定消息的源头和目的地。源对象是这样的对象，在执行任务时它需要消息，或帮助执行一个任务从而启动消息。目的对象拥有帮助找到解决方案的消息，还要接收并处理消息。决定好初始和目的对象后，我们必须命名消息。当消息请求一个服务目的对象时，消息名称应当反映所请求的服务。例如，在目的对象中，当一定数量的需求被更新时，消息名应该表示更新数量的要求过程。同时需要注意的是，输入参数

提供了目的对象需要用来提供服务的消息。

让我们基于第 10 章和本章前面部分讨论的好的设计原则——耦合度、内聚度、对象职责和用例控制器——来分析一下解决方案。

用例控制器为内部对象和外部对象提供了联系，用例控制器连接到内部视图层。因此，问题域对象并不直接和视图层连接。:CustomerHandler 的职责是捕捉进入的消息，将它们分配到正确的内部域对象，以及给外部环境返回所需要的消息。

:Customer 的职责是创建自身并控制所有需要的更新。:Adress 和 :Account 对象创建自身并将自身存入数据库。耦合度比较简单，在分层结构中呈现垂直分布。这样看来，任务和对应消息的分配似乎符合了一个好的设计应该遵循的准则。随着设计扩展至三层，将会加入其他的问题。

11.3.4 “加入购物车”用例的初步顺序图

当转向多层设计时，让我们研究一下稍许复杂的初步顺序图样例。图 11-9 是加入购物车的活动图。你会记得图 3-16 这个用例包含其他三个用例，正如图 11-9 所示。通过包含其他用例，以这种方式设计用例使我们的解决方案聚焦于向购物车中增加商品的功能上。

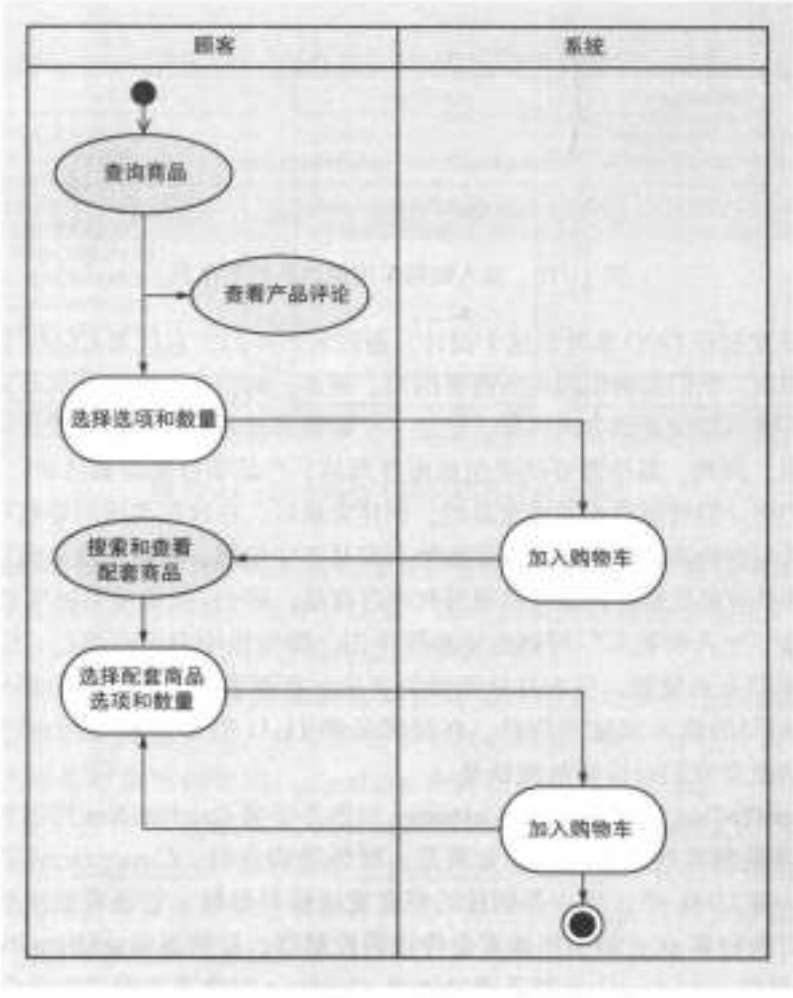


图 11-9　加入购物车用例的活动图

这个用例的 SSD 相当简单。图 11-10 展示了仅有的两条消息输入系统：增加一个商品以及增加一个配套商品。分析 SSD 时，注意在购物车中增加商品和在购物车中增加配套商品是相同的操作。仅有的不同是增加配套商品需要一个环形结构来为相同商品增加配套商品。因为这仅有的不同，我们能通过限制初始消息的方案来简化图表。（注意：搜索和查看配套商品需要一个另外的类——配套商品，但我们并不设计这一用例，因此在解决方案中这并不需要。）

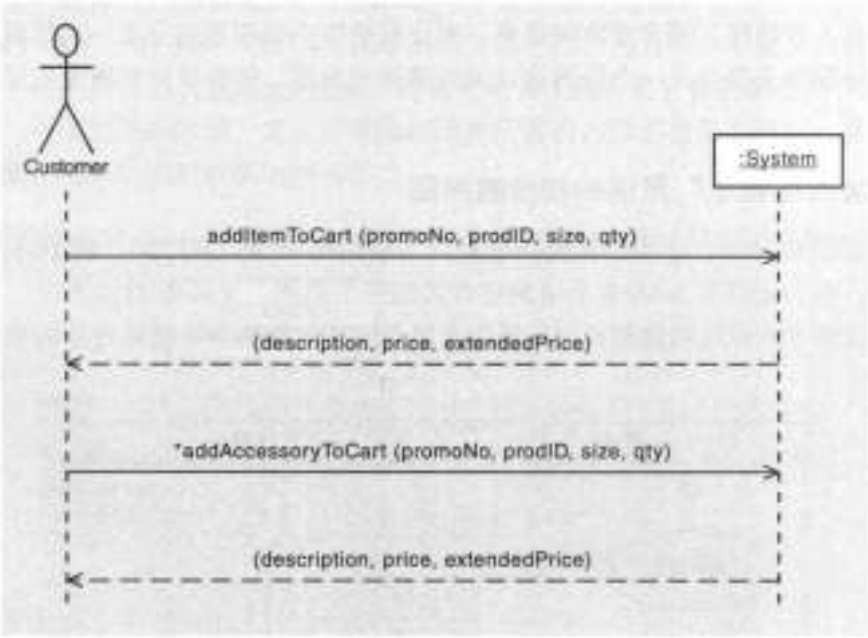


图 11-10 加入购物车用例的系统顺序图

我们通过开发初步 DCD 来开始这个设计。返回到图 4-21，它代表 CSMS 销售子系统的类图。通过这个图表，我们能确定用例所需要的类。顾客、购物车、购物车商品是必需的，因为用例会为顾客将商品添加至顾客购物车。创建一个购物商品时，系统需知道所需的是否有存货以及商品的价格。因此，其他需要的类包括库存商品、产品项目和促销活动。开发解决方案，我们可能会增加类，但目前看来类是充足的。创建完成后，在这些类间的导航可见性将会从控制器到顾客类再到购物类。购物车类可在购物车商品类中使用。购物车商品类应该对那些包含必要消息的其他类有可见性，例如产品项目和库存商品。图 11-11 展示了初步 DCD。

图 11-12 是“加入购物车”用例的初步顺序图。即使图中有很多信息，但请注意下半部分的图只是上半部分的复制，仅有几处细微的变化。在图表的左侧可以看到来自图 11-10 的 SSD 中的四个相同的输入和输出信息。在顶部是图 11-11 初步 DCD 的七个类的各个对象。图表中的其他信息是我们设计活动的结果。

输入 addItemToCart 消息来源于 :Customer 对象并指向 CartHandler 控制器对象。控制器对象决定这是否是顾客的首要商品，如果是，则传送消息给：Customer 对象来创建一个新的在线购物车。在 UML 中，当一条创建的消息发送给对象时，它通常直接绘制到对象框而不是生命线。对购物车 aCrt 的引用接着会传送到控制器，控制器将 addItemToCart 消息传送给 :OnlineCart 对象。:OnlineCart 对象通过创建 :CartItem 对象来处理这些消息。:CartItem 对象负责获得它的价格和描述并查看库存是否更新，这将通过传送合适的消息给对象来完成。

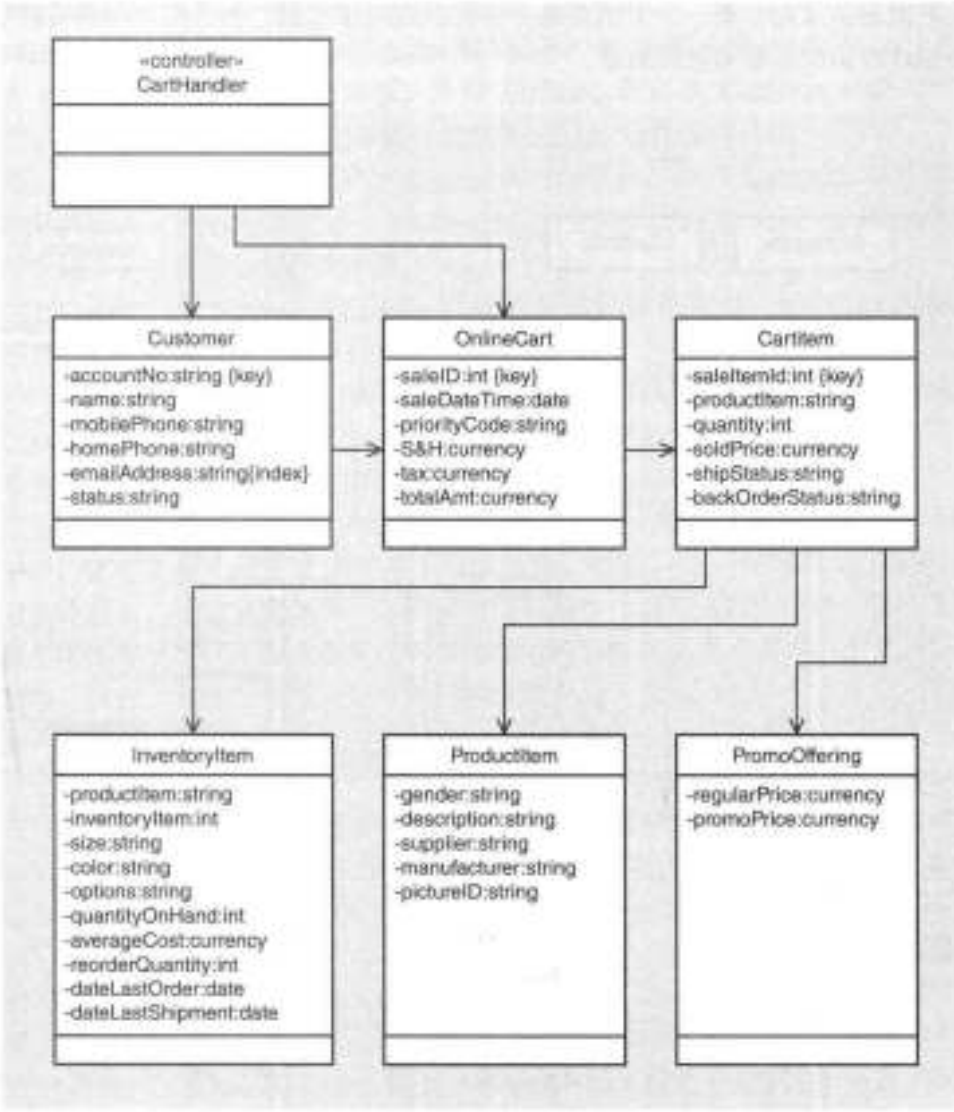


图 11-11　加入购物车用例的初步设计类图

顺序图的底部是相似的，除了增加的配套商品可能需要每个基本商品增加多层配件。因此，它表示在含有循环符号的矩形框中。注意，循环的源头在消息上标注了星号，如 SSD 所示。它分解成一个环形框来显示所有参与循环的封闭消息。

相关的设计原则是，若某个对象拥有其他对象，则该对象负责创建这些对象。例如，一个顾客创建了在线购物车，并在购物车中创建商品。这一方法可以确保购物车对象不是由于缺乏顾客对象而创建的。:CartItem 负责获得所属的消息。一个有趣的设计决定是 :CartHandler 是否应该直接发送 addItemToCart 消息给 :OnlineCart，或者应该发送给 :Customer 并允许 :Customer 对象转发给 :OnlineCart 对象。正如我们所指出的，一旦购物车被创建，控制器将直接发送消息给购物车。这个决定促进我们以更少信息来简化解决方案。这是基于好的设计原理的可靠设计。

在确定具体的消息以及源头和目的地传递的参数时，我们需要考虑一些关键问题。正如前面所说，一个重要的问题是哪个对象是消息的源头或起始程序。如果消息是一个疑问消

息，源头就需要消息的对象。如果消息是一个更新或创建消息，源头需要的则是控制其他对象或用来创建所需的必要消息的对象。

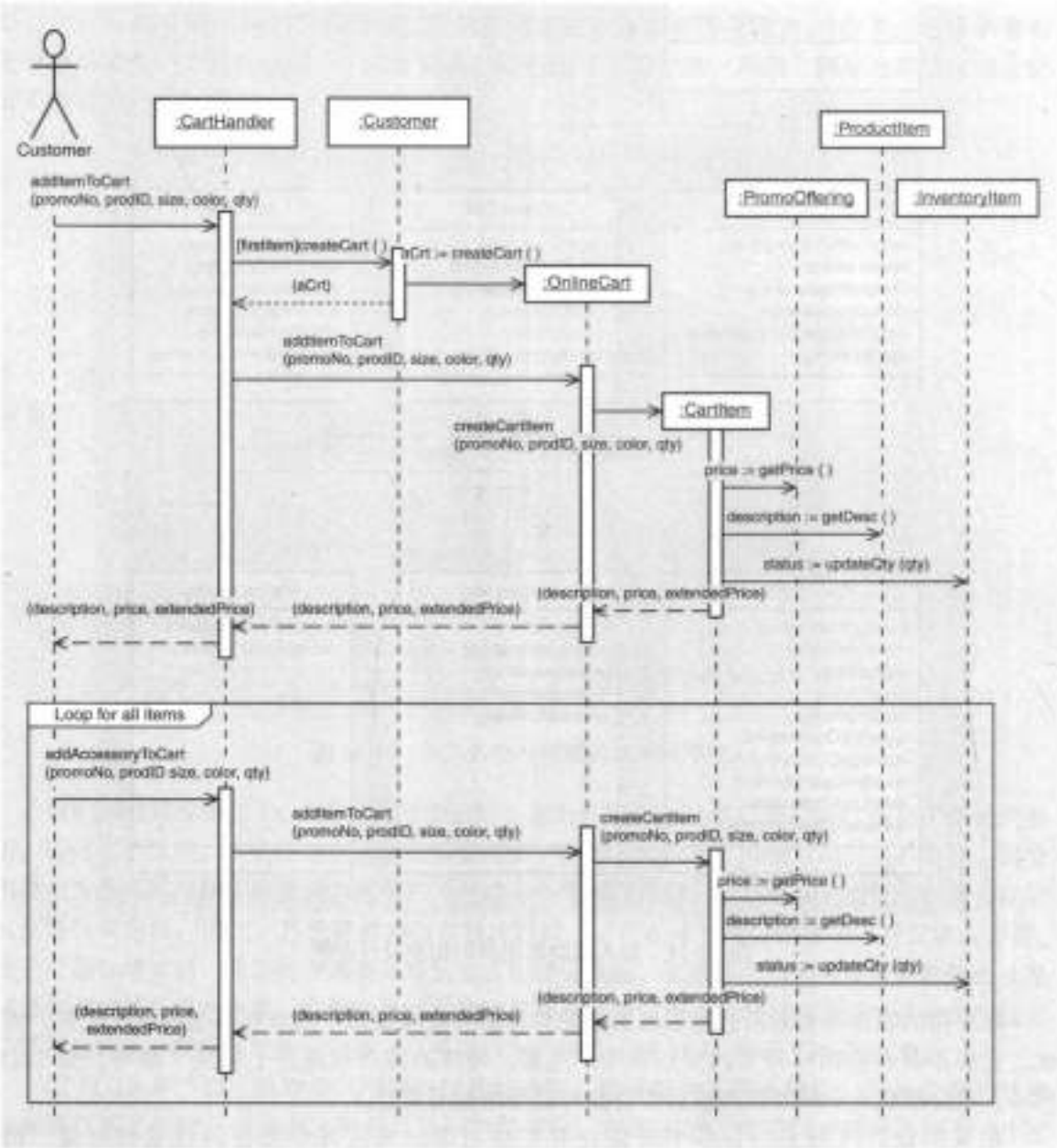


图 11-12　加入购物车用例的初步顺序图

另一个重要的考虑是导航可见性。为了将消息发送到正确的目的对象，源头对象必须对目的对象有可见性。设计的目的是为编程做准备。作为一个设计者，你必须知道程序如何工作并考虑编程问题。鉴于这两点考虑以及在先前段落中讨论的关于源头的考虑，我们决定所需的内部消息如下。对于每个消息，源头对象和目的对象已被确定了。

- creatCart()——:CartHandler 对象将知道是否接受了更早的消息来增加商品。如果没有，它将告知 :Customer 对象来创建一个购物车。

- createCart()——Customer 对象拥有 :OnlineCart 对象。
- addItemToCart()——从 :CartHandler 到 OnlineCart 的输入信息的上一个版本。因为 :CartItem 对象依赖于购物车，所以 :OnlineC 是创建 :CartItem 对象的逻辑对象。当 aCrt 返回时，控制器对先前返回消息的 :OnlineCart 有可见性。
- creatCartItem()——从 :Cart 到 :CartItem 的内部消息。因为 :CartItem 将负责获取对其汇集的数据，所以它需要 :PromoOffering、:ProductItem 和 :InventoryItem 的可见性。作为结果，这些主键将作为参数。
- getPrice()——来自 :PromoOffering 对象的获得价格的信息。:CartItem 开启了消息。因为它有主键值，所以有可见性。
- getDescription()——由 :CartItem 初始化的消息，以从 :ProductItem 获得描述。
- UpdateQty（qty）——检查手边是否有足够数量的消息。这一消息也初始化了手边数量的更新。:CartItem 对象初始化该消息。

只注重域类，我们可以设计用例的核心处理，而不必担心用户界面或数据库的使用情况。图 11-12 要更复杂，即使它只包含了域的对象。然而，这一设计提供了编程的可靠数据。借助设计模型，设计师能处理一个用例的所有要求，而不必担心代码。更重要的是，它使设计师在不需要丢弃代码或是重写代码的情况下可以修改或更正设计。在下一节中，我们将把视图层和数据访问层对象添加到电话订单场景中。

11.3.5　顺序图初步设计的指南和假设

从先前的两个例子中，我们能提取一些任务来帮助读者学习如何使用顺序图设计用例或者场景。该设计过程中也需要基于几点假设。（下面的任务不是按顺序完成的，而是在建立顺序图过程中根据需要完成的。我们把它们看作三个分离的任务并确保完成。）

指南

通过顺序图设计一个用例或场景涉及执行这些任务：

- 接收每条输入消息，并明确由这条输入消息产生的所有的内部消息。对于每条消息，要明确消息的目标，需要什么信息，哪个类（即目的地）需要这条消息，以及哪个类（即消息源）提供这条消息。定义是否有人和对象作为输入的结果被创建。
- 在处理每条消息的时候，一定要明确会受之影响的完整的类的集合。即从域类图中找出该消息所涉及的所有对象。在第 5 章中，我们学习了用例的前提条件和后续条件。在前提条件或者后续条件中罗列的任何类都应该包含在设计中。其他一些类即使不在前提条件或者后续条件中，如果满足下列条件，也应该包含到设计中。这些类包括被创建的类、创建用例对象的类、用例、期间更新的类以及为用例提供所需信息的类。
- 此外，要充实消息的结构，即添加迭代、真 / 假条件、返回值和传递参数。传递参数应该参考域类图的属性。返回值和传递参数可以是属性，也可以是类中的对象。

这三个步骤能够实现了初步的设计。修改与完善是必要的，但我们这里只专注于用例中的问题域类。

假设

开发初步顺序图的过程需要基于一些假设，包括：

- 技术假设：在第 3 章中确定业务事件的时候，我们首次提到了这个假设。同样，此处也需要该假设。在这里，并没有包括用户登录以及检测网络可用性的步骤。
- 内存假设：你可能已经发现，前面我们已经假设必要的类位于内存中，并可供用例使用。此外，我们不会理会对象是否在内存中生成这个问题。而在多层设计中，我们需要对该假设做适当修改，即需要包含在内存中创建对象的逻辑。
- 异常假设：初步顺序图假设不存在异常，其中没有处理查询的产品为空的逻辑。当然，以后可以陆续添加其他消息和异常处理逻辑。同样，我们也采用这种方式，对初学设计的人来说，更应该如此。

11.3.6 开发多层设计

初步设计图的开发只专注于问题域层上的类。在许多情况下，无论是对于自己或另一个程序员，可能有足够的文档来对解决方案编程。一旦你有了可靠的问题域类的设计，增加视图层和数据访问层是一个简单的过程。根据敏捷建模的原则，我们不希望创建图表，除非确实需要。我们也通常不保存设计文档，因为随着时间的推移，系统将被修改，它们也将成为过时的图表。敏捷建模表明，开发模型需要谨慎。有时，重要的是要看到全景，以及识别需要使用的视图层类和数据访问层类。对于那些实例，系统开发人员需要知道必要的时候如何做完整的设计。

每一个系统都需要视图层的类来表示应用程序在屏幕上的输入和输出。数据访问层类并不一定总是必需的。当业务逻辑很复杂，而且还需要与 SQL 语句访问的数据库相分离时，就需要数据访问层了。图 11-8 的序列图说明了在新顾客账户的创建中，如何将业务逻辑和数据访问逻辑结合起来。每个域层对象还包含了写入数据库中的 SQL insert 语句。我们可以做到这一点，因为无需业务逻辑要求。因此，图 11-8 显示了一个添加视图层后的两层设计。在本节中，我们将展示一个完整的三层设计来表示加入购物车的使用情况。

