第七章 继承——派生类

7.1 继承概述

软件复用

定义：在开发一个新软件时，把现有软件的一些功能拿过来用（子程序、类）

问题：

- 子程序（函数）：除了一些最基本的子程序（如数学函数）外，已有软件的子程序不是为新软件设计的，常常不完全符合新软件的功能要求。
- 类：对某个应用领域来讲，虽然类具有通用性，但针对不同的应用它们仍然存在一些差别

类的继承

定义：定义一个新的类，先把已有的一个或多个类的功能全部包含进来，然后再在新的类中给出新功能的定义或对已有类的某些功能进行重新定义。（不需要已有软件的源代码，属于目标代码复用）

- 类继承中存在两个类：
  - 基类（父类）：被继承
  - 派生类（子类）：继承后得到。派生类拥有基类的所有成员，并可以定义新的成员和对基类的一些成员函数进行重定义。
- 继承分为：单继承和多继承
  - 单继承：一个类只有一个直接基类。
  - 多继承：一个类有多个直接基类。
- 类继承的作用：
  - 支持软件复用
  - 按层次对对象进行分类：有利于问题的描述和解决。
  - 对概念进行组合：给新类的设计进一步带来便利。
  - 支持软件的增量开发：给软件开发和维护带来便利。

7.2 单继承

7.2.1 单继承派生类的定义

```
class <派生类名>:[<继承方式>] <基类名>
{
    <成员说明表>
};
```

`<继承方式>`：public，private，protected

`<成员说明表>`：在派生类中新定义的和对基类成员重定义的成员

- 派生类除了拥有新定义的成员外，还拥有基类的所有成员（基类的构造函数、析构函数和赋值操作符重载函数除外）
- 派生类的对象包含了基类的一个子对象，该子对象的内存空间位于派生类对象内存空间的前部。
- 定义派生类时一定要见到基类的定义。

```
class A;  //声明
class B: public A  //Error
{
    int z;
    public:
    void h() { g(); }  //Error，编译程序不知道基类中是否有函数g以及函数g的原型。
};
......
B b; //Error，编译无法确定b所需内存空间的大小。
```

- 友元：
  - 如果在派生类中没有显式指出，则基类的友元不是派生类的友元
  - 如果基类是另一个类的友元，而该类没有显式指出，则派生类不是该类的友元

7.2.2 在派生类中访问基类成员——protected访问控制

- C++中，派生类不能直接访问基类的私有成员

```
class A
{
    int x, y;

public:
    void f();
    void g() { ... x , y... }
};
class B : public A
{
    int z;

public:
    void h()
    {
        ... x, y... // Error，x、y为基类的私有成员。
            f();    // OK
        g();        // OK，通过函数g访问基类的私有成员x和y。
    }
};
```

- **继承与封装的矛盾**：在派生类中定义新的成员函数或对基类已有成员函数重定义时，往往需要直接访问基类的一些private成员（特别是private数据成员），否则新的功能无法实现；而类的private成员是不允许外界使用的（数据封装）
  - C++中引进`protected`成员访问控制来缓解矛盾：用protected说明的成员不能通过对象使用，但可以在派生类中使用
  - C++类向外界提供两种接口：
    - public：供类的实例用户使用（通过对象）
    - public+protected：供派生类使用

```
class A
{
protected:
    int x, y;
```

```
public:
    void f();
};
class B : public A // A的派生类
{
    ..... void h()
    {
        f();          // OK
        ... x...      // OK
            ... y... // OK
    }
};
void g() // A的实例用户
{
    A a;
    a.f();          // OK
    ... a.x...      // Error
    ... a.y... // Error
}
```

- 派生类成员标识符的作用域
  - 派生类对基类成员的访问除了受到基类的访问控制的限制以外，还要受到标识符作用域的限制；对基类而言，派生类成员标识符的作用域是嵌套在基类作用域中的。
  - 如果派生类中定义了与基类同名的成员，则基类的成员名在派生类的作用域内不直接可见（被隐藏，Hidden），访问基类同名的成员时要用基类名受限。
  - 即使派生类中定义了与基类同名但参数不同的成员函数，基类的同名函数在派生类的作用域中也是不直接可见的，仍然需要用基类名受限方式来使用
  - 也可以在派生类中使用using声明把基类中某个的函数名对派生类开放

```
class B : public A
{
    int z;

public:
    using A::f;
    void f(int);
    void h()
    {
        f(1); // OK
        f();  // OK，等价于A::f();
    }
};
```

