CHAPTER

第 3 章 面向对象开发方法

面向对象(Object-Oriented,OO)方法的应用非常广泛,已经扩展到数据库系统、分布式系统、CAD 系统、人工智能系统等的研发、管理与维护。面向对象方法注重在软件研发过程中面向客观现实世界或问题域中的事物,采用人类在认识客观世界的过程中习惯的思维方式,更加直观、自然地描述客观世界中的有关事物,成为一种快速高效的软件开发方法。

教学课件

第 3 章 面向对象开发方法

教学目标

(1) 掌握面向对象方法的有关概念和特点。

(2) 理解面向对象软件的主要开发任务及过程。

(3) 掌握面向对象分析和面向对象设计的方法。

(4) 应用面向对象分析和面向对象设计的方法。

3.1 面向对象的相关概念

【案例 3-1】 面向对象方法是主流软件开发方法。

(1) 从世界观的角度认为:世界是由各种具有各自运动规律和内部状态的对象组成,不同对象之间的相互作用和通信构成了完整的现实世界。人类应当按照现实世界的本来面貌理解世界,直接通过对象及其相互关系反映世界,以此构建的系统才能符合现实世界。

(2) 从方法学的角度认为:面向对象方法是面向对象的世界观在开发方法中的直接运用,强调系统的结构应该与现实世界的结构相对应,应该围绕现实世界中的对象构造系统,而不应围绕功能构造系统。

知识拓展

面向对象方法论的产生

3.1.1 对象与类

1. 对象及其三要素

对象(Object)是系统中的基本运行的实体(如窗口、控件),是代码和数据的集合,即现实世界中的具体事物,是构成软件系统的基本单位。面向对象系统是数据抽象与过程抽象的综合。面向对象方法以对象分解代替传统方法的功能分解。面向对象的系统由对象组成,复杂的对象由简单的对象组合而成。**对象具有三要素**:对象标识、属性和服务。其中,**对象标识**即对象的名称,**用于**唯一地识别系统内部对象,在定义或使用对象时指定。**属性**(Attribute)也称为状态(State)或数据,**用于**描述对象的静态特征。在某些面向对象程序设计(Object-Oriented Programming,OOP)语言中,属性通常被称为**成员变量**(Member Variable)或简称**变量**(Variable)。**服务**(Service)也称为**操作**(Operation)、**行为**(Behavior)或**方法**(Method)等,用于描述对象的动态特征,在某些 OOP 语言中,服务通常被称为**成员函数**或简称为**函数**。

知识拓展

面向对象的三大特征

2. 封装的概念及含义

封装(Encapsulation)是指将软件模块内部具体实现进行隐藏,将数据与操作数据的源代码进行有机结合,形成“类”,类的成员包括数据和函数。封装将抽象得到的数据和行为(或功能)结合,形成一个有机的整体。封装是对象的一个重要特性,在面向对象的系统中,对象是一个封装了数据属性和操作行为的实体。使用某一对象时,只需知道其向外界提供的接口形式,不需要知道其数据结构细节和实现操作的算法。**封装具有两层含义**:一是对象是其全部属性和服务紧密结合而形成的一个整体;二是对象如同一个密封的“黑盒子”,表示对象状态的数据和实现操作的代码都被封装在其中。封装的好处有三点:一是将类的内部实现细节与外界隔离,减少内部修改的副作用;二是方便类的使用;三是通过接口限制对数据属性的操作,提高了安全性。

知识拓展

对象的两个视图与封装

3. 类和实例的概念

类(Class)也称为**对象类**(Object Class),是对具有相同属性和服务的一组对象的抽象定义。类与对象是抽象描述与具体实例的关系,一个具体的对象被称为**类的一个实例**(Instance)。对象的状态包含在实例的属性中。

【案例 3-2】 由数据结构教材、软件工程教材、数据库原理教材等具体教材,可以得到“教材”类,而这些具体的教材就是“教材”类的实例,如图 3-1 所示。

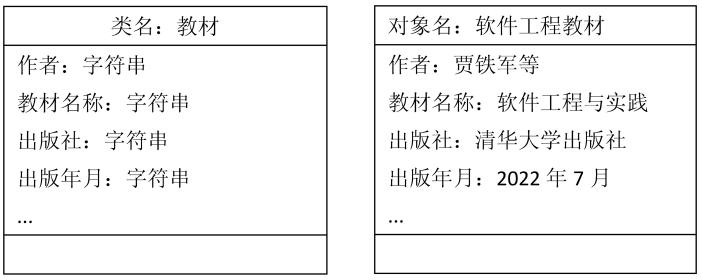


图 3-1　类与实例

类定义了各个实例所共有的结构，类的每个实例都可使用类中定义的属性和操作。实例的当前状态由实例的属性值确定，属性值的变化由所执行的操作定义。通常类可视为一个抽象数据类型(ADT)的实现，也可将类看作表示某种概念的一个模型。类实际是单个的语义单元，可以使开发者自然地管理系统中的对象，匹配数据定义及操作。类加入操作给类赋予语义，可以提供各种级别的可访问性。

3.1.2　继承及多态性

1. 继承的概念和种类

继承(Inheritance)是父类和子类之间共享数据结构和方法的一种机制，是以现存的定义的内容为基础，建立新定义内容的技术，是类之间的一种关系。**继承有两种**：一是单重继承，指子类只继承一个父类；二是多重继承，指子类继承了多个父类。继承性通常表示父类与子类的关系，如图 3-2 所示。

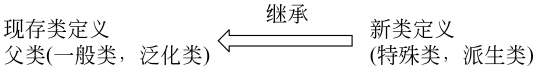


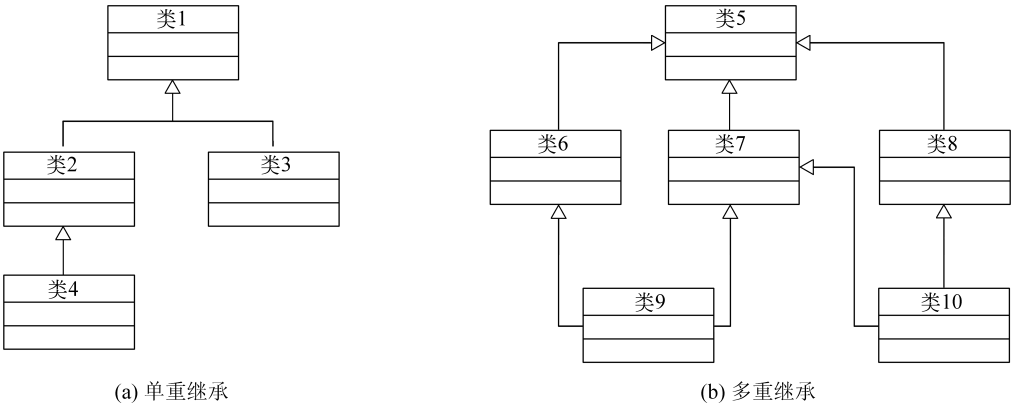
图 3-2　继承性

图 3-3 是继承性描述的一种图示方法。通过继承关系还可构成层次关系，单重继承构成的类之间的层次关系为树状，多重继承构成的类之间的关系为网格状，而且继承关系可传递。

