第5章 面向对象方法学

5.1 面向对象概述

对象是由数据和允许的操作组成的封装体，与客观实体有直接对应关系。一个对象类定义了具有相似性质的一组对象。面向对象的思想最初出现于挪威奥斯陆大学和挪威计算机中心共同研制的 Simula 67 语言中。随着的 Smalltalk 76 和 80 语言的推出，面向对象的程序设计方法得到了比较完善的实现。20 世纪 70 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了出来，如对于系统中的某个模块或构件，可表示为问题空间的一个对象或一类对象。到了 20 世纪 80 年代，面向对象的程序设计方法得到了很快的发展，并显示出其强大的生命力，因此使面向对象技术在系统工程、计算机、人工智能等领域得到了广泛的应用。

谈到面向对象，起初，“面向对象”专指在程序设计中采用封装、继承、抽象等设计方法。可是，这个定义显然不能再适合现在的情况。面向对象的概念和应用已超越了程序设计和软件开发，扩展到如数据库系统、交互式界面、应用结构、应用平台、分布式系统、网络管理结构、CAD 技术和人工智能等领域。在更高的层次上、更广泛的领域内开展面向对象技术的热点研究始于 20 世纪 90 年代。目前，这种技术已得到了广泛的应用，面向对象方法已成为软件工程的一种新途径、新方法、新技术，如表 5.1 所示。

表 5.1 面向对象技术的发展历程

阶段	内　　容
初始阶段	20 世纪 60 年代：Simula 编程语言 20 世纪 70 年代：Smalltalk 编程语言
发展阶段	20 世纪 80 年代：理论基础已形成，出现了许多面向对象编程语言，如 C++、Objective-C 等
成熟阶段	20 世纪 90 年代：面向对象分析和设计方法（Booch、OMT、OOSE 等），Java 语言 1997 年：OMG 组织的统一建模语言

早期的OO更多的是指使用的一系列面向对象程序设计语言的软件开发方法。现今，面向对象方法包含完整的软件工程过程。Edward Berard曾给出这样的评论："如果在软件工程过程的早期和全程采用面向对象技术，则该技术将产生更多的优势。那些考虑面向对象技术的人们必须评估它对整个软件工程过程的影响。仅仅使用面向对象程序设计将不会产生最好的结果，软件工程师及其管理者必须考虑面向对象需求分析、面向对象设计、面向对象领域分析、面向对象数据库管理系统和面向对象计算机辅助软件工程等。"

面向对象技术强调在软件开发过程中面向客观世界或问题域中的事物，采用人类在认识客观世界的过程中普遍运用的思维方法，直观、自然地描述客观世界中的有关事物。所谓面向对象，就是基于对象概念，以对象为中心，以类和继承为构造机制，来认识、理解、刻画客观世界，设计、构建相应的软件系统。

在研究对象时主要考虑对象的属性和对象的操作，有些不同对象会呈现相同或相似的属性和操作，例如大巴车、轿车、越野车等。通常将属性及操作相同或相似的对象归为一类。类可以看成是对象的抽象，代表了此类对象所具有的公共属性和操作。在面向对象的程序设计中，每个对象都属于某个特定的类，如同结构化程序设计中每个变量都属于某个数据类型一样。类声明要包括类的属性和类的操作。

Yourdon给出的定义为"面向对象＝对象＋类＋继承＋通信"。一般情况下，采用这四个概念开发的软件系统是面向对象的。面向对象程序的基本组成单位是类。程序在运行时由类生成对象，对象之间通过发送消息进行通信，互相协作完成相应的功能。对象是面向对象程序的核心。

面向对象方法现已成为主流开发方法，不论在管理层面，还是在技术层面均具有优势。可以从下列三方面来分析其原因：

(1) 从认知学的角度来看，面向对象方法符合人们对客观世界的认识规律；

(2) 面向对象方法开发的软件系统易于维护，其高内聚、低耦合的体系结构易于理解、扩充和修改；

(3) 面向对象方法中的继承机制可有力支持软件的复用，使软件开发速度更快，软件程序质量更高。

5.1.1 面向对象的基本概念

1. 对象

对象是客观事物或概念的抽象表述，即对客观存在的事物的描述统称为对象。客观世界中任何有确定边界、可触摸、可感知的事物以及可思考或者可认知的概念都可以认为是对象。对象是人们要进行研究的任何事物，从最简单的整数到复杂的飞机等均可看作对象，它不仅能表示具体的事物，还能表示抽象的规则、计划或事件，是将一组数据和使用该数据的一组基本操作或过程封装在一起的实体。

(1) 对象的属性。对象的属性通常是一些数据，是对象本身的性质，有时它也可以是另一个对象。每个对象都有它自己的属性值，可以由施加于该对象上的行为动作而变更，表示该对象的状态。对象中的属性只能通过该对象所提供的操作来存取或修改。

(2) 对象的操作。对象的操作(也称方法或服务)规定了对象的行为，表示对象所能提供的服务。给对象定义一组运算，用于改变对象的状态。对象实现了数据和操作的结合，

使数据和操作封装于对象的统一体中。

对象将它自身的属性及运算“包装起来”，称为“封装”。一个对象通常可由对象名、属性和操作三部分组成。对象最基本的特征是封装性和继承性。

系统中的一个对象，在软件生命周期的各个阶段可能有不同的表现形式。在分析阶段，对象主要是从问题域中抽象出来，反应概念的实体对象；在设计阶段，主要结合实际环境增加了用户界面对象和数据存储对象；到了实现阶段，主要用一种程序设计语言写出来详细的源程序代码。这说明，系统中的对象要经过若干演化阶段，其表现重点和形式不同，但在概念上是一致的——都是问题域中的某一事物的抽象表示。

2. 类

类又称对象类，是一组具有相同属性和相同操作的对象的集合。类代表一种抽象，是具有相似特征和共同行为的对象的模板，可以用来生成对象。因此，对象的抽象是类，类的具体化就是对象，也可以说类的实例是对象。在一个类中，对象都可以使用类中提供的函数。类是创建对象的模板，从同一个类实例化的每个对象都具有相同的结构和行为。

(1) 类的属性。类具有属性，它是对象状态的抽象，用数据结构来描述类的属性。

(2) 类的操作。类具有操作，它是对象行为的抽象，操作实现的过程称为方法。方法有方法名、方法体和参数。类的操作用操作名和实现该操作的方法来描述。

由于对象是类的实例，在进行分析和设计时，通常把注意力集中在类上，而不是具体的对象上。

类一般采用“类图”来描述，如图 5.1 所示。

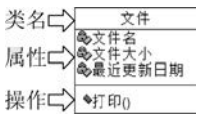

图 5.1　类图

在客观世界中有若干类，这些类之间有一定的结构关系。通常有两种主要的结构关系，即“一般-特殊”结构关系和“整体-部分”结构关系。

“一般-特殊”结构称为分类结构，也可以说是“或”关系或者是 is a 关系，如图 5.2(a)所示。

“整体-部分”结构称为组装结构，它们之间是一种“与”关系或者是 has a 关系，如图 5.2(b)所示。

图 5.2　类之间的结构关系

3. 消息和方法

消息就是向对象发出的服务请求(互相联系、协同工作等)。对象之间的联系可表示为对象间的消息传递，即对象间的通信机制。

一个消息应该包含以下信息：消息名、接收消息对象的标识、服务标识、消息和方法、输

入信息、回答信息。发送一条消息至少要包括接收消息的对象名、发送给该对象的消息名(即对象名、方法名)。一般还要对参数加以说明,参数可以是认识该消息的对象所知道的变量名或者是所有对象都知道的全局变量名。

