第1章　绪　　论

教学提示

本章主要介绍面向过程程序设计和面向对象程序设计两种程序设计方法的特点和差异；重点介绍面向对象程序设计的基本概念、原理和方法。其中，类、对象、消息、继承和多态是面向对象程序设计的核心概念。正确理解这些概念，了解面向对象程序的构造原理，对后续章节的学习是非常重要的。

程序设计是针对特定问题规划并编写程序的过程，是软件构造活动中的重要组成部分。在计算机技术发展的早期，由于机器资源比较昂贵，程序的时间和空间代价往往是程序设计者关心的主要因素。这个时期的程序员追求编程的技巧，将注意力集中在问题求解本身，无暇关注求解的过程，更谈不上考虑程序结构的合理性和可指导性。随着硬件技术的飞速发展和软件规模的日益庞大，程序的结构、可维护性、复用性、可扩展性等因素日益重要。然而，早期的软件开发过度依赖于程序员的个人经验，缺乏科学理论和方法作为指导，在大型软件的开发过程中出现了复杂程度高、研制周期长和正确性难以保证三大难题。遇到的问题找不到解决办法，致使问题堆积起来，形成了人们难以控制的局面。20世纪60年代末最终出现了所谓的“软件危机”。

为了化解这一危机，一方面，需要对程序设计方法、程序的正确性和软件的可靠性等问题进行系统的研究；另一方面，也需要对软件的编制、测试、维护和管理的方法进行研究，从而产生了程序设计方法学。在这些程序设计方法中，最值得关注的包括面向过程的结构化程序设计方法和面向对象的程序设计方法。

1.1　面向过程的结构化程序设计

面向过程的结构化程序设计(structured programing，SP)思想最早由E. W. Dijikstra在1965年提出，是软件发展的一个重要的里程碑。它的主要观点是采用自顶向下、逐步求精的程序设计方法。任何程序都可由顺序、选择、循环三种基本控制结构构造；以模块化设计为中心，将待开发的软件系统划分为若干个相互独立的模块，这样使每一个模块的设计工作变得单纯而明确，为设计一些较大的软件打下了良好的基础。各模块按要求单独编程，再将各模块连接，组合构成相应的软件系统。该方法强调程序的结构性，所以容易做到易读、易懂。结构化程序设计方法思路清晰，做法规范，深受设计者青睐，成为20世纪七八十年代软件开发的潮流。

结构化程序设计的主要特点是采取“自顶向下、逐步细化、模块化设计、结构化编码”的指导思想，将软件的复杂度控制在一定范围内，因此从整体上降低了软件开发的复杂度。按照这种原则和方法可设计出结构清晰、容易理解、容易修改、容易验证的程序。结构化程序设计的目标在于使程序具有一个合理的结构，以保证和验证程序的正确性，从而开发出正确、合理的程序。

1. 自顶向下分析问题的方法

自顶向下分析问题的方法，就是把大的复杂的问题分解成若干小问题后，再逐个解决这些小问题的策略。面对一个复杂的问题，首先进行上层（整体）的分析，按组织或者功能将问题分解成子问题，如果子问题仍然十分复杂，再做进一步分解，直到处理对象相对简单，容易解决为止。当所有子问题都得到了解决，整个问题也就解决了。在这个过程中，每一次分解都对上一层问题进行细化和逐步求精，最终通过一种类似于树形的层次结构描述分析的结果。按照自顶向下的方法分析问题，有助于后续的模块化设计和测试，以及系统的集成。

2. 模块化设计

经过问题分析，在设计好层次结构图后就进入模块化设计阶段了。在这个阶段，需要将模块细分，组织成良好的层次系统，顶层模块调用其下层模块以实现程序的完整功能，每个下层模块再调用更下层的模块，从而完成程序的一个子功能，最下层的模块完成最基础的功能。

模块化设计要遵循独立性的原则，即模块之间的联系应该尽量简单，主要体现在以下四个方面。

（1）一个模块只完成一个指定的功能。

（2）模块之间只通过参数进行调用。

（3）一个模块只有一个入口和一个出口。

（4）模块内慎用全局变量。

模块化设计使程序结构清晰，易于设计和理解。当程序出错的时候，只需要改动出错的模块及与之直接关联的模块。模块化设计有利于大型软件的开发，程序员们可以分工编写不同的模块。

