面向对象程序设计
420420

虚函数

- 1、虚函数实现多态的条件
- 2、类成员函数的指针与多态性

38.1 虚函数实现多态的条件

- 被virtual关键字修饰的成员函数，就是虚函数。虚函数的作用就是实现多态性——以共同的方法，对不同的对象采取不同的策略。

1. 虚函数的定义

- 虚函数只能是类中成员函数，且不能是静态的。在成员函数定义或声明前面加上关键字virtual，即定义了虚函数：

```
class 类名 { //类体
        …
        virtual 返回类型 函数名(形式参数列表); //虚函数
        …
};
```

```
class Point { //Point类表示平面上的点
        …
        virtual double area(); //虚函数声明
        virtual double volumn() { return 0; } //虚函数定义
};
```

- 需要注意，virtual只在类体中使用。
- 当在派生类中定义了一个同名的成员函数时，只要该成员函数的参数个数、参数类型以及返回类型与基类中同名的虚函数完全一样，则派生类的这个成员函数无论是否使用virtual，它都将成为一个虚函数。
- 程序员习惯给派生类的同名函数也加上virtual，便于阅读理解。
- 利用虚函数，可在基类和派生类中使用相同的函数名，定义函数的不同实现，从而实现“一个接口，多种方式”。当用基类指针或引用对虚函数进行访问时，系统将根据运行时指针或引用所指向或引用的实际对象来确定调用对象所在类的虚函数版本。

- 关键字virtual指示C++编译器在调用虚函数时进行动态绑定。这种多态性是程序运行到相应的语句时才动态确定的，所以称为运行时的多态性。

- 不过，使用虚函数并不一定产生多态性，也不一定使用动态绑定。例如，在调用中对虚函数使用成员名限定，可以强制C++对该函数的调用使用静态绑定。

【例38.1】

```
 1 #include  <iostream>
 2 using  namespace  std;
 3 class  Point {   //Point基类，表示平面上的点
 4         double  x,y; //私有数据成员，坐标值
 5    public:
 6         Point(double  x1=0,double  y1=0)  : x(x1),y(y1)  {  } //构造函数
 7         virtual  double  area()  { return  0;  } //虚函数
 8 };
 9 class  Circle:public  Point { //Circle派生类，表示圆
10         double  r;    //私有数据成员，半径
11    public:
12         Circle(double  x,double  y,double  r1):Point(x,y),r(r1)  {  }   //构造函数
13         virtual  double  area()  { return  3.14*r*r;  } //虚函数
14 };
```

```
15 int main()
16 {
17         Point a(2.5,2.5); Circle c(2.5,2.5,1);
18         Point *pc=&c;
19         cout<<“Circle  area=”<<pc->area()<<endl;//动态绑定
20         cout<<“Circle  area=”<<pc->Point::area()<<endl; //静态绑定
21        return 0;
22 }
```

运行结果：

```
Circle area=3.14
Circle area=0
```

2. 虚函数实现多态的条件

- ①类之间的继承关系满足赋值兼容性规则；
- ②改写了同名的虚函数；
- ③根据赋值兼容性规则使用指针（或引用）。

- 满足前两条并不一定产生动态联编，必须同时满足第3条才能保证实现动态联编。

第3条分为两种情况：

- （1）使用基类指针（或引用）访问虚函数。例如：

```
Point  *p=new  Circle;  //基类指针p指向派生类
cout<<p->area();  //动态联编
```

- （2）把指针（或引用）作为函数参数，这个函数不一定是类的成员函数，可以是普通函数，而且可以重载。例如：

```
void  fun(Point  *p)     //普通函数，参数是基类的指针
{  cout<<p->area();  }  //动态联编
```

38.2 类成员函数的指针与多态性

- 在派生类中，当一个指向基类成员函数的指针指向一个虚函数，并且通过指向对象的基类指针（或引用）访问这个虚函数时，仍将发生多态性。

38.2 类成员函数的指针与多态性

【例38.2】

```
#include <iostream>
using namespace std;
class Base {    //基类
        public: virtual void print() { cout<<"Base"<<endl; } //虚函数
};
class Derived: public Base {    //派生类
        public: void print() { cout<<"Derived"<<endl; } //虚函数
};
void display(Base *p,void(Base::*pf)())
          { (p->*pf)(); }
int main()
{
        Derived d;      Base b;
        display(&d,&Base::print); //产生多态，输出Derived
        display(&b,&Base::print); //输出Base
        return 0;
}
```