在对象的操作中,当一个消息发送给某个对象时,消息包含接收对象去执行某种操作的信息,但并不指示接收者怎样完成操作,消息完全由接收者解释执行。

4. 抽象类

抽象类是没有实例的类,它把一些类组织起来,提供一些公共的行为,但并不需要使用这个类的实例,而仅使用其子类的实例。

在抽象类中可以定义抽象操作,抽象操作是指只定义这个类的操作接口,不定义它的实现,其实现部分由其子类定义。抽象操作的操作名用斜体字表示,也可以在操作特征后面加上特征字符串,如图 5.3 所示。

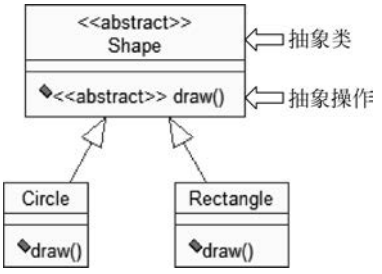

图 5.3 抽象类与子类示例

5. 永久对象

永久对象指生命周期可以超越程序的执行时间而长期存在的对象。

目前,大多数面向对象程序设计语言不支持永久对象。如果一个对象要长期保存,必须依靠于文件系统或数据库管理系统实现。程序员需要完成对象与文件系统或数据库之间数据格式的转换,以及保存和恢复所需的操作等烦琐的工作。

要实现永久对象,使上述烦琐工作由系统自动完成,需要较强的技术支持。如永久对象管理系统和能够描述和处理永久对象的编程语言。

5.1.2 面向对象技术的基本特征

面向对象技术的基本特征主要是抽象性、封装性、继承性、多态性和动态绑定。

1. 抽象性

抽象是指强调实体的本质、内在的属性。在系统开发中,抽象指的是在决定如何实现对象之前的对象的意义和行为。使用抽象可以尽可能避免过早考虑一些细节。类实现了对象的数据(即状态)和行为的抽象。

2. 封装性

面向对象技术的封装特征和抽象特征紧密相关。封装是一种信息隐蔽技术,就是利用抽象数据类型将数据和基于数据的操作封装在一起。用户只能看见对象封装界面上的信息,对象的内部实现对用户是隐蔽的。封装的目的是使对象的使用者和生产者分离,使对象的定义和实现分开。

封装是保证软件部件具有优良模块性的基础。面向对象的类是封装良好的模块,类定义将其说明(用户可见的外部接口)与实现(用户不可见的内部实现)显式地分开,其内部实现按其具体定义的作用域提供保护,尽可能隐蔽对象的内部细节。对象是封装的最基本单位。封装防止了程序相互依赖性而带来的变动影响。

所以封装具有两个基本前提条件:其一,对象的完整性,即对象的概念可以描述整个问题的各个方面,包括系统边界、接口等;其二,对象的私有性,即封装技术,对象内部的设计

细节，包括数据和操作代码都被封装在边界内部，外部是看不到的。对象与对象之间只能通过定义的接口消息进行通信，这样可以有效地保护内部实现。

面向对象的封装比传统语言的封装更为清晰，更为有力。在面向对象的程序设计中，抽象数据类型是用“类”来实现的，类封装了数据以及对数据的操作，是程序中的最小模块。由于封装特性禁止了外界直接操作类中的数据，模块与模块之间只能通过严格控制的接口进行交互，这可以大大减低模块之间的耦合度，从而保证了模块具有较好的独立性，使得程序的维护和修改较为容易。

3. 继承性

继承性是使用现存的定义作为基础，建立新定义的技术。继承是类与类之间的基本关系，它是基于层次关系的父类和子类之间共享数据和操作的一种机制。父类中定义了其所有子类的公共属性和操作，在子类中除了定义自己特有的属性和操作外，可以继承其父类（或祖先类）的属性和操作，还可以对父类（或祖先类）中的操作重新定义其实现方法。继承关系如图 5.4 所示。

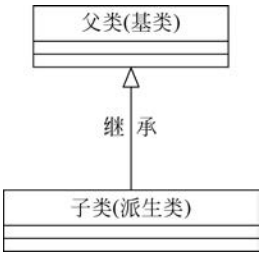

图 5.4　继承关系

在类层次中，继承可分为单重继承和多重继承两种。

在单重继承中，一个子类只有一个父类，即子类只继承一个父类的数据结构和方法，如图 5.5 所示。

在多重继承中，一个子类可有多个父类，即子类继承多个父类的数据结构和方法，如图 5.6 所示。

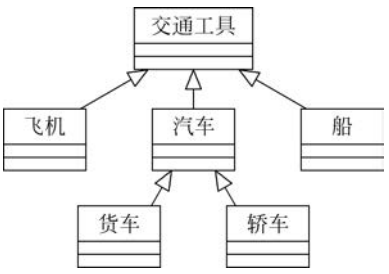

图 5.5　单重继承

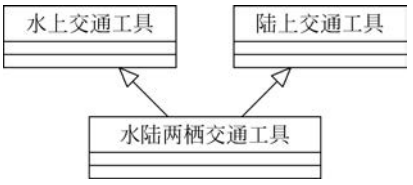

图 5.6　多重继承

在面向对象程序设计中，可使用继承方法来设计两个或者多个不同的但具有很多共性的实体。继承是一种联结类的层次模型，为类的重用提供了方便，也提供了明确表述不同类之间共性的方法。

在软件开发中，类的继承性使所建立的软件具有开放性、可扩充性，这是信息组织与分类行之有效的方法。继承简化了人们对现实世界的认识和描述，在定义子类时不必重复定义那些已在父类中定义过的属性和服务，只要说明它是其父类的子类并定义自己特有的属性和服务即可。它简化了对象、类的创建工作量，增加了代码的可重性。继承性提供了类的规范的等级结构。通过类的继承关系，公共的特性能够共享，提高了软件的复用性。

4. 多态性和动态绑定

（1）多态性。一般来讲，多态就是多种形态，是指同一个操作作用于不同的对象上可以

有不同的解释,并产生不同的执行结果。多态性实际上提供了一种具体情况具体分析的问题解决方案。具体来说,多态性是指相同的操作或函数、过程作用于不同的对象上并获得不同的结果。即相同的操作消息发送给不同的对象时,每个对象将根据自己所属类中定义的操作去执行,产生不同的结果。

在面向过程程序设计中,要求所有编写的过程和函数是不能重名的。例如,在一个应用程序中,要求分别对数值型数据和字符型数据进行排序。虽然针对不同数据类型的数据进行排序的方法是相同的,但是定义了不同的过程来实现。

在面向对象程序设计中,利用重名可以提高程序的抽象度和简洁性。例如,“支付”的操作名称相同,但是支付方式的具体实现是不同的,可以使用“现金支付”“支票支付”“POS机支付”以及“电子网银支付”等多个实现方法。

多态性允许每个对象以适合自身的方式去响应共同的消息,增强了软件的灵活性和复用性。

【案例 5-1】 用绘图操作实现多态性。

将绘图操作作用在“椭圆”和“矩形”上,画出不同的图形。实现多态性的基本步骤如表 5.2 所示。

表 5.2 实现多态性的基本步骤(以 VC 为例)

