第 5 章 类的继承

面向对象程序设计有 4 个主要特点：**抽象**、**封装**、**继承**和**多态性**。通过前几章的学习，初步了解了抽象和封装。要更好地掌握面向对象程序设计，还必须了解面向对象程序设计另外两个重要特征——继承性和多态性。本章介绍有关继承的知识，在第 6 章将介绍多态性。

继承性是面向对象程序技术重要的特征。在传统的程序设计中，人们往往要为每一种应用项目单独地进行一次程序的开发，因为每一种应用有不同的目的和要求，程序的结构和具体的编码是不同的，人们无法使用已有的软件资源。即使两种应用具有许多相同或相似的特点，程序设计者可以吸取已有程序的思路，作为自己开发新程序的参考，人们也不得不重起炉灶，重写程序或者对已有的程序进行较大的改写。显然，这种方法的重复工作量是很大的，这是因为过去的程序设计方法和计算机语言缺乏**软件重用**的机制。人们无法利用现有的丰富的软件资源，这就造成软件开发中人力、物力和时间的巨大浪费，效率较低。

面向对象技术强调**软件的可重用性**(software reusability)。C++语言提供了类的继承机制，解决了软件重用问题。

5.1 继承与派生

在C++中，可重用性是通过“继承”(inheritance)这一机制来实现的。因此，继承是C++功能的一个重要组成部分。

前面介绍了类，一个类中包含了若干数据成员和成员函数。在不同的类中，数据成员和成员函数是不相同的。但有时两个类的内容基本相同或有一部分相同。例如声明了学生基本数据的类 Student：

```
class Student
  { public:
     void display()                              //对成员函数 display 的定义
       {cout<<"num: "<<num<<endl;
```

```
        cout<<"name: "<<name<<endl;
        cout<<"sex: "<<sex<<endl; }
    private:
      int num;
      string name;
      char sex;
  };
```

如果学校的某一部门除了需要用到学号、姓名、性别以外,还需要用到年龄、地址等信息。当然可以重新声明另一个类 class Student1:

```
class Student1
  { public:
      void display();                          //此行原来已有
        {cout<<"num: "<<num<<endl;             //此行原来已有
         cout<<"name: "<<name<<endl;           //此行原来已有
         cout<<"sex: "<<sex<<endl;             //此行原来已有
         cout<<"age: "<<age<<endl;
         cout<<"address: "<<addr<<endl;}
    private:
        int num;                               //此行原来已有
        string name;                           //此行原来已有
        char sex;                              //此行原来已有
        int age;
        char addr[20];
  };
```

可以看到有相当一部分是原来已有的。很多人自然会想到能否利用原来声明的类 Student 作为基础,再加上新的内容即可,以减少重复的工作量。C++提供的**继承**机制就是为了解决这个问题。

在第 2 章已举了马的例子来说明继承的概念。“公马”继承了“马”的全部特征,再加上“雄性”的新特征。“白公马”又继承了“公马”的全部特征,再增加“白色”的特征。“公马”是“马”派生出来的一个分支,“白公马”是“公马”派生出来的一个分支,见图 5.1。

在C++中,所谓“继承”就是在一个已存在的类的基础上建立一个新的类。已存在的类(例如“马”)称为“基类”(base class)或“父类”(father class)。新建立的类(例如“公马”)称为“派生类”(derived class)或“子类”(son class),见图 5.2。

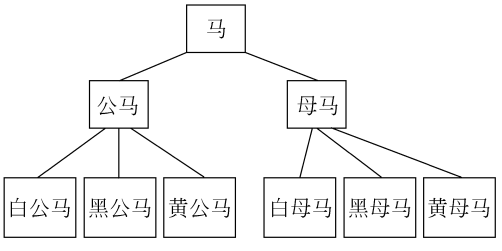

图　5.1

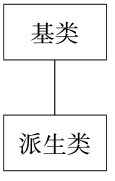

图　5.2

一个新类从已有的类那里获得其已有特性,这种现象称为**类的继承**。通过继承,一个新建子类从已有的父类那里获得父类的特性。从另一角度说,从已有的类(父类)产生一个新的子类,称为类的派生。类的继承是用已有的类来建立专用类的编程技术。

