第3章 UML与面向对象开发方法

知识目标

- 理解面向对象的概念；
- 掌握面向对象建模的基本步骤；
- 理解面向对象的分析与设计方法；
- 理解面向对象的实现方法；
- 理解UML对面向对象开发的支持。

技能目标

- 能够利用已有知识学习新知识。

学习过面向对象程序设计语言(例如Java、C++)的读者对面向对象的概念已经有了一定的了解。本章将简要复习面向对象的概念，并介绍面向对象分析与设计、面向对象建模等面向对象技术。

3.1 面向对象概念

20世纪60年代中期以来，基于计算机的系统规模越来越大，系统的复杂性也不断增加，人们很难把握软件开发过程，开发出来的软件质量越来越难以保证，软件生产率急剧下降。由于大型软件开发的工程性和实际开发工作中手工性的不相适应，出现了软件危机。为解决上述问题，在1968年的北大西洋公约学术会议上提出了软件工程的思想，希望采用工程方法来开发软件。四十多年来这一思想逐步为人们所重视，其发展迅速，并取得了令人瞩目的成就。但这一时期该领域采用的方法主要是结构化的分析与设计方法，这种方法对现代日益复杂的软件开发而言存在严重的不足，由此产生的软件危机并未真正得到解决。因此，人们正设法寻找一种新的有效途径来解决这一问题。

面向对象概念自20世纪60年代出现至今，面向对象技术已成为一种完整的思想与方法体系，并且在计算机领域中得到广泛应用，如在程序设计中的面向对象程序设计、在人工智能中的面向对象知识表示、在数据库中的面向对象数据库、在人机界面中的面向对象图形用户界面、在计算机体系结构中的面向对象结构体系等都有突出表现。由于面向对象技术在软硬件开发方面呈现出巨大的优越性，人们将其视为解决软件危机的一个很有希望的突破口。

面向对象的方法启发我们从现实世界中客观存在的事物出发构造软件系统，并在系统构造过程中尽可能地运用人的自然思维方式。它强调直接以问题域中的事物为中心来思考问题、认识问题，并根据这些事物的本质特征把它们抽象为解空间中的对象，以对象作为系统的基本构成单位。这样可以使系统直接映射问题域，最大限度地保持问题域中的事物及其相互关系的本质，使得解空间和问题域能够在结构上取得尽可能一致。这样做就向着减少语义断层的方向迈出了一大步，在许多系统中解空间对象都可以直接模拟问题空间的对象，因此，这样的程序易于理解和维护。

面向对象方法比以往的方法更接近人类的自然思维方式。虽然结构化开发方法也采用了符合人类思维习惯的原则与策略，但是与传统的结构化开发方法不同，面向对象方法更加强调运用人类在日常生活中的逻辑思维中采用的思想方法，并以其他人也能理解的方式将自己的思想表达出来。

面向对象的软件技术以对象(Object)为核心，用这种技术开发出的软件系统由对象组成。对象是对现实世界实体的正确抽象，它是由描述内部状态、表示静态属性的数据，以及可以对这些数据施加的操作(表示对象的动态行为)封装在一起所构成的统一体。对象之间通过传递消息互相联系，以模拟现实世界中不同事物彼此之间的联系。

面向对象的设计方法与传统的面向过程的方法有本质不同，这种方法的基本原理是，使用现实世界的概念抽象地思考问题从而自然地解决问题。它强调模拟现实世界中的概念而不强调算法，它鼓励开发者在软件开发的绝大部分过程中都用应用领域的概念去思考。在面向对象的设计方法中，计算机的观点是不重要的，现实世界的模型才是最重要的。面向对象的软件开发过程从始至终都围绕着建立问题域的对象模型来进行。对问题域进行自然的分解，确定需要使用的对象和类，建立适当的类等级，在对象之间传递消息实现必要的联系，从而按照人们习惯的思维方式建立起问题域的模型，模拟客观世界。

传统的软件开发方法可以用“瀑布”模型来描述，这种方法强调自顶向下按部就班地完成软件开发工作。事实上，人们认识客观世界，解决现实问题的过程，是一个渐进的过程，人的认识需要在继承以前的有关知识的基础上，经过多次反复才能逐步深化。在人的认识深化过程中，既包括从一般到特殊的演绎思维过程，也包括从特殊到一般的归纳思维过程。人在认识和解决复杂问题时使用的最强有力的思维工具是抽象，也就是在处理复杂对象时，为了达到某个分析目的集中研究对象的与此目的有关的本质部分，忽略该对象的那些与此目的无关的部分。

面向对象方法学的基本原则是按照人类习惯的思维方法建立问题域的模型，开发出尽可能直观、自然的表现求解方法的软件系统。面向对象的软件系统中广泛使用的对象，是对客观世界中实体的抽象。对象实际上是抽象数据类型的实例，提供了比较理想的数据抽象机制，同时又具有良好的过程抽象机制(通过发消息使用公有成员函数)。对象类是对一组相似对象的抽象，类等级中上层的类是对下层类的抽象。因此，面向对象的环境提供了强有力的抽象机制，便于用户在利用计算机软件系统解决复杂问题时使用习惯的抽象思维工具。此外，面向对象方法学中普遍进行的对象分类过程，支持从特殊到一般的归纳思维过程；面向对象方法学中通过建立类等级而获得的继承特性，支持从一般到特殊的演绎思维过程。

面向对象的软件技术为开发者提供了随着对某个应用系统的认识逐步深入和具体化的过程，而逐步设计和实现该系统的可能性，因为可以先设计出由抽象类构成的系统框架，随

着认识的深入和具体化再逐步派生出更具体的派生类。这样的开发过程符合人们认识客观世界解决复杂问题时逐步深化的渐进过程。

面向对象的软件工程方法包括面向对象的分析(OOA)、面向对象的设计(OOD)、面向对象的编程(OOP)和面向对象的软件维护(OOSM)等内容。在每一个开发阶段,面向对象的方法都要求对系统建立模型,为系统在本阶段的构建提供蓝图。不同阶段的模型包含的内容是不同的,既可以包括详细的计划,也可以包括从很高的层次考虑系统的总体计划。一个好的模型包括那些有广泛影响的主要元素,而忽略那些与给定的抽象水平不相关的次要元素。每个阶段的模型都是一个在语义上闭合的系统抽象。模型可以是结构性的,强调系统的组织,例如系统的静态结构模型;它也可以是行为性的,强调系统的动态方面,例如系统的交互协作模型。

3.1.1 对象和类

面向对象概念是在20世纪60年代末期由使用SIMULA语言的人开始提出的,20世纪70年代初成为Xerox PARC开发的Smalltalk的重要组成部分。与此同时,对软件的开发仍然采用功能分解法来解决设计与实现问题,很少讨论面向对象的设计,更没有对面向对象分析的讨论。但自20世纪80年代以来,面向对象方法与技术日益受到计算机领域的专家、研究和工程技术人员的重视。20世纪80年代中期相继出现了一系列描述能力强、执行效率高的面向对象编程语言,标志着面向对象的方法与技术开始走向实用。自20世纪80年代末到20世纪90年代,面向对象方法与技术向软件生命周期的前期阶段发展,人们对面向对象方法的研究不再局限于编程而是从系统分析和系统设计阶段就开始采用面向对象方法,这标志着面向对象方法已经发展成一种完整的方法论和系统化的技术体系,下面介绍一些面向对象的基本概念。

