徐德启　窦世成　编著

面向对象基础与C++程序设计教程

兰州大学出版社

目　录

第一章　面向对象基础

§1.1　面向对象方法学的产生

1.1.1　面向对象方法学产生的历史

六十年代以来，计算机科学不论在硬件、还是软件方面都有了飞速的发展。应用领域不断扩大，计算机已渗透到社会的各个领域。特别是九十年代计算机已成为人们生活中必不可少的工具，从而对计算机软件的要求越来越高，这就对传统的软件设计技术提出了挑战。那么能否用更一般、更接近于自然的软件设计方法来开发大众化的软件呢？对这个问题的探讨必须追溯到软件的产生与发展历史。

软件的发展受到应用和硬件发展的推动和制约，软件的发展大致经历了三个时代，即程序设计时代、软件时代和软件工程时代。

1945 年—1956 年为程序设计时代。这个时期设计的程序主要为开发者自用和小范围交流，程序设计语言是机器语言及其符号化的汇编语言，追求的目标是编程技巧和时空效率。突出的问题是程序设计与编制工作费时、费力、而且容易出错。

1956 年—1968 年为软件时代。在这个时期，随着计算机硬件的迅速发展和应用范围的扩大，各种高级语言如雨后春笋般地出现，软件正式出现，而且作为商品开始进入市场。

软件被定义为：

$$\text{软件} = \text{程序} + \text{文档}$$

$$\text{程序} = \text{数据结构} + \text{算法}$$

同时软件的复杂程度迅速提高，可靠性和可维护性问题突出，产生了第一次软件危机。

1968 年—现在为软件工程时代。这个时代，软件危机加剧。为了解决危机给软件设计带来的困难，人们提出以系统工程的观点来对待软件设计，从而产生了一些成熟的软件设计方法。SA/SD、JSP/JSD 方法就是在这个时代产生的。

结构化设计方法虽然对改善软件开发产生了巨大的影响，但它仍然把数据和代码作为分离的实体，存在着使用错误的数据调用正确的程序模块，或使用正确的数据调用错误的程序模块的问题。随着软件系统的庞大化，程序的查错、纠错越来越困难，导致第二次软件危机。七十年代末，面向对象方法学的一些基本概念已在系统工程领域内萌发出来了，到八十年代，面向对象的程序设计思想得到了很快的发展，并显示出其强大的生命力。人们预言，九十年代面向对象技术(OOT)将会在更深、更广、更高的方向上取得进展。

八十年代，面向对象的方法学产生，它的产生引起了计算机界的极大关注。因为面向对象的技术对于软件工程学濒临的困境和人工智能所遇到的障碍都是一个很有希望的突

破口，它既提供了从一般到特殊的演绎手段（继承），又提供了从特殊到一般的归纳方法（类）。它尽可能地使分析、设计和实现同我们认识世界的过程相一致。目前面向对象的研究已遍及计算机软、硬件的各个领域，如面向对象的方法学、面向对象的程序设计语言、面向对象的操作系统、面向对象的数据库、面向对象的体系结构等。

1.1.2 面向对象方法学产生的必然性

1.1.2.1 历史发展的必然

从上节我们知道，软件方法学的每一步发展都是在不断的解决或克服软件危机的过程中前进和发展的。随着计算机应用领域的迅速扩大，软件的复杂程度迅速提高，软件研制周期越来越长、正确性难以保证、可靠性问题突出，尤其是很难适应于动态变化的环境，这就引起软件危机的加剧，人们自然要考虑另一条解决危机的途径，这正是面向对象方法学产生的基础，八十年代面向对象方法学已经部分地解决或某种程度上缓解了软件危机的加剧。因此，面向对象方法学的产生也是软件工程发展历史的必然。

1.1.2.2 程序设计语言发展的必然

计算机诞生以来，程序设计语言是操纵计算机为人类服务的桥梁和工具，人们只有通过程序设计语言才能设计各种大型的、复杂的软件系统。然而随着软件系统规模的扩大与复杂性的增加，软件可靠性和可维护性明显降低，从而产生了软件危机。纵观这种危机产生的原因，主要是因为冯·诺依曼计算机直接可接受的语言是面向过程的机器语言，因而按这种面向过程的机器语言来编写的程序是比较难于理解的，它与人的自然语言距离较远，它们之间的这种鸿沟可如图 1 1 表示。