建立继承结构的优点有 3 个：一是易编程、易理解且代码短，结构清晰；二是易修改，共同部分只在一处修改即可；三是易增加新类，只需描述不同部分。

2. 多态性和动态绑定

(1) **多态性**(Polymorphism)是指具有继承关系的不同类的对象在相同的操作或函数、过程中取得不同结果的特性。利用多态技术，用户可发送一个通用消息，而实现的细节则由接收对象自行决定，这样同一消息就可调用不同的方法。多态性不仅增加了面向对象软件的灵活性，进一步减少了信息冗余，而且可显著提高软件的可重用性和可扩充



(a) 单重继承　　(b) 多重继承

图 3-3　继承性描述

性。多态有多种不同形式，其中参数多态和包含多态统称为**通用多态**，过载多态和强制多态则统称为**特定多态**。

(2) **动态绑定**(Dynamic-binding)是多态性的基石之一。将函数调用与目标代码块的连接延迟到运行时进行，只有发送消息时才与接收消息实例的一个操作绑定。动态绑定同多态性可使建立的系统更灵活且便于扩充。

3.1.3　消息与方法

1. 消息与消息通信

(1) **消息**(Message)是向对象发出的服务请求，**包含的信息**有：提供服务的对象标识、消息名、输入信息和回答信息。对象与传统的数据有本质区别，不是被动地等待外界对其进行操作，而是进行处理的主体，必须对其发送消息请求以执行其某个操作，处理其私有数据，而不能从外界直接对其私有数据进行操作。

(2) **消息通信**(Communication with Messages)与对象的封装原则密切相关。封装使对象成为各司其职、互不干扰的独立单位；消息通信则为其提供唯一合法的动态联系途径，使其行为可以互相配合，构成一个有机的系统。

2. 方法的基本概念

方法(Method)是指对象内的操作。属性描述对象的状态，操作可操纵私有属性，改变对象的状态。当收到其他对象发送的消息并响应时，其操作才得以实现。方法是类中操作的实现过程，**方法包括**方法名、参数及方法体。方法描述了类与对象的行为，每个对象都封装了数据和算法两方面，数据由一组属性表示，算法则是当一个对象接收到一条消息后，所对应的方法执行的动作。消息与方法之间存在一一对应关系，通过向对象发送消息调用相应的方法。

讨论思考

(1) 什么是对象及其三要素？什么是类及实例？

(2) 怎样理解继承及多态性？举例说明。

(3) 举例说明消息与方法及其之间的关系。

3.2 面向对象方法概述

3.2.1 面向对象方法的概念

关于**面向对象**的定义,Coad 和 Yourdon 将其表述为:面向对象=对象+类+继承+消息通信。

面向对象方法(Object-Oriented Method,OOM)是指具有上述 4 个概念的软件开发方法。面向对象方法是面向对象技术和方法在软件工程中的全面运用,主要包括面向对象分析(Object-Oriented Analysis,OOA)、面向对象设计(Object-Oriented Design,OOD)、面向对象编程(Object-Oriented Program,OOP)、面向对象测试(Object-Oriented Test,OOT)和面向对象维护等方法,在此主要介绍前面两部分。面向对象方法的出发点和基本原则是使软件开发方法和过程尽可能接近人类认识现实世界解决问题的思维方式。只有同时使用对象、类、继承与消息通信,才能体现面向对象的特征和方法。

知识拓展
面向对象方法的形成

3.2.2 面向对象方法的特点

面向对象的开发方法(Object-Oriented Software Development,OOSD)的基本思想是尽可能按照人类认识世界的方法和思维方式分析和解决问题,可提供更加清晰的需求分析和设计,是一种主流的软件开发方法。OOSD 贯穿于整个软件生命期,其中面向对象的分析与设计是面向对象开发的关键。**OOM** 具有 4 个主要特点。

(1) 符合人类分析解决问题的习惯思维方式。OOM 以对象为核心,强调模拟现实世界中的概念而非算法,尽量用符合人类认识世界的思维方式渐进地分析解决问题,使问题空间与解空间一致,有利于对开发过程各阶段综合考虑,有效地降低开发复杂度,提高软件质量。

(2) 各阶段所使用的技术方法具有高度连续性。传统的软件开发过程用瀑布模型等描述,其主要缺点是将充满回溯的软件开发过程硬性地分为几个阶段,而且各阶段所使用的模型、描述方法不相同。而 OOM 使用喷泉模型作为其工作模型,软件生存期各阶段无明显界限,开发过程回溯重叠,用相同的描述方法和模型保持连续。

知识拓展
传统软件开发方法

(3) 开发阶段有机集成有利于系统稳定。将 OOA、OOD、OOP 有机集成,始终围绕建立问题领域的对象(类)模型进行开发过程,而各阶段解决的问题又各有侧重。由于构造软件系统以对象为中心,而不是基于对系统功能分解,所以当功能需求改变时不会引起其结构变化,使其具有稳定性和可适应性。

(4) 重用性好。利用复用技术构造新软件具有很大灵活性,因为对象所具有的封装性和信息隐蔽性,使对象的内部实现与外界隔离,具有较强的独立性,所以,对象类提供了较理想的可重用软件成分,而其继承机制使得面向对象技术实现可重用性更方便、自然和准确。

3.2.3 面向对象开发过程及范型

1. 面向对象开发过程

OOM 不但是一些具体的软件开发技术与策略,而且是一整套处理软件系统与现实世界的关系并进行系统构造的软件方法学。面向对象软件的开发过程与其他方法不同,从问题论域开始,历经从问题提出到问题解决的一系列过程。**面向对象开发过程**分为如下 6 个阶段。

(1) 分析阶段。**包括两个步骤**:论域分析和应用分析。标识问题论域中的抽象,在分析时找到特定对象,基于对象的公共特性将其组合成集合,标识出对此问题的一个抽象,并标识抽象之间的关系,建立对象之间的消息连接。

① 论域分析。主要利用开发问题论域模型,在应用分析前进行论域分析,对问题集思广益,考查问题论域内的较宽范围,分析的覆盖范围应比直接解决问题更广泛。

② 应用分析。应用(或系统)分析细化在论域分析阶段所开发的信息,并将注意力集中在要解决的问题上。通过论域分析,分析人员可具有较宽的论域知识,有助于更好地抽象。

(2) 高层设计。在 OOD 中,软件体系结构设计与类设计常为同样的过程,但还应将体系结构设计与类设计分开。在高层设计阶段,设计应用系统的顶层视图。如同开发一个代表系统的类,通过建立该类的一个实例并发送给它一个消息以完成系统的“执行”。

