第3章

虚函数与多态性

内容安排

- 3.1 多态性
- 3.2 静态绑定和动态绑定
- 3.3 赋值兼容规则
- 3.4 虚函数
- 3.5 纯虚函数和抽象类

3.1 多态性

- 多态性是指同样的消息被类的不同对象接收时导致完全不同行为的一种现象。
- 这里的消息实质是对类的成员函数的调用，不同的行为指不同的实现，也就是调用了不同的函数。

3.2 静态绑定和动态绑定

- 绑定(Binding，也称联编)：
  - 是指把一个消息和一个方法（或成员函数）联系在一起
  - 也就是把一个函数名与其实现代码联系在一起
  - 实质是把一个标识符名和一个存储地址联系在一起的过程。
- 根据实现绑定的阶段的不同，绑定可分为2种：
  - 静态绑定（也叫静态联编）
  - 动态绑定（也叫动态联编）

两种绑定过程分别对应着多态的两种实现

- 静态绑定是在程序编译阶段实现的多态性，这种多态性称为静态多态性，也称编译时的多态性，可通过函数重载实现。
- 动态绑定是在程序执行阶段实现的多态性，这种多态性称为动态多态性，也称运行时的多态性，可通过继承、虚函数、基类的指针或引用等技术来实现

3.3 赋值兼容规则

- 赋值兼容是指不同类型数据之间的自动转换和赋值。
- 赋值兼容规则是指在公有继承情况下，对于某些场合，一个派生类的对象可以作为基类对象来使用，也就是在需要基类对象的任何地方都可以使用公有派生类的对象来替代。

3.3 赋值兼容规则

- 赋值兼容规则包括以下三种情况（假设类B为基类，类D为类B的公有派生类）：

（1）基类对象 = 派生类的对象

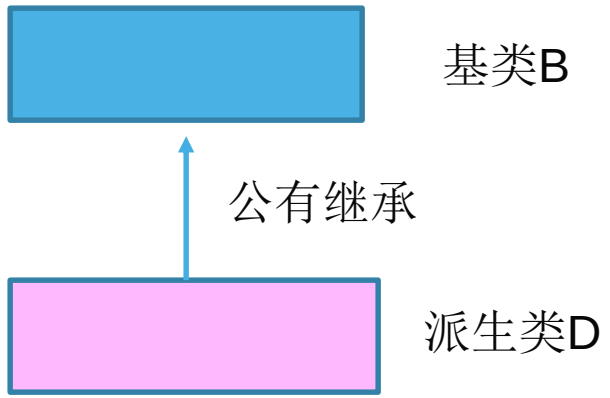
```
D d;
B b;
b=d;
```

（2）派生类的对象可以初始化为基类的引用

```
D d;
B &b=d;
```

（3）指向基类类型的指针 = 派生类对象的地址

```
D d;
B *bp=&d;
```



3.3 赋值兼容规则

- 对于通过基类类型的指针（或引用）对成员函数的调用，编译时无法确定对象的类，而只有在运行时才能确定调用哪个类的成员函数。
- 【例3-1】Base, Derived类，全局函数fun()接受 Base &参数，测试赋值兼容规则

```
#include<iostream>
using namespace std;
class Base
{
private:
        int b;
public:
        Base(int i=10)
        {
                b = i;
        }
        void show();
};

void Base::show()
{
        cout<<"In class Base, show() is called."<<endl;
}
```

```
class Derived : public Base
{
private:
        int d;
public:
        Derived(int j=20)
        {
                d = j;
        }
};
void fun(Base &ref)
{
        ref.show();
}
int main()
{
        Base b;
        fun(b);
        Derived d;
        fun(d);
        return 0;
}
```

```
In class Base, show() is called.
In class Base, show() is called.
```

基类指针（引用）和公有派生类对象

- 【例3-2】基类指针（引用）、派生类指针（引用）、基类对象和派生类对象四者间组合的使用情况示例。

```
#include<iostream>
using namespace std;
class Base
{
private:
        int b;
public:
        Base(int i=10)
        {
                b = i;
        }
        void show();
        int getb()
        {
                return b;
        }
};
void Base::show()
{
        cout<<"b="<<b<<endl;
}
```

```
class Derived : public Base
{
private:
        int d;
public:
        Derived(int j=20)
        {
                d = j;
        }
        void show()
        {
                cout<<"d="<<d<<endl;
         }
        int getd()
        {
                return d;
        }
};
```

```
int main()
{
    Base b(1), *pb;
    Derived d(2), *pd;
    pb = &b;          //基类指针指向基类对象
    pb->show(); //输出基类的b
    pd = &d;          //派生类指针指向派生类对象
    pd->show();       //输出派生类的d
    pb = &d;          //基类指针指向派生类对象
    pb->show();       //基类指针只能访问基类的show
    pd = (Derived *)&b;//派生类指针指向基类对象
    Base &ba = d;       //基类引用指向派生类对象
    ba.show();        //基类引用只能访问基类的show
    return 0;
}
```