设计数据访问层

职责分离的准则是数据访问层设计背后的激发因素。对于更大、更复杂的具有三层设计的系统来说，所创建类的唯一任务就是执行数据库 SQL 语句，得出查询结果以及向域层提供信息。随着硬件及网络的日趋复杂，支持多层网络的多层设计诞生了。在多层设计中，数据库服务器位于一台机器上，业务逻辑在另一个服务器上，而用户界面在顾客机的桌面上。这种新的设计思路不仅能提高系统的健壮性，而且能提升系统的灵活性。

在大多数情况下，问题域类是**持久类**，这意味着它们的数据值必须存储于应用系统，即使本身并不执行。关系型数据库的全部目的是提供这方面的能力，使问题域对象持久化。在数据库中执行 SQL 语句，使程序从数据库中访问一条或一组记录。使用关系数据库与面向对象程序的问题之一是编程语言和数据库 SQL 语句之间存在少量不匹配。例如，在数据库中，链接表需使用外键（见第 12 章），如将有 CustomerID 的购物车作为一列，从而能够和相关加入的顾客结合。然而，在面向对象的编程语言中，导航往往在相反的方向（即顾客类可能有一个数组参数指向的在线购物车对象，这是在计算机内存中的，并正由系统处理）。换句话说，设计类没有外键。

在这一章中，我们选择了一个简单的方法来完成设计，目的是教授基本的思想，而不纠缠于数据库访问的复杂性中。假设每个域对象在相关数据库中都有一个数据表。（在更复杂的情况下，表必须相结合来提供存储器中的正确对象。）使业务逻辑层与数据访问层关联起

来的方法有多种（提供不同的设计）。一种方法是在每个业务对象的内部调用数据访问对象，从而获取必要的信息来完成实例化工作。另外一种方法是给数据访问对象发送消息，然后数据访问对象读取数据库并完成新对象的实例化工作。两种方法都可行，并且都不失为好的设计。

“加入购物车”用例的数据访问层

要设计数据访问层，我们就不能继续假设对象是在内存中自动生成的了。现在，我们抛弃前面提到的内存假设。在设计用例时，我们需要额外的消息来将数据存入数据库中，以及获取数据来初始化类。因为两套消息几乎相同，所以我们根据图 11-12 将只说明上半部的消息。

图 11-13 是包含所有数据访问类的用例设计图。但要注意的是，来自图 11-12 的内部消息依然存在。然而，与每个问题域对象相联系的是一个数据访问类，且与每个初始内部消息相联系的，是通过数据访问对象从数据库中获取数据或将数据保存到数据库中的附加消息。

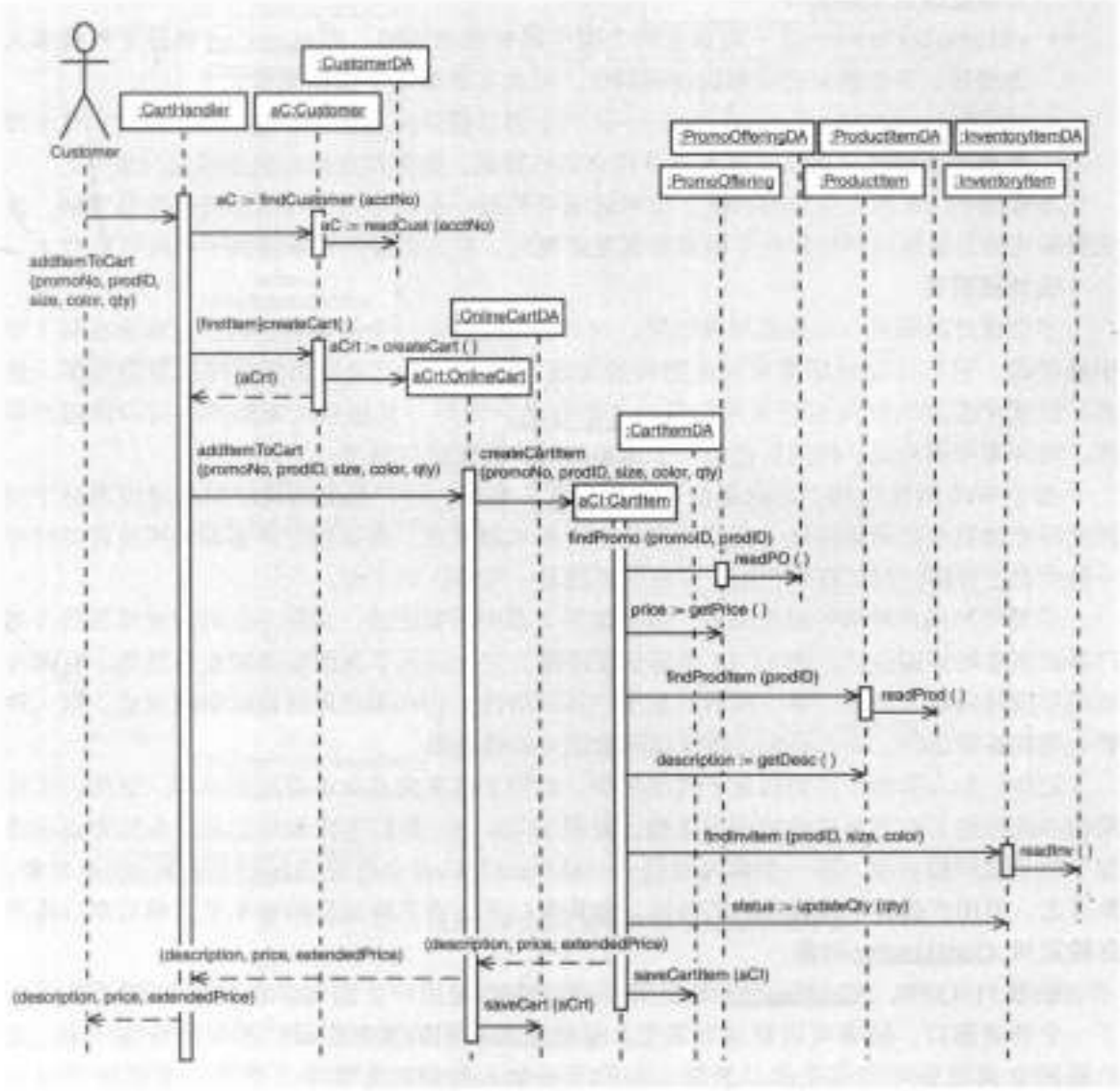


图 11-13　含有数据访问层的加入购物车用例的顺序图

在这个过程中，确保源对象具有导航可见性且消息可以被发送到目的对象是很重要的。我们假设但不显示数据访问对象具有全局的可见性。（在编程课中，你将了解工厂类或单例类通常是由全局方法设计的。）在合适的问题域创建对象后，对于它的引用则被返回到需要可见性的对象。仔细看图 11-13，注意，每个对象发送消息到另一个对象时必须对那个对象有导航可见性。当你开发设计解决方案时，记住这个重要的设计点。

理解图 11-13 的有效的方法是首先从图 11-12 初步顺序图的内部消息开始。让我们逐一回顾，并看看需要什么改变。

- [firstTime]createCart——处理程序将消息发送到一个顾客对象来创建一个购物车。首先，它需要确保内存中有一个顾客对象在。它将 findCustomer 消息发送给 aC:Customer 对象来从数据库中寻找并创建自身。它通过发送消息给 :CustomerDA 对象来阅读数据库并回锁给合适的顾客对象。在这之后才可以传送 Cartmessage 给 aC:Customer。同样，在这一过程的结尾，aCrt:OnlineCart 会发送消息给数据库对象来保存数据至数据库。
- addItemToCart——这一消息在两个图中最初是相同的。当 aCrt:Cart 被建立并被添入数据后，消息被发送到数据访问对象，以此在数据库中保存数据。
- getPrice，getDesc，updateQty——这三个消息都访问或更新着数据库。因此，每个都要求先前的消息来从数据库中寻找合适的数据，数据被存储在内存的域对象上。

尽管图 11-13 看上去显得拥挤，但问题域类的每一条消息则使得图形变得容易理解。首先要知道的是数据访问对象对于检索数据是必要的，它为必要的对象提供了导航可见性。

设计视图层

多层设计的最后一步是添加视图层。用户界面设计是一个复杂的过程，正如你在第 7 章中所学的。它往往以顺序图来预示用例的实现，因为它比记录的用例顺序图复杂得多。然而，以顺序图表的视图层记录用户界面常常有助于设想与其他系统对象类的用户界面的集成。对许多用例来说，视图层仅由一个简单的用户界面窗口组成。

基于 Web 的接口和一个内部的网络接口这一事实，用户界面的设计和集成视图层序列图变得更加复杂。幸运的是，浏览器正在变得越来越复杂，所以现在许多新的系统可以只有一种类型的界面。设计有多个用户界面的系统是一项艰巨的工作。

尽管增加用户界面听起来简单，但正如第 7 章中所描述的，实际上它的完成需要结合用户界面表单的详细设计。图 11-14 是部分顺序图，它仅展示了视图层类和控制器类。有两种视图层设计的输入源头。第一种当然是用户界面组件，它在用户界面设计期间完成。第二种要么是初步顺序图，要么是包含数据访问类定义的数据图。

记住，加入购物车用例包含了其他用例，以用于搜索商品和查看配套商品。显然，所有用例共同构成了丰富而高效的用户体验。在图 11-14 中，我们为搜索商品和查看配套商品增加了两个视图层对象。第一个输入信息——addItemToCart 会经过 :SearchItemWindow 对象。换言之，当用户找到了他喜欢的东西后，会从窗口开始将其添加到购物车里。稍后窗口将消息转发到 :CartHandler 对象。

在图 11-13 中，:CartHandler 对象要求有一个确定用户。因此，在我们的例子里展示了一个登录窗口，顾客可以登录到系统。记住之前讲到的技术假设，不要忽略这一步。无论哪种方式都是可以接受的。最终，它需要被加入到解决方案中，因此，我们把它包含进来。

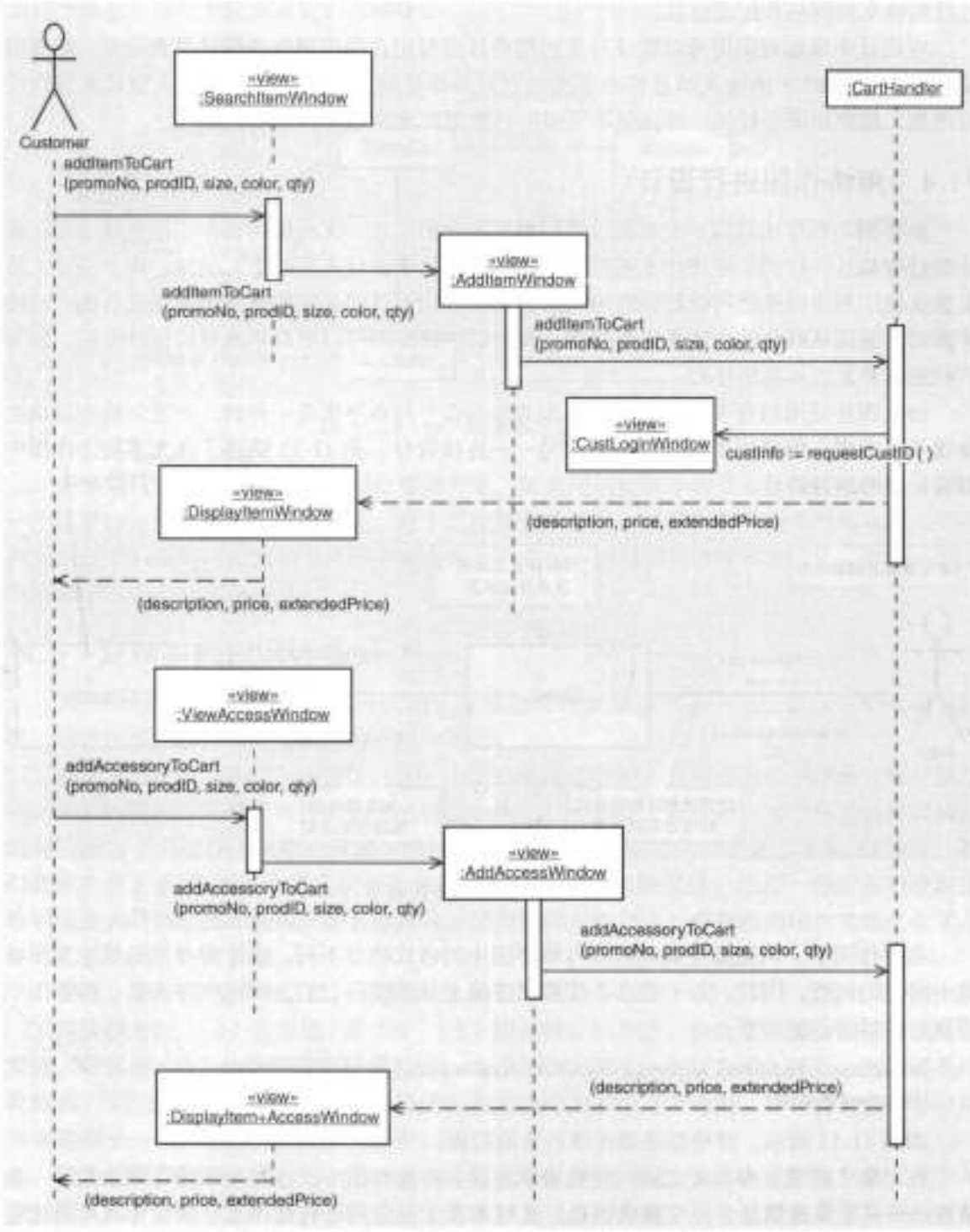


图 11-14 含有视图层的加入购物车用例的部分顺序图

一旦商品被加入购物车，另一个窗口陈列展示了增加新商品的结果。根据用户界面的设计，这个窗口可以展示单一的新增商品或者总购物车。

接下来的三个视图层对象是 :ViewAccessWindow、:AddAccessWindow，和 :DisplayItem+AccessWindow，功能方式与其他视图层对象相同。仅有的不同点是数据包含了已加入

在线购物车的商品和配套商品。

在设计中增加视图层可以验证开发的用户界面与用户的应用程序设计是否一致。所有确定和记录在 SSD 中的输入消息都必须由用户界面来处理。如果消息没有输入窗口或窗口没有消息，你就知道设计的一部分是不完整的并且需要更多定义。

11.4 用协作图进行设计

协作图与顺序图都属于交互图，它们捕捉同样的信息。无论使用协作图还是顺序图，设计的过程都是一样的。使用什么模型来设计主要取决于设计人员的个人偏好。许多设计人员更喜欢使用顺序图来进行设计，因为用例描述和对话设计都是按照顺序步骤来进行的。而协作图则更强调从耦合的角度来审视用例。在会议中协作图可以更简单地草拟设计思路，因为它们更容易改变和重新排列。

协作图所使用的有关参与者、对象和消息的符号与顺序图是一样的。而生命线和活动生命线不再出现。但使用了一个不同的符号——连接符号。图 11-15 描述了在大多数协作图中都会出现的四种符号。

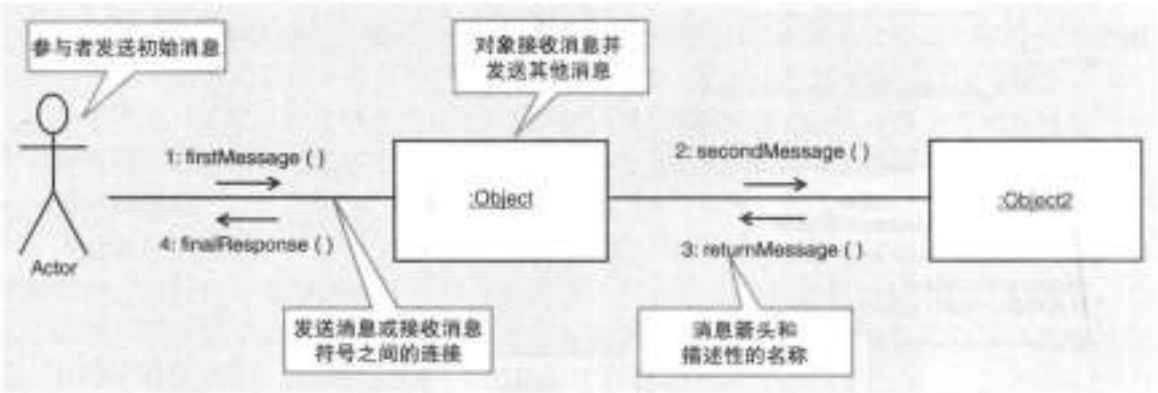


图 11-15 协作图中的符号

在协作图中，消息描述器的格式与顺序图中的格式稍有不同。由于没有生命线来表示场景中时间的流逝，因此，每个消息都按照顺序编上号来说明它们之间的次序关系。协作图中消息描述器的语法如下：

[true/false condition] sequence-number: return-value := message-name (parameter-list)

如图 11-15 所示，冒号总是跟在序列号的后面。

在对象之间或在参与者之间的连线表示连接。在协作图中，连接表示两个对象共享一条消息——一个发送消息，一个接收消息。连线本质上仅仅用于传递消息，所以你也可以把它们想象为用于传输消息的线路。消息上的数字代表了消息的先后顺序。使用分层点编号方案消息时会依赖于以前的消息。

图 11-16 是“加入购物车”用例的协作图。该图仅仅包含域模型对象，而不含有视图层和数据访问层。使用协作图来进行多层设计与使用顺序图一样高效。

将协作图与顺序图相比较，很容易就可以看出，协作图主要关注对象本身。描绘协作图能有效地统观协同工作的对象。同时你会发现，在顺序图中确定消息的顺序比较困难，你必

须找到消息的编号才能知道它们的顺序。

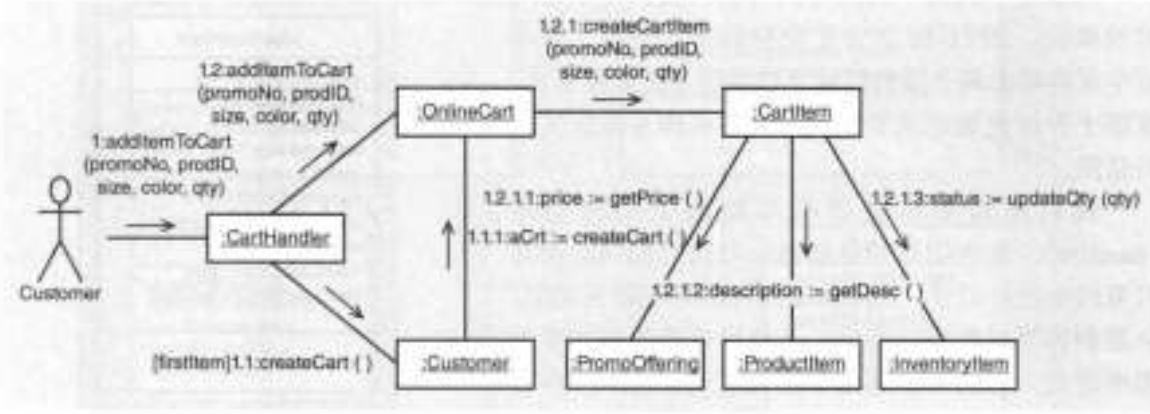


图 11-16　加入购物车用例的协作图

许多设计人员用协作图来草拟出解决方案。如果用例比较小而且不是很复杂的话，那么一个简单的协作图就足够了。但是，对于比较复杂的情况，就需要顺序图来可视化消息的流向及顺序了。通常，在同一个说明里面会夹杂两者：一些用例用协作图来描述，而另一些用例用顺序图来表示。

11.5　更新和打包设计类图

现在每层都可以开发一个设计类图。在视图层和数据访问层中，一些新的类必须要清楚。域层也有一些为用例控制器而添加的新类。

在图 11-15 中，我们为域层设计了一个初步的设计类图，它基于的是创建顾客账户的用例。那个时候还没有开发出任何的方法特征。既然已经创建了顺序图，方法信息就可以添加到类里面去了。而且，在顺序图建立的过程中，导航箭头也要同时更新。在第 10 章中，我们简单介绍了基于 CRC 卡确定的任务来在类中创建方法名的思路。然而，那时我们没有足够的信息去严格定义方法特征的名称以及返回类型和参数列表。顺序图的用例实现产生了足够的信息来严格定义方法。

首先，基于顺序图的信息，更新 DCD 来添加方法特征。大部分类中主要有三种方法：（1）构造器方法；（2）数据读 / 写方法；（3）用例特定的方法。构造器方法会创建新的对象实例，数据读 / 写方法读取或更新属性值。由于每个类都有一个构造器，并且大多数都是用数据读 / 写方法，因此把这些方法的特征包含到设计类图中是可选的。第三种方法——使用具体案例——一定要包含在设计类图中。

就像在顺序图中一样，每一条消息都有源对象和目的对象。如果一条消息发给了某个对象，那么这个对象必须准备好接收这个对象并初始化一些行为。这个过程无非就是调用对象上的某个方法。换句话说，顺序图上出现的每条消息都需要目的对象的一个方法。事实上，消息的语法很像方法的语法。因此，给设计类图添加方法特征的过程就是浏览每一幅顺序图并找到发给类的消息的过程。每条消息对应着一个方法。

我们通过一个基于 InventoryItem 类的例子来说明如何添加方法特征。在图 11-14 中，两条消息被传送到 InventoryItem。第一条消息是一个构造器，另外一条消息 updateQty（qty）用于更新。返回的更新消息可以作废，也可以作为一个字符串值返回成功状态。我们需要添

加此消息对应的方法特征。将这种方法添加到 InventoryItem 类，如图 11-17 所示。

对域层中的每个类及用例控制器类执行上述同样的操作。图 11-18 包含了完整的设计类图，即本章中所描绘的两个用例的域层的类图。正如你所见，该图十分出色地记录了设计类，并将作为系统实现的蓝图。

我们还需要给域层类添加两个用例句柄（handler）。加进附加的导航箭头可以对用例控制器可见的类进行记录。还增加了额外的导航箭头记录，从控制器类的种类便可见。其他导航箭头在初步类图中定义，它们对于这两个用例来说足够了。在额外的用例开发中，我们将增加更多的导航箭头。

图 11-17 带有方法特征的 InventoryItem 类的设计类图

11.5.1 包图——将主要部分结构化

UML 中的包图是一个高层次的图，它使得设计人员可以将相关组中的类联系起来。前面几节讲述了三层设计，包括视图层、域层和数据访问层。在交互图中，每层中的对象都画在同一幅图中。但是有些时候，设计人员需要记录不同层次中对象间相互关系的相同点和不同点——可根据分布的处理环境对对象进行分组。将每层都表示为一个单独的包就能够捕捉到这些信息。图 11-19 分析了这些层次是如何记录的。