(3) 开发类。主要依据高层设计所标识的对各类的要求和类的规格说明,进行类开发。由于一个应用系统通常是一个类的继承层次,对这些类的开发是最基本的设计活动。

(4) 建立实例。建立各对象的实例,实现问题的解决方案。

(5) 组装测试。按照类与类之间关系在组装一个完整的应用系统的过程中进行测试。各类的封装和类测试的完备性可减少组装测试所需的成本。

(6) 维护。维护的要求将影响应用和各类。继承关系可支持对现有应用的扩充,或加入新的行为,或改变某些行为的工作方式。

2. 面向对象的软件开发范型

(1) 改进了传统软件开发方法。结构化方法在控制问题求解的规模和复杂度、提高软件系统的易理解性等方面起到了重要作用,但是,这种方法很难实现软件重用,导致软件生产效率低下,质量难以保证且难以维护。

(2) 面向对象的软件开发方法具有同传统软件开发方法一样的步骤,同样要经历分析、设计、编码和测试的生命周期。在软件开发的每个阶段中都运用了面向对象的思想。面向对象技术使软件构件可以方便地复用,特别是基于程序构件的复用,通过组装可重用的构件快速地开发新软件系统。

(3) 大部分**面向对象软件开发模型**都包括如下内容。

① 分析用户的需求,提炼对象。

② 将现实中问题论域的对象抽象成计算机软件中的对象。

③ 分析并描述对象之间的关系。

④ 根据用户的需求,不断地修改并完善。

3.2.4 面向对象开发方法

1. 面向对象软件工程方法

面向对象软件工程(OOSE)方法是 I. Jacobson 在 1992 年出版的专著《面向对象的软件工程》中提出的。OOSE 方法采用 5 类模型建立目标系统,将面向对象的思想应用于软件工程中。这 5 类**模型**如下。

(1) 需求模型(Requirements Model,RM)。主要用于获取用户的需求、识别对象,主要的描述手段有用例图(Use Case)、问题域对象模型及用户界面。

知识拓展

用例对 OOSE 的重要作用

(2) 分析模型(Analysis Model,AM)。主要用于定义系统的基本结构。将 RM 中的对象分别识别到 AM 中的实体对象、界面对象和控制对象 3 类对象中。每类对象都有各自的任务、目标并模拟系统的某一方面。

(3) 设计模型(Design Model,DM)。AM 只注重系统的逻辑构造,而 DM 需要考虑具体的运行环境,将在分析模型中的对象定义为模块。

知识拓展

分析模型的实体对象

(4) 实现模型(Implementation Model,IM)。即用面向对象语言来实现。

(5) 测试模型(Testing Mode,TM)。测试的重要依据是 RM 和 AM,测试的方法与技术同第 7 章介绍的类似,而底层是对类(对象)的测试。TM 实际上是一个测试报告。

OOSE 的开发活动主要分为分析、构造和测试 **3 个过程**,如图 3-4 所示。其中,**分析过程分为**需求分析和健壮分析(Robustness Analysis)两个子过程,分析活动分别产生需求模型和分析模型。**构造活动**包括设计和实现两个子过程,分别产生设计模型和实现模型。测试过程包括单元测试、集成测试和系统测试 3 个过程,共同产生测试模型。

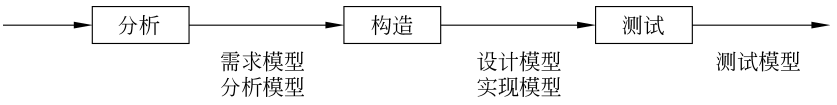


图 3-4 OOSE 的开发活动

2. 常见的面向对象开发方法

目前,面向对象开发方法的研究日趋成熟,已有很多面向对象产品问世。其**开发方法**有 Booch 方法、Coad 方法、OMT 方法和 UML 等。

(1) Booch 方法。Booch 最先描述了面向对象的软件开发方法的基础问题,指出面向

对象开发是一种根本不同于传统的功能分解的设计方法。面向对象的软件分解更接近人对客观事物的理解，而功能分解只通过问题空间的转换获得。

(2) Coad 方法。Coad 方法是 Coad 和 Yourdon 于 1989 年提出的面向对象开发方法。该方法的主要优点是通过多年来大系统开发的经验与面向对象概念的有机结合，在对象、结构、属性和操作的认定方面，提出了一套系统的原则。该方法完成了从需求角度进一步进行类和类层次结构的认定。尽管 Coad 方法没有引入类和类层次结构的术语，但事实上已经在分类结构、属性、操作、消息关联等概念中体现了类和类层次结构的特征。

(3) OMT 方法。对象建模技术(Object Modeling Technique，OMT)是美国通用电气公司提出的一套系统开发技术。它以面向对象的思想为基础，通过对问题进行抽象，构造出一组相关的模型，从而能够全面地捕捉问题空间的信息。该方法是一种新兴的面向对象的开发方法，开发工作的基础是对真实世界的对象建模，然后围绕这些对象使用分析模型来进行独立于语言的设计。面向对象的建模和设计促进了对需求的理解，有利于开发出更清晰、更容易维护的软件系统。该方法为大多数应用领域的软件开发提供了一种实际的、高效的保证，努力寻求一种问题求解的实际方法。

(4) UML。1995—1997 年软件工程领域取得重大进展，其成果超过软件工程领域过去十多年的总和，最重要的成果之一是统一建模语言(Unified Modeling Language，UML)的出现。UML 成为面向对象技术领域内占主导地位的标准建模语言，是一种定义良好、易于表达、功能强大且普遍适用的建模技术和方法，融入了软件工程领域的新思想、新方法和新技术。其作用域不限于支持面向对象的分析与设计，还支持从需求分析开始的软件开发全过程。不仅统一了 Booch 方法、OMT 方法、OOSE 方法的表示方法，而且对其做了进一步的发展，最终成为大众接受的标准建模语言。具体将在后续内容中概述。

讨论思考

(1) 面向对象包括哪些主要概念？具体含义是什么？

(2) 面向对象具有哪些特征？

(3) 面向对象的软件开发过程是怎样的？

3.3 面向对象分析

面向对象分析(OOA)的**目标**是获取用户需求并建立一系列问题域的精确模型，描述满足用户需要的软件。OOA 所建立的模型应表示出系统的数据、功能和行为 3 方面的基本特征。先要进行调研分析，在理解需求的基础上建立并验证模型。对复杂问题的建模，需要反复迭代构造模型，先构造子集，后构造整体模型。