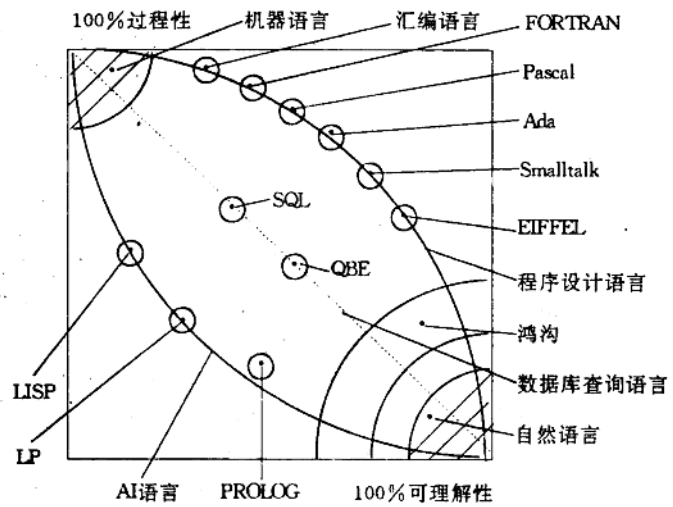

图 1.1 计算机语言与自然语言间的鸿沟

从上述可见，计算机语言的发展也迫切需要一种能使机器与人之间比较容易沟通的且更直观描地述计算的程序设计语言，这种语言便是面向对象的程序设计语言。显然它的产生也是促使面向对象设计方法的产生与发展的动力。

1.1.2.3 人类认识发展的必然

人类认识问题是以自然界对象为基础的，而结构化的程序设计方法遵循结构化的需求分析 SA(Structured requirement Analysis)、结构化的系统设计 SD(Structred system Design)和结构化程序设计 SP(Structured programming)的软件开发方法。而用这种 SA－SD－SP 的程序设计方法对基于计算机问题域对象的求解，存在着与实际问题域之间具有不能直接反映出人类认识问题的过程的鸿沟，这种鸿沟可如图 1.2 表示。

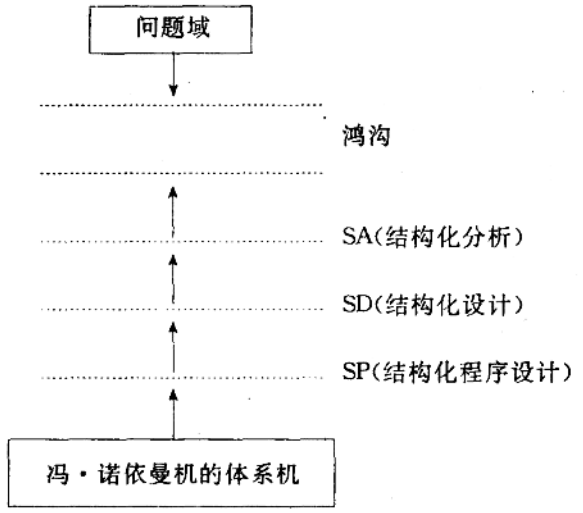

图 1.2 冯·诺依曼机与问题域的鸿沟

面向对象的方法则是摸拟人类认识问题的方法，它的出发点是自然界对象，这种认识问题的过程和人类认识问题的过程相一致，即从一般到特殊的分类演绎过程和从特殊到一般的归纳过程。因此面向对象的概念与现实问题概念之间存在必然的映射，这种映射可如图 1.3 所示。