类根据它们所属的层被放在合适的包里面。在相关图中建立类的时候，它们就与不同层联系起来。为了创建包图，我们只需从每个用例的设计类图和交互图里提取信息。图 11-19 只是包图的一部分，因为这些包仅包含本章所创建的用例交互图中的类。

包图使用的另一个符号是虚线箭头，它表示**依赖关系**。箭头的尾部连接着有依赖性的包，而箭头连接着被依赖的包。依赖关系在包图、类图甚至交互图中都有所应用。可以这样理解依赖关系，如果其中的一个元素发生了变化（被依赖部分），那么另一个元素（依赖部分）也一定会发生变化。依赖关系可以存在于包与包之间，或者包中的类与类之间。图 11-19 意味着视图层中的两个类依赖域层中的类。因此，如果 ProductItem 类发生变化，那么 SearchItemWindow 类也应该响应 ProductItem 类的变化。然而，反过来就不一定了。视图层中的变化往往不能传递给域层。

有关依赖关系的两个例子如图 11-19 所示。第一个是类与类之间，我们已经讨论过。另一个例子不是很详细，描述的是包与包之间的依赖关系。图 11-20 说明视图层和域层都依赖于数据访问层。对于一些简单的对数据库的查询，视图层可以直接访问数据层，无须参与任何域层。因此，数据结构的变化体现为数据库访问层的变化，通常会引起域层和视图层的变化。

包图也能通过嵌套来描述不同层次的包。图 11-20 说明包和它们所包含的一些类都是订单输入子系统的一部分。RMO 系统可以分割成一些子系统。一种记录这些子系统的方法就是使用包图。这种方法的好处是，可以将不同的包交由不同的开发小组完成。依赖关系箭头可以帮助开发小组确定何时需要相互交流，以确保系统的完整性。

综上所述，包图可以用来表示各部分的相关性和依赖关系。在通常情况下，我们使用包图来关联类或者其他的系统组成部分，例如网络结点。前面的图说明了包图的两个应用：将系统分割成子系统和表示包的嵌套。

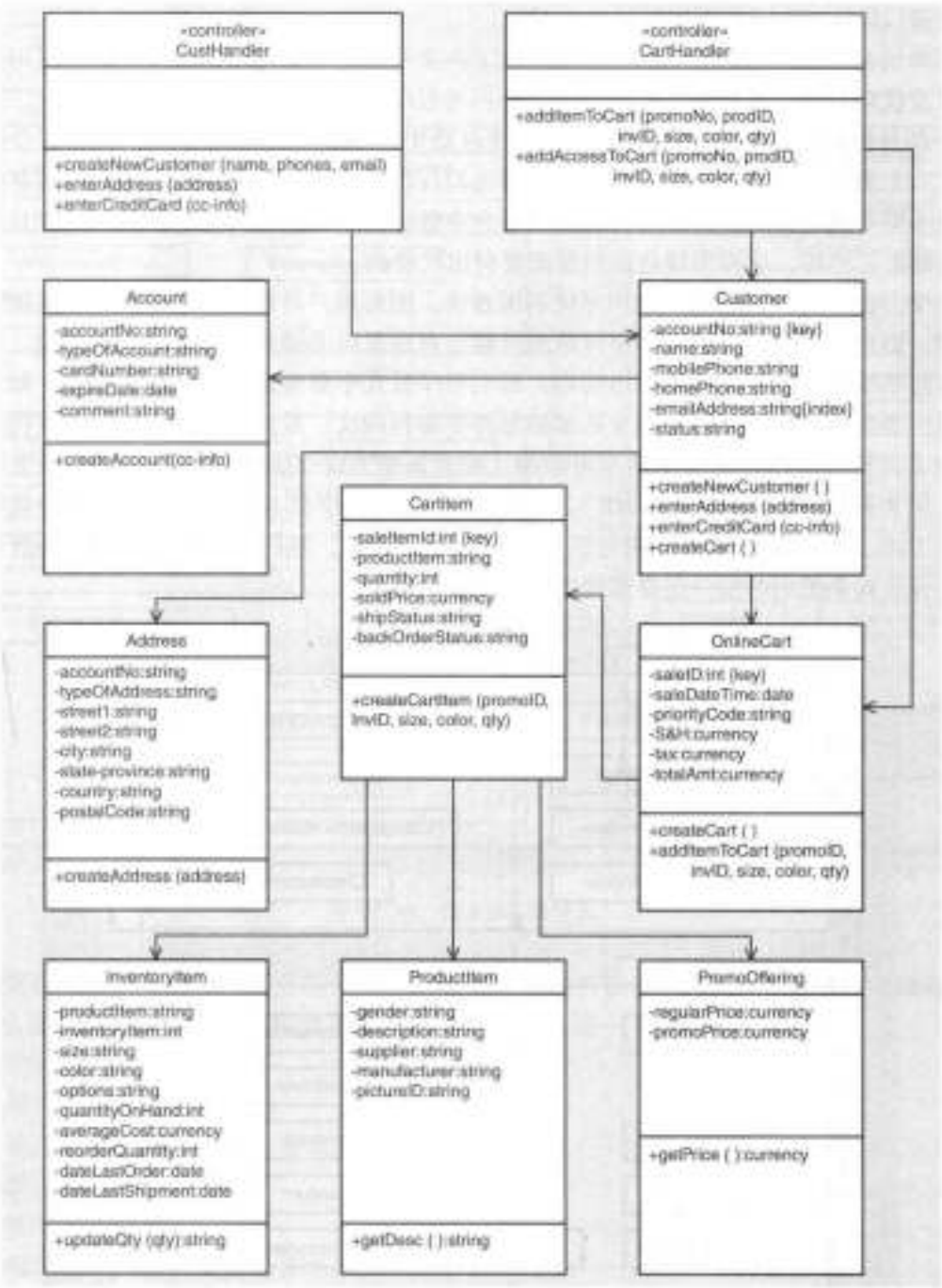


图 11-18　更新后的域层设计类图

11.5.2　三层设计的实现问题

有了设计类图、交互图和包图，编程人员就可以开始创建系统的各个组成部分了。实现是指通过语言（Java、PHP 或 VB、C# 等 VS 语言）编程来搭建系统。过去几年，人们开发出了功能强大的集成开发环境（IDE）工具来帮助编程人员搭建系统。这些工具包括：与 Java 有关的 Jbuilder 和 Eclipse，与 Visual Basic 及 C# 有关的 Visual Studio，与 C++ 有关的 C++Builder。这些工具从较高的层次上提供了编程支持，特别是在搭建视图层类（如系统窗

口以及窗口组件）的时候更加明显。

不幸的是，正是这些工具开发人员养成了许多不良的编程习惯。轻松建立图形用户界面窗体以及代码自动生成所带来的便利，使得一些程序员将所有的代码都放到窗体里。每个窗体组件都自带几个事件，这些事件的代码是自动生成的。这样程序员使用 IDE 就会很容易地建立一个窗体，工具自动生成类定义，而他们仅仅需要插入业务逻辑代码，不需要定义新类，也不需要其他代码。许多这样的工具还含有数据库引擎，这样整个系统就可以只使用窗体类来建造。然而，这样走捷径以后是需要付出代价的。

这个方法存在的问题在于维护系统的困难性。图形用户界面类中散落的代码段很难定位和维护。而且，当用户界面类需要升级的时候，程序员也必须找到并升级业务逻辑。如果基于网络的系统需要升级到包括网络前端，那么程序员几乎要重建整个系统。或者，如果需要两个用户界面的话，那么所有的业务逻辑都需要编程两次。最后，如果不使用产生代码的工具，那么保持系统的通用性就是不可能的。随着新版 IDE 工具的发布，这个问题会进一步恶化，因为新版 IDE 与原有的 IDE 可能不兼容。许多程序员必须完全重写系统的前端，因为新版 IDE 工具产生代码的方式与原有 IDE 不同。因此，我们建议未来的准分析员和程序员们在开发新系统的时候一定要遵循设计准则。

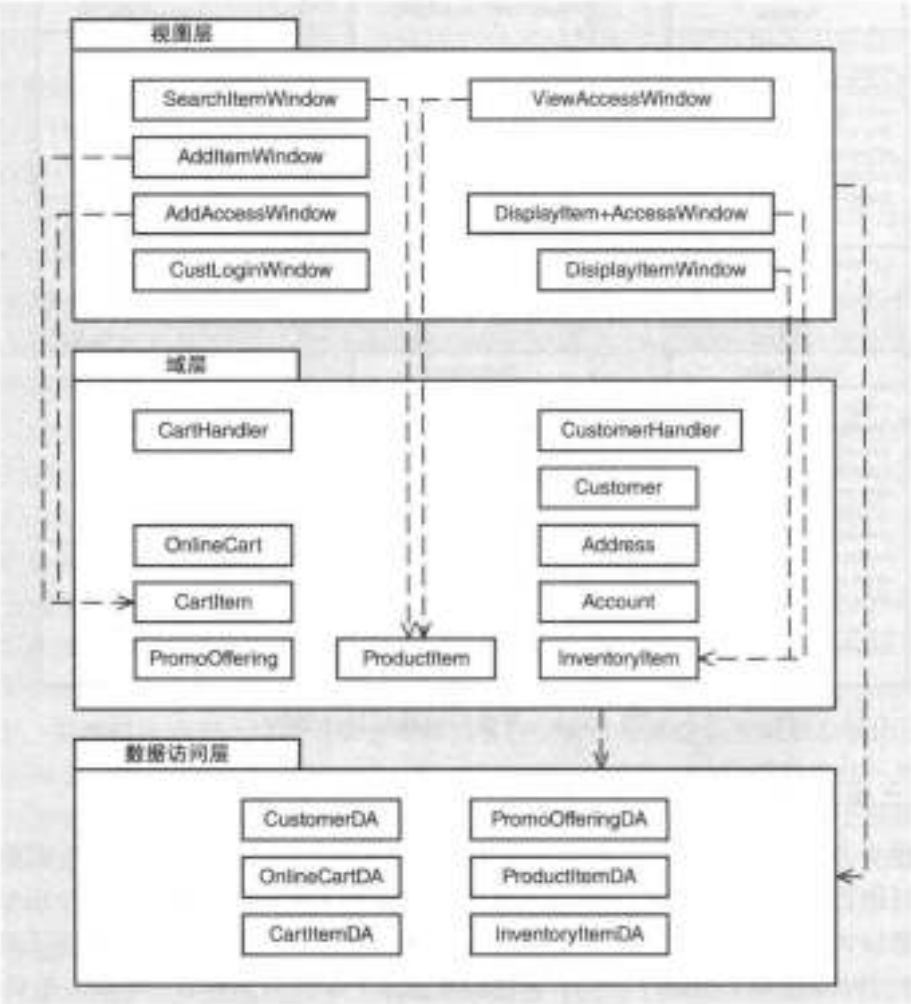


图 11-19　RMO 三层包图的部分设计

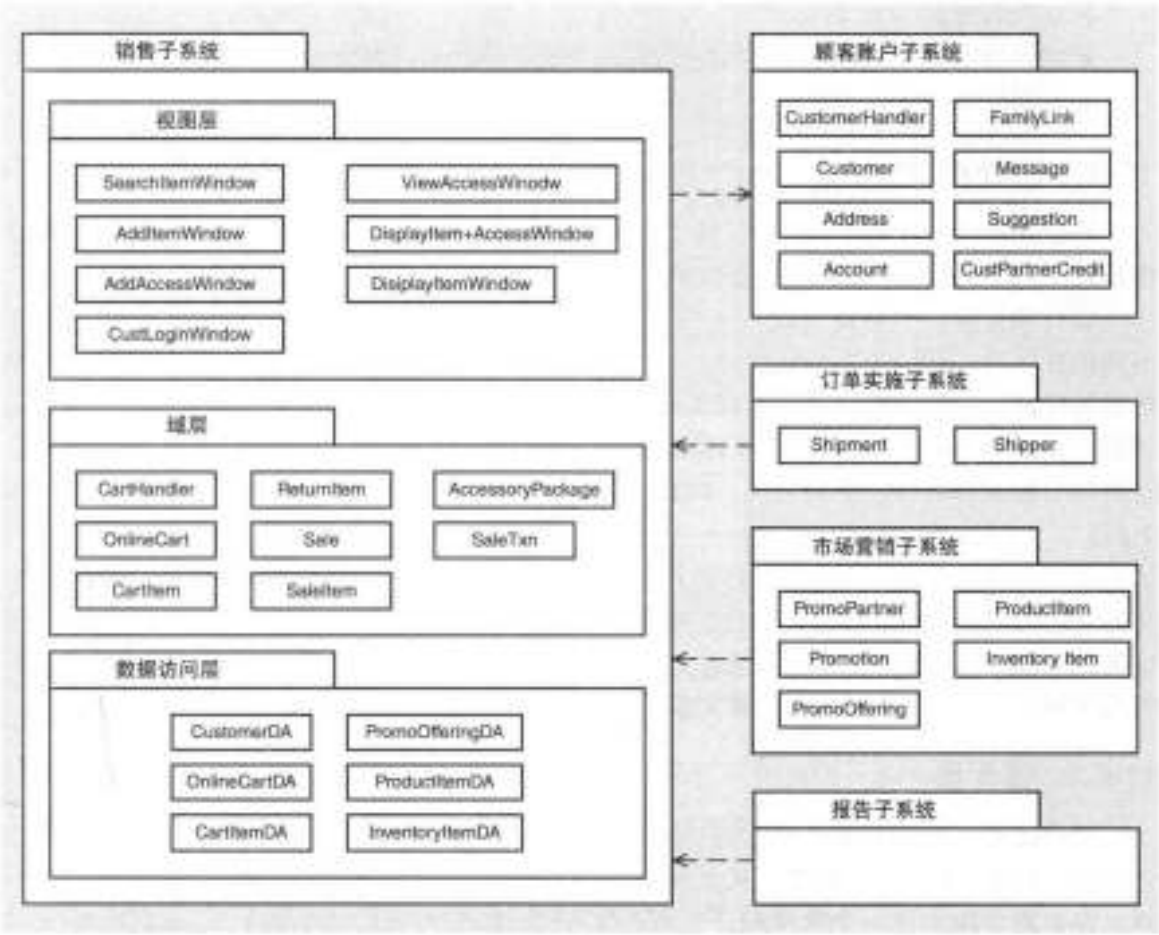


图 11-20　RMO 子系统包

根据设计准则中的对象职责部分，可以定义每一层的任务。如果按照这些规定来编写代码，那么新系统在其生命周期内就很容易维护。我们总结一下每层的主要任务。

视图层类完成以下任务：

- 展示电子表单和报告。
- 捕捉输入，例如单击、滚动和键盘输入等事件。
- 显示数据字段。
- 接收输入数据。
- 编辑并校验输入数据的合法性。
- 将输入数据传递给域层类。
- 启动与关闭系统。

域层类完成以下任务：

- 创建问题域（持久）类。
- 以适当的逻辑处理所有的业务规则。
- 准备持久类以便数据库存储。

数据访问层完成以下任务：

- 建立并维护数据库之间的连接。

- 包含所有的 SQL 语句。
- 处理结果集（SQL 语句的执行结果），并附给合适的域对象。
- 适时断开与数据库的连接。

11.6　设计模式

基于良好设计原则的系统不仅易于开发和初次投产运行，它们也更容易维护。这种概念涉及对象职责、耦合、内聚、变量保护以及第 10 章介绍并贯穿第 11 章讨论的间接。

设计模式和两个特定模式——三层设计和用例控制器也是我们所熟悉的。模式存在于不同的抽象层。在一个具体标准中，一个模式也许是由设计者用代码编写的类定义。在最抽象的标准中，一个模式也许仅仅是解决问题的方法。例如，多层设计模式趋向于更抽象并且建议最好是单独的分为三个层类的系统功能；GUI 逻辑被放置在一组可分开的视图层，且区别于域层和数据访问层的类。因此，多层设计是比特殊的解决方法更好的建立系统的方法。

用例控制器模式更为具体。它定义了一个特定的一类或多类作为所有来自环境的传入消息的交换机。通过所有的模式，有多种方式来实现控制器模式。单个控制器类可以从视图层定义，从域层来处理所有消息。或者，一个类可以被定义为单个用例或两者的某种组合。不管具体做法是什么，控制器模式确实需要一个单独的、指定的类。

11.6.1　适配器

我们从适配器模式开始，因为它的概念简单明了。适配器是“变量保护”和“间接”设计原理的很好的例子。适配器模式大致类似于用于国际旅行的电源适配器。当你去英国旅行时，你可能会决定带一个吹风机，它有一个用于转换 110V 到 220V 的开关，所以你可以在任一电压下运行。然而，电源线的插头底部有两个插脚，而英国的壁式插座却有三个插脚。

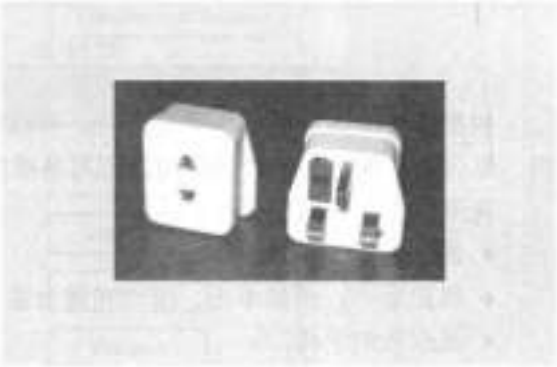


图 11-21　电源适配器

图 11-21 展示了你可能用过的典型的电源适配器。

适配器设计模式就像电源适配器一样运作：它在现存的系统中插入一个外部类。外部类的方法特征和从系统内被调用方法的名称不同，所以插入转换适配器类可将从系统内调用的方法转换为外部类中的方法名称。

图 11-22 描述了适配器模式的细节。样例图中有四个 UML 类。类 System 代表整个系统。系统中的类使用如 getSTax() 和 getUTax() 的方法名来访问税收程序。TaxCalculator 类有 findTax1() 和 findTax2() 两个方法名。中间的两个 UML 类代表适配器。顶部中间的类符号代表一个接口类。接口对于指定方法是有用的；尽管不是必要的，但这是一个指定和强制使用正确的方法名称的简单方法。适配器类接着继承这些方法，并提供了对于这些方法的方法逻辑。每个方法主体简单地扩展了最终方法名 findTax1() 和 findTax2() 的称呼。换句话说，它们“改写”或者“翻译”出一个个不同的方法名。

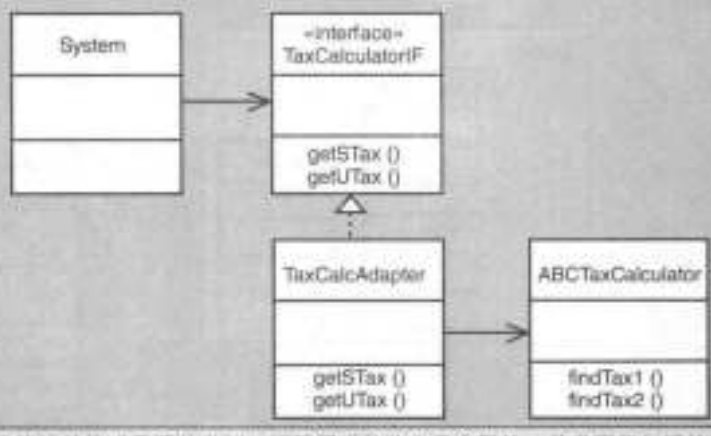
名称	适配器
问题	一个类必须通过另一个标准类或者购买类而被替代。替换的类已经有了一套定义好的与原来的方法不同的方法特征。你怎样以最小的影响连接新类，从而不必在整个系统中修改新类中的方法名？
解决方案	写一个新的适配器类，这个类作为初始系统和被替换的类之间的连接。这个类的方法特征与初始类相似（也和系统所期望的相似）。每个方法要求替换类有正确的方法特征。其实，因为它“适应”替换类，所以它看起来像初始的类。
样例	在 RMO 系统的许多地方，需要购买类库来提供特别处理。这些库可提供特殊服务，比如计算税务和运输包裹的费用。有时，这些服务库被更新成新的版本。有时，一个服务库甚至被替换给一个完全不同的供应商。RMO 系统的员工通过在每个替换类前部署一个适配器间接地从设计规则中获得保护。 
优点和结果	被改造的类能被替换成需要的类型。适配器类的改变是有限的，并且不能与系统连锁。 定义了两个类，一个界面类和一个适配器类。 已通过的参数可能变得更复杂，并且很难限制适配器类的改变。

图 11-22　适配器模式样例

当你熟悉了设计模式后，你会发现它有多种用途。这是一个强大且讲究的解决方法，它使得系统更可以持续。有经验的设计者会频繁使用模式——对于外健和需要频繁更新的内部编写的类。这是一个阻隔系统频繁变化类的好方法。

11.6.2　工厂

在细节设计的讨论中，我们常表示需要工具类，包括数据访问对象或者控制器类。适配器在适配器模式的情况下也是一个实用工具类。什么类需要创建这些工具对象呢？在多数情况下，域类来创建它们并没有意义，因为这不是域类的责任之一。面向对象程序中一个流行的解决方案是有一些工厂类。换句话说，这些类通过工具类来实例化对象。

例如，一个可执行的顾客对象可能需要编写代码。如果工厂类是由静态方法设计的，这代表它们有全局可见性，顾客对象可以对工厂说：“给我一个关于顾客表的数据访问对象的引用。”代理商将会创建一个新的数据访问对象，并回馈参考。如果顾客数据访问对象已经存在于内存中，则简单地返回引用。顾客对象不需要关心如何创建对象来访问数据库。它只是使用任何传递给它的东西。这减少了耦合，增加了内聚，并将责任分配给了正确的类。图 11-23 是工厂类的样例。

工厂类有私有属性来保存所创建的数据对象的引用。当请求获得一个数据对象的引用时，该方法简单地检查了属性是否为空。如果是这样，它就创建了一个新对象，将引用放置

于属性，并返回该值；否则，它只需返回带有已有引用的参数。

名称	工厂或工厂方法
问题	谁应该为创建了不明确属于问题域类的工具类型对象负责？这些工具类可被系统内的不同地方访问，因此许多类的指定对象需要被实例化。
解决方案	创建一个人工工厂类。它的责任仅仅是实例化工具类。在许多情况下，一个特殊的工具类只能使用一种实例。因此，所有的类需要通过工厂来访问其他类。工厂必须确保每个类只创建一种实例。

