第七章 多态性和虚函数

面向对象程序设计的四个重要特征：

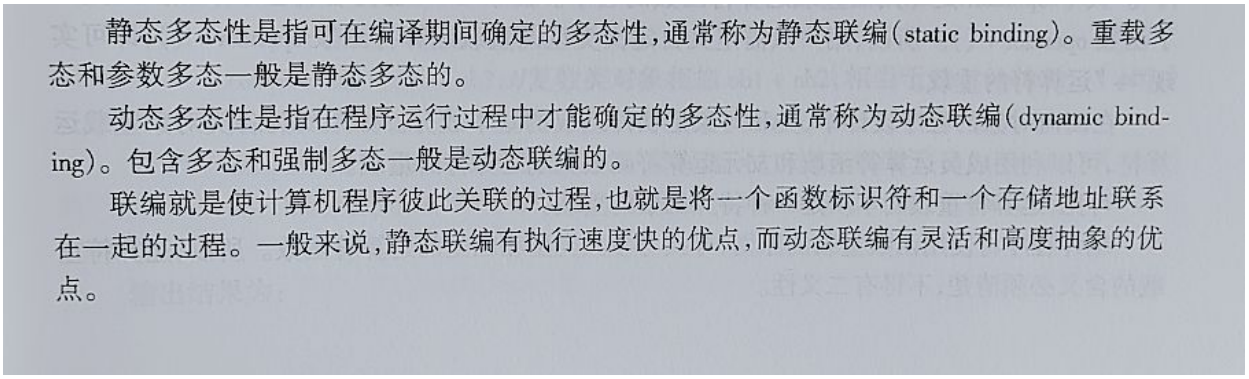
- 封装性是基础
- 继承性是关键
- 多态性是补充
- 抽象性贯穿始终

多态性概述

1.定义：多态性是指发出同样的消息被不同类型的对象接收时导致的不同行为
具体来说，是指类中具有相似功能的不同函数使用同一个名称，当调用这个同名函数的时候，根据需要完成不同功能。

2.分类

- 根据概念分：参数多态、包含多态、重载多态、强制多态（前两个是通用多态，后两个是专用多态）
  参数多态：由函数模板和类模板的实例化实现的多态（模板中定义的操作相同，操作对象类型可以不同）
  包含多态：由虚函数对各类中同名成员函数实现的多态
  重载多态：由函数重载、运算符重载等实现的多态由函数重载、运算符重载等实现的多态
  强制多态：将一个变元的类型强制转换
- 根据实现时机分：静态多态和动态多态



静态多态性是指可在编译期间确定的多态性，通常称为静态联编(static binding)。重载多态和参数多态一般是静态多态的。

动态多态性是指在程序运行过程中才能确定的多态性，通常称为动态联编(dynamic binding)。包含多态和强制多态一般是动态联编的。

联编就是使计算机程序彼此关联的过程，也就是将一个函数标识符和一个存储地址联系在一起的过程。一般来说，静态联编有执行速度快的优点，而动态联编有灵活和高度抽象的优点。

关于联编

在C语言中，每个函数名都对应一个不同的函数，所以用谁是谁，一对一，在编译过程就能完成联编，很明显的静态联编。

而在C++中，由于函数重载的缘故，就不能简单的一对一去联编了，编译器必须查看函数名以及函数参数才能确定使用哪个函数(在C++中，编译器看到的并不是我们所定义的函数名，我们所定义的每个函数，都有一个函数签名，就算我们函数名相同，但我们参数不同，类型不同，甚至是const与non-const的关系，都会导致函数签名不相同)
不过，这种程度在C/C++编译器中也能够在编译过程完成联编，这种在编译过程完成联编的就叫做静态联编。但C++中有一个东西使得编译器很难在编译阶段确定你要使用哪一个函数，那就是虚函数，因为编译器不知道用户将选择哪种类型的对象，所以，编译器必须生成能够在程序运行时选择正确的虚函数的代码，这种称为动态联编。

一般来说，编译器对非虚方法使用**静态联编**，对虚方法使用**动态联编**

运算符重载

第四章 函 数 > 运算符重载

虚函数

1.定义：虚函数是一种成员函数，而且是非static的成员函数。一个函数被说明或者定义为虚函数后，说明它在派生类中可能有多种不同的实现。

2.作用：应用程序不必为每一个子类的功能调用编写代码，只需要对抽象的父类进行处理

3.特点：

- 虚函数只能通过**(基类)指针或者引用**所表示的对象调用**！**
- **！！**构造函数不允许为虚函数，但析构函数允许为虚函数

虚函数与动态联编

1.虚函数的一般格式：

```
virtual 类型名 函数名(形参表)
        函数体
```

一个函数被说明或者定义为虚函数后，表明它在派生类中可能有多钟不同的实现，即重新定义。
但是，在派生类中重新定义时，其函数类型（包括返回类型、函数名、参数个数和类型以及各参数顺序）都必须与基类中的原型相同，否则则处理为静态联编

2.示例：

```cpp
#include<iostream>
using namespace std;

class Maker {
public:
        virtual void speak() {
                cout << "Maker" << endl;
        }
};
class sonOfMaker :public Maker {
public:
        void speak() {
                cout << "sonOfMaker" << endl;
        }
};

int main()
{
        sonOfMaker * m1 = new sonOfMaker;  //m1静态类型动态类型都是sonOfMaker
        Maker *m2 = m1;  //m2静态类型是Maker，动态类型是sonOfMaker
        m1->speak();  //输出sonOfMaker
        m2->speak();  //有virtual发生动态绑定，输出sonOfMaker。    无virtual发生静态绑定，输出Maker
        delete m1;
        return EXIT_SUCCESS;
}
```

虚析构函数

1.一般形式：

```cpp
virtual ~类名() 虚析构函数体
```

2.注意：

- 如果一个类的析构函数为虚函数，则由**它派生而来的所有派生类的析构函数均为虚析构函数**
- 一般使用delete运算符删除动态分配的对象

纯虚函数与抽象类

1.背景：在根基类中定义虚函数时必须给出一个函数体，但是这个函数体没有任何实际意义，这种情况下可以将该函数定义为纯虚函数。带有纯虚函数的类称为抽象类。

2.定义形式：

```
virtual 类型名 函数名(形参表)=0;
```

3.对抽象类的规定：

- **抽象类只能作为根基类，不能建立抽象类的对象**，但是可以说明抽象类的指针或者引用，以便使用这些指针和引用实现动态多态性；
- 抽象类不能作为函数返回类型、参数类型、也不能作为转换结果的类型；
- 抽象类主要作为继承结构中的根基类，继承抽象类纯虚函数的派生类仍然是一个抽象类，只有在派生类中给出基类纯虚函数的实现时，该派生类才成为可以建立对象的具体类。