7.2.3 基类成员在派生类中对外的访问控制——继承方式

基类成员 / 派生类 / 继承方式	public	private	protected
public	public	不可访问	protected
private	private	不可访问	private
protected	protected	不可访问	protected

- public继承与子类型
  - public继承有特殊的作用，它可以实现类之间的子类型关系：
    - 对用类型T表达的所有程序P，当用类型S去替换程序P中的所有的类型T时，程序P的功能不变，则称类型S是类型T的**子类型。**
  - 在C++中，把类看作类型，把以public方式继承的派生类看作是基类的子类型：
    - 对基类对象能实施的操作也能作用于派生类对象。
    - 在需要基类对象的地方可以用派生类对象去替代 。

7.2.4 派生类对象的初始化和消亡处理

- 派生类对象的初始化由基类和派生类共同完成：
  - 从基类继承的数据成员由基类的构造函数初始化；
  - 派生类新的数据成员由派生类的构造函数初始化。
- 当创建派生类的对象时，
  - 先执行基类的构造函数，再执行派生类构造函数。
  - 默认情况下，调用基类的默认构造函数，如果要调用基类的非默认构造函数，则必须在派生类构造函数的成员初始化表中显式指出。
- 当派生类对象消亡时，先调用本身类的析构函数，执行完后会去自动调用基类的析构函数。
- 对一个未提供任何构造/析构函数的类，如果它有基类，编译程序有时也会隐式地为之提供一个构造/析构函数
- 如果一个类D既有基类B、又有成员对象类M，则在创建D类对象时，构造函数的执行次序为：
  B->M->D
  当D类的对象消亡时，析构函数的执行次序为：
  D->M->B

派生类拷贝构造函数

- 派生类的隐式拷贝构造函数（由编译程序提供）：
  - 对派生类中新定义的成员进行拷贝初始化外。
  - 调用基类的拷贝构造函数实现对基类成员的初始化。
- 派生类自定义的拷贝构造函数：
  - 在默认情况下调用基类的默认构造函数对基类成员初始化。
  - 需要在“成员初始化表”中显式地指出调用基类的拷贝构造函数来实现对基类成员的初始化。

派生类对象的赋值操作

- 派生类隐式的赋值操作：
  - 对派生类成员进行赋值。
  - 调用基类的赋值操作对基类成员进行赋值。
- 派生类自定义的赋值操作：
  - 不会自动调用基类的赋值操作。
  - 需要在自定义的赋值操作符重载函数中显式地指出调用基类的赋值操作。

7.3 消息（成员函数调用）的动态绑定

7.3.1 消息的多态性

C++OOP中出现以下多态：

- 对象标识的多态。基类的指针（或引用）可以指向（或引用）基类对象，也可以指向（或引用）派生类对象，即一个对象标识可以标识多种类型的对象
- 消息的多态。一条可以发送到基类对象的消息，也可以发送到派生类对象。如果在基类和派生类中都给出了对这条消息的处理，那么这条消息存在特殊的多态性。

静态绑定：在编译时刻根据对象的类型来决定采用哪一个消息处理函数

动态绑定：需要在运行时刻，根据func1（或func2）中x（或p）实际引用（或指向）的对象来决定是调用A::f还是B::f

7.3.2 虚函数与消息的动态绑定

在C++中，在基类中用虚函数来指出动态绑定

虚函数是指加了关键词virtual的成员函数。其格式为：virtual <成员函数声明>;

虚函数有两个作用：

- 指定消息采用动态绑定。
- 指出基类中可以被派生类重定义的成员函数。

对于基类中的一个虚函数，在派生类中定义的、与之具有相同型构的成员函数是对基类该成员函数的**重定义**（或称覆盖，override）

相同型构是指：

- 派生类中定义的成员函数的名字、参数个数和类型与基类相应成员函数相同；
- 其返回值类型与基类成员函数返回值类型或者相同，或者是基类成员函数返回值类型的public派生类。

