C++语言程序设计
第六章 C++的多态性

王焦乐

http://faculty.hitsz.edu.cn/jlwang

哈尔滨工业大学（深圳）

机电工程与自动化学院

本章主要内容

- 什么是多态性
- 一个典型的例子
- 利用虚函数实现动态多态性
- 纯虚函数与抽象类

0、什么是多态性

- 多态性（polymorphism）：向不同的对象发送同一个消息，不同的对象在接收时会产生不同的行为。
- C++中的表现形式：
  - 具有不同功能的函数可以用同一个函数名，实现调用不用内容的函数。
- 静态多态性
  - 函数重载，编译时决定
- 动态多态性
  - 运行时决定，通过虚函数实现

1、一个典型的例子

- 例6.1 先建立一个点类(point)，有数据成员x，y(坐标)。以它为基类派生一个圆类，增加数据成员r(半径)，再以圆为直接基类派生出一个圆柱体类，再增加数据成员h(高)要求重载运算符<<和>>使之能输出以上的类对象。

(1)声明基类point类

(2)声明派生类circle

在(1)的基础上，再写出声明派生类circle的部分：

(3)声明circle 的派生类cylinder

以circle为基础，从circle 类派生出cylinder类

2、利用虚函数实现动态多态性

- 虚函数的作用
- 用同一个调用形式，即能调用派生类的函数也能调用基类的同名函数
- 赋值兼容性规则是多态性的基础。同一个函数可以统一处理具有公有派生关系的基类的对象和派生类对象
- 虚函数的工作原理是在派生类中定义与基类函数同名的函数，通过基类指针或引用来访问基类或派生类中的同名函数
- 基类中通过virtual修饰
- 派生类中重新定义函数体

2、利用虚函数实现动态多态性

□ 虚函数的使用方法

(1)在基类用virtual声明成员函数为虚函数。在派生类中重新定义同名函数，让它具有新的功能。

(2)在派生类中重新定义此函数时，要求函数名、函数类型、参数个数和类型与基类的虚函数相同，根据需要重新定义函数体。C++规定，当一个成员函数被声明为虚函数后，其派生类中的同名函数自动成为虚函数。

(3)定义一个指向基类对象的指针变量，并让它获得同一类族中某个对象的地址。

(4)用该指针变量调用虚函数，调用的就是该对象所属类的虚函数。

2、利用虚函数实现动态多态性

- 静态关联与动态关联
- 确定调用的具体对象的过程称为关联（binding）：如把一个函数名与一个类对象建立关联。又叫联编、编联、束定、绑定。
- 函数重载、通过对象名调用的虚函数，在编译时即可确定其调用的虚函数属于哪一个类，其过程称为静态关联（static binding）、早期关联（early binding）。
- 在运行阶段，指针可以先后指向不同的类对象，从而调用同一类族中不同类的虚函数，这个过程称为动态关联（dynamic binding）、滞后关联（late binding）。

2、利用虚函数实现动态多态性

- 在什么情况下声明虚函数?
- 使用虚函数需要注意:

（1）只能将类的成员函数声明为虚函数。

（2）一个成员函数被声明为虚函数后，在同一类族不能再定义一个与该虚函数相同的非虚函数。

- 声明虚函数时，主要考虑下面几点:

(1)首先看成员函数的类是否会作为基类。然后看在派生类里看它是否会被改变功能，如要改变，且通过基类指针调用，一般应该将它声明为虚函数。否则不要将它声明为虚函数。

(2)有时在定义虚函数时，函数体是空的。具体功能留给派生类去添加。

2、利用虚函数实现动态多态性

□ 虚析构函数

1. 通过指向基类的指针撤销对象时，只执行基类析构函数
2. 当基类的析构函数为虚函数，无论指针指向的是同一类族中的那一个对象，系统都会动态关联，调用相应的析构函数进行清理。

```
class Point
{public:
  Point() {cout<<"constructor Point"<<endl;}
  // ~Point() { cout << "destructor Point" << endl; }
  virtual ~Point() {cout <<"destructor Point"<<endl;}
private:
  int x,y;};
```

```
class Circle : public Point
{public:
  Circle() {radius=0;cout<<"constructor Circle"<<endl;}
  ~Circle() { cout << "destructor Circle" << endl; }
private:
  int radius;};
```

```
int main()
{
  Point *p = new Circle;
  delete p;
  return 0;
}
```

3、纯虚函数与抽象类

没有函数体的纯虚函数

- 只声明，不定义的虚函数称为纯虚函数
- 一般格式

  virtual 函数类型 函数名(参数表)=0;

- 纯虚函数的作用

  在许多情况下，基类中不能为虚函数给出一个有意义的定义，而将它说明为纯虚函数，其作用是：**为派生类提供一个一致的接口(界面)。它的定义留给派生类来做，派生类根据需要来定义各自的实现。**

3、纯虚函数与抽象类

没有函数体的纯虚函数

注意：

- 一个类可以说明一个或多个纯虚函数。
- 纯虚函数与函数体为空的虚函数的**区别**
  - 纯虚函数
    - 根本没有函数体
    - **所在的抽象类，不能直接进行实例化**
  - 空的虚函数
    - 函数体为空
    - 所在的类可以实例化
  - 共同的特点
    - 可以派生出新的类

