Chap05 多态性

程勇

北京化工大学

信息科学与技术学院计算机系

Sept. 2020

课程提纲

- Chap01 绪论
- Chap02 面向对象程序设计方法
- Chap03 重载与类型转换
- Chap04 继承与派生
- Chap05 多态性
- Chap06 输入输出流
- Chap07 容错与异常处理
- Chap08 模版

本章提纲

- 多态性概念
- 虚函数
  - 静态关联和动态联编
  - 虚成员函数
  - 虚析构函数
  - 虚函数默认参数
  - 使用引用调用虚函数
- 纯虚函数和抽象类
  - 纯虚函数
  - 抽象类

基本概念

- 多态性
  - 多态性是面向对象的关键特性之一，利用多态性可以调用同一个函数名的函数，实现完全不同的功能；
  - 为了实现类的多态性，必须将该类定义为至少包含一个虚函数的派生类；
- 静态解析
  - 在执行程序之前就已经明确了函数调用，称为静态解析或静态绑定；
  - 通过静态解析的基类指针来调用函数，都会调用基类的函数；
  - 使用对象来调用虚函数总是静态解析；
- 动态解析
  - 在程序执行期间确定函数调用关系，称为动态解析或动态绑定；
  - 为实现动态绑定，需要将函数声明为基类中的虚函数，然后该类的派生类中，该函数都是动态绑定的；

虚函数

- 虚函数
  - 虚函数是一个类的成员函数，在声明时要使用关键字virtual，格式如下：

```
virtual 返回类型 函数名(参数列表){
    …
};
```

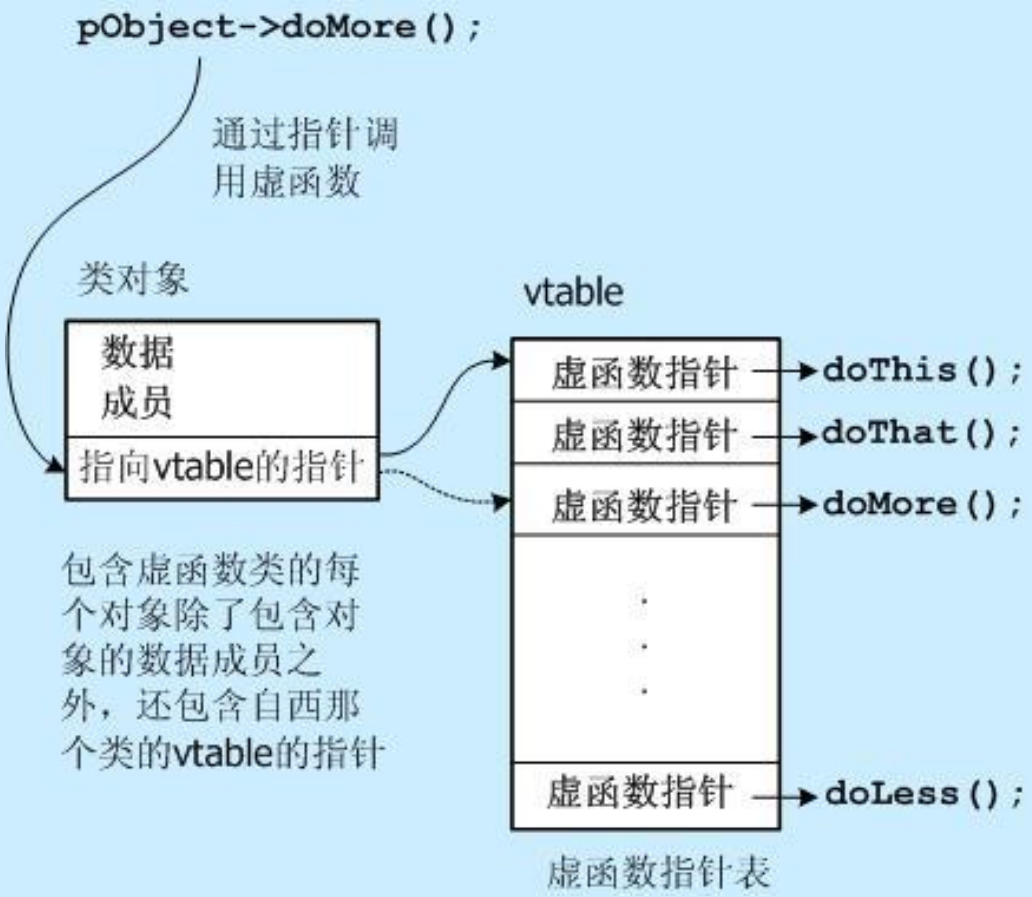
  - 关键字virtual指明该成员函数为虚函数，不要在函数定义时添加关键字virtual，否则将导致错误；
  - 当一个类的某个成员函数被定义为虚函数，则由该类派生出来的所有派生类中，该函数始终保持虚函数的特性；
  - 当在派生类中需要定义虚函数时，不必加关键字virtual，但重新定义时不仅要同名，而且它的参数表和返回类型全部与基类中的虚函数一样，否则将导致错误；
  - 如果把基类函数声明为const，则必须把派生类函数也声明为const；

