第3章 软件体系结构建模和UML

构建一个软件系统最困难的部分是确定构建什么。其他部分的工作不会像这部分工作一样，在出错之后会如此严重地影响随后实现的系统，并且在以后修补会如此的困难。

——Fred Brooks

软件体系架构是一系列相关的抽象模式，用于指导大型软件系统各个方面的设计。软件体系架构描述的对象是直接构成系统的抽象组件，各个组件之间的连接则明确和相对细致地描述组件之间的通信。软件体系结构研究的主要内容涉及软件体系结构的描述、软件体系结构的风格、软件体系结构的形式化方法等。软件体系结构的研究完全独立于软件工程的研究，成为计算机科学的一个新的研究方向和独立的学科分支。UML 是一种通用的标准建模语言，可以对任何具有静态结构和动态行为的系统进行建模。

本章共分两大部分，在第一部分中，3.1 节介绍软件体系结构建模概述，3.2 节介绍基于软件体系结构的开发。在第二部分中，3.3 节介绍 UML 概述，3.4 节介绍面向对象方法，3.5 节介绍 UML 2.0 中的结构建模，3.6 节介绍 UML 2.0 中的行为建模。

3.1 软件体系结构建模概述

研究软件体系结构的首要问题是如何表示软件体系结构，即如何对软件体系结构建模。

根据建模的侧重点不同，可以将软件体系结构的模型分为 5 种，即结构模型、框架模型、动态模型、过程模型和功能模型。在这 5 种模型中最常用的是结构模型和动态模型。

1. 结构模型

结构模型是一种最直观、最普遍的建模方法。这种方法，以体系结构的构件、连接件和其他概念来刻画结构，并力图通过结构来反映系统的重要语义内容，包括系统的配置、约束、隐含的假设条件、风格、性质。研究结构模型的核心是体系结构描述语言。

2. 框架模型

框架模型与结构模型类似，但它不侧重于描述结构的细节，而更侧重于描述整体的结构。框架模型主要以一些特殊的问题为目标建立只针对和适应该问题的结构。

3. 动态模型

动态模型是对结构或框架模型的补充，其研究系统的“大颗粒”的行为性质。例如，描述系统的重新配置或演化。动态可能指系统总体结构的配置、建立或拆除通信通道或计算的过程。这类系统常是激励型的。

4. 过程模型

过程模型研究构造系统的步骤和过程，因而结构是遵循某些过程脚本的结果。

5. 功能模型

功能模型认为，体系结构是由一组功能构件按层次组成下层向上层提供服务，它可以看作是一种特殊的框架模型。这 5 种模型各有所长，也许将 5 种模型有机地统一在一起形成一个完整的模型来刻画软件体系结构更合适。例如，Kruchten 在 1995 年提出了“4＋1”视图模型。“4＋1”视图模型从 5 个不同的视图，包括逻辑视图、过程视图、物理视图、开发视图和场景视图来描述软件体系结构（具体内容参见 4.2）。每一个视图只关心系统的一个侧面，5 个视图结合在一起才能够反映系统的软件体系结构的全部内容。

3.2　基于软件体系结构的开发

良好的体系结构，可以为软件开发和维护带来好处，主要体现在以下方面：

（1）经过 40 多年的软件开发实践，今天很少有待开发的软件系统和以前的系统没有任何相似之处，识别相似系统的通用结构模式，有助于理解系统之间的高层联系，使得新系统可以作为以前系统的变种来构造。

（2）合适的体系结构是软件系统成功的关键，而不合适的体系结构往往导致灾难性的后果。

（3）对软件体系结构的准确理解，可以使开发人员在不同的设计方案中做出理性的选择。

（4）体系结构对于分析和描述复杂系统的高层属性，通常是十分必要的。

（5）各种体系结构风格的提炼、描述和普遍采用，可以丰富设计人员的“词汇”，便于在系统设计中互相交流。

（6）目前，相当大的维护工作量花费在程序理解方面，如果在软件开发文档中清楚地记录了系统的体系结构，不仅可以显著地节省理解软件的工作量，而且便于在软件维护全过程中保持系统的总体结构和特性不变。

由此可见，体系结构在整个软件生命周期中扮演着重要的角色。

体系结构的软件开发过程包括几个主要活动：

（1）通过对特定领域应用软件进行分析，提炼其中的稳定需求和易变需求，建立可复用的领域模型。根据用户需求和领域模型，产生应用系统的需求规格说明。

（2）在领域模型的基础上提炼面向特定领域的软件体系结构。高层设计的任务是根据需求规格说明进行体系结构设计，通过复用体系结构库中存放的面向特定领域的体系结构，或创造适合该应用环境的体系结构，并加以提炼入库，以备将来复用。在体系结构的框架指导下，把系统功能分解到相应的构件和连接件。构件和连接件往往不是简单的模块或对象，它们甚至可能包含复杂的结构，因此，可能需要多层次的系结构设计，直至构件和连接件可以被设计模式或单个的对象处理为止。

（3）低层设计主要解决具体构件和连接件的设计问题，通过复用设计件库中存放的设计模式、对象和其他类型的可复用设计件，或根据情况设计新的构件，并提炼入库。低层设计的结果可以直接编程实现。

体系结构从宏观上描述系统的总体结构，设计模式和对象从微观上解决实际的设计问题，分别对应于高层设计和低层设计阶段的分析需求规格说明书，获取系统的功能性需求和非功能性需求，需要确定系统的边界，识别出所有的参与者和用例，功能性需求和非功能性需求使用用例和场景进行描述。并且，需要将具有相同或相似的属性和方法的一类对象抽象为一个

类，比较各个类的属性和方法，确定不同类之间的关系。根据类之间的关系生成类图，将密切相关的类划分为一组，形成构件，然后根据构件端口，确定构件之间的关联关系，根据功能性需求和非功能性需求确定系统应该采用的体系结构风格。在体系结构设计方案中，若存在相同或相似的解决方案将直接进行复用，或经过简单的修改之后再复用，如果没有，则需要进行重新设计。体系结构设计师、系统分析人员和客户以及相关的技术实现人员对体系结构设计结果进行评审，确定所提出的解决方案是否能够满足用户的要求，是否能够提高资源的复用效率。