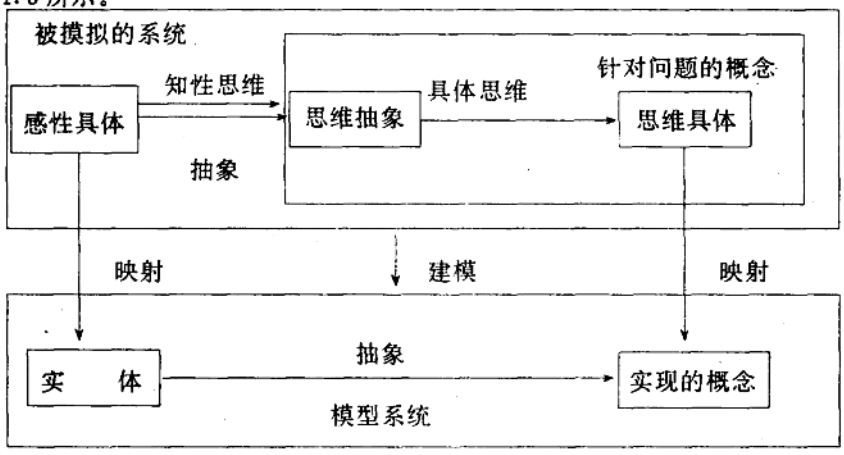

图 1.3 现实问题概念与被摸拟概念之间的映射

从这种认识上来看程序设计活动是在被摸拟的系统与模型系统之间的一种建模活

动，被摸拟的系统是“程序设计活动中我们的注意力所聚集的那一部分世界（主题范围）”，模型系统是在计算机上“摸拟被摸拟系统的一部分”的执行程序，被摸拟系统作为客体，它或者是具体的物理世界，或者是未来物理世界的某种想象或假想。

上面这种摸拟系统是一种最自然的认识活动。因此，面向对象方法是认识发展的必然，它是一种更高层次的抽象。

§1.2 面向对象的程序设计语言

自从八十年代面向对象的语言 Smalltalk 及其环境向计算机界推行以来，面向对象技术的研究已遍及计算机软硬件的各个领域。各种面向对象的语言系统如雨后春笋般地出现了。程序设计的范式必将以九十年代的“对象＋消息”的范式取代过去的“数据结构＋算法”范式。

1.2.1 面向对象的语言系统及分类

纵观现有的面向对象的语言系统，它们大致可以分为如下四类：

（1）纯面向对象的语言系统。如：

. 由Alan kay 提出，Xerox Palo Alto 研究中心开发的 Smalltalk(80)

. 由美国交互软件工程公司(ISE Inc.)开发的 Eiffel(85)

（2）传统语言仿真面向对象的语言。即以传统语言结合 Smalltalk 的面向对象的思想，并利用通信模型。如：

. Apple 公司开发的 Object Pascal(85)

. PPI(美国)公司开发的 Object－C(85)

. Apple 公司开发的 68000 用的汇编 Object Assembler(84)

. Kriya System 开发的 Neon 是面向对象的 Forth(84)

. Coral Software 开发的 Object Logo(86)

（3）传统语言采纳面向对象的思想。即纯过程语言中增加面向对象的结构，形成一种混合型面向对象的语言。如：

. AT&T 公司 Bell 实验室以 C 语言扩充类设施增加继承性而开发的 C＋＋(85)

. Xerox 公司在 Intel Lisp 上扩充面向对象的概念而开发的 Loops(83)

. 符号处理公司在 MIT 的 Lisp 机上开发的具有多继承的类对象的 Flavors(84)

. Xerox 公司推出的 Common Loop(85)，它是面向对象的 Common Lisp 版本

. 日本 IBM 分部以面向对象扩充 Prolog 而开发的 SPOOL(85)

. Keio 大学开发的 Orient 84K(84)是基于 Prolog 和 Smalltalk，能并行执行的语言

. 西德 Kaiserslautern 大学开发的 Mod Pascal(86)是 Pascal 面向对象的扩充

. DEC 公司86年推出的 Trellis/Owl 具有多继承性、强类型静态聚束，其母语言为 Pascal

（4）非面向对象的语言可作为基于对象的设计。

这类语言除原有的 Ada(80)、modula－2(79)、Alphard(81)、Clu(77)外还有：

. 美国 Place 大学 White Water 小组为知识表示开发的 Actor(88)，它是接近于 Smalltalk 的传统语言
. 卡内基梅隆大学开发的在 Macintosh 机上运行的 Lisp 方言 OakLisp(85)
. 日本开发的 Concurrent Prolog(92)
. Xerox 公司开发的并发 Prolog 预处理程序系统 Vulcan(86)