消息的动态绑定将绑定到派生类中与基类同型构的成员函数

只有类的成员函数才可以是虚函数，但静态成员函数不能是虚函数。
构造函数不能是虚函数，析构函数可以（往往）是虚函数。
只要在基类中说明了虚函数，在派生类、派生类的派生类、...中，同型构的成员函数都是虚函数（virtual可以不写）。
只有通过基类的指针或引用访问基类的虚函数时才进行动态绑定。
类的构造函数和析构函数中对虚函数的调用不进行动态绑定。

采用显式类型转换把基类指针或引用转换成派生类指针或引用（不安全），可以采用C++的动态类型转换操作（dynamic_cast）实现转换

7.3.3 纯虚函数和抽象类

纯虚函数是只给出声明没给出实现的虚函数，函数体用“=0”表示（virtual <成员函数声明>=0;）

包含纯虚函数的类称为**抽象类**。

- 抽象类不能用于创建对象
- 抽象类的作用是为派生类提供一个基本框架和一个公共的对外接口

由于在C++中使用某个类时必须要见到该类的定义，因此，使用者能够见到该类的一些实现细节（如：非public成员），这样会使得类的抽象作用大打折扣！有手段绕过类的访问控制而使用类的非public成员，类的封装作用可能被破坏。

可以通过给类提供一个抽象基类解决

7.4 多继承

7.4.1 多继承概述

多继承增强了语言的表达能力，它使得语言能够自然、方便地描述问题领域中的存在于对象类之间的多继承关系

7.4.2 多继承派生类的定义

多继承是指派生类可以有一个以上的直接基类。多继承的派生类定义格式为：

```
class <派生类名>: [<继承方式>] <基类名1>,
                  [<继承方式>] <基类名2>, ...
{     <成员说明表>
};
```

- 继承方式及访问控制的规定同单继承。
- 派生类拥有所有基类的所有成员。
- 基类的声明次序决定：
  - 对基类数据成员的存储安排。
  - 对基类构造函数/析构函数的调用次序

把多继承派生类对象的地址赋给基类的指针时将会自动进行地址调整

多继承带来的问题

多继承使得语言特征复杂化、加大了编译程序的难度以及使得消息绑定复杂化等，从而给正确使用多继承带来困难

- 名冲突问题
- 重复继承问题

7.4.3 名冲突问题

多个基类中包含同名的成员时，在派生类出现名冲突问题

解决方法：基类名受限访问

7.4.4 重复继承——虚基类

继承的类有公共的基类，出现重复继承。公共基类中的数据成员在多继承的派生类就有多个拷贝

```
class A
{  int x;
    ......
};
class B: public A { ...... };
class C: public A { ...... };
class D: public B, public C { ...... };
D d;
```

应把类A定义为类B和类C的**虚基类**

```
class A { int x; ......};
class B: virtual public A {......};
class C: virtual public A {......};
class D: public B, public C {......};
D d;
```

虚基类构造函数的调用

创建包含虚基类的类对象时：

- 虚基类的构造函数由该类的构造函数直接调用。
- 虚基类的构造函数优先非虚基类的构造函数执行。

7.5 类之间的聚合/组合关系

类之间的整体与部分的关系可以是

- 聚合（aggregation）关系
- 组合（composition）关系

聚合与组合的关系与区别

- 在聚合关系中，被包含的对象与包含它的对象独立创建和消亡，被包含的对象可以脱离包含它的对象独立存在

  在组合关系中，被包含的对象随包含它的对象创建和消亡，被包含的对象不能脱离包含它的对象独立存在
- 聚合类的成员对象一般是采用对象指针表示，用于指向被包含的成员对象，而被包含的成员对象是在外部创建，然后加入到聚合类对象中来的

  组合类的成员对象一般直接是对象，有时也可以采用对象指针表示，但不管是什么表示形式，成员对象一定是在组合类对象内部创建并随着组合类对象消亡

继承与聚合/组合的比较

- 继承与封装存在矛盾，聚合/组合与封装则不存在这个矛盾。
- 在基于继承的代码复用中，一个类向外界提供两种接口：
  - public：对象（实例）用户
  - public+protected：派生类用户

  在基于聚合/组合的代码复用中，一个类对外只需一个接口：public。

- 继承的代码复用功能常常可以用组合来实现
- 继承更容易实现子类型：在C++中，public继承的派生类往往可以看成是基类的子类型。在需要基类对象的地方可以用派生类对象去替代。发给基类对象的消息也能发给派生类对象。

  具有聚合/组合关系的两个类不具有子类型关系