3.3.1 面向对象分析的任务

OOA 的**关键**是定义所有与待解决问题相关的类，包括类的操作和属性、类与类之间

的关系及其表现出的行为，主要完成 6 项任务。

(1) 全面深入调研分析，掌握用户各项业务需求的细节及来龙去脉。

(2) 准确标识类，包括定义其属性和操作。

(3) 认真分析定义类的层次关系。

(4) 明确表达对象与对象之间的关系(对象的连接)。

(5) 对对象的行为进行初步建模。

(6) 建立系统模型。反复运用前面的过程，通过上述分析，建立系统的 3 种模型：描述系统数据结构的对象模型，描述系统控制结构的动态模型，描述系统功能的功能模型。主要从不同侧面描述或表示系统的内容，以及相互影响、相互制约、有机结合，全面表达对目标系统的需求。

3.3.2 面向对象分析的过程

OOA 是利用面向对象的概念和方法为软件需求建造模型，使用户需求逐步精确化、一致化、完全化的分析过程，也是提取需求的过程，主要包括理解、表达和验证 3 个过程。通常，由于现实世界中的问题较为复杂，分析过程中的交流又具有随意性和非形式化等特点，软件需求规格说明的正确性、完整性和有效性需要进一步验证，以便及时进行修正。

OOA 中构建的模型主要有对象模型、动态模型和功能模型 3 种。其**关键**是识别出问题域中的对象，在分析其之间相互关系的基础上，建立问题域的简洁、精确和可理解的模型。**对象模型常由 5 个层次组成**：类与对象层、属性层、服务层、结构层和主题层，其层次对应着 OOA 过程中建立对象模型的 5 项主要活动：发现对象、定义类、定义属性、定义服务、划分结构。面向对象分析过程如图 3-5 所示。

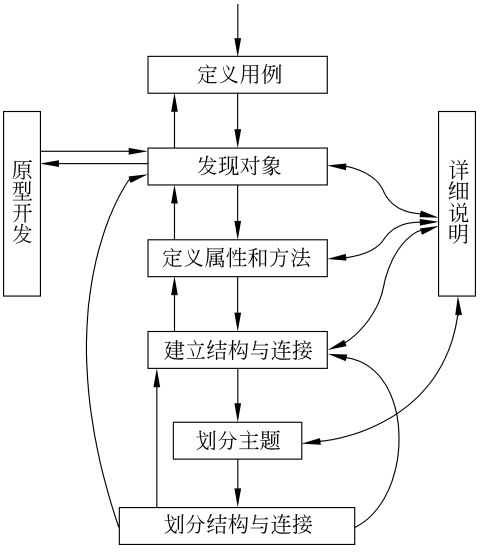


图 3-5 面向对象分析过程

3.3.3 对象建模技术

对象建模技术(Object Modeling Technique,OMT)主要用于 OOA、系统设计和对象级设计。可将分析时获取的需求信息构建在对象模型、功能模型和动态模型 3 类模型中。各模型分别侧重系统的一方面,从不同角度构成了对系统的完整描述,解决了对象模型定义“对谁做”,状态模型定义“何时做”,功能模型定义“做什么”的问题。

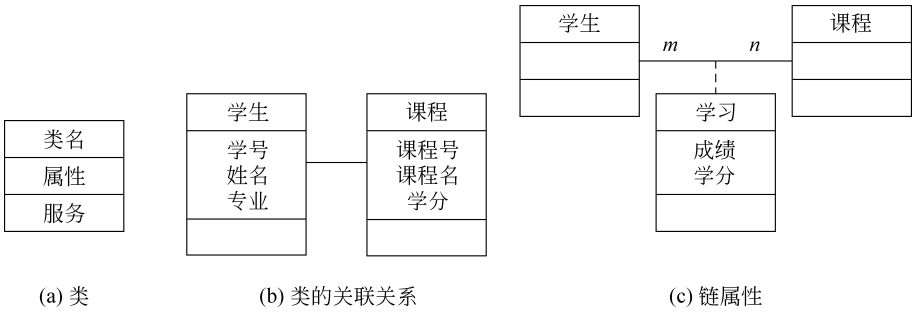
1. 对象模型的建立

对象模型是 OOA 最关键的模型之一,主要描述系统中对象的静态结构、对象之间的关系、对象的属性和操作。利用包含对象和类的关系图表示,通过表示静态的、结构上的、系统的“数据”特征,为动态模型和功能模型提供基本框架。

对象模型的建立需要先确定对象和类,然后分析对象的类及其相互关系。**对象类与对象间的关系可分为 3 种**:一般-特殊(继承或归纳)关系、聚集(组合)关系和关联关系。对象模型用类符号、类实例符号、类的继承关系、聚集关系和关联等表示。有些对象具有主动服务功能,称为**主动对象**。对复杂系统,可划分主题并画出主题图,这样有助于对问题的理解。

对象模型描述系统的静态结构包括:类和对象,它们的属性和操作,以及它们之间的关系。构造对象模型的目的是找出与应用程序密切相关的概念。

【案例 3-3】 对象模型以包含的对象及其关系图表示。在对象模型中用于表示“类、类的关联关系和链属性”的图形符号,分别如图 3-6(a)、图 3-6(b)和图 3-6(c)所示。其中,类的关联关系反映对象之间相互依赖及作用关系,链属性是关联中链(实例对象间的物理或概念上的连接)的性质。



(a) 类　　(b) 类的关联关系　　(c) 链属性

图 3-6　表示类、类的关联关系和链属性

利用 OMT **建立对象模型的主要步骤**如下。

(1) 确定对象类。通过分析确定所有的对象类。

(2) 定义数据词典。主要用于描述类、属性和关系。

(3) 组织并简化对象类。通过继承进行组织和简化对象类。

(4) 测试访问路径。测试所有的访问路径。

(5) 对象分组建立模块。由对象之间的关系和对象的功能将对象进行分组,并建立模块。

2. 动态模型的建立

动态模型主要用于系统的控制逻辑,注重对象及其关系的改变,描述涉及时序和改变的状态。**主要包括**状态图和事件跟踪图。状态图是一个状态和事件的网络,主要描述每一类对象的动态行为。**事件跟踪图**则主要说明系统执行过程中的一个特定"场景",也称为**脚本(Scenarios)**,是完成系统某个功能的一个事件序列。脚本通常从一个系统外部的输入事件开始,以一个系统外部的输出事件结束。**建立动态模型的主要步骤**如下。

(1) 准备场景。为典型的交互序列准备好场景。

(2) 建立事件跟踪图(UML 顺序图、通信图)。确定对象之间的事件,为每个场景建立事件跟踪图。

(3) 绘出事件流程图。为每个系统准备一个事件流程图。

(4) 建立状态图。为具有重要动态行为的对象建立状态图。

(5) 检验。检验不同状态图中共享的事件的一致性和完整性。

【案例 3-4】 宾馆信息系统中旅客和床位的状态转换图,如图 3-7 和图 3-8 所示。

图 3-7 旅客的状态转换图

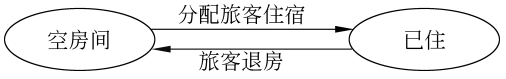


图 3-8 房间的状态转换图

3. 功能模型的建立

功能模型主要用于系统内部数据的传送和处理。功能模型表明,经过处理,从输入数据能得到具体的输出数据,但忽略参加处理的数据以怎样的时序执行。功能模型由多个数据流图**组成**,指明从外部输入,通过操作和内部存储,直到外部输出的整个数据流情况。功能模型还包括了对象模型内部数据间的限制。

