第3章　软件体系结构描述

随着软件系统规模的扩大和复杂度的提高，系统的设计问题早已超出简单的数据结构以及计算设计，系统结构整体上的设计、软件系统的整体组织、更高层次系统构件的内部联系以及全局上的系统行为与特性成为软件开发设计阶段的核心。而面对越来越庞大的系统，运用严谨统一的描述方法或描述语言成为业界新的问题。如何应用软件体系结构的概念描述一个系统的体系结构是解决这一问题的关键。同时，系统体系结构的描述还要确保可以促进系统的开发、验证、测试。本章将会介绍软件体系结构的描述方法，目前软件体系结构描述方法分为两个角度，分别是将软件体系结构设计、描述与表示同传统的软件系统建模视为一体，将软件建模技术直接用来描述软件体系结构的 UML 和侧重于软件体系结构形式化理论研究的体系结构描述语言（Architecture Description Language，ADL）。这是两种不同的解决思路，各有优劣，可以从不同角度解决软件体系结构描述问题。

3.1　软件体系结构建模概述

3.1.1　软件体系结构建模问题

建模是对物理实体进行必要简化抽取其主要特征的一种手段。通过这种方法，人们获得了具体事物到抽象模型的一个映射。模型是经过抽象的原事物的一种特征提炼，通过对模型的分析和认识，人们可以更好地认识原事物的本质，从而达到开发利用的目的。在软件工程领域，软件模型可以将复杂庞大的系统进行必要的简化和抽象，使人们更专注软件本身的结构。在软件的整个生命周期里，需要使用不同的方法对系统建立各种模型，可以说，软件工程本身就是一个不断建立模型和转化模型的过程。

软件体系结构建模就是利用图形、文字、数学表达式等将软件的主要特征描述出来。通过软件建模过程，方便了设计人员、开发人员、测试人员、产品需求人员进行沟通、设计、开发、测试等一系列软件工程过程。通过建模可以将复杂的系统根据功能、过程、场景、交互、动作等诸多因素进行划分。建模过程将诸多交织在一起的元素进行划分，使系统模型简单、易懂，使人们更容易得到系统在某一领域的实质。通过建模过程加快了人们认识软件的过程，提高了开发效率和系统的稳定性和安全性，并可以对软件的可用性和稳定性进行评估。软件建模贯穿了软件开发的每一个环节。

软件体系结构建模具有如下优点：①有助于从整体上掌握软件架构；②有助于在软件的不同阶段形成清晰目标；③有助于从不同的角度掌握软件模型；④方便了需求沟通、设计修改、开发、测试等软件开发过程。

要建立软件体系结构模型，就离不开具体的方法，需要运用特定的工具达成目标。在软件体系结构建模发展过程中占主要地位的有两种方法：一种是运用图形可视化软件体系结构的方法（如 UML），该方法在产业界有大量运用并且已经形成一种产业标准，带有强烈的实践风格；另一种是运用体系结构描述语言（ADL）对软件体系结构进行描述，这种方法注重用数理逻辑等形式化的方法深度刻画软件体系结构，带有强烈的学术风格。

3.1.2　软件体系结构描述方法

软件体系结构是对构成软件的诸多要素在抽象层次上的一种描述。通过对软件体系结构的抽象描述客户、项目管理人员、开发人员、测试人员、维护人员等诸多参与方达成对软件的精确认识。为了保证软件的产品质量，需要有一种规范的方法描述软件体系结构。

1. 图形表示方法

通过矩形框表示抽象构件，运用文字标注构件名称，运用有向的线段符号代表构件进行控制、通信以及关联的连接件。这种方法的最大优势是易于理解沟通。尽管这种通过图形符号抽象软件的方法不够精确，在许多场合中存在语义模糊的问题，但是由于其简单、易用的特点，在实际的软件工程中，设计和开发阶段大量使用此类办法进行沟通，这种简洁、易懂的方法仍然受到开发人员或项目干系人的青睐，在软件设计中占据主导地位。

最简单的表示方式是没有任何规则与规范约束的软件结构示意图，这样的沟通大多在开发人员面对面交流时使用，但其缺陷也显而易见——这样的表达方式大多需要配有同步解说，才能理解信息输出者要表达的具体含义，单一的框图完全不能传达完整的设计意图，甚至由此产生的误解会对软件开发造成巨大的灾难。为了克服传统图形表达方法中缺乏的语义特征，有关研究人员也试图通过增加含有语义的图元素的方式开发图文法理论。当今应用最广泛的是 UML，它是一种半形式化的图形建模语言，给出了一定图形规范与对应的语义含义，包括用例图、类图等十几种。后文会有详细介绍。

2. 模块内连语言

通过将一种或几种程序设计语言的模块进行连接，得到的方法称为模块内连语言（Module Interconnection Language）。这种方法具有程序语言的严格语义，因此有能力对大型软件单元进行描述，尤其在模块化的程序设计和分段编译等程序设计与开发技术中发挥了很大作用。但是，语言处理和描述的软件设计开发层次过于依赖程序设计语言，限制了其处理和描述比程序设计语言元素更抽象的高层次软件体系结构元素的能力。

3. 基于软构件的系统描述语言

这种描述方法是以构件为单位的软件系统描述方法。它将许多特殊软件实体以特定的相互作用形式进行构造和组织，而这些软构件实体多是一些层次较低的以程序设计为基础的软件实体，以通信协作的方式相连接。因此，这种描述语言存在一定的局限性，通常只用于面向特定应用的特殊系统，所以并不适合一般的软件体系结构描述问题。

4. 软件体系结构描述语言

参照传统程序设计语言,针对软件体系结构描述这一问题,形成了专门的 ADL。ADL 由于具有程序设计语言的精确语义并有良好的整体性和抽象性,因此是当前软件开发和设计方法学中一种发展很快的软件体系结构描述方法。目前已经有几十种常见的 ADL,如 ACME、Aesop、ArTek、C2、Darwin、LILEANNA、MetaH、Rapide、SADL、UniCon、Weaves、WRIGHT、AADL(由 SAE 标准化)、AO-ADL、SysADL 等。

3.2 基于 UML 的软件体系结构描述

UML 是一种用于绘制软件蓝图的标准语言。可以用 UML 对复杂的软件系统进行可视化、详述、构造和文档化。UML 正如该描述标准名称一样,它的主要功能和意义体现在三方面:①统一,表示该描述标准是一种通用的标准,UML 被 OMG(Object Management Group)认可,成为软件工业界的一种标准。UML 提供一种统一的图形元素表达方式,具有确切的语义基础,表述的内容能被各类人员理解,包括客户、领域专家、分析师、设计师、程序员、测试工程师及培训人员等。他们可以通过 UML 充分理解和表达自己关注的内容。②建模,即建立软件系统的模型。为了说明建模的价值,Booch 曾给过一个类比:搭建一个狗窝和修建摩天大楼的区别。搭建一个狗窝也许无须使用建模方法,但是,如果想构建一栋摩天大楼模型,就必然会用到建模方法。③语言,它是一套按照特定规则和模式组成的符号系统。它用半形式化方法定义,即采用直观的图形符号、自然语言和形式化语言相结合的方法描述软件体系结构。

3.2.1 UML 概述

UML 是软件界第一个统一的建模语言,该方法结合了 Booch、OMT 和 OOSE 方法的优点,统一了符号体系,并从其他方法和工程实践中吸收了许多经过实际检验的概念和技术。

在 UML 之前,已经有一些试图将各种方法中使用的概念进行统一的初期尝试,比较有名的是 Coleman 和他的同事们[Coleman-94]创造的,包括 OMT[Rumbaugh-91]、Booch[Booch-91]、RC[Wirfs-Brock-90]方法的 Fusion 方法。面向对象软件工程的概念最早由 Booch 提出,他是面向对象方法最早的倡导者之一。后来,Rumbaugh 等人提出面向对象的建模技术(OMT)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。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,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。由于这项工作没有这些方法的原作者参与,实际上仅形成一种新方法,不能替换现存的各种方法。1994 年,Jacobson 提出了 OOSE 方法,其最大的特点是面向用例,并在用例的描述中引入了外部角色的概念。1994 年 10 月,Grady Booch 和 Jim Rumbaugh 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 UM 0.8,后来 Jacobson 也加盟到这一工作中,经过 Booch、Rumbaugh 和 Jacobson 三人的共同努力,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。1996 年,一些公司和组织(如 DEC、HP、IBM、Microsoft、Oracle、Rational Software 等)成立了 UML 成员协会,以完善、加强和