3.3 UML 概述

3.3.1 UML 的发展历程

公认的面向对象建模语言出现于 20 世纪 70 年代中期，从 1989 年到 1994 年，其数量从不到 10 种增加到了 50 多种。在众多的建模语言中，语言的创造者努力推崇自己的产品，并在实践中不断完善。但是 OO 方法(Object-Oriented Method，面向对象方法)的用户并不了解不同建模语言的优/缺点及相互之间的差异，因而很难根据应用特点选择合适的建模语言，于是爆发了一场方法大战。在 20 世纪 90 年代中期，一批新方法出现了，其中最引人注目的是 Booch 1993、OOSE(Object-Oriented Software Engineering)和 OMT-2(Object Modeling Techaology)等。

Grady Booch 是面向对象方法最早的倡导者之一，他提出了面向对象软件工程的概念。1991 年，他将以前面向 Ada 的工作扩展到整个面向对象设计领域。James Rumbaugh 等人提出了面向对象的建模技术方法，采用了面向对象的概念，并引入各种独立于语言的表示符号。这种方法，用对象模型、动态模型、功能模型和用例模型共同完成对整个系统的建模，所定义的概念和符号可用于软件开发的分析、设计和实现的全过程，软件开发人员不必在开发过程的不同阶段进行概念和符号的转换。

Jacobson 于 1994 年提出了 OOSE 方法，其最大的特点是面向用例(Use Case)，并在用例的描述中引入了外部角色的概念。OOSE 比较适合支持商业工程和需求分析。Coad/Yourdon 方法，即著名的 OOA/OOD(Object-Oriented Analysis Object-Oriented-Design)，它是最早的面向对象的分析和设计方法之一。该方法简单、易学，适合面向对象技术的初学者使用，但由于该方法在处理能力方面有限，目前已很少使用。

1994 年 10 月，Grady Booch 和 Jim Rumbaugh 开始致力于这一工作。他们首先将 Booch 93 和 OMT-2 统一起来，并于 1995 年 10 月发布了第一个公开版本，称为统一方法 UM 0.8 (Unitied Method)。1995 年秋，OOSE 的创始人 Ivar Jacobson 加入到这一工作。经过 Booch、Rumbaugh 和 Jacobson 的共同努力，于 1996 年 6 月和 10 月分别发布了两个新的版本，即 UML 0.9 和 UML 0.91，并将 UM 重新命名为 UML(Unified Modeling Language)。1996 年，一些机构将 UML 作为其商业策略已日趋明显。UML 的开发者得到了来自公众的正面反应，并倡议成立了 UML 成员协会，以完善、加强、促进 UML 的定义工作。这一机构对 UML 1.0 及 UML 1.1 的定义和发布起到了重要的促进作用，2001 年，推出了 UML 2.0 新的业界标准。

UML 是一种定义良好、易于表达、功能强大且普遍适用的建模语言，它融入了软件工程

领域的新思想、新方法和新技术。它的作用域，不仅支持面向对象的分析与设计，而且支持从需求分析开始的软件开发的全过程。

3.3.2 UML的特点和用途

标准建模语言UML的主要特点可以归结为下面几点。

(1) 统一标准：UML统一了Booch、OMT和OOSE等方法中的基本概念，已成为OMG的正式标准，提供了标准的面向对象的模型元素的定义和表示。

(2) 面向对象的特性：UML还吸取了面向对象技术领域中其他流派的长处，其中也包括非面向对象方法的影响。UML符号表示，考虑了各种方法的图形表示，删掉了大量易引起混乱的、多余的和极少使用的符号，并添加了一些新符号。因此，在UML中汇入了面向对象领域中很多人的思想。这些思想，并不是UML的开发者们发明的，而是开发者们依据最优秀的OO方法和丰富的计算机科学实践经验综合提炼而成的。

(3) UML在演变过程中提出了一些新的概念：在UML标准中新加了模板(Stereotypes)、职责(Responsibilities)、扩展机制(Extensibility Mechanisms)、线程(Threads)、过程(Processes)、分布式(Distribution)、并发(Concurrency)、模式(Patterns)、合作(Collaborations)、活动图(Activity Diagram)等新概念，并清晰地区分类型(Type)、类(Class)、实例(Instance)、细化(Refinement)、接口(Interfaces)和组件(Components)等概念。

(4) 立于过程：UML作为建模语言，不依赖特定的程序设计，独立于开发过程。

(5) UML对系统的逻辑模型和实现模型都能清晰的表示，可以用于复杂软件系统的建模。

因此可以认为，UML是一种先进、实用的标准建模语言，但其中的某些概念尚待实践来验证，UML也必然存在一个进化过程。

3.3.3 UML 2.0的建模机制

UML 2.0的新特性如下。

(1) 语言定义的精确度提高：这是支持自动化高标准需要的结果。自动化意味着模型将消除不明确和不精密，可以保证计算机程序能转换并熟练地操纵模型。

(2) 改良的语言组织：该特性由模块化组成，模块化的优点在于它不仅使语言更加容易地被新用户所采用，而且促进了工具之间的相互作用。

(3) 重点改进大规模的软件系统模型性：一些流行的应用软件表现出将现有的独立应用程序集中到更加复杂的系统中去。

(4) 对特定领域改进的支持：使用UML的实践经验，证明了其所为的扩展机制的价值。这些机制使基础语言更加简洁，更加准确、精练。

(5) 全面的合并，合理化、清晰化各种不同的模型概念：该特性导致一种单一化、更加统一化的语言产生，使UML 2.0的图形符号也进行了一些调整。

3.4 面向对象方法

从事软件开发的工程师们常常有这样的体会：在软件开发过程中，使用者会不断地提出各种更改要求，即使在软件投入使用后，也常常需要对其做出修改，在用结构化开发的程序中，

这种修改往往是很难的，而且会因为计划或考虑不周，不仅旧错误没有得到彻底修改，还引入了新的错误；另外，在过去的程序开发中，代码的重用率很低，使得程序员的编程效率并不高，为提高软件系统的稳定性、可修改性和可重用性，人们在实践中逐渐创造出软件工程的一种新途径——面向对象方法学。目前，面向对象开发方法的研究已日趋成熟，在国际上已有不少面向对象产品出现。面向对象开发方法有 Coad 方法、Booch 方法和 OMT 方法等。UML 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，最终统一为大众接受的标准建模语言。UML 是一种定义良好、易于表达、功能强大且普遍适用的建模语言，它融入了软件工程领域的新思想、新方法和新技术。