功能模型中的数据流图可形成一个层次结构,一个数据流图中的过程可由下一层的数据流图做进一步的说明。**建立功能模型的主要步骤**如下。

(1) 确定输出和输出值。

(2) 用数据流图表示功能的依赖性。

(3) 具体描述每个功能。

(4) 确定具体限制。

(5) 确定功能优化的准则并实施优化。

【案例 3-5】 宾馆客房信息系统数据流图，如图 3-9 所示。

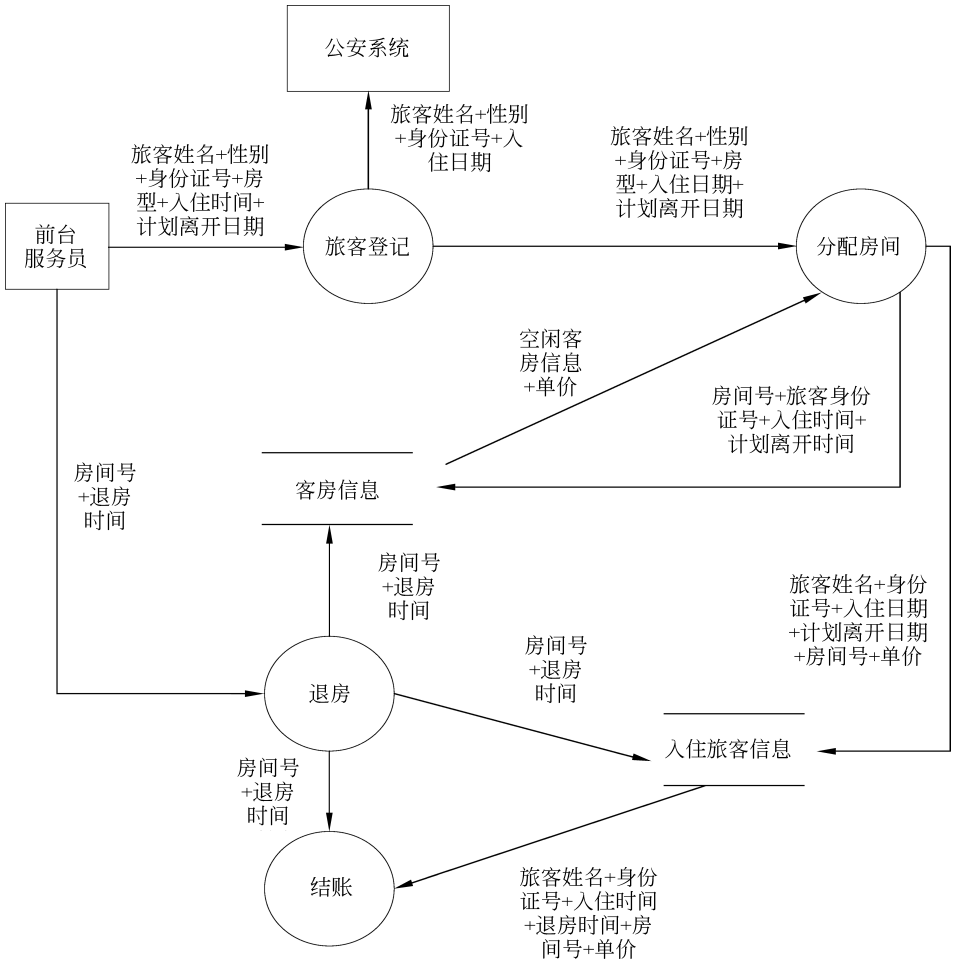


图 3-9 宾馆客房信息系统数据流图

* 3.3.4 UML 概述

统一建模语言(UML)又称为标准建模语言，是一种定义良好、易于表达、功能强大且普遍适用的结构化建模语言；融入了软件工程领域的新思想、新方法和新技术；支持从需求分析开始的软件开发的全过程；目标是用面向对象的图形方式来描述系统。

1. UML 的体系结构及模型元素

UML 是综合 OOM 中使用的各种图形描述的技术，旨在给出这些图形描述的语法和语义的语言，是一种标准的图形化(即可视化)建模语言。从语法和语义上，UML 由图和元模型**构成**，图是 UML 的语法，而元模型给出图的含义，称为 UML 语义。

1) UML 的体系结构

UML 2.0 的体系结构如图 3-10 所示。从体系结构上,UML 由 3 部分组成:基本构造块、规则和公用机制。**基本构造块包括 3 种类型**:事物、关系和图。其中**事物划分为 4 类**。