1. 对象

对象含义广泛,难以精确定义,不同的场合有不同的含义。一般来说,任何事物均可看作对象。任何事物均有各自的自然属性和行为,当考察其某些属性与行为并进行研究时,它便成为有意义的对象。采用面向对象方法进行软件开发时,需要区分三种不同含义的对象:客观对象、问题对象和计算机对象。客观对象是现实世界中存在的实体;问题对象是客观对象在问题域中的抽象,用于根据需要完成某些行为;计算机对象是问题对象在计算机系统中的表示,它是数据和操作的封装体。三种对象间的关系如图3.1所示。

图3.1 三种对象间的关系

对象是理解面向对象技术的关键。可以发现现实世界中的对象具有共同点:它们都有状态和行为。图3.2中的汽车对象有自己的状态(有速度、油量等)及行为(如发动汽车、关闭发动机、刹车和加速等)。对象是封装了数据结构及可以施加在这些数据结构上的操作的封装体,这个封装体可以唯一地标识其名字,而且向外界提供一组服务(即公有的操作)。对

象中的数据表示对象的状态，一个对象的状态只能由该对象的操作来改变。每当需要改变对象的状态时，只能由其他对象向该对象发送消息。对象响应消息时，按照消息模式找出与之匹配的方法，并执行该方法。图 3.2 中的汽车对象，它的状态就只能通过暴露出来的方法来修改。

从上述对象概念的描述中，可以归纳出对象具有如下特点。

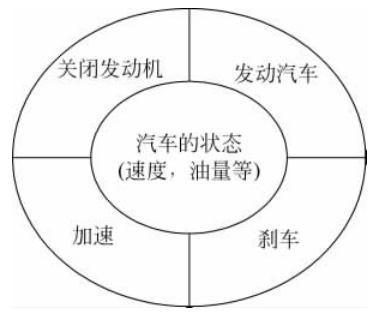

图 3.2　汽车对象

(1) 自治性：对象的自治性是指对象具有一定的独立计算能力。给定一些输入，经过状态转换，对象能产生输出，说明它具有计算能力。对象自身的状态变化是不直接受外界干预的，外界只有通过发送的消息对它产生影响，从这个意义上说，对象具有自治性。

(2) 封闭性：对象的封闭性是指对象具有信息隐蔽的能力。具体说来，外界不直接修改对象的状态，只有通过向该对象发送消息来对它施加影响。对象隐蔽了其中的数据及操作的实现方法，对外可见的只是该对象所提供的操作(即能接收和处理的消息)。

(3) 通信性：对象的通信性是指对象具有与其他对象通信的能力，具体说来，就是对象能接收其他对象发来的消息，同时也能向其他对象发送消息。通信性反映了不同对象间的联系，通过这种联系，若干对象可协同完成某项任务。

(4) 被动性：对象的被动性是指对象的存在和状态转换都是由来自外界的某种刺激引发的。对象的存在可以说是由外界决定的，而对象的状态转换则是在它接收到某种消息后产生的，尽管这种转换实际是由其自身进行的。

(5) 暂存性：对象的暂存性有两层含义。一是指对象的存在是可以动态地引发的，而不是必须在计算的一开始就存在；二是指对象随时可以消亡(而不是必须存在到计算结束)。虽然可以在计算过程中自始至终保存某些对象，但从对象的本质或作用来说，它具有暂存性。

上面 5 个性质分别刻画了对象的不同方面的特点。自治性、封闭性和通信性刻画的是对象的能力；被动性刻画的是对象的活动；暂存性刻画的是对象的生存特性；自治性反映了对象独立计算的能力；封闭性和通信性则说明对象是既封闭又开放的相对独立体。

2. 类

对象是系统中运行时刻的基本成分，它们在程序中又如何反映呢？事实上，系统中往往存在多个具有共同特性的对象。例如，张三、李四、王五、赵六……都具有姓名、身份证号、性别和年龄等特性，而具有姓名、身份证号、性别和年龄等特性的抽象对象便是张三、李四、王五、赵六……对象的一个抽象。在语言中，这样的抽象称为“类”，类刻画了一组具有共同特性的对象。比如，上述张三、李四、王五、赵六……可以抽象为一个具有姓名、身份证号、性别和年龄等特性的“人”的对象，称为类“人”。

类的作用可归纳为两种：一是作为对象的描述机制，刻画一组对象的公共属性和行为；二是作为程序的基本单位，它是支持模块化设计的设施，并且类上的分类关系是模块划分的规范标准。

与对象的组成部分相对应，类也有三个组成部分：数据、操作和接口。数据刻画对象的状态，操作刻画对象的行为，类中所有数据均为私有，接口使操作对外可见。从类自身的内容看，它描述了一组数据及其上的操作，这些数据为类所私有，只有操作对外可见。

类的概念可从下面4个方面去理解。

(1) 类是对对象的进一步抽象。由现实世界中的张三、李四、王五、赵六……都具有姓名、身份证号、性别和年龄等特性我们获得一个抽象的概念——对象，每一个对象都有自己的特性。描述这些对象的共同部分就是对这些对象的进一步抽象，由此得到类。类是静态概念，对象是动态概念。

(2) 类描写了一组相似对象的共同特性。面向对象程序执行时体现为一组对象状态的变化，这里的对象便是由类来刻画的。类刻画了一组具有相似特性的对象，在运行时可根据类中的描述动态地创建对象。

(3) 类既可以与具体的程序设计语言无关，也可以与具体的程序设计语言相关。与具体的程序设计语言无关的类称为分析类；与具体的程序设计语言相关的类称为设计类或实现类。

(4) 类的概念尽管来源于程序设计语言，但不限于程序设计语言。随着面向对象方法与技术向软件生命周期的前期阶段的延伸，在软件开发的各个阶段都不同程度地会涉及类的概念。

3.1.2 消息与方法

1. 消息

上面介绍了对象是一个相对独立的具有一定计算能力的自治体，对象之间不是彼此孤立而是互相通信的，面向对象程序的执行体现为一组相互通信的对象的活动。那么面向对象程序是如何实施计算(运行)的呢？计算是由一组地位等同的称作对象的计算机制合作完成的，合作方式是通信即相互交换信息，这种对象与对象之间所互相传递的信息称为消息。消息可以表示计算任务，也可以表示计算结果。

在面向对象计算中，每一项计算任务都表示为一个消息，实施计算任务的若干相关联的对象组成一个面向对象系统。提交计算任务即由任务提交者(系统外对象)向承担计算任务的面向对象系统中的某对象发送表示该计算任务的消息。计算的实施过程是面向对象系统接收到该消息后所产生的状态变化过程，计算的结果通过面向对象系统中的对象向任务提交者回送。