样例	RMO 系统的几个位置需要从订单对象中获得数据并获得 Order_DA（数据访问）对象的引用。Order_DA 对象不知是否已经被实例化。定义一个数据访问工厂并创建一个界面。请求的对象使用在界面中被定义的方法以获取 Order_DA 对象的引用。然后它就能读取订单的数据库。
	:System
	Customer
	Order
	Shipment
	«uses»
	Order_DA
	«requiresDA»
	«creates»
	«interface»
	DAFactory_IF
	getOrder_DA ( )
	getCustomer_DA ( )
	getShipment_DA ( )
	public synchronized Order_DA getOrder_DA ( ) {
	if (myODA == null) {
	myODA = new Order_DA ( );
	}
	return myODA;
	}
	DAFactory
	-myODA: Order_DA
	-myCDA: Customer_DA
	-mySAD: Shipment_DA
	getOrder_DA ( )
	getCustomer_DA ( )
	getShipment_DA ( )

优点和结果	使问题域类拥有更高的内聚性。 使业务逻辑和数据层之间的耦合更少。 更小、更加可维持的类。

图 11-23　工厂方法模式样例

11.6.3　单例

一些类必须有确定的一个实例——例如，一个工厂类或是主窗口类。因为这些类仅从一处来实例化，所以很容易限制逻辑来创造仅仅一个对象。

其他类必须有精确的实例，但不能简单地只通过一处顾客调用来控制。依赖于系统逻辑流，一个特定的类可能从多种位置得到一个实例。然而，仅仅需要创建一个实例，所以第一个需要它的类创建它，其他的每个类则使用最初创建的。通常，这些类是管理系统资源的服务类，例如数据库连接。事实上，刚刚描述的工厂类是一个很好的样例。这一常见的问题有一个标准解决方案：单例模式。

图 11-24 展示了单例模式的描述模板。单例模式提供了类本身只控制一个实例创建的解决方案。

名称	单例
问题	一个类只允许使用一种实例。这个实例可能被系统的许多地方调用。第一个引用应该得出一个新实例，后来的尝试应该返回一个已经被实例化的对象。你怎样定义一个类使其只创建一种实例？
解决方案	单例类有一个静态变量指向它自己的一种实例。所有类的构造器都是私有的，可以通过一种或多种方法访问，比如 getInstance()。GetInstance() 方法检查变量，如果它为空，则需要构造器；如果不为空，会返回被引用的对象。

样例	在 RMO 系统中，数据库的连接是通过一个名为 Connection 的类。然而，就效率而言，我们想要每个桌面系统与数据库仅连接一次，并尽可能快。只有一种情况下可以连接，那就是被请求时。连接类被编译成一个单例。下面就是一个编译例子，类似于 C# 和 Java。
	`Class Connection`
	`{`
	`private static Connection conn = null;`
	`public synchronized static getConnection ( )`
	`{`
	`if (conn == null) {`
	`conn = new Connection ( );}`
	`return conn;`
	`}`
	`}`
	另一个单例模式的例子是为系统提供服务的工具类，比如一个工厂模式。因为服务是面对整个系统的，所以如果多种类提供相同的服务就会产生混乱。
	还有一个例子可能是一个播放音频片断的类。因为在同一时间只能播放一个音频片断，所以音频片断的管理者会进行控制。然而，如果这样工作，就必须有一个音频片断管理者的实例。

优点和结果	在某些时间只需要一个对象实例，但如果它只从一个地方实例化，那么我们可能不需要这个单例。这个单例对象控制它自己并确保只创建了一个实例，无论它被调用多少次，无论它在系统的什么地方出现。 实施单例的代码是很简单的，具备优秀设计模式的必要特征。

图 11-24　单件模式技术

单例模式与工厂方法模式有相同的基础逻辑。不同之处是单例类有适用于自身的静态方法的代码。单例解决方案是指类有一个引用所创建对象的静态变量。例如，getConnection 方法被定义并用来获得对象的引用。当 getConnection 方法被初次调用时，它实例化对象，并返回了一个对该对象的引用。以后再调用该方法时，只是返回一个对已经实例化的对象的引用。代码简单且讲究。样例中未显示构造器，然而，为了确保只有一个实例被创建，所有的构造器都被规定为私有——不可进入，所以没有其他的类可以调用它。

在单例样例中，模式用代码呈现。为了在你的设计中实现特定的代码，你应该套用 <singleton> 的类。好的程序员将识别这一模式并知道如何准确地编写类的代码。

本章小结

新系统的多层设计不受结构设计的限制。面向对象的细节设计也定义了系统中的不同阶段。类的定义和三层模式的职责在这一章中给出了解释。三层是指视图层、业务（逻辑）层以及数据访问层。

三层设计是基于设计模式的系统设计的整个活动的一部分。设计模式是指对系统设计的某一特定需求有效的标准解决方案或模版。在第 10 章中介绍过的其他模式是用例控制器，它是解决需要隔离从业务层到视图层两个层之间的耦合的一种简单的方法。

细节设计是单独设计每个用例的驱动用例。这一设计类型被称作用例实现。两个运用在

细节设计中的基本模型是设计类图和顺序图。设计类图在第 10 章中已讨论过。

用例的细节设计需要识别通过合作来开展用例的问题域类。每条从外部活动中输入的消息触发了一系列内部消息。使用顺序图或协作图，设计师能确定并定义所有的内部信息。接着，通过增加来自视图层和数据访问层的类和信息，解决方案得以完善。

最终的步骤是传送每条消息以及参数和返回值，使其到达位于正确的类的方法特征。这些信息用于更新设计类图。我们也会更改设计类图来展示类之间需要的可见性，以便在顺序图中传递消息。

当类在设计过程中被定义好后，它们被添加入 DCD 中。DCD 也可划分到几个层中或是子系统中。包图用来将 DCD 划分到合适的包中。类和包之间的独立性也会加入包图中。

流行的设计模式包括适配器模式、工厂模式、单例模式和观察者模式。适配器模式实现了保护变量的设计原理，它通过更改一点系统来简化系统，使之更稳定。当系统中可插拔的部分需要改变时，可以将其拔下，更新组件则可以插入。

工厂模式和单例模式有许多共同点。它们都向特定对象返回参数，都只允许一个对象的实例存在于系统中。不同点是工厂模式只执行一个单一的工具类，而单例模式只强制执行单例本身这一个类。

复习题

1. 用例实现是什么意思？
2. 了解和使用设计模式的优点是什么？
3. “四人帮”对系统开发的贡献是什么？
4. 标准设计模式定义的五个组成是什么？
5. 罗列出五个顺序图中所包含的元素。
6. 顺序图和 SSD 的区别是什么？
7. 设计 CRC 卡和设计顺序图的区别是什么？
8. 解释在顺序图中消息的语法。
9. 初步顺序图的目的是什么？包含了哪些类？
10. 用例控制器的目的是什么？
11. 活动生命周期的意义是什么？它如何运用在顺序图中？
12. 描述为初步顺序图开发一系列消息的三个主要阶段。
13. 当制作最初的用例实现时，开发者通常有什么假设？
14. 当制作多层设计时，每层的设计顺序是什么？
15. 职责分离的原理是什么？
16. 解释进入数据库创建新的内存中的新对象的两种方法。
17. 协作图的两个符号是什么，它们有什么含义？
18. 解释协作图中消息语法的组成。这些语法和顺序图中的消息有什么区别？
19. 解释设计类的语法方法。
20. 独立关系是什么意思？在图表中如何表示？
21. 罗列出三层设计中每层实现的职能。
22. 适配器模式的目的是什么？
23. 单例模式和工厂模式的相同元素是什么？这两个模式的基本不同点又是什么？

问题和练习

问题 1 ~ 问题 7 是基于第 5 章中有关大学图书馆系统的问题 1 和问题 2 的解决方案而设计的。作为选择，你的老师可能会给你提供用例图和类图。

1. 图 11-25 所示为大学图书馆中“借书”用例的系统顺序图。完成以下习题：

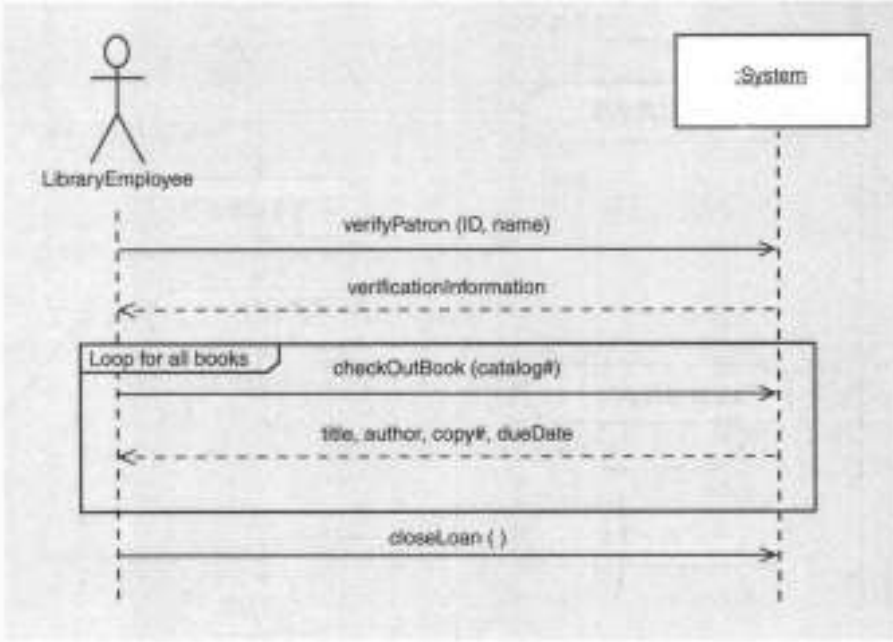


图 11-25　借书的系统顺序图

   a. 建立初步顺序图，只包含参与者和问题域类。
   b. 基于你的解决方案开发设计类图。确定包含你的控制器类。
2. 使用问题 1 的解决方案，完成以下习题：
   a. 在你的图中增加视图层类和数据访问类。你可以用两个独立的流程图来使它们更易于阅读和操作。
   b. 开发展示三层解决方案的带有视图层、域层和数据访问层的包图。
3. 图 11-26 所示为大学图书馆系统中“还书”用例的活动图。完成以下习题：
   a. 建立初步顺序图，只包含参与者和问题域类。
   b. 基于域类图创建设计类图。
4. 将你的解决方案应用到问题 3 中，完成以下习题：
   a. 在你的图中增加视图层类和数据访问类。
   b. 开发通过视图层、域层和数据访问层展示三层解决方案的包图。
5. 图 11-27 所示为大学图书馆系统中“接收新书”用例的完整描述，完成以下习题：
   a. 建立初步顺序图，只包含参与者和问题域类。
   b. 基于域类图创建设计类图。
6. 将解决方案运用到问题 5 中，完成以下习题：
   a. 在图中增加视图层类和数据访问类。
   b. 建立展示三层解决方案的带有视图层、域层和数据访问层的包图。

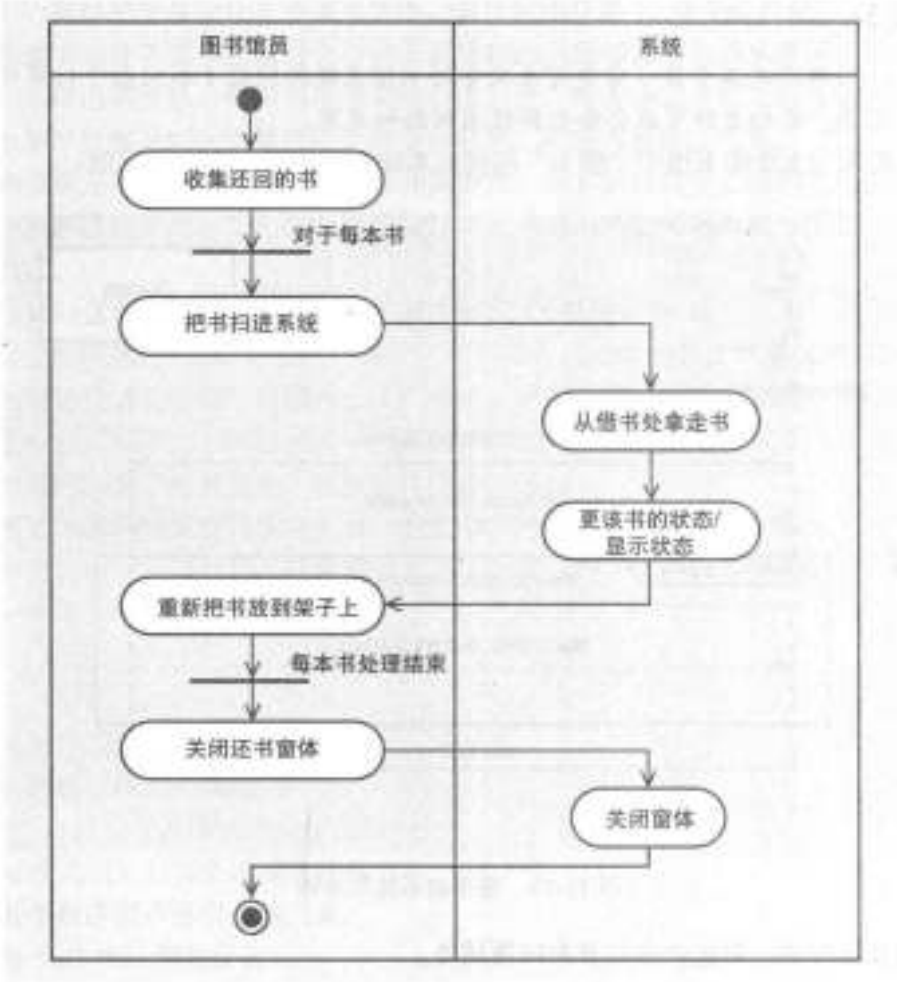


图 11-26 还书活动图

7. 将在问题 1、问题 3、问题 5 中得到的设计类图解决方案结合起来形成一个设计类图。

问题 8 ~ 问题 14 是基于第 5 章中问题 3 和问题 4 的解决方案。不同的是，你的老师会将用例图和类图给你。

8. 图 11-28 所示为牙科门诊系统中“记录看病过程”用例的活动图。完成以下习题：
   a. 建立初步顺序图，只包含参与者和问题域类。
   b. 基于域类图创建设计类图。
9. 将解决方案应用到问题 8 中，完成以下习题：
   a. 在你的图中增加视图层类和数据访问类。
   b. 基于域类图创建设计类图。
10. 图 11-29 所示为牙科门诊系统中“输入新病人信息”用例的活动图。完成以下习题：
   a. 建立初步顺序图，只包含参与者和问题域类。
   b. 基于域类图创建设计类图。
11. 将你的解决方案应用于问题 10，完成以下习题：
   a. 在你的图中增加视图层类和数据访问类。
   b. 建立展示三层解决方案的带有视图层、域层和数据访问层的包图。

用例名称	接收新书	
场景	接收新书	
触发事件	新购买的书到达	
简单描述	图书馆员计划采购一些新书，并发出了订单（先于这个用例）。新书运来了。每本新书都被分配了一个图书馆目录号码。有些书只是已有图书的副本，有些书是已有图书的新版本，有些书的书名是新的或者内容是新的。新书的信息被添加到系统里。	
参与者	图书馆员	
利益相关者	图书馆员，图书馆长	
前提条件	无	
后续条件	存在书名，书存在	
活动流程	参与者	系统
	1. 按发货单接收新书。 2. 对每本书都检查数目和目录号码，分配临时号码。 3a. 如果是已有图书的副本，则将图书信息和目录号码输入系统。 3b. 如果是已有图书的新版本，则输入图书信息、版本信息和目录号码。 3c. 如果是个新名字，则分配目录号码，分配图书副本号码。 4. 给图书编号。 5. 将图书放到车上。 6. 每本书都要重复以上步骤（返回 2）。	3a.1 用新号码更新目录。核实没有重复。 3b.1 用新号码更新目录。核实没有重复。 3c.1 核实目录号码没有重复。
异常条件	重复的号码需要进一步调查，并重新分配目录号码。	

图 11-27　接收新书用例的完全展开描述

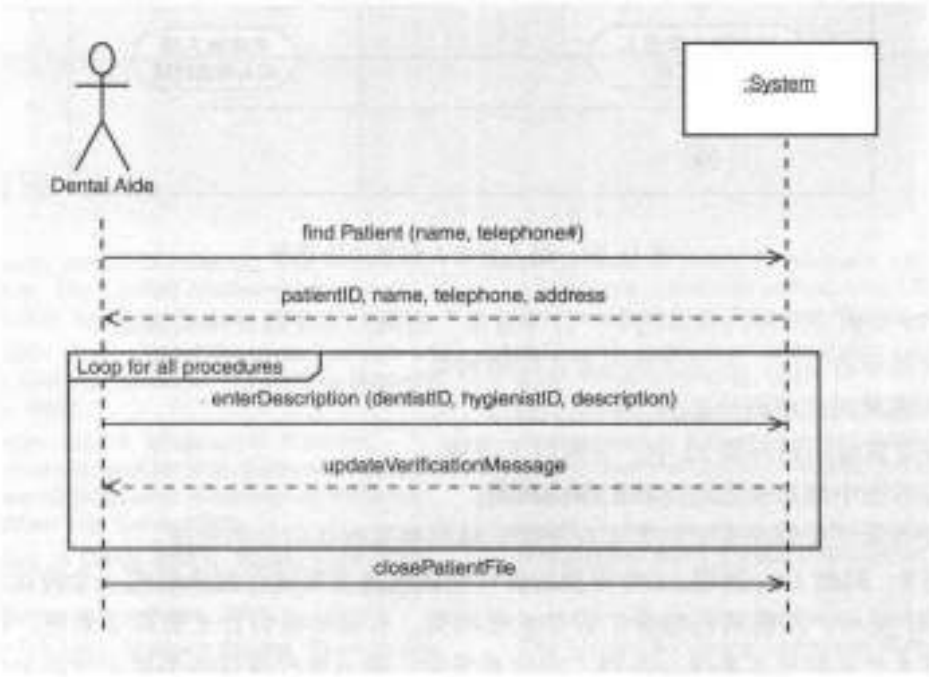


图 11-28　记录看病过程的系统顺序图

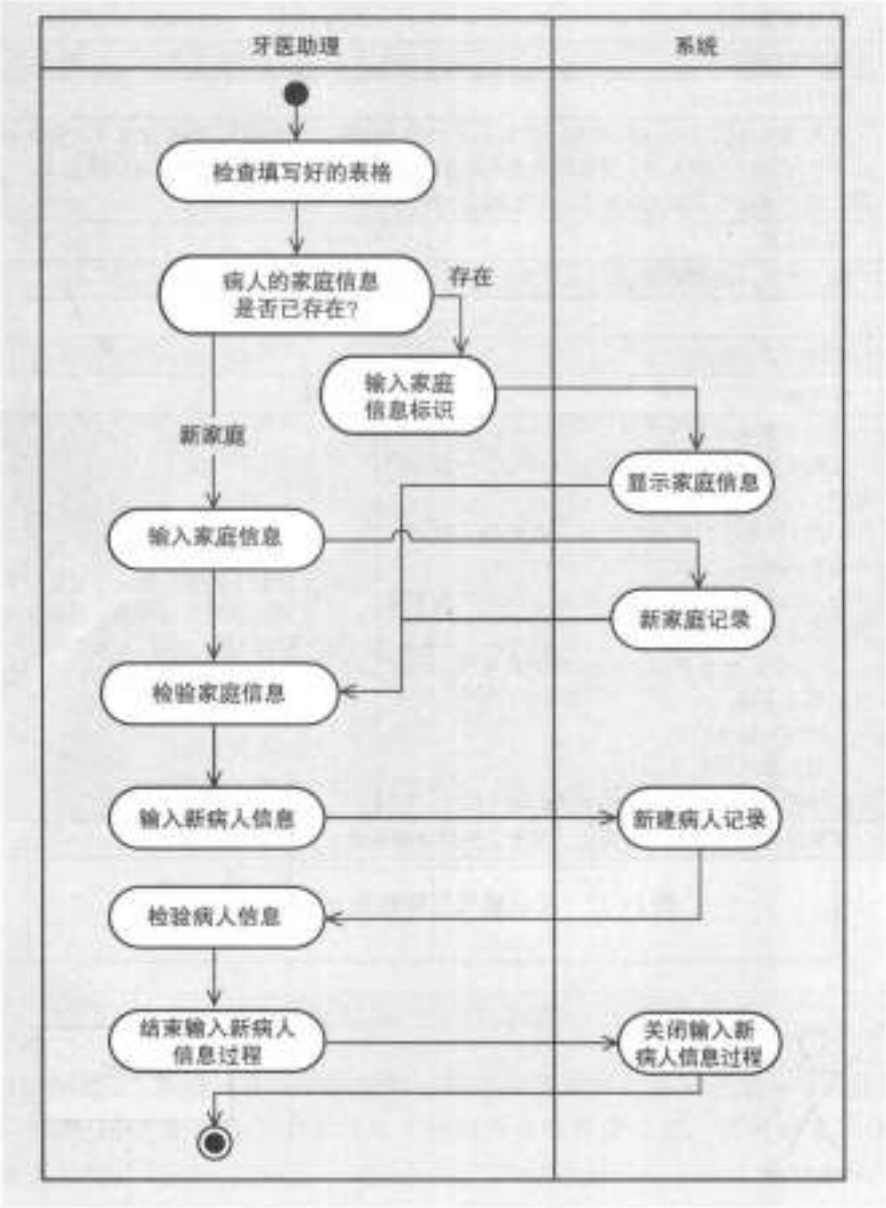


图 11-29 输入新病人信息的活动图

12. 图 11-30 所示为牙科门诊系统中"打印清单"用例的完全展开用例描述。
    a. 建立初步协作图，只包含参与者和问题域类。
    b. 基于域类图创建设计类图。
13. 将解决方案运用到问题 12 中，完成以下习题：
    a. 在你的图中增加视图层类和数据访问类。
    b. 建立展示三层解决方案的带有视图层、域层和数据访问层的包图。
14. 将问题 8、问题 10、问题 12 中得到的设计类图解决方案结合起来形成一个设计类图。
15. 图 11-31 所示，左侧的包包含工资单系统的类，右侧的包包含工资税子系统。你将使用什么技术把工资税子系统合并到工资单系统中？通过更改现存的类展示你将如何解决问题（任选一幅图）。你将增加什么新的类？使用 UML 符号。

用例名称	打印清单	
场景	打印清单	
触发事件	月末，打印清单	
简单描述	记账员工人工检查以确保已经收集了所有的过程。员工通过在系统中查看是否存在书面记录的方式进行抽样调查，以确保所有过程均已录入。最后，打印清单报表。	
参与者	记账员工	
利益相关者	记账员工，牙医	
前提条件	病人记录必须存在，过程必须存在	
后续条件	病人记录上次的记账日期来更新	
事件流程	参与者	系　统
	1. 收集本月完成的有关过程的所有书面记录。 2. 查看几个病人，确保过程信息已经输入。 3 检查已经接收支付记录，核实支付已经输入。 4. 输入月末日期，并请求订单。 5. 核实清单的正确性。 6. 关闭清单打印处理。	2.1 显示病人信息，包括过程记录。 3.1 显示病人信息，包括账户余额和上次支付事务。 4.1 检查每个病人记录。找到未支付过程。根据年龄或当前状态列在报表上。计算并用公司支付或者保险支付逐步分解。
异常条件	无	

图 11-30　打印清单用例的完全展开描述

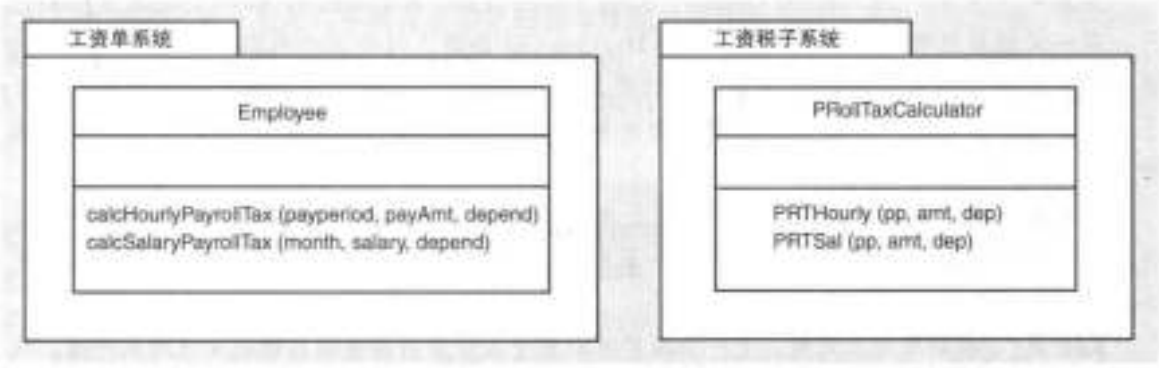


图 11-31　系统的包和类

扩展资源

Grady Booch, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

Grady Booch, et al., *Object-Oriented Analysis and Design with Applications*, 3rd edition. Addison-Wesley, 2007.

