第1章 绪　论

面向对象程序设计是软件系统设计与实现的新方法，这种方法是通过增加软件的可扩充性和可重用性来提高程序员的生产能力，控制软件的复杂性，降低软件维护的开销。因此，它的应用使软件开发的难度和费用大幅度降低，已为世界软件产业带来了革命性的突破。

1.1 面向对象方法的起源

“对象”一词在现实生活中经常会遇到，它表示现实世界中的某个具体的事物。社会的进步和计算机科学的发展是相互促进的，随着计算机的普及和应用，人们越来越希望能更直接地与计算机进行交互，而不需要经过专门学习和长时间训练后才能使用它。这使得软件设计人员的负担越来越重，软件的实现越来越复杂，并且对计算机领域自身的发展也提出了新的要求。利用传统的程序设计思想无法满足这一要求时，人们就开始寻求一种能帮助人类解决问题的自然方法，这就是“面向对象”技术。

20 世纪 50 年代的程序都是用指令代码或汇编语言编写的，这种程序的设计相当复杂，编制和调试一个稍大一点的程序常常要花费很长时间，培养一个熟练的程序员更需经过长期训练和实践，这种局面严重影响了计算机的普及和应用。

20 世纪 60 年代高级语言的出现大大简化了程序设计，缩短了软件开发周期，显示出了强大的生命力。此后，编制程序已不再是专业软件人员才能做的事了，一般工程技术人员花较短的时间学习后，也可以使用计算机解题。这个时期，随着计算机日益广泛地渗透到各个学科和技术领域，一系列不同风格、为不同目标服务的程序设计语言发展起来了，其中较为著名的有 FORTRAN、COBOL、ALGOL、LISP、PASCAL 等十几种语言。高级语言的蓬勃兴起，使得编译原理和形式语言理论日趋完善，这是该时期的主要特征。但是就整个程序设计方法而言，并无实质性的改进。

自 20 世纪 60 年代末到 20 世纪 70 年代初，出现了大型软件系统，如操作系统、数据库，这给程序设计带来了新的问题。大型系统的研制需要花费大量的资金和人力，可是研制出来的产品却可靠性差、错误多、不易维护和修改。一个大型操作系统有时需要每年几千人的工作量，而所获得的系统又常常会隐藏着几百甚至几千个错误。当时，人们称这种现象为“软件危机”。

为了克服 20 世纪 60 年代出现的软件危机，1968 年北约组织提出“软件工程”的概念。对程序设计语言的认识从强调表达能力为重点转向以结构化和简明性为重点，将程序从语句序列转向相互作用的模块集合。1969 年，E. W. Dijkstra 首先提出了结构化程序设计的

概念,他强调从程序结构和风格上来研究程序设计。在软件工程迫切需要改进的背景下,20 世纪70 年代结构化语言获得蓬勃发展并得到广泛应用。使用结构化程序设计方法可显著地减少软件的复杂性,提高软件的可靠性、可测试性和可维护性。经过几年的探索和实践,结构化程序设计的应用确实取得了成效,用结构化程序设计的方法编写出来的程序不仅结构良好、易写易读,而且易于验证其正确性。

进入 20 世纪 80 年代,由于一系列高技术的研究,如第五代计算机、计算机辅助制造(CAM)和知识工程等领域的研究,都迫切要求大型的软件系统作支撑。它们所用的数据类型也超出了常规的结构化数据类型的范畴,并且提出了对图像、声音、规则等非结构化信息的管理。为了满足这些应用领域的需要,就迫切要求软件模块具有更强的独立自治性,以便于大型软件的管理、维护和重用。由于结构化语言的数据类型较为简单,采用过程调用机制也不够灵活,独立性较差,所以不能胜任对非结构化数据的定义与管理。

为了适应高技术发展的需要,消除结构化程序设计语言的局限,自 20 世纪 80 年代以来,出现了面向对象程序设计流派,研制出了多种面向对象程序设计语言(Object Oriented Programming Language,OOPL),如 Ada、Smalltalk、C++ 语言和当前使用在 Internet 上与平台无关的 Java 语言等。

由于 OOPL 的对象、类具有高度的抽象性,所以能很好地表达复杂的数据类型,并且 OOPL 也允许程序员灵活地定义自己所需要的数据类型。类本身具有完整的封装性,可以使用它作为编程中的模块单元,满足模块独立自治的要求。另外,类的继承性和多态性功能更有助于简化大型软件和大量重复定义的模块,从而增强了模块的可重用性,提高了软件的可靠性,缩短了软件的开发周期。

1.2 面向对象是软件方法学的返璞归真

客观世界是由许多具体事物、抽象概念、规则等组成的,人们将任何感兴趣或要加以研究的事、物、概念统称为对象(object)。每个对象都有各自的内部状态和运动规律,不同对象之间通过消息传递进行相互作用和联系就构成了各种不同的系统。面向对象的方法正是以对象作为基本元素的一种分析问题和解决问题的方法。

传统的结构化方法强调的是功能抽象和模块化,每个模块都是一个过程,结构化方法处理问题是以过程为中心的。对象包含数据和对数据的操作,是对数据和功能的抽象和统一。而面向对象强调的是功能抽象和数据抽象,用对象来描述事物和过程,面向对象方法处理问题的过程是对一系列相关对象的操纵,即发送消息到目标对象,由目标对象执行相应的操作。因此面向对象方法是以对象为中心的,这种以对象为中心的方法更自然、更直接地反映现实世界的问题空间,从而具有独特的抽象性、封装性、继承性和多态性的特点,更好地适应了复杂大系统不断发展与变化的要求。

采用对象的观点看待所要解决的问题,并将其抽象为应用系统是极其自然与简单的,因为它符合人类的思维习惯,使得应用系统更容易理解。同时,由于应用系统是由相互独立的对象构成的,系统的修改可以局部化,因此系统维护更加容易。