消息一般由以下三个部分组成。

(1) 接收消息的对象。

(2) 接收对象应采用的方法。

(3) 方法所需要的参数。

计算任务通常先由某一对象“受理”(该对象接收到某种消息)，然后通过对象间的通信，计算任务就分散到各个有关对象中，最后再由某些对象给出结果(通过发送消息)。发送消息的对象称为发送者，接收消息的对象称为接收者。消息中包含发送者的要求，它告诉接收者需要完成哪些处理，但并不指示接收者如何完成这些处理。消息完全由接收者解析，接收

者独立决定采用什么方式完成所需处理。一个对象能够接收不同形式、不同内容的多个消息，相同形式的消息可以发往不同的对象，不同的对象对于形式相同的消息可以有不同的解析，并做出不同的反应。对于传来的消息，对象可以返回相应的应答消息。

对象可以动态地创建，创建后即可以活动。对象在不同时刻可处于不同状态，对象的活动是指对象状态的改变，它是由对象所接收的消息引发的。对象一经创建，就能接收消息，并向其他对象发送消息。对象接收到消息后，可能出现：①自身状态改变；②创建新对象；③向其他对象发送消息。

从对象之间的消息通信机制可反映出面向对象计算具有如下特性。

(1) 协同性。协同性表现在计算是由若干对象共同协作完成的。虽然计算任务可能首先由面向对象系统中的某个特定对象“受理”(即接收到表示该任务的消息)，但往往并不是由该对象独立完成的，而是通过对象间的通信被分解到其他有关对象中，由这些对象共同完成，对象间的这种协同性使计算具有分布性。

(2) 动态性。动态性表现在计算过程中对象依通信关系组成的结构会动态地改变，新对象会不断创建，老对象也会不断消亡。面向对象系统最初由若干初始对象组成，一旦外界向这些初始对象发送了表示计算任务的消息，面向对象系统即活动起来直至给出计算结果。在此过程中，面向对象系统的组成因创建新对象而不断地改变。

(3) 封闭性。封闭性表现在计算是由一组相对封闭的对象完成的。从外界看一个对象，只是一个能接收和发送消息的机制，其内部的状态及其如何变化对外并不直接可见，外界只有通过给它发送消息才能对它产生影响。对象承担计算的能力完全通过它能接收和处理的消息体现。

(4) 自治性。自治性表现在计算是由一组自治的对象完成的。对象在接收了消息后，如何处理该消息(即自身状态如何改变，需创建哪些新对象，以及向其他对象发送什么消息)，完全由该对象自身决定。在面向对象计算中，数据与其上的操作地位同等，两者紧密耦合在一起形成对象，亦即数据及其上的操作构成对象。因此，在面向对象计算中，数据与其上的操作之间的联系处于首要地位，何时对何数据施行何种操作完全由相应数据所在对象所接收到的消息及该对象自身决定。由于对象的封装性和隐蔽性，对象的消息仅作用于对象的接口，通过接口进一步影响和改变对象状态。

2. 方法

方法反映对象的行为，是对象固有的动态表示，可审视并改变对象的内部状态。一个对象往往可以用若干方法表示其动态行为，在计算机中，方法也可称为操作。它的定义与表示包含两部分：一是方法的接口，它给出了方法的外部表示，包括方法的名称、参数及结果类型；二是方法的实现，它用一段程序代码表示，这段代码实现了方法的功能。

把所有对象抽象成各种类，每个类都定义一组方法，代表允许作用于该类对象上的各种操作。方法描述了对象执行操作的算法，响应消息的方法。

3.1.3 面向对象的要素

1. 继承性

类之间的继承关系是现实世界中遗传关系的直接模拟，它表示类之间的内在联系以及

对属性和操作的共享。继承是类与类之间的一种关系，它使程序人员可以在已有类的基础上定义和实现新类。继承是实现利用可重用软件构件构造系统的有效语言机制。

继承能有效地支持软件构件的重用，使得当需要在系统中增加新特征时所需的新代码最少，并且当继承和多态动态绑定结合使用时，为修改系统所需变动的源代码最少。

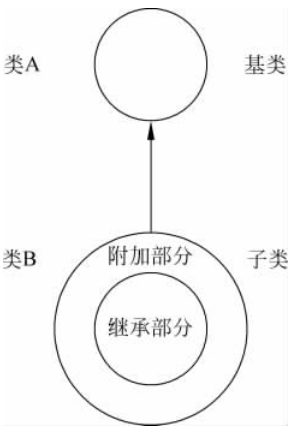

图 3.3　继承描述

继承按照子类与父类的关系，把若干个类组成一个类层次结构。在这种层次结构中，通常下层的派生类具有和上层的基类相同的特性(包括数据和方法)。继承是子类自动地共享父类中定义的数据和方法的机制，一个直观的描述如图 3.3 所示。

图 3.3 中描述了类 A 和类 B 之间的继承关系。类 B 继承了类 A，因此它包含类 A 中的所有数据结构和方法。同时，类 B 也定义了自己的附加内容，形成了自己的、不同于父类的数据结构和方法，这是类 B 对父类的增量定义。

当一个类只允许有一个父类时，类的继承是单继承；当允许一个类有多个父类时，类的继承是多重继承。

判断两个类之间是否具有继承关系的一个重要准则是替换原理：凡是父类可以胜任的场合，子类也一定可以胜任。

再举一个常用的例子是学校组成人员之间的关系，本科生是一类特殊的学生，这一关系很容易用如图 3.4 所示的继承关系来刻画。

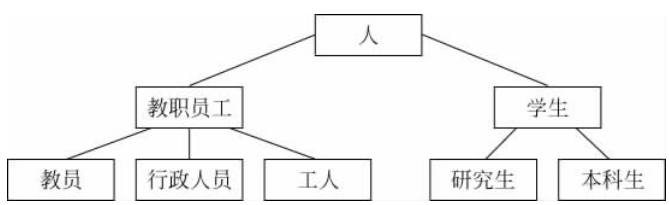

图 3.4　学校组成人员间的继承关系

作为学生的后继，本科生具有学生的所有特征。另外，由于学生是人的后继，本科生也继承了人的特征。在最高抽象层上，所有学校组成人员都具有身份号、姓名、年龄和性别等属性，也有一些相同的操作，这些特征在根类人中定义。学生类增加了成绩、课程等属性。由于继承的关系，因此本科生也具有身份号、姓名、年龄、性别、成绩、课程等属性。继承的使用大大地减少了添加新特征所需的工作，同样在面向对象开发过程中，继承也有助于使用原型速成开发方法。

继承机制的强有力之处还在于它允许程序员重用一个未必完全符合要求的类，允许对该类进行修改而又不至于在该类的其他部分引起副作用。另外，在面向对象设计中，还注重在较高抽象层次上提取和封装公共性质。如果把这些高层次的类保存在类库中，那么，对于用户所需要的类，在类库中都可能有一个更一般的类与之对应。