Frank Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley and Sons, 1996.

Alur Deepak, J. Crupi, and D. Malks, *Core J2EE Patterns: Best Practices and Design Strategies*. Sun Microsystems Press, 2001.

Hans-Erik Eriksson, Magnus Penker, Brian Lyons, and David Fado, *UML 2 Toolkit*. John Wiley and Sons, 2004.

Erich Gamma, R. Helm, R. Johnson, and J. Vlissides, *Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

Mark Grand, *Patterns in Java*, Volumes I and II. John Wiley and Sons, 1999.

Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*, 3rd edition. Prentice Hall, 2004.

David S. Linthicum, *Next Generation Application Integration: From Simple Information to Web Services*. Addison-Wesley, 2004.

James Rumbaugh, Ivar Jacobson, and Grady Booch, *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

实现系统的可操作性

学习目标

阅读本章后，你应该具备的能力：

- 描述实施和部署活动。
- 描述软件测试的不同类型并解释如何使用每种测试及使用的原因。
- 解释部署管理、改变管理和源代码控制对一个系统的实施、控制和部署的重要性。
- 列出数据转换和系统部署的不同方法，并描述每种方法的优点和缺点。
- 描述新的可操作系统的培训和用户支持。

开篇案例　Tri-State Heating Oil 公司：系统开始运行时的优先顺序调整

星期一上午 8：30，Maria Grasso、King Song、Dave Williams 和 Rajiv Gupta 正准备进行一周一次的项目情况总结会。Tri-State Heating Oil 公司已经在 5 个月以前就开始开发一种新的用户订货和服务电话调度系统，预计完成时间是 10 周以后，但是现在的进展却已落后于计划。由于主要用户对新系统需求提出种种异议，使得系统规模比以前要大得多，早期项目迭代的完成远远低于预期。

Maria 在会议开始的时候说："自从上次会议之后，因为单元测试结果比预期的要好，所以我们已经赢得了一两天的时间。上一周所有的开发方法均通过了测试，因此这周我们不必再检查这些代码是否有误。"

Kim 说："我们不能太大意，上一个项目时所有令人讨厌的错误都出现在集成测试阶段。我们这周将完成用户界面所需的类，所以应该可以在下周的某个时间开始用事务类进行集成测试。"

Dave 热情地点了点头并说："太好了！我们必须尽快完成对用户界面类的测试，因为我们计划在 3 周以后对用户进行培训。我需要时间准备用户培训材料并且制定出最后的培训计划。"

Rajiv 看上去很疑惑，并说："我不确定我们是否应该在系统许多部分还处于开发状态时便准备早期培训计划。如果集成测试出现严重错误，我们要花很多时间去修改怎么办？还有，未完成的事务和数据库的类怎么办？我们能用这样一个在用户界面后面只存在一半的系统对用户进行培训吗？"

Dave 回答说："但是我们在 3 周以后必须开始培训，我们雇了 12 个临时工以便在新系统上培训我们的员工。他们中有一半人计划得在两周后开工，其余人再过两周开工，现在要重新协商他们的开工时间已经来不及了，我们可以延长他们在这里的时间，但是推迟他们的开工时间就意味着我们要在他们不工作时给他们发工资。"

Maria 大声说："我认为 Rajiv 的顾虑是有道理的，系统刚刚才完成和测试了一小部分，在 3 周以后便进行培训是不现实的，我们已经比原计划至少滞后了 5 周，在以后的几周内，我们无法夺回多余 4、5 天的时间。我已经进行了安排，对一部分剩余的代码做了调整，从而使得用户培训的关键工作可以优先进行，而有一些过程可以推迟一些进行。Kim，你能重

新安排一下你的测试计划，先处理所有的交互式应用吗？”

Kim 回答说：“那我得回办公室，看看这些程序是否可以。我暂时同意，但是我需要几个小时的时间来确认一下。”

Maria 回答说：“好吧，那我们继续开会。假设重新整理编码并完成了测试，能为 5 周后的培训提供一个可用的系统吗？过一会儿，Kim 有了确定的消息后，我会用电子邮件通知大家。我还要安排与 CIO 人员会面，通知他们关于临时雇员开销的坏消息。”

会场沉默了一会，Rajiv 问道：“那么我们还需要考虑什么别的问题吗？”

Maria 回答说：“让我再想一想……还有硬件交付和安装、操作系统和数据库管理系统的安装、网络升级和对分布式数据库访问的加强测试。”

Rajiv 笑着对 Maria 说：“你小时候一定是个魔术师，把项目的所有方面都包含进来很不错，管理部门为这个付你钱了吗？”

Maria 咯咯笑着回答：“我有时的确认为自己是个魔术师。如果管理部门因此付我钱的话，那么项目一结束我就可以退休了。”

12.1　引言

开发任何一个复杂的系统本来就是非常困难的。试看一下制造汽车的复杂性：要制造或购买数万种零件，工人们和机器必须把那些零件组装成为小组件，如电子仪表板、线束和刹车装置，这些子系统又组装成大组件，如仪表板、引擎和变速器，然后这些零部件再被构造、测试并交给后续装配步骤。每个步骤花费的精力、时效性、成本和输出质量取决于所有的程序步骤。

实施和部署一个信息系统在许多方面都类似于汽车制造，即这就是一个复杂的产品，装配过程必须确保有效使用资源、减少施工时间和提升产品质量。但是不同于汽车制造的是，这个过程不能只设计一次，然后根据它来制造成千上万个相似单元。相反，对每个新项目而言，实施和部署都是独一无二的，必须满足项目的特征。

我们已经用了很多章节来详细阐述系统开发生命周期的四个核心过程。那些核心过程是本书的焦点，但是要完成一个系统并让它能正常使用还是需要附加的处理过程和活动的。涵盖在本章中的核心过程和活动总结在了图 12-1 中。

图 12-1　安装和部署阶段的活动

事实上，在一个简单的章节中包含两个核心过程并不意味着这两个过程是简单的或是不

重要的。恰恰相反，这两个核心过程是你将通过完成其他课程、阅读其他书籍以及工作培训和实践经历而会学习到的复杂的处理过程。

正如图 12-1 所示的那样，编程和测试是主要的实施活动。你将从其他课程和书籍中学习编程的知识，所以我们不会在本书中讨论软件组件是如何构造的。但是，我们会详细讨论测试活动，因为它们是多个核心过程中的主要部分，包括项目计划和监控、设计、实施和部署。

12.2　测试

尽管每个核心过程中都会使用不同类型的测试，但是测试活动是实施和部署活动的关键部分。测试是一个检查组件、子系统或系统的过程，以此来决定系统的可操作性特征以及它是否包含任何缺陷。为了进行一次测试，开发人员必须要为功能需求和非功能需求制定一份完整定义的标准。从需求中，做测试的开发人员要精确定义所需的可操作性特征及缺陷的组成部分。开发人员可以通过审查软件的架构和构成，或通过设计并建立软件、使用软件功能和检查结果来测试软件。如果结果显示有一个缺点或缺陷，那么开发人员会一直循环早期的实施或部署活动，直到缺点被补救或缺陷被消除。

测试类型、相关的核心过程和测量的可操作性特征都总结在了图 12-2 中。每种类型的测试在后面部分会详细描述。

测试类型	核心过程	被测试的缺陷和可操作性特征
单元测试	实施	独立测试时不能正确执行功能的软件组件——例如，计算销售税的组件在一个或多个位置不断地出现计算错误
集成测试	实施	能独立正确执行，但是与其他组件一起测试时就不正确的软件组件——例如，订单输入和运输成本计算组件，它能通过单元测试但是一起测试时就失败，原因是数据从一个组件到另一个组件的转换错误
可用性测试	实施	能运行但是不能满足一个或多个与功能或易用性相关的用户需求的软件——例如，用户界面组件迫使用户遵循一个不必要的复杂程序来完成常见、简单的任务
系统和加强测试	部署	不能正确执行其功能或不能在常用操作环境下满足非功能需求的系统或子系统——例如，在用虚拟数据库单独测试时，订单检索功能在两秒内可显示结果，但是在使用真实数据库且与其他功能一起测试时却需要 30 秒

图 12-2　测试的缺陷和可操作性特征

开发测试的一个重要部分是确定测试实例和测试数据。**测试实例**的正式描述如下：

- 开始状态。
- 软件必须响应的一个或多个事件。
- 期望得到的响应或结束状态。

开始状态和结束状态以及事件都是由一组**测试数据**代表的。例如，系统的开始状态也许会代表一组特定的数据，如存在某个特定顾客和顾客的订单。事件可以代表一组输入的数据项，如顾客的账户和用于查询订单状态的订单号。所期望的响应可被描述为行为（如显示某种信息）或存储数据的特定状态，如一个取消了的订单。

准备测试实例和测试数据是一段枯燥而且耗时的过程。在组件和方法层中，必须至少将每个指令执行一次。要保证所有的指令在测试过程中都被执行过是一个复杂的问题。幸运的是，基于证明数学技术的自动化工具可用于生成一组完整的测试实例。应该为每个场景准备多个代表着正常和意外处理情况的测试实例。

12.2.1　单元测试

单元测试是在与其他软件集成之前，测试单个方法、类或组件的处理过程。单元测试的目标是在模块组成大的软件单元（如程序、类和子系统）之前，尽可能地确定并修复其中的许多错误。当许多单元结合之后再进行定位和修复错误就变得更加困难和代价昂贵了。

设计出来的单元很少是孤立运行的。相反，设计出来的一组单元会作为一个集成化的整体来运行。如果一个方法被认为是独立单元，该方法就可以被称作消息接收方法，从一个类或者更多类接收消息，反过来会发送消息给自己类中的方法或其他类中的方法。在顺序图中很容易看到这些关系，如图 12-3 所示，而图 12-3 是复制图 11-13 的。例如，当 CartItem 这个类收到 createCartItem() 类的消息时，它会执行外部处理过程并发送消息给六个方法——findPromo()、getPrice()、findProdItem()、getDescription()、findInvItem() 以 及 updateQty()，而这六个方法分别属于其他三个类——PromoOffering、ProductItem、InventoryItem。

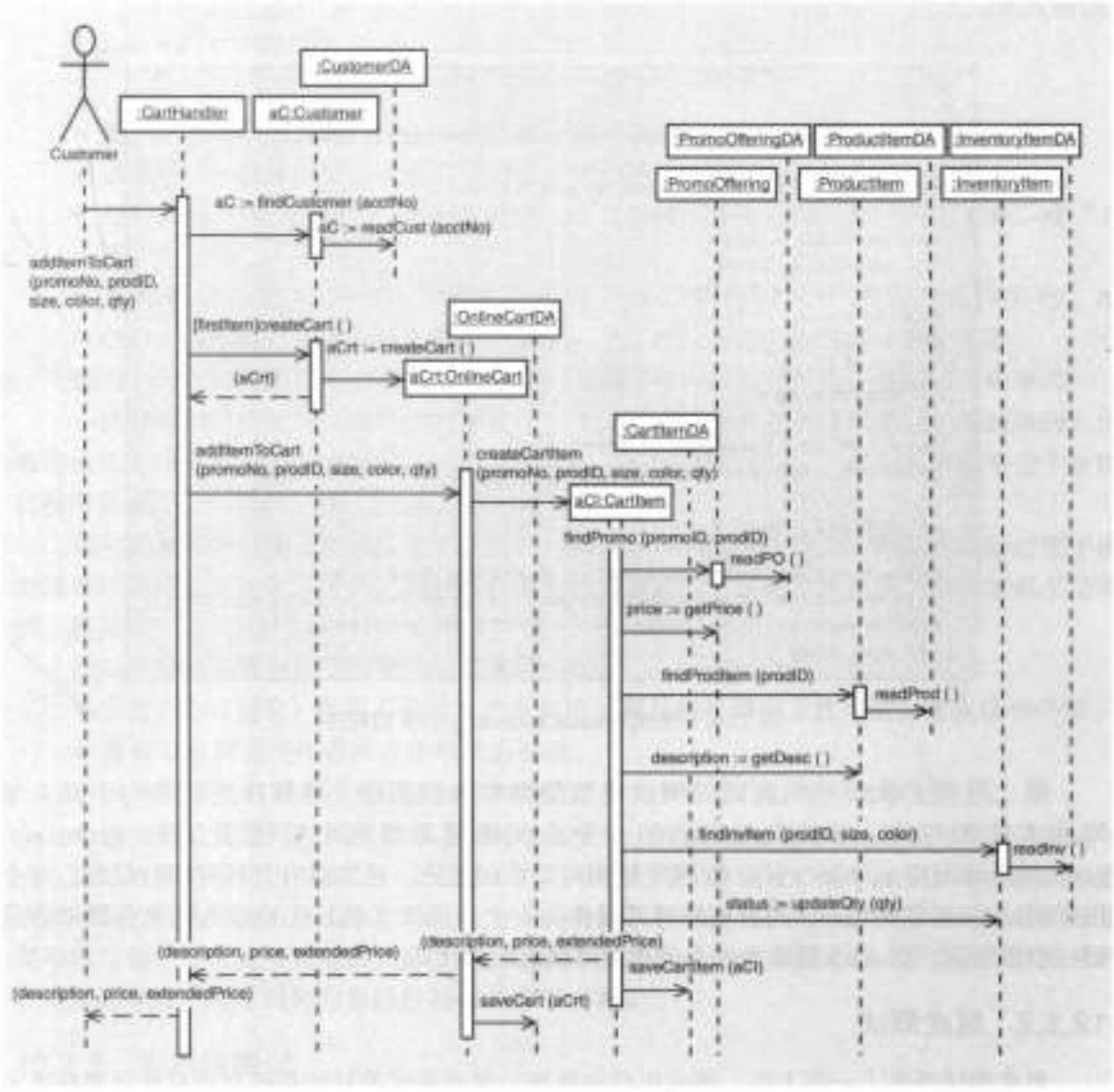


图 12-3　创建新订单的顺序图

如果 CartItem 的 createCartItem() 方法正在被单独测试，那么就需要两种类型的测试模型。第一个方法称为驱动程序。**驱动程序**用来模仿方法的行为，而这个方法要发送消息给正在被测试的方法。在这个例子中，是通过 OnlineCart 这个对象来调用类 createCartItem() 的。驱动程序模块实施了以下这些功能：

- 为输入参数设置参数值。
- 调用要测试的单元，并把输入参数传递给它。
- 接收被测试单元返回的参数并打印、显示，或者依靠所期望的结果来测试它们的值并打印或显示结果。

图 12-4 显示了测试 createCartItem() 的一个简单的驱动程序模块。一个更复杂的驱动程序模块也许要使用存储在文件或数据库中的数百个或数千个测试输入和正确输出的测试数据。驱动程序将循环进行数据测试并重复调用 createCartItem()，检查返回参数和期望值之间的差别，并打印或显示有任何差异的警告。使用驱动程序使得次级模块能够在被编写调用之前得到测试。

```
main()
{
 // driver method to test CartItem::createCartItem()

 // declare input parameters and values

 int promoID = 23;
 int prodID = 1244;
 String size = "large";
 String color = "red";
 int quantity = 1;

 // perform test

 cartItem cartItem = new cartItem();
 cartItem.createcartItem(promoID,prodID,size,color,quantity);

 // display results

 System.out.println("price=" + cartItem.getPrice());
 System.out.println("description=" + cartItem.getDescription());
 System.out.println("status=" + cartItem.getStatus());
} // end main()
```

图 12-4　测试 createCartItem() 的驱动程序

第二种用于执行单元测试的测试模型类型称为桩程序。**桩程序**可模仿一个尚未编写的方法的行为。createCartItem() 的一个单元测试需要三个桩程序方法：getPrice()、getDescription()、updateQty()。桩程序是相对简单的方法，通常只有几行可执行状态。每个用于测试 createCartItem() 的桩程序都可以作为一个声明来实施，无论输入什么参数都会返回一个常数值。图 12-5 所示为三个桩程序模块的示例代码。

12.2.2　集成测试

集成测试评价了一组方法、类或组件的性能。集成测试的目的是确定没有或没能在单元测试中发现的错误。这样的错误可能来源于以下一些问题：

```
float getPrice()
{
 // stub method for CatalogProduct::getPrice()

 return(24.95);
} // end getPrice()

String getDescription()
{
 // stub method for Product::getDescription()

 return("mens khaki slacks");
} // end getDescription()

String updateQty(int decrement)
{
 // stub method for InventoryItem::updateQty()

 return("OK");
} // end updateQty()
```

图 12-5　用于 createCartItem() 的桩模块

- 接口不兼容——例如，一个方法传递了带有错误数据类型的参数给另外一个方法。
- 参数值——被传递的方法或返回的值是我们不希望的，如价格的值是负数。
- 运行时间异常——由于资源需求冲突，方法生成了一个错误，如“内存不够”或“文件正在使用”。
- 意外的状态交互——两个或多个对象交互的状态引起了复杂的操作失败，如 OnlineCart 类的方法能够正确操作除了一个之外的所有可能的顾客对象状态。

这四个问题属于最普遍的集成测试错误，但是还有许多其他可能存在的错误和原因。

一旦集成错误被发现，就要追究那些导致不正确行为的具体方法的责任。测试执行人员通常也负责确定产生错误的原因。一旦确定了产生错误的特定方法，就要要求编写这个方法的程序员重写代码来改正这个错误。

面向对象软件的集成测试是非常复杂的。因为面向对象程序是由一组能在执行过程中被创建或销毁的交互对象组成的，所有没有清晰的层次结构。因此，对象交互和控制流是动态的、复杂的。

使面向对象集成测试变得复杂的因素还包括：

- 方法可以（通常）被许多其他方法所调用，而且这个调用方法可能分布在许多类中。
- 类可以从其他类中继承方法和状态变量。
- 具体被调用的方法是根据消息参数的数量和类型在运行过程中动态决定的。
- 由于第一次调用或两次调用之间的状态变化，对两次相同调用的响应可能是不同的。

这些因素的综合使得确定一个理想的测试顺序变得十分困难，同时也使预测一组交互方法和对象的行为变得十分困难。因此，为面向对象软件开发和执行一个集成测试计划是非常复杂的过程。处理这些难题的具体方法和技术不在本书的讨论范围之内。可参考本章最后的扩展资源部分中关于面向对象软件测试的参考文献。

12.2.3　可用性测试

可用性测试是确定一个方法、类、子系统或系统是否满足用户需求的测试。因为有

多种类型的需求（功能需求和非功能需求），所以可用性测试的许多类型会在不同时间来执行。

最常用的可用性测试类型是评估功能需求和用户界面质量。与用户交互的系统部分决定了系统是否满足期望的功能和用户界面是否容易使用。随着用户界面的开发，需要频繁使用这样的测试来提供快速的反馈给开发人员，以此来改进界面和改正任何软件组件中潜在的错误。

12.2.4 系统、性能和强化测试

系统测试是对整个系统或独立子系统行为的集成测试。集成测试通常与实施核心过程相关联，而系统测试通常与部署核心测试相关联。集成测试与系统测试的分界线是很模糊的，就像实施活动和部署活动之间的分界线一样。这两者之间的重要区别是范围和时间。集成测试会频繁执行，并且以较小的组件为一组。系统测试执行的频率没有那么高，而且是以整个系统或子系统为单位执行的。

对于使用传统的瀑布型系统开发生命周期开发的系统来说，系统测试通常集中在项目的最后。在一个典型的迭代项目中，一些包括系统测试活动的部署活动通常是在每个迭代的最后执行的。因此，系统是逐步被实施和部署的。

系统测试可能也会频繁地执行。**构造和冒烟测试**是日常执行或每周执行几次的系统测试。系统被完全编译和连接（构造），然后执行一组测试来检查在一个明显的路径上是否存在故障（“冒烟”）。