促进 UML 的定义工作。为了组织、简化和改善互操作，1996 年 6 月 6 日，OMG 成立了面向对象分析设计工作组（OA&DTF，即 ADTF 的前身），颁布了一个征求建议书（RFP）。19 家公司共同组成 6 个团队，负责制作各自的建议书相互竞争。接下来的几个月，这些团队一起工作，把他们的工作成果合并成一份 UML 建议书。1997 年 9 月 25 日，面向对象分析设计工作组投票，一致同意接纳。从 UML 属于 OMG 之日起，OMG 对 UML 的修改从来没有停止过。从 1997 年到现在，OMG 对 UML 进行了多次修改（UML 1.1，UML 1.3，UML 1.4，UML 2.0，UML 2.1，UML 2.2，UML 2.3，UML 2.4.1，UML 2.5，UML 2.5.1）。UML 目前还在发展和完善中。

2017 年 12 月，OMG 发布了最新版本的 UML 2.5.1。UML 2.x 规范的目标是对文档本身进行了重构和简化，以使其更易读，阐明一个规范文档，减少实施中的问题，并且促进工具间的互用性。UML 也将持续改进，完善体系，形成一个业界更全面、更广泛、更严谨的建模方式，提供长期有保证的更高水平的抽象层，从而促进软件开发生产力的飞跃。

3.2.2　UML 体系

UML 主要包含三方面内容：①UML 的基本构造块；②支配这些模块组合的规则；③整个语言的公共机制，如图 3-1 所示。UML 是一种软件体系结构的描述语言，属于软件开发方法中的一种。虽然 UML 独立于过程，但是 UML 最适于以用例为驱动、以体系结构为中心、增量和迭代的过程。

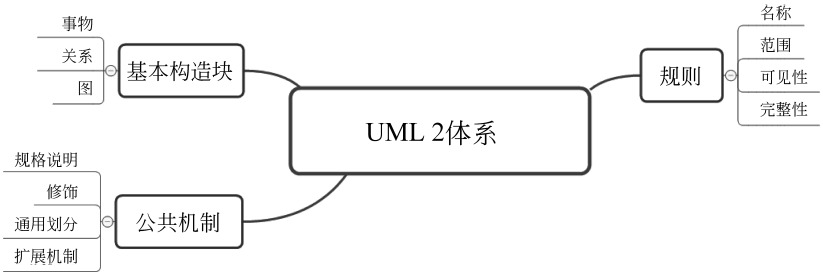

图 3-1　UML 2 体系

- **UML 基本构造块**

构造块是指 UML 的基本建模元素，包括事物（Thing）、关系（Relationship）和图（Diagram）3 部分，如图 3-2 所示。

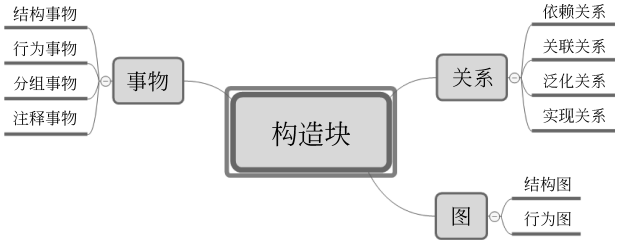

图 3-2　构造块

（1）事物是 UML 中最基本的构造块，根据构造块的属性，分为 4 类（结构事务、行为事

务、分组事务、注释事务)。

(2) 关系事务之间的关系可以归纳为4种(关联关系、泛化关系、依赖关系、实现关系)。

(3) 图是由一组事物通过关系组成的图形,图形的顶点代表事物,图形的弧表示关系。

UML 2.x提供了14种不同类型的图(UML 1.x中为9种),按照UML 2.5的分类方法可以划分成两类,如图3-3所示。

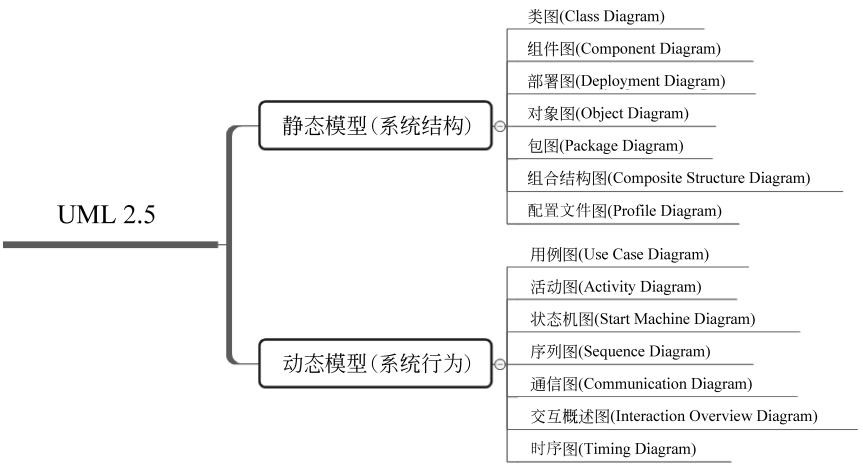

图3-3 UML图的分类

(1) 静态(或结构)视图:使用对象、属性、操作和关系强调系统的静态结构,包括类图和组合结构图等。

(2) 动态(或行为)视图:通过显示对象之间的协作以及对象内部状态的更改强调系统的动态行为,包括序列图、活动图和状态机图等。

对一个现实系统建模,一般通过如下7种图之一观察系统的静态(或结构)部分。

(1) 类图是最常用的图,描述了系统的静态设计,表达了系统的静态交互,其包含的主要元素为类、接口、协作。

(2) 组件图描述了一个封装的类,以及该类的接口、端口、内嵌的构件和连接件构成的内部结构。构件图同样用于表示系统的静态设计实现,可以将其理解为类图的变体。

(3) 部署图展示了系统所包含的节点以及节点之间的关系和配置。通常用部署图说明体系结构的静态部署结构。

(4) 对象图描述一组对象以及它们之间的关系。对象图描述了在类图中建立的事物实例的静态快照。同样,对象图也是用于表达系统的静态设计视图或静态交互图,但是对象图从实现或原型方面具体描述系统。

(5) 包图表示系统组织的子系统和区域。它还可以模拟包之间的依赖关系,并帮助将业务实体与用户界面、数据库、安全性和管理包分开。

(6) 组合结构图在运行时模拟组件或对象行为,显示系统执行期间组件的布局、关系和实例。

(7) 配置文件图允许创建可扩展的配置文件,这些配置文件可应用于从配置文件继承的元素。这些图表通过以受控方式扩展标准增加价值。

此外,还要用到另外7种图描述系统的动态(或行为)部分。

(1) 用例图描述了一组使用用例和参与者之间的关系。因为用例图描述了系统的静态

使用情况，所以对系统行为的组织和建模尤其重要。

（2）活动图用于描述系统动作中的控制流和数据流。活动图用于描述系统的动作、动作顺序和产生动作相关的命令和数据。

（3）状态机图显示内存中对象的运行时生命周期。这样的生命周期包括对象的所有状态以及状态改变的条件。

（4）序列图是描述系统消息时间次序的交互图。序列图描述了一组角色和隶属于该角色的实例发送和接收消息的时序，并且动态地描述了系统的交互。

（5）通信图用于描述收发消息对象的结构组织。序列图和通信图表达了类似的系统结构，但是侧重点不同，序列图强调时序，通信图强调消息流经的数据结构。

（6）交互概述图以一般的高级别呈现系统内交互的概述；它还使得能够理解 UML 图（如序列图）如何依赖于彼此并且彼此相关。

（7）时序图根据对象的时间轴模拟对象之间的交互。对象可以在这些图上具体显示，也可以是属于类的匿名对象。运行时对象之间的消息执行顺序由这些图很好地建模。

为了描述一个系统使用图为上述某种图之一或是组织自己定义的一类图，每种图必须拥有清晰的名称，以便能够和其他图区分，一般要将图组织成包。

- **UML 规则**

在 UML 中，代表事物的构造块在使用时应遵守的语法规则为每个构造块必须有名称、范围、可见性、完整性等。

名称：每个构造块都应该有一个名称。

范围：每个构造块都有一定的作用范围或者作用域。

可见性：构造块在类或包中的访问权限或访问级别。例如，Java 语言中的类有 public、protected、private、package 4 种访问级别。

完整性：同一个构造块代表的事物在不同的模型中的语义必须一致。

- **UML 公共机制**

UML 提供了 4 种公共机制，如图 3-4 所示，描述了达到对象建模目标的 4 种策略，通过对规格说明、修饰、公共划分、扩展机制的运用保持整体的简化，这里不做详细介绍。