面向对象方法是当今主流的软件开发方法，其基础在于将客观世界中的应用问题看成是由实体及其相互关系组成的，将与某一应用问题有关的实体抽象为问题空间的对象。面向对象开发以系统化的方法学进行指导，其中包含了各种概念、技术和过程。面向对象方法学的出发点和基本原则是尽可能模拟人类习惯的思维方式，使开发软件的方法与过程尽可能接近人类认识世界、解决问题的方法与过程。由于客观世界的问题都是由客观世界中的实体及实体相互间的关系构成的，因此我们把客观世界中的实体抽象为对象(Object)。持面向对象观点的程序员认为计算机程序的结构应该与所要解决的问题保持一致，而不是与某种分析或开发方法保持一致。所以，“面向对象”是一种认识客观世界的世界观，是从结构组织角度模拟客观世界的一种方法。人们在认识和了解客观现实世界时，通常运用一些构造法则。

- ◇ 区分对象及其属性：例如区分具体的一辆汽车和它的重量、最大速度。
- ◇ 区分整体对象及其组成部分：例如区分台式计算机的组成(主机、显示器等)。
- ◇ 不同对象类的形成以及区分：例如所有类型的计算机(大、中、小型计算机、服务器、工作站和普通微型计算机等)。

可以看出，面向对象所带来的好处是程序的稳定性与可修改性(由于把客观世界分解成一个一个的对象，并且把数据和操作都封装在对象的内部)、可复用性(通过面向对象技术，我们不仅可以复用代码，而且可以复用需求分析、设计、用户界面等)。

3.4.1 面向对象方法中的基本概念

1. 对象

对象(Object)指的是一个独立的、异步的、并发的实体，它能“知道一些事情”(即存储数据)，“做一些工作”(即封装服务)，并“与其他对象协同工作”(通过交换消息)，从而完成系统的所有功能。因为所要解决的问题具有特殊性，所以对象是不固定的。对象是现实世界中的个体或事物的抽象表示，是其属性和相关操作的封装。属性表示对象的性质，属性值规定了对象所有可能的状态。对象的操作是指该对象可以展现的外部服务。对象是事物的本质，是指不会随周围环境改变而变化的相对固定的最小的集合。对象实现了数据和操作的结合，使数据和操作封装于对象的统一体中。

例如，在计算机屏幕上画多边形，每个多边形是一个用有序顶点的集所定义的对象。这些顶点的次序决定了它们的连接方式，顶点集定义了一个多边形对象的状态，包括它的形状和它在屏幕上的位置，在多边形上的操作包括 draw(屏幕显示)、move(移动)、contains(检查某点是否在多边形内)。

2. 类

类(Class)的定义包括一组数据属性和在数据上的一组合法的操作。在一个类中，每个对

象都是类的实例(Instance),类的对象具有相同的方法集,有相同或相似性质的对象的抽象就是类。因此,对象的抽象是类,类的具体化就是对象,也可以说,类的实例是对象。类具有属性,它是对象的状态的抽象,用数据结构来描述类的属性。类具有操作,它是对象的行为的抽象,用操作名和实现该操作的方法来描述。

3. 继承性

广义地说,继承(Inheritance)是指能够直接获得已有的性质和特性,而不必重复定义它们。在面向对象的软件技术中,继承是子类自动地共享基类中定义的数据和方法的机制。一个类直接继承其父类的全部描述(数据和操作)。继承具有传递性,继承性使得相似的对象可以共享程序代码和数据结构,从而大大减少了程序中的冗余信息,使得对软件的修改变得比过去容易多了。继承性使得用户在开发新的应用系统时不必完全从零开始,可以继承原有的相似系统的功能或者从类库中选取需要的类,再派生出新的类以实现所需要的功能,所以,继承的机制主要是支持程序的重用和保持接口的一致性。父类是高层次的类,表达共性,子类是低层次的类,表达个性。子类通过继承机制获得父类的属性和操作。例如,电视机、电话、计算机等都是电子产品,它们具有电子产品的公共特性,当定义电视机类 Video、电话类 Telephone 和计算机类 Computer 时,为避免它们公共特性的重复编码,可将这些电子产品的公共特性部分定义为电子产品类,将 Video、Telephone 和 Computer 定义为它的子类。子类继承了父类的所有属性和操作,而且子类还可以扩充定义自己的属性和操作,如电子产品类具有型号、价格、颜色等属性,Computer 则继承了这些属性,并扩充自己的属性,包括显示类型、内存大小等属性。又如汽车是轿车、吉普车及卡车的父类,轿车、吉普车及卡车是汽车的子类。父类和子类是相对的,父类之上可有另一个父类,而成为其子类。

4. 多态性

在面向对象的软件技术中,多态性(Polymorphism)是指子类对象可以像父类对象那样使用,同样的消息既可以发送给父类对象也可以发送给子类对象。也就是说,在类等级的不同层次中可以共享(公用)一个行为(方法)的名字,然而不同层次中的每个类却各自按自己的需要来实现这个行为。当对象接收到发送给它的消息时,根据该对象所属的类动态选用在该类中定义的实现算法。

5. 重载

在面向对象软件技术中有两种重载(Overloading),其中,函数重载是指在同一作用域内的若干个参数特征不同的函数可以使用相同的函数名字;运算符重载是指同一个运算符可以施加于不同类型的操作数上面。当然,当参数特征不同或被操作数的类型不同时,实现函数的算法或运算符的语义是不同的。重载进一步提高了面向对象系统的灵活性和可读性。

6. 消息

在面向对象领域,两个对象的交互是通过消息(Message)的发送和接收来完成的。消息分为简单消息、同步消息和异步消息 3 种类型。

(1) 简单消息:只是表示控制如何从一个对象发送给另一个对象,并不包含控制的细节。

(2) 同步消息:同步意味着阻塞和等待,如果对象 A 给对象 B 发送一个消息,对象 A 会等待对象 B 执行完这个消息,接着再进行自身的工作。

(3) 异步消息:异步意味着非阻塞,如果对象 A 给对象 B 发送一个消息,对象 A 不必等待对象 B 执行完这个消息,就可以接着进行自身的工作。

消息传递是对象与其外界时间相互关联的唯一途径。对象之间进行通信的结构称为消

息。在对象的操作中，当一个消息发送给某个对象时，消息包含接收对象去执行某种操作的信息。发送一条消息至少要包括说明接收消息的对象名、发送给该对象的消息名(即对象名、方法名)。