派生类继承了基类的所有数据成员和成员函数,并可以对成员作必要的增加或调整。一个基类可以派生出多个派生类,每一个派生类又可以作为基类再派生出新的派生类,因此基类和派生类是相对而言的。一代一代地派生下去,就形成类的**继承层次结构**。相当于一个大的家族,有许多分支,所有的子孙后代都继承了祖辈的基本特征,同时又有区别和发展。与之相仿,类的每一次派生,都继承了其基类的基本特征,同时又根据需要调整和扩充原有特征。

以上介绍的是最简单的情况:一个派生类只从一个基类派生,这称为**单继承**(single inheritance),这种继承关系所形成的层次是一个树形结构,可以用图5.3表示。

请注意图中箭头的方向,一般约定:箭头表示继承的方向,从派生类指向基类。

一个派生类不仅可以从一个基类派生,也可以从多个基类派生,也就是说,一个派生类可以有两个或多个基类(或者说,一个子类可以有两个或多个父类)。例如马与驴杂交所生下的骡子,就有两个基类——马和驴。骡子既继承了马的一些特征,也继承了驴的一些特征。又如"计算机专科",是从"计算机专业"和"大专层次"派生出来的子类,它具备两个基类的特征。一个派生类有两个或多个基类的称为**多重继承**(multiple inheritance),这种继承关系所形成的结构如图5.4所示。

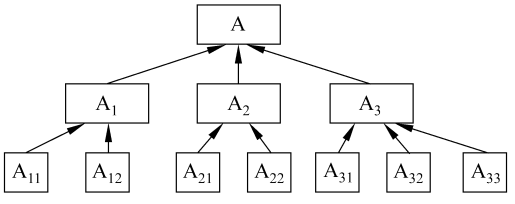

图 5.3

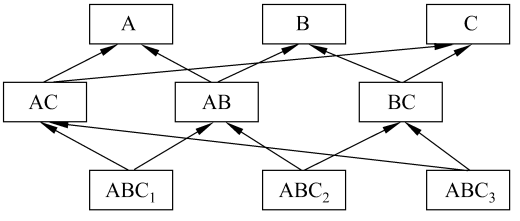

图 5.4

关于基类和派生类的关系,可以表述为:**派生类是基类的具体化,而基类则是派生类的抽象**。从图5.5中可以看到:小学生、中学生、大学生、研究生、留学生是学生的具体化,他们是在学生的共性基础上加上某些特点形成的子类。而**学生**则是对各类学生共性的综合,是对各类具体学生特点的抽象。基类综合了派生类的公共特征,派生类则在基类的基础上增加某些特性,把抽象类变成具体的、实用的类型。

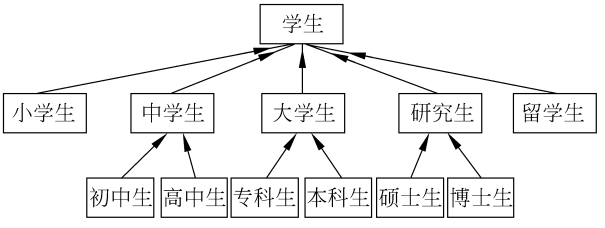

图 5.5

5.2　派生类的声明方式

先通过一个例子说明怎样通过继承来建立派生类,从最简单的单继承开始。

假设已经声明了一个基类 Student(见 5.1 节),在此基础上通过单继承建立一个派生类 Student1:

```
class Student1: public Student                    //声明基类是 Student
  { public:
      void display_1()                            //新增加的成员函数
      {cout<<"age: "<<age<<endl;
       cout<<"address: "<<addr<<endl;}
    private:
      int age;                                    //新增加的数据成员
      string addr;                                //新增加的数据成员
  };
```

仔细观察第一行:

```
class Student1: public Student
```

在 class 后面的 Student1 是新建的类名。冒号后面的 Student 表示是已声明的基类。在 Student 之前有一关键字 public,用来表示基类 Student 中的成员在派生类 Student1 中的**继承方式**。基类名前面有 public 的称为"**公用继承**"(public inheritance)。

