面向对象的程序设计方法及其实现

张兴滔　（四川红泉仪表厂）

摘要:本文介绍了新一代面向对象的程序设计方法,论述了它的基本特性和概念以及如何在 C 语言环境中进行具体实现的方法。

一、引 言

面向对象的程序设计方法 OOP(Object-Oriented Programming)被誉为九十年代的程序设计方法,越来越多的程序设计语言加入 OOP 编程技术,已成大势所趋,如 Borland C++ V2.0、Turgo PASCAL V5.5 以上版本以及 Smalltalk 等,都先后引进了这种新的程序设计思想。下面谈谈 OPP 的一些基本概念和编程技术的优越性以及在 Borland C++(以下简称 C++)中的实现方法。

二、OOP 的出现和背景

OOP 真正形成以前,熟悉程序设计的人,大都采用从顶向下、逐步求精的方法来进行程序设计,但是在程序设计中常常遇到许多难解决的问题,而且这种方法受到许多尖锐的挑战。在这种情况下就提出了 OPP 这种全新的程序设计方法。

OOP 是一种人们为模仿建立现实世界模型的程序设计方法,这种思想源于对现实世界自然结构的认识。人们为了应付现实世界的复杂性,逐渐形成了很好的概括、分类和抽象的能力,并在此基础上应用于程序设计就形成了面向对象的概念,它与使用非面向对象编程语言相比,可设计出更加结构化、可扩充性、易移植和易维护的程序,它能够解决传统软件设计中的许多难题。下面就谈谈它的几个基本特征。

三、OOP 的基本特征

1.封装性(Encapsulation)

正是由于有了封装性,才使软件的可靠性得到了充分保证,使软件设计向模块化大大迈进了一步。

(1)具体含义。所谓封装性,就是指把数据和处理数据的代码衔接在一起,构成一个具有类类型的实例(即对象)。例如,开发一个数组之类的数据结构用以存储一些信息,另外又开发了一系列处理数据函数。在传统的程序设计方法下,人们就会把数据和相关的函数放入一个单独进行编译的源文件中,以便把代码数据本身作为模块来看待。其实这远远不够,比如数据和代码之间的隶属关系、访问权限以及存在范围都不能表达清楚;相反,正是由于封装性

的出现,彻底解决了上述问题,在 C++ 中通过关键字 struct、union 和 class 将数据(称作数据成员)和函数(称作成员函数) 封装成一个表达数据和函数关系的对象类。

(2)成员访问控制。对于 struct、class 二个关键字提供了对数组或类成员的存取控制,它是通过把适当的关键字(后带冒号)放在所影响的成员说明前来完成的。

私有的(protected):该关键字后的成员只能被在相同类中说明的成员函数访问;

保护的(protected):该关键字后的成员只能被在相同类中说明的成员函数访问,以及从该类派生的类的成员函数访问;

公有的(public):该关键字后的成员可以与类说明在同一作用域内的任一地方访问;

在对类的定义中同时考虑了代码和数据的结合,使数据引导代码流代码处理数据,这就可以避免使用错误数据调用正确的函数和使用正确数据调用错误的函数的危险。例如,定义屏幕上一个物理象素点的类:

```
class Point  {
   private :                  / * 表明数据访问的限制和约束条件
     int X,Y;                / * 定义私有数据
   protected:                / * 表明数据访问的限制和约束条件
       boolean visible;      / * 定义保护数据
   public :                  / * 表明数据仿问的限制和约束条件
      int GetX();            / * 定义公有的成员函数
      int GetY();            / * 定义公有的成员函数
}
```

可见封装性有利于用户集中精力开发系统,只要采用合适的模块并正确连接,而无需去了解模块实现的细节;另外在有了正确、高效的模块后,可以不断提高软件模块的可重用性和可移植性,大大减少了软件开发的周期,提高了软件的质量。

2.继承性(Inheritance)

与传统的程序设计方法相比,OOP 的继承性是得以广为流行的关键,它提供了通用的和公用的功能,允许程序员根据需要在继承原有类的基础上进行更具体类的定义。

(1)具体含义。若类 D 继承类 B,则称 B 为基类(Base Class),D 为派生类(Derived Class),并且 D 将根据用户所给的继承和访问控制机制而具备了基类 B 中的全部或部分语义(它包括数据和函数以及访问控制机制的继承),在 C++中表达为:

```
class D : public B                  / * 缺省为 private
{
   …                                / * 其他定义项
}
```

(2)继承的传递性。它的传递性体现在这种继承能够无限地继续下去,也就是可以从派生类 D 中继续派生 DD 类、DDD 类...,而且这些派生类能够逐次地、自动地继承其基类(父类)的全部或部分语义。正是由于这种多层继承(即传递性) 使数据和函数得以进行共享。