7. 聚集

在客观世界中有若干类，聚集(Aggregation)表示类之间的关系是整体与部分的关系。通常有两种主要的结构关系，即一般-具体结构关系、整体-部分结构关系。

(1) 一般-具体结构：称为分类结构，也可以说是“或”关系，或者是 is-a 关系。

(2) 整体-部分结构：称为组装结构，它们之间的关系是一种“与”关系，或者是 has-a 关系。

面向对象开发方法的核心是利用面向对象的概念和方法对软件进行需求分析和设计，建立面向对象的软件分析和设计模型。

3.4.2 面向对象方法的优势

相对于传统的结构化方法和面向数据的方法，面向对象方法拥有以下优势。

1. 支持软件的重用性

重用性是指同一事物不经过修改或稍加修改就可以多次重复使用的性质。软件重用是软件工程追求的目标之一。在面向对象方法被广泛应用以前，基于结构化方法的软件复用几乎没有取得有意义的进展。在面向对象方法中情形发生了变化，在源代码级复用方面，面向对象方法通过继承和接口等机制，使得复用者不需要直接修改被复用的类；在设计级复用方面，迅速发展的设计模式技术在软件业大显身手，现在全球范围内实施的软件项目大多离不开面向对象的重用技术的支持，例如 Java 库、J2EE 框架、.NET 的 Framework 4.0 框架。

2. 提高软件的可维护性和安全性

面向对象方法通过对属性和操作的封装实现了软件工程倡导的信息隐藏的原则。在面向对象的软件设计中，每个类拥有完成其操作所必需的数据，这些数据通过访问权限控制关键字 private 隐藏于类的内部，或者通过 protected 关键字隐藏于类及子类的内部，外界对类的内部数据的访问或修改只能通过该类对外公开的接口函数来实现，这样安全性就有了保障。相对于传统的结构化方法，面向对象方法更容易造就高质量的软件结构。

为了发挥面向对象开发方法所具备的优势，必须采用面向对象的思维方式来设计面向对象软件，避免采用结构化思维方式和工具来开发面向对象软件。UML 就是为此目标出现的，UML 通过提供多种视图模型，使开发人员能够采用面向对象方法对软件进行全面的分析和设计，从而提高软件开发的效率和质量。

3.5 UML 2.0 中的结构建模

结构图用于显示建模系统的静态结构，关注系统的元件，而无须考虑时间。在系统内，静态结构通过显示类型和它们的实例进行传播。除了显示系统类型和它们的实例之外，结构图至少显示了这些元素间的一些关系，如果有可能，甚至会显示它们的内部结构。

贯穿整个软件生命周期，结构图对于各团队成员都是有用的。一般而言，这些图支持设计验证，以及个体与团队间的设计交流。举例来说，业务分析师可以使用类或对象图为当前的资

产和资源建模，例如分类账、产品或地理层次。架构设计师可以使用组件和部署图来测试/确认他们的设计是否充分。开发者可以使用类图来设计并为系统的代码(或即将成为代码的)类写文档。UML 2.0 中的结构建模包括类图、包图、对象图、构件图、组合结构图、部署图。

3.5.1　类图

类图是 UML 中最基本也是最重要的一种视图，它用来刻画软件中类等元素的静态结构和关系。在大多数 UML 模型中，这些类型包括类、接口、数据类型、组件。

1. 类

类(Class)是来描述具有相同特征、约束和语义的一类对象，这些对象具有共同的属性和操作。类图中的一个类可以简单地只给出类名，也可以具体列出该类拥有的成员变量和方法，甚至可以更详细地描述可见性、方法参数、变量类型等信息。类的 UML 表示是一个长方形被垂直地分为 3 个区域。顶部区域显示类的名字，中间区域列出类的属性，底部区域列出类的操作。当在一个类图上画一个类元素时，用户必须要有顶端的区域，下面的两个区域是可选择的(当图描述仅仅用于显示分类器间关系的高层细节时，下面的两个区域是不必要的)。图 3-1 中显示了一个航线班机如何作为 UML 类建模。正如我们所见到的其名字是 Flight，我们可以在中间区域看到 Flight 类的 3 个属性，即 flightNumber、departureTime 和 flightDuration。在底部区域中我们可以看到 Flight 类有两个操作，即 delayFlight 和 getArrivalTime。

首先看矩形，它代表一个类。该类图分为 3 个层，第一层显示类的名称，如果是抽象类要用斜体显示。第二层是类的特性，通常是字段和属性。第三层是类的操作，通常是方法和行为。

它们从上到下分为 3 个部分，分别是类名、属性和操作。

类名是必须有的，类如果有属性，则每一个属性都必须有一个名字，另外还可以有其他的描述信息，如可见性、数据类型、默认值等。类如果有操作，则每一个操作也都有一个名字，其他可选信息包括可见性、参数的名字、参数类型、参数默认值和操作的返回值的类型等。

2. 抽象类

抽象类(Abstract Class)是指一个类只提供操作名，而不对其进行实现。对这些操作的实现可以由其子类进行，并且不同的子类可以对同一操作具有不同的实现。抽象类和类的符号区别在于抽象类的名称用斜体字符表示。图 3-2 所示为 BankAccount 类的类图。

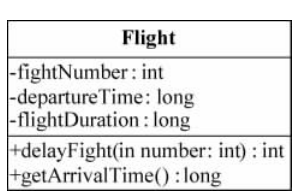

图 3-1　Flight 类的类图

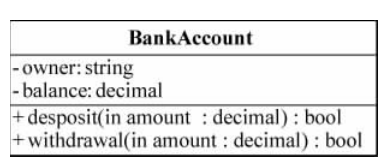

图 3-2　BankAccount 类的类图

3. 接口

“飞翔”矩形框表示一个接口图，它与类图的区别主要是顶端有接口(Interface)的显示，第一行是接口名称，第二行是接口方法。接口还有另一种表示方法，俗称棒棒糖表示法，就是唐老鸭类实现了“讲人话”的接口，如图 3-3 所示。

```
interface Ifly
{
  void Fly();
}
interface Ilanguage
{
  void Speak();
}
```

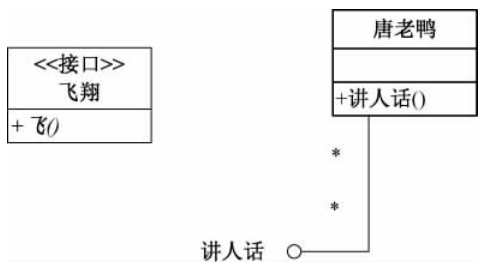

图 3-3 接口的描述

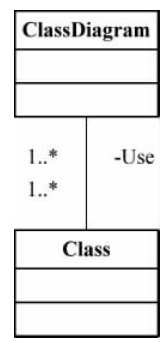

图 3-4 关联关系的描述

4. 关联关系

关联关系(Association)描述了类结构之间的关系,具有方向、名字、角色和多重性等信息,如图 3-4 所示。当关联是双向的时,可以用无向连线表示。一般的关联关系语义较弱,也有两种语义较强,分别是聚合与组合。