1.2.2 面向对象语言的演变

尽管有如此多的面向对象的语言系统，但它们都是从某种原有语言的基础上演变而来的，面向对象语言的产生和演变如图1.4所示。

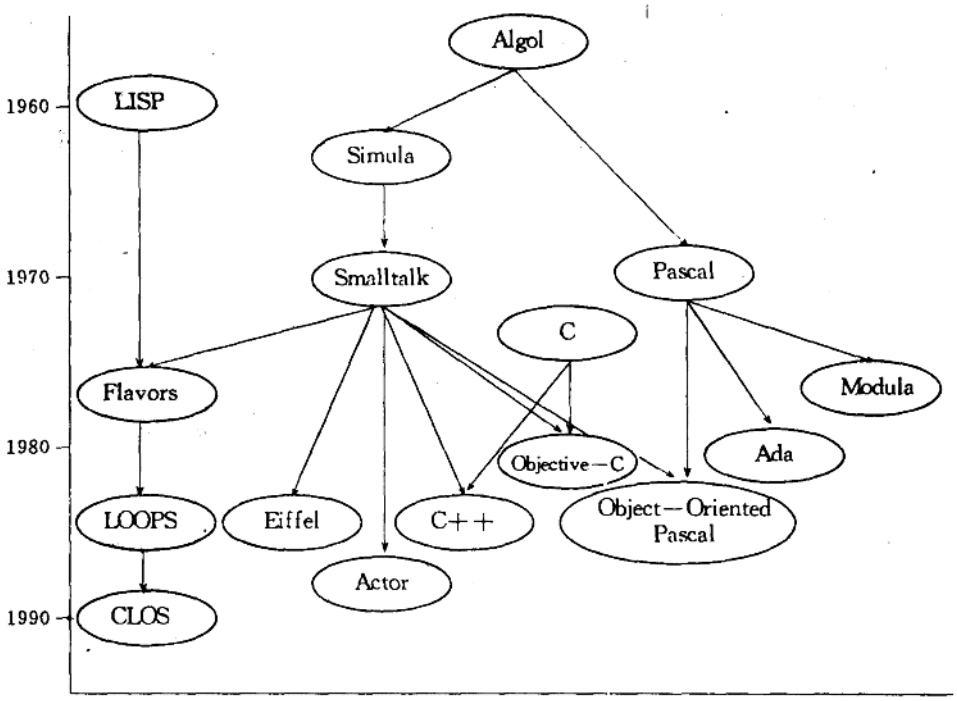

图1.4 面向对象语言的形成

面向对象的语言可以理解为是从基于类的语言加上某些属性而产生的，这种产生方法可用图1.5来表示。即语言特性可视为语言设计空间的设计量纲，这些语言特征包括对象、类、继承性、数据抽象、强类型、并发性与持续性。在图1.5中用带有一对圆边的框表示。

我们必须清楚，面向对象方法并不仅仅指用某种语言系统所能表示的。因为面向对象的技术是一种全新的程序设计范型，这种范型把分析、设计和实现很自然地联系在一起，并且面向对象的技术还在不断的发展和成熟，目前尚无公认的面向对象的语言和面向对象的系统的明确的文本描述和形式定义。因此，目前的面向对象的语言系统(如 Smalltalk 和 C++)都还不能完全胜任这一职责，只能部分地处理相关的问题，但是目前较受青睐的语言要属 C++了。