图 3-4　公共机制

3.2.3　UML 的软件体系结构描述

下面对几类重要的图进行说明。

1. 类图

类图是以类为单元进行组织，描述系统各个类之间关系的静态结构。类图常用于面向对象系统建模，描述了一组类、接口、协作以及它们之间的关系。类图包括名称（Name）部

分、属性(Attribute)部分和操作(Operation)部分。

类用来定义一组具有类似属性和行为的对象。属性通常包括类所描述事物的特征以及这些特征具有的数据类型,如整型、浮点型、布尔型等。事物的行为用操作描述,具体的方法则是操作的实现。通常通过关联关系描述给定类与具体对象之间的关系。类元之间有关联、泛化、流等依赖关系,见表3-1。

表3-1 UML中类的关系

关 系	功 能	表 示 法
关联	类实例之间连接的描述	———
依赖	两个模型元素间的关系	- - - - - -→
流	在相继时间内一个对象的两种形式的关系	- - - - - -→
泛化	父类与子类的关系	———▷
实现	说明和实现的关系	- - - - - -▷
使用	一个元素需要其他元素提供适当功能的情况	- - - - - -→

例如,一个图书管理系统类图中表示了几个重要的类,如借阅者(borrower)、书刊(title)、物理书刊(Book)、借阅记录(loan)、预定记录(reservation)。其中借阅者有身份(如身份证可表征其身份)、相关行为(如借阅、返还、预订等)。书刊有身份(如ISBN/ISSN可表征其身份)、相关行为(如可被预订或取消预订等)。物理书刊有身份(如索引号可表征其身份)、相关行为(如可被借阅或返还等)。借阅记录有身份,若同一人借不同的书则记录不同;借阅记录有相关行为,如可被预订或取消预订等。预订记录有身份,如同一书刊被不同人预订则记录不同;预订记录有相关行为,如可被创建或删除等。这些类的描述如图3-5和图3-6所示。

2. 用例图

用例实例是系统在运行时产生的一系列动作,这些动作表示了某个特定的输入被系统捕获之后,如何被处理程度处理,并将输出传递给参与者的过程。用例图描述了一个和系统进行交互的动作序列。用例模型描述了系统外的使用者所理解的系统功能。通过用例图可以在不揭示系统内部构造的情况下定义连贯的行为。

用例除与参与者发生关联外,还可以参与系统中的多个关系,见表3-2。

表3-2 用例之间的关系

关 系	功 能	表 示 法
关联	参与者与参与执行的通信途径	———
扩展	在基础用例上插入基础用例不能说明的扩展部分	<<extend>> - - - - - -→
包括	在基础用例上插入附加的行为,并且具有明确的描述	<<include>> - - - - - -→

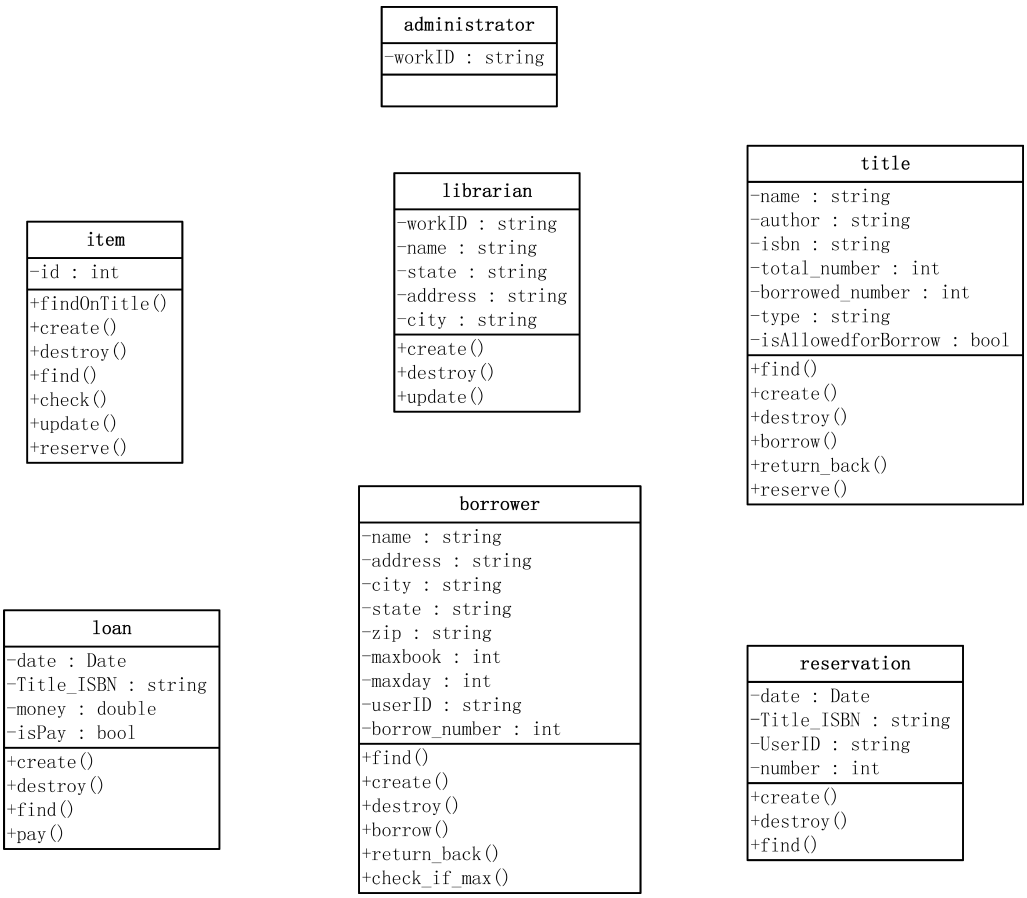

图 3-5　图书管理系统类图 1

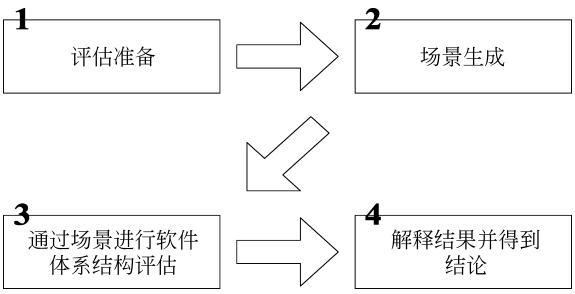

图 3-6　图书管理系统类图 2

图 3-7 是图书管理系统的用例图，参与者包括借书者与图书管理员。在图书管理系统中，借书者不直接使用系统，而是通过图书管理员进行借书、还书、取消预订等操作。

3. 序列图

序列图用来描述对象之间动态的交互关系，着重体现消息传递的时间顺序。序列图直观勾画出了系统内对象的生存期，在生存期内，对象可以接收消息并做出响应，如图 3-8 所

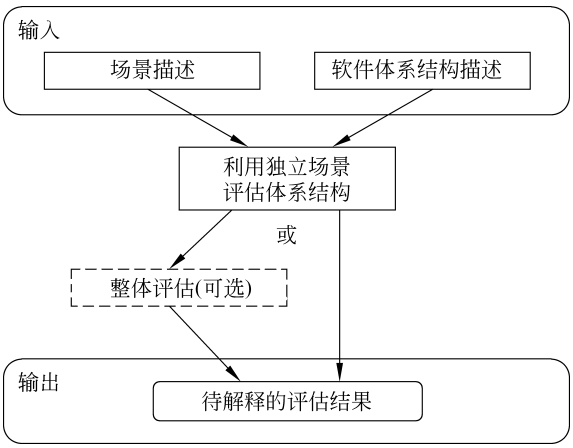

图 3-7　图书管理系统的用例图

示。在图书管理系统中,序列图存在两个轴:水平轴上包含不同的对象,垂直轴表示时间,表示对象及类的生命周期。消息可以用消息名以及相应的参数表示,可以含有消息序号。有时消息上具有条件表达式,表示决定或者分支是否发送消息。