(1) 在基类中,定义成员函数为虚函数;
(2) 定义基类的公有派生类;
(3) 在基类的公有派生类中“重载”该虚函数;
(4) 定义指向基类的指针变量,它指向基类的公有派生类的对象。
注意:重载虚函数不是一般的重载函数,它要求函数名、返回类型、参数个数、参数类型和顺序完全相同。

用绘图操作实现 figure 类的类图,如图 5.7 所示。

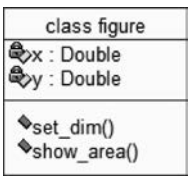

图 5.7 figure 类

用绘图操作实现 figure 类的实例代码如下:

```
#include <iostream.h>
class figure                              //定义基类
{
    protected:
        double x,y;
    public:
        void set_dim(double i; double j = 0)
        { x = i; y = j; }
        virtual void show_area()          //(1)定义虚函数
        {
            cout <<"No area computation define ";
            cout <<"for this class.\n";
        }
};
class triangle:public figure              // (2)定义基类的公有派生类
{
    public:
    virtual void show_area()              // (3)"重载"该虚函数
```

```
    { 求三角形面积 }
};
class square:public figure                    // (2)定义基类的公有派生类
{
    public:
    virtual void show_area()                  // (3)"重载"该虚函数
    { 求矩形面积 }
};
class circle:public figure                    // (2)定义基类的公有派生类
{
    public:
    virtual void show_area()                  // (3)"重载"该虚函数
    { 求圆面积 }
};
void main()                                   //(4)运行过程中实现"动态绑定"
{
figure * p;                                   // 定义指向基类的指针变量
    triangle t;
    square s;                                 // 定义基类的公有派生类的对象
    circle c;
    p = &t;                                   // 指向三角形对象
    p -> set_dim(10.0,5.0);
p -> show_area();
p = &s;                                       // 指向矩形对象
p -> set_dim(10.0,5.0);
    p -> show_area();
    p = &c;                                   // 指向圆形对象
    p -> set_dim(9.0);
    p -> show_area();
}
```

应用程序不必为每个派生类编写功能调用，只需要对抽象基类进行处理即可，这可以大大提高程序的可复用性（这是接口设计的复用，而不是代码实现的复用）。

派生类的功能可以被基类指针引用，这叫向后兼容，可以提高程序的可扩充性和可维护性。

（2）动态绑定。动态绑定是在运行时根据对象接收的消息动态地确定要连接的服务代码。使用虚函数可实现动态联编，不同联编可以选择不同的实现，这便是多态性。继承是动态联编的基础，虚函数是动态联编的关键。

在一般与特殊关系中，子类是父类的一个特例，所以父类对象可以出现的地方，也允许其子类对象出现。因此在运行过程中，当一个对象发送消息请求服务时，要根据接收对象的具体情况将请求的操作与实现的方法进行连接，即动态绑定。

5.2 面向对象开发方法概述

面向对象开发方法（Object Oriented Software Development，OOSD）是一种把面向对象的思想应用于软件开发过程中，指导开发活动的系统方法，简称 OO 方法，是建立在“对象”概念基础上的方法学。

5.2.1 软件开发过程

软件开发过程就是将软件系统所涉及的应用领域和业务范围(现实世界)的问题空间映射到用于解决某些问题的软件系统的解空间。问题空间和解空间的映射如图 5.8 所示。

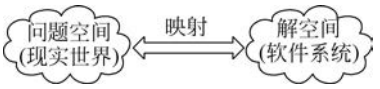

图 5.8 问题空间和解空间的映射

在面向对象软件方法出现之前,软件工程实践中通常使用基于数据流图、自顶向下的层次分解等方法。面向对象方法和以往的基于过程的方法是不矛盾的。实际上,使用面向对象方法是将基于过程的方法带到一个更有效的使用环境下。但是,使用面向对象方法确实需要软件开发者的思维模式发生变化:面向对象方法使用类来模塑概念——从而发现或创造出许多类,刻画类的粒度,清晰定义类的职责,让类之间互相协作;程序设计的基本思维从一个函数调用另一个函数转换到如何对真实世界进行建模上,这样的转换确实不是一个小的转换。

面向对象方法把问题域作为一系列相互作用的对象,在此基础上构造出基于对象的软件系统结构。面向对象方法作为一种新型的独具优越性的新方法正引起全世界越来越广泛的关注和高度的重视,它被誉为"研究高技术的好方法",更是当前计算机界关心的重点。

面向对象开发方法包括面向对象分析(OOA)、面向对象设计(OOD)和面向对象编程实现(OOP)。在进行面向对象系统开发时是按照 OOA—OOD—OOP 的顺序进行的,但面向对象方法是按照 OOP—OOD—OOA 的顺序逐渐发展成熟起来的。

目前,面向对象开发方法的研究已日趋成熟,国际上已有不少面向对象的产品。面向对象开发方法有 Booch 方法、Coda/Yourdon 方法、OMT 方法和 OOSE 方法等。

面向对象的软件开发过程如图 5.9 所示。

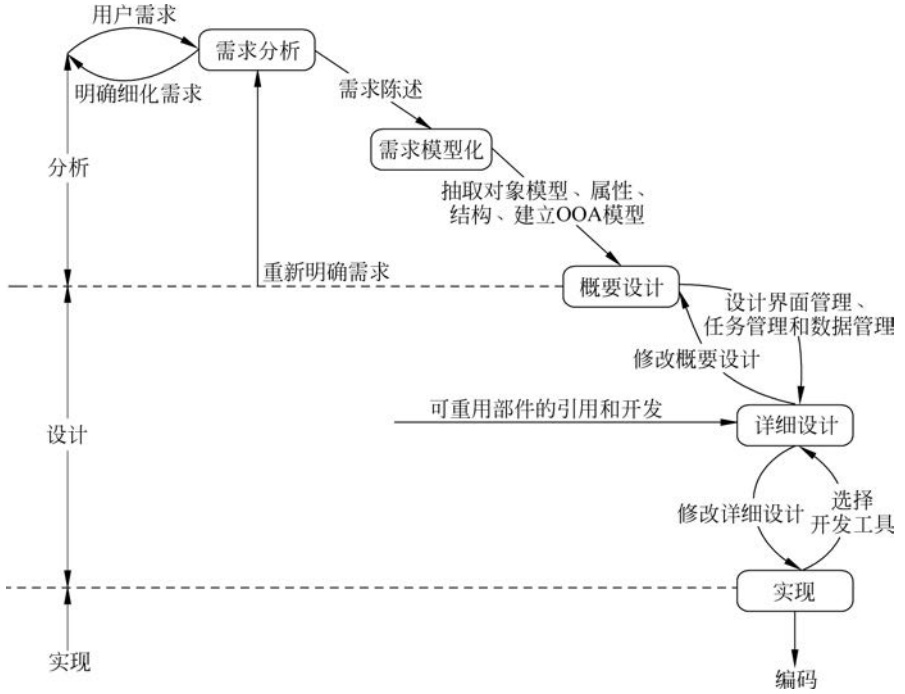

图 5.9 面向对象的软件开发过程

5.2.2 传统开发方法存在的问题

传统的设计方法将问题域分解成一系列任务来完成，这些任务形成过程式软件的基本结构。传统结构化技术的缺点是：软件结构分析与结构设计技术的本质是功能分解，是围绕实现处理功能的过程来构造系统的。结构化方法强调过程抽象和模块化，是以过程（或操作）为中心来构造系统和设计程序的。然而用户需求的变化大部分是针对加工的，因此，这种变化对基于过程的设计来说是灾难。