虚函数(续)

- 虚函数默认参数
  - 如果虚函数在基类声明时带有默认参数值，则通过基类指针调用该函数时，就总是从函数的基类版本中接受默认的参数值，派生类版本中的默认参数值根本不起作用，因为默认值都是在编译期间处理的；
- 使用引用调用虚函数
  - 使用引用来调用虚函数与通过指针来调用具有同样的效果；
- 注意事项
  - 内联函数和模版函数都不能用作虚函数；
  - 静态成员函数，是所有同一类对象共有，不受限于某个对象，不能作为虚函数；
  - 析构函数可定义为虚函数；
  - 虚函数在派生类中声明时，所用的访问指定符可以不同于其在基类中声明时的访问指定符；
  - 只要访问运算符允许，可以使用作用域运算符调用成员函数的基类版本；

多态性函数调用原理

- 多态性函数调用过程
  - 首先，使用指向vtable的对象指针查找类的vtable的开头；
  - 在类的vtable中，查找被调用函数所对应的数据项，通常采用偏移量来实现；
  - 通过vtable中的函数指针来间接的调用函数；
- 多态性的主要问题
  - 消耗更多的内存；
  - 需要额外的系统开销；



纯虚函数

- 纯虚函数
  - 纯虚函数(pure virtual function)指被标明为不具体实现的虚函数；
  - 把函数声明为纯虚函数与声明为虚函数的语法相同，但要在声明中加上=0，纯虚函数通常没有实现代码；
  - 一般格式

```
virtual 返回类型 函数名(参数列表) = 0;
virtual void draw() const = 0;
```

  - 含有纯虚函数的基类不能用来构造对象，因为纯虚函数没有具体实现；
- 抽象类
  - 含有纯虚函数的类成为抽象类；
  - 抽象类存在的主要用于定义派生于它的其他类，另外其指针或引用可以用作参数或返回类型；
  - 不能创建抽象类的对象；
  - 在派生类构造函数中可以调用抽象类的构造函数；

运行期类型识别与强制转换

- 运行期类型识别
  - 使用typeid()运算符可以在运行期间确定类型，为使用该运算符需要在源文件中包含头文件<typeinfo>；
  - typeinfo类实现了==运算符，因此可以比较两个info_info对象；
    typeid(*pVessel) == typeid(Carton)
  - 使用typeid()运算符确定动态类型时，通常需要解除基类指针的引用；
- 动态强制转换
  - 在运行期要指定动态强制转换，需要使用dynamic_cast<>运算符；
  - 该运算符只能应用于多态类类型的指针和引用，即至少包括一个虚函数的类类型；
  - 如果动态强制转换失败，其结果可能为空指针；
  - 动态转换指针有两种类型：downcast和crosscast，前者沿类层次结构向下进行强制转换，后者是跨类层次结构进行强制转换；
  - 不能使用dynamic_cast<>来去除指针的const性质，必须使用const_cast<>运算符；

类成员的指针

- 类成员指针
  - 类成员指针指向类的数据成员和函数成员；
  - 类数据成员成员指针仅在与类对象组合使用时，才指向内存中的某个位置；
  - 声明类的成员函数指针涉及类类型、函数的参数列表和返回类型等等；
- 数据成员指针使用

```
typedef double Box::*pBoxMember;
pBoxMember pData=&Box::length;
cout << “Data member value is “ << pBox->*pData << endl;
```

- 成员函数指针使用

```
typedef return_type(class_type::*ptr_name)(param_type_list)<const>;
ptr_name ptr_fun = &class_type::fun_member;
cout << “Width member of myBox is “ << (myBox.*ptr_fun)() << endl;
```

本章小结

- 多态性概念
- 虚函数
- 纯虚函数和抽象类

- 课程网站
  - http://www.jiaowu.buct.edu.cn