如果一个软件系统是用面向对象方法进行分析和设计，并用面向对象程序设计语言实现的，那么在分析阶段所识别的对象和分类关系就能在设计阶段得到保留和丰富，而且可以直接用代码实现，这种直接用代码刻画和封装抽象结构的能力，代表了软件技术上的进步。

2. 抽象

抽象同样是一种人类认识客观世界的方式。为了记忆或区分，人类常常把客观世界的一些事物的基本特征、内在的属性概念化，用逻辑模型表达出来，这样的过程就是抽象。例如，在世界地图上可以用一个点代表城市，用曲线来表示河流等，它们忽略了城市的大小和河流的宽度，却保留了它们的主要信息——位置和长度。通过抽象，达到了简化表达、突出重点的目的。面向对象提供的抽象表达能力，符合人类认识世界的规律，因而它是对面向过程的进化。可以说软件工程的发展历史就是人们不断追求更高水平的抽象、封装和模块化的历史。

3. 封装性

对象不仅包含数据，同时也包含操作这些数据的方法。对象是进行数据处理的主体，必须发消息请求对象执行它的某个操作，处理它的私有数据，而不能从外界直接对它的私有数据进行操作。也就是说，一切从属于该对象的私有信息，都被封装在该对象类的定义中，就好像装在一个不透明的黑盒子中一样。

封装性和信息的隐蔽性是联系在一起的。软件的内部构件都是具有良好外部边界的。使其隐藏不应该被外部直接访问的类成员，它们在外部是不可见的。而只能通过类提供的方法或外部接口来进行访问，从而将外部接口和内部实现分开。面向对象方法鼓励隐藏信息，除了需要共享的结构和方法外，都要使用私有成员。这样使类的接口尽可能简单，减少类之间的相互依赖，从而达到高内聚，低耦合。

封装的基本单位是对象。例如，一个咖啡机被封装起来，使用者无法看到咖啡机内部的工作细节。使用者只需按下代表不同口味的咖啡的按钮，就可以喝到咖啡了。封装将使用者和设计者分开，只需要使用对象暴露出来的方法就可以获得相应的功能。这样大大提高了软件的可维护性。

4. 多态性

多态的一般含义是，某一领域中的元素可以有多种解释，程序设计语言中的“一名多用”即是支持多态的设施。继承机制是面向对象程序设计语言中所特有的另一种支持多态的设施。

在面向对象的软件技术中，多态是指在类继承层次中的类可以共享一个行为的名字，而不同层次的类却各自按自己的需要实现这个行为。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地选择在该类中定义的行为实现。

在面向对象程序设计语言中，一个多态的对象指引变量可以在不同的时刻，指向不同类的实例。由于多态对象指引变量可以指向多类对象，所以它既有一个静态类型又有多个动态类型。多态对象指引变量的动态类型在程序执行中会不时地改变，在强类型的面向对象环境中，运行系统自动地为所有多态对象指引变量标记其动态类型。多态对象指引变量的静态类型由程序正文中的变量说明决定，它可以在编译时确定，它规定了运行时刻可接受的有效对象类型的集合，这种规定是通过对系统的继承关系图进行分析得到的。

在强类型的面向对象程序设计语言中，继承所反映的概念包含关系和多态的思想密切

有关。如果类 Y 是类 Z 的后继，则概念上 Z 包含 Y，那么在所有期望 Z 的实例的场合，都允许用 Y 的实例来代替。多态的特点在很大程度上提高了程序的抽象程度和简洁性，最大限度地降低了类和程序模块间的耦合性，提高了类模块的封闭性，使得它们不需了解对方的具体细节就可以很好地共同工作，这对程序的设计、开发和维护都有很大的益处。

5. 关联

在现实世界中，事物不是孤立的，而是彼此之间存在各种各样的联系。关联描述了系统中对象之间的离散连接。例如，在一个学校中，有教师、学生和教室等事物，它们之间存在某种特定的联系。关联在一个含有两个或多个对象的序列中建立联系，序列中的对象允许重复。

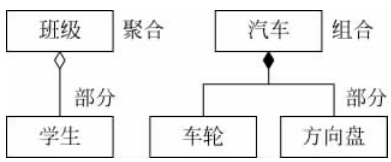

图 3.5　聚合和组合关联

关联有两种比较特殊的类型：聚合和组合。聚合表示部分与整体关系的关联。组合是更强形式的关联，整体有管理部分的职责，比如为它们分配和释放空间。一个对象最多属于一个组合关系。表示部分的类与表示整体的类之间有单独的关联，但为方便起见，可以把连线结合在一起，如图 3.5 所示表示了聚合和组合关联。

3.2　面向对象建模

众所周知，在解决问题之前必须首先理解所要解决的问题。软件开发项目失败的一个主要原因是对所要解决的问题理解不够，或者说没有很好地识别用户需求。

在对目标系统进行分析的初始阶段，面对大量模糊的、涉及众多专业领域的、错综复杂的信息，系统分析员往往感到无从下手。模型提供了组织大量信息的一种有效机制。用面向对象方法成功地开发软件的关键，同样是对问题域的理解。面向对象方法最基本的原则，是按照人们习惯的思维方式，用面向对象观点建立问题域的模型，便于参入软件开发的各方交流对问题的理解，对所要完成的软件产品规格达成共识。

3.2.1　按模型的用途对模型分类

用面向对象方法开发软件，通常需要建立三种形式的模型，它们分别是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和描述系统功能的功能模型。这三种模型都涉及数据、控制和操作等共同概念，只不过每种模型描述的侧重点不同。这三种模型从三个不同但又密切相关的角度模拟目标系统，它们各自从不同侧面反映了系统的实质性内容，综合起来则全面地反映了对目标系统的需求。一个典型的软件系统组合了上述三方面内容：它使用数据结构(对象模型)，执行操作(动态模型)，并且完成数据值的变化(功能模型)。为了全面地理解问题域，对任何大系统来说，上述三种模型都是必不可少的。当然，在不同的应用问题中，这三种模型的相对重要程度会有所不同。但是，用面向对象方法开发软件，在任何情况下，对象模型始终都是最重要、最基本、最核心的。在整个开发过程中，三种模型一

直都在发展、完善：在面向对象分析过程中，构造出完全独立于实现的应用域模型；在面向对象设计过程中，把求解域的结构逐渐加入到模型中；在实现阶段，把应用域和求解域的结构都编成程序代码并进行严格的测试验证。

(1) 对象模型。对象模型表示静态的、结构化的系统的“数据”性质。它是对模拟客观世界实体的对象以及对象彼此间的关系的映射，描述了系统的静态结构。面向对象方法强调围绕对象而不是围绕功能来构造系统。对象模型为建立动态模型和功能模型提供了实质性的框架。在建立对象模型时，人们的目标是从客观世界中提炼出对具体应用有价值的概念。

