C++ Programming

多态性与虚函数II

Polymorphism & Virtual Functions

2025年4月14日

学而不厌 诲人不倦

Chapter 6 多态性与虚函数

1. 虚函数的特点

C++ 虚函数对于多态具有决定性的作用，有虚函数才能构成多态。

1. 只需要在**虚函数声明处**加上 **virtual 关键字**，**函数定义**处可以加也可以不加。
2. 只将基类中的函数声明为虚函数，所有派生类中具有遮蔽关系的同名函数都将自动成为虚函数。
3. 在基类中定义了虚函数时，如果派生类没有定义新的函数来遮蔽此函数，那么将使用基类的虚函数。
4. 只有派生类的虚函数覆盖基类的虚函数（函数原型相同）才能构成多态（通过基类指针访问派生类函数）。
5. 构造函数不能是虚函数。
6. 析构函数可以声明为虚函数，而且有时候必须要声明为虚函数

➢ 2. 虚函数构成多态的条件

1) 必须存在继承关系；
2) 继承关系中必须有同名的虚函数，并且它们是覆盖关系（函数原型相同）。
3) 存在基类的指针，通过该指针调用虚函数。

```
#include <iostream>
using namespace std;
//基类Base
class Base{
public:
  virtual void func();
  virtual void func(int);
};
void Base::func(){
  cout<<"void Base::func()"<<endl;
}
void Base::func(int n){
  cout<<"void Base::func(int)"<<end
```

```
//派生类Derived
class Derived: public Base{
public:
  void func();
  void func(char *);
};
void Derived::func(){
  cout<<"void Derived::func()"<<endl;
}
void Derived::func(char *str){
  cout<<"void Derived::func(char *)"<<endl;
}
```

6.3 虚函数

➢ Demo01

```
#include <iostream>
using namespace std;
//基类Base
class Base{
public:
  virtual void func();
  virtual void func(int);
};
void Base::func(){
  cout<<"void Base::func()"<<endl;
}
void Base::func(int n){
  cout<<"void Base::func(int)"<<end
```

```
//派生类Derived
class Derived: public Base{
public:
  void func();
  void func(char *);
};
void Derived::func(){
  cout<<"void Derived::func()"<<endl;
}
void Derived::func(char *str){
  cout<<"void Derived::func(char *)"<<endl;
}
```

```
int main(){
  Base *p = new Derived();
  p -> func(); //输出void Derived::func()
  p -> func(10); //输出void Base::func(int)
  p -> func("http://c.biancheng.net");
  //compile error
  return 0;
}
```

➢ 3. 虚析构函数

当派生类的对象撤销时一般先调用派生类的析构函数，然后调用基类的析构函数。

用**new运算符**建立一个动态对象，如基类中有析构函数，并且定义了一个**指向基类的指针变量**。

在程序中用带指针参数的**delete运算符撤销对象**时，系统**只会执行基类的析构函数**，而不执行派生类的析构函数。

➢ 虚析构函数--Demo02

```
#include <iostream>
using namespace std;
class Point
{
public:
    Point() {}
    virtual ~Point() { cout << "executing Point destructor" << endl; }
};
```

```
int main()
{
    Point *p = new Circle;
    delete p;
    return 0;
}
```

```
class Circle : public Point
{
public:
   Circle() {}
   ~Circle() { cout << "executing Circle destructor" << endl; }
private:
   int radus;
};
```

➢ 纯虚函数与抽象类

在C++中，可以将虚函数声明为纯虚函数，语法格式为：

virtual 返回值类型 函数名 (函数参数) = 0;

纯虚函数没有函数体，只有函数声明，在虚函数声明的结尾加上=0，表明此函数为纯虚函数。最后的=0并不表示函数返回值为0，它只起形式上的作用，告诉编译系统“这是纯虚函数”。

包含纯虚函数的类称为抽象类（Abstract Class）。

之所以说它抽象，是因为它无法实例化，也就是无法创建对象。

纯虚函数没有函数体，不是完整的函数，无法调用，也无法为其分配内存空间。

抽象类通常是作为基类，让派生类去实现纯虚函数。派生类必须实现纯虚函数才能被实例化。

➢ 纯虚函数与抽象类—Demo03

```
#include <iostream>
using namespace std;
//线
class Line{
public:
   Line(float len);
   virtual float area() = 0;
   virtual float volume() = 0;
protected:
   float m_len;
};
Line::Line(float len): m_len(len){ }
```

```
//矩形
class Rec: public Line{
public:
   Rec(float len, float width);
   float area();
protected:
   float m_width;
};
Rec::Rec(float len, float width): Line(len),
m_width(width){ }
float Rec::area(){ return m_len * m_width; }
```

➢ 纯虚函数与抽象类—Demo03

```
//长方体
class Cuboid: public Rec{
public:
    Cuboid(float len, float width, float height);
    float area();
    float volume();
protected:
    float m_height;
};

Cuboid::Cuboid(float len, float width, float height): Rec(len, width),
m_height(height){ }

float Cuboid::area(){ return 2 * ( m_len*m_width + m_len*m_height +
m_width*m_height); }

float Cuboid::volume(){ return m_len * m_width * m_height; }
```