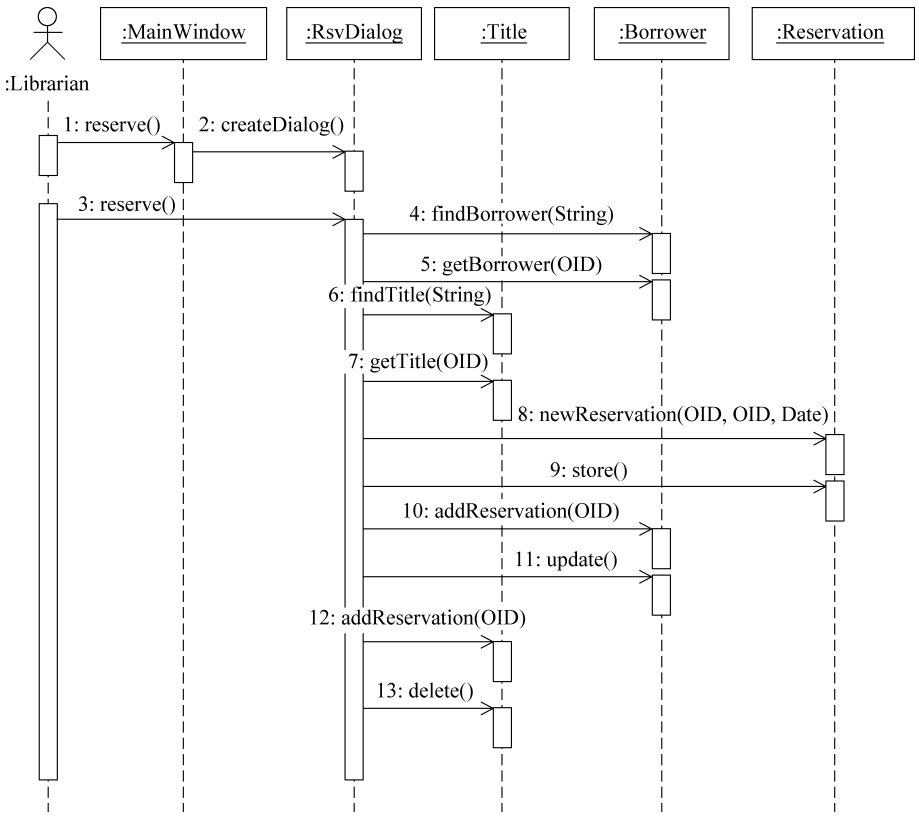

图 3-8　图书管理系统的序列图

4. 状态机图

状态机图用来描述系统状态和事件之间的关系，通常用状态图描述单个对象的行为。通过状态图确定了系统在事件激励下的状态序列。状态图用于表述在不同用例之间的对象行为，但并不适合表述协作对象行为。状态图通常使用在业务流程、控制对象、用户界面设计方面。图 3-9 是一本图书的入库、出借、出库等状态以及相应动作激励的状态图描述。

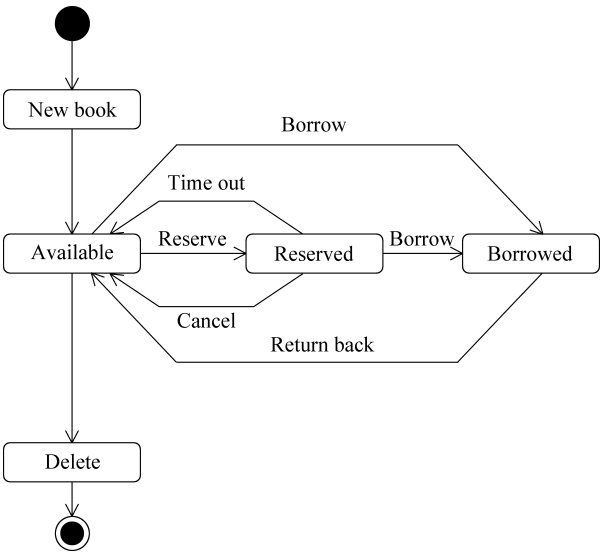

图 3-9　图书管理系统的状态机图

5. 部署图

部署图也称为实施图，用于描述位于节点实例上的运行构件实例的组织情况。部署图清晰地反映了系统硬件的物理拓扑结构，以及在此结构上可执行的软件。部署图有助于各方理解计算节点的拓扑结构和通信路径、节点上的软件构件等部署情况。图 3-10 是图书管理系统的部署图，该图表示了构成图书管理系统的各节点的通信互联情况以及各节点的软硬件情况。

6. 包图

系统模型可以从某一个角度以一定的精确性对系统进行描述，它由一系列模型元素（如类、状态机和用例）构成的包组成，模型的每一部分隶属于一个特定的包。包图是从维护控制角度对系统总体结构进行建模的一种方法，通过包图可以方便地处理整个模型。图 3-11 所示为图书管理系统的包图。

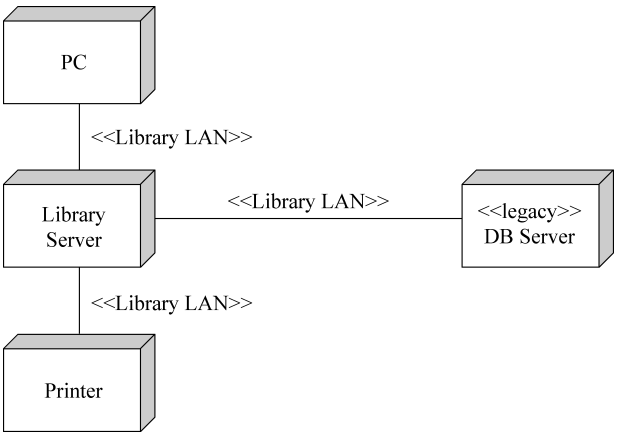

图 3-10 图书管理系统的部署图

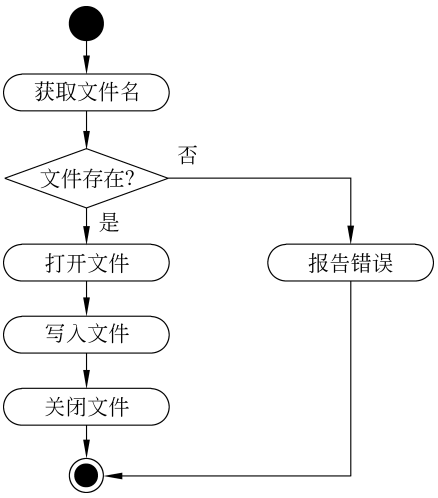

图 3-11 图书管理系统的包图

3.3 UML 体系结构描述方式案例分析

3.3.1 "4+1"视图模型

UML 是一种通用可视化的建模语言,在工业界有大量应用。在软件工程过程中,UML 往往应用于用例驱动以体系结构为中心的、迭代的和渐增的开发过程。由于 UML 在发展过程中结合了面向对象技术,UML 图形具有相对一致的外观和语义,出现了许多 UML 建模工具,较好地支持了 UML 建模过程。UML 中组件和概念之间没有显著的划分界限,一般用视图区分组件和概念。视图只是 UML 表达软件结构的一种手段,在每类视图中使用一种或几种特定的图可视化表示视图中的概念。1995 年,Kruchten 提出著名的"4+1"视图模型,从 5 个不同的角度刻画软件体系结构。这 5 种视图分别为逻辑视图、开发视图、过程视图、物理视图和场景视图。每类视图从一个角度描述软件的一个侧面,几种视图相结合就可以对软件体系结构有整体的把握。在 OMG 建立的 UML 标准中,使用了 Kruchten 定义的"4+1"视图。"4+1"视图模型如图 3-12 所示。