软件开发从本质上讲就是对软件所要处理的问题域进行正确的认识,并把这种认识正确地描述出来。既然如此,那就应该直接面对问题域中客观存在的事物来进行软件开发,这

就是面向对象。另一方面，人类在认识世界的过程中形成的普遍有效的思维方法，在软件开发中也是适用的。在软件开发中尽量采用人们日常生活中习惯的思维方式和表达方式，这就是面向对象方法所强调的基本原则。软件开发从过分专业化的方法、规则和技巧中脱离出来，并重新回到了客观世界，回到了人们的日常思维当中，所以说面向对象方法是软件方法学的返璞归真。

1.3 结构化程序设计与面向对象程序设计

要想真正了解面向对象程序设计，首先需要回顾一下结构化程序设计的含义。

1. 结构化程序设计

结构化程序设计是 20 世纪 60 年代诞生的，在 70 年代到 80 年代已遍及全球，成为软件开发设计所有领域及每个程序员都采用的程序设计方法，它的产生和发展形成了现代软件工程的基础。

结构化程序设计的设计思想是：自顶向下、逐步求精；其程序结构按功能划分为若干个基本模块，这些模块形成一个树状结构；各模块之间的关系尽可能简单，在功能上相对独立；每一模块内部均由顺序、选择和循环三种基本结构组成；其模块化实现的具体方法是使用子程序、过程或函数。

结构化程序设计由于采用了模块分解和功能抽象、自顶向下、分而治之的手段，从而有效地将一个复杂的软件系统的设计任务分成许多容易控制和处理的子任务，这些子任务都是可独立编程实验的子程序模块。每一个子程序都有一个清晰的界面，使用起来非常方便。

结构化程序设计方法虽然具有许多的优点，但它仍是一种面向过程的设计方法，它把数据和过程分离为相互独立的实体，程序员在编程时必须时刻考虑所要处理的数据格式。对于不同的数据格式即使做同样的处理或对相同的数据格式做不同的处理都需要编写不同的程序。因此结构化程序的可重用性不好；另一方面，当数据和过程相互独立时，总存在着用错误的数据调用正确的程序模块或用正确的数据调用错误的程序模块的可能性。因此，要使数据和程序始终保持相容，已成为程序员的一个沉重负担，并且随着软件系统的规模越来越大，程序的复杂性越来越难以控制。上述这些问题，结构化程序设计方法本身是解决不了的，需要借助于下面要讨论的面向对象程序设计方法给予解决。

程序设计的任务是描述问题并解决问题，在结构化程序设计中可以用下面的式子表示程序：

程序 = 数据结构 + 算法 + 程序设计语言 + 语言环境

图 1.1 所示为结构化程序设计中程序的结构。

2. 面向对象程序设计——程序设计的新思维

面向对象程序设计既吸取了结构化程序设计的一切优点，又考虑了现实世界与面向对象空间的映射关系，它所追求的目标是将现实世界问题的求解尽可能简单化。

面向对象程序设计将数据及对数据的操作放在一起，作为一个相互依存、不可分割的整体来处理，它采用了数据抽象和信息隐藏技术。它将对象及对对象的操作抽象成一种新的数据类型——类，并且考虑不同对象之间的联系和对象所在类的可重用性。

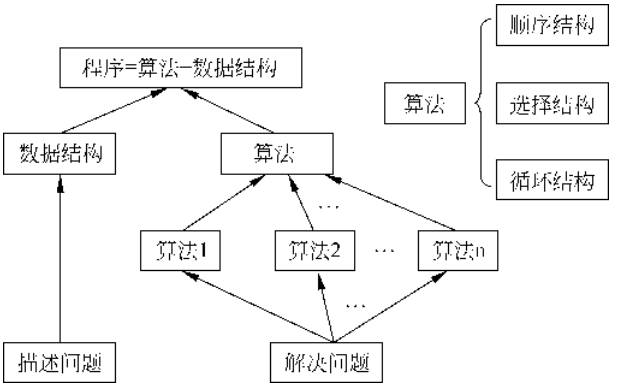

图 1.1　结构化程序设计中程序的结构

面向对象程序设计优于传统的结构化程序设计,其优越性表现在,它有希望解决软件工程的两个主要的问题——软件复杂性控制和软件生产率的提高,此外它还符合人类的思维习惯,能够自然地表现现实世界的实体和问题,它对软件开发过程具有重要的意义。

面向对象程序设计能支持的软件开发策略有:

(1) 编写可重用代码。

(2) 编写可维护代码。

(3) 共享代码。

(4) 精减已有代码。

有了高质量的可重用代码就能有效地降低软件的复杂度、提高开发效率。面向对象方法,尤其是它的继承性,是一种代码重用的有效途径。开发者在设计软件时可以利用一些已经精心设计好并且经过测试的代码,这些可重用的代码以类的形式被组织存放在程序设计环境的类库中。类库中的这些类的存在,使以后的程序设计过程变得简单,程序复杂性不断降低、正确性不断提高,程序越来越容易理解、修改和扩充。

在面向对象程序设计中可以用下面的式子表示程序:

程序 = 对象 + 对象 + … + 对象
对象 = 算法 + 数据结构 + 程序设计语言 + 语言环境

图 1.2 所示为面向对象程序设计中程序的结构。

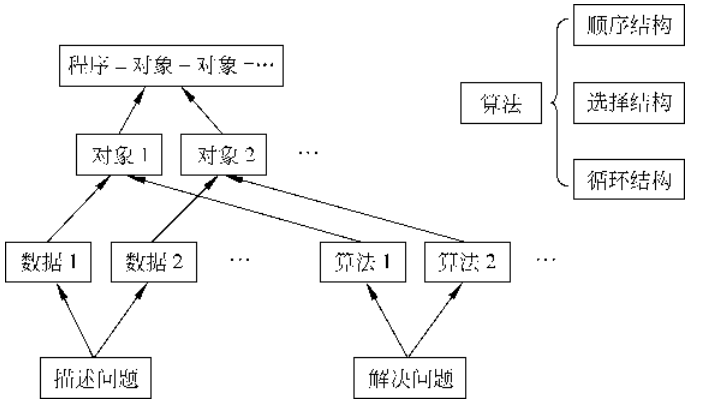

图 1.2　面向对象程序设计中程序的结构

1.4 面向对象的基本概念和面向对象系统的特性

1.4.1 面向对象的基本概念

1. 对象(object)