在C语言中，模块一般通过函数来实现，一个模块对应一个函数。在设计某一个具体的模块时，模块中的语句一般不要超过50行，这既便于编程者思考和设计，也利于程序的阅读。

3. 结构化编码

经模块化设计后，每一个模块都可以独立编码。编程时应该选用顺序、选择和循环三种控制结构，对于复杂问题可以通过这三种结构的组合、嵌套实现，以清晰表达程序的逻辑结构。

1.2　面向对象的程序设计方法

1.2.1　面向对象的程序设计方法的产生

早期的计算机主要应用于科学计算，但随着计算机硬件的不断发展，计算机的性能越来越强，应用的领域也越来越广泛，不再局限于科学计算。随之而来的是软件规模越来越大，程序越来越复杂和庞大。面向过程的程序设计方法逐渐暴露出它的不足和缺陷。

1. 结构化程序在规模变大时难以理解和维护

在结构化的程序中，函数与其所操作的数据（全局变量）之间的关系没有清晰和直观地体现出来。随着程序规模的增加，程序逐渐难以理解，很难立刻看出函数之间存在怎样的调用关系，某项数据到底有哪些函数可以对它进行操作，某个函数到底是用来操作哪些数据的。

图 1-1 所示为结构化程序的模式，箭头代表“访问”或“调用”，主函数 main()与变量（var1、var2、var3）、其他函数[Sub1()、Sub2()、Sub3()、Sub1_1()等]之间的访问或调用关系呈现为复杂的网状结构。在此情况下，当传递的值不正确时，很难找出到底是哪个函数导致的，因而程序的查错也变得困难。

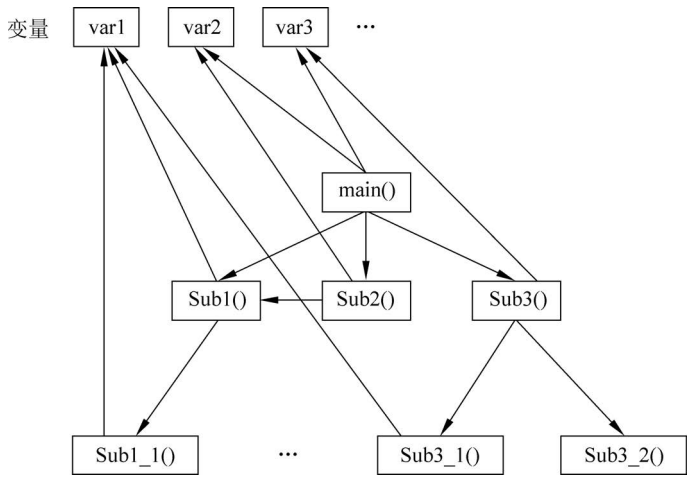


图 1-1　结构化程序的模式

2. 结构化的程序不利于修改和扩充（增加新功能）

结构化程序设计没有“数据封装”和“隐藏”的概念，当要访问某个全局变量时可以直接访问，当该全局变量的定义有改动时，就要把所有访问该变量的语句找出来修改，然而这些语句可能分散在上百个函数之中，因此十分费时费力。

3. 结构化程序不利于代码的重用

在编写某个程序时，常常会发现其需要的某项功能在现有的某个程序中已经有了相同或类似的实现，程序设计者自然希望能够将那部分源代码抽取出来，在新程序中使用，这种做法称为代码的重用。但是，在结构化的程序设计中，随着程序规模的增大，大量函数、变量之间的关系错综复杂，要抽取可重用的代码往往变得十分困难。

例如，要重用一个函数，可是这个函数又调用了新程序所不需要的其他函数，因此在重用该函数时，就不得不将其他函数也一并抽取出来；更糟糕的是，所要重用的函数访问了某些全局变量，这样还要将不相干的全局变量也抽取出来，或者修改被重用的函数以取消对全局变量的访问。

在结构化的程序中，各个模块之间耦合度高，有千丝万缕牵扯不清的联系，想要重用某个模块，就像是在杂乱摆放着一大堆设备的计算机桌上拿走笔记本电脑的电源适配器一样，拿起电源适配器，它的连接线却扯出一大堆乱七八糟的音箱、耳机、鼠标、外接光驱等东西。