(3)继承的多重性。一个派生类可以继承多个基类,这就是继承的多重性,它使得派生类的实例(对象)具备更加灵活的、功能更完善的特性。

(4)继承和访问控制机制。在派生类继承其基类的过程中,继承和访问控制机制是指派生类以什么方式继承基类,就是直接影响到派生类的实例(对象) 如何访问和存取其基类中的数据或函数,在 C++中定义了继承控制方式,这种继承控制关系可以由下表得知(注意:在 C++中 union 既不能作为基类也不能作为派生类):

在定义一个依赖已存在类的新类时,首先要清楚基类和派生类之间的继承关系,即由说明符 private 和 public 授与的访问级别,存取权要仔细地在父～子～孙之间传递或抑制,不要把你的数据暴露给非家族者或非成员者,因此当某个类要被继承时,应仔细控制它们的访问级别如何被继承。

3.多态性(Polymorphism)

多态性主要强调可以在一个类等级中使用相同函数的多个版本,在运行时决定使用特定版本,也就是相同的操作随着不同类型的入口参数、存取方式以及返回值可在不同运行时间出现,在 C++中非常重要的和极具影响力的 OOP 概念,主要由以下几种情况:

基类存取	控制方式	派生类继承存取
public	public	public
private	public	not accessible
protected	public	protected
public	private	private
private	private	not accessible
protected	private	private

(1)重载运算符。同一个运算符可以再定义而对其他类型变量进行操作。在 C++再定义一个运算符可以

由关键字 operator 后跟运算符来完成。例如:"+"运算符可对任何标准数值型(int、float、double 等)值进行加运算,经过重载定义后,其操作意义可发生改变,比如可以合并两个串对象,返回适当长度和内容的一个串对象的结果。

(2)重载函数。当几个不同的函数(无论是成员函数。表面看来,操作函数名相同,但当函数被调用时,通过调用时实参类型和函数定义时的形参类型加以区别,这可由编译器区分开来,不会产生二义性。例如,求立方函数 Cube 可以用于以下两种类型 ing、float 的操作:

```
int Cube(int number);
float cube(float float-number);
```

(3)虚函数。虚函数是用来解决重载函数在程序执行时,而不是编译时根据具体数据类型业先定先前定义的那个类型的同名函数,这就称为迟后联编,也叫动态联编(Dynamic Binding),与之相对的是重载函数在编译时确定下来,称为先前(功静态)联编(Static Binding);另一方面,因为调用一个非虚函数比调用一个虚函数要快点,而且虚函数需要较多的内存和更多的存储访问周期,因而在不考虑扩充性而性能是主要时,使用普通的成员函数,否则使用虚函数,这种虚函数与迟后联编简化了系统开发,使得操作函数名的定义更为方便而又不会因为同名而导致混乱,使系统开发效率大大提高。

四、结束语

OOP 是新一代的具有全新概念的编程方法,它解决了在软件产业发展中的许多瓶颈问题,是未来开发软件的程序设计方法的主流越来越为人们所接受,希望有越来越多的软件开法者投入到 OOP 程序设计方法中去,真正了解 OOP 程序设计方法的优越性。下面举一个完整的 OOP 编程方法的例子,在本程序中将说明 OOP 编程方法中的一些基本特性和概念,还将说明如何在 C++中具体实现,其源程序清单如下:

```
#include<iostream.h>              //C++中标准的 I/O 流
class Point{                       //定义 Point 类
    int X;                         //定义坐标 X 和 Y,并默
    int Y;                         //认为私有的成员数据
pubilc:                            //定义构造函数和公有函数
    Point(int InitX,int IntY){X=InitX;Y=InitY;}
    int GetX( ){return X;}         //公有的成员函数
    int GetY( ){return Y;}
};                                 //类定义结束
int main( )                        //主程序
{
    int YourX,YourY;
    cout <<"Set X coordinate:";    //利用 I/O 流在屏幕提示
                                                      X 坐标
    cin >>YourX;                   //等待键盘输入
    cout <<"Set Y coordinate:";    //屏幕又提示输入 Y 坐标
    cin>> YourY;                   //等待键盘输入
    Point YourPoint(YourX,YourY);  //Point 类实例(对象)
                                               YourPoint
                                   //的初始化构造
    ccout<<"X is"<<YourPoint.GetX();  //调用成员函数
                                               GetX( )
    cout <<'\n';                   //回车换行
    cout <<"Y is" <<YourPoint.GetY(); //调用成员函数
                                               GetX( )
    cout <<'\n';                   //回车换行
    retrun 0;
}
```

参考文献:

[1]Paul Simon.OOP development and application. 1989

[2]古新生 面向对象的程序设计方法 计算机世界月刊。1992(10)

* *