面向对象程序设计(OOP)

C++(第7章：虚函数和多态性)

福州大学·软件学院·软件工程系
王灿辉(wangcanhui@fzu.edu.cn)

第7章：虚函数和多态性

概述

- 封装、继承和多态性是面向对象程序设计的3个基本要素。通过继承，派生类能够在增加新功能的同时，吸收现有类的数据和行为，从而提高软件的可重用性。而多态性使得程序能够以统一的方式来处理基类和派生类的对象行为，甚至是未来的派生类，从而提高系统的可扩展性和可维护性。

第7章：虚函数和多态性

概述

- 结合“抽象基类”和“多态”有如下优点：

  1、应用程序不必为每一个派生类编写功能调用，只需要对抽象基类进行处理即可。“以不变应万变”，可大大提高程序的可复用性。

  2、派生类的功能可以被基类指针引用，这叫向后兼容，可以提高程序的可扩充性和可维护性。以前写的程序可以被将来写的程序调用不足为奇，但将来写的程序可被以前写的程序调用那可了不起。

第7章：虚函数和多态性

概述：静态联编和动态联编

- 当程序中调用同名函数时，编译器会根据函数的参数类型、个数、顺序、返回值类型以确定执行哪一个同名函数的代码，这种把一个函数的调用与适当的函数实现代码联系在一起的过程称为联编(binding，绑定，束定)。根据联编的实现阶段的不同，可以将其分为**静态联编和动态联编**两种。

第7章：虚函数和多态性

概述：静态联编和动态联编

- 静态联编(早绑定)是在程序编译阶段确定一个函数的调用与函数实现代码间的对应关系，这种绑定关系确定后在程序的执行过程中始终保持不变。如：标准函数的调用等。
- 动态联编(晚绑定)是在程序运行过程中根据需要处理的对象类型来(动态)决定执行哪个类的成员函数(虚函数)。

第7章：虚函数和多态性

概述：多态性

- C++支持多态性的基础由继承和基类指针(或引用)组成。实现多态性的具体功能就是虚函数(virtual function)。
- 如果通过基类指针调用虚函数，C++就会根据指针所指向的对象类型(而不是指针本身的类型),在运行时确定实际执行的虚函数。通过基类引用调用虚函数也会达到相同的效果。

第7章：虚函数和多态性

概述：多态性

- 所谓多态性就是指同样的消息被类的不同对象接收时导致的完全不同的行为的一种现象。这里所说的消息即对类的成员函数的调用。
- C++支持两种类型的多态，一种是编译时的多态(静态多态，由静态联编实现)，另一种是运行时的多态(动态多态，由动态联编实现)。

第7章：虚函数和多态性

虚函数

- 声明虚函数的方法是在基类中的成员函数原型前加上关键字virtual：

```
class 类名 {
    //……
    virtual 类型 函数名(参数表);
    //……
}//函数定义前不能加virtual关键字！
```

第7章：虚函数和多态性

虚函数

- 虚函数父类和子类的声明必须完全一致！否则只是函数覆盖。

  基类：virtual void show(int x);

  派生类：

  virtual void show(int x) const;

- 析构函数一般要设为虚函数。

第7章：虚函数和多态性

虚函数

- 只有类的成员函数和析构函数可以声明为虚函数，而全局(普通)函数、类的构造函数以及静态成员函数不能声明为虚函数。
- 当一个类的成员函数被声明为虚函数后，就意味着该成员函数在派生类中可能有不同的实现，即该函数在派生类中可能需要**定义与其基类虚函数原型完全相同的函数(包括参数类型、个数、顺序、返回值类型)**。

第7章：虚函数和多态性

虚函数与多态性

- 虚函数是动态联编的基础。当用基类型的指针或引用的方法指向不同派生类的对象时，系统会在程序运行中根据所指向对象的不同自动选择适当的虚函数，从而实现了运行时的多态性。
- **虚函数与函数重载不同**。虚函数的实现原型必须完全相同，如果原型不同就只是普通的函数覆盖，也就失去其虚拟的本质。因此一般用覆盖(overriding)来描述虚函数的重定义过程。

第7章：虚函数和多态性

虚函数与多态性

- 可以采用标准的.或->运算符语法来正常调用虚函数，不过采用这种方式调用虚函数就会忽略它的多态属性。只有当通过基类指针(或引用)访问虚函数时，才能实现运行时的多态性！用基类指针又不想进行动态绑定时可以在虚函数前加类名::限定。

第7章：虚函数和多态性

虚函数与多态性

- 当在基类的构造函数和析构函数中调用虚函数时，不会调用派生类的虚函数。
- 虽然没有进行强制，但一般不要重写继承而来的非虚函数(否则已经定义为虚函数了)
- 虚函数有默认值时(静态绑定)，派生类不能重新定义不同的值。
- **实现多态性必要条件：必须是虚的并且正确覆盖，用基类指针(或引用)访问虚函数！**