构造和冒烟测试是很有价值的，因为它们能对重大集成问题做出快速反应。由于先前的测试，所以任何发生在构造和冒烟测试中的问题来源肯定是软件的修改和添加。日常测试能确保迅速地发现错误，并且能够很容易地找到错误源头。如果不经常测试，那么这种好处就会逐渐消失了，因为当软件中的改动越来越多时，找到错误来源也越来越困难。

性能测试也称为**强化测试**，它要确定系统或子系统是否能满足时间方面的性能条件，如响应时间或吞吐量。**响应时间**需求说明了对查询或更新的软件响应所允许的预期的或最大的时间限制。**吞吐量**需求说明了每小时或每分钟内必须处理的预期的或更少的查询和交易数目。

性能测试是很复杂的，因为它们可以包含多个程序、子系统、计算机系统和网络结构。它们需要大量的测试数据来源以模拟正常负载量或最大负载量下的系统操作。诊断和纠正性能测试的故障也相当复杂。首先要确定遇到瓶颈和运行不佳的组件。纠正行为可能会包括以下内容的任何一种组合：

- 应用软件调试或重新实施。
- 硬件、系统软件或网络的重新部署。
- 运行不佳组件的更新或代替。

用户可接受性测试

用户可接受性测试是确定系统是否满足用户需求的一种系统测试。用户可接受性测试也许会在项目临近结束时执行，或者也可能在每次迭代的最后被分解成一系列测试来实施。可接受性测试在大多数开发项目中是非常正式的活动。当一个新系统由外部组织建立或引进时，可接受性测试的细节有时会存在于征求建议书（RFP）和采购合同之中。顾客支付给开发人员的费用通常取决于具体通过的可用性测试。

12.3　部署活动

一旦新系统开发和测试完毕，就必须安装和付诸运行。部署活动（如图 12-6 所示）包含许多不一致的约束，包括成本、维护积极顾客关系的需求、支持员工的需求、复杂的后勤以及对组织而言的全部风险。用户接受和其他测试类型在先前部分都已经描述过了。多个测试类型的执行通常会同时发生，因为项目后期的迭代通常会包括实施和部署活动。以下部分主要讨论除了测试以外的部署活动。

图 12-6　SDLC 部署活动

12.3.1　转换与初始化数据

一个可操作系统需要一个充实的数据库来支持正在进行的处理过程。例如，RMO 的 CSMS 系统的在线订单输入和管理功能依靠的是有关产品、促销、顾客和以往订单的存储信息。开发人员必须确保在子系统可操作时这些信息已存在于数据库中。

系统启动时需要的数据可以从以下来源获得：

- 正被替代的系统中的文件或数据库。
- 手工记录。
- 组织内其他系统中的文件或数据库。
- 正常系统操作期间的用户反馈。

重新使用现有的数据库

大多数新的信息系统是用来代替或增强现有手工或自动化系统的。在最简单的数据转换形式中，新系统可以直接使用旧系统的数据库，稍加改变或不改变数据库的结构。重新使用现有数据库相当普遍，因为重新创建新数据库的难度和花费都很大，尤其是在目前的企业资源计划（ERP）系统中，一个单独的系统通常要支持多个信息系统。

尽管旧数据库在新的或升级的系统中普遍得到了再利用，但我们还是经常需要对数据库内容进行一些改变。典型的改变包括添加新表、新属性和修改现有的表或属性。现代化的数据库管理系统（DBMS）通常允许数据库管理员修改现有的完全充实的数据库的结构。添加新属性或改变属性类型这类简单的改变完全可以利用 DBMS 来实现。

重新装载数据库

更复杂的数据库结构变化可能需要创建一个全新的数据库并且将旧数据库中的数据全部复制、转换到新数据库中。只要是有可能的，由 DBMS 提供的实用程序都要被用来复制和转换数据。在更复杂的转换中，实施员工必须开发程序来执行转换并转移一些或全部数据。

图 12-7 上面的部分就显示了这两种方法。在这两种方法中，一旦完成了转换和转移过程，就可以将旧数据库丢弃了。

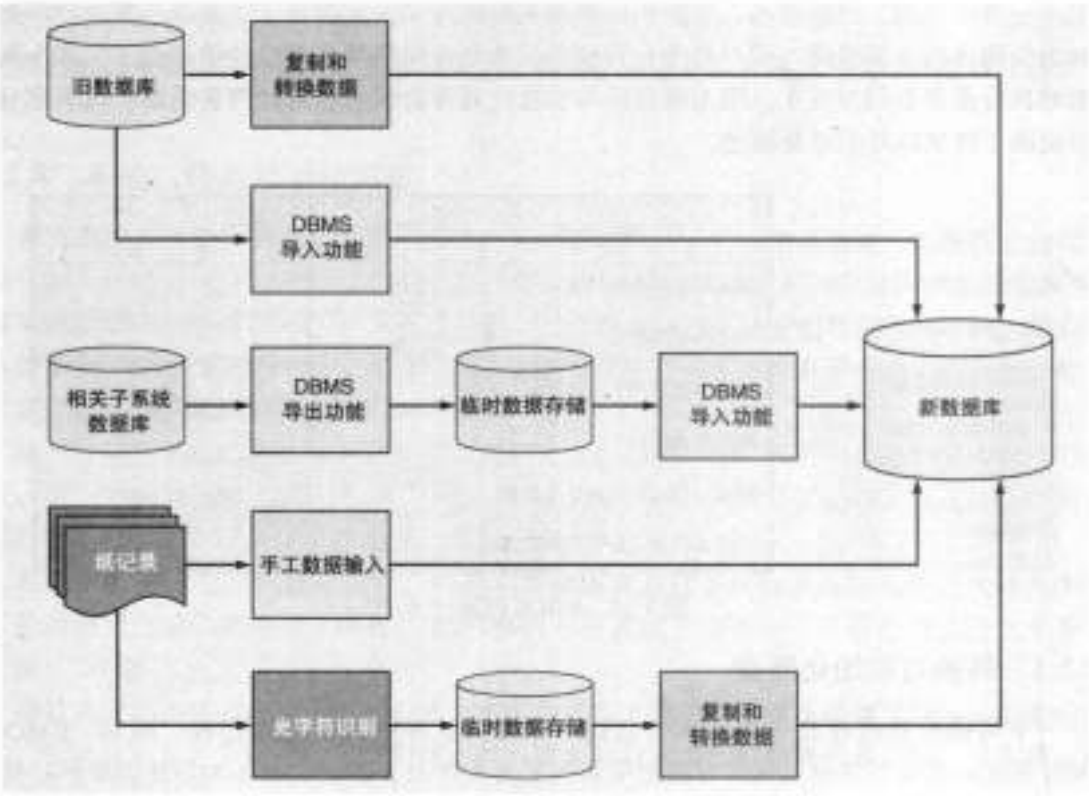


图 12-7　复杂的数据交互样例

图 12-7 中间的部分显示了一个更复杂的方法，即使用导出工具、导入工具和临时数据存储。在来源和目标数据库使用不同数据库技术时，可能会采用这个方法，不会真正存在能直接从一个数据库转化为另一个数据库这种情况，但是存在一种“中立”格式可以作为桥梁。

来自手工记录中的数据输入可以使用为可操作系统开发的相同程序来进行。在那种情况下，通常要尽可能早地开发与测试数据输入程序。初始数据的输入可以被构造成用户培训练习。为了提高效率，纸质记录中的数据也要被扫描到光学字符识别程序中，然后通过使用定制开发转换程序或 DBMS 的导入工具来输入到数据库中。

在某些情况下，操作系统可能起始于部分数据库或全部为空数据库。例如，顾客订单输入系统不必要求现有顾客信息事先装载到数据库中。顾客信息可以在其第一次下订单时或基于电话订单输入人员与顾客之间的一次对话的信息而增加到数据库中。这种在交互时添加数据的方式降低了数据转换的复杂性，但代价是初始事务处理变慢。

12.3.2　培训用户

培训两种用户——最终用户和系统操作员——是任何系统开发项目的必要部分。最终用户是为了获取系统业务目标而每天使用这个系统的人。系统操作员是为了保持系统的可操作性而执行管理功能和日常维修的人。图 12-8 显示了每个角色的代表性活动。在小型系统中，一个人可能要同时扮演两种角色。

最终用户活动	系统操作活动
创建记录或交易	开始或停止系统
修改数据库内容	查询系统状态
生成数据库	从档案中备份数据
查询数据库	从档案中恢复数据
导入或导出数据	安装或更新软件

图 12-8　最终用户和系统操作者的典型活动

培训的本质会随着目标受众的不同而不同。对最终用户的培训必须强调具体业务过程或功能的实用性，如订单输入、库存控制或财务等。如果用户还不熟悉，那么培训内容必须包括这些内容。用户技能水平和经验的千差万别要求我们至少组织一些实用培训，包括实践练习、答疑和单独指导。自学培训材料能够满足一些需求，但是复杂系统一般也需要一些面对面培训。如果最终用户的数量很多，就可以用成组培训的方式，也可以先培训一些素质较高的最终用户，再由他们对其他用户进行培训。

当系统操作员不是最终用户时，系统操作员培训可以采取非正式方式。经验丰富的计算机操作员和管理员能够通过自学获得他们所需的知识。因此不必进行正式的培训。此外，如果需要，对少量的系统操作员也可采取单独面授的方式。

决定正式培训的最佳开始时间可能是比较困难的。一方面，用户可以作为被开发和被测试系统的一部分来培训，这能确保他们积极地做到最好。另一方面，过早开始培训活动也可能会使用户和培训者都感到挫折，因为此时的系统不稳定也不完整。对最终用户来说，使用一个在不断变化、有可能出错或崩溃的系统是非常容易遭到挫败的。

理想的情况是，在系统界面完成、安装好测试版本并且排除故障之后，再进行培训活动。但是到了项目结尾的关键时刻，培训颇耗费较大，通常都要做出很大牺牲。相反，培训材料通常是在界面比较合理稳定时才开发的，并且最终用户的培训在此之后也会尽早开始。如果系统界面在早期的项目迭代中完成，那么进行培训就变得更容易了。

文档和其他培训材料通常是在正式用户培训开始前进行开发的。文档大致可以分为两类：

- **系统文档**——系统需求、结构和构建细节的描述。
- **用户文档**——如何与系统交互、如何使用系统的描述。

每种文档类型都与不同的目标和大众相对应。系统文档的目的是支持现在和将来的开发活动。用户文档在实施阶段创建。开发团队不能过早地创建用户文档，因为许多用户界面和系统操作的细节都还没有确定或者在开发期间会发生变化。

系统文档

系统文档的主要目标是为建立、维护和更新系统的开发人员与其他技术人员提供信息。系统文档是在系统开发生命周期中通过每个核心过程和许多开发活动生成的。在早期项目迭代中开发的系统文档能引导后期迭代中的活动，并且系统开发生命周期中开发的文档能引导未来系统的维护和更新。

为顾客部署的系统是计算和网络硬件、系统软件和应用程序软件的集合。一旦系统被开发，关于系统本身的单独描述（如写下来的文本和图像模型）就都是系统本身的冗余。在计算的早期，用来支持分析与设计模型开发的自动化工具很少，甚至从这些模型中生成应用软件的自动化处理过程的支持也很少。在那个时代的开发人员面临的是一个反复出现的困境：在确保系统文档与实际文档同步发生的同时，如何使更新模型和应用软件所花费的重复精力

降到最少。在匆忙完成和部署系统以及不停维护和更新时，系统文档更新通常会被忽略，而且文档也会丢失。因此，系统通常会在时限之前就被废弃，因为建立新系统比修复或更新一个文档破旧的现有系统来得更便宜。

现代化的应用开发工具和方法极大地解决了早期存在的系统文档困境。现代化集成开发环境提供自动化工具来支持所有系统开发生命周期的核心过程。需求和设计模型，如用例描述、类图和顺序图，都是通过使用开发工具开发的，并且会存储在项目库中（如图 12-9 所示）。模型的变化都会自动地与相关模型同步。应用软件通常会在部分或整个设计模型中生成。当应用软件在以后发生变化时，开发工具可以利用“反向工程”来适应模型的变化。由于这些性能，系统文档总是与被部署的系统同步完成，因此就简化了未来的维护和更新。

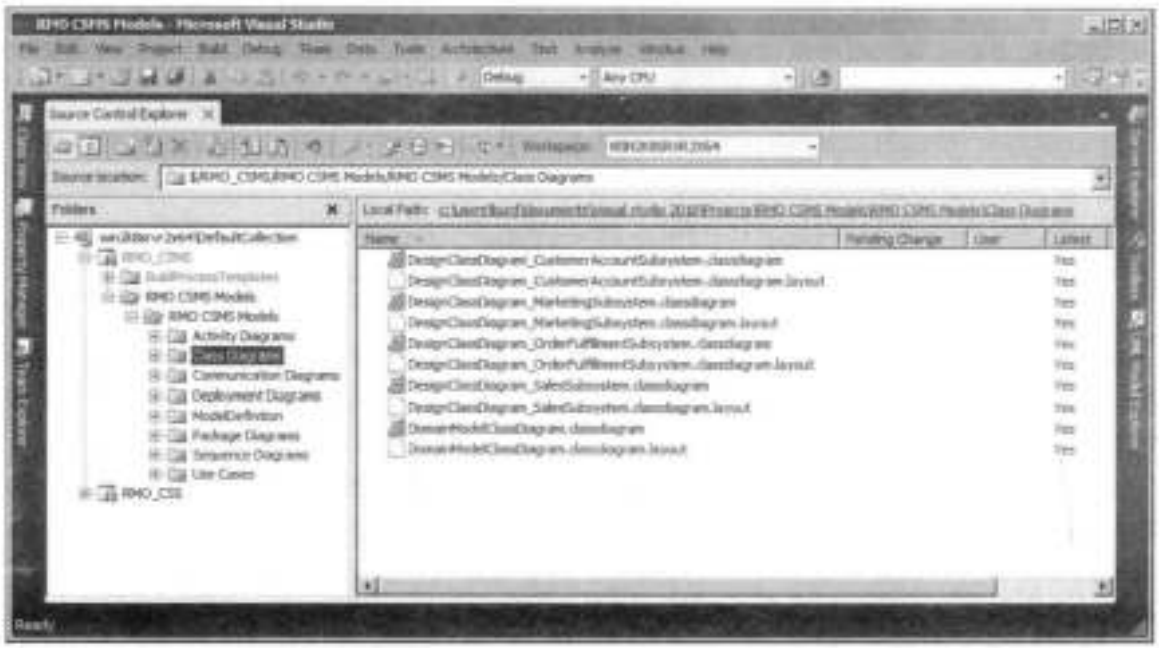


图 12-9　存储在 Visual Studio 中的系统文件

用户文档

用户文档为系统最终用户提供运行支持。它主要描述了系统的例行操作，包括数据输入、输出生成和定期维护这样的功能。其主要内容包括：

- 软件启动和关闭。
- 执行特定功能时需要的按键、鼠标或命令序列。
- 实施特定业务程序需要的程序功能（如输入新顾客订单时的步骤）。
- 常见错误和纠正错误的方式。

为了方便使用，用户文档包括内容表、程序或系统的目标或功能的大体描述、术语表和索引。

现代化系统的用户文档几乎都是电子版本的，并且通常是集成到应用程序之中的。大多数的现代操作系统都提供标准工具来支持嵌入式文档。图 12-10 显示的是在一个典型的 Windows 应用程序中的电子用户文档。通过点击工具栏上方书本形状的图标来显示内容表，并且可以通过使用搜索工具来搜索特定的单词或短语。样例页面包括嵌入式术语表定义的热键（绿色）以及连接其他文档页面的热键（蓝色）。

图 12-10　Windows 帮助和支持显示的样例

如何使用系统的知识是和系统本身一样都非常重要。完成初始培训之后，用户会记住这些实践知识。但是并不能保证这些知识能够保持有效地传递给后来的用户。员工离职、重新招募员工以及其他因素使得人与人之间有关操作性知识的交流变得困难和不确定。相比之下，书面文档或电子文档更容易得到而且保存时间更久。

开发好的用户文档需要专门的技术与大量的时间和资源。在要求高但资源有限的情况下，文档开发技术要求语言清晰简洁、制作出令人印象深刻的演示图片、组织的信息便于学习和获得，并且与非专业人员进行有效交流。开发工作耗费时间，只有通过审查和测试才能取得高质量的成果。不幸的是，准备用户文档的技术人员常常缺乏一种或多种必要的技能。此外，由于工作进度超标或者忙于最后的实施完工，准备时间、审查和测试通常会被忽略。

12.3.3　部署产品环境

现代化应用程序都是从基于交互标准的软件组件中建立起来的，交互标准有公共对象请求代理结构（CORBA）、简单对象访问协议（SOAP）和 Java 平台企业版（Java EE）。每个标准都定义了组件定位和与另一个组件通信的特定方式。每个标准也定义了一组支持系统软件来提供必要的服务，如维护组件指南、强制执行安全需求以及编码与解码网络和其他传输协议中的消息。准确的系统软件、硬件和部署需求在组件交互标准中都有本质上的差别。

图 12-11 所示为一个应用程序的典型支持结构，这个程序是使用 Microsoft .NET（SOAP 的一种变化形式）来部署的。用 VB 和 C# 这样的编程语言编写的应用软件组件都会被存储在一个或多个应用程序服务器中。其他需要的服务包括基于浏览器界面的 Web 服务器、管理数据库的数据库服务器、识别用户和批准访问信息及软件资源的活动目录服务器、路由器和防火墙、操作域名（DNS）和网络地址分配（DHCP）这样的低级网络服务的服务器。

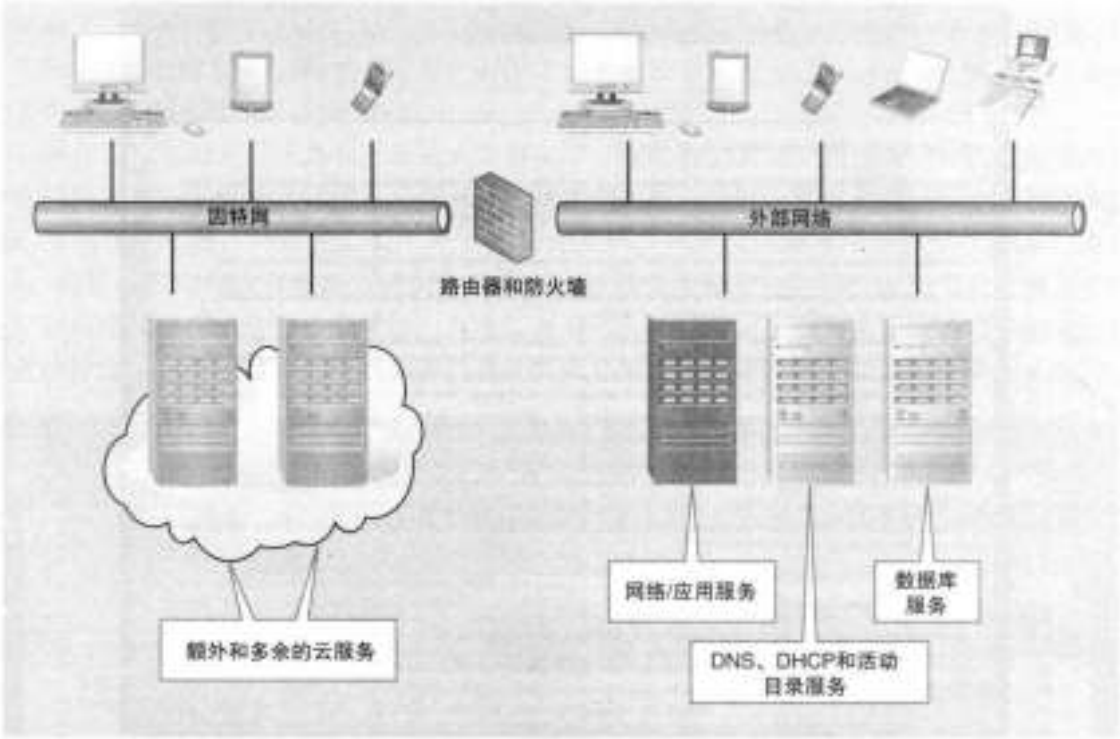


图 12-11 一个典型的 .NET 应用的基础设施和顾客

在应用软件被安装和测试之前，一定要完成、安装和部署好所有的硬件和系统软件的底层结构，除非它本来就已经存在。在很多情况下，所有或某些底层结构会已经存在——以此来支持现有的信息系统。在那种情况下，开发人员会和那些管理现有底层结构的人员一起工作，这样可以为新系统规划支持。不论发生何种情况，部署活动通常会在项目早期就开始，这样软件组件才能被开发、测试和部署，就像它们在后期的项目迭代中被开发。

12.4 计划与管理实施、测试和部署

先前已经单独讨论过了实施、测试和部署活动。在这部分内容中，我们集中在影响所有这些活动以及其他核心过程的问题上，包括项目计划、监控、分析和设计。在迭代开发项目中，所有核心过程中的活动都会集成到每个迭代中，并且系统会逐步进行分析、设计、实施和部署。但是项目经理如何决定系统的哪部分会在早期迭代中处理以及哪些部分会在后期迭代中处理？项目经理如何管理那么多模型、组件和测试的复杂性呢？

这些问题的一些已经在前面章节中提到过了。但是现在你要更深入地理解实施、测试和部署活动，你可以看到有很多要说明的互相依赖关系。在开发一个可行的迭代计划时，必须完整地确定和考虑这些依赖关系。而且，自动化工具必须用于管理开发项目的每个部分，且要确保迭代、核心过程和活动中的最大协调性。

12.4.1 开发顺序

开发系统所要做的一个最基本决定就是确定软件组件的建立（或获取）、测试和部署的顺序。选择在哪个迭代中实施系统的哪个部分是很困难的，并且开发人员必须要考虑很多因素。只有部分因素是来自于软件需求，还有某些在先前章节中讨论过的其他因素，包

括验证需求和设计决策的要求，以及尽可能通过解决技术和其他风险来最小化项目风险的要求。

开发顺序可以直接基于系统本身的结构和相关的问题，如用例、测试和开发员工的有效利用。可能的顺序有几种，包括：

- 输入、处理、输出。
- 自上而下。
- 自下而上。

每个项目都必须采用这些方法的一个或多个组合来说明项目需求和限制。

输入、处理、输出的开发顺序

输入、处理、输出（IPO）开发顺序是基于一个系统或程序的数据流。先开发包含外部输入的程序或模块，再开发处理输入（如转换成输出）数据的程序或模块，最后开发产生输出结果的程序或模块。分析的关键问题是依赖关系——也就是哪些类或方法能通过其他类或方法来捕捉或生成需要的数据？依赖信息会记录在包图中，也可能记录在类图中。因此，无论是哪种类型的图都可以引导实施顺序决策。