对象是现实世界中一个实际存在的事物,它可以是有形的(如一支粉笔、一块橡皮等),也可以是无形的或无法整体触及的抽象事件(如一项计划、一场球赛、一次借书行为等)。对象是构成世界的一个独立单位,它具有自己的静态特征和动态特征。静态特征可以用某种数据来描述,动态特征即对象所表现的行为或对象所具有的功能。

在面向对象系统中,对象是系统中用来描述客观事物的一个实体,它是构成系统的一个基本单位。一个对象由一组属性和对这组属性进行操作的一组服务构成。属性和服务是构成对象的两个主要因素,属性是一组数据结构的集合,表示对象的一种状态,对象的状态只供对象自身使用,用来描述静态特性。而服务是用来描述对象动态特征(行为)的一个操作序列,是对象一组功能的体现,包括自操作和它操作。自操作是对象对其内部数据(属性)进行的操作,它操作是对其他对象进行的操作。

一个对象可以包含多个属性和多个服务,对象的属性值只能由这个对象的服务存取和修改。对象是其自身所具有的状态特征及可以对这些状态施加的操作结合在一起所构成的独立实体。对象具有如下的特性:

(1) 具有唯一标识名,可以区别于其他对象。

(2) 具有一个状态,由与其相关联的属性值集合所表征。

(3) 具有一组操作方法即服务,每个操作决定对象的一种行为。

(4) 一个对象的成员仍可以是一个对象。

(5) 模块独立性。从逻辑上看,一个对象是一个独立存在的模块,模块内部状态不因外界的干扰而改变,也不会涉及其他模块;模块间的依赖性极小或几乎没有;各模块可独立地被系统组合选用,也可被程序员重用,不必担心影响其他模块。

(6) 动态连接性。客观世界中的对象之间是有联系的,在面向对象程序设计中,通过消息机制,把对象之间动态连接在一起,使整个机体运转起来,便称为对象的连接性。

(7) 易维护性。由于对象的修改、完善功能及其实现的细节都被局限于该对象的内部,不会涉及外部,这就使得对象和整个系统变得非常容易维护。

对象从形式上看是系统程序员、应用程序员或用户所定义的抽象数据类型的变量,当用户定义了一个对象,就创造出了具有丰富内涵的抽象数据类型的实例。

2. 类(class)

在面向对象系统中,并不是将各个具体的对象都进行描述,而是忽略其非本质的特性,找出其共性,将对象划分成不同的类,这一过程称为抽象过程。类是对象的抽象及描述,是具有共同属性和操作的多个对象的相似特性的统一描述体。在类的描述中,每个类要有一个名字标识,用以表示一组对象的共同特征。类的每个对象都是该类的实例。类提供了完整的解决特定问题的能力,因为类描述了数据结构(对象属性)、算法(服务、方法)和外部接

口(消息协议),是一种用户自定义的数据类型。

3. 消息(message)

消息是面向对象系统中实现对象间的通信和请求任务的操作,是要求某个对象执行其中某个功能操作的规格说明。发送消息的对象称为发送者,接受消息的对象称为接收者。对象间的联系,只能通过消息来进行。对象在接收到消息时才被激活。

消息具有三个性质:

(1) 同一对象可接收不同形式的多个消息,产生不同的响应。

(2) 相同形式的消息可以发送给不同对象,所做出的响应可以是截然不同的。

(3) 消息的发送可以不考虑具体的接收者,对象可以响应消息,也可以对消息不予理会,对消息的响应并不是必需的。

对象之间传送的消息一般由三部分组成:接收对象名、调用操作名和必要的参数。

在面向对象程序设计中,消息分为两类:公有消息和私有消息。假设有一批消息发向同一个对象,其中一部分消息是由其他对象直接向它发送的,称为公有(public)消息;另一部分消息是它向自己发送的,称为私有(private)消息。

4. 方法(method)

在面向对象程序设计中,要求某一对象完成某一操作时,就向对象发送一个相应的消息,当对象接收到发向它的消息时,就调用有关的方法,执行相应的操作。方法就是对象所能执行的操作。方法包括界面和方法体两部分。方法的界面就是消息的模式,它给出了方法的调用协议;方法体则是实现这种操作的一系列计算步骤,也就是一段程序。消息和方法的关系是:对象根据接收到的消息,调用相应的方法;反过来,有了方法,对象才能响应相应的消息。所以消息模式与方法界面应该是一致的。同时,只要方法界面保持不变,方法体的改动不会影响方法的调用方式。在 C++ 语言中方法是通过函数来实现的,称为成员函数。

1.4.2 面向对象系统的特性

1. 抽象性(abstract)

面向对象的方法鼓励程序员以抽象的观点看待程序,即程序是由一组对象组成的。程序员可以将一组对象的共同特征进一步抽象出来,从而形成“类”的概念。抽象是一种从一般的观点看待事物的方法,它要求程序员集中于事物的本质特征,而不是具体细节或具体实现。类的概念来自于人们认识自然、认识社会的过程。在这一过程中,人们主要使用两种方法:从特殊到一般的归纳法和从一般到特殊的演绎法。在归纳的过程中,人们从一个个具体的事物中把共同的特征抽取出来,形成一个一般的概念,这就是“归类”;在演绎的过程中,人们又把同类的事物,根据不同的特征分成不同的小类,这就是“分类”。对于一个具体的类,它有许多具体的个体,面向对象方法称这些个体为“对象”。

2. 封装性(encapsulation)

所谓数据封装就是指一组数据和与这组数据有关的操作集合组装在一起,形成一个能动的实体,也就是对象。数据封装就是给数据提供了与外界联系的标准接口,无论是谁,只有通过这些接口并使用规范的方式,才能访问这些数据。数据封装是软件工程发展的必然产物,它使得程序员在设计程序时可以专注于自己的对象,同时也切断了不同模块之间数据

的非法使用途径，从而减少了出错的可能性。

3. 继承性(inheritance)