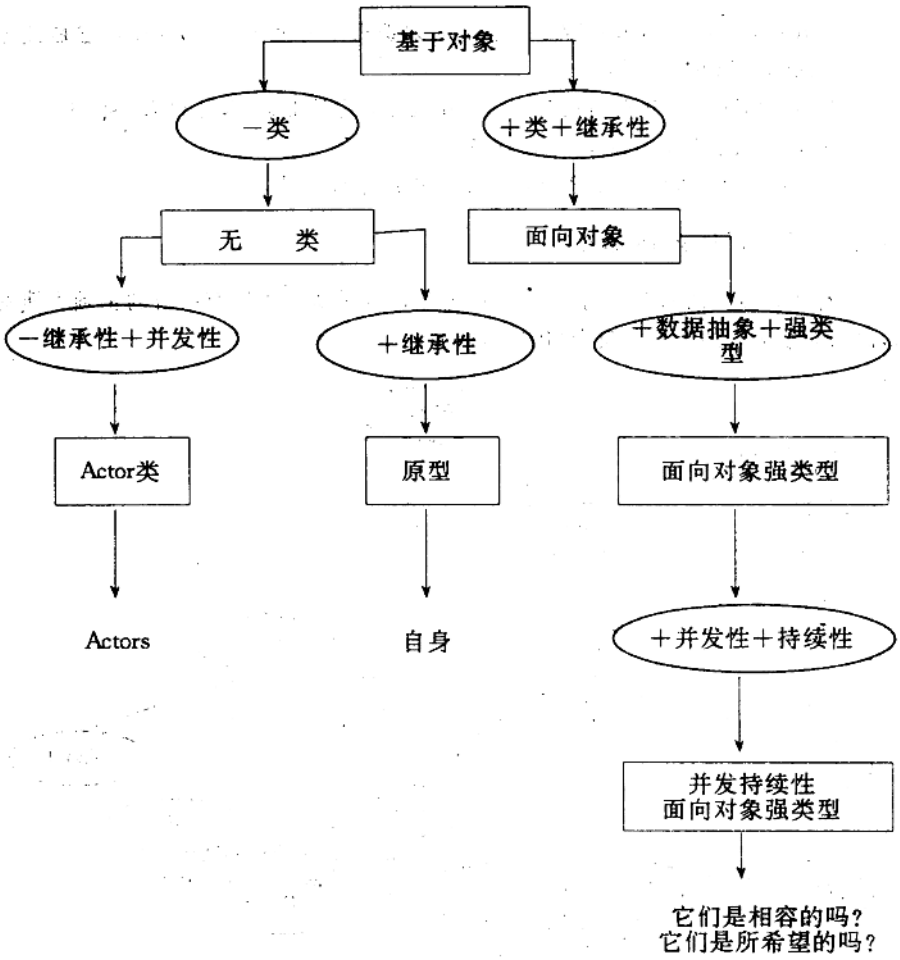

图1.5 基于对象的语言派生图

§1.3 面向对象的思想、概念和特性

1.3.1 面向对象的思想

计算机解决问题的方法实际上是从问题域到问题解空间的一种映射，如果这种映射能以人们通常的思维方式来解决，则可以极大地提高软件的开发效率、可靠性和可维护性。客观世界是由许多不同种类的对象构成，每个对象都有自己的内部状态和运动规律，不同对象间的相互联系和相互作用构成了完整的、千变万化的客观世界。这些对象是离散的、可认识的，因此，可以通过模拟这些对象的行为，即从结构上模拟客观世界。面向对象方法学的基本原则是：按人们通常的思维方式建立问题域的模型，尽可能自然地表现求解

问题的软件设计方法。换言之，面向对象就是不以控制为中心，而以事物（对象）的行为为中心来考虑计算机上的处理体系。这儿所说的事物不仅包括人、器具等物理实体，而且还包括象“销售表”、“库存表”之类的逻辑事物。世界上的任何事物都有其“本身”、“本身能做什么”和“本身能得到什么”这样的意义和功能。把事物的形象（数据）同其意义、功能（过程）一体化，作为处理的基本单位，便是面向对象的。所以，在面向对象中，只要向对象发送出“执行它”这样的消息，则对象便可完成相应的处理。

非面向对象的方法则不同，它把数据和过程截然分开，因此即使规定了数据的详细结构，如果没有记述一个一个的处理过程，那么什么处理也不能进行。

1.3.2 面向对象的概念

为了实现面向对象方法学的上述原则，我们必须建立直接表现组成问题域的事物以及这些事物之间的相互联系的概念。在面向对象的方法中类（Class）、对象（Object）、方法（Method）、属性（Attribute）、消息（Message）和消息传递（Message Passing）分别是表现事物和事物之间联系的概念。下面我们分别对这些概念作以说明：