请读者仔细阅读以上声明的派生类 Student1 和 5.1 节中给出的基类 Student,并将它们放在一起进行分析。

声明派生类的一般形式为

```
class 派生类名:[继承方式] 基类名
  {
    派生类新增加的成员
  };
```

继承方式包括:**public**(**公用的**),**private**(**私有的**)和 **protected**(**受保护的**),继承方式是可选的,如果不写此项,则默认为 private(私有的)。

5.3　派生类的构成

派生类中的成员包括从基类继承过来的成员和自己增加的成员两大部分。从基类继承的成员体现了派生类从基类继承而获得的共性,而新增加的成员体现了派生类的个性。正是这些新增加的成员体现了派生类与基类的不同,体现了不同派生类之间的区别。为了形象地表示继承关系,本书采用图 5.6 来示意。在基类中包括数据成员和成员函数(或称数据与方法)两部分,派生类分为两大部分:一部分是从基类继承来的成员,另一部分是在声明派生类时增加的部分。每一部分均分别包括数据成员和成员函数。

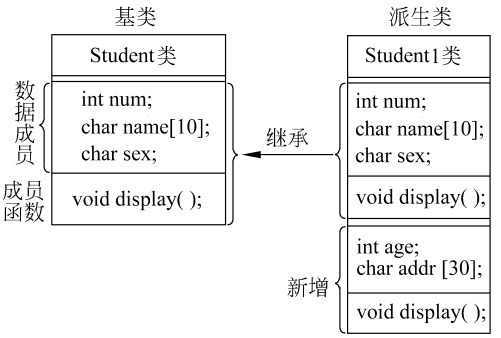

图 5.6

实际上,并不是把基类的成员和派生类自己增加的成员简单地加在一起就成为派生类。构造一个派生类包括以下 3 部分工作。

(1) **从基类接收成员**。派生类把基类**全部**的成员(不包括构造函数和析构函数)接收过来,也就是说是没有选择的,不能选择接收其中一部分成员,而舍弃另一部分成员。从定义派生类的一般形式中可以看出是不可选择的。

这样就可能出现一种情况:有些基类的成员,在派生类中是用不到的,但是也必须继承过来。这就会造成数据的冗余,尤其是在多次派生之后,会在许多派生类对象中存在大量无用的数据,不仅浪费了大量的空间,而且在对象的建立、赋值、复制和参数的传递中,花费了许多无谓的时间,从而降低了效率。这在目前的C++标准中是无法解决的,要求我们根据派生类的需要慎重选择基类,使冗余量最小。不要随意地从已有的类中找一个作为基类去构造派生类,应当考虑怎样能使派生类有更合理的结构。实际上,有些类是专门作为基类而设计的,在设计时充分考虑到派生类的要求。

(2) **调整从基类接收的成员**。接收基类成员是程序人员不能选择的,但是程序人员可以对这些成员的属性作某些调整。例如可以改变基类成员在派生类中的访问属性,这是通过指定继承方式来实现的。如可以通过继承把基类的公用成员指定为在派生类中的访问属性为私有(派生类外不能访问)。此外,可以在派生类中声明一个与基类成员同名的成员,则派生类中的新成员会覆盖基类的同名成员,但应注意:如果是成员函数,不仅应使函数名相同,而且函数的参数表(参数的个数和类型)也应相同,如果不相同,就成为函数的重载而不是覆盖了。用这样的方法可以用新成员取代基类的成员。

(3) **在声明派生类时增加的成员**。这部分内容是很重要的,它体现了派生类对基类功能的扩展。要根据需要仔细考虑应当增加哪些成员,精心设计。例如在 5.2 节的例子中,基类的 display 函数的作用是输出学号、姓名和性别,而在派生类中要求输出学号、姓名、性别、年龄和地址。在建立派生类时不必另外写一个输出这 5 个数据的函数,可以利用基类的 display 函数输出学号、姓名和性别,另外再定义一个 display_1 函数输出年龄和地址,先后执行这两个函数。也可以在 display_1 函数中调用基类的 display 函数,再输出另外两个数据,在主函数中只须调用一个 display_1 函数即可,这样可能更清晰一些,易读性更好。