从已有的对象类型出发建立一种新的对象类型，使它继承原对象的特点和功能，这种思想是面向对象设计方法的主要贡献。继承是对许多问题中分层特性的一种自然描述，因而也是类的具体化和实现重用的一种手段，它所表达的就是一种不同类之间的继承关系。它使得某类对象可以继承另外一类对象的特征和能力。继承所具有的作用有两个方面：一方面可以减少冗余代码，另一方面可以通过协调性来减少接口和界面。

从继承源上划分继承可分为单一继承(单继承)和多重继承(多继承)，子类对单个直接父类的继承称为单一继承，子类对多于一个直接父类的继承称为多重继承。父类也称为基类或超类，子类也称为派生类。如图 1.3 所示为单一继承举例，图 1.4 所示为多重继承举例。

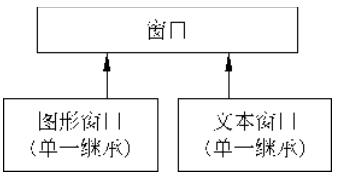

图 1.3　单一继承举例

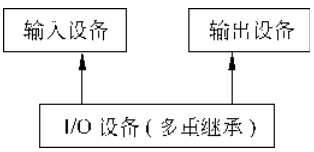

图 1.4　多重继承举例

从继承内容上继承可分为取代继承、包含继承、受限继承、特化继承。

(1) 取代继承：例如一个徒弟从其师傅那里学到师傅的所有技术，则在任何需要师傅的地方都可以由徒弟来代替。

(2) 包含继承：例如交通工具是一类对象，汽车是一种特殊的交通工具。汽车具有了交通工具的所有特征，任何一辆汽车都是一种交通工具，这便是包含继承，即汽车包含了交通工具的所有特征。

(3) 受限继承：例如鸵鸟是一种特殊的鸟，它不能继承鸟会飞的特征。

(4) 特化继承：例如教师是一类特殊的人，他们比一般人具有更多的特有信息，这就是特化继承。

4. 多态性(polymorphism)

不同的对象接收到相同的消息时产生完全不同的行为的现象称为多态性。C++ 语言支持两种多态性，即编译时的多态性和运行时的多态性。编译时的多态性通过重载函数实现，而运行时的多态性通过虚函数实现。使用多态性可以大大提高人们解决复杂问题的能力。

1.5　面向对象程序设计语言的四大家族

面向对象程序设计语言经历了一个比较长的发展阶段，以下从几个大的程序设计语言家族来介绍面向对象程序语言的发展。

1. LISP 家族

LISP 是 20 世纪 50 年代开发出来的一种语言，它以表处理为特色，是一种人工智能语言，20 世纪 70 年代以来，在 LISP 基础上开发了很多 LISP 家族的面向对象语言。

2. Simula

Simula 语言是 20 世纪 60 年代开发出来的,在 Simula 中引入了几个面向对象程序设计语言中最重要的概念和特性,即数据抽象、类和继承机制。Simula 67 是具有代表性的一个版本,20 世纪 70 年代发展起来的 CLU、Ada、Modula-2 等语言是在 Simula 67 的基础上发展起来的。

3. Smalltalk

Smalltalk 是第一个真正的面向对象程序设计语言,它体现了纯粹的 OOP 设计思想,是最纯的 OOP 语言。它起源于 Simula 语言。尽管 Smalltalk 不断完善,但在那个时期,面向对象程序设计语言并没有得到广泛的重视,程序设计的主流方法是结构化程序设计。

4. C 家族

在 20 世纪 80 年代,C 语言成为一种极其流行、应用非常广泛的语言。C++ 语言是在 C 语言的基础上进行扩充,并增加了类似 Smalltalk 语言中相应的对象机制。它将类看做是用户定义类型,使其扩充比较自然。C++ 语言以其高效的执行效率赢得了广大程序员的青睐,C++ 语言提供了对 C 语言的兼容性,因此,很多已有的 C 语言程序稍加改造甚至不加改造就可以重用,许多有效的算法也可以重新利用。它是一种混合型的面向对象程序设计语言,由于它的出现,才使面向对象的程序设计语言得到越来越多的重视和应用。

Java 语言是一种适用于分布式计算的新型面向对象程序设计语言,可以看做是 C++ 语言的派生,它从 C++ 语言中继承了大量的语法成分,抛弃了 C++ 语言中复杂的、容易引起问题的功能,并增加了多线程、异常处理、网络程序设计等方面的支持,掌握了 C++ 语言就可以很快学会 Java 语言。

面向对象语言可以分为两大类,纯粹的面向对象语言和混合型的面向对象语言。在纯粹的面向对象语言中,几乎所有的语言成分都是“对象”,这类语言强调开发快速原型的能力;而混合型的面向对象语言,是在传统的面向过程语言中加入了各种面向对象的语法机制,它所强调的是运行效率。真正的面向对象程序设计语言提供了特定的语法成分来保证和支持面向对象程序设计,并且提供了继承性、多态性和动态链接机制,使得类和类库成为可重用的模块。

1.6 面向对象的系统开发方法

计算机应用系统的开发是一个相当复杂的过程,使用面向对象方法进行系统开发,首要任务是采用面向对象的概念及其抽象机制将开发的系统对象化和模型化,建立应用系统模型;然后使用面向对象的程序设计语言来实现系统中的对象。面向对象的开发方法可分为四个阶段。

(1) 系统调查和需求分析:即对应用系统将要实现的功能以及用户对系统开发的需求进行调查研究。这是所有开发方都必须进行的。

(2) 分析问题的性质和求解问题:在繁杂的问题域中抽象出对象及其行为、结构、属性、方法等。这一阶段称为面向对象分析,简称为 OOA。

(3) 整理问题:即对分析的结果进一步抽象、归类、整理,最终以规范的形式描述对象和类。这一步称为面向对象设计,简称为 OOD。

(4) 程序实现：即用面向对象的程序设计语言将上一步整理的对象和类的描述映射为应用程序软件。这一步一般称为面向对象程序设计，简称为 OOP。

面向对象开发方法的几个阶段中最主要的工作是系统的对象化和模型化，并使用对象模型描述系统。OOA 阶段建立初步的对象模型，然后在 OOD 阶段对对象或类进行归类，对类及对象的属性、方法和结构等进行归类，抽象出它们的共同部分，使类的层次结构更加合理。