对象（Object）：对象是数据及可以对这些数据施加的操作结合在一起所构成的具有通信能力的独立实体的总称。

对象从建模的角度讲，它是指问题域的概念抽象或具有明确边界的事物，它是对客观世界事物的表示或描述，世界上任何事物都可以称为对象。

对象是一个十分广泛的概念，它指人认识的所有东西，它直接对应于应用中的概念。如：一本书可以看作一个对象，一个国家可以看作一个对象，一家图书馆可以看作一个对象，一幢房子也是一个对象等等，甚至矩阵、堆栈等都可以是对象。世界上的事物都是由各种各样的对象组成的，它又是某个对象类的一个元素。复杂的对象可以由相对简单的对象组成。对象表现在面向对象的设计中，它是一个建模的实体。而表现在面向对象的程序设计中，对象是视运行时的需要而创建的某个类的实例，是一个动态概念，它是一些基本运行实体，即是具有特殊属性（数据）和行为方式（方法）的实体。

对象具有标识唯一性、分类性、多态性和继承性。

类（Class）：类是一组对象的抽象定义或概括，每个对象都是某个类的一个具体实例（Instance）。

具体地讲，类是对一组具有相同数据和操作的对象的描述（或定义）。所以类可以看作一类对象的集合，是对一组客观对象的抽象，它将该组对象所具有的共同特性（包括操作特性和存储特性）集中起来，以说明该组对象的能力和特征，一个类实质上是一种对象类型，换言之，类描述了该类型的所有对象具有的共同性质。因此，类由数据和方法（亦即操作）两部分组成。

类的一大特性是具有层次结构，一个类的上层可以有超类（或父类）（SuperClass），下层可以有子类（SubClass），这种层次结构的特点是可继承性（Inheritance）。这同人们认识事物的过程完全相同，人们认识事物也正是把事物分类而体系化，类层次结构便相当于人的这种分类。

从程序设计的角度讲，类是一个抽象数据类型的实现，是一个用户定义的类型，它指

出了构造使用某一特定类型的对象所必须的全部信息，因此，类是一个静态概念。而面向对象程序设计的任务就是设计类及由类组装程序。

在C++中类是通过关键字class来定义的，对类中定义的成员的存取控制则是通过关键字private、protected和public来进行的，而成员可以是数据(称为该类的数据成员)，也可以是函数(称为该类的成员函数，用于操作类的数据成员)。

例如C++中queue类的定义如下"：

```
class queue                    //queue为类名
  {
      int   q[100];            //数据成员
      int   sloc,rloc;         //数据成员
    public:                    //存取控制
      void  init(void);        //成员函数
      void  qput(int  i);      //成员函数
      int   qget(void);        //成员函数
  };
```

类可以看作是用户定义的一种类类型，具有类类型的实体就是对象。在说明对象的时候，类被用作"样板"去创建对象，该对象称为类的一个实例。

例如对上例定义的类可如下创建类的对象：

```
queue queue_obj;
```

消息(Message)与消息传递(Message passing)：消息是客观世界中对象之间的通信途径，是对象之间相互请求和相互协作的一种方式。消息为要求某个对象执行类中的某个操作的规格说明书，它通常由三部分构成，即接受消息的对象、消息名、零或多个变元。

消息建立了对象之间的联系，它是对象之间的唯一接口。对象间的接口正如人与人之间的会话，如说"上学去"，则不考虑上学的人是如何处理的。这样的接口方式称为消息传递。也就是说，消息传递是指使对象做什么的唯一手段。显然，它是防止错误使用对象的有效方法，因而可以大大提高软件的可靠性。消息传递与通常的子程序调用类似，但又不同于子程序调用的移交程序控制。

在面向对象程序设计中，数据与对数据的操作地位等同，两者紧密耦合在一起形成对象，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对象自身决定。因此，消息统一了数据流与控制流，一般地把发送消息的对象称为消息发送者，接收消息的对象称为消息接收者。消息的形式用消息模式(Message Pattern)刻划，它是一个十分复杂的概念，值得进一步研究。