例如，在图 12-12 所示的包图中，顾客账户与营销子系统不取决于任何其他子系统。销售子系统取决于顾客账户与市场营销子系统，订单实施和报告子系统取决于销售子系统。

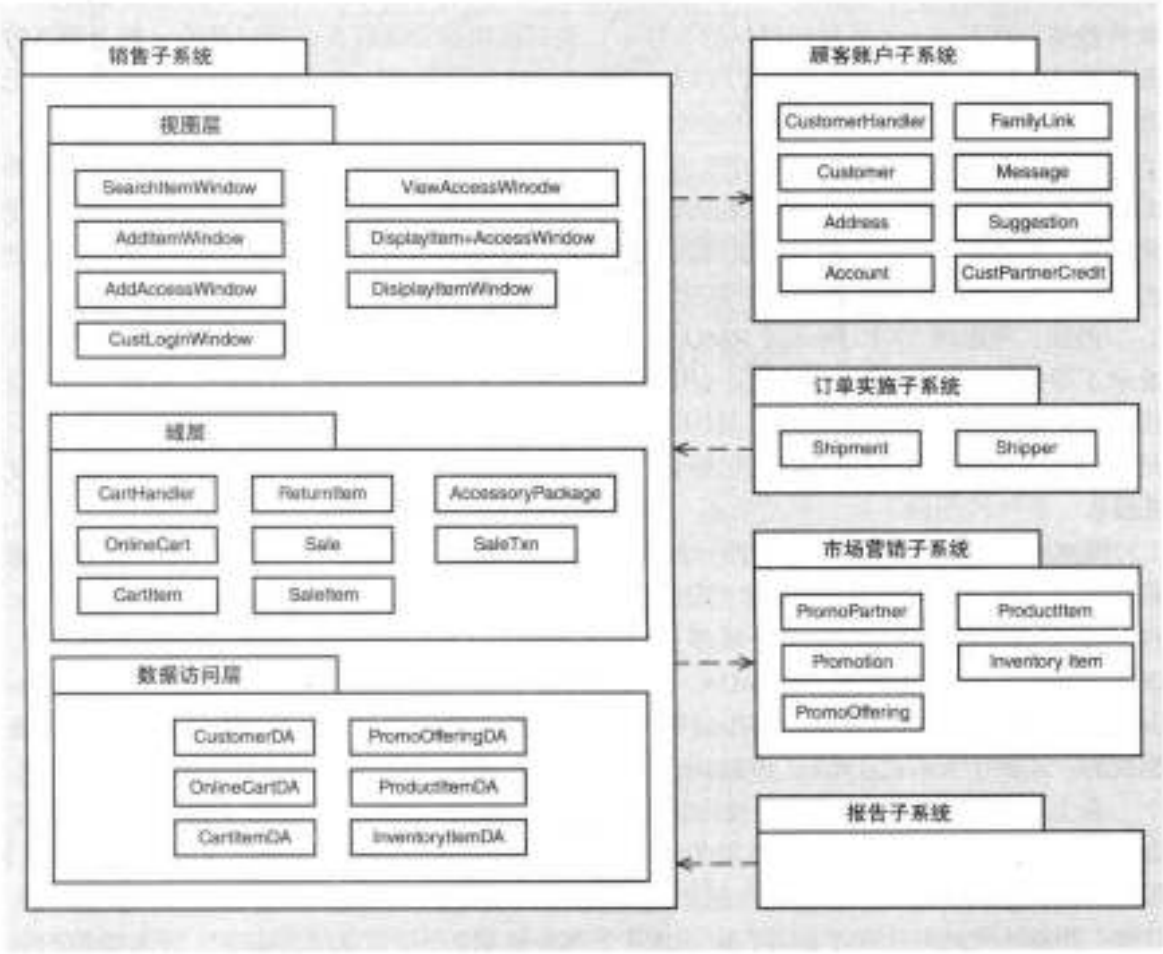


图 12-12　RMO 子系统的程序包图

存在于程序包（子系统）中的数据依赖说明了嵌入类之间的数据依赖性。因此，类CustomerHandler、Customer、Address、Account、FamilyLink、Message、Suggestion、CustPartnerCredit、PromoPartner、Promotion、PromoOffering、ProductItem 和 InventoryItem 在剩余的 RMO 类中没有数据依赖。按照 IPO 开发顺序，那三个类要首先实施。

IPO 开发顺序的主要优势是简化了测试过程。因为输入程序和模块会首先开发，所以它们可以用于为处理过程、输出程序和模块输入测试数据。IPO 开发顺序的另一个优势是，重要的用户界面（如数据输入程序）会在早期就被开发。用户界面部分比起系统其他部分而言，更容易被要求修改，所以早期的开发允许早期的测试和用户评价。如果需要改动，那么仍然有时间来完成。用户界面的早期开发也为相关活动提供了有利的开端，如培训用户和编写文档。

IPO 开发顺序的一个缺点是输出部分的滞后实施。输出程序对于测试面向过程的模块和程序十分有用，分析员可以通过人工检查打印的报告和显示的输出结果来发现处理过程的错误。IPO 的开发方法直到系统开发后期才进行这样的测试。然而，分析员可以经常使用查询程序或是数据库管理系统的报告编写能力来产生替代的输出结果。

自上而下和自下而上的开发顺序

术语自上而下和自下而上来自于传统结构化设计和结构化编程。传统结构化设计将软件分解成一组模块或功能，它们都是分层次地彼此互相关联。做一个直观类比的话，想象一个顶部是 CEO 的典型的组织结构图。在结构化设计中，一个简单的模块（CEO）能控制整个软件程序。在下一个更高层次模块的指导下，底部模块执行低层次的专门任务。**自上而下的开发方法**开始于最高层次模块（如 CEO），然后继续往下处理。**自下而上的开发方法**开始于最低层次的详细模块，然后继续往上处理到最高层次模块。

自上而下和自下而上的程序开发也可以用在面向对象设计和程序中，尽管用面向对象图的直观类比不是很明显。关键问题是依赖性方法——即哪些方法能调用其他方法。在一个面向对象子系统或类中，依赖性方法可以按照导航可见性来检查，像第 10 章和第 11 章中讨论的那样。

例如，考虑图 12-13 所示的 RMO 订单输入子系统的三层设计部分。包和类之间的箭头显示了导航可见性的需求。视图层（用户界面）中的方法调用域层中的方法，域层中的方法接着再调用数据访问层中的方法。自上而下方法的实施首先实现视图层中的类和方法，然后是域层中的类和方法，最后是数据访问层中的类和方法。自下而上开发方法的实现顺序正好相反。

依赖性方法也可以源于顺序图。例如，在图 12-3 中，依赖性方法源于对象之间从左到右的消息流图。把图向右旋转 90° 就成了自上而下和自下而上可视化分析，类似于结构图。自上而下开发方法的处理过程顺序是 CartHandler、Customer、CustomerDA、OnlineCart、OnlineCartDA、CartItem、CartItemDA。一组类是 PromoOffering、PromoOfferingDA、ProductItem、ProductItemDA、InventoryItem 和 InventoryItemDA。自下而上开发方法的顺序则是相反的，开始于 InventoryItem 和 InventoryItemDA，结束于 CartHandler。

自上而下开发的主要优点是程序总有一个工作规则。例如，图 12-3 中的自上而下开发方法会开始于 CartHandler 类的部分或完整版本，以及 Customer 和 OnlineCart 类的虚拟（桩）版本。这一组类构成的程序是可以建立、部署和执行的，尽管在执行的时候还不能实现什么功能。当每个方法或类被实施后，方法或类的桩会被添加到下个较低层次中。开发阶段的每一步中，都会执行和测试程序，并且它的行为会变得像开发成果那样更复杂、更现实。

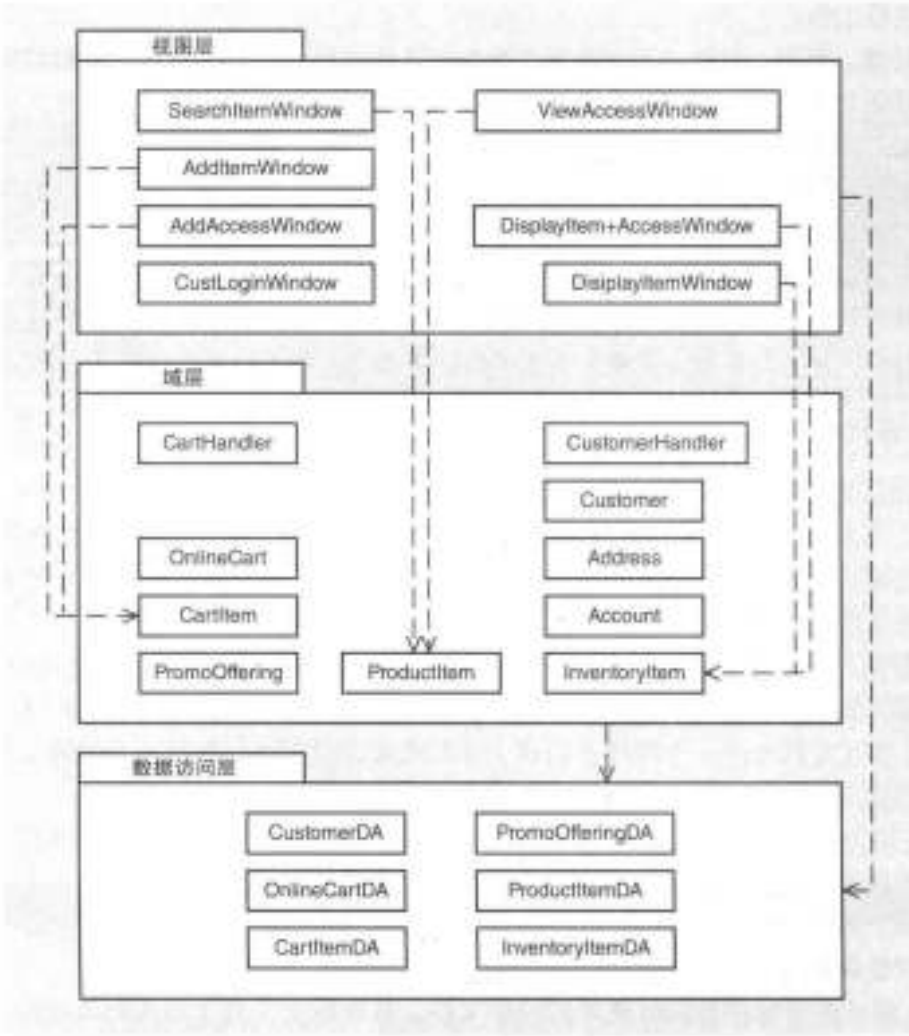


图 12-13　一个三层 OO 设计的包图

自上而下开发顺序的主要缺点是在系统构造初期不能很有效地利用程序员的人力资源。在大量的模块和方法被同时开发之前，开发工作只是进行两三个重复性的活动。相比之下，自下而上的开发顺序能使许多程序员立即投入工作，处理支持很多不同用例的类。遗憾的是，自下而上的开发顺序要求编写大量的驱动程序来测试底层类和方法，这给实施和测试类和方法添加了许多麻烦。而且，整个系统直到最顶层模块被实现后才算完整。因此，单独用例的测试和集成测试也被推迟了。

其他开发顺序的考虑

IPO、自上而下和自下而上的开发方法仅仅是创建实施和迭代计划的起点。其他还要考虑的因素包括用例驱动开发、用户反馈、培训、文档和测试。在决定开发顺序时要特别注意用例，因为用例是将开发项目分解成各个迭代的主要基础。

在大多数项目中，开发人员会为一次简单的迭代选择一组相关用例，然后完成分析、设计、实施和部署活动。开发人员选择用例时会首先考虑这样的因素：项目风险最小、能有效利用非技术员工或能提前部署系统的某些部分。例如，将不确定需求或高技术风险作为早期的迭代。明确不确定需求需要可用性和其他非技术员工的测试，这些员工仅仅只能在项目特

定时间段进行工作。

用户反馈、培训、文档和测试主要都取决于系统的用户界面。用户界面的尽早实施过程能使用户培训和用户文档开发在早期的开发过程中就开始。它也能收集关于界面的质量和可用性的反馈。这个问题在本章的开篇案例中扮演了重要的角色。

在决定开发顺序的时候测试也是一个重要的考虑因素，随着单个软件组件的构建，必须对它们进行测试。程序员们必须尽早发现程序中的错误，因为随着组件被集成化为大单元，要找到这些错误将越来越困难，修改这些错误的开销也越来越大。找出软件中易受错误影响的部分和找出软件中可能产生影响整个系统的重大错误的部分同等重要。不管这些部分存在于基本的 IPO、自上而下和自下而上方法中的任何部位，都应该尽早地构建和测试。

12.4.2 源代码控制

开发团队需要一些工具来帮助协调他们的编程任务。**源代码控制系统**（SCCS）是一种自动工具，用来跟踪记录源代码文件并控制这些文件的改动。SCCS 把项目的源代码文件存储在一个仓库中。SCCS 就像一个图书管理员——完成登记和检查手续，跟踪记录每个程序员拥有哪些文件，确保只有授权过的用户才有权访问这个仓库。

当程序员只想检查而不改动代码时（例如，检查某个模块与别的模块之间的接口），他可以以只读方式来访问文件。当程序员想改动代码时，他可以以读 / 写方式来访问文件。在同一时间，SCCS 只允许一个程序员以读 / 写方式来访问文件。在另一个程序员能够以读 / 写方式访问该文件之前，此文件必须已经被放回仓库中。

图 12-14 显示了 Microsoft Visual Studio 的源代码界面。主界面上显示了来自于 RMO 顾客支持系统的不同源代码文件。一些文件正在被程序员访问。对每一个以读 / 写方式访问的文件，界面上都将列出正在访问的程序员名字、访问日期和时间，以及当前存储在中心仓库的副本是否是最新版本。

SCCS 禁止多个程序员同时更新同一个文件，从而防止了对于源代码改动的不一致性。当多个程序员合作开发程序时，对于源代码的控制是绝对必要的。它防止了程序改动的不一致性，并且能够自动地在程序员和团队之间进行协调。这个仓库也可以作为通用工具来实现备份和恢复操作。

12.4.3 打包、安装和部署组件

正如在本章讨论过的其他规则一样，部署活动与其他规则中的活动也有高度的相互依赖性。简言之，一个系统或子系统直到它被实施和测试后才能进行部署。如果一个系统或子系统是庞大且复杂的，通常会将其分为多个层次或版本来进行部署，因此某些正式的部署方法和变化管理是必要的。

在计划部署时，要考虑以下重要问题：

- 并行运行两个系统所带来的开销。
- 新系统的检错和纠错。
- 可能会扰乱公司及其信息系统正常运作的潜在因素。
- 对人员进行培训并使客户熟悉新程序。

不同的部署方法代表了不同成本、复杂度和风险的权衡方式。最常用的部署方法是：

- 直接部署。

- 并行部署。
- 阶段部署。

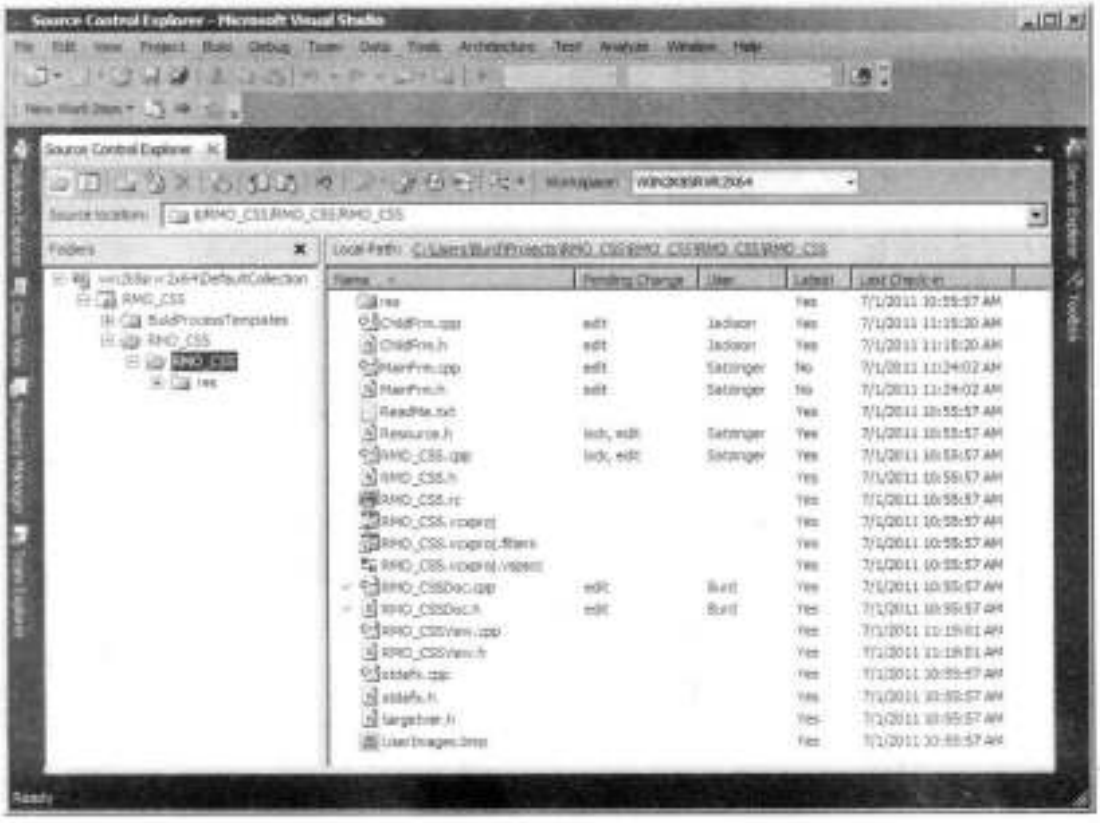


图 12-14　由源代码控制系统管理的工程文件

每种方法都有不同的优点与弱点，但没有一种方法是适用于所有系统的。每种方法的详细描述如下。

直接部署

在**直接部署**中，我们会安装新系统并使系统快速进入运行状态，重复的系统会立即被关闭。直接部署有时也叫作**立即接入**。当系统安装测试完后，新旧两个系统只同时运行很短的一段时间（通常是几天或几周）。图 12-15 显示了直接部署的时间进程。

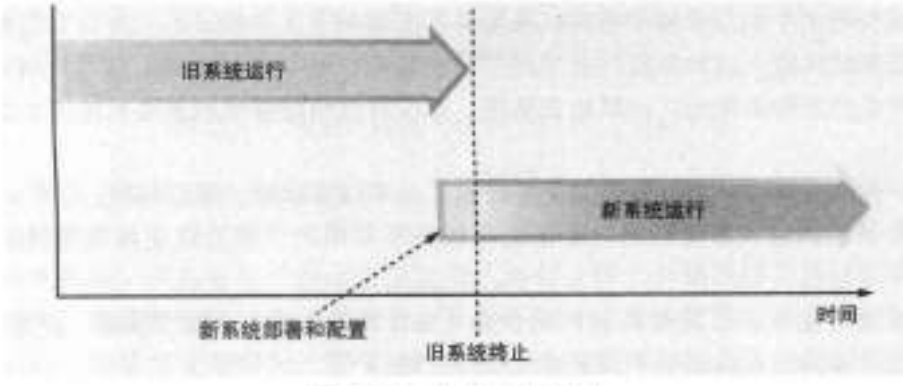


图 12-15　直接安装和接入

直接部署的主要优点是它的简单性。既然新旧两个系统不能并行运行，那么就没有太多的后勤管理问题，耗费的资源也少得多。直接部署的缺点是它的风险性，因为旧系统不能并行运行，所以在新系统运行失败的情况下就没有备份。风险的大小取决于系统的特性、在系统失败时工作区的耗费及系统不能使用或非最佳状态所带来的损失。

并行部署

并行部署可以是新旧系统在很长的一段时间内同时运行（数周或数月）。图 12-16 说明了并行部署的时间进度。旧系统将一直运行到新系统被全面测试、确信无错和可以独立运行为止。从实践来看，并行运行的时间通常是预先确定好的并且是有限制的，以便尽量减少双系统运行带来的耗费。

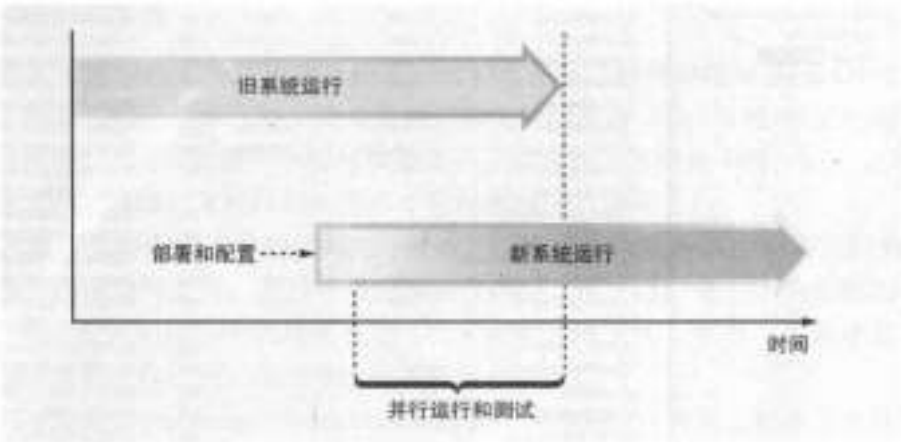


图 12-16　并行安装和接入

并行部署的主要优点是可操作性风险较小。如果两个系统同时运行（即用所有的数据和训练所有的函数），那么旧系统可以作为新系统的备份，新系统的任何失败所造成的的损失都可以从旧系统中得到恢复。

并行部署的主要缺点是成本。在两个系统同时运行时，要为两个系统都分配资源。在并行运行中，会带来许多与之相关的额外开销，包括：

- 雇佣临时员工或者给已有人员增加临时任务。
- 为计算和网络设备增加额外空间。
- 增加了管理和后期的复杂度。

当系统失败产生的后果较严重时，采用并行安装的方法是最好的。并行安装确实降低了系统失败带来的风险，这种风险的减少对“苛求任务”是非常重要的，如客户服务、生产控制、基本财务功能和多种形式的联机交易等。很少有组织能够承担重要系统停止运行所造成的损失。