此外,在声明派生类时,一般还应当自己定义派生类的构造函数和析构函数,因为构造函数和析构函数是不能从基类继承的。

通过以上的介绍,可以看到,派生类是基类定义的延续。在实际的工作中往往先声明

一个基类,在此基类中只提供某些最基本的功能,而另外有些功能并未实现,然后在声明派生类时加入某些具体的功能,形成适用于某一特定应用的派生类。通过对基类声明的延续,将一个抽象的基类转化成具体的派生类。因此,**派生类是抽象基类的具体实现**。

5.4　派生类成员的访问属性

既然派生类中包含基类成员和派生类自己增加的成员,就产生了这两部分成员的关系和访问属性的问题。在建立派生类的时候,并不是简单地把基类的私有成员直接作为派生类的私有成员,把基类的公用成员直接作为派生类的公用成员。实际上,对基类成员和派生类自己增加的成员是按不同的原则处理的。

具体说,在讨论访问属性时,需要考虑以下几种情况:

(1) 基类的成员函数访问基类成员。

(2) 派生类的成员函数访问派生类自己增加的成员。

(3) 基类的成员函数访问派生类的成员。

(4) 派生类的成员函数访问基类的成员。

(5) 在派生类外访问派生类的成员。

(6) 在派生类外访问基类的成员。

对于第(1)和(2)种情况,比较简单,按第 2 章介绍过的规则处理,即基类的成员函数可以访问基类成员,派生类的成员函数可以访问派生类成员。私有数据成员只能被同一类中的成员函数访问,公用成员可以被外界访问。第(3)种情况也比较明确,基类的成员函数只能访问基类的成员,而不能访问派生类的成员。第(5)种情况也比较明确,在派生类外可以访问派生类的公用成员,而不能访问派生类的私有成员。

对于第(4)和第(6)种情况,就稍微复杂一些,也容易混淆。例如,有人提出这样的问题:

① 基类中的成员函数是可以访问基类中的任一成员的,那么派生类中新增加的成员(如 display1 函数)是否可以同样地访问基类中的私有成员。

② 在派生类外,能否通过派生类的对象名访问从基类继承的公用成员(例如,已定义 stud1 是 Student1 类的对象,能否用 stud1.display( )调用基类的 display 函数)。

这些涉及如何确定基类的成员在派生类中的访问属性的问题,不仅要考虑对基类成员所声明的访问属性,还要考虑派生类所声明的对基类的继承方式,根据这两个因素共同决定基类成员在派生类中的访问属性。

前面已提到,在派生类中,对基类的继承方式可以有 public(公用的),private(私有的)和 protected(保护的)3 种。不同的继承方式决定了基类成员在派生类中的访问属性。

简单地说:

(1) **公用继承(public inheritance)**

基类的公有成员和保护成员在派生类中保持原有访问属性,其私有成员仍为基类私有。

(2) **私有继承(private inheritance)**

基类的公有成员和保护成员在派生类中成了私有成员。其私有成员仍为基类私有。

(3) **受保护的继承(protected inheritance)**

基类的公有成员和保护成员在派生类中成了保护成员,其私有成员仍为基类私有。

保护成员的意思是:不能被外界引用,但可以被派生类的成员引用,具体的用法将在稍后介绍。

下面分别介绍这 3 种访问属性。

5.4.1 公用继承

在建立一个派生类时将基类的继承方式指定为 public 的,称为**公用继承**,用公用继承方式建立的派生类称为**公用派生类**(public derived class),其基类称为**公用基类**(public base class)。

采用公用继承方式时,基类的公用成员和保护成员在派生类中仍然保持其公用成员和保护成员的属性,而基类的私有成员在派生类中并没有成为派生类的私有成员,它仍然是基类的私有成员,只有基类的成员函数可以引用它,而不能被派生类的成员函数引用,因此就成为派生类中的**不可访问的成员**。