总之，随着软件系统规模的不断扩大，结构化的程序越来越难以扩充、查错和重用。结构化程序设计越来越难以适应软件开发的需要。因此，面向对象的程序设计方法应运而生。面向对象的程序设计方法由于有封装、继承、多态性的特性，可以设计出低耦合的系统，使系统更加灵活、更加易于维护。当然，事物都有两面性，面向对象的程序设计方法性能比面向过程低，对硬件要求高。面向对象的程序设计方法并不能完全取代面向过程的程序设计方法，面向对象更适合于需求不断变化的应用软件，而面向过程更适合需求稳定且对程序执行效率要求高的软件，如 Linux/UNIX 操作系统软件就是使用 C 语言采用面向过程方法开发的。另外，单片机、嵌入式开发以及实时性软件的开发一般都使用面向过程的结构化程序设计方法。

1.2.2 基本概念

面向对象程序设计是将数据与对数据的操作封装在一起，成为一个不可分割的整体——对象，同时将具有相同特征的对象抽象成一种新的数据类型——类。程序的基本组成单位就是类，将程序和数据封装其中，以提高软件的重用性、灵活性和扩展性。程序在运行时，由类定义对象，对象之间通过发送消息（简单来说，就是一个对象调用另一个对象所属类中的函数）进行通信，相互协作完成相应的功能。

可以从两个方面来认识和理解面向对象的程序设计方法。一方面，面向对象是针对面向过程数据与操作数据的函数分离的缺点，引入了“类”这种数据类型，把数据和操作数据的函数定义在一个类中。一般情况下，只有同一个类中的函数（称为成员函数）才能访问类中定义的数据（称为成员变量）。函数不再是程序的基本单元，取而代之，类是程序的基本单元，不允许在类之外定义变量和函数，这样就取消了全局变量和全局函数，所有的变量和函数都要定义在某个类中。类中成员变量的定义有改动时，波及范围仅限于类的内部的成员函数，而不会扩散到整个程序，这样就极大增强了程序的可维护性。

另一方面，面向对象的程序设计方法的本质是主张参照人们认识一个现实系统的过程，完成分析、设计与实现一个软件系统，用人类在现实生活中常用的思维方法来认识、理解和描述客观事物，强调最终建立的系统能映射问题域，使得系统中的对象（类类型的变量称为对象），以及对象之间的关系能够如实地反映问题域中固有的事物及其关系。

面向过程是一种以过程为中心的编程思想，其原理就是将问题分解成一个一个详细的解决步骤，然后通过函数实现每一个步骤，并依次调用这些函数从而实现解决问题。面向过程所关心的是解决一个问题的步骤。例如，汽车发动、汽车熄火，这是两个不同的操作，对于面向过程而言，关心的是操作本身，因此会使用两个函数完成以上两个操作，然后依次调用即可。

面向对象则是一种以对象为中心的编程思想，就是通过分析问题域，分解出问题域中一个一个的事物，然后通过不同对象之间的组合协作解决问题。建立对象的目的不是完成某个步骤，而是为了描述某个事物在解决整个问题的过程中的行为。

如上面的例子，汽车发动、汽车熄火，对于面向对象而言，关心的是汽车这类对象，两个操作只是这类对象所具备的行为。

1. 对象

对象是用来描述客观事物的一个实体。一个对象具有一组属性和操作，属性是用来描述对象静态特征的数据，代表对象的状态；操作也称为方法，是用来描述对象动态特征的行为，代表对象对外提供的功能。对象由类描述，并通过类实例化生成。

例如，一个时钟对象。

时钟对象的属性：

属性名	属性值
时	10
分	30
秒	30

时钟对象的方法：设置时、设置分、设置秒、显示时间。

2. 类

类是对具有相同属性和方法的对象的抽象，它为属于该类的全部对象提供统一的抽象描述。图 1-2 所示为从时钟对象抽象出的时钟类。图 1-3 所示为时钟类的类图。类是生成对象的模板，对象是类的实例。在面向对象程序设计语言里，类是一种数据类型，对象是这种数据类型的实例。类里的属性一般是私有成员，只能在类内访问，在类外不能访问。类里的方法（表示行为）一般是公有成员，可供类外通过类的对象调用。一个类的不同对象一般具有不同的属性值，如两个时钟对象，一个代表北京时间，另一个代表纽约时间，它们的属性值是不同的，如图 1-4 所示。

说明：图 1-3 和图 1-4 所示的统一建模语言（unified modeling language，UML）类图和对象图都分为三部分，第一部分是类名，对象图里是“对象名：所属类名”，并且带下画