对于一些情况来说，实行系统完全并行也许是不切实际的，原因包括：

- 一个系统的输入数据对另一个系统可能是不可用的，而且使用两种类型的输入也许是不可行的。
- 新系统可能与旧系统使用相同的设备（如计算机、输入/输出设备、网络），这样有可能造成新旧系统都得不到足够的运行资源。
- 没有足够的操作和管理人员来同时运行两个系统。

如果完全的并行运行是不可能的或不可行的，那么可代之以部分并行运行。运行并行模式的方式包括以下几种：

- 在其中一个系统中只处理输入数据的一个子集，子集可由事务类型、图像或抽样（如每十个输入事务进行一次抽样）决定。
- 仅仅运行处理功能的一个子集（刷新账目的历史记录，但是并不打印出每月的账单）。
- 可综合运行数据和处理功能子集。

部分并行运行总是伴随着可能存在未被检出的严重错误或问题的风险。例如，带有部分输入的并行运行增加了检测不出那些和未经测试的数据有关的错误的风险。

阶段部署

在**阶段部署**中，系统被部署并在一系列的步骤或阶段中投入运行。每一阶段都为运行的系统增加一些组件或功能。在每一阶段中，系统都要经过测试以确保为下一阶段做好准备。分阶段部署可以和并行部署结合使用，尤其是在新系统将要取代现有系统的运行阶段。

图 12-17 显示的是在独立的阶段存在直接和并行安装的阶段系统。新系统取代了两个现有系统，部署分为三个阶段。第一阶段是直接取代其中的一个现有系统，第二和第三阶段是取代另一个现有系统的并行部署的不同部分。

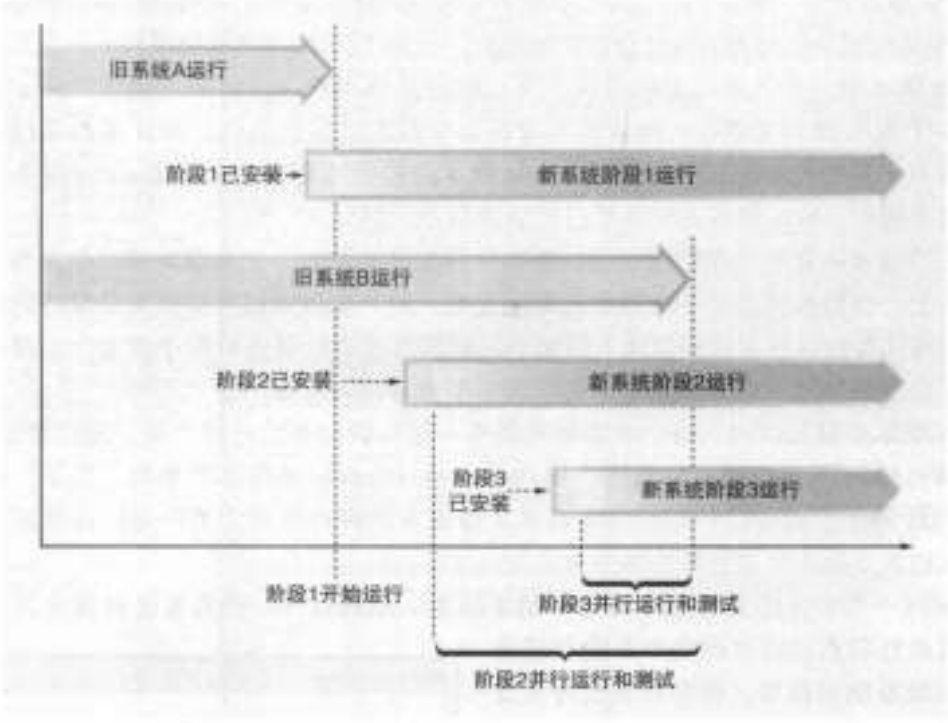


图 12-17　带有立即接入和并行运行的分阶段安装

阶段部署的主要优点是降低了风险，因为单个阶段的失败带来的问题要比整个系统失败带来的问题少。阶段部署的主要缺点是增加了复杂度，将部署分为几个阶段会产生更多的活动和里程碑，从而使整个过程更加复杂，然而，每个单独的阶段又存在许多更小的问题需要解决。如果系统是因为太大或太复杂而不能一次完成安装，那么采用分阶段安装降低风险的意义就显得更为重大，尽管这种安装会使得管理和协调多个阶段的复杂度有所增加。

12.4.4　改动和版本控制

改动和版本控制是管理软件开发、测试和部署的关键部分。尽管它不是实施或部署核心过程中的正式活动。中型和大型系统是很复杂的，而且会不断变化。在实施过程中改动发生得很快，在部署过程中发生得却很缓慢。系统复杂性和快速变化创造了一系列的管理问题，尤其是关于测试和后期部署支持的问题。

改动和版本控制工具和处理过程通过多个版本来处理与测试、支持系统相关的复杂性。工具和处理过程在系统周期的开始到结束期间通常都会包含到实施活动中。大多数组织机构为他们的所有系统都使用了一组常用的工具和程序。

版本

为简化测试、部署和支持的工作，复杂系统的开发、安装和维护是在一系列的版本中进行的。对于最终用户，在被开发的系统中有多个版本以及在不同的开发阶段有更多版本的情况并不常见。在开发过程中生成的版本称为测试版本。测试版本有一些完好定义的特征，代表了最终完成系统过程中坚实的一步。测试版本提供了一个静态的系统映像和用于评估项目进程的检查点。

alpha 版本是一个未完成的但是已经准备好接受严格的集成化或可用性测试的系统。多个 alpha 版本是根据系统的大小和复杂性来建立的。alpha 版本的生命周期很短——几天或几周。

beta 版本是一个足够稳定的系统，可以接受最终用户一段时间的测试。beta 版本是在一个或多个版本测试完毕后，确认已知错误都被改正之后产生的。最终用户通过使用 beta 版本实践真正的工作来测试它。因此，beta 版本必须更加完整，并且比 alpha 版本产生重大错误的概率要小。beta 版本通常要经过数周或数月的测试。

对用户发布的能够长期使用的系统版本被称为**产品版本**、**发布版本**或**产品发布**。尽管在传统意义上，软件系统几乎不可能在此期间完成，但产品版本通常被认为是最终的产品。最小产品版本（有时也称为**维护版本**）则允许纠错和对已有特征进行较小改动。主要产品版本则增加了许多新功能，也有可能是旧版本的全部重写。

版本跟踪是很复杂的。每个版本都需要唯一地标识出是给开发人员、测试者还是用户。对于在 Windows 下运行的应用程序，用户可以通过选择标准帮助菜单的“关于”选项以查看版本信息（如图 12-18 所示）。寻求技术支持或者想报告错误的用户可以根据这个特征向测试者或技术支持人员报告系统的版本。

控制同一个系统的多个版本需要有精密的版本控制软件，这通常会被建立在开发工具中，或者通过独立的源代码和本章先前描述的版本控制系统来获取。程序员和支持人员可以选取当前的版本或任何以前的版本来执行、测试或修改。修改以后会保存一个新的版本，这样可以使历史映像保持原样。

图 12-18　关于 Windows 应用显示样例

beta 版本和产品版本只要被安装在用户服务器或机器上，就必须存储。存储的版本用于测试未来的错误报告。例如，当用户报告 1.0 版本出错时，支持人员将从存档中调出此版本并安装，力图重现用户出过的错

误。反馈给用户的信息也是针对 1.0 版本而言的，即使最近的产品是更高版本的。

提交错误报告和改动请求

为控制因改动带来的风险，大多数组织对所有运行的系统采取正式的控制程序。设计正规控制是为了确保在实施之前，能够充分地描述、考虑和规划潜在的变化。典型的改动控制程序包括：

- 标准的报告方法。
- 项目经理或改动控制委员会会对申请进行审查。
- 对于可操作的系统，对设计和实施进行广泛计划。

图 12-19 显示的是一个错误报告，是由测试或系统开发人员填写的。在这种情况下，错误报告会被集成到应用程序工具和源代码控制系统中，这样做能使项目经理真正地管理所有报告、安排报告给特定的开发人员并且跟踪它的解决方法。

图 12-19　Visual Studio 的错误报告样例

相似的工具也可用于为可操作系统的新特征报告和管理错误和请求。在新特征的例子中，通常是提交请求给改动控制委员会，以便他们评估其对现有计算机硬件和软件、系统性能和可用性、安全性和操作预算的影响。被批准的改动将添加到预算、日程安排、计划和实施等问题待定的列表中。

实施改动

改动的实施就是遵循了 SDLC 的一个缩小版本。大多数 SDLC 活动都会执行，尽管它们会在范围上缩小或有时被取消了。实质上，维护改动是一种进一步的项目开发过程，这时的用户和技术需求要提前全部了解。分析活动通常会被省略或跳过，设计活动实质上是在范围上被缩小了，并且整个项目通常会在一次或两次迭代中完成。

改动计划包括以下活动：

- 确定系统的哪些部分需要改动。
- 保护实现改动所需的资源（例如人员）。
- 安排设计和实施活动。
- 为改动后的系统开发测试标准和测试计划。

只要可能，改动会在运行系统的拷贝版本上进行的。**产品系统**指的是用户天天使用的系统版本。测试系统指的是被修改后用来测试改动的产品系统的拷贝版本。测试系统的开发和测试可在单独的硬件或冗余系统中进行。测试系统只有在经过完全和成功的测试后才能成为可运行的系统。

12.5 整体回看——再访 RMO

在中型或大范围项目开发中，管理人员通常会因为要执行活动的数量、它们之间的依赖关系和包含的风险而感到不堪重负。在这部分中，我们将会给出这些问题互相作用的例子，即 Barbara Halifax 的团队如何为 RMO 的顾客支持系统开发迭代计划。但是要记住，没有简单的例子可以帮你为一个复杂的项目充分准备处理迭代计划。那也是为什么迭代计划和其他项目计划任务通常会让有着几年经验的开发人员来执行。

在阅读以下内容之前，你要回顾之前第 2、3、4、6、9 章中对 RMO 例子的描述。项目的一些基本参数已经描述过了，包括子系统边界、项目长度和迭代数量。

第 9 章描述了 Barbara 的早期计划决策。在这里，她要拓展那些决策，对早期决策做出一些改动，并做出一些附加决策，她还开发了图 12-20 中所示的修正过的迭代计划。接下来会讨论成为那个计划的基础的关键问题和决策。

12.5.1 更新或代替？

适当更新当前的 CSS 系统在项目计划的早期就被排除了，原因有：

- 当前的底层结构接近饱和。
- RMO 希望有一个外部供应商来托管 CSMS 系统，以此来节约资金。
- 现有的 CSS 和网页界面已经开发了 15 年。
- 当前的系统软件有几个版本已经过时了。
- 支持 CSS 的底层结构要重新定义才能拓展 SCM 容量。

总之，在不破坏操作系统的基础上更新当前的 CSS 太过复杂，而且更新旧的底层结构和应用软件的风险太大了。通过建立和部署一个全新的系统，RMO 会彻底清除现有的 CSS 和它的底层支持结构。一个新的被托管的底层结构会为 CSMS 而开发。在第一个部署阶段之后，现有的 CSS 底层结构会被更新以适合托管环境，并且作为以后的开发过程和部署活动的测试环境。

迭代	描述
1	定义业务模型和开发 / 部署环境。定义必要的用例和粗糙的类图。描述销售处理过程。完成部署环境。选择和获取网络组件、系统软件、硬件和开发工具。用最少的数据内容创建 CSS 数据库副本以作为 CSMS 数据库的开始。为添加顾客订单构造一个简单的原型（没有数据库更新）和执行可用性测试。
2	定义类、用例、顺序图和程序，关注最关键的用例（搜索商品、加入购物车、检查购物车、查找顾客和创建顾客账户）。到迭代的中期，部署底层结构组件，包括操作系统、Web/ 应用程序服务器和 DBMS。基于新定义的或修复过的类和关联来更新数据库模式。执行可用性、单元和集成测试来验证数据库设计、顾客 / 销售功能组和用户界面。
3	再一次查看第二次迭代的用例，并确保之前迭代中的所有改动已完成。拓展需求和设计来覆盖附加的销售用例以及必要的顾客账户和订单实施用例。执行可用性、单元和集成测试。
4	再一次查看第三次迭代中的用例，确认之前迭代中的所有改动。拓展需求和设计来覆盖关于产品和促销的剩余的营销子系统用例。开发面向顾客的在线帮助，所有功能要在先前的迭代中实施。准备培训材料、培训电话以及零售商店销售人员。完成新数据库并准备数据迁移。开发数据迁移程序（导入）。通过导出所有 CSS 数据库中的数据来测试和修改数据转移程序。
5	再一次查看第四次迭代中的用例，确定之前迭代中的所有改动。继续培训电话和零售商店销售人员。在大量真实或模仿的顾客之间组织可用性测试。对用户界面做出任何有需要的改动，包括在线帮助。组织执行和加强测试并且做出有必要的改动。在帕克城数据中心创建 CSMS 部署环境的副本，作为版本 2.0 开发的测试系统。组织用户可接受性测试。在最后的导出过程中导出所有 CSS 数据库改动。
6	监控系统执行过程和用户评价。开发改动列表和分类并将它分为“ASAP”或“版本 2.0”。实施 ASAP 改动。拓展需求和设计来覆盖报告子系统中必要的用例和那些关于社交网络的用例。从 CSMS 转移数据库更新到 CSS 数据库，每天两次。如果在 CSMS 中没有出现问题，那么在本次迭代的最后要终止数据转移和旧系统的操作。
7	再一次查看第 6 次迭代中的用例，确认之前迭代中的所有改动。拓展需求和设计来覆盖所有剩余的用例。更新数据库设计作为支持版本 2.0 的用例。为迭代 7 和用例进行编程，组织单元和集成测试。
8	开发面向顾客的在线帮助，所有的功能在第 6 个和第 7 个迭代中实施。准备培训材料并组织销售、营销和管理人员的培训。在大量真实或模仿的顾客之间组织可用性测试。对用户界面做任何有必要的改动，包括在线帮助。用测试数据库中的结构化改动来更新产品数据库。
9	继续销售、营销和管理人员的培训。组织执行和加强测试，并做出有必要的改动。组织用户可接受性测试。将版本 2.0 放入产品中。

图 12-20　CSMS 的迭代计划

12.5.2　最小化风险的阶段化部署

第 9 章描述的进度表不会调用阶段化部署，但是它也不会直接考虑像数据库开发、数据移动和培训这样的部署问题。为了最小化部署中的风险，CSMS 会在两个版本中进行部署。版本 1.0 会用最小的改动来重新实施大多数现有的 CSS 用例。版本 2.0 会合并版本 1.0 中修复好的问题和增加的改进，然后会添加 CSS 中没有的额外的功能，包括社交网络、反馈 / 建议、业务伙伴和山地雄鹿公司。

两个阶段的部署活动通过将一个大部署环境分成两个小部分来最小化项目风险。另一个降低风险的关键是作为备份维护当前 CSS 和它的数据库，至少是在版本 1.0 部署之后的一个迭代。如果一个严重的问题在版本 1.0 中出现，那么 RMO 可以恢复到当前的 CSS，通过直接访问网站链接回到内部服务器中。

12.5.3　数据库开发和数据转换

CSMS 的类图中的许多类都已经在现有的 CSS 数据库中存在。然而，对现有类来说，

有一些新类和关联以及改动。因此，新旧数据库之间某种程度的兼容性还不足以使当前数据库的更新版本直接与两个系统相连。因此，要建立新 CSMS 数据库，从 CSS 数据库中获取数据来部署版本 1.0。

数据库开发和转移到版本 1.0 的部署会发生在多个迭代中。迭代计划在项目早期会调用并创建 CSS 数据库副本，对它做出增量改动。在第 4 次迭代的最后，产品 CSS 数据库中的所有数据都会转移到 CSMS 数据库中。如果碰到问题，它们会在第 5 次迭代中被尽早解决并转移。在第 4 次迭代中转移的很多数据能够在第 5 次迭代中使用，即用真实数据来测试用户界面，这些数据来自从真实顾客、产品、系统和压力测试中获取的真实数据，构成一个"生产规模"数据库。

在第 5 次迭代的最后，所有的 CSS 数据库都会改变，因为最后全部的转移会被复制到 CSMS 数据库中。只需复制变化了的数据，数据转移过程在一个小时之内完成。CSS 系统会在转移期间脱机。一旦转移完成，CSMS 就会转换到。为了最小化风险，在第 5 次迭代中，额外的数据转换路径会复制两次新系统中的数据到 CSS 数据库中。如果发生意外，CSS 可以用当前完整的数据库进行重启。如果 CSMS 版本 1.0 在第 5 次迭代中通过了所有用户可接受性测试，那么 CSS 会被关闭并且数据转移会终止。

12.5.4　开发顺序

IPO 开发顺序是开发计划中的主要基础。通过开始于 CSS 数据库的副本，一组测试数据会在第 1 次迭代中生成，因此能首先处理最高风险的用例。这些部分包括整个销售子系统和订单实施子系统中面向顾客的部分。这些风险来自于新技术、需求的不确定性和 RMO 的销售与订单实施的重要操作。通过首先处理那些用例，Barbara 允许她的开发员工花费大量时间来解决不确定性并测试相关软件。要注意的是，对这些功能的大量测试开始于第 2 次迭代，然后基本持续在整个项目的开发过程中。

12.5.5　文档和培训

对于两种产品版本来说，培训活动是贯穿于后期的项目迭代中的。在部署之前，初始培训练习包含了系统中风险最高的部分。它们也能使开发人员在部署之前进行集成化工作和执行与销售相关用例的测试。当新功能被添加到系统时，额外的培训也要继续，以不断提升用户的技能和开发人员的工作量。

本章小结

实施和部署是很复杂的处理过程，因为它们由很多相互依赖的活动组成。测试是实施和部署中的一个关键活动。软件组件必须按顺序构造，这样能使开发资源的使用最小化，同时使测试系统与纠正错误的能力最大化。遗憾的是，那两个目标通常是相互冲突的。因此，程序开发计划是在可利用资源、可利用时间和期望之间的权衡，以此来发现并改正错误，而这是优先于系统部署的。

部署和改动管理活动在多个系统版本中跟踪模型和软件的改动，这能使开发人员按照层次来测试并部署系统。版本也能通过使开发人员跟踪特定系统版本的问题支持来提高后期部署支持。源代码控制系统使得开发团队能够协调他们的工作。

复习题

1. 列出并简述 SDLC 核心过程“建立、测试和集成系统组件、完成系统测试和部署”中的每个活动。
2. 定义术语单元测试、集成测试、系统测试和用户可接受性测试。执行的每个测试类型是在哪个 SDLC 活动中？
3. 什么是测试用例？一个好的测试用例的特征是什么？
4. 什么是驱动程序？什么是桩程序？与它们联系最紧密的分别是什么类型的测试？
5. 列出用于初始化新系统中数据库的可能数据来源。简述用于数据库初始化数据装载的工具和方法。
6. 最终用户和系统操作员之间的用户文档和培训活动的区别是什么？
7. 列出并简述三个基础的程序开发顺序方法。每个方法的优点和缺点是什么？
8. 自上而下和自下而上的开发顺序的概念要如何应用到面向对象软件中？
9. 什么是源代码控制系统？在多个程序员建立一个程序或系统时，为什么这样的系统是有必要的？
10. 简述直接、并行和阶段部署。每个部署方法的优点和缺点是什么？
11. 定义术语 alpha 版本、beta 版本和产品版本。在 alpha 版本成为 beta 版本或 beta 版本成为产品版本时，有没有定义好的用于决策的标准？

问题和练习

1. 用 IPO、自上而下和自下而上的开发顺序描述测试软件的处理过程。哪个开发顺序能使得测试所需的资源最少？在每个开发顺序中，哪种类型的错误可能最先被发现？哪种开发顺序是最好的？（通过所需测试资源的结合和捕捉早期测试过程中的重要错误的能力来衡量。）
2. 假设你和三个同学负责开发第一个原型来实施 RMO 用例创建 / 更新顾客账户。创建一个开发和测试计划并且写下测试的类和方法。假设你有两周的时间来完成所有任务。
3. 与一个计算机中心或 IS 管理人员进行交流，讨论关于一个最近才被部署的系统或子系统中用到的测试过程。哪种类型的测试被执行了？测试用例和测试数据是如何生成的？哪种类型的团队开发和实施测试？
4. 试想只使用由一个集成开发工具开发的电子模型来记录系统的问题，如 Microsoft Visual Studio 或 Oracle JDeveloper。优点是显而易见的（如，分析员修改了模型来反映新需求和自动化生成一个已更新的系统），但是是否存在缺点？（提示：系统可能会被维护一段时间或更长。）
5. 与你所在学校的最终用户进行交谈，或者是参考近期安装或发布的业务应用程序所提供的培训和文档。需要提供哪种类型的培训和文档？用户认为培训充分吗？用户认为文档有用且完备吗？
6. 假设你负责一个新系统的实施与部署，这个新系统是要替换一个 24 小时使用的关键现有系统。为了使风险最小，在最新子系统部署完成之前，你要计划超过六周的新子系统的阶段部署和至少三周的并行部署操作。因为没有足够的人力资源来操作两个系统，所以在并行操作期间，你要计划雇佣 30 个临时员工。你要如何利用这些员工？回答时要考虑以下这些问题：

- 某些员工在子系统部署前要进行培训，并且那些员工要培训其他员工。
- 刚被培训过的员工可能要在几周之后才能达到完全有效的工作效率。

扩展资源

Robert V. Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley, 2000.

Mark Fewster and Dorothy Graham, *Software Test Automation*. Addison-Wesley, 1999.

Jerry Gao, H.-S. Jacob Tsao, and Ye Wu, *Testing and Quality Assurance for Component-Based Software*, Artech House Publishers, 2003.

William Horton, *Designing and Writing Online Documentation: Hypermedia for Self-Supporting Products* (2nd edition). John Wiley & Sons, 1994.

William Horton, *Designing Web-Based Training: How to Teach Anyone Anything Anywhere Anytime*. John Wiley & Sons, 2000.

William Horton, *e-Learning by Design*. Pfeiffer, 2011.

International Association of Information Technology Trainers (ITrain) Web site, http://itrain.org.

David Yardley, *Successful IT Project Delivery*. Addison-Wesley, 2002.