公用基类的成员在派生类中的访问属性见表 5.1。

表 5.1 公用基类在派生类中的访问属性

在基类的访问属性	继 承 方 式	在派生类中的访问属性
private(私有)	public(公用)	不可访问
public(公用)	public(公用)	public(公用)
protected(保护)	public(公用)	protected(保护)

例如,派生类 Student1 中的 display_1 函数不能访问公用基类的私有成员 num,name,sex。在派生类外,可以访问公用基类中的公用成员函数(如 stud1.display())。

有人问:既然是公用继承,为什么不能访问基类的私有成员呢?要知道,这是C++中一个重要的软件工程观点。因为私有成员体现了数据的封装性,隐藏私有成员有利于测试、调试和修改系统。如果把基类所有成员的访问权限都原封不动地继承到派生类,使基类的私有成员在派生类中仍保持其私有性质,派生类成员能访问基类的私有成员,那么岂非基类和派生类没有界限了?这就破坏了基类的封装性。如果派生类再继续派生一个新的派生类,也能访问基类的私有成员,那么在这个基类的所有派生类的层次上都能访问基类的私有成员,这就完全丢弃了封装性带来的好处。保护私有成员是一条重要的原则。

例 5.1 访问公有基类的成员。

编写程序:

先写类的声明部分。

```
Class Student                                    //声明基类
  {public:                                       //基类公用成员
    void get_value()                             //输入基类数据的成员函数
      {cin>>num>>name>>sex;}
```

```
    void display()                              //输出基类数据的成员函数
      {cout<<" num: "<<num<<endl;
       cout<<" name: "<<name<<endl;
       cout<<" sex: "<<sex<<endl;}
   private:                                     //基类私有成员
    int num;
    string name;
    char sex;
  };

class Student1: public Student                  //以 public 方式声明派生类 Student1
  {public:
   void get_value_1()                           //输入派生类数据
     {cin>>age>>addr;}
   void display_1()
     { cout<<" num: "<<num<<endl;               //试图引用基类的私有成员,错误
       cout<<" name: "<<name<<endl;             //试图引用基类的私有成员,错误
       cout<<" sex: "<<sex<<endl;               //试图引用基类的私有成员,错误
       cout<<" age: "<<age<<endl;               //引用派生类的私有成员,正确
       cout<<" address: "<<addr<<endl;}         //引用派生类的私有成员,正确
   Private:
       int age;
       string addr;
  };
```

由于基类的私有成员对派生类来说是不可访问的,因此在派生类中的 display_1 函数中直接引用基类的私有数据成员 num,name 和 sex 是不允许的。只能通过基类的公用成员函数来引用基类的私有数据成员。

可以将派生类 Student1 的声明改为

```
class Student1: public Student                  //以 public 方式声明派生类 Student1
  {public:
    void get_value_1()
      {cin>>age>>addr;}
    void display_1()
      {cout<<" age: "<<age<<endl;               //引用派生类的私有成员,正确
       cout<<" address: "<<addr<<endl;          //引用派生类的私有成员,正确
      }
     private:
      int age;
      string addr;
  };
```

然后在 main 函数中分别调用基类的 display 函数和派生类中的 display_1 函数,先后输出 5 个数据。

可以这样写 main 函数(假设对象 stud 中已有数据):

```
int main()
  { Student1 stud;          //定义派生类 Student1 的对象 stud
    stud.get_value();       //调用基类的公用成员函数,输入基类中 3 个数据成员的值
    stud.get_value-1();     //调用派生类的公用成员函数,输入派生类两个数据成员的值
    stud.display();         //调用基类的公用成员函数,输出基类中 3 个数据成员的值
    stud.display_1();       //调用派生类的公用成员函数,输出派生类中两个数据成员的值
    return 0;
  }
```

请读者根据上面的分析,写出完整的程序。

运行结果:

```
1001 Zhang m 21 Shanghai↙          (输入 5 个数据)
name: Zhang                        (输出 5 个数据)
num: 1001
sex: m
age: 21
address: Shanghai
```

程序分析:

请分析在主函数中能否出现以下语句:

```
stud.age=18;                       //错误。在类外不能引用派生类的私有成员
stud.num=10020;                    //错误。在类外不能引用基类的私有成员
```