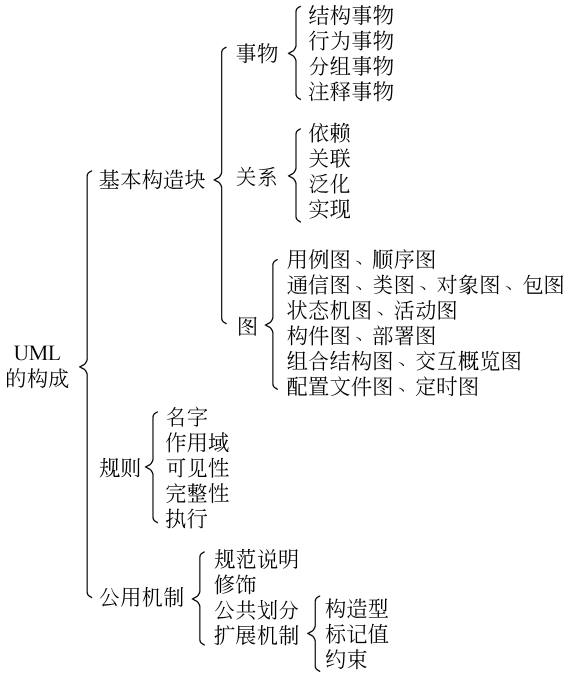


图 3-10　UML 2.0 的体系结构

(1) 结构事物。包括类、接口、协作、用例、主动类、组件和节点。

(2) 行为事物。包括交互机和状态。

(3) 分组事物。UML 中的分组事物是包。整个模型可以看成是一个根包,它间接包含了模型中的所有内容。子系统是另一种特殊的包。

(4) 注释事物。注释主要用于给建模者提供有关的具体说明信息,提供关于任意信息的文本说明,但无语义作用。

2) UML 的语义和模型元素

UML 是专门设计的一种统一描述面向对象方法的符号系统。

(1) UML 的语义。UML 是一种基于面向对象的可视化建模语言,语义被定义在一个 4 层(4 个抽象级别)的建模概念框架中。

① UML 的基本元模型层。由 UML 最基本的元素“事物”组成,代表要定义的所有事物。

② 元模型层。由 UML 的基本元素组成,包括面向对象和面向构件的概念。

③ 模型层。由 UML 模型组成,这一层的每个概念都是元模型层中概念的实例。此层的模型通常称为类模型或类型模型。

④ 用户模型层。由 UML 模型的例子组成,此层中的每个概念都是模型层的一个实例,也是元模型层概念的一个实例。这一层的模型通常称为对象模型或实例模型。

UML 用图形符号隐含表示了模型元素的语法,用这些图形符号组成元模型表达语

义，组成模型描述系统结构（或称为**静态特征**）以及行为（或称为**动态特征**）。

（2）UML 模型元素。UML 定义了两类模型元素：一类模型元素用于表示模型中的某个概念，如类、对象、用例、节点、构件、包、接口等；另一类模型元素用于表示模型元素之间相互连接的关系，主要有关联、泛化（表示一般与特殊的关系）、依赖、聚集（表示整体与部分的关系）等。**模型元素图形表示**如图 3-11 所示。

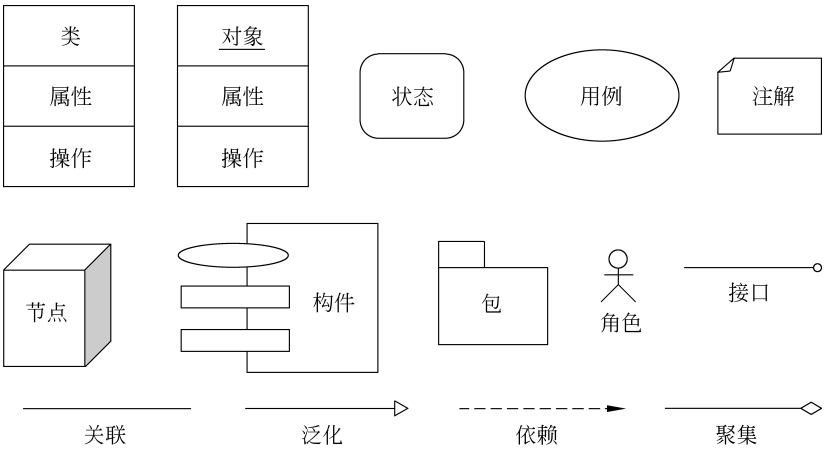


图 3-11　模型元素图形表示

（3）UML 模型图及表示法。模型通常以一组图进行表示，UML 2.5 模型图有两大类（结构图和行为图）14 种图形（见图 3-10）。各种图的作用见表 3-1。

表 3-1　UML 图的分类、用途及主要概念

图分类	图　名　称	用　　途	主 要 概 念
结构图	类图（Class Diagram）	以类和接口以及它们的特征、约束和关系（关联、泛化、依赖等）的形式显示所设计的系统、子系统或构件的结构	类、接口、特征、约束、关联、泛化、依赖
	对象图（Object Diagram）	显示一组相关对象在系统运行的某个时间点的详细状态	实例规范、对象、插槽、链接
	包图（Package Diagram）	包用于对图进行分组，包图显示包中的元素及包之间的关系	包、可打包元素、依赖关系、元素导入、包导入、包合并
	组合结构图（Composite Structure Diagram）	用来显示类目（classifier）的内部构造（类、接口、构件）及一次协作，以此来描述一项功能	类、接口、包、组件、端口和连接器
	构件图（Component Diagram）	显示构件和它们之间的依赖关系	构件、接口、提供的接口、必需的接口、类、端口、连接器、工件、组件实现、使用
	部署图（Deployment Diagram）	描述各工件（artifact）在运行节点（服务器）上的分布情况	部署、工件、部署目标、节点、设备、执行环境、通信路径、部署规范

续表

图分类	图 名 称	用 途	主要概念
行为图	配置文件图(Profile Diagram)	允许定义自定义原型、标记值和约束,作为 UML 标准的轻量级扩展机制。允许为不同的对象调整 UML 元模型平台(如 J2EE 或.NET),或者域(例如实时或业务流程建模)	配置文件、元类、构造型、扩展名、参考文件、配置文件应用程序
	用例图(Use Case Diagram)	描述系统与外部用户(参与者)协作执行的一组操作(用例),这些操作向参与者提供一些可观察的和有价值的结果	用例、参与者、主题、扩展、包含、关联
	活动图(Activity Diagram)	描述某种业务或功能包含的各活动的执行顺序和条件	活动、分区、行动、对象、控制、活动传递
	状态机图(State Machine Diagram)	通过有限状态转换对离散行为建模。可用于表示系统(对象)的行为外,还可以用来表示系统(对象)的使用协议	状态、事件、转换、动作
	顺序图(Sequence Diagram)	描述某种场景下对象之间按时间顺序进行的消息交换	生命线、执行申明、消息、复合片段、交互使用、状态不变式、销毁
	通信图(Communication Diagram)	以有顺序的消息展现多个对象在协同工作过程中互相通信的情况	对象、链接、消息
	定时图(Timing Diagram)	采用一种带数字刻度的时间轴来精确地描述消息的顺序	生命线、状态或状况时间表、销毁事件、持续约束、时间限制
	交互概览图(Interaction Overview Diagram)	以交互图(顺序图、通信图、定时图)或交互图的引用代替活动图中的活动,显示了各交互节点之间的控制流	初始节点、流最终节点、活动最终节点、决策节点、合并节点、分叉节点、连接节点、交互、交互使用、持续约束、时间限制

注意:容易混淆的是有时也将图称为模型,因为两者都包含一组模型元素的信息。这两个概念的区别是:模型描述的是信息的逻辑结构,而图是模型的特殊物理表示。

2. UML 模型及建模规则

UML 可从不同视角为系统建模,形成不同的视图。每个视图是系统完整描述中的一个抽象,代表该系统的一个特定方面;每个视图又由一组图构成,图包含了强调系统某一方面的信息。OOM 主要有 4 种**模型**:用例模型、静态模型、动态模型和实现模型。

UML 的模型图不是由 UML 语言成分简单堆砌而成,必须按特定的规则有机地组成合法的 UML 图。一个完备的 UML 模型图在语义上应一致,并且和一切与它相关的模型和谐地组合在一起。UML **建模规则**包括对以下**内容**的**描述**。