5. 依赖关系

两个类之间存在依赖关系(Dependency),表明一个类使用或需要知道另一个类中包含的信息。依赖关系有多种表现形式,例如绑定(bind)、友元(friend)等。模板类 Stack<T>定义了与栈相关的操作; IntStack 将参数 T 与实际类型 int 绑定,使得所有操作都针对 int 类型的数据。依赖关系使用虚线箭头表示("动物"、"氧气"与"水"之间)。动物有几大特征,例如有新陈代谢,能繁殖。而动物要有生命,需要有氧气、水以及食物等。也就是说,动物的生存依赖于氧气和水。它们之间是依赖关系,用虚线箭头来表示,如图 3-5 所示。

```
abstract class Animal
{
  public bolism(Oxygen oxygen,Water water)
  {
  }
}
```

6. 聚合关系

聚合关系(Aggregation)表明两个类的实例之间存在一种拥有或属于关系,可以看作一种较弱的整体-部分关系。在一个聚合关系中,子类实例可以比父类存在更长的时间。为了表现一个聚合关系,试画一条从父类到部分类的实线,并在父类的关联末端画一个未填充菱形。对于"大雁"和"雁群"这两个类,大雁是群居动物,每只大雁都属于一个雁群,一个雁群可以有多只大雁,所以它们之间满足聚合(Aggregation)关系。聚合表示一种弱的"拥有"关系,体现的

是 A 对象可以包含 B 对象，但 B 对象不是 A 对象的一部分。聚合关系用空心的菱形＋实线箭头表示，如图 3-6 所示。

```
class WideGooseAggregate
{
  private WideGoose[ ] arrayWideGoose;
  //在雁群 WideGooseAggregate 类中，有大雁数组对象 arrayWideGoose
}
```

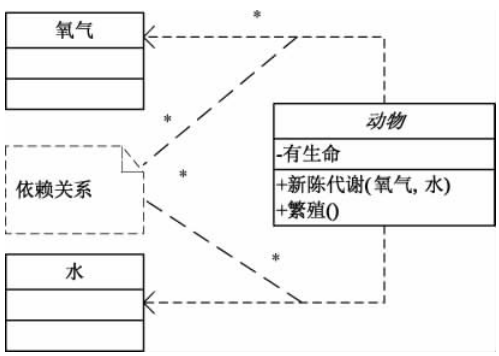

图 3-5　依赖关系的描述

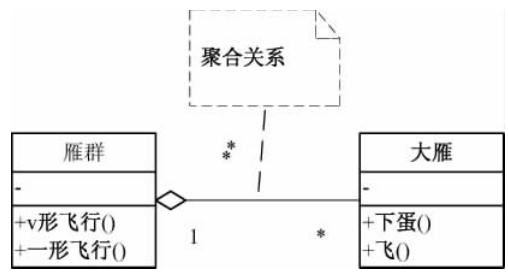

图 3-6　聚合关系的描述

7. 合成关系

合成(Composition)是一种强的“拥有”关系，体现了严格的部分和整体的关系，部分和整体的生命周期一样。合成关系用实心的菱形＋实线箭头来表示。另外，合成关系的连线两端还有一个数字“1”和数字“2”，它们被称为基数，表明这一端的类可以有几个实例。对于“鸟”和“翅膀”这两个类，鸟和翅膀类似整体和部分的关系，并且翅膀和鸟的生命周期是相同的，在这里“鸟”类和其“翅膀”类就是合成关系，如图 3-7 所示。很显然，一个鸟应该有两个翅膀。如果一个类可能有无数个实例，则用 *n* 来表示。关联关系、聚合关系也是可以有基数的。

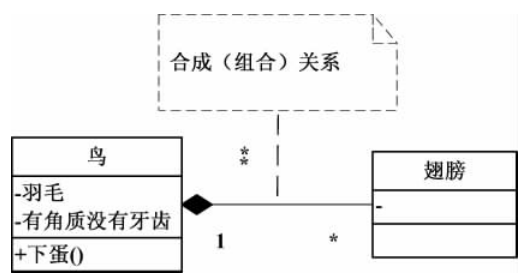

图 3-7　合成关系的描述

```
class Bird {
  private Wing wing;
  public Bird()
  {
    wing = new Wing();
    //在鸟 Bird 类中，初始化时，实例化翅膀 Wing 它们之间同时生成
  }
}
```

聚合和组合的区别在于：聚合关系是 has-a 关系，组合关系是 contains-a 关系；聚合关系表示整体与部分的关系比较弱，而组合关系表示的比较强；聚合关系中代表部分事物的对象与代表聚合事物的对象的生命周期无关，删除了聚合对象不一定删除了代表部分事物的对象，

而在组合关系中一旦删除了组合对象，同时也就删除了代表部分事物的对象。

8. 泛化关系

泛化关系(Generalization)在面向对象中一般称为继承关系，存在于父类与子类、父接口与子接口之间，如图 3-8 所示。

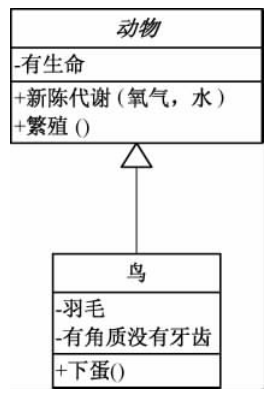

图 3-8 泛化关系的描述

3.5.2 对象图

对象是类的实例，对象图可以看作类图的实例，对象之间的连接(Link)是类之间关联关系的实例。对象图和类图的不同在于，对象图显示类的多个对象实例而不是真实的类。对象图显示某时刻对象和对象之间的关系，是类图的变化，由于对象存在生命周期，因此对象图只能在系统的某一时间段存在。

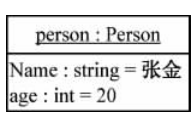

图 3-9 对象的表示

一个对象图可以看成一个类图的实例(Example)，对象图表示的是类的对象实例而不是真实的类。

对象图中并无新的表示法(除了对象名下要加下划线以外)，与类图中的表示法一样，读者可以认为，只有对象而无类的类图就是一个“对象图”，如图 3-9 所示。

在对象图中，对象名可以有以下 3 种表示形式：

(1) 对象名:类名

(2) :类名

(3) 对象名

实际上，对象图几乎很少被用到，其使用远没有类图广泛。

3.5.3 构件图

构件图用于静态建模，是表示构件类型的组织以及各种构件之间依赖关系的图。构件图通过对构件间依赖关系的描述来估计修改系统构件可能给系统带来的影响。由于基于构件的软件开发日益普及和应用，UML 对构件图进行了较大的改进。构件的根本特征在于它的封装性和可复用性，其内部结构被隐藏起来，只能通过接口向外部提供服务或请求外部的服务。