1.6.1 面向对象分析 OOA

面向对象分析是用面向对象方法对问题域和系统任务进行分析和理解，找出描述问题域及系统责任所需的对象(包括类、继承与派生关系等)和消息传递，定义对象的属性、服务以及它们之间的关系，最后形成面向对象模型，为面向对象设计打下基础。最终的目标是建立一个符合用户需求，并能够直接反映问题和系统责任的 OOA 模型及其详细说明。

问题域是被开发系统的应用领域，即在现实世界中由这个系统进行处理的业务范围。系统责任是所开发的系统应该具备的职能。

1. 面向对象分析模型

在 OOA 过程中，要将本质的或逻辑的系统需求确定为系统基本行为，最终形成 OOA 模型，它由一组相关的类组成，是软件规格说明最重要的组成部分。OOA 模型采用层次结构，分为五层，这五个层次不是构成软件系统的层次，而是分析过程中的层次，可以说是问题的不同侧面。这五个层次分别如下：

(1) 对象——类层

该层是表达待开发系统的最基本单位，标出系统中的对象和类，并用符号进行规范性的描述和命名。

(2) 属性层

属性层定义对象和某些结构中的数据单元，继承结构中所有类的公共属性可放于通用类中。

(3) 服务层

服务层标识对象的服务，即类功能的确定，列出对象需要做什么(方法)，给出对象间的消息连接(并用箭头指示消息从发送者到接受者)，消息系列用执行线程来表达，服务用类似流程图的方式表达。

(4) 结构层

结构层识别现实世界中对象之间的关系，当一个对象是另一个对象的一部分时，用整体-部分关系标识，一个对象类属于另一个对象时，用一般-特殊的继承关系表示。

(5) 主题层

主题层是管理大系统的一种方法，一个主题可以看做是一个子系统或一个子模型，可将相关类或对象分别归类到各个主题中，并赋予标号和名称。

图 1.5 是常用的描述类和对象的符号，图 1.6 是对象和类的实例。

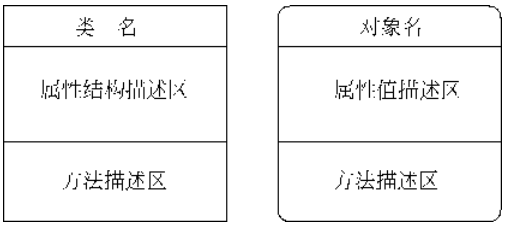

图 1.5 常用的描述类和对象的符号

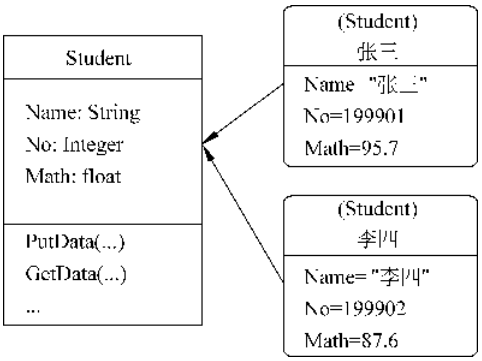

图 1.6 对象和类的实例

2. 面向对象分析步骤

面向对象分析的关键是对问题域中事物的识别及其相互关系的判定,即识别对象和对象之间的消息传递方式。基于面向对象的方法学原则进行系统分析时,一般要进行如下的步骤。

(1) 定义对象。这一步主要包括确定类与对象层、对象属性层、服务层、对象之间的消息。

(2) 确定对象之间的结构关系,即结构层。具体的分析原则:第一是按照一般-特殊结构,确定标识类间的继承关系;第二是按照整体-部分结构,确定一个对象怎样由其他对象构成,或者是如何将一些小对象组合成大对象。

(3) 标识主题,即主题层。主题可以看做是高层次的子模块或子系统,是面向对象模型的整体框架,更精练地表示面向对象模型。

1.6.2 面向对象设计 OOD

面向对象设计 OOD 阶段是 OOA 阶段工作的继续,很难将 OOD 和 OOA 截然分开。OOD 是对在 OOA 中得到的结果进行改进和增补,逐渐扩充 OOA 模型的过程,面向对象设计分两个阶段:高层设计和低层设计。低层设计集中于类的详细设计阶段。这一阶段应考虑对时间与空间的折中、内存管理、开发人员的调配、增加类、属性和关系等。

高层设计阶段主要是开发系统的结构,构造待开发软件的总体模型。该阶段主要解决软件系统所处的计算机环境中的概念和类,主要包括:

1. 问题域的设计

对 OOA 模型的改进和修补,对 OOA 中的类、对象、属性、操作及结构等进行组合与分解。

2. 用户界面的设计

通常在 OOA 阶段给出对象所需的属性和操作,在设计阶段必须根据需要把交互的细节加入到用户界面的设计中,如人机交互所需的显示和输入信息提示等。

3. 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,根据需要设计数据的管理方式,如文件管理形式、关系型数据库管理系统或面向对象的数据库管理系统。

4. 任务管理的设计

当系统中有许多并发过程时,需要依照各个过程的协调和通信关系,划分成不同的进程,每个进程就是一个任务,进程之间协调运行。

1.6.3 OOA 和 OOD 的基本步骤

1. 标识对象和对象的属性

标识应用系统的对象和对象的属性是面向对象设计过程中最艰难的工作。首先要搞清楚系统要解决的问题到底涉及哪些事物以及它们在系统中的作用。按照面向对象的观点，可以将事物归纳为以下三类。

(1) 客观存在物：包括有形对象和角色对象，体现问题的结构特性。

(2) 行为：包括事件对象和交互对象。行为是对象的一部分，行为依赖于对象。它体现问题的行为特性。

(3) 概念：现实世界中事物和它们行为规律的抽象，是识别对象时的一类认识和分析对象。

标识对象可以从应用系统非形式化描述中的名词来导出。标识出对象后，还应注意对象之间的类似之处，以建立对象类。例如，Windows 多窗口用户界面中，不同的窗口具有类似的特性，每个窗口都可以看做是某些窗口类的实例，每个窗口都具有大小、位置、标题等属性。