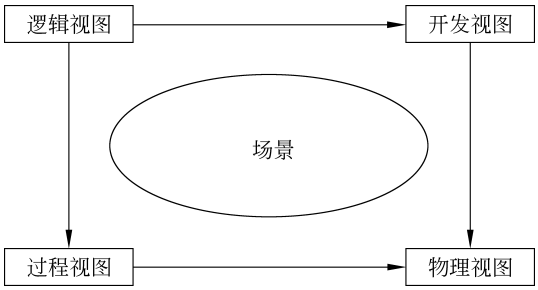

图 3-12 "4+1"视图模型

逻辑视图主要支持功能性需求,即在为用户提供服务方面系统应该提供的功能。系统

可分解为一系列的关键抽象，其中大多数来自问题域，表现为对象或对象类的形式。它们采用了抽象、封装和继承的原理。分解并不仅是为了功能分析，也用来识别遍布系统各个部分的通用机制和设计元素。类图用来显示一个类的集合和它们的逻辑关系：关联、使用、组合、继承等。相似的类可以划分成类集合。类模板关注单个类，它们强调主要的类操作，并且识别关键的对象特征。如果需要定义对象的内部行为，则使用状态转换图或状态图完成。公共机制或服务可以在类功能（class utilities）中定义。对于数据驱动程度高的应用程序，可以使用其他形式的逻辑视图，如用 E-R 图代替面向对象的方法（OO Approach）。

过程视图考虑一些非功能性的需求，如性能和可用性。它解决并发性、分布性、系统完整性、容错性的问题，以及逻辑视图的主要抽象如何与进程结构配合在一起，即在哪个控制线程上，对象的操作被实际执行。进程架构可以在几种层次的抽象上进行描述，每个层次针对不同的问题。在最高的层次上，进程架构可以视为一组独立执行的通信程序（processes）的逻辑网络，它们分布在整个一组硬件资源上，这些资源通过 LAN 或者 WAN 连接起来。多个逻辑网络可能同时并存，共享相同的物理资源。例如，独立的逻辑网络可能用于支持离线系统与在线系统的分离，或者支持软件的模拟版本和测试版本的共存。进程是构成可执行单元任务的分组。进程代表了可以进行策略控制过程架构的层次（即开始、恢复、重新配置及关闭）。另外，进程可以就处理负载的分布式增强或可用性的提高而不断被重复。

开发视图关注软件开发环境下实际模块的组织。软件可打包成小的程序块（程序库或子系统），它们可以由一位或几位开发人员开发。子系统可以组织成分层结构，每个层为上一层提供良好定义的接口。系统的开发架构用模块和子系统图表达，显示了“输出”和“输入”关系。完整的开发架构只有当所有软件元素被识别后，才能加以描述。但是，可以列出控制开发架构的规则：分块、分组和可见性。大部分情况下，开发架构考虑的内部需求与以下几项因素有关：开发难度、软件管理、重用性、通用性及由工具集、编程语言带来的限制。开发架构视图是各种活动的基础，如需求分配、团队工作的分配（或团队机构）、成本评估和计划、项目进度的监控、软件重用性、移植性和安全性。

物理视图主要关注系统非功能性的需求，如可用性、可靠性（容错性）、性能（吞吐量）和伸缩性。软件在计算机网络或处理节点上运行，被识别的各种元素（网络、过程、任务和对象）需要被映射至不同的节点；人们希望使用不同的物理配置：一些用于开发和测试，另外一些用于不同地点和不同客户的部署。因此，软件至节点的映射需要高度的灵活性及对源代码产生最小的影响。

场景视图综合了所有的视图，四种视图的元素通过数量比较少的一组重要场景（更常见的是用例）进行无缝协同工作，人们为场景描述相应的脚本（对象之间和过程之间的交互序列）。场景视图是其他视图的冗余（因此“+1”），但它起到了两个作用：作为一项驱动因素发现架构设计过程中的架构元素；作为架构设计结束后的一项验证和说明功能，既以视图的角度说明，又作为架构原型测试的出发点。

在 UML 中，逻辑视图可以用用例图实现。用例图将系统功能划分成对参与者有用的需求，从参与者的角度描述对系统的理解，每一个用例相当于一个系统功能。开发视图可以用 UML 中的类图、对象图和构件图表示模块，用包表示子系统，用连接表述事物间的关联。

过程视图可以采用UML中的状态图、顺序图和活动图实现。通过状态图可以帮助设计人员更清晰地了解用例,以及用例之间的交互关系,得到体系结构的动态过程。物理视图则可以使用UML中的配置图和部署图实现。此外,运用协作图描述构件之间的消息传递及其空间分布。

3.3.2 教务管理系统的非形式化描述案例

下面以一个教务管理系统为例说明如何运用"4+1"思想和UML刻画一个真实的系统。

教务管理系统包含用户登录、学籍管理、选课管理、成绩管理、教学管理、排课管理、系统维护功能。其中用户登录功能包括用户信息、用户注销退出,学籍管理功能包括学生信息查询(个人信息查询、查询专业计划、查询课程信息)、学生异动、生源录入注册、学生资料修改,选课管理功能包括网上选课、个人课表查询、课程详情查询、选课约束设置、增删课程,成绩管理功能包括查询本学期成绩、不及格成绩、专业计划完成情况、成绩错误报告、监控成绩录入情况、核实成绩表,教学管理功能包括课程库管理、教工库管理、教学日历查询、课表查询(个人课表查询、全校课表查询)、评估结果查询、历年数据查询,排课管理功能包括课程录入、课程表生成,系统维护功能包括数据维护、代码维护。

1. 教务管理系统逻辑视图

图3-13是教务管理系统的顶层用例图。通过该图可以清楚地看到不同身份的参与者使用了系统的哪些功能,与系统哪些模块进行了交互。例如,教师主要访问系统中的用户登录、教学管理和成绩管理模块。学生主要访问学籍管理、选课管理等模块。系统维护模块、排课管理模块只有教务员才有权限访问。

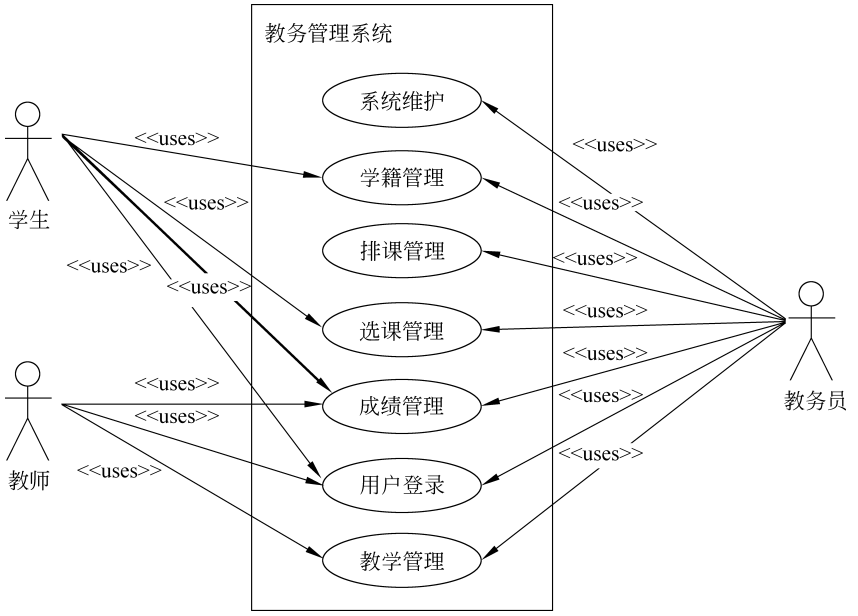

图3-13 教务管理系统的顶层用例图

具体到一个角色，可以根据软件模块与功能和该角色拥有的操作权限进行具体描述，如图 3-14 所示。该图给出了参与者作为学生时使用的系统功能，并绘制出了系统功能与子模块之间的联系。同理，可对教务管理系统中教师和教务员的角色与系统交互情况进行用例描述。