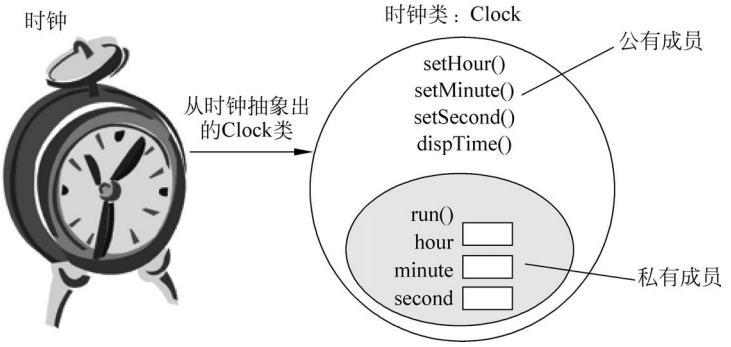


图 1-2　时钟类

线以和类图相区别。第二部分是属性，类图中的格式是：[可见性][属性名：类型]，“-”表示私有成员，“+”表示公有成员；对象图里的格式是：[可见性][属性名：类型]=[属性值]。第三部分是方法，对象图中省略了。

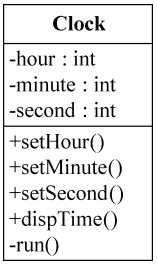
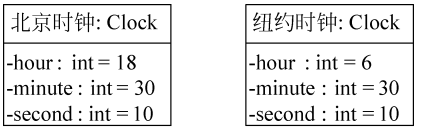


图 1-3　时钟类的类图



(a) 北京时钟　　(b) 纽约时钟

图 1-4　时钟类的两个对象图

3. 消息

对象之间的联系是通过消息传递完成的。一个消息就是一个对象对另一个对象发出的“请求”或“命令”。接收者收到消息后，调用有关的方法，执行相应的操作。一个消息由三部分组成：接收消息的对象、消息名（接收对象要调用的方法）和方法需要的参数。简单地说，给一个对象发消息，就是调用这个对象的方法。

请求者发送消息，接收者响应消息，这是面向对象程序工作的基本方式，消息是驱动面向对象程序运转的源泉。在面向对象的程序系统中，消息分为两类，即私有消息和公有消息。私有消息是对象发给自身的消息，通常表现为对象的行为在执行过程中，调用了一个自身私有的行为操作，私有消息对外是不公开的；公有消息是外部对象发送给这个对象的消息。

1.2.3　面向对象的基本特征

1. 封装性

封装（encapsulation）就是把类（对象）的属性和行为结合成一个独立的单位，并尽可

能隐蔽类(对象)的内部细节。封装有两层含义：一是把类(对象)的全部属性和行为结合在一起,形成一个不可分割的独立单位,对象的属性值(除了公有的属性值)只能由这个对象的行为来读取和修改；二是尽可能隐蔽类(对象)的内部细节,对外形成一道屏障,与外部的联系只能通过外部接口实现。封装的信息隐蔽作用反映了事物的相对独立性,可以只关心它对外所提供的接口,即能做什么,而不用关注其内部细节,即怎么提供这些服务。例如电视机,我们只关心它对外提供哪些功能按钮,而不关心它的内部有哪些部件以及功能是如何实现的。

封装的结果一方面使对象以外的部分不能随意存取对象的内部属性,从而有效地避免了外部错误对它的影响,大大减小了查错和排错的难度；另一方面,当对对象内部进行修改时,由于它只通过少量的外部接口对外提供服务,因此同样减小了内部的修改对外部的影响,如图 1-5 所示。