➢ 纯虚函数与抽象类—Demo03

```
//正方体
class Cube: public Cuboid{
public:
  Cube(float len);
  float area();
  float volume();
};
Cube::Cube(float len): Cuboid(len, len, len){ }
float Cube::area(){ return 6 * m_len * m_len; }
float Cube::volume(){ return m_len * m_len * m_len; }
```

➢ 纯虚函数与抽象类—Demo03

```
int main(){
    Line *p = new Cuboid(10, 20, 30);
    cout<<"The area of Cuboid is "<<p->area()<<endl;
    cout<<"The volume of Cuboid is "<<p->volume()<<endl;
    p = new Cube(15);
    cout<<"The area of Cube is "<<p->area()<<endl;
    cout<<"The volume of Cube is "<<p->volume()<<endl;
    return 0;
}
```

➢ 纯虚函数说明

1)一个类可以说明一个或多个纯虚函数。所在的抽象类，不能直接进行实例化。
2) 一个纯虚函数就可以使类成为抽象基类，但是抽象基类中除了包含纯虚函数外，还可以包含其它的成员函数（虚函数或普通函数）和成员变量。
3) 只有类中的虚函数才能被声明为纯虚函数，普通成员函数和顶层函数均不能声明为纯虚函数。

```
//顶层函数不能被声明为纯虚函数
void fun() = 0; //compile error
class base{
public :
    //普通成员函数不能被声明为纯虚函数
    void display() = 0; //compile error
};
```

➢ 抽象类说明

1. 带有纯虚函数的类是抽象类，只能用作其他类的基类，不能定义对象。
2. 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刻画了一组子类的公共操作接口的通用语义，这些接口的语义也传给子类。一般而言，抽象类只描述这组子类共同操作接口，而完整的实现留给子类。

3. 从一个抽象类派生的类**必须提供纯虚函数的实现代码**或在该派生类中**仍将它说明为纯虚函数**，否则编译错。
4. **抽象类不能用作参数类型、函数返回类型或显式转换的类型**，但可以说明指向抽象类的指针和引用，此指针可以指向它的派生类，实现多态性
5. 构造函数不能是虚函数，析构函数可以是虚函数。

Demo04

```
#include <iostream>
using namespace std;
class Shape
{
public:
	virtual float area() const { return 0.0; } // 虚函数
	virtual float volume() const { return 0.0; } // 虚函数
	virtual void shapeName() const = 0; // 纯虚函数
};
```

Demo04

```
class Point : public Shape // Point是Shape的公用派生类
{
protected:
        float x, y;
public:
        Point(float = 0, float = 0);
        void setPoint(float, float);
        float getX() const { return x; }
        float getY() const { return y; }
        // 对纯虚函数进行定义
        virtual void shapeName() const { cout << "Point:"; }
        friend ostream &operator<<(ostream &, const Point &);
};
```

Demo04

```
Point::Point(float a, float b)
{
	x = a;
	y = b;
}

void Point::setPoint(float a, float b)
{
	x = a;
	y = b;
}

ostream &operator<<(ostream &output, const Point &p)
{
	output << "[" << p.x << "," << p.y << "]";
	return output;
}
```

➢ Demo04

```
class Circle : public Point // 声明Circle类
{
protected:
        float radius;

public:
        Circle(float x = 0, float y = 0, float r = 0);
        void setRadius(float);
        float getRadius() const;
        virtual float area() const;
        // 对纯虚函数进行再定义
        virtual void shapeName() const { cout << "Circle:"; }
        friend ostream &operator<<(ostream &, const Circle &);
};
```

Demo04

```
Circle::Circle(float a, float b, float r) : Point(a, b), radius(r) {}
void Circle::setRadius(float r)
{
radius = r;
}

float Circle::getRadius() const { return radius; }

float Circle::area() const
{
return 3.14159 * radius * radius;
}

ostream &operator<<(ostream &output, const Circle &c)
{
output << "[" << c.x << "," << c.y << "], r=" << c.radius;
return output;
}
```

Demo04

```
// 声明Cylinder类
class Cylinder : public Circle
{
public:
    Cylinder(float x = 0, float y = 0, float r = 0, float h = 0);
    void setHeight(float);
    float getHeight() const;
    virtual float area() const;
    virtual float volume() const;
    // 对纯虚函数进行再定义
    virtual void shapeName() const { cout << "Cylinder:"; }
    friend ostream &operator<<(ostream &, const Cylinder &);

protected:
    float height;
};
```