何时需要虚函数？

（1）首先看成员函数所在的类是否会作为基类。然后看成员函数在类的继承后有无可能被更改功能，如果希望派生类更改其功能的，一般应该将它声明为虚函数。

（2）如果成员函数在类被继承后功能不需修改，或派生类用不到该函数，则不要把它声明为虚函数。不要仅仅考虑到要作为基类而把类中的所有成员函数都声明为虚函数。

（3）应考虑对成员函数的调用是通过对象名还是通过基类指针或引用去访问，如果是通过基类指针或引用去访问的，则应当声明为虚函数。

- 使用虚函数，系统要增加一定的空间开销用来存储虚函数表，但系统在进行动态联编时的时间开销是很少的，因此，多态性是高效的。

虚函数

- 3、虚析构函数
- 4、纯虚函数和抽象类

- 派生类的对象从内存中撤销时一般先调用派生类的析构函数，然后再调用基类的析构函数。

- 但是，如果用new运算符建立了派生类对象，且定义了一个基类的指针指向这个对象，那么当用delete运算符撤销对象时，系统会只执行基类的析构函数，而不执行派生类的析构函数，因而也无法对派生类对象进行真正的撤销清理操作。

例如：

```
Point  *pp=new  Circle;  //基类指针指向派生类
delete  pp;  //仅执行基类析构函数
```

- 如果希望“delete pp”执行Circle的析构函数，那么基类Point的析构函数要声明为虚函数，称为虚析构函数。

- 如果将基类的析构函数声明为虚函数，由该基类所派生的所有派生类的析构函数也都自动成为虚函数，即使派生类的析构函数与基类的析构函数名字不相同。

【例38.3】

```
1  #include  <iostream>
2  using  namespace  std;
3  class  Base  //基类
4  {
5  public:
6      Base()  {};   //构造函数
7      virtual  ~Base()  {};  //虚析构函数
8      virtual  void  fun()  {  cout  <<  "in  class  Base!"  <<  endl;  }; //虚函数
9  };
```

```
10  class  Derived  :  public  Base  //派生类
11  {
12  public:
13      Derived()  {};   //构造函数
14      ~Derived(){  cout<<"the  destructor  of  class  Derived!"<<endl;  }; //虚函数
15       void  fun()  {  cout  <<  "in  class  Derived!"  <<  endl;  }; //虚函数
16  };
17  int  main()
18  {
19          Base  *pTest  =  new  Derived; //基类的指针pTest指向派生类的对象
20          pTest->fun();        //动态绑定，调用派生类fun
21         delete  pTest;        //调用派生类和基类的析构函数
22         return  0;
23  }
```

- 当基类的析构函数为虚函数时，无论指针指的是同一类族中的哪一个类对象，系统总会采用动态联编，调用正确的析构函数，对该对象进行清理。

- C++支持虚析构函数，但不支持虚构造函数，即构造函数不能声明为虚函数。

1. 纯虚函数

- 在许多情况下，不能在基类中为虚函数给出一个有意义的定义，这时可以将它说明为纯虚函数（pure virtual function），将具体定义留给派生类去做。纯虚函数的定义形式为：

```
virtual  返回类型  函数名(形式参数列表)=0;
```

- 即在虚函数的原型声明后加上“=0”，表示纯虚函数根本就没有函数体。

- 纯虚函数的作用是在基类中为其派生类保留一个函数的名字，以便派生类根据需要对它进行定义。如果在一个类中声明了纯虚函数，而在其派生类中没有对该函数定义，则该虚函数在派生类中仍然为纯虚函数。

- 包含有纯虚函数的类称为抽象类（abstract class）。一个抽象类只能作为基类来派生新类，所以又称为抽象基类（abstract base class）。抽象类不能定义对象 。

- 如果在派生类中给出了抽象类的纯虚函数的实现，则该派生类不再是抽象类。否则只要派生类仍然有纯虚函数，则派生类依然是抽象类。抽象类至少含有一个虚函数，而且至少有一个虚函数是纯虚函数。

【例38.4】

```
#include <iostream>
using namespace std;
class Shape { //基类，抽象类
public:
    virtual double area() =0; //纯虚函数
    virtual double volumn() =0; //纯虚函数
};
class Circle : public Shape { //Circle类，派生类，表示圆
public:
    Circle(double a):r(a) { }     //构造函数
    virtual double area() { return 3.1415926*r*r; } //虚函数
    virtual double volumn() { return 0; }; //虚函数
private:
    double r; //私有数据成员，半径
};
```

```
16 class Cylinder : public Circle { //Cylinder表示圆柱体
17 public:
18     Cylinder(double a,double b):Circle(a),h(b) { } //构造函数
19     virtual double volumn() { return area()*h; }; //虚函数
20 private:
21     double h; //私有数据成员，高度
22 };
23 int main()
24 {
25     Circle a(10.0); //定义Circle对象a
26     Cylinder b(1.0,10.0); //定义Cylinder对象b
27     cout<<a.area()<<" , "<<b.volumn()<<endl; //静态绑定
28     Shape *pb; //定义基类指针pb
29     pb=&b; //pb指向 Cylinder 对象b
30     cout<<pb->area()<<" , "<<pb->volumn()<<endl; //动态绑定
31     return 0;
32 }
```