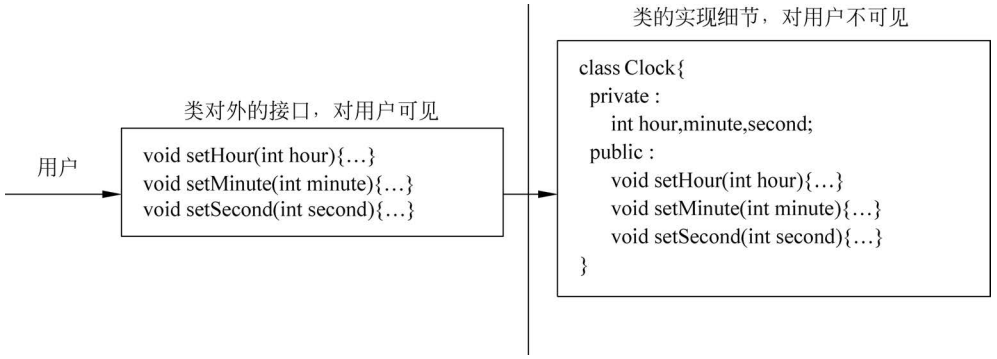


图 1-5　类(对象)的封装性

2. 继承性

继承是针对类之间的相交关系,使得某类可以继承另外一类的特征和功能。被继承的现有类称为父类(基类或超类),从现有类继承的新类称为子类(或派生类)。由一个父类可以派生出任意多个子类,这样就形成了类的继承体,如图 1-6 所示,“教师”类和“学生”类作为子类继承了父类“人”的不同特征和功能；同样,“教师”类可作为“任课教师”类和“教辅人员”类的父类,“学生”类可作为“本科生”类和“研究生”类的父类。这是现实世界中事物的分类问题在计算机中的解的形式。继承使软件复用变得简单、易行,可以通过继承复用已有的类。继承有以下四个作用。

(1) 清晰地体现类之间的结构层次关系。

(2) 减少代码和数据的冗余度,增强软件的可重用性。

(3) 是代码复用的有力工具,建立和扩展新类的有效手段。

(4) 通过增强一致性来统一一个继承体系下类的对外接口,增加软件的可维护性。

继承允许和鼓励类的重用,提供了一种明确表述共性的方法。一个特殊类既有自己新定义的属性和行为,又有继承下来的属性和行为。尽管继承下来的属性和行为是隐式的,但无论在概念上还是在实际效果上,都是这个类的属性和行为。当这个特殊类又被它

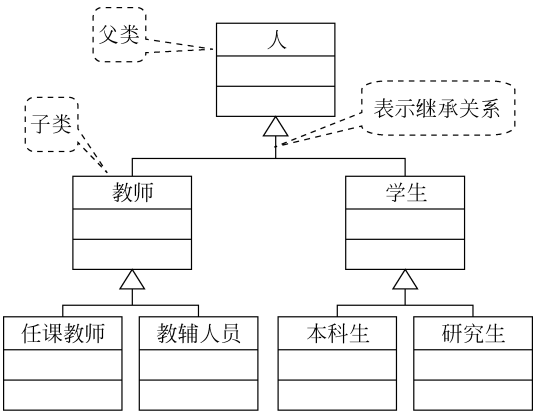


图 1-6　类的继承体系

更下层的特殊类继承时，它继承来的与自己定义的属性和行为又被下一层的特殊类继承下去。因此，继承是传递的，体现了大自然中特殊与一般的关系。

在软件开发过程中，继承性实现了软件模块的可重用性、独立性，缩短了开发周期，提高了软件开发的效率，同时使软件易于维护和修改。这是因为要修改或增加某一属性或行为，只需在相应的类中进行改动，而它派生的所有类都自动地、隐含地发生了相应的改动。由此可见，继承是对客观世界的直接反映，通过类的继承，能够实现对问题的深入抽象描述，反映人类认识问题的发展过程。

3. 多态性

多态分为编译时多态和运行时多态。编译时多态是通过方法重载实现；运行时多态是通过方法覆盖实现（子类覆盖父类方法）。通常说的多态性（polymorphism）是指运行时多态性，是指同一个继承体系中不同类的对象收到相同的消息时产生多种不同的行为方式。例如，在一般类“几何图形类”中定义了一个“绘图”行为，但并不确定执行时到底画一个什么图形。特殊类“圆类”和“三角形类”都继承了几何图形类的“绘图”行为，但其功能却不同，一个是要画出一个圆，另一个是要画出一个三角形。这样在一个绘图的消息发出后，圆类、三角形类等的对象接收到这个消息后各自执行不同的绘图函数，画出不同的图形，这就是多态性的表现。

继承性和多态性的结合，可以生成一系列虽类似但独一无二的对象。由于继承性，这些对象共享许多相似的特征；由于多态性，针对相同的消息，不同对象可以有独特的表现方式，实现个性化的设计。