6.4 纯虚函数与抽象类

➢ Demo04

```
Cylinder::Cylinder(float a, float b, float r, float h):Circle(a, b, r), height(h) {}

void Cylinder::setHeight(float h) { height = h; }
float Cylinder::getHeight() const { return height; }

float Cylinder::area() const
{
        return 2 * Circle::area() + 2 * 3.14159 * radius * height;
}

float Cylinder::volume() const
{
        return Circle::area() * height;
}

ostream &operator<<(ostream &output, const Cylinder &cy)
{
        output << "[" << cy.x << "," << cy.y << "], r=" << cy.radius << ", h=" << cy.height;
        return output;
}
```

Demo04

```
int main()
{
	Point point(3.2, 4.5); // 建立Point类对象point
	Circle circle(2.4, 12, 5.6); // 建立Circle类对象circle
	Cylinder cylinder(3.5, 6.4, 5.2, 10.5);
	// 建立Cylinder类对象cylinder
	point.shapeName(); // 静态关联
	cout << point << endl;

	circle.shapeName(); // 静态关联
	cout << circle << endl;

	cylinder.shapeName(); // 静态关联
	cout << cylinder << endl<< endl;
```

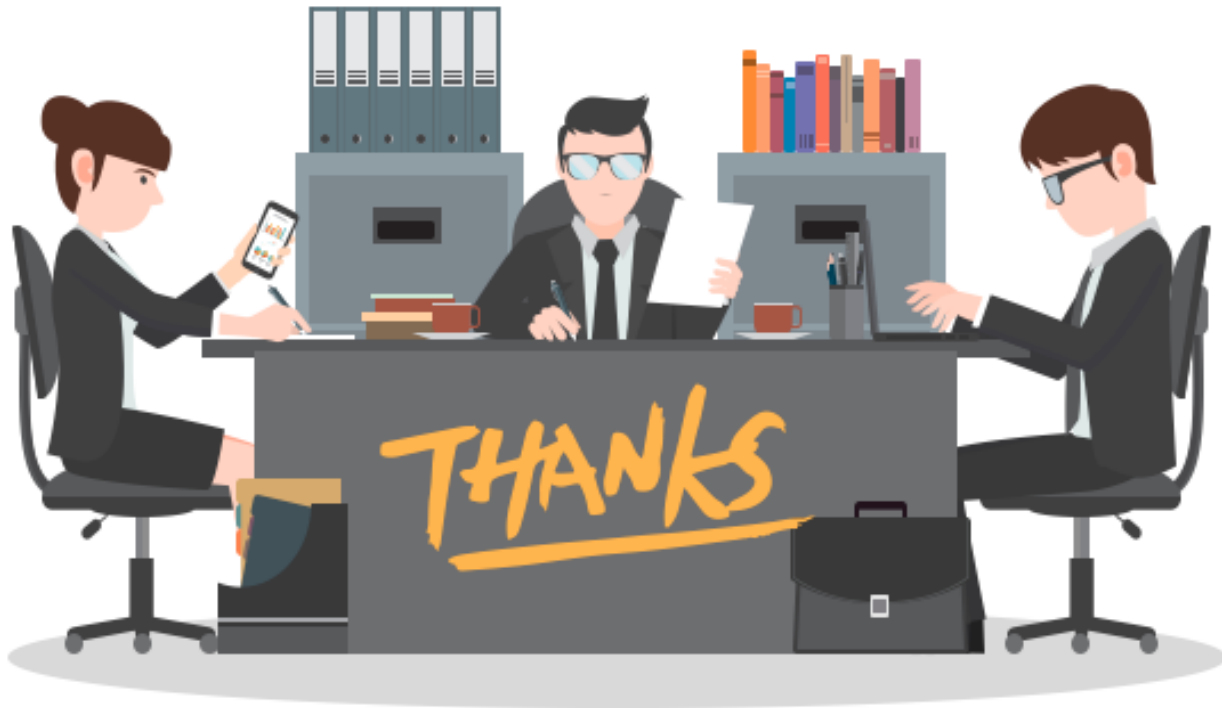
Demo04

```
    Shape *pt; // 定义基类指针
    pt = &point; // 指针指向Point类对象
    pt->shapeName(); // 动态关联
    cout << "x=" << point.getX() << ",y=" << point.getY() << "\narea=" << pt->area() << "\nvolume=" << pt-
    >volume() << "\n\n";

    pt = &circle; // 指针指向Circle类对象
    pt->shapeName(); // 动态关联
    cout << "x=" << circle.getX() << ",y=" << circle.getY() << "\narea=" << pt->area() << "\nvolume=" << pt-
    >volume() << "\n\n";

    pt = &cylinder; // 指针指向Cylinder类对象
    pt->shapeName(); // 动态关联
    cout << "x=" << cylinder.getX() << ",y=" << cylinder.getY() << "\narea=" << pt->area() << "\nvolume=" << pt-
    >volume() << "\n\n";
    return 0;
}
```

小结

Chapter 6 多态性与虚函数

- 6.1 多态性的概念
- 6.2 多态的典型实例
- 6.3 虚函数
- 6.4 纯虚函数与抽象类



Thank You !

Q & A