

# 多态性与虚函数



朱晓旭

苏州大学计算机科学与技术学院



# 多态性概述

所谓多态性就是不同对象收到相同的消息时，产生不同的动作。

直观地说，多态性是指用一个名字定义不同的函数，这些函数执行不同但又类似的操作，从而可以使用相同的调用方式来调用这些具有不同功能的同名函数。



## 多态的实现

多态从实现的角度来讲可以划分为两类：  
**编译时的多态**和**运行时的多态**。

**编译时的多态**是通过静态联编来实现的。静态联编就是在编译阶段完成的联编。编译时多态性主要是通过函数重载和运算符重载实现的。

**运行时的多态**是用动态联编实现的。动态联编是运行阶段完成的联编。运行时多态性主要是通过虚函数来实现的。



## 虚函数

虚函数提供了一种更为灵活的多态性机制。虚函数允许函数调用与函数体之间的联系在运行时才建立，也就是在运行时才决定如何动作，即所谓的动态联编。

## 虚函数的引例1。

```
#include<iostream.h>
```

```
class A{
```

```
public:
```

```
    void show(){ cout<<"A"; }
```

```
};
```

```
class B:public A {
```

```
public:
```

```
    void show(){ cout<<"B"; }
```

```
};
```

```
main()
```

```
{  A a,*pc;
```

```
    B b;
```

```
    pc=&a; pc->show();
```

```
    pc=&b; pc->show();
```

```
    return 0;
```

```
}
```



# 虚函数的作用和定义

## 1. 虚函数的作用

虚函数同派生类的结合可使C++支持运行时的多态性，实现了在基类定义派生类所拥有的通用接口，而在派生类定义具体的实现方法，即常说的“同一接口，多种方法”，它帮助程序员处理越来越复杂的程序。

## 例2 虚函数的作用。

```
#include<iostream.h>
```

```
class Base {
```

```
public:
```

```
    Base(int x,int y)
```

```
    { a=x; b=y; }
```

```
    virtual void show()    //定义虚函数show()
```

```
    { cout<<"Base-----\n"; cout<<a<<" "<<b<<endl;}
```

```
private:
```

```
    int a,b; };
```

```
class Derived : public Base {
```

```
public:
```

```
    Derived(int x,int y,int z):Base(x,y){c=z; }
```


```
    void show()            //重新定义虚函数show()
```

```
    { cout<<"Derived-----\n"; cout<<c<<endl;}
```

```
private:
```

```
    int c;
```

```
};
```



```
void main()  
{  
    Base mb(60,60),*pc;  
    Derived mc(10,20,30);  
    pc=&mb;  
    pc->show();    //调用基类Base的show()版本  
    pc=&mc;  
    pc->show();    //调用派生类Derived的show()版本  
}
```

程序运行结果如下:

**Base-----**

**60 60**

**Derived-----**

**30**





## 2. 虚函数的定义

定义虚函数的方法如下：

**virtual 函数类型 函数名(形参表)**

**{**

**// 函数体**

**}**

### 例3 虚函数的定义举例。

```
#include<iostream.h>
```

```
class Grandam {
```

```
    public:
```

```
        virtual void introduce_self() // 定义虚函数introduce_self()
```

```
        { cout<<"I am grandam."<<endl; }
```

```
};
```

```
class Mother:public Grandam {
```

```
    public:
```

```
        void introduce_self() // 重新定义虚函数introduce_self()
```

```
        { cout<<"I am mother."<<endl;}
```

```
};
```


```
class Daughter:public Mother {
```

```
    public:
```

```
        void introduce_self() // 重新定义虚函数introduce_self()
```

```
        { cout<<"I am daughter."<<endl;}
```

```
};
```



```
void main()  
{  
    Grandam *ptr;  
    Grandam g;  
    Mother m;  
    Daughter d;  
    ptr=&g;  
    ptr->introduce_self();//调用基类Grandam的introduce_self()  
    ptr=&m;  
    ptr->introduce_self();// 调用派生类Mother的introduce_self()  
    ptr=&d;  
    ptr->introduce_self(); //调用派生类  
                                // Daughter的introduce_self()  
}
```

# 虚函数与重载函数的关系

在一个派生类中重新定义基类的虚函数是函数重载的另一种形式，但它不同于一般的函数重载。

- ◆ 普通的函数重载时，其函数的参数或参数类型必须有所不同，函数的返回类型也可以不同。

- ◆ 当重载一个虚函数时，也就是说在派生类中重新定义虚函数时，要求函数名、返回类型、参数个数、参数的类型和顺序与基类中的虚函数原型完全相同。

- ◆ 如果仅仅返回类型不同，其余均相同，系统会给出错误信息；


- ◆ 若仅仅函数名相同，而参数的个数、类型或顺序不同，系统将它作为普通的函数重载，这时将丢失虚函数的特性。



# 虚函数与析构函数

- 基类的析构函数通常需要是虚函数
  - 为何？
  - 为了便于基类的指针操作





```
#include<iostream>
using namespace std;
class ClxBase{
public:
    ClxBase() {} ;
    virtual ~ClxBase() {cout << "基类析构!" << endl;};
    virtual void DoSomething() { cout << "基类干活!" << endl; };
};

class ClxDerived : public ClxBase{
public:
    ClxDerived() {} ;
    ~ClxDerived() { cout << "派生类析构!" << endl; };
    void DoSomething() { cout << "派生类干活!" << endl; };
};

int main(){
    ClxBase *p = new ClxDerived;
    p->DoSomething();
    delete p;
    return 0;
}
```

# 纯虚函数和抽象类

## 纯虚函数


纯虚函数是一个在基类中说明的虚函数，它在该基类中没有定义，但要求在它的派生类中必须定义自己的版本，或重新说明为纯虚函数。

纯虚函数的定义形式如下：

**virtual 函数类型 函数名(参数表)=0;**

## 例4纯虚函数的使用。

```
#include<iostream.h>
class Circle {
public:
    void setr(int x){ r=x; }
    virtual void show()=0; // 纯虚函数
protected:
    int r;
};
class Area:public Circle{
public:
    void show(){ cout<<"Area is "<<3.14*r*r<<endl;}
}; // 重定义虚函数show( )
class Perimeter:public Circle{
public:
    void show(){cout<<"Perimeter is "<<2*3.14*r<<endl;}
}; // 重定义虚函数show( )
```



```
void main()  
{  
    Circle *ptr;  
    Area ob1;  
    Perimeter ob2;  
    ob1.setr(10);  
    ob2.setr(10);  
    ptr=&ob1;  
    ptr->show();  
    ptr=&ob2;  
    ptr->show();  
}
```



# 抽象类

如果一个类至少有一个纯虚函数，那么就称该类为抽象类。

抽象类只能作为其他类的基类来使用，不能建立抽象类对象，其纯虚函数的实现由派生类给出。





# 抽象类作用

## ■ 统一接口

- 使得子孙类都有希望的接口函数

- 便于开发可扩展的通用程序