1.3　C++与面向对象程序设计

C++语言是在 C 语言的基础上发展而来的，是一种混合型程序设计语言，它既可以进行 C 语言的过程化程序设计，又可以进行以继承和多态为特点的面向对象的程序设计。

AT&T 公司 Bell 实验室实验计算机科学研究中心的 B. Stroustrup 于 20 世纪 80 年代初首先设计并实现了 C++语言。C++语言是对 C 语言的扩充，并且借鉴了许多其他著名程序设计语言的精华特征。例如，C++语言从 Simula67 语言中引入了类的构造，从 ALGOL60 语言吸取了引用及在分程序中声明变量的技术，并且综合了 Ada 语言的类属、抽象类和异常处理等方法。

C++语言为类的定义提供了两种方式，其一是对 C 语言中原有的结构体(struct)进行扩充，其二是提供新的类(class)构造。C++语言保持了 C 语言紧凑、灵活、高效和易于移植的优点。它用类的机制实现数据抽象，用虚函数体现面向对象程序设计的动态联编。由于 C++语言兼容了使用广泛的 C 程序设计语言，因此从 C 语言向 C++语言过渡比较平滑，而且大批的 C 程序可以在 C++上得到重用。C++没有对复杂操作系统的依赖性，它可以使用存在于其所处主机系统上的任何一种环境，所以 C++允许人们开发出能和正在运行的任何程序交互的程序。正是由于 C++语言有丰富的应用基础和广泛的开发环境的支持，使得它颇受程序设计工作者的青睐，很快成为面向对象的主流程序设计语言之一。

C++语言自从 1979 年年底 B. Stroustrup 完成预处理程序 Core 到现在，经历了四十多年的发展，其内容和风格在不断地演变。在 B. Stroustrup 于 1983 年采纳 Rick Mascitti 的建议，将其新设计的语言命名为 C++语言之前，这个语言普遍被人们称为“带类的 C”，由此可见 C++语言与 C 语言的密切关系。

1984 年，C++语言引入了借助操作符重载技术实现的 I/O 流技术，这为 C++语言树立了全新的风格。从技术角度讲，这种全新语法的引入弥补了 C 语言中 printf()函数族缺乏类型安全机制和扩展能力的弱点。从代码风格上看，“<<”等通俗易懂的运算符使程序中的输入和输出变得容易编程，而输入/输出的控制正是许多程序设计语言中让编程人员深感头痛的地方。

1998 年 9 月，C++标准化委员会正式发布了 C++的国际标准，正式确认了包括模板、容器类、I/O 流类库、异常处理等典型语言特征的现代 C++风格。该标准通常简称 C++ 98 标准，这个版本的 C++被认为是标准 C++。所有的主流 C++编译器都支持这个版本的 C++，包括微软的 Visual C++和 Borland 公司的 C++ Builder。

2011 年 8 月，C++ 11 标准发布，C++ 11 包含了核心语言的新机能，拓展了 C++标准程序库，并且加入了大部分的 C++ Technical Report 1 程序库(数学上的特殊函数除外)。此次标准为 C++ 98 发布后 13 年来第一次重大修改。

2020 年 12 月，C++ 20 标准发布，在 C++ 20 中，最重要的两个特性是模块(modules)和协程(coroutine)。此外，C++ 20 的新特性还包括范围、概念与约束(concepts and constraints)、指定初始化(designated initializers)、计时、并行算法和对并发编程的一些改进等。

通过 C++语言发展的历史可以看出，C++语言是一种在实践中诞生、成长和发展起来的语言。与 Smalltalk 那样纯粹的面向对象语言不同，软件开发实践对程序开放性、高效率、兼容性和扩展性的需求把 C++语言塑造成一种典型的多模式语言，这也是 C++被称为混合型语言的原因。C++语言对面向对象技术的全面支持，使得 C++语言得到普遍的应用，这也是我们选择它作为学习面向对象程序设计的描述语言的理由。通过学习 C++语

言，由学习到创新，期待早日能有国产优秀的编程语言出现。

本章小结

本章分别介绍了面向过程和面向对象的程序设计方法，包括它们的基本原理、基本概念，讨论了面向对象程序设计方法的基本特征，最后介绍了C++面向对象的程序设计语言。虽然本章对上述内容的介绍只是简要的、原则性的，但这些内容提供了一个全局的视图，掌握这些内容对于今后的学习无疑是十分关键的。

思考题

1. 面向对象程序设计的基本原理和基本特征分别是什么？
2. 面向过程的结构化程序设计的主要特点是什么？