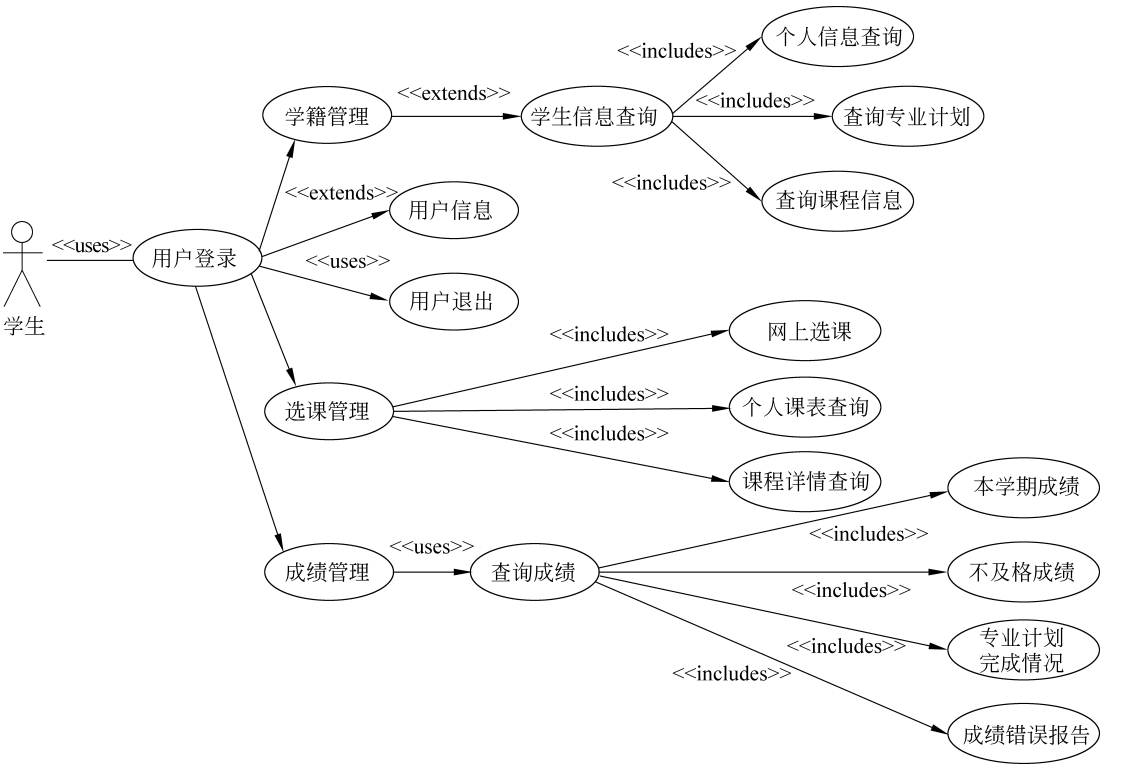

图 3-14　教务管理系统的学生用例图

2. 教务管理系统的过程视图

图 3-15 是教务管理系统学生选课活动图，它展示了学生选课时要进行的活动。框图之间的箭头说明了活动的时间依赖关系。例如，在选课前必须进行登录，而在选课过程中需要考查专业是否冲突、人数是否选满等因素。

图 3-16 是教务管理系统学生选课顺序图。通过该顺序图可以勾勒出学生选课时同系统之间以及系统内各模块间的基本通信过程。

图 3-17 是教务管理系统学生选课协作图，该图表明了选课操作涉及的各个对象间的操作关系，虽然顺序图和协作图都可以表示各对象间的交互关系，但是二者的侧重点不同。顺序图用消息的几何排列表达消息的时间顺序，各角色之间的相关关系是隐含的。协作图中使用各个角色的几何排列图形表达角色之间的关系。