构件(Component)是系统中遵从一组接口且提供其实现的物理的、可替换的部分。构件

能够完成独立功能，它是软件系统的组成部分。在功能划分的软件系统中，软件被分成一个个模块。随着面向对象技术的引用，软件系统被分成若干个子系统、构件。每个构件能够实现一定的功能，为其他构件提供使用接口，方便软件的复用。

在此绘制地铁售票信息系统的投币构件图进行说明，如图 3-10 所示。

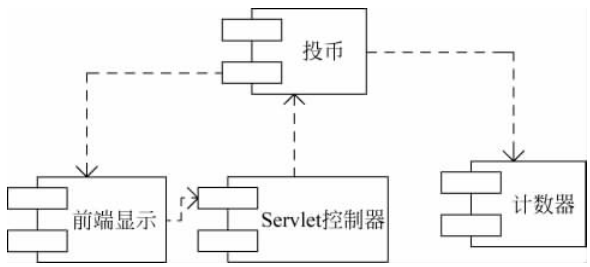

图 3-10　构件图

3.5.4　部署图

部署图(Component Diagram)描述的是系统运行时的结构，展示了硬件的配置及其软件如何部署到网络结构中。一个系统模型只有一个部署图，部署图通常用来帮助用户理解分布式系统。部署图用于静态建模，是表示运行时过程结点结构、描述软件与硬件是如何映射的、描述构件实例及其对象结构的图，如图 3-11 所示。

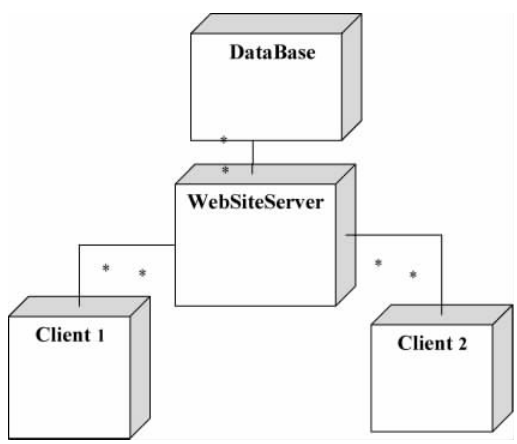

图 3-11　部署图

3.6　UML 2.0 中的行为建模

行为建模被称为动态建模，它主要用来刻画系统中的动态行为、过程和步骤。UML 行为建模中提供的视图可以从不同侧面来描述软件系统的动态过程，例如，业务或算法过程与步骤、多个对象为完成一个场景所进行的交互、消息传递的过程、一个对象在生命周期中根据接收到的不同事件进行响应的过程等。

3.6.1 用例图

1. 用例图的定义

用例图是被称为参与者的外部用户所能观察到的系统功能的模型图。用例图列出了系统中的用例和系统外的参与者，并显示哪个参与者参与了哪个用例的执行(或称为发起了哪个用例)。用例图多用于静态建模阶段(主要是业务建模和需求建模)。

2. 参与者

参与者(Actor)指在系统外部与系统直接交互的人或事物(如另一个计算机系统或一些可运行的进程)。我们需要注意的是：

- ◇ 参与者是角色(Role)而不是具体的人，它代表了参与者在与系统“打交道”的过程中所扮演的角色。所以在系统的实际运作中，一个实际用户可能对应系统的多个参与者。不同的用户也可以只对应一个参与者，从而代表同一个参与者的不同实例。
- ◇ 参与者作为外部用户(而不是内部)与系统发生交互作用，是它的主要特征。

3. 用例

用例(Use Case)是系统外部可见的一个系统功能单元。系统的功能由系统单元所提供，并通过一系列系统单元与一个或多个参与者之间交换的消息所表达。

1) 参与者与用例之间的关系

关联表示参与者与用例之间的交互、通信途径。关联有时候也用带箭头的实线来表示，这样的表示能够显式地表明发起用例的是参与者。

2) 用例之间的关系

(1) 包含：箭头指向的用例为被包含的用例，称为包含用例；箭头出发的用例为基用例。包含用例是必选的，如果缺少包含用例，基用例就不完整；包含用例必须被执行，不需要满足某种条件，其执行并不会改变基用例的行为。

(2) 扩展：箭头指向的用例为被扩展的用例，称为扩展用例；箭头出发的用例为基用例。扩展用例是可选的，如果缺少扩展用例，不会影响基用例的完整性，扩展用例在一定条件下才会执行，并且其执行会改变基用例的行为。

3) 参与者之间的关系

泛化关系指发出箭头的事物 is-a 箭头指向的事物。泛化关系是一般和特殊的关系，发出箭头的一方代表特殊的一方，箭头指向的一方代表一般的一方，特殊一方继承了一般方的特性并增加了新的特性。

4. 实例

1) 参与者之间的泛化关系

考虑如图 3-12 所示的关系：

在参与者之间不存在泛化关系的情况下，各个参与者参与用例的情况分别是：经理参与用例管理人事和批准预算；安全主管参与用例批准安全证书；保安参与用例监视周边。由于安全主管与经理、安全主管与保安之间存在泛化关系，意味着安全主管可以担任经理和保安的角色，即能够参与经理和保安参与的用例。这样，安全主管就可以参与全部 4 个用例。但经理和保安不能担任安全主管的角色，也就不能参与用例批准安全证书。