2. 标识每个对象所要求的操作和提供的操作

标识出每一个对象执行的功能，这些功能描述对象的行为，例如窗口被打开、关闭、缩放和滚动等。同时还应关心由其他对象提供给它的操作，通过标识这些操作有可能导出新对象。

3. 建立对象之间的联系和每个对象的接口

建立对象和对象类之间的联系，标识出与每一个对象有关的对象和对象类。这一步中可能找出一些对象的模式，并决定是否要建立一个新类以表示这些对象的共同行为特性。

识别出系统中的对象和类以后，还应该识别出对象之间的相互作用，即对象的外部接口。在面向对象系统中，对象和对象之间的联系是通过消息的发送和响应来完成的。类和对象之间的相互关系可以用类的层次结构图和对象间的消息流图等图形工具来描述。

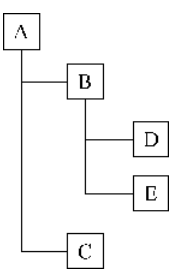

图 1.7　一个简单的类层次结构图

类的层次结构图用来描述系统中类的层次关系和结构，可以跟踪基类和派生类之间的关系。图 1.7 是一个简单的类层次结构图，表示类 A 派生出类 B 和类 C，类 B 又派生出类 D 和类 E。

对象间的消息流图用于描述系统中对象间的消息流，它只描述那些相关对象间交换的主要消息。

消息流图有两种，即内向消息流图和外向消息流图。内向消息流图描述一个对象如何从其他对象处接受消息。图 1.8 是一个内向消息流图的例子，对象 B 和对象 C 分别向对象 A 发送一条消息。外向消息流图描述一个特定的对象发送给其他对象的所有消息，与内向消息流图正好相反。图 1.9 是一个外向消息流图的例子，对象 A 分别发送了一个消息给对象 B 和对象 C。

面向对象系统中类的描述可以使用专门的类描述语言来完成，也可以借助于面向对象程序设计语言中类的定义语法来描述，限于篇幅，本书不再详细介绍。

没有完全形式化的方法可以保证使用面向对象的方法进行分析和设计结果的唯一性，对象及类的识别、划分以及相互之间的关系并没有唯一的标准，分析和设计的结果是否合理

很大程度上依赖于设计人员的经验和技巧。

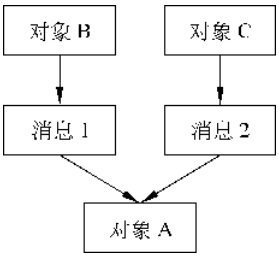

图 1.8　内向消息流图

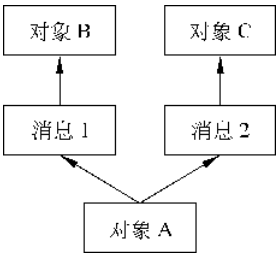

图 1.9　外向消息流图

1.7　面向对象程序设计举例

下面介绍一个简单的面向对象程序,然后从中分析面向对象程序的特点。

【例 1.1】 使用面向对象程序设计方法,编写一对堆栈进行处理的程序,包括压栈和弹栈操作。

```
#include <iostream.h>
class Stack                        // 定义堆栈类
{
  struct Node
  {
        int content;
        Node * next;
  } * top;
public:
  Stack() { top = NULL; }          // 构造函数的定义
  bool push(int i);                // 压栈成员函数的声明
  bool pop(int& i);                // 弹栈成员函数的声明
};

bool Stack::push(int i)            // 压栈成员函数的定义
{
    Node * p = new Node;
    if (p == NULL)
    {
      cout << "can't allocate memory.\n";
      return false;
    }
    else
    {
        p->content = i;
        p->next = top;
        top = p;
        return true;
    }
}
```

```
bool Stack::pop(int& i)                          // 弹栈成员函数的定义
{
      if (top == NULL)
      {
        cout << "Stack is empty.\n";
        return false;
      }
      else
      {
        Node * p = top;
        top = top->next;
        i = p->content;
        delete p;
        return true;
      }
}

void main()
{
     Stack st1,st2;                              // 定义对象 st1 和 st2
     int x;
     for(int i = 1;i<= 5;i ++ )
     {
        st1.push(i);                             // 压栈成员函数的调用
        st2.push(i);                             // 压栈成员函数的调用
     }
     cout<<"stack1:"<<endl;
     for(i = 1;i<= 3;i ++ )
     {
          st1.pop(x);                            // 弹栈成员函数的调用
          cout<<x<<endl;
     }
     st1.push(20);
     for(i = 1;i<= 4;i ++ )
     {
       if(st1.pop(x))
            cout<<x<<endl;
       else
            break;
     }
     cout<<"stack2:"<<endl;
     while(st2.pop(x))
        cout<<x<<endl;
}
```

程序的运行结果为：

```
stack1:
5
4
3
```

```
20
2
1
Stack is empty.
stack2:
5
4
3
2
1
Stack is empty.
```

分析：

由上述程序可知，在面向对象程序设计中，程序是由不同的对象构成的；类是一种用户自定义的数据类型，通过类定义的变量称为对象；类和对象构成面向对象程序设计的不同模块；通过不同的对象发送消息即可完成相应的操作。

习　　题

1. 什么是面向对象程序设计？它与传统的结构化程序设计有什么不同？
2. 面向对象程序设计语言有哪几类？
3. 面向对象系统有哪些特性，分别加以解释。
4. 阐述类、对象、消息和方法的概念。

第2章 C++语言对C语言的扩充

C++语言是在C语言的基础上扩充了面向对象的概念及相应的处理机制而形成的一种混合型程序设计语言。本章主要介绍C++语言在传统的非面向对象方面对C语言的扩充，以便为后面章节的学习和编程打下基础。

2.1 C++语言的特点

随着面向对象程序设计思想的日益普及，很多支持面向对象程序设计方法的语言也相继出现了，C++语言就是这样一种语言。C++是Bjarne Stroustrup于1980年在AT&T的贝尔实验室开发的一种语言，它是C语言的超集和扩展，是在C语言的基础上扩充了面向对象的语言成分而形成的。最初这种扩展后的语言称为带类(class)的C语言，1983年才被正式称为C++语言。