实际上,程序还可以改进,在派生类的 display_1 函数中调用基类的 display 函数,这样,在主函数中只要写一行:

```
stud.display_1();
```

即可输出 5 个数据。以上只是为了说明派生类成员的引用方法,在学习 5.4.2 节中的例 5.2 后就会清楚了。

5.4.2 私有继承

私有继承和 5.4.3 节的保护继承,在初步编程中用得不多,但为了使读者对它有一定的了解,本书作了简单的介绍。初学时可以跳过这两节不学,以后有需要时可查阅参考。

在声明一个派生类时将基类的继承方式指定为 private 的,称为**私有继承**,用私有继承方式建立的派生类称为**私有派生类**(private derived class),其基类称为**私有基类**(private base class)。

私有基类的公用成员和保护成员在派生类中的访问属性相当于派生类中的**私有成员**,即派生类的成员函数能访问它们,而在派生类外不能访问它们。**私有基类的私有成员在派生类中成为不可访问的成员**,只有基类的成员函数可以引用它们。一个基类成员在基类中的访问属性和在派生类中的访问属性可能是不同的。私有基类的成员可以被基类的成员函数访问,但不能被派生类的成员函数访问。私有基类的成员在私有派生类中的访问属性见表 5.2。

表 5.2　私有基类在派生类中的访问属性

在基类的访问属性	继 承 方 式	在派生类中的访问属性
private(私有)	private(私有)	不可访问
public(公用)	private(私有)	private(私有)
protected(保护)	private(私有)	private(私有)

图 5.7 表示了各成员在派生类中的访问属性。若基类 A 有公用数据成员 i 和 j,私有数据成员 k(见图 5.7(a)),采用私有继承方式声明了派生类 B,新增加了公用数据成员 m 和 n,私有数据成员 p(见图 5.7(b))。在派生类 B 作用域内,基类 A 的公用数据成员 i 和 j 呈现私有成员的特征,在派生类 B 内可以访问它们,而在派生类 B 外不可访问它们。在派生类内不可访问基类 A 的私有数据成员 k。此时,从派生类的角度看,相当于有公用数据成员 m 和 n,私有成员 i,j,p(见图 5.7(c))。基类 A 的私有数据成员 k 在派生类 B 中成为“不可见”的。

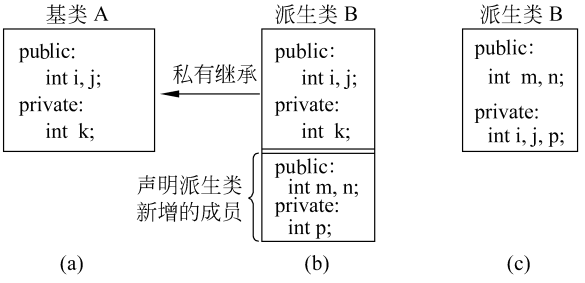

图　5.7

对表 5.2 的规定不必死记,只须理解:既然声明为私有继承,就表示将原来能被外界引用的成员隐蔽起来,不让外界引用,因此私有基类的公用成员和保护成员理所当然地成为派生类中的私有成员。私有基类的私有成员按规定只能被基类的成员函数引用,在基类外当然不能访问他们,因此它们在派生类中是隐蔽的,不可访问的。

对于不需要再往下继承的类的功能可以用私有继承方式把它隐蔽起来,这样,下一层的派生类无法访问它的任何成员。

可以知道:一个成员在不同的派生层次中的访问属性可能是不同的。它与继承方式有关。

例 5.2　将例 5.1 中的公用继承方式改为用私有继承方式(基类 Student 不改)。

可以写出私有派生类如下:

```
class Student1: private Student              //用私有继承方式声明派生类 Student1
  {public:
     void get_value_1()                      //输入派生类数据
       {cin>>num>>name>>sex;}
     void display_1()                            //输出两个数据成员的值
       { cout<<"age: "<<age<<endl;               //引用派生类的私有成员
         cout<<"address: "<<addr<<endl;}         //引用派生类的私有成员
```