传统软件开发方法存在的问题如下。

（1）问题空间不能直接映射到解空间。传统软件开发方法无法实现从问题空间到解空间的直接映射，如图 5.10 所示。

（2）软件复用程度低。复用性是指同一事物不经修改或稍加修改就可多次重复使用的性质。软件复用性是软件工程追求的目标之一。传统软件开发方法无法实现高效的软件复用，因为传统软件开发方法数据与代码（操作）是分离的。

（3）分析不能直接过渡到设计。传统软件开发方法难以实现从分析到设计的直接过渡，其分析到设计的转换如图 5.11 所示。

图 5.10　传统开发方法中问题空间到解空间的映射

图 5.11　传统开发方法中从分析到设计的转换

（4）软件可维护性差。软件工程强调软件的可维护性，强调文档资料的重要性，规定最终的软件产品应该由完整、一致的配置成分组成。在软件开发过程中，始终强调软件的可读性、可修改性和可测试性是软件的重要的质量指标。实践证明，用传统方法开发出来的软件，维护时其费用和成本仍然很高，其原因是可修改性差，维护困难，导致可维护性差。

（5）软件不满足用户需要。用传统的结构化方法开发大型软件系统涉及各种不同领域的知识。在开发需求模糊或需求动态变化的系统时，所开发的软件系统往往不能真正满足用户的需要。

用结构化方法开发的软件，其稳定性、可修改性和可复用性都比较差，这是因为结构化方法的本质是功能分解。在结构化方法中首先考虑的是过程的抽象，从代表目标系统整体功能的单个处理着手，自顶向下分解下去，直到仅剩下若干容易实现的子功能为止，然后用相应的工具来描述各个最低层的处理。

然而，用户需求的变化大部分是针对功能的，因此，这种变化对于基于过程的设计来说是灾难性的。用这种方法设计出来的系统结构常常是不稳定的，用户需求的变化往往造成系统结构的较大变化，从而需要花费很大代价才能实现这种变化。

5.2.3 面向对象开发方法的特点

（1）对软件开发过程的所有阶段进行综合考虑。综合考虑可使问题空间与解空间具有一致性，降低复杂性。

（2）具有高度的连续性。软件生命周期各阶段所使用的方法、技术具有高度的连续性，用符合人类认识世界的思维方式来分析、解决问题。

(3) 增强系统稳定性。将 OOA、OOD 以及 OOP 有机地集成在一起,有利于系统的稳定性。以对象为中心来构造系统,而不是以功能为中心,能很好地适应需求变化,使得软件系统具有较好的稳定性和可适应性。

(4) 具有良好的可复用性。对象具有封装性和信息隐蔽性,具有很强的独立性。

5.2.4 Booch 方法

Booch 方法描述了面向对象的软件开发方法的基础问题,指出面向对象开发是一种根本不同于传统的功能分解的设计方法。面向对象的软件分解更接近人对客观事务的理解,而功能分解只通过问题空间的转换来获得。

在面向对象分析中,Booch 方法从用来说明应用问题的词法和概念中识别对象,通过对具体对象的抽象化来发现类。类和对象的识别包括找出问题空间中关键的抽象和产生动态行为的重要机制。开发人员可以通过研究问题域的术语发现关键的抽象。语义的识别主要是建立前一阶段识别的类和对象在完成系统功能上应承担的责任和所起到的作用,在这个基础上确定类的行为(即方法)和类及对象之间的互相作用(即行为的规范描述)。密切相关的一些对象协同作业,可完成部分的系统功能,同时也构成系统的一个必要组成部分,Booch 称之为"机构"。该阶段利用状态机图描述对象的状态模型,利用时序图(系统中的时态约束)和对象图(对象之间的互相作用)描述系统的动态行为模型。这些活动不仅是一个简单的步骤序列,而且是对系统的逻辑和物理视图不断细化的迭代和渐增的开发过程。

在面向对象设计方法中,Booch 方法说明每个类的界面及实现,同时将类和对象分配到不同的模块中,将可同时执行的进程分配到不同的处理机上。这是对已有定义的细化和完善过程,往往有助于发现新的类和对象。

Booch 方法的开发模型包括静态模型和动态模型。静态模型分为逻辑模型和物理模型,描述了系统的构成和结构;动态模型分为状态机图和时序图。Booch 方法强调基于类和对象的系统逻辑视图与基于模块和进程的系统物理视图之间的区别。然而,Booch 方法偏向于系统的静态描述,对动态描述支持较少。

在 Booch 方法丰富的符号体系中,用于类和对象建模的符号体系使用注释和不同的图符(如不同的箭头)表达详细的信息。Booch 方法建议在设计的初期可以用符号体系的一个子集,随后不断添加细节。每一个符号体系还有一个文本的形式,由每一个主要结构的描述模板组成。符号体系由大量的图符定义,但是其语法和语义并没有严格的定义。

Booch 方法对每一步都作了详细的描述,描述手段丰富、灵活,它不仅建立了开发方法,还提出了对设计人员的技术要求和不同开发阶段的资源人力配置。

5.2.5 Coda/Yourdon 方法

Coda/Yourdon 方法是 1989 年 Coda 和 Yourdon 提出的面向对象开发方法,即著名的面向对象分析/设计(OOA/OOD),它是最早的面向对象分析和设计方法之一。

1. Coda/Yourdon 方法的面向对象分析

Coda/Yourdon 方法对复杂问题建立问题域的分析模型,构造和评审 OOA 概念模型的顺序由五个层次组成。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,即分析的不同侧面。这五个层次是:主题层、类与对象层、结构层、属性层和服务层。

（1）主题层。主题给出分析模型的总体概貌，是控制开发人员和读者在同一时间所能考虑的模型规模的机制。

（2）类与对象层。对象是数据及其处理的抽象，它反映了保存有关信息和与现实世界交互的能力。

（3）结构层。结构表示问题域的复杂性。类-成员结构反映了一般-特殊关系，整体-部分结构反映了整体-部分的关系。

（4）属性层。属性是数据元素，用来描述对象或分类结构的实例，可在图中给出并在对象的储存中指定，即给出对象定义的同时指定属性。

（5）服务层。服务是接收到消息后必须执行的一些处理，可在图上标明它并在对象的储存中指定，即给出对象定义的同时定义服务。

面向对象模型的五个层次对应着分析建模的五个主要活动，这五个活动的工作可以不按顺序进行。OOA 利用五个层次和活动定义和记录系统行为、输入和输出。经过五个层次活动后的结果是一个分成五个层次的问题域模型，用类及对象图表示。

当系统分析人员在确定类-对象时想到该类的服务，则可以先确定服务后，再返回去继续寻找类-对象，没有必要遵循自顶向下、逐步求精的原则。

2. Coda/Yourdon 方法的面向对象设计

Coda/Yourdon 方法的 OOD 模型是在 OOA 模型五个层次的基础上，建立四个组元的设计模型：问题域组元、人机交互组元、任务管理组元和数据管理组元。Coda/Yourdon 方法的 OOD 模型如图 5.12 所示。