为了建立对象模型，需要定义一组图形符号，并且规定组织这些符号以表示特定语义的规则。也就是说，需要用适当的建模语言来表达模型，建模语言由记号(即模型中使用的符号)和使用记号的规则(语法、语义和语用)组成。

一些著名的软件工程专家在提出自己的面向对象方法的同时，也提出了自己的建模语言。但是，面向对象方法的用户并不了解不同建模语言的优缺点，很难在实际工作中根据应用的特点选择合适的建模语言，而且不同建模语言之间存在的细微差别也极大地妨碍了用户之间的交流。面向对象方法发展的现实，要求在精心比较不同建模语言的优缺点和总结面向对象技术应用经验的基础上，把建模语言统一起来。

(2) 动态模型。动态模型表示瞬时的、行为化的系统的“控制”性质，它规定了对象模型中的对象的合法变化序列。

一旦建立起对象模型之后，就需要考察对象的动态行为。所有对象都具有自己的生命周期(或称为运行周期)。对一个对象来说，生命周期由许多阶段组成，在每个特定阶段中，都有适合该对象的一组运行规律和行为规则，用以规范该对象的行为。生命周期中的阶段也就是对象的状态。所谓状态，是对对象属性值的一种抽象。当然，在定义状态时应该忽略那些不影响对象行为的属性。各对象之间相互触发(即作用)就形成了一系列的状态变化，人们把一个触发行为称作一个事件。对象对事件的响应，取决于接收该触发的对象当时所处的状态，响应包括改变自己的状态或者又形成一个新的触发行为。

状态有持续性，它占用一段时间间隔。状态与事件密不可分，一个事件分开两个状态，一个状态隔开两个事件。事件表示时刻，状态代表时间间隔。通常，用UML提供的状态图来描绘对象的状态、触发状态转换的事件以及对象的行为(对事件的响应)。每个类的动态行为用一张状态图来描绘，各个类的状态图通过共享事件合并起来，从而构成系统的动态模型。也就是说，动态模型是基于事件共享而互相关联的一组状态图的集合。

(3) 功能模型。功能模型表示变化的系统的“功能”性质，它指明了系统应该“做什么”，因此更直接地反映了用户对目标系统的需求。

软件工程早期，功能模型通常由一组数据流图组成。在面向对象方法学中，数据流图远不如在结构化分析设计方法中那样重要。一般说来，与对象模型和动态模型比较起来，数据流图并没有增加新的信息。但是，建立功能模型有助于软件开发人员更深入地理解问题域，改进和完善自己的设计，因此，不能完全忽视功能模型的作用。通常，软件系统的用户数量庞大(或用户的类型很多)，每个用户只知道自己如何使用系统，但是没有人准确地知道系统的整体运行情况。因此，使用用例模型代替传统的功能说明，往往能够更好地获取用户需求，它所回答的问题是“系统应该为每个(或每类)用户做什么”。

3.2.2 按软件开发过程对模型分类

功能模型、对象模型、动态模型是按模型的用途对模型的分类。在软件开发过程中，在不同阶段需要建立不同的模型，并且这些模型的目的是不同的。模型按软件开发的阶段性可以分为以下6种。

(1) 业务模型：展示业务过程、业务内容和业务规则的模型。参加创建业务模型的主要人员有领域专家和需求分析师，业务模型通常用对象模型表示。

(2) 需求模型：展示用户要求和业务要求的模型。参入创建需求模型的主要人员有需求分析师和系统设计师。需求模型通常由用例模型表示(用例模型主要由用例图构成)。

(3) 数据库模型：指物理数据模型，该模型是可选的。数据库应用软件需要创建数据库模型。以数据库设计人员为主，架构师提供指导，资深开发(设计)人员予以配合，共同设计该模型。

(4) 分析模型：系统分析是系统实现的第一步，其目的是在比较高的抽象层次上帮助理清需求和设计。分析模型和设计模型关心的都是系统是如何被实现的，但是它们的侧重点不同。

(5) 设计模型：包含架构模型和详细设计模型。架构模型展示软件系统的宏观结构；详细设计模型展示软件系统的微观组成。架构师设计架构模型(架构模型通常由包图组成)。详细设计模型则以资深开发人员为主，架构师提供指导，共同设计(详细设计模型通常用对象模型表示)。

(6) 实现模型：描述了软件组件(该组件能够运行)及其关系(通常由构件图或部署图组成)。以资深开发人员(设计人员)为主，架构师提供总体指导共同设计完成该模型。

3.2.3 IBM RSA面向对象建模的主要步骤

利用IBM Rational Software Architect对软件系统建模时，采用的就是以上的模型分类方法。基于IBM RSA的面向对象建模过程一般遵循如下主要步骤。

(1) 识别系统的用例和角色。

在这一阶段，可以利用RSA建立系统的需求模型。首先对项目进行需求调研，依据项目的业务流程图和数据流程图以及项目中涉及的各级操作人员，通过分析，识别出系统中的所有用例和角色；接着分析系统中各角色和用例间的联系。在分析模型中需要包括系统的用例图以及用户与系统交互的活动图。它们从用户的角度分别描述了系统的功能和使用流程。借助需求模型可以解决“做什么”的问题。

(2) 进行系统分析，并抽象出类。

在这一阶段，可以利用RSA建立分析模型。系统分析的任务是找出系统中所有需求并加以描述，同时为每一个用例提供相应的用例实现。通过对系统的深入分析，要从中抽取出实体类、边界类和控制类，并描述这些类之间的关系。

(3) 设计系统和系统中的类及其行为。

这个阶段需要建立的模型是设计模型。设计阶段由结构设计和详细设计组成。结构设计是高层设计，其任务是定义包(子系统)，包括包间的依赖关系和主要通信机制。包有利于

描述系统的逻辑组成部分以及各部分之间的依赖关系。详细设计就是要细化包的内容，清晰描述所有的类，包括每个类的属性和方法的定义。经过这个阶段，就可以使用合适的面向对象语言来实现系统了。

UML的应用贯穿在系统开发的每个阶段。

(1) 需求分析：UML的用例视图可以表示客户的需求。通过用例建模可以对外部的角色以及它们所需要的系统功能建模。角色和用例是用它们之间的关系、通信建模的。每个用例都指定了客户的需求。不仅要对软件系统进行分析，对商业过程也要进行需求分析。

(2) 分析：分析阶段主要考虑所要解决的问题，可用UML的逻辑视图和动态视图来描述。类图描述系统的静态结构，协作图、状态图、序列图、活动图描述系统的动态特征。在分析阶段只为问题领域的类建模，不定义软件系统的解决方案的细节，如用户接口的类、数据库等。

(3) 设计：在设计阶段把分析阶段的结果扩展成技术解决方案。加入新的类来提供技术基础结构——用户接口、数据库操作等。分析阶段的领域问题类被嵌入在这个技术基础结构中，设计阶段的结果是构造阶段的详细的规格说明。