消息分为公有消息和私有消息。

在C++中消息的传递是通过成员函数的调用来完成的，C++中的操作可表示成带有属性的类成员，使用相似的表示来访问属性，对于类成员的访问在C++中可通过运算符"."和"->"运算来完成。

例如对于类queue的对象queue_obj的成员qput()可如下访问：

```
queue queue_obj;      //定义类对象
```

```
queue_obj.qput();    //访问对象的成员函数
```

方法(Method):方法是对象所能执行的操作。具体地说,是在对象中定义的函数或过程。通常,对象拥有多个方法,根据从其它对象接收到的消息内容,决定实际使用的方法,也正是由于在实际开始处理的阶段才决定使用的方法,故容易生成柔性软件。方法定义了对象与外界的接口。

方法定义了对象的处理能力。方法是类区别于其它数据类型的一个标志。方法通过类成员函数实现,它分为私有方法与公有方法。C++中的方法是通过成员函数的定义来实现的,所有的成员函数定义了C++中对象所具有的方法(操作)的集合。

如上面例子中类queue所包含的方法有init(),qput()和qget()三个方法。

属性(Attribute):属性是指类中对象所具有的数值特性,它决定了能对对象的内部数据进行的操作及存取权限。

属性用来反映问题域和系统的任务,一个属性是一些数据(状态信息),一般来说每一个对象中均有自己的值,属性的值(状态)将由该对象的方法来处理,属性和对这些属性进行操作的方法看成不可分割的整体,如果系统的某一部分需要访问和处理某个对象中的值,它必须指定一个消息连接到该对象所定义的方法上去,对属性的存取由其控制权限来决定。

C++中属性表现为对象内部的数据。对属性的存取是由类的访问控制private、protected和public来实现的。

1.3.3 面向对象的特性

按对象、类、消息、方法的程序设计范型所构成的软件具有如下特性:

特性1:封装(Encapsulation)性

封装可以定义为:

(1)一个清楚的边界,所有对象的内部软件范围被限定在这个边界内;

(2)一个接口,这个接口描述这个对象和其它对象之间的相互作用;

(3)受保护的内部实现,这个实现给出了由软件对象提供的功能细节,实现细节不能在定义这个对象的类外访问。

简言之,封装性是指把数据和操作数据的过程一体化,制成软件部件。从外界看,一个对象只是一个能接收和发送消息的机制,其内部状态及状态如何变化对外界是不可见的,外界只有通过给它发送消息才能对它产生影响。

封装与类的说明有关,但它同样提供如何将一个问题解的各个组件组装在一起的求精过程。封装的基本单位是对象,对象的性质由它的类说明来描述,并且,这个性质被具有同样类的其它对象所共享。

面向对象的类定义正好实现了这种封装,它使得数据和加工处理该数据的过程紧密结合在一起,具有很强的独立性。

特性2:继承(Inheritance)性

继承性是自动地共享类、子类或对象中的方法和数据的机制。

现实世界中的事物(对象)在一定程度上具有一定的联系(或相似的部分),因而面向

对象的方法中也增加了继承机制，即某一对象可以继承其父类的全部或部分特征，这种机制可用图1.6表示。

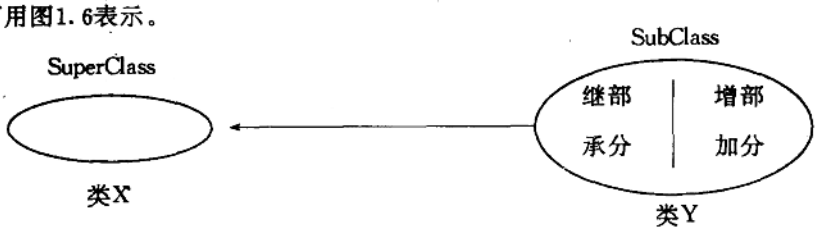

图1.6 继承机制

当类Y继承类X时，则称类Y是类X的子类，而类X则称为类Y的超类(或父类)，类Y由继承部分和增加部分组成，继承部分是从类X继承而来，增加部分是专为类Y编写的新代码。继承性使得程序的代码因大量共享而减少。