3、纯虚函数与抽象类

- 不能用来定义对象的类——抽象类
  - **带有纯虚函数的类是抽象类**
  - 抽象类的主要作用
    - 通过它为一个类族建立一个公共的接口，使它们能够更有效地发挥多态特性
  - 抽象类刻画了一组子类的**公共操作接口的通用语义**，这些接口的语义也传给子类。一般而言，抽象类只描述这组子类共同操作接口，而完整的实现留给子类。

口抽象类的说明

- 抽象类是一个特殊的类，是为了抽象和设计的目的而建立的，它处于继承层次的结构的较上层，即**只能用作其他类的基类，抽象类是不能定义对象的。**
- 从一个**抽象类派生的类**必须提供纯虚函数的实现代码或在该派生类中仍将它说明为纯虚函数，否则编译错。
- 抽象类**不能用作参数类型、函数返回类型或显式转换的类型**，但可以说明指向抽象类的指针和引用，此**指针可以指向它的派生类**，实现多态性
- 构造函数不能是虚函数，析构函数可以是虚函数。

3、纯虚函数与抽象类

□ 应用实例

```
class B0 //抽象基类B0声明
{
public:                          //外部接口
  virtual void display() = 0; //纯虚函数成员
};

class B1 : public B0 //公有派生
{
public:
  void display(){cout<<"B1::display()"<<endl;}
};

class D1 : public B1 //公有派生
{
public:
  void display(){cout<<"D1::display()"<<endl;}
};
```

```
void fun(B0 *ptr) //普通函数
{
  ptr->display();
}

void main() //主函数
{
  B0 *p; //声明抽象基类指针
  B1 b1; //声明派生类对象
  D1 d1; //声明派生类对象
  p = &b1;
  fun(p); //调用派生类B1函数成员
  p = &d1;
  fun(p); //调用派生类D1函数成员
}
```

3、纯虚函数与抽象类

- 应用实例
- 改写例6.1中的“点-圆-圆柱体”类的层次结构，使用虚函数和抽象基类。顶层是抽象基类Shape。Point、Circle、Cylinder都是Shape类的直接派生类和间接派生类。

(1) 声明抽象基类Shape

```
class Shape
{
public:
  virtual float area() const { return 0.0; }   //虚函数
  virtual float volume() const { return 0.0; } //虚函数
  virtual void shapeName() const = 0;          //纯虚函数
};
```

3、纯虚函数与抽象类

- 改写例6.1中的“点-圆-圆柱体”类的层次结构，使用虚函数和抽象基类。顶层是抽象基类Shape。Point、Circle、Cylinder都是Shape类的直接派生类和间接派生类。

(2) 声明Point类

```
class Point : public Shape // Point是Shape的公用派生类
{
protected:
  float x, y;

public:
  Point(float = 0, float = 0);
  void setPoint(float, float);
  float getX() const { return x; }
  float getY() const { return y; }
  virtual void shapeName() const { cout << "Point:"; }  //    对纯虚函数进行定义
  friend ostream &operator<<(ostream &, const Point &);
};
```

3、纯虚函数与抽象类

- 改写例6.1中的“点-圆-圆柱体”类的层次结构，使用虚函数和抽象基类。顶层是抽象基类Shape。Point、Circle、Cylinder都是Shape类的直接派生类和间接派生类。

(3) 声明Circle类

```
class Circle : public Point //    声明Circle类
{
protected:
  float radius;

public:
  Circle(float x = 0, float y = 0, float r = 0);
  void setRadius(float);
  float getRadius() const;
  virtual float area() const;
  virtual void shapeName() const { cout << "Circle:"; } //   对纯虚函数进行再定义
  friend ostream &operator<<(ostream &, const Circle &);
};
```

3、纯虚函数与抽象类

- 改写例6.1中的“点-圆-圆柱体”类的层次结构，使用虚函数和抽象基类。顶层是抽象基类Shape。Point、Circle、Cylinder都是Shape类的直接派生类和间接派生类。

(3) 声明Cylinder类

```
class Cylinder : public Circle //    声明Cylinder类
{
protected:
  float height;
public:
  Cylinder(float x = 0, float y = 0, float r = 0, float h = 0);
  void setHeight(float);
  float getHeight() const;
  virtual float area() const;
  virtual float volume() const;
  virtual void shapeName() const { cout << "Cylinder:"; }   // 对纯虚函数进行再定义
  friend ostream &operator<<(ostream &, const Cylinder &);
};
```

3、纯虚函数与抽象类

- 抽象基类包含一个或以上纯虚函数
- 抽象基类不能也不必要定义对象
- 抽象基类与普通基类不同，不是现实存在的对象的抽象。
- 类的层次结构中，顶层或最上几层可以是抽象基类。
- 抽象基类体现了本类族的共性，把各类中共有的成员函数集中在抽象基类中声明。
- 抽象基类是本类族的公共接口，程序员即使不了解细节，也可以调用其中的函数。
- 区别静态关联和动态关联。如果是用对象名调用，属于静态；通过基类指针，属于动态。
- 基类声明虚函数后，同名、同类型、同参数的函数均为虚函数。
- 使用虚函数提高了程序的可扩充性。