(4) 构造：在构造或程序设计阶段，把设计阶段的类转换成某种面向对象程序设计语言的代码。在对UML表示的分析和设计模型进行转换时，最好不要直接把模型转化成代码。因为在早期阶段，模型是理解系统并对系统进行结构化的手段。

(5) 测试：对系统的测试通常分为单元测试、集成测试、系统测试和验收测试几个不同级别。单元测试是对几个类或一组类的测试，通常由程序员进行；集成测试集成组件和类，确认它们之间是否恰当地协作。系统测试把系统当作一个黑箱，验证系统是否具有用户所要求的所有功能。验收测试由客户完成，与系统测试类似，验证系统是否满足所有的需求。不同的测试小组使用不同的UML图作为他们工作的基础，单元测试使用类图和类的规格说明，集成测试典型地使用组件图和协作图，而系统测试使用用例图来确认系统的行为符合这些图中的定义。

3.3 面向对象的分析与设计

在20世纪60年代末期出现面向对象程序设计的同时，业界正在沿着诸如COBOL、FORTRAN之类的语言蹒跚而行，并用功能分解来解决设计与实现问题，极少讨论到面向对象的设计，更不用说面向对象的分析了。过去几十年软件发展的4个重大变化，是促进面向对象分析与设计方法迅速发展的重要因素。

(1) 软件领域中面向对象方法的基本概念经历了几十年的成长道路，人们的注意力逐渐从编码问题转移到设计与分析问题。

(2) 构造系统的基本技术变得更加有力，设计思想受预想的如何编码的思想影响，而编码思想受人们可用的程序设计语言的强烈影响。当选用的语言是汇编语言和FORTRAN语言时考虑结构化程序设计是很困难的。使用Pascal、PL/1和ALGOL时，结构化程序设计就容易得多。类似地，当选用COBOL或C语言进行面向对象的程序设计时也是比较困难的，而用C++则比较容易。

(3) 现代软件系统规模更大、更复杂也更多变，传统的软件分析与设计方法难以满足要

求，而面向对象的分析与设计方法将实现比较稳定的系统。另外，现代软件系统更注重系统用户界面的开发，对于此类系统采用面向对象方法进行分析、设计和编码是一种非常自然的途径。

(4) 现代软件系统构造比20世纪70年代和80年代更加面向领域，对功能复杂性的关心比以前少，数据建模的优先程度较为适当，问题域模型的理解及系统职能处于较高的优先地位。

尽管面向对象语言取得了令人振奋的发展，但编程并不是软件开发项目失败的主要根源，需求分析与设计问题更为普遍并且更值得解决。因此，面向对象开发技术的焦点不应该只注重编程阶段，而应更全面地考虑软件工程的其他阶段。面向对象方法真正意义和深远的目标是它适合于解决分析与设计期间的复杂性并实现分析与设计结果的重用。面向对象的开发不仅仅是编程，必须在整个软件生命周期采用一种全新的方法，这一观点已被人们所接受。

3.3.1 面向对象分析

面向对象分析，就是抽取和整理用户需求并建立问题域模型的过程。面向对象分析的关键是识别出问题域内的对象，并分析它们之间的关系。面向对象分析的目的是认知客观世界的系统。面向对象分析的工作成果是系统的分析模型。分析模型的作用有两个方面：一是用于在用例模型的基础上进一步明确问题域的需求；二是为参加软件开发的各方提供一个协商的基础。面向对象分析是面向对象方法从编程领域向分析领域发展的产物。通过面向对象分析可以加强开发人员对问题域的理解，改善与软件开发有关的各类人员的交流。

面向对象分析以用例模型作为输入，将得到两种输出模型：对象模型和功能模型。对象模型把系统分解成相互协作的分析类，通过类图、对象图描述对象、对象的属性和对象间的关系。功能模型从用户的角度描述系统提供的服务，也就是描述系统的动态行为。通过时序图、协作图描述对象的交互，揭示对象间如何协作完成每个具体的用例。单个对象的状态变化或动态行为可以通过状态图表达。

3.3.2 面向对象设计

如前所述，分析是提取和整理用户需求，并建立问题域模型的过程；设计则是把分析阶段得到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。从面向对象分析到面向对象设计，是一个逐渐扩充模型的过程。或者说，面向对象设计就是用面向对象观点建立求解域模型的过程。

尽管分析和设计的定义有明显区别，但是在实际的软件开发过程中二者的界限是很模糊的。许多分析结果可以直接映射成设计结果，而在设计过程中又往往会加深和补充对系统需求的理解，从而进一步完善分析结果。因此，分析和设计活动是一个多次反复迭代的过程。面向对象方法学在概念和表示方法上的一致性，保证了在各项开发活动之间的平滑(无缝)过渡，领域专家和开发人员能够比较容易地跟踪整个系统开发过程，这是面向对象方法与传统方法比较起来所具有的一大优势。

生命周期方法学把设计进一步划分成总体设计和详细设计两个阶段，类似地，也可以把

面向对象设计再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。系统设计与对象设计之间的界限,比分析与设计之间的界限更模糊。

进行面向对象设计时需要遵循下面一些基本准则。

(1) 模块化。面向对象软件开发模式,很自然地支持把系统分解成模块的设计原理,对象就是模块。它是把数据结构和操作这些数据的方法紧密地结合在一起所构成的模块。

(2) 抽象。面向对象方法不仅支持过程抽象,而且支持数据抽象。类实际上是一种抽象数据类型,它对外开放的公共接口构成了类的规格说明(即协议),这种接口规定了外界可以使用的合法操作符,利用这些操作符可以对类实例中包含的数据进行操作。使用者无须知道这些操作符的实现算法和类中数据元素的具体表示方法,就可以通过这些操作符使用类中定义的数据。通常把这类抽象称为规格说明抽象。

此外,某些面向对象的程序设计语言还支持参数化抽象。所谓参数化抽象,是指当描述类的规格说明时并不具体指定所要操作的数据类型,而是把数据类型作为参数。这使得类的抽象程度更高,应用范围更广,可重用性更高。例如,C++语言提供的“模板”机制就是一种参数化抽象机制。

(3) 信息隐藏。在面向对象方法中,信息隐藏通过对象的封装性实现:类结构分离了接口与实现,从而支持信息隐藏。对于类的用户来说,属性的表示方法和操作的实现算法都应该是隐藏的。

(4) 低耦合。耦合是指一个软件结构内不同模块之间互连的紧密程度。在面向对象方法中,对象是最基本的模块,因此,耦合主要指不同对象之间相互关联的紧密程度。低耦合是优秀设计的一个重要标准,因为这有助于使系统中某一部分的变化对其他部分的影响降到最低程度。在理想情况下,对某一部分的理解、测试或修改,无须涉及系统的其他部分。