2) 用例之间的扩展和包含关系

用例的上下文是：短途旅行，但汽车的油不足以应付全部路程。那么为汽车加油的动作

在旅行的每个场景(事件流)中都会出现,不加油就不能完成旅行。吃饭则可以由司机决定是否进行,不吃饭不会影响旅行的完成,如图 3-13 所示。

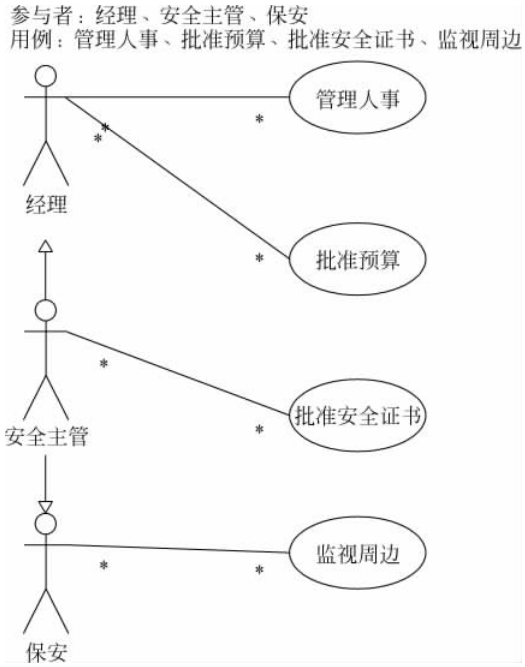

图 3-12　用例图

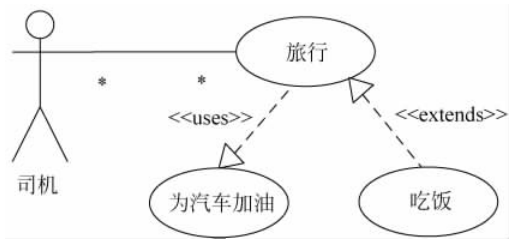

图 3-13　扩展功能的用例图

3.6.2　顺序图

顺序图描述对象之间的动态交互关系,着重表现对象间消息传递的时间顺序。顺序图有两个坐标轴,其纵坐标轴表示时间,横坐标轴表示不同的对象。顺序图中的对象用一个矩形框表示,框内标有对象名(对象名的表示格式与对象图中相同)。从表示对象的矩形框向下的垂直虚线是对象的“生命线”,用于表示在某段时间内该对象是存在的。

对象间的通信用对象生命线之间的水平消息线来表示,消息箭头的形状表明消息的类型(同步、异步或简单)。当收到消息时,接收对象立即开始执行活动,即对象被激活了。激活用对象生命线上的细长矩形框表示。消息通常用消息名和参数表来表示。消息还可以带有条件表达式,用于表示分支或决定是否发送消息。如果用条件表达式表示分支,则会有若干个互斥的箭头,也就是说,在某一时刻仅可以发送分支中的一个消息。一个顺序图显示了一系列对象和这些对象之间发送和接收的消息。

图书管理系统中图书入库的顺序图如图 3-14 所示。对于顺序图,往往在文字表述上会出现“当……时……”、“首先”、“然后”、“接着”、“……发出……消息”,“……响应……消息”等词汇。例如图 3-14 所示的顺序图可用文字表达为：当管理人员把新书入库时,首先要登录(输入用户名和口令),经系统的“注册表单”验证,若正确无误,则可继续下一步交互,否则拒绝该管理人员进入系统。若登录正确,管理人员可发出查询请求消息,系统的“图书入库表单”对象响应请求。若管理人员发出增加或删除库存图书的请求,“库存图书”对象将响应该消息,找出数据库中的相关数据并执行相应的操作。此后,管理人员应按下提交键确认请求,“图书入库

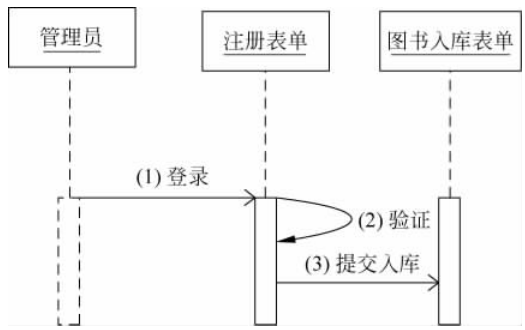

图 3-14 图书入库的顺序图

表单”接口对象应该响应该请求，并发出存储消息，再由“库存图书”对象响应存储消息，进行数据库存储操作。如果管理人员结束图书入库，发出退出系统的请求，则系统的“注册表单”接口对象响应请求，关闭系统。

3.6.3 通信图

UML 2.0 的通信图是由 UML 1.x 中的协作图发展而来的。与顺序图不同，通信图主要关注参与交互的对象通过连接组成的结构。通信图的对象没有生命线，其消息以及方向都附属于对象间的连接，并通过编号表示消息的顺序。Actor 发送 Print 消息给 Computer，Computer 发送 Print 消息给 PrintServer，如果打印机空闲，PrintServer 发送 Print 消息给 Printer，如图 3-15 所示。顺序图着重描述对象之间消息交换的时间顺序，通信图主要强调接收和发送消息的对象之间的结构组织。它们从不同角度表达了系统中的交互，它们之间是可以相互转换的。

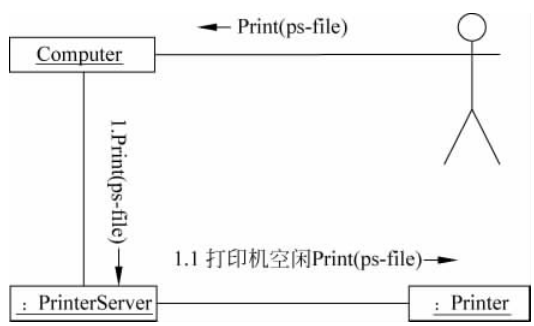

图 3-15 通信图

3.6.4 交互概览图

交互概览图通过类似于活动图方式，描述交互之间的流程，给出交互控制流的概览。在交互概览图中，结点不像活动图中那样是动作，而是一个交互图或是对交互图的引用。交互概览图有以下两种形式：

(1) 以活动图为主线，对活动图中某些重要的活动结点进行细化，即用一些小的顺序图对重要的活动结点进行细化，描述活动结点内的对象之间的交互。

(2) 以顺序图为主线，用活动图细化顺序图中的某些重要对象，即用活动图描述重要对象

的活动细节。

3.6.5　时序图

时序图(Sequence Diagram)是显示对象之间交互的图,这些对象是按时间顺序排列的,如图 3-16 所示。顺序图中显示的是参与交互的对象及其对象之间消息交互的顺序。时序图中包括的建模元素主要有对象(Actor)、生命线(Lifeline)、控制焦点(Focus of Control)、消息(Message),等等。时序图最常应用到实时或嵌入式系统的开发中,但它并不局限于此。被建模的系统类型对交互的准确时间进行建模都是非常必要的。在时序图中,每个消息都有与其有关联的信息,准确描述了何时发送消息,消息的接收对象会花多长时间收到该消息,以及消息的接收对象需要多少时间处于某个特定状态。