(1) 名字。任何 UML 成员都必须包含一个名字。

(2) 作用域。UML 成员所定义的内容起作用的上下文环境。某个成员在每个实例中代表一个值,还是代表这个类元的所有实例的一个共享值,由上下文决定。

(3) 可见性。UML 成员能被其他成员引用的方式。

(4) 完整性。UML 成员之间互相连接的合法性和一致性。

(5) 运行属性。UML 成员在运行时的特性。

一个完备的 UML 模型必须对以上内容给出完整的说明,这是建造系统所必需的,但在不同的视图中,对不同的交流侧重点,其表达可以是不完备的。在系统开发中,其**模型可以出现以下 3 种情况**。

(1) 被省略,即模型本身是完备的,但在图上某些属性被隐藏起来,以简化表达。

(2) 不完全,即在设计过程中某些元素可以暂时不存在。

(3) 不一致,即在设计过程中暂时不保证设计的完整性。

此**建模原则的目的**是为使开发人员在设计模型时将注意力集中在某一特定时期内对分析设计活动最重要的问题上,而暂时不过分纠缠具体细节,使模型逐步趋向完备。

3. UML 的特点及应用

1) UML 的特点

UML 的特点主要如下。

(1) 统一标准,易使用,可视化,表达力强,易于在不同背景的人员之间进行交流。

(2) 可用于任何软件开发过程,即前面所述各种软件工程模型都可使用 UML 建模。

(3) UML 内部有扩展机制,可以对一些概念进行进一步的扩展。

(4) UML 的一个最重要的特征是用于建模,而不是一种方法,只是一种建模工具。

(5) 为了模型的可视化,UML 为每一个模型元素规定了独特的图形表示符号,这些符号简洁明了,能够容纳足够的语义,并且容易绘制。

2) 使用准则

主要包括:选择使用合适的 UML 图,只对关键事物建立模型,分层次地画出模型图,模型应具有协调性,模型和模型的元素大小适中。

【案例 3-6】 保险业务系统的用例视图,如图 3-12 所示。

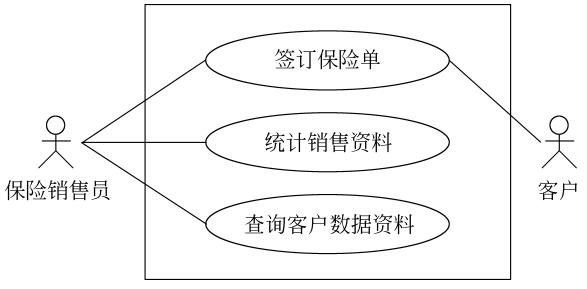


图 3-12 保险业务系统的用例视图

讨论思考

(1) OOA 的主要原则是什么?

(2) OOA 的主要任务和过程有哪些?

(3) 使用 OMT 建立对象模型的步骤有哪些?

3.4 面向对象设计

分析阶段主要模拟问题域和系统任务,而 OOD 是在分析的基础上扩充完善,建立求解域模型的过程,增加各种组成部分。OOM 对软件分析和设计不严格区分,只有分工。

3.4.1 面向对象设计的准则及任务

1. 面向对象设计的准则

由于 OOA 与 OOD 在概念、术语、描述方式上的一致性,建立一个针对具体实现的 OOD 模型,可视为按照设计的准则,对分析模型细化。**OOD 准则**包括 5 方面。

(1) 抽象。抽象是指强调实体的本质属性,忽略一些无关紧要的属性。在 OOA 阶段使用抽象仅涉及应用域的概念,在理解问题域前不考虑设计与实现。而在 OOD 阶段,使用不同层次的类实现不同层次的抽象,以满足对细节的不同需要。

(2) 信息隐蔽。在 OOM 中即为"封装性",是保证软件部件具有优良的模块性的基础;也是将对象的属性及操作(服务)结合为一个整体,尽可能屏蔽对象的内部细节,软件部件外部对内部的访问通过接口实现。类是封装良好的部件,类的定义将其说明(用户可见的外部接口)与实现(用户内部实现)分开,而对其内部的实现按照具体定义的作用域提供保护。

(3) 高内聚。指子系统内部是由一些关系密切的类构成,除了少数的"通信类"外,子系统中的类应只与该子系统中的其他类协作,从而构成具有强内聚性的子系统。

(4) 低耦合。按照抽象与封装性,使子系统之间的联系尽量少。子系统应具有良好的接口,子系统通过接口与系统的其他部分联系。

(5) 可重用。软件重用是提高开发效率和质量的重要途径。高内聚、低耦合的子系统和类,才能有效地提高所设计部件的可重用性。重用从设计阶段开始,有两方面的含义:一是尽量使用已有的类,包括开发环境提供的类库和以往开发类似系统时创建的类;二是若确实需要创建新类,则在设计新类时,应考虑将来可重复使用。

2. OOD 的基本任务

OOD 是 OOM 在软件设计阶段的应用与扩展,是将 OOA 所创建的分析模型转换为设计模型,解决"怎么做"的问题。其**主要目标**是提高开发效率、质量和可维护性。在 OOD 中为了实现系统,需要以 OOA 模型为基础,重新定义或补充一些新的类,或在原有类中补充或修改一些属性及操作。所以,具体目标是产生一个满足用户需求、可实现的 OOD 模型。

OOD 可细分为系统设计和对象设计。系统设计确定实现系统的策略和目标系统的

高层结构。对象设计确定解空间中的类、关联、接口形式及实现服务的算法。

(1) 系统设计。**主要任务**是：将分析模型中紧密相关的类划分为子系统(也称为主题)，子系统应具有良好的接口，且其中的类相互协作。标识问题本身的并发性，建立子系统之间的通信。子系统划分是系统设计的关键，将划分的子系统组织成完整的系统时，有水平层次和垂直块组织两种方式，层次结构又分为封闭式和开放式。封闭式指每层子系统仅使用其直接下层的服务，可降低各层之间相互依赖，提高易理解性和可修改性。开放式则允许各层子系统使用其下属任一层子系统提供的服务。块状组织将软件系统垂直地划分为几个相对独立、弱耦合的子系统，一个子系统(块)提供一种类型服务。

知识拓展

OOD 模型的组成

(2) 对象设计。模块、数据结构及接口等都集中地体现在对象和对象层次结构中，系统开发的全过程都与对象层次结构直接相关，是面向对象系统的基础和核心。OOD 通过对象的认定和对象层次结构的组织，确定解空间中应存在的对象和对象层次结构，并确定外部接口和主要的数据结构。

对象设计是对各类的属性和操作的详细设计，包括属性及操作的数据结构和实现算法，以及类之间的关联。此外，在 OOA 阶段，将一些与具体实现条件密切相关的对象，如与图形用户界面(GUI)、数据管理、硬件及操作系统有关的对象推迟到 OOD 阶段考虑。在进行对象设计的同时也要进行消息设计，即设计连接类与其协作者之间的消息规约。

(3) 设计优化。主要涉及提高效率的技术和建立良好的继承结构的方法。提高效率的技术包括增加冗余关联以提高访问效率，以及调整查询次序、优化算法等。建立良好的继承关系是优化设计的重要内容，通过对继承关系的调整实现。