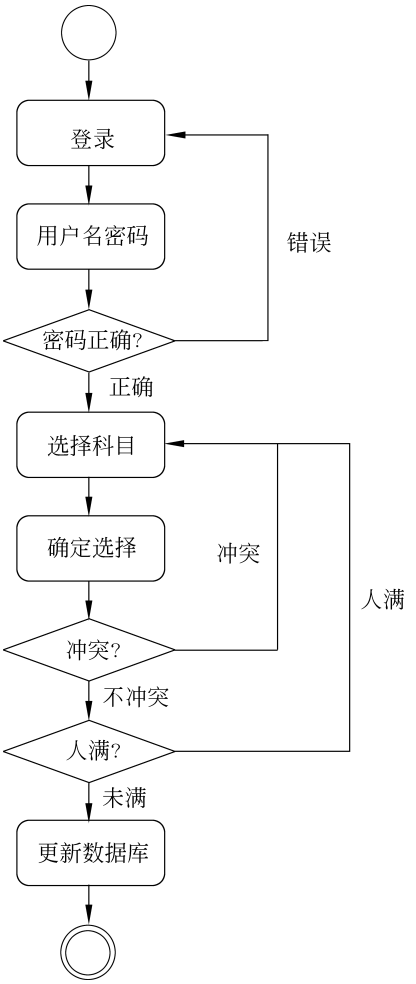

图 3-15　教务管理系统学生选课活动图

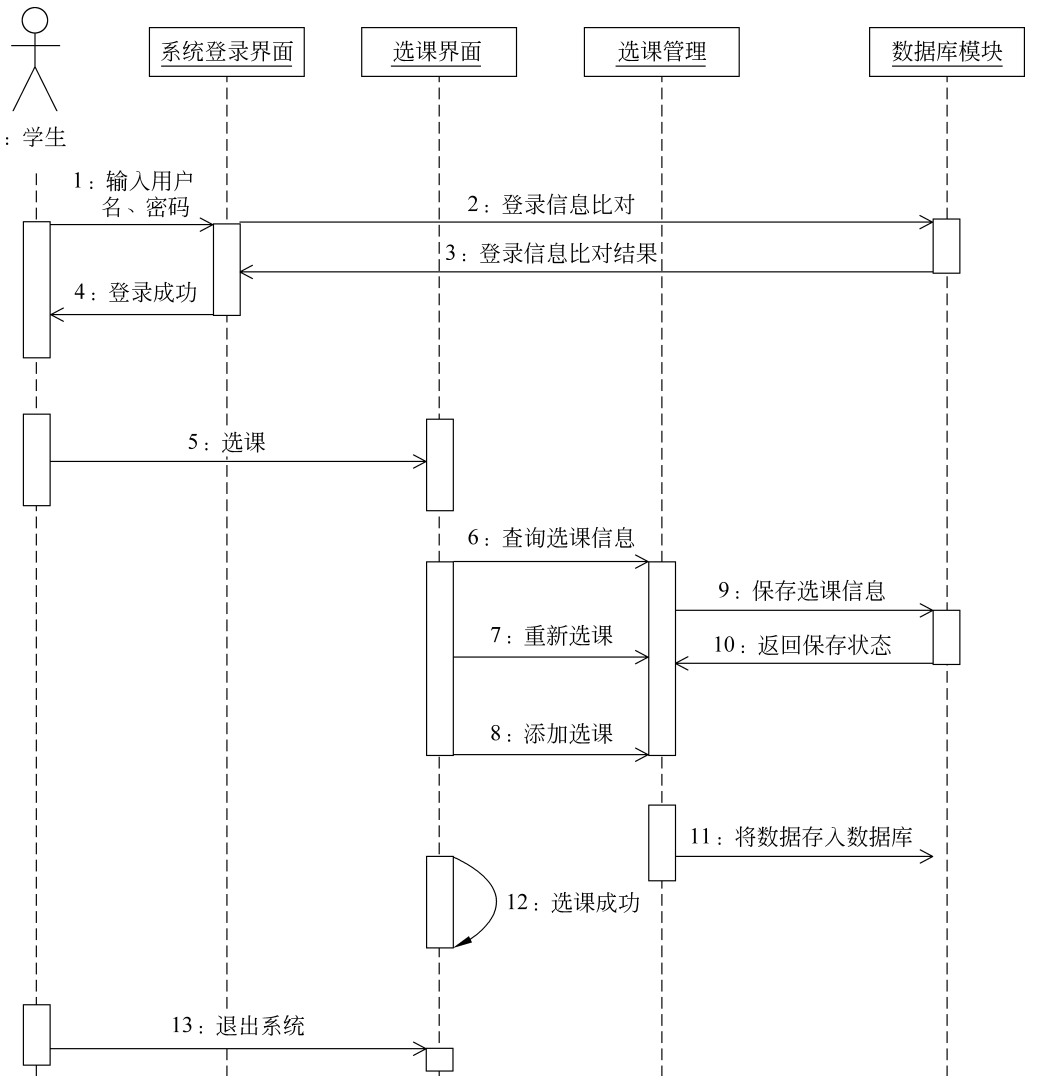

图 3-16　教务管理系统学生选课顺序图

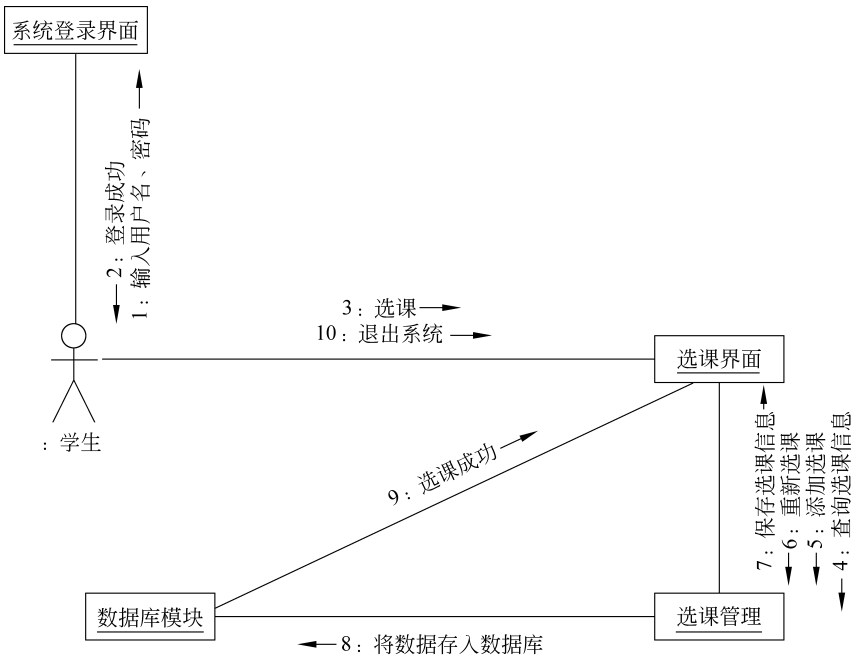

图 3-17　教务管理系统学生选课协作图

3. 教务管理系统的开发视图

类图和构件图用于描述软件体系结构中的开发视图，如图 3-18～图 3-20 所示。在教务管理系统中，通过类图展示不同身份的对象具有的不同属性和操作，以及不同的消息实体具有的属性。

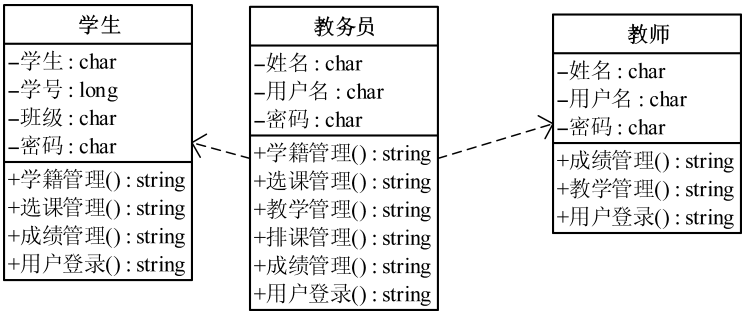

图 3-18　教务管理系统人员类图

构件图表示了构件之间的依赖关系。构件之间定义了良好的物理实现单元，每个构件包含了系统设计中相关性比较强的一组类。构件图展示了构件间相互依赖的网络结构。如图 3-20 所示为教务管理系统的构件图。