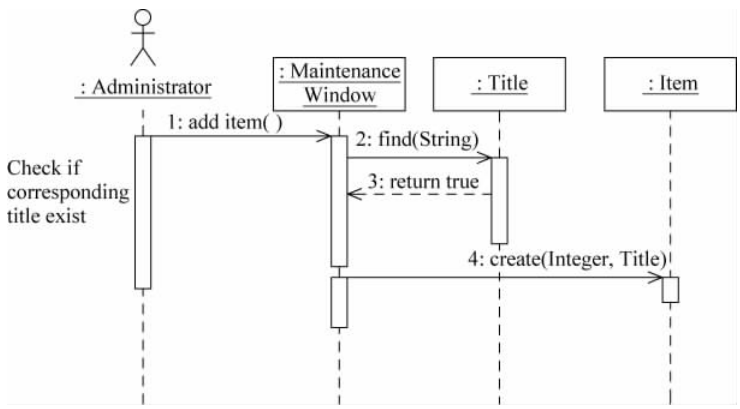

图 3-16　时序图

3.6.6　状态图

状态图使用有穷状态变迁图的方式刻画系统或元素的离散行为,可以用来描述一个类的实例、子系统甚至整个系统在其生命周期中所处状态如何随着外部激励而发生变化。

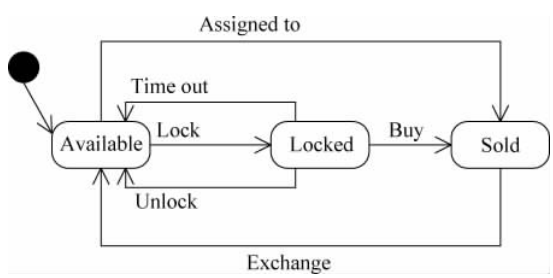

图 3-17　状态图

图 3-17 中包含 4 种状态,即初始状态、Available 状态、Locked 状态、Sold 状态。状态之间的转移如下。

(1) 初始状态转换到 Available 状态。

(2) 票被预订(Lock):Available 转换到 Locked 状态。

(3) 预定后付款(Buy):Locked 转换到 Sold 状态。

(4) 预定解除(Unlock)：Locked 转换到 Available 状态。
(5) 预定过期(Time out)：Locked 转换到 Available 状态。
(6) 直接购买(Assigned to)：Available 转换到 Sold 状态。
(7) 换其他票(Exchange)：该票重有效，Sold 转换到 Available 状态。

3.6.7 活动图

活动图是 UML 用于对系统的动态行为建模的另一种常用工具，它描述活动的顺序，展现从一个活动到另一个活动的控制流。活动图在本质上是一种流程图。活动图着重表现从一个活动到另一个活动的控制流，是内部处理驱动的流程，如图 3-18 所示。活动图主要用于：业务建模时，用于详述业务用例，描述一项业务的执行过程；设计时，描述操作的流程。UML 的活动图中包含的图形元素有动作状态、活动状态、动作流、分支与合并、分叉与汇合、泳道和对象流等。

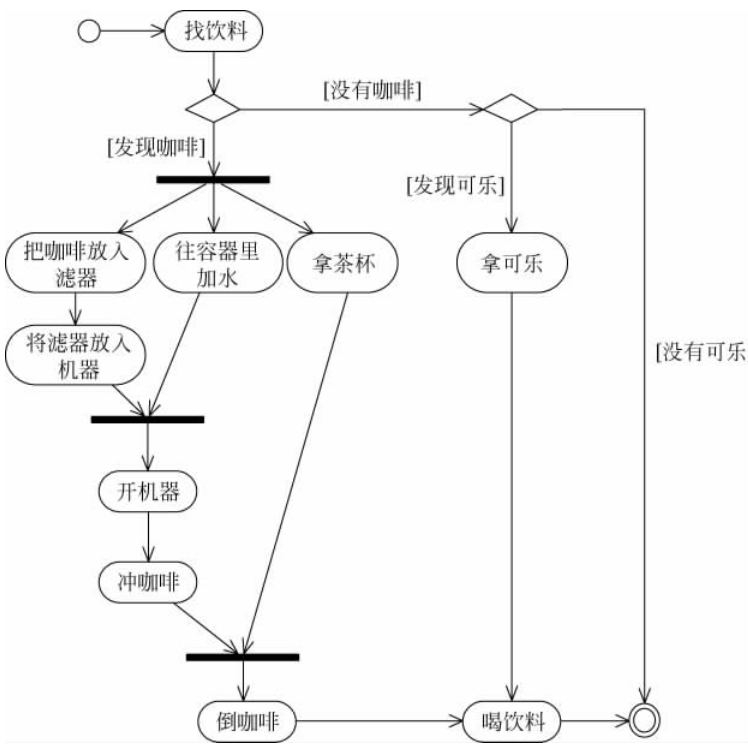

图 3-18 活动图

活动图与流程图的区别：活动图描述系统使用的活动、判定点和分支，看起来和流程图没什么两样，并且传统的流程图所能表示的内容在多数情况下也可以用活动图表示，但两者是有区别的，不能将两个概念混淆。

3.7 小结

本章详细介绍了软件体系结构和统一建模语言 UML 及其开发过程。软件体系结构模型用于对系统的用例、类、对象、接口、构件以及相互间的协作进行描述。面向对象系统的开发过

程以体系结构为中心，以用例为驱动，是一个反复、渐增的过程。UML 作为一种快速进行系统分析和设计的技术支持，适用于以面向对象技术来描述的系统及其整个系统开发的不同阶段。最后，本章对 UML 2.0 主要从结构建模和行为建模两个方面进行阐述，介绍了类图、对象图、构件图、部署图、用例图、顺序图、通信图、时序图、交互概览图、状态图以及活动图。

3.8 思考题

1. 在整个开发过程中，UML 主要起到什么作用？
2. 如何利用模式解决在面向对象系统分析与设计中遇到的问题？
3. UML 中都包含哪些图？简述这些图的作用。
4. 简述用例之间的关系。
5. 简述协作图和序列图的区别。
6. 神舟六号是神州系列飞船的一种，它由轨道舱、返回舱、推进舱和逃逸救生塔组成。航天员使用返回舱来驾驭飞船，轨道舱是航天员工作和休息的场所。在紧急情况下，航天员使用逃逸救生塔逃离。飞船的两侧有多个太阳能电池翼，它为飞船提供电能。根据以上描述画出能正确表示它们之间关系的 UML 图。
7. 某个网上银行的用户登录过程如下：用户先填写用户名和口令，要求登录。如果用户名和密码正确，则要求输入一个验证码。此时，该用户的手机上将收到一个短信，包含一个验证码，用户将此码填入下一个页面，再提交服务器。如果验证码正确，则能正常登录，并且验证码只能有效一次，用一个时序图描述这个过程。