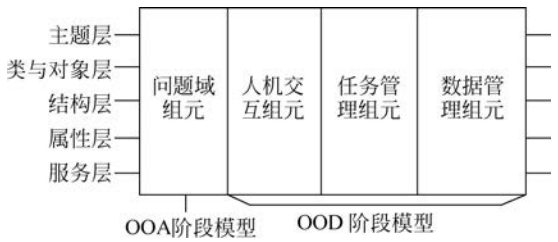

图 5.12　Coda/Yourdon 方法的面向对象设计模型

Coda/Yourdon 方法的主要优点是通过多年来大系统开发的经验与面向对象概念的有机结合，在对象、结构、属性和操作的认定方面，提出了一套系统的原则。该方法简单、易学，对于对象、结构、服务的认定较系统、完整，可操作性强。该方法从需求角度进一步进行了类和类层次结构的认定。

5.2.6　OMT 方法

面向对象的方法学是 1991 年由 UML 创始人 James Rumbaugh 等五人提出来的，又称为对象模型技术（OMT），是一种软件工程方法学，支持整个软件生命周期，它覆盖了问题构成、分析、设计和实现等阶段。OMT 方法开发工作的基础是对真实世界的对象建模，然后围绕这些对象使用分析模型来进行独立于语言的设计。面向对象的建模和设计促进了对需求的理解，软件开发人员不必在开发过程的不同阶段进行概念和符号的转换，有利于开发得清晰、更容易维护的软件系统。OMT 方法为大多数应用领域的软件开发提供了一种

实际的、高效的问题求解方法。

OMT 方法使用了建模的思想，讨论如何建立一个实际的应用模型，从三个不同而又相关的角度建立了三类模型：对象模型、动态模型和功能模型。对象模型描述对象的静态结构和它们之间的关系，主要的概念包括类、属性、操作、继承、关联（即关系）和聚合等；动态模型描述系统那些随时间变化的方面，其主要概念有状态、子状态、超状态、事件、行为和活动等；功能模型描述系统内部数据值的转换，其主要概念有加工、数据存储、数据流、控制流和角色等。OMT 方法为每一个模型都提供了图形表示。

OMT 方法讨论的核心就是建立三类模型，三类模型描述的角度不同，却又相互联系。

1. 对象模型

几乎解决任何一个问题，都需要从客观世界实体及实体间相互关系抽象出极有价值的对象模型。

对象模型描述系统的数据结构，是三个模型中最基础、最核心、最重要的。对象模型描述了由对象和相应实体构成的系统静态结构，包括构成系统的类和对象、它们的属性和操作以及它们之间的联系。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。

构成对象模型的基本元素有对象（类）和它们之间的关系。

【案例 5-2】 银行网络 ATM 系统

银行网络 ATM 系统的对象模型实型如图 5.13 所示。

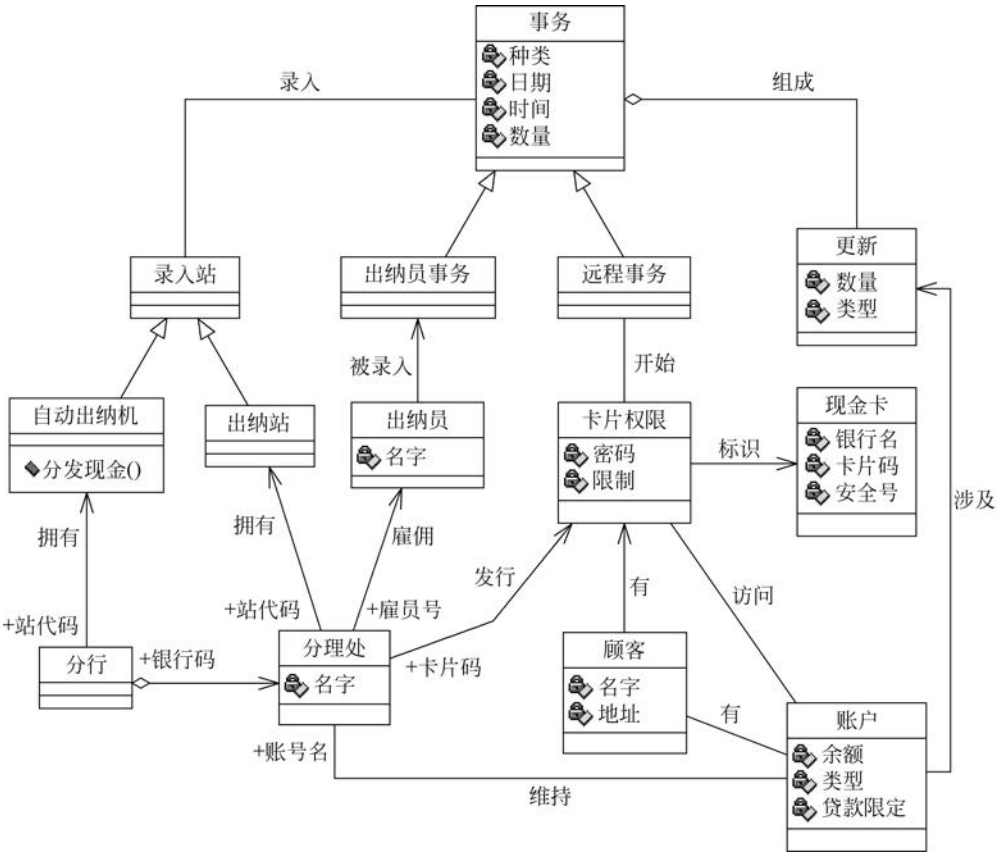

图 5.13 建立对象模型

2. 动态模型

动态模型根据事件和状态描述了系统的控制结构、系统中与时间和操作顺序有关的内容。动态模型着重于系统的逻辑结构，描述某时刻对象及其联系的改变。动态模型描述了系统的交互次序，当问题涉及交互作用和时序时（例如用户界面及过程控制等），动态模型是重要的。

动态模型包括状态机图和时序图。

（1）事件和状态。

事件：对于对象的触发行为，指从一个对象到另一个对象的信息的单向传递。

状态：对象所具有的属性值，具有时间性和持续性。

脚本：在系统某一执行期间内的一系列事件。

在系统中具有属性值、链路的对象，可能相互激发，引起状态的一系列变化。有的事件传递的是简单信号，有的事件则传递的是数据值。由事件传送的数据值称为“属性”。

（2）状态机图。状态机图是一个状态和事件的网络，侧重于描述每一类对象的动态行为和状态的迁移。

动态模型由多个状态机图组成，每个有重要行为的类都有一个状态机图。各状态机图可并发地执行及独立改变状态。

【案例 5-3】 打电话

“打电话”的状态机图如图 5.14 所示。

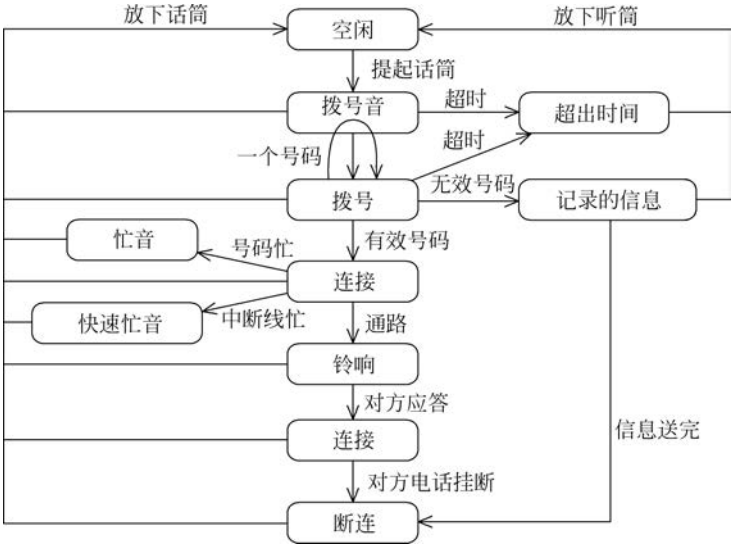

图 5.14　打电话的状态机图

（3）时序图。时序图侧重描述系统执行过程中的一个特定“场景”。场景有时也叫“脚本”，是完成系统某个功能的一个事件序列，用来描述多个对象的集体行为。

“打电话”的场景如图 5.15 所示。

“打电话”的时序图如图 5.16 所示。

1. 拿起电话受话器
2. 电话忙音开始
3. 拨电话号码数 7
4. 电话忙音结束
5. 拨电话号码数 6
6. 拨电话号码数 2
7. 拨电话号码数 6
⋮
11. 对方电话开始振铃
12. 打电话者听见振铃声
13. 对方接电话
14. 接话方停止振铃
15. 打电话方停止振铃声
16. 通电话
17. 对方挂电话
18. 电话切断
19. 打电话者挂电话