第7章：虚函数和多态性

虚函数与多态性实例：多态类

```
struct Base {/*包含虚函数的类称为多态类
(Polymorphic class)，继承包含虚函数的多态
类的派生类自动成为多态类*/
  virtual void show(void) const{
    cout<<"Base\n";
  }//虚函数(virtual function)的定义
};
```

第7章：虚函数和多态性

虚函数与多态性实例：派生类

```
struct Derived1:Base {/*一旦将函数说明为虚函数，不管它通过多少层派生类，它都是虚函数。派生类重新定义虚函数时，不需要重复使用关键字virtual。这样做不会错误，为了提高程序可读性建议加上关键字virtual。*/
  virtual void show(void) const
    {cout<<"Derived1\n";}
};
```

第7章：虚函数和多态性

虚函数与多态性实例：使用基类引用

```
void showAll(const Base &br)
{/*普通(全局)函数，参数为基类引用！基类引用参数可以接受基类的对象和从该基类公有派生而来的任何其它类型的对象。该函数会根据引用指向的对象类型来确定调用函数的哪个版本。*/
  br.show();
}
```

第7章：虚函数和多态性

虚函数与多态性实例：正常调用

- 可以采用标准的.或->运算符语法来正常调用虚函数，不过采用这种方式调用虚函数就会忽略它的多态属性。只有当通过基类指针(或引用)访问虚函数时，才能实现运行时的多态性！

```
b0bj.show();   //正常调用虚函数
d0bj1.show();
d0bj2.show();
g0bj.show();
```

虚函数与多态性完整实例

第7章：虚函数和多态性

虚函数的简单应用

- 该示例程序创建一个基类Figure，这个类保存了多种二维对象的尺寸并且可以计算它们的面积。函数set_dim()对派生类来说是公共的，而函数show_area()被声明为虚函数。
- 虚函数的简单应用实例

第7章：虚函数和多态性

虚析构函数

- 构造函数不能声明为虚函数，而析构函数却可以声明为虚函数。例如：

```
virtual ~Base(){delete[] pb;}
void main() {
    Base *pb=new Derived(100,120);
    //.....
    delete pb;
}
```

- 需要将析构函数声明为虚函数的实例

第7章：虚函数和多态性

纯虚函数与抽象类

- 有时创建基类就只定义了与其所有派生类共享的一般形式，而由每个派生类来填充其详细信息，这时可以采用纯虚函数。纯虚函数的声明格式为：

  virtual 类型 函数名(参数表)=0;

- 纯虚函数是在基类中声明但没有定义的虚函数。

第7章：虚函数和多态性

纯虚函数与抽象类

- 声明了纯虚函数的基类(称为抽象类)仅用于继承，作为一个接口(interface，界面)，具体功能在其派生类中实现。
- 抽象类只能作为基类用来派生新类，而不能用来创建对象(区别于函数体为空的虚函数)。例如：现实世界中食品代表一个抽象的概念(能吃的东西)，不可能有一个食品对象。
- 但可以声明抽象基类的指针和引用，用于指向派生类的对象。

第7章：虚函数和多态性

纯虚函数与抽象类：实例

```
struct Base
{//抽象类，不能用于声明对象！
  virtual void show(void) const =0;
}; //纯虚函数
struct Derived1:Base {
  virtual void show(void) const
    {cout<<"Derived1\n";}
};
struct Derived3:Base {};
//派生类未覆盖纯虚函数，所以仍然是抽象类！
```

第7章：虚函数和多态性

纯虚函数与抽象类：实例

```
//Base   bObj;//抽象类，不能用于声明对象
Base     *bp; //可以用于声明指针
Derived1 dObj1,*dp1=&dObj1;
Derived2 dObj2,*dp2=&dObj2;
//Derived3 dObj3; //也是抽象类!
```


纯虚函数与抽象类完整实例

第7章：虚函数和多态性

纯虚函数与抽象类的简单应用

- 该示例程序对前述的虚函数的简单应用实例进行修改：将函数show_area()修改为纯虚函数，删除Figure对象(抽象类不能声明对象)的定义及相应的处理语句。
- 纯虚函数与抽象类的简单应用实例

第7章：虚函数和多态性

使用虚函数：总结

- 多态性是面向对象编程的基本功能，它允许一般化的类指定类的所有派生类共有的那些函数，同时允许派生类定义所有或其中一部分函数的具体实现。即基类规定派生类所共有的接口，让派生类定义用于实现该接口的实际方法(接口和实现的分离)，所以多态性经常也被描述为：**“一个接口多种方法”**。

第7章：虚函数和多态性

早绑定或晚绑定

- 早绑定的示例包括：标准函数调用、函数重载、运算符重载等，其最大优点是高的时空效率，缺点是不够灵活。
- 晚绑定使用虚函数和派生类实现，优点是灵活，缺点是运行慢、占用更多内存。
- 早绑定或晚绑定的使用取决于程序的设计，一般不考虑效率。

本章内容讲授到此结束！

福州大学·软件学院·软件工程系
王灿辉(wangcanhui@fzu.edu.cn)