4. 教务管理系统的物理视图

通过图 3-21 所示的部署图描述教务管理系统的物理部署情况。部署图显示了不同组

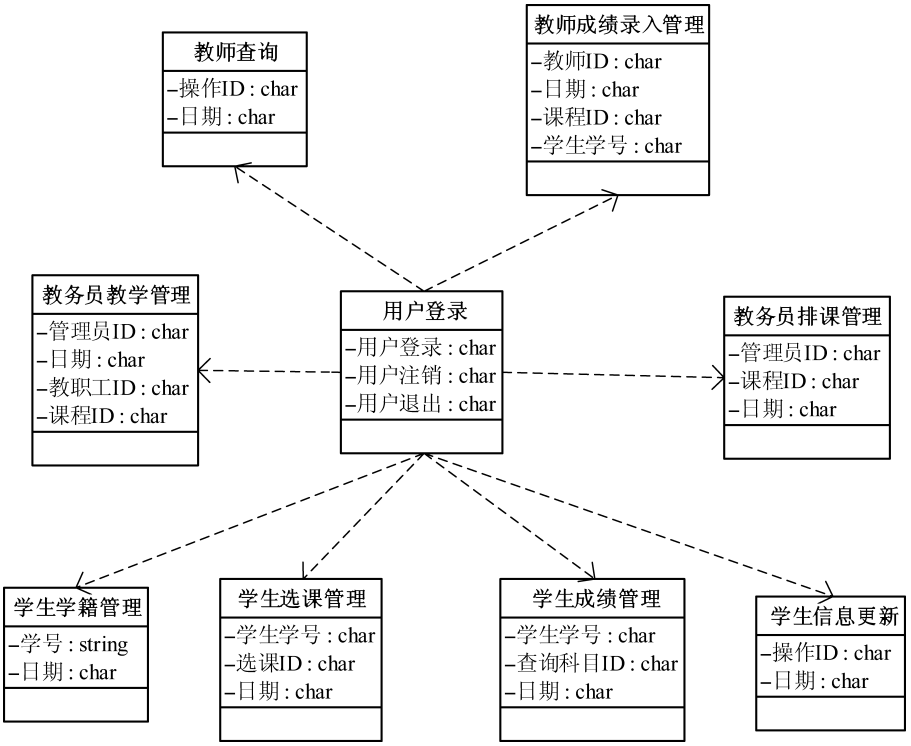

图 3-19　教务管理系统事物信息类图

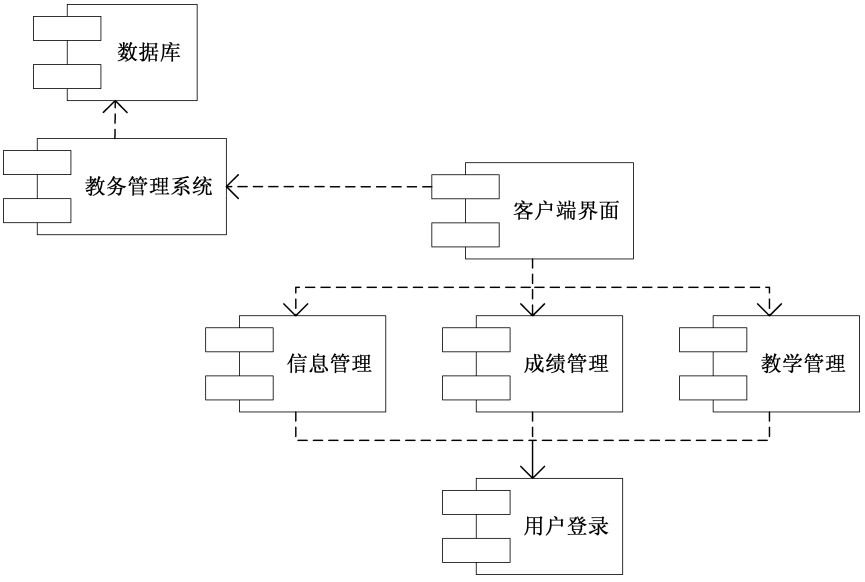

图 3-20　教务管理系统的构件图

件在硬件上的物理分配情况，以及这些节点间的通信互联情况。通过部署图可以明确软件部署的运行情况。

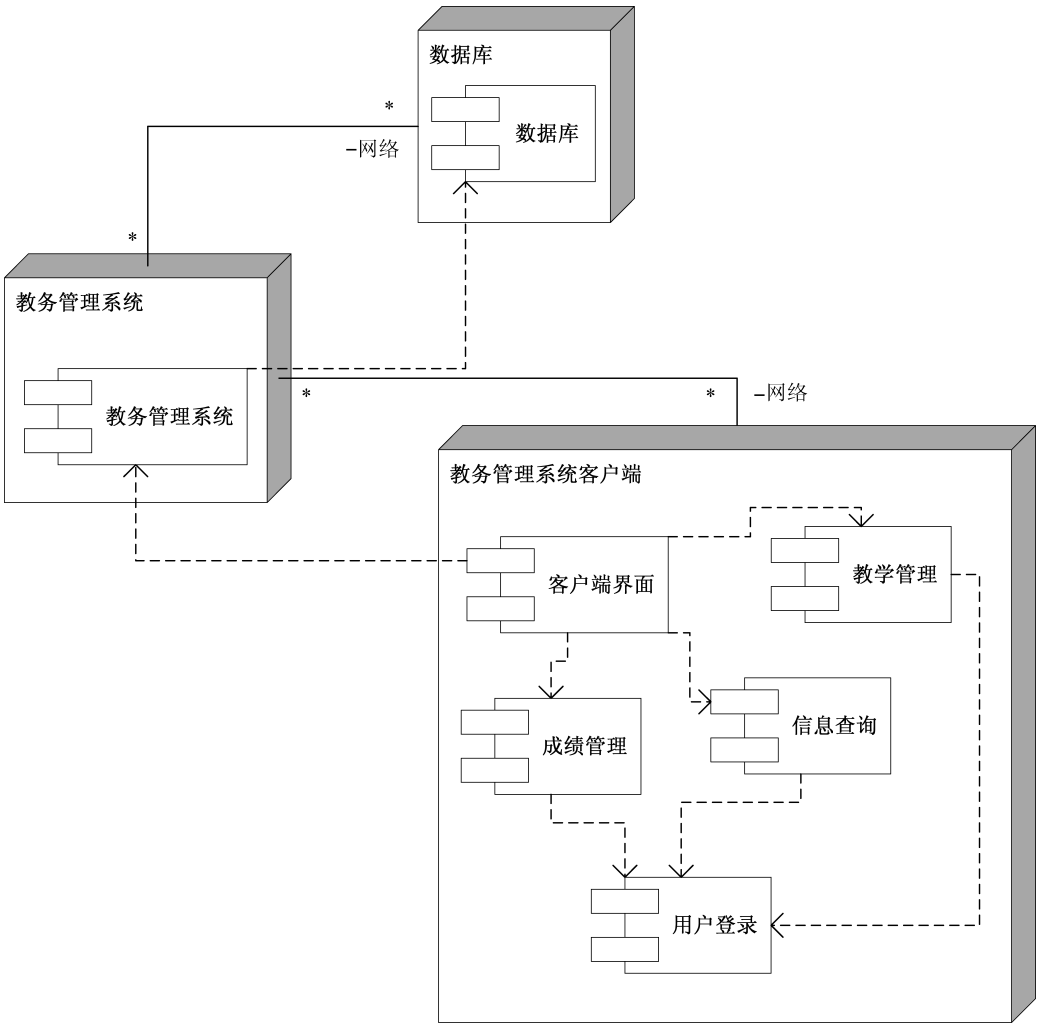

图 3-21 教务管理系统部署图

3.4 基于 ADL 的软件体系结构描述

在计算机科学中,形式化方法指的是用于软件与硬件系统的说明、开发与验证的数学化方法。所谓形式化的基础就是数学化理论。

在一个软件开发过程中,形式化方法可以被应用到各个方面。形式化方法在体系结构描述上具有如下特点:

(1) 形式化方法可以用于系统描述,而且可以在不同层次上进行描述。

(2) 通过提供系统结构抽象级别上的精确语义,系统的形式化模型可以对系统的关键属性提供严谨的分析。形式化方法实现了对体系风格与结构风格的建模与分析。精确的数学符号和计算规则消除了语义上的模糊。软件设计者可以精确表述其理念与系统需求。在开发阶段,消除了二义性,所有的设计工作都将遵循某种数学理论。形式化方法拥有唯一的

标准与解释规则，全程指导开发者的探讨与工作，使得由于语意模糊造成的沟通障碍也迎刃而解。

与非形式化相比，形式化描述的主要优点如下：

(1) 形式化方法更利于表达与计算。

(2) 形式化方法提供了形式化且准确的定义，用于描述行为、行为模式、行为分析等。例如，进程代数提供了代数法则，用于处理分析等。

对行为的分析与建模成为系统的重要组成部分。

对行为描述的良好支持有利于系统验证。由于精确定义了系统必须符合的约束，所以可以判别一个系统是否与某一个体系结构相符，或者给定的体系结构是否属于某一个风格。

利用形式化方法的原理，系统正确性的验证可以通过自动化的方法证明。一般来说，自动化方法包含如下两类。

(1) 自动化理论证明，在给定系统描述的情况下，基于逻辑原语与推理规则进行形式化证明。

(2) 模式识别，系统通过对可能达到的状态进行完全搜索的方式验证特定的属性。

基于上述属性，形式化方法在一些高综合性系统应用中尤其重要，如那些对安全性要求比较高的系统。形式化方法在需求和说明方面十分有效，可用于完全的形式化开发。

关于形式化方法的争议虽然会持续很久，但形式化方法已经显示了杰出的技术能力。作为成功的指导，它们同样是软件体系结构描述的必要方法。必须注意到形式化方法仅拥有数学理论，而缺乏对系统结构的表达。没有对软件系统拓扑信息的描述，数学理论将毫无意义，需要的是一条将形式化方法与软件体系结构连接在一起的纽带。正是基于此考虑，才引入了 ADL。

3.4.1　ADL 概述

许多软件系统的开发都开始于体系结构设计，尤其是大型系统。好的设计通常是软件成功的重要因素之一。

给 ADL 下一个非正式的定义，ADL 是一种用于描述软件与系统结构的计算机语言。这个粗略的定义关注的仍然是 ADL 的使用意图，而缺乏必要的规范定义。人们当然希望能够通过查字典查到 ADL 的准确定义，很遗憾，相关研究领域对 ADL 仍有争议。争议集中在 ADL 是什么？ ADL 应该对系统从哪些角度建模？ 在交互语言(interchange language)中应该交互什么？ 对此，学术界仍然很难达成共识。随着 ADL 家族的发展壮大，学术争论愈演愈烈，对其定义达成共识就愈发困难了。

ADL 是一门用于描述的语言，可以在指定的抽象层次上描述软件体系结构。它通常拥有形式化的语法语义以及严格定义的表述符号，或者是简单易懂的直观抽象表达。前者可以向设计者提供强而有力的分析工具、模式识别器、转化器、编译器、代码整合工具、支持运行工具等；后者可以借助图形符号提供可视化模型，便于理解对系统的相关分析。大多数 ADL 都依靠形式化方法支持对系统描述的分析与验证，也有一些 ADL 仅关注结构化的语法语义，并且结合其他 ADL 完成形式化的描述与分析。后一种语言被认为是交互语言，交互语言也是 ADL。很遗憾，还不能提出准确、简练的 ADL 的定义，而是采用一个段落的篇幅对现今的 ADL 研究进行了总结。