图 5.15 打电话的场景

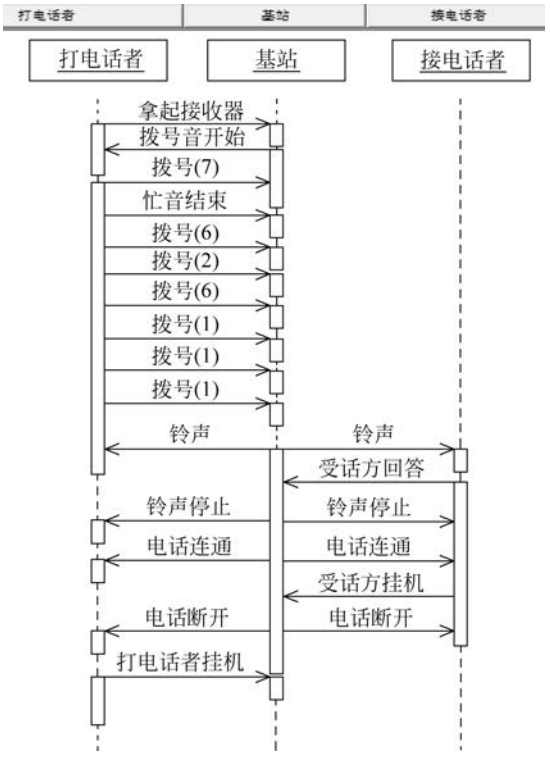

图 5.16 打电话的时序图

3. 功能模型

功能模型着重于系统内部数据的传递与处理，如函数、映射、约束和函数作用等。功能模型定义“做什么”的问题，表明值之间的依赖关系及其相关的功能。它描述了系统的数据变换。如果问题涉及大量数据变换，则功能模型非常重要。

功能模型的描述手段为分层数据流图。数据流图有助于表示功能的依赖关系，其中的处理对应于状态机图的活动和动作，数据流对应于对象图中的对象或属性。

要解决运算量很大的问题（例如高级语言编译、科学与工程计算等）时，则涉及重要的

功能模型。动态模型和功能模型中都包含了对象模型中的操作，即服务或方法。

OMT 方法将开发过程分为四个阶段。

（1）分析阶段。分析阶段基于问题和用户需求的描述，建立现实世界的模型，主要产物如下：

① 问题描述；

② 对象模型＝对象图＋数据词典；

③ 动态模型＝状态机图＋全局事件流图；

④ 功能模型＝数据流图＋约束。

（2）系统设计阶段。系统设计阶段结合问题域的知识和目标系统的体系结构（求解域），将目标系统分解为子系统。该阶段的主要产物如下。

系统设计文档：基本的系统体系结构和高层次的决策。

（3）对象设计阶段。对象设计阶段基于分析模型和求解域中的体系结构等添加的实现细节完成系统设计，主要产物如下：

① 细化的对象模型；

② 细化的动态模型；

③ 细化的功能模型。

（4）实现阶段。实现阶段将设计转换为特定的编程语言或硬件，同时保持可追踪性、灵活性和可扩展性。

面向对象建模得到的模型包含系统的三个要素，即对象模型、动态模型和功能模型。解决的问题不同，这三个子模型的重要程度也不同。三类模型描述的角度不同，却又相互联系。三个子模型分别从不同角度分析系统，如图 5.17 所示。

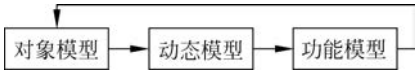

图 5.17　三个模型分别从不同角度分析系统

5.2.7　OOSE 方法

OOSE 方法是 Jacobson 于 1994 年提出的。OOSE 的开发活动主要分为三类：分析、构造和测试。OOSE 将面向对象的思想应用于软件工程中，建立如下五个模型。

（1）需求模型。用例模型通过需求分析建立。……
（2）分析模型。用例模型通过分析来构造。……　} 分析

（3）设计模型。用例模型通过设计来具体化。……
（4）实现模型。该模型依据具体化的设计来实现用例模型。……　} 构造

（5）测试模型。用来测试具体化的用例模型。…… 测试

OOSE 方法的最大特点是面向用例，并在用例的描述中引入了外部角色的概念。

贸易销售系统的用例图如图 5.18 所示。

OOSE 方法支持商业工程和需求分析。在开发各种模型时，用例贯穿了 OOSE 活动的核心，描述了系统的需求及功能。

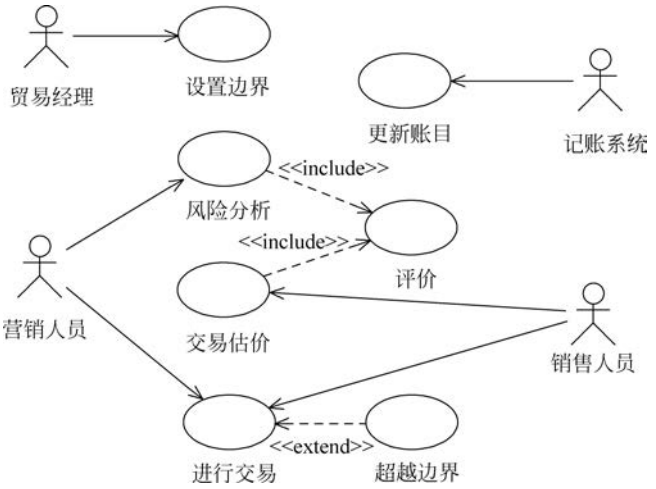

图 5.18 用例图

5.3 UML

Booch 方法、Coda/Yourdon 方法、OMT 方法及 OOSE 方法代表了面向对象方法的主要流派，它们都采用建模技术建立了各种视图来描述软件系统。模型是对系统的抽象表示，建模是在不同层次上对系统的描述。鉴于软件尤其是大型软件所具有的复杂性，以及人们对复杂问题理解的局限性，建立一种共同的建模语言来推动 OO 方法的发展是十分必要的，UML 应运而生。

UML 是一种基于面向对象的可视化通用建模语言，该方法结合了 Booch 方法、OMT 方法和 OOSE 方法的优点，统一了符号体系，并从其他的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。不论在计算机学术界、软件产业界还是商业界，UML 已经逐渐成为系统建模、描述系统体系结构、商业体系结构和商业过程时常用的统一建模工具，并且在实践过程中还在不断扩展应用领域。