如果一个类对象过多地依赖其他类对象来完成自己的工作,则不仅给理解、测试或修改这个类带来很大困难,而且还将大大降低该类的可重用性和可移植性。显然,类之间的这种相互依赖关系是紧耦合的。当然,对象不可能是完全孤立的,当两个对象必须相互联系相互依赖时,应该通过类的协议(即公共接口)实现耦合,而不应该依赖于类的具体实现细节。

(5) 高内聚。内聚衡量一个模块内各个元素彼此结合的紧密程度。也可以把内聚定义为:设计中使用的一个构件内的各个元素,对完成一个定义明确的目的所做出的贡献程度。在设计时应该力求做到“低耦合、高内聚”。

(6) 可重用。软件重用是提高软件开发生产率和目标系统质量的重要途径。重用基本上从设计阶段开始,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,及以往开发类似系统时创建的类),二是如果确实需要创建新类,则在设计这些新类的协议时,应该考虑将来的可重复使用性。

面向对象设计的具体内容主要有4个方面,即问题域设计、人机接口设计、任务管理设计和数据管理设计。

(1) 问题域的设计:通过面向对象分析得到的问题域模型,为设计问题域的解决方案打下了良好基础,建立了完整框架。要尽可能地保持面向对象分析所建立的问题域结构。从实现角度对问题域模型做进一步补充或修改,增添、合并或分解类与对象、属性,调整继承

关系。当问题域过分复杂庞大时，应该把它进一步分解成若干较小的子系统。设计在分析和实现之间充当桥梁作用，而问题域设计是关键。问题域设计的任务是确定软件系统的架构，尽量保持问题域设计成果的稳定性。设计的变与不变是相对的，只要系统的架构不变，就不会对软件开发进度产生大的影响，确保按时提交符合质量要求的软件产品。

(2) 人机界面的设计：人机界面突出人如何命令系统以及系统如何向用户提交信息，人机界面构建是在问题域的上下文内实现，接口本身表示大多数现代应用的一个特别重要的子系统。面向对象分析模型中包含的使用情景、用户与系统交互时扮演角色的描述等，这些都作为人机界面设计过程的输入。

(3) 任务管理的设计：任务是进程的别称，若干任务的并发执行叫多任务。对一些应用，任务能简化总体设计和代码。独立的任务把必须并发进行的行为分离开，这种并发行为可以在多个独立的处理机上模拟。任务特征的确定可通过了解任务是如何开始而实现，事件驱动和时钟驱动是最为常见的情况，两者都是通过中断激活，但前者接收来自外部资源的中断，而后者则由系统时钟控制。除了任务的开始方式外，还需确定任务的优先级别和临界状态，高优先级别的任务必须具有立即存取系统资源的能力，高临界状态的任务甚至在资源有效性减少或系统处在一个低能操作状态下仍必须继续操作。在确定任务特征后，定义与其他任务协同和通信所需的对象属性和方法。

(4) 数据管理的设计：数据管理部分提供了在数据管理系统中存储和检索对象的基本结构。数据管理包括对两个不同区域的考虑：一个是对应用本身至关重要数据的管理，另一个是建立对象存储和检索的基础。通常数据管理采用分层设计的方式，思想是从处理系统属性的高级需求中分离出操纵数据结构的低级需求。通常有三种主要的数据管理方法：普通文件、关系型数据库管理系统和面向对象数据库管理系统。

从方法学上看，由于面向对象的分析和设计方法按适合人们的思考方式进行系统的分析与设计，使得从面向对象的分析到设计不存在概念和表达的转换问题。面向对象的分析与设计都是基于相同的面向对象基本概念，因此从面向对象的分析到面向对象的设计是一个累进的模型扩充过程。

3.4 面向对象实现

面向对象实现就是用某种面向对象程序设计语言把面向对象设计成果转化为用户可使用的软件产品。

从1951年到2014年，人类一共发明了256种编程语言，每一种语言的出现都带有某些新特征。1967年诞生的第一个面向对象语言Simula 67，是面向对象程序设计(Object Oriented Programming，OOP)的鼻祖，它提出了对象的概念并且支持类和继承。随后相继出现了Smalltalk、C++、Java、C#等面向对象的编程语言。C++、Java对面向对象的程序设计思想的传播起到了非常重要的作用。C#和ASP.NET、PHP、JavaScript、Python、Ruby、Groovy、Go等这些编程语言都声称支持面向对象编程。

每一种语言都有它的特点和不足，没有一种语言是万能的。本书不打算比较各种语言的孰优孰劣，仅给出几个常用语言的应用领域。

3.4.1 C++

C++的应用领域主要集中在以下几个方面。

游戏：C++具有超高效率，而且近年来C++凭借先进的数值计算库、泛型编程等优势，在游戏领域应用很多。目前，除了一些网页游戏，很多游戏软件客户端都是使用C++开发的。

网络软件：C++拥有很多成熟的用于网络通信的库，其中最具有代表性的是跨平台的、重量级的ACE库，该库可以说是C++语言最重要的成果之一，在许多重要的企业、部门甚至是军方都有应用。

数字图像处理：这个也是发展很快的计算机领域，目前各种数字地球、数字城市、虚拟地理环境等方面，出现了大量的数字图像处理，C++在数字图像处理编程中具有很大的优势。

科学计算：在科学计算领域，FORTRAN是使用最多的语言之一。但是近年来，C++凭借先进的数值计算库、泛型编程等优势在这一领域也应用颇多。

嵌入式系统：因为C++具有很高的效率，而且保持对C语言的兼容性，能使底层平台有很高的效率，同时具有很大的灵活性，使得它在底层开发中具有优势。

系统级软件：例如，操作系统。在该领域，C语言是主要使用的编程语言。但是C++凭借其对C的兼容性、面向对象性质，也开始在该领域崭露头角。

C++的应用范围很广，在某些对硬件、操作系统或速度有要求的应用中，C++仍是首选。

3.4.2 Java

Java的应用领域很广，它包含三个版本：Java ME、Java SE和Java EE。这三个版本对应于Java应用的三大领域。

Java ME：是一种高度优化的Java运行环境，主要针对消费类电子设备，例如蜂窝电话和可视电话、数字机顶盒、汽车导航系统等。是为机顶盒、移动电话和PDA之类嵌入式消费电子设备提供的Java语言平台，包括虚拟机和一系列标准化的Java API。Java ME技术在1999年的JavaOne Developer Conference大会上正式推出，它将Java语言的与平台无关的特性移植到小型电子设备上，允许移动无线设备之间共享应用程序。今天，不止是桌面上的计算机，手中的电话、汽车中的通信设备、家中的冰箱、洗衣机等都将连入互联网，这是一个移动的互联网。J2ME(Java2平台微型版)就是Java程序在这些连接设备上的执行平台和开发环境，其基本思想和J2SE类似，就是在各种设备上安装适合它的Java虚拟机，应用程序则在虚拟机之上运行。

Java SE：用于开发和部署桌面、服务器以及嵌入设备和实时环境中的Java应用程序。Java SE包括用于开发Java Web服务的类库，同时，Java SE为Java EE提供了基础。Java SE是Java平台的核心。