每一种面向对象的语言都提供一套实现继承的机制，语言级提供的关键字用来表示期望映射的种类，某些映射还要求一些附加的信息。继承关系常用于反映抽象和结构。

在C++中，一个类从另一个类继承特征称为派生一个类，所生成的新类称为派生类或子类。在C++中有两种继承，即单一继承和多重继承。

C++中的派生类提供了一种简单灵活且有效的机制，派生类的存在使得程序员更容易表达类之间的共性。

例如类stack由类queue派生出来，则可用C++派生类描述如下：

```
class stack:public queue
  {
      stack(int,int);
      int operator_1(void);
  };
```

类stack是派生类，而类queue是基类。

特性3：多态(Polymopnism)性

多态性就是一个名字具有多种语义，或同样的消息可以被送到父类对象和子类对象上。多态性的使用增加了程序的灵活性，实现了同一接口多种方法或同一行为多种实现，故容易生成柔性软件。

或者说多态性是指若类P是类S的超类，s是类S的对象，p是类P的对象，则对象s可以用在对象p可以使用的任何地方。

在面向对象的语言中，多态引用表示可引用多个类的实例，由于多态性具有可表示对象多个类的能力，因而多态性既与动态类型有关，又与静态类型有关。在C++语言中，多态性可由虚函数和函数重载来实现。

特性4：动态聚束(Dynamic Binding)

动态聚束是与多态性和继承性密切相关的实现级的概念，即在程序运行时将执行代码聚束在一起，这样增加了程序代码的效率。动态聚束结合继承和多态使用时，为修改系统所需变动的原代码最少。这非常符合近代软件对可复用、可修改和可扩充的要求。

根据面向对象的这些特性，应用面向对象的方法来设计的软件符合人们自然的思维方式，这样的软件一定具有容易修改、容易扩充和复用性好等特点。

§1.4 面向对象的分析概述

1.4.1 面向对象分析的基本概念

面向对象的分析是面向对象技术中最为重要的组成部分，分析质量的好坏直接关系到软件的成败，而事实上，面向对象的分析是对现实问题域的一种摸拟，是现实问题的一种建模活动。目前面向对象的系统分析最为典型的代表有如下的三大流派：

1. Peter Coad 及 Edward Yourdon 的面向对象的分析与设计思想，以 Coad 的《OOA》和《OOD》两本著作为其代表。

2. James Rumbaugh 等人的面向对象的建模思想，以 James 的《OO Modelling and Design》一书为其代表。

3. Booch 的面向对象的设计思想。以 Booch 的《OOD》一书为其代表。

下面我们先来介绍如下一些相关的概念：

建模：是按照某些概念来建立或组织模型的思考问题的方式。

面向对象的建模：是围绕真实世界的概念来组织模型的方式，其基本构造是对象，面向对象的模型能帮助我们加深对问题域的理解。

面向对象的建模技术：用面向对象(OO)的软件开发方法来组织软件开发的方法学称为面向对象的建模技术(OMT)。

面向对象的建模技术贯穿于软件的分析、设计和实现的始终。分析模型应包括应用域的对象、对象性质及对象行为的描述，在实现及优化中加入设计策略及详情以完善模型。应用域对象组成了设计模型的框架。

面向对象的建模技术通常使用了三种模型来描述软件系统，它们是对象模型、动态模型、功能模型。

对象模型：对象模型是描述系统中的对象及对象之间的关系的一种系统模型。

对象模型可用以对象类为结点，以对象之间的关系为弧线而构成的图来表示，这种图称为对象图。

动态模型：动态模型是描述系统中对象的交互行为的一种系统模型。它描述了不断变化的系统中的各种因素。用来指明和实现系统中的控制因素。

动态模型可用以结点表示状态，弧线为事件触发的状态之间的变迁而构成的图来表示，这种图称为状态图。

功能模型：功能模型是描述系统中数值的变化过程的一种系统模型。它指明系统发生了什么。

功能模型可用以结点表示处理，以弧线表示数据的流向而构成的图来表示，这种图称为数据流图(DFD)。

上述三种模型相辅相承组成一个完整的系统正交视图。各模型描述的是系统的不同