5.3.1 UML 概述

软件工程领域在 1995—1997 年取得了前所未有的进展，其成果超过该领域过去近二十年的成果总和，其中最重要的成果之一就是 UML 的出现。UML 是具有指定建模元素（图式符号）、严格语法（构图规则）、明确语义（逻辑含义）的建模语言，是在面向对象技术领域内占主导地位的标准建模语言。

UML 三个字母的含义如下。

U：对多种经典的 OO 建模方法进行了统一，形成了规范。

M：用于建立软件开发过程中的各种工程模型。

L：是一种可视化的（图式）语言。

建模语言虽然众多，但用户由于没有能力区别不同语言之间的差别，因此很难找到一种比较适合其应用特点的语言。虽然不同的建模语言大多相似，但仍存在某些细微的差别，极大地妨碍了用户之间的交流。因此客观上极有必要在精心比较不同建模语言的优缺点及总结面向对象技术应用实践的基础上，组织联合设计小组，根据应用需求，取其精华，

去其糟粕，求同存异，统一建模语言。

建模方法应包括建模语言和建模过程两部分，其中建模语言提供用于表示建模结果的符号，建模过程用于描述建模时需要遵循的步骤。

UML 不仅统一了 Booch 方法、OMT 方法、OOSE 方法的表示方法，而且对其作了进一步的发展，最终统一为大众接受的标准建模语言。UML 是一种定义良好、易于表达、功能强大且普遍适用的建模语言，它融入了软件工程领域的新思想、新方法和新技术。它的作用域不限于支持面向对象分析与设计，还支持从需求分析开始的软件开发全过程。

UML 的目标之一就是为开发团队提供标准通用的设计语言来开发和构建计算机应用。UML 提出了一套 IT 专业人员期待多年的统一的标准建模符号。使用 UML，这些人员能够阅读和交流系统架构和设计规划，就像建筑工人多年来所使用的建筑设计图一样。

5.3.2 UML 的内容

UML 是一种标准化的图形建模语言，它是面向对象分析与设计的一种标准表示，由模型元素、图、通用机制以及视图四个部分构成。

1. 模型元素

模型元素代表面向对象中的类、对象、关系和交互等概念，是构成图的最基本的常用元素。一个模型元素可以用在多个不同的图中，无论怎样使用，它总是具有相同的含义和相同的符号表示。模型元素之间的连接关系也是模型元素，常见的关系有关联、泛化、依赖和实现。

(1) 类。在 UML 模型中，类是用一个矩形表示的，它包含三个区域，最上面是类名，中间是类的属性，最下面是类的方法，如图 5.19 所示。

(2) 对象。在 UML 模型中，对象是用一个矩形表示的在矩形框中，不再写出属性名和方法名，只是在矩形框中用“对象名：类名”的格式表示一个对象，如图 5.20 所示。

(3) 接口。外界对类(或构件)的使用是通过类(或构件)的方法来实现的，因此把类或构件的方法集合称为接口。接口向外界声明了它能提供的服务。

在 UML 模型中，接口用一个小圆圈表示，如图 5.21 所示。

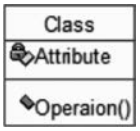

图 5.19 类的表示

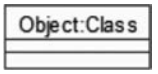

图 5.20 对象的表示

图 5.21 接口

(4) 用例。在系统中，为完成某个任务而执行一系列动作，以实现某种功能，我们把这些动作的集合称为用例实例。用例是对一组用例实例共同特征的描述，用例与用例实例的关系正如类与对象的关系。用例是 Jacobson 首先提出的，现已经成为面向对象软件开发中一个需求分析的最常用工具。

在 UML 模型中，用例是用一个实线椭圆来表示的，在椭圆中写入用例名称，如图 5.22 所示。

(5) 参与者。参与者是与系统、子系统或类发生交互作用的外部用户、进程或其他系统的理想化概念。在系统的实际运作中，一个实际用户可能对应系统的多个参与者，不同的用户也可以对应于一个参与者，从而代表同一参与者的不同实例。

在 UML 模型中，参与者用一个小人图标表示，如图 5.23 所示。

(6) 组件。组件也称构件,在系统设计中,一个相对独立的软件部件会把功能实现部分隐藏在内部,对外声明一组接口(包括供给接口和需求接口)。因此,两个具有相同接口的组件可以相互替换。

组件是比"类"更大的软件部件,例如一个 COM 组件、一个 DLL 文件、一个 JavaBeans、一个执行文件等。为了更好地在 UML 模型中表示它们,引入了组件。

在 UML 模型中,组件用带有两个小方框的矩形表示,如图 5.24 所示。

图 5.22 用例

图 5.23 参与者

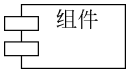

图 5.24 组件

(7) 节点。节点是指硬件系统中的物理部件,它通常具有存储空间或处理能力。如 PC、打印机、服务器等都是节点。

在 UML 模型中,用一个立方体表示一个节点,如图 5.25 所示。

(8) 交互。交互是为了完成某个任务对象之间的相互作用,这种作用是通过信息的发送和接收来完成的。

在 UML 模型中,交互的表示法很简单,用一条有向直线来表示对象间的交互,并在有向直线上面标注消息名称,如图 5.26 所示。

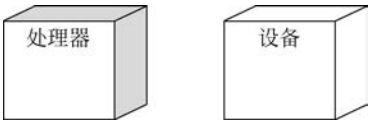

图 5.25 节点

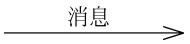

图 5.26 消息

(9) 状态机。在对象生命周期内,在事件的驱动下,对象从一种状态迁移到另一种状态的状态序列构成了状态机,即一个状态机由多个状态组成。

在 UML 模型中,状态表示为一个圆角矩形,并在矩形内标识状态名称,如图 5.27 所示。

(10) 包。中大型的软件系统通常会包含大量的类、接口、交互,因此也就会存在大量的结构元素、行为元素。为了能有效地对这些元素进行分类和管理,需要对其进行分组。UML 中提供了"包"来实现这一目标。

在 UML 模型中,表示"包"的图形符号与 Windows 中表示文件夹的图形符号很相似。包的作用与文件夹的作用也相似,如图 5.28 所示。

图 5.27 状态机

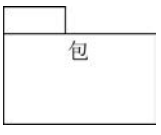

图 5.28 包

(11) 注释。用来对其他元素进行解释的部分(文本解释)称为注释。

在 UML 模型中，注释元素是用一个右上角折起来的矩形，解释的文字就写在矩形中。如图 5.29 所示。

(12) 关联关系。关联表示两个类之间存在某种语义上的联系，这种语义是人们赋予事物的联系。关联关系提供了通信的路径，它是所有关系中最通用、语义最弱的关系。

在关联关系中，有两种比较特殊的关系，它们是聚合关系和组合关系。

关联关系是聚合关系和组合关系的统称，是比较抽象的关系；聚合关系和组合关系是更具体的关系。

在 UML 模型中，使用一条实线来表示关联关系，如图 5.30 所示。

聚合是一种特殊形式的关联。聚合表示类之间的关系是整体与部分的关系。聚合关系是一种松散的对象间关系，计算机和它的外围设备就是一例。一台计算机和它的外设之间只是很松散地结合在一起。这些外设可有可无，可以与其他计算机共享，而且没有任何意义表明它由一台特定的计算机所“拥有”——这就是聚合。