3.4.2 系统设计的过程

在 OOM 设计软件时，OOD 模型(求解域对象模型)与 OOA 模型(问题域对象模型)类似，**其 5 个组成层次**为主题层、类与对象层、结构层、属性层和服务层。大多数系统的 OOD 模型，逻辑上都由 4 部分组成，**对应目标系统的 4 个子系统**：问题域子系统、人机交互子系统、任务管理子系统和数据管理子系统，包括有效的人机交互所必需的实际显示和输入；放置 OOA 结果并管理设计的某些类及对象、结构、属性和方法；任务定义、通信和协调、硬件分配及外部系统；对永久性数据的访问和管理。**OOD 层次模型**如图 3-13 所示。

图 3-13 OOD 层次模型

OOD 是将分析阶段获得的需求,转变成符合成本和质量要求的、抽象的系统实现方案的过程。**OOD 系统设计过程**主要按照以下 5 个**步骤**进行。

1. 系统分解及组成

系统分解有利于降低设计的难度,便于分工协作和对系统的理解与维护。通常由所提供的功能划分子系统。一般应尽量减少子系统的数量,各子系统间的接口尽可能简单明确。可相对独立地设计各个子系统。在划分和设计子系统时,应尽量减少子系统间的依赖性。

软件系统中**子系统结构**的**组成**有**两种方案**:层次组织和块状组织。

(1) 层次组织。层次结构可以分为两种模式:封闭式和开放式。封闭式的各层子系统只用其直接下层提供服务。不仅降低了各层次之间的相互依赖性,而且更容易理解和修改。开放式的各层子系统可利用下面的任何一层子系统提供的服务。优点是减少了需要在各层重新定义的服务数量,使系统更加高效紧凑。缺点是不利于信息隐蔽,对子系统的修改将影响更高层次的子系统。

(2) 块状组织。将系统分解成几个相对独立的、低耦合的子系统,每一子系统相当于一块,每块提供一种类型的服务。

设计系统的拓扑结构,可利用层次和块的各种组合,将多个子系统构成完整的软件系统。此时,典型的拓扑结构为管道型、树状、星状等。可采用与问题结构相适应的、尽量简单的拓扑结构,以减少子系统之间的交互数量。

2. 问题域子系统的设计

OOD 实际只需对分析阶段的问题域模型做补充或修改,主要是增添、合并或分解类与对象、属性及服务,调整继承关系等。利用 OOM 开发软件,可保持问题域组织框架的稳定性,从而便于追踪分析、设计和编程。基于问题域的总体框架的系统,在设计与实现过程中进行细节修改,如增加具体类及属性或服务,并不影响开发结果的稳定性。**设计问题域子系统**的**主要工作**为:调整需求、重用已有类设计、组合问题域有关的类、添加一般化类等。

(1) 调整需求。当用户需求或外部环境发生变化,分析员对问题域理解不确切或缺乏领域专家帮助,使分析模型不能完整准确地反映用户真实需求时,OOA 需要修改。

(2) 重用已有类设计。这是 OOD 的重要工作。步骤是:先选择可能被重用的类,并标明重用类中,对问题域不需要的属性和操作,增加从重用类到问题域类间的一般-特殊化的关系,最后标出应用类中因继承重用类而无须定义的属性和操作,修改应用类的结构和连接。

(3) 组合问题域有关的类。在类库中分析查找一个作为层次结构树的根类,将所有与问题域有关的类关联,建立类的层次结构。再将同一问题域的一些类整理存放在类库中。

(4) 添加一般化类。某些特殊类有时要求一组类似的服务,这时需要添加一个一般化的类,定义所有此特殊类共用的一组服务,在此类中定义其实现。

3. 任务管理子系统的设计

很多对象之间的相互依赖关系将影响不同对象的并发工作。需要确定必须同时动作的对象和相互排斥的对象，然后进一步设计任务管理子系统。任务也称为进程，是执行一系列活动的一段程序。当系统中出现较多并发行为时，需要依照各行为的协调和通信关系划分各种任务，简化并发行为的设计和编码。**任务管理主要包括**任务选择和调整，先分析任务的并发性，后设计任务管理子系统定义任务。

(1) 分析并发性。主要利用 OOM 建立的动态模型，这是分析并发性的主要依据。若两个对象彼此不存在交互，或同时接受事件，则这两个对象在本质上是并发的。

(2) **设计任务管理子系统。通常包括 6 项工作。**

① 确定事件驱动任务：如一些负责与硬件设备通信的任务。

② 辨识时钟驱动任务：以固定时间间隔激发某种事件，以执行某些处理。

③ 辨识优先及关键任务：以处理的优先级别或以某种特殊情况安排各任务。

④ 明确协调者：当有 3 个或更多任务时，可增加一个起协调作用的任务进行协调。其行为可用状态转换图进行描述。

⑤ 评审任务：为了确保满足任务的事件驱动，需要对各项任务进行评审，以时钟驱动确定优先级，或以关键任务确定任务的协调者。

⑥ 确定资源需求：由任务确定资源，可使用具体的软硬件实现某些子系统。

(3) **定义任务。主要工作**包括明确具体任务、协调工作方法和通信方式。

① 明确具体任务：对任务进行命名，并进行简要说明。

② 协调工作方法：确定各个任务协调具体工作方法，指出时间驱动或时钟驱动。

③ 定义通信方式：定义各个任务之间的通信方式，任务取/送值位置。

4. 数据管理子系统的设计

在数据管理系统中存储和检索对象的基本结构由数据管理部分提供，包括对永久性数据的访问和管理。可在某种数据存储管理系统上，建立隔离数据管理机构所关心的事项。

(1) 选取数据存储管理模式。**数据存储管理模式主要有 3 种**：文件管理、关系数据库管理和面向对象数据库管理。文件管理提供基本的文件处理能力；关系数据库管理利用多个表格管理数据；面向对象数据库管理以对自身扩充或扩充的 OOL 两种方法实现。3 种模式各有其特点和适用范围，可根据应用系统的特点具体选取使用不同的模式。如设计 ATM 系统中的任务管理子系统时，重点是选择数据存储管理模式。

(2) 设计数据管理子系统。数据管理子系统是系统存储或检索对象的基本设施，建立在某种数据存储管理系统上，并隔离数据存储管理模式(文件、关系数据库或面向对象数据库)的影响。设计此子系统主要是设计数据格式和设计相应的服务。设计数据格式的方法，应根据所用的数据存储管理模式具体确定。不同模式、属性和服务的设计方法不同。

5. 人机交互子系统的设计

人机交互设计对用户的使用和工作效率将产生重要影响。子系统之间一般有两种交互方式：客户-供应商(Client-Supplier)关系和平等伙伴(Peer-to-Peer)关系，尽量使用前者。