Java EE：用于开发和部署可移植、健壮、可伸缩且安全的服务器端Java应用程序。Java EE是在Java SE的基础上构建的，它提供Web服务、组件模型、管理和通信API，可以用来实现企业级的面向服务体系结构(Service-Oriented Architecture，SOA)和Web 2.0应用程序。目前流行的企业级B/S(Browser/Server，浏览/服务器)架构的应用主要是用Java EE/JSP和C#/ASP.NET开发的。

3.4.3 C#和ASP.NET

C#是微软公司发布的一种面向对象的、运行于.NET Framework之上的高级程序设计语言。C#看起来与Java有着惊人的相似；它包括诸如单一继承、接口、与Java几乎同样的语法和编译成中间代码再运行的过程。但是C#与Java有着明显的不同，它借鉴了Delphi的一个特点，与COM(组件对象模型)是直接集成的，而且它是微软公司.NET Windows网络框架的主角。C#主要用于桌面应用程序的开发，它和ASP.NET一起，就像Java和JSP一样，是Web应用程序开发的主力。

3.4.4 PHP

PHP(Hypertext Preprocessor，超文本预处理器)是一种通用开源脚本语言。PHP语法吸收了C语言、Java和Perl的特点，易于学习，使用广泛，主要适用于Web应用开发领域。PHP独特的语法混合了C、Java、Perl以及PHP自创的语法。它可以比CGI或者Perl更快速地执行动态网页。用PHP制作出的动态页面与其他的编程语言相比，PHP是将程序嵌入到HTML(标准通用标记语言下的一个应用)文档中去执行，执行效率比完全生成HTML标记的CGI要高许多；PHP还可以执行编译后代码，编译可以达到加密和优化代码运行，使代码运行更快。

3.4.5 JavaScript

JavaScript是一种直译式脚本语言，是一种动态类型、弱类型、基于原型的语言，内置支持类型。它的解释器被称为JavaScript引擎，为浏览器的一部分，广泛用于客户端的脚本语言，最早是在HTML(标准通用标记语言下的一个应用)网页上使用，用来给HTML网页增加动态功能。主要应用于Web应用开发领域。

3.4.6 Python

Python是一种面向对象、解释型计算机程序设计语言，由Guido van Rossum于1989年发明，第一个公开发行版发行于1991年。Python具有丰富和强大的库。它常被昵称为胶水语言，能够把用其他语言制作的各种模块(尤其是C/C++)很轻松地连接在一起。常见的一种应用情形是，使用Python快速生成程序的原型(有时甚至是程序的最终界面)，然后对其中有特别要求的部分，用更合适的语言改写，比如3D游戏中的图形渲染模块，性能要求特别高，就可以用C/C++重写，而后封装为Python可以调用的扩展类库。

3.4.7 Ruby

Ruby是一种简单快捷的面向对象脚本语言，主要应用于Web应用开发领域。

3.4.8 Groovy

Groovy是一种基于JVM(Java虚拟机)的敏捷开发语言，它结合了Python、Ruby和

Smalltalk的许多强大的特性，Groovy代码能够与Java代码很好地结合，也能用于扩展现有代码。由于其运行在JVM上的特性，Groovy可以使用其他Java语言编写的库。Groovy是一种脚本语言，可以很好地与Java结合编程。该语言特别适合与Spring的动态语言支持一起使用，设计时充分考虑了Java集成，这使Groovy与Java代码的互操作很容易。主要应用于Web应用开发领域。

3.4.9 Go

Go语言是谷歌公司2009年发布的第二款开源编程语言。

Go语言专门针对多处理器系统应用程序的编程进行了优化，使用Go编译的程序可以媲美C或C++代码的速度，而且更加安全、支持并行进程。它是一种通用型的语言，可以用来开发任何软件——从普通应用到系统编程。虽然这种语言还不成熟，各种语言特征和规格还在变化，但程序员如今已经用它来进行软件开发工作了。能否像一些技术分析家所说的那样——“Go将最终完全替代Java”，我们将拭目以待。

3.5 UML对面向对象开发的支持

按照软件的生命周期，可以把面向对象软件开发过程分为：需求分析、面向对象分析、面向对象设计、面向对象实现4个主要阶段。UML提供了对需求分析、面向对象分析、面向对象设计的支持。软件开发实践中仍然存在“重实现，轻建模”的现象。这个现象产生的原因，一是除了实现的工作以外，又增加了建模的工作，工作量加大了；二是建模和实现没有能够很好地衔接。IBM RSA是基于Eclipse的，提供了从建模到实现的较好衔接，并且支持模型驱动的开发。

3.5.1 用例模型

用例模型描述的是外部执行者所理解的系统功能。用例模型用于需求分析阶段，它的建立是系统开发者和用户反复讨论的结果，表明了开发者和用户对需求规格达成的共识。首先，它描述了待开发系统的功能需求；其次，它将系统看作黑盒，从外部执行者的角度来理解系统；第三，它驱动了需求分析之后各阶段的开发工作，不仅在开发过程中保证了系统所有功能的实现，而且被用于验证和检测所开发的系统，从而影响到开发工作的各个阶段和UML的各个模型。用例模型从用户角度描述系统功能，并指出各功能的操作者。

用例模型描述了一个系统的功能需求，主要包括系统要实现的功能（用例）、环境（参入者）以及用例与参加者之间的关系。用例与参加者之间的关系可以用用例图来表示，用例图中可以包含事件流的文字说明，以及参入者和系统之间的交互信息等。对于一些比较复杂的系统，还可以使用活动图来表示用例中的事件流。

3.5.2 分析模型

分析模型描述了正在建模的系统或应用程序的结构。它包括描述用例模型中所确定的功能需求的逻辑实现的类和序列图。

分析模型确定出系统中的主要类，并且包含一组描述系统如何构建的用例实现。类图通过利用原型对系统的功能部分建模，对系统的静态结构进行描述。序列图通过描述用例中事件执行时的流对用例进行实现。这些用例实现对系统的一些部分如何在具体用例环境中交互进行建模。

分析模型描述了系统的逻辑结构，因而它是设计模型的基础。

3.5.3 设计模型

设计模型是基于分析模型的，它向系统的实际实现中添加了详细信息。设计模型通过使用各种图（包括时序图、状态机图、组件图和部署图），详细地描述了应用程序是如何构成，以及如何实现的。它还描述了程序设计构想及技术，例如那些用于持久性、部署、安全，及记录的内容。

思考题

1. 什么是面向对象方法学？与其他软件开发方法相比较，它有什么特点？
2. 什么是对象？它与传统的数据有何异同？
3. 什么是类、继承和多态？
4. 什么是模型？开发软件为什么要建模？
5. 简述面向对象分析的基本过程。
6. 简述面向对象设计的基本过程。
7. 简述面向对象程序设计的基本特征。
8. 简述常用面向对象分析和设计方法的异同。
9. 简述 UML 与面向对象分析与设计的关系。