基类指针仅能访问派生类中的基类部分，隐式类型转换

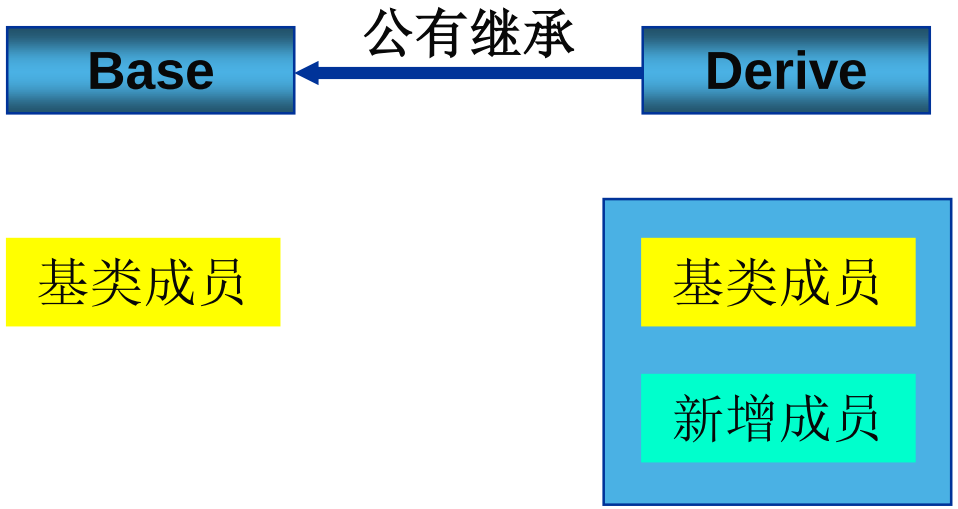
```
b=1
d=2
b=10
b=10
```

不提供隐式类型转换，需强制类型转换

基类引用仅能访问派生类中的基类部分，隐式类型转换

如何通过基类指针(引用)访问派生类中定义的成员

基类指针指向公有派生类对象



基类指针仅能访问派生类中的基类部分，因此不能实现真正的多态，需要引入虚函数的概念。

3.4 虚函数

```
class 类名
{
    virtual 类型 函数名（参数表）；
};
```

说明

- （1）虚函数声明只能在类声明中的函数原型声明中，实现部分可以在类内也可以在类外，在类体外定义时，前面不能加“virtual”。
- （2）只有类的普通成员函数才能声明为虚函数，全局函数及静态成员函数不能声明为虚函数。
- （3）虚函数可以在一个或多个派生类中被重新定义。
  - 要求在派生类中重新定义时必须与基类中的函数原型完全相同。
  - 无论在派生类的相应成员函数前是否加上关键字virtual，都将其视为虚函数；
- （4）当一个类的成员函数声明为虚函数后，就可以在该类的（直接或间接）派生类中定义与其基类虚函数原型相同的函数。
  - 当用基类指针指向这些派生类对象时，系统会自动用派生类中的同名函数来代替基类中的虚函数。
  - 也就是说，当用基类指针指向不同派生类对象时，系统会在程序运行中根据所指向对象的不同，自动选择适当的成员函数，从而实现了运行时的多态性。

说明

- （5）静态成员函数不能声明为虚函数。因为静态成员函数不属于某一个对象，没有多态性的特征。
- （6）构造函数不能是虚函数。构造函数在对象创建时调用，完成对象的初始化，此时对象还没有完全建立，所以，将构造函数声明为虚函数是没有意义的。
- （7）析构函数可以是虚函数，且往往被声明为虚函数。