1988年出现了第一个用于PC的Zortech C++ 2.0编译系统，次年出现了Turbo C++ 2.0编译器。之后，Borland公司从1991年起陆续推出了Borland C++ 2.0/3.0/4.0系统。相比之下Microsoft公司的动作迟缓了一些，直到1992年才推出了基于DOS平台的MS C/C++ 7.0系统，但由于其把握着PC操作系统的命脉，所以发展势头强劲，很快于1993年推出了基于Windows系统平台的Visual C++ 1.0，1994年推出了Visual C++ 1.5和可用于Windows 95和Windows NT系统平台的Visual C++ 2.0，直至1998年推出Visual C++ 6.0。

C++语言既保留了C语言的有效性、灵活性、便于移植等全部精华和特点，又添加了面向对象编程的支持，具有强大的编程功能，可方便地构造出模拟现实问题的实体和操作；编写出的程序具有结构清晰、易于扩充等优良特性，适合于各种应用软件、系统软件的程序设计。用C++语言编写的程序可读性好，生成的代码质量高，运行效率仅比汇编语言慢10%～20%。

C++语言由C语言扩展而来，并且全面兼容C，这就使许多C代码不经修改就可以为C++所用；同时它又对C语言的发展产生了一定的影响，ANSI C语言在标准化过程中吸收了C++语言中某些语言成分。

2.2 C++语言的文件扩展名

为了使编译器能够区别是C语言还是C++语言，C++语言体系规定用.cpp(意即C Plus Plus)作为C++语言源文件的扩展名以区别于C语言用的.c文件扩展名。虽然仅差两个字母，但编译时所做的处理却相差甚远。所有操作系统的C++语言源文件扩展名均为

.cpp。与 C++ 语言源文件相关的头文件扩展名一般仍用 .h,但有些操作系统也有规定使用 .hpp 充当头文件扩展名的。

2.3 注 释 符

在 C 语言中用"/*…*/"符号作为程序注释,这种注释被称为段注释,在 C++ 语言中也可以使用段注释符,此外 C++ 语言还增加了注释符"//"。当只做单行注释时便可用"//"符号表示从此符号起至行尾均为行注释内容。程序编译时将忽略所有的注释内容。

2.4 名 字 空 间

名字空间(namespace)也称名空间,是随标准 C++ 语言而引入的。它相当于一个更加灵活的文件域(全局域),可以用花括号把文件的一部分括起来,并以关键字 namespace 开头给它起一个名字,例如:

```
namespace ns1
{
      float a,b,c;
      fun1(){…}
}
```

花括号括起来的部分称为声明块。声明块中可以包括:类、变量(带有初始化)、函数(带有定义)等。在域外使用域内的成员时,需加上名字空间名作为前缀,后面加上域操作符"::"。这里添加了名字空间名称的成员名被称为限定修饰名,例如 ns1::a,ns1::fun1()等。

最外层的名字空间称为全局名字空间(global namespace scope),即文件域。

名字空间可分层嵌套,同样有分层屏蔽作用。例如:

```
namespace n1
{
    namespace n2
    {     // 名字空间嵌套
          class matrix{…}     // 名字空间类成员 matrix
    }
}
```

访问 matrix,可写 n1::n2::matrix。

使用 using 声明可只写一次限定修饰名。using 声明以关键字 using 开头,后面是被限定修饰的(qualified)名字空间成员名,例如:

```
using n1::n2::matrix;        // 名字空间中的类成员 matrix 的 using 声明
```

以后在程序中使用 matrix 时,就可以直接使用成员名,而不必使用限定修饰名。

注意:如果写成 using namespace n1::n2::matrix; 是错误的。using namespace 后只能加名空间,而不能加类名或变量名。

使用 using 指示符可以一次性地使名字空间中所有成员都可以直接被使用,比 using 声明

方便。using 指示符以关键字 using 开头，后面是关键字 namespace，然后是名字空间的名称。

标准 C++ 库中的所有组件都是在一个被称为 std 的名字空间中声明和定义的。在采用标准 C++ 的平台上使用标准 C++ 库中的组件，只要写一个 using 指示符：

```
using namespace std;
```

这样就可以直接使用标准 C++ 库中的所有成员，这是很方便的。需要注意的是，如果使用了名空间 std，则在使用 #include 编译预处理命令包含头文件时，必须去掉头文件的扩展名.h，否则会出错。

名字空间可以不连续，分为多段，但它们仍是同一个名字空间。名字空间不能定义在函数声明、函数定义或类定义的内部。

2.5 C++ 语言的输入输出

在 C 语言中，输入输出是依靠函数来实现的（如标准输入输出用的 scanf()、printf()等）。通常只要在程序的最前端用预处理命令包含头文件，如：#include<stdio.h>，就可以调用这类函数。由于此类函数的调用语句书写起来比较冗长，为了简化起见 C++ 语言又另外定义了一套保留字与运算符来替代 C 语言中对标准输入、输出函数的调用。C++ 语言中输出和输入的一般格式如下：

```
cout << 输出内容 1 << 输出内容 2 << …;
// cout 为标准输出流对象(默认输出到显示器),输出内容可以为常量、变量或表达式
cin >> 变量 1 >> 变量 2 >> …;  // cin 为标准输入流对象(默认从键盘输入)
```

支持 cout 和 cin 这两个流对象的内部函数体在一个名为 iostream.h（即标准 I/O 流）的头文件中予以声明，所以应用时一定要将其放置在对应的头文件声明部位。有关 C++ 语言的输入输出流参见第 7 章。

【例 2.1】 C++ 语言的输入输出举例。

```
#include<iostream>          // 使用名空间 std,则必须去掉.h 扩展名
using namespace std;
void main()
{   char name[10];
    int age;
    cout<<"please input your name:";
    cin>>name;
    cout<<"How old are you:";
    cin>>age;
    cout<<"name is "<<name<<endl;
    cout<<"age is "<<age<<endl;
}
```