在 UML 模型中，聚合关系是一条实线，用空心菱形端表示事物的整体部分，另一端表示事物的部分，如图 5.31 所示。

图 5.29　注释

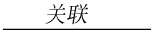

图 5.30　关联关系

图 5.31　聚合关系

如果发现“部分”类的存在是完全依赖于“整体”类的，那么应该使用“组合”关系来描述。组合关系是一种非常强的对象间关系，例如树和它的树叶之间的关系。树和它的叶子紧密联系在一起，叶子完全属于这树，它们不能被其他的树所分享，并且当树死掉，叶子也会随之死去。这就是组合，组合是一种强的聚合关系。

在 UML 模型中，组合关系是一条实线，用实心菱形端表示事物的整体部分，另一端表示事物的部分，如图 5.32 所示。

(13) 依赖关系。有两个元素 X、Y，如果修改元素 X 的定义可能会引起对另一个元素 Y 的定义的修改，则称元素 Y 依赖于元素 X。

在 UML 模型中，依赖关系用带箭头的虚线表示，用箭头端表示被依赖的事物，另一端是与之依赖的事物，如图 5.33 所示。

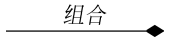

图 5.32　组合关系

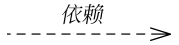

图 5.33　依赖关系

(14) 泛化关系。泛化关系描述了从特殊事物到一般事物之间的关系，也就是子类到父类之间的关系。从父类到子类的关系则是特化关系。

在 UML 模型中，泛化关系是一条实线，用空心三角箭头端表示一般事物，另一端表示特殊事物，如图 5.34 所示。

(15) 实现关系。实现关系用来规定接口和实现接口的类或组件之间的关系。接口是操作的集合，这些操作用于规定类或组件提供的服务。

在 UML 模型中，实现关系是一条虚线，用空心三角箭头端表示接口，另一端表示实现

接口的类或组件，如图 5.35 所示。

图 5.34 泛化关系

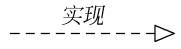

图 5.35 实现关系

2. UML 图

最常用的 UML 图包括用例图、类图、时序图、状态机图、活动图、组件图和部署图。

(1) 用例图。用例图描述了系统提供的一个功能单元，用来展示各类外部执行者与系统所提供的用例之间的连接。用例图的主要目的是帮助开发团队以一种可视化的方式理解系统的功能需求，包括基于基本流程的“角色”，也就是与系统交互的其他实体关系以及系统内用例之间的关系。

此外，在用例图中，没有列出的用例表明了该系统不能完成的功能。在用例图中应提供清楚、简要的用例描述，用户就很容易看出系统是否提供了必需的功能。

(2) 类图。类图表示不同的实体(人、事物和数据)如何彼此相关。换句话说，它显示了系统的静态结构。

可以把若干个相关的类包装在一起作为一个单元(包)，相当于一个子系统。一个系统可以有多张类图，一个类也可以出现在几张类图中。

(3) 时序图。时序图表示具体用例(或者是用例的一部分)的详细流程，显示了流程中不同对象之间的调用关系，同时还可以很详细地显示对不同对象的不同调用。时序图有两个维度：垂直维度，以发生的时间顺序显示消息/调用的序列；水平维度，显示消息被发送到的对象实例。

(4) 状态机图。状态机图通常是对类描述的补充，表示某个类所处的不同状态和该类的状态转换信息。

(5) 活动图。活动图表示在处理某个活动时，两个或者更多类对象之间的过程控制流。活动图通常用来描述完成一个操作所需要的活动。当然它还能用于描述其他活动流，如描述用例。活动图由动作状态组成，包含完成一个动作的活动规约(即规格说明)。当一个动作完成时，将离开该动作状态。活动图中的动作部分还可包括消息发送和接收的规约。活动图可用于在业务单元级别对更高级别的业务过程进行建模或者对低级别的内部类操作进行建模。

(6) 组件图。组件图用于展示系统中的组件(即来自应用的软件单元)、组件间通过接口的连接以及组件之间的依赖关系。组件图可以在一个非常高的层次上显示，从而仅显示粗粒度的组件，也可以在组件包层次上显示。

(7) 部署图。部署图展示了运行时处理的节点和在节点上存在的制品的配置。节点是运行时的计算资源，制品是物理实体，如构件、文件。部署图的用途是显示该系统中不同的组件将在何处物理地运行以及它们之间如何彼此通信。因为部署图是对物理运行情况进行建模，系统的生产人员可以很好地利用这种图。

3. UML 视图

一个系统应从不同的角度进行描述，从一个角度观察到的系统称为一个视图。视图由多个图构成，它不是一个图表，而是在某个抽象层上对系统的抽象表示。

如果要为系统建立一个完整的模型图，需定义一定数量的视图，每个视图表示系统的一个特殊方面。另外，视图还应把建模语言和系统开发时选择的方法或过程连接起来。

UML 视图的结构如图 5.36 所示。

图 5.36　UML 视图

一个系统有多种视图。对于同一个系统，不同人员所关心的内容是不相同的。

（1）用例视图。描述系统的外部特性、系统功能等。分析人员和测试人员关心的是系统的行为，因此会侧重于用例视图。

（2）逻辑视图。描述系统的设计特征，包括结构模型视图和行为模型视图。前者描述系统的静态结构，后者描述系统的动态行为。最终用户关心的是系统的功能，因此会侧重于逻辑视图。

（3）进程视图。表示系统内部的控制机制。常用类图描述过程结构，用交互图描述过程行为。系统集成人员关心的是系统的性能、可伸缩性、吞吐率等问题，因此会侧重于进程视图。

（4）配置视图。描述系统的物理配置特征，用部署图表示。系统工程师关心的是系统的发布、安装、拓扑结构等问题，因此会侧重于部署视图。

（5）实现视图。表示系统的实现特征，常用构件图表示。程序员关心的是系统的配置、装配等问题，因此会侧重于实现视图。

4. 通用机制

通用机制用于表示其他信息，比如详述模型元素的语义、修饰、通用划分、扩展机制、注释等，适用于软件系统或业务系统中每个事物的方法或规则。另外，为了适应用户的需求，通用机制允许在不修改基础元模型的前提下对 UML 作有限的变化，如提供了扩展机制，包括构造型、标记值和约束。使用 UML 语言能够适应一个特殊的方法（或过程）或扩充至一个组织或用户。

习题五

一、填空题

1. 对象的抽象是_____，类的实例化是_____。

2. 继承性是_____自动共享父类的属性和_____的机制。

3. 面向对象技术的基本特征主要是抽象性、_____、继承性和_____。

4. OMT 方法使用建模的思想建立了三类模型：_____、_____和_____。

5. OOSE 将面向对象的思想应用于软件工程中，建立的五个模型分别是需求模型、_____、_____、实现模型和_____。

6. UML 是一种标准化的图形建模语言，它的内容包括_____、_____、模型元素、_____四个部分。

二、简答题

1. 什么是面向对象？

2. 面向对象的基本特征是什么？
3. 什么是软件开发过程？
4. 传统软件开发方法存在什么问题？
5. 面向对象开发方法的特点是什么？
6. 什么是统一建模语言？
7. 简述 Coda/Yourdon 方法的面向对象设计模型。
8. 简述 UML 视图结构。

三、综合题

1. 举例说明并解释类、属性、操作、继承性、多态性、封装及抽象类的概念。
2. 列举面向对象开发方法，并说明每个方法的特点。
3. 试举一个抽象类与子类的设计实例。
4. 单重继承和多重继承设计各举一个实例。
5. 试举一个多态设计的实例。