- 【例3-3】虚函数的定义与应用。

```
#include<iostream>
using namespace std;
class Base
{
private:
        int b;
public:
        Base(int i=10)
        {
                b = i;
        }
        virtual void show()
        {
                cout<<"Base class\n";
        }
        int getb()
        {
                return b;
        }
};
class Derived1 : public Base
{
private:
        int d;
public:
        Derived1(int j=20)
        {
                d = j;
        }
        void show()
        {
                cout<<"Derived class 1\n";
        }
        int getd()
        {
                return d;
        }
};
class Derived2 : public Base
{
public:
        void show()
        {
                cout<<"Derived class 2\n";
        }
};
int main()
{
        Base b(1), *pb;
        Derived1 d1(2);
        Derived2 d2;
        pb = &b;
        pb->show();
        pb = &d1;
        pb->show();
        Base &bref = d2;
        bref.show();
        return 0;
}
```

```
Base class
Derived class 1
Derived class 2
```

虚析构函数

- 虚析构函数使用virtual说明，格式为：
- virtual ~<类名>( )
- 基类中虚析构函数的作用：当用一个基类的指针删除一个派生类的对象时，派生类的析构函数会被调用
- 如果一个类的析构函数是虚函数，那么，由它派生而来的所有子类的析构函数也是虚函数。
- 示例 虚析构函数.cpp

```
#include<iostream>
using namespace std;
class Base
{
public:
        Base(){   }
        virtual ~Base()
         {cout << "destructor in Base\n";  }
        virtual void show()
        {
                cout<<"show in Base\n";
         }
};
class Derived : public Base
{
public:
        Derived(){ }
        ~Derived(){
                cout<<"destructor in Derived\n";
        }
        virtual void show()
        {
                cout<<"show in Derived\n";
        }
};
```

```
int main()
{
        Base * p =  new Derived;
        p->show();
        delete p;
        return 0;
}
```

```
show in Derived
destructor in Derived
destructor in Base
```

返回

3.5 纯虚函数与抽象类

- 在定义一个表达抽象概念的基类时，有时可能会无法给出某些成员函数的具体实现。这时，就可以将这些函数声明为纯虚函数。

  virtual 类型 函数名（参数表）=0；

  - 纯虚函数是只在基类中声明虚函数但未给出具体的函数定义体；
  - 它的具体定义放在各派生类中。

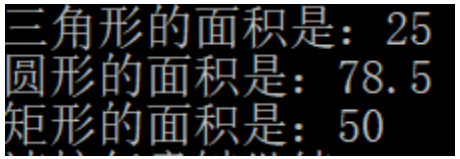
抽象类

- 类体内声明了纯虚函数的类，称为抽象类。
- 抽象类的主要作用：通过它为一个类族建立一个公共的接口，使它们能够更有效地发挥多态特性。
- 抽象类声明了一族派生类的共同接口，而接口的完整实现，即纯虚函数的函数体，要由派生类自己定义。
- 【例3-4】设计一个抽象类Shape，它表示具有形状的东西，在它下面可以派生出多种具体形状，比如直角三角形、圆形、矩形等，并求出各种形状的面积。

```
#include<iostream>
using namespace std;
class Shape
{
protected:
        double x,y;
public:
        void set(double i=0, double j=0)
        {
                x = i;
                y = j;
        }
        virtual void area()=0;
};
```

```
class Triangle: public Shape
{
public:
        void area()
        {
                cout<<"三角形的面积是： "<<0.5*x*y<<endl;
        }
};
class Circle: public Shape
{
public:
        void area()
        {
                cout<<"圆形的面积是： "<<3.14*x*x<<endl;
        }
};
class Rectangle: public Shape
{
public:
        void area()
        {
                cout<<"矩形的面积是： "<<x*y<<endl;
        }
};
```

```
int main()
{
    Shape *p;
    Triangle t;
    Circle c;
    Rectangle r;
    p = &t;
    p->set(5,10);
    p->area();
    p = &c;
    p->set(5);
    p->area();
    p = &r;
    p->set(5,10);
    p->area();
    return 0;
}
```



使用纯虚函数与抽象类的注意事项

（1）抽象类只能用作基类来派生新类，不能声明抽象类的对象，但可以声明指向抽象类的指针变量和引用变量。

（2）抽象类中可以有多个纯虚函数。

（3）抽象类中也可以定义其他非纯虚函数。

（4）抽象类派生出新类之后，如果在派生类中没有重新定义基类中的纯虚函数，则必须再将该虚函数声明为纯虚函数，这时，这个派生类仍然是一个抽象类；

（5）在一个复杂的类继承结构中，越上层的类抽象程度就越高，有时甚至无法给出某些成员函数的具体实现。

（6）引入抽象类的目的主要是为了能将相关类组织在一个类继承结构中，并通过抽象类来为这些相关类提供统一的操作接口。