用户输入 karl、30，程序的运行结果为：

```
please  input  your  name:Karl
How  old  are  you:30
name  is  Karl
age  is  30
```

将此例的标准输入、输出语法与 C 语言的 printf()和 scanf()函数的书写语法对比后可以很明显地看出：此种标准的输入、输出语法的书写大大简化了函数格式的描述，而由C++语言编译器按被操作数的类型和具体内容代为选定输入输出格式。

2.6 变量的定义

C++ 语言除了新增的类数据(即 class)类型以外，还继承了 C 语言所支持的所有数据类型。在 C 语言中，局部变量说明必须置于可执行代码段之前，不允许局部变量声明和可执行代码混合在一起。但 C++ 在变量的定义方面做了两种较大的改变，一是允许变量的定义语句出现在程序的任何位置，使得局部变量的定义位置与使用位置不至于离得太远，增强了程序的可读性，而且也不必在编写某一程序块的开始时就考虑要用到哪些变量；二是允许直接使用结构体名定义变量，这种扩展为程序员在编程中提供了不少方便。类似地在 C++ 语言中联合名、枚举名也可在定义后独立地作为类型名使用。

【例 2.2】 C++ 语言的变量定义举例。

```
#include<iostream>
using namespace std;
void main()
{
    struct student
    {
        int no;
        float math;
    };
    int n;
    cin>>n;
    student wang;
    // C++ 中变量的定义语句可以出现在程序的任意位置；可以使用结构体名定义变量
    wang.no = n;
    cin>>wang.math;
    cout<<wang.no<<" "<<wang.math<<endl;
}
```

2.7 强制类型转换

在 C 语言中可以利用强制类型转换运算符将一个表达式转换成所需类型，其一般形式如下：

(数据类型)(表达式)

C++ 语言除了保留这种转换方式外，还提供了一种更为方便、合理的转换方式，形式如下：

数据类型(表达式)

说明：

(1) 通过强制类型转换，得到一个所需类型的中间值，该中间值被引用后即自动释放，

原来表达式值的类型并未改变。如下列代码段：

```
int b;
float f;
f = float(b);              // 此时变量 b 仍然为 int 类型
```

(2) 强制类型转换符优先级较高，只对紧随其后的表达式起作用，而对其他部分不起作用，如表达式 float(i) * f 的含义是先将变量 i 强制类型转换为 float 类型，然后与变量 f 运算。

(3) 强制类型转换应当用在不做转换将影响表达式结果的正确性或精度、或不能完成相应运算的场合，而对于系统可以自动转换类型的场合，则没有必要使用。

2.8 动态内存的分配与释放

在程序运行时可使用的内存空间称为堆(heap)。堆内存就是在程序运行时获得的内存空间，在程序编译和连接时不必确定它的大小，它随着程序的运行过程变化，时大时小，因此堆内存是动态的，堆内存也称为动态内存。

在软件开发中，常常需要动态地分配和释放内存空间。C 语言是利用库函数 malloc()和 free()分配和释放堆内存空间的。但是使用 malloc()函数时必须指定需要开辟的内存空间的大小，其调用形式为 malloc(size)。size 是字节数，需要事先求出或用 sizeof 运算符由系统求出。此外，malloc()函数只能从用户处知道应开辟空间的大小而不知道数据的类型，因此无法使其返回的指针指向具体的数据，其返回值一律为 void * 类型，且必须在程序中进行强制类型转换，才能使其返回的指针指向具体的数据。

C++ 语言提供了两种方法进行内存的动态分配和释放，可以使用从 C 标准库中继承来的 malloc()和 free()函数(此时要包含头文件 malloc.h)，也可以使用 new 和 delete 运算符。

1. new 运算符

运算符 new 用于分配堆内存，它的使用形式如下：

```
指针变量 = new 数据类型;
```

new 从堆内存中为程序分配可以保存某种类型数据的一块内存空间，并返回指向该内存的首地址，该地址存放于指针变量中。

堆内存可以按照要求进行分配，程序对内存的需求量随时会发生变化，有时程序在运行中可能会不再需要由 new 分配的内存空间，而且程序还未运行结束，这时就需要把先前占用的内存空间释放给堆内存，以便重新分配，供程序的其他部分使用。

2. delete 运算符

运算符 delete 用于释放 new 分配的内存空间，它的使用形式如下：

```
delete  指针变量;
```

其中的指针变量保存着 new 动态分配的内存的首地址。

3. 注意事项

在使用 new 和 delete 运算符时，应注意以下几点：

(1) 用 new 获取的内存空间，必须用 delete 进行释放。

(2) 对一个指针只能调用一次 delete。

(3) 用 delete 运算符作用的对象必须是用 new 分配的内存空间的首地址。

【例 2.3】 new 与 delete 应用举例。

```
#include <iostream>
using namespace std;
void main()
{
  int *p;
  p=new int;  // 分配内存空间
  *p=5;
  cout<<*p;
  delete p;    // 释放内存空间
}
```

程序的运行结果为:

```
5
```

说明:

该程序用 new 分配可用来保存 int 类型数据的内存空间,并将指向该内存的地址放在指针变量 p 中。在该变量不再使用后,又使用 delete 将内存空间释放。这样,通过使用 new 和 delete,就在程序中建立了可由程序员控制生存期的变量。

new 所建立的变量的初始值是任意的,故在程序中使用表达式 *p=5 为变量赋初始值,也可以在用 new 分配内存的同时进行初始化。使用形式如下:

```
指针变量 = new 数据类型(初始值);
```

例如例 2.3 中的

```
p=new int;
*p=5;
```

也可写成:

```
p=new int(5);              // 圆括号内给出用于初始化这块内存的初始值
```

4. 用 new 建立数组类型的变量

用 new 也可以建立数组类型的变量,使用形式如下:

```
指针变量 = new 数据类型[数组大小];
```

此时指针变量指向第一个数组元素的地址。使用 new 分配数组时,不能提供初始值。使用 new 建立的数组变量也由 delete 释放。使用形式如下:

```
delete 指针变量;
```

或

```
delete[ ]指针变量;
```

同样,也可以用 new 来为多维数组分配空间,但是除第一维可以为变量外,其他维数都