C++ 实验：继承和多态

郭 XX

xxxxxxxxx，物联网 xxxx，XX 大学

2018 年 6 月 18 日

摘要

这是使用 LaTeX 写的实验 C++ 报告：类的继承以及多态。

关键词：C++，类继承，多态，虚函数

目录

1 C++ 继承

1.1 简介

面向对象程序设计中最重要的一个概念是继承。继承允许我们依据另一个类来定义一个类，这使得创建和维护一个应用程序变得更容易。这样做，也达到了重用代码功能和提高执行时间的效果。

当创建一个类时，您不需要重新编写新的数据成员和成员函数，只需指定新建的类继承了一个已有的类的成员即可。这个已有的类称为**基类**，新建的类称为**派生类**。

继承代表了 **is a** 关系。例如，哺乳动物是动物，狗是哺乳动物，因此，狗是动物，等等。

1.2 继承类型

这里也是引用在线【菜鸟教程】的说明：

当一个类派生自基类，该基类可以被继承为 **public**、**protected** 或 **private** 几种类型。继承类型是通过上面讲解的访问修饰符 access-specifier 来指定的。

我们几乎不使用 **protected** 或 **private** 继承，通常使用 **public** 继承。当使用不同类型的继承时，遵循以下几个规则：

公有继承（**public**）：当一个类派生自公有基类时，基类的公有成员也是派生类的公有成员，基类的保护成员也是派生类的保护成员，基类的私有成员不能直接被派生类访问，但是可以通过调用基类的公有和保护成员来访问。

保护继承（**protected**）：当一个类派生自保护基类时，基类的公有和保护成员将成为派生类的保护成员。

私有继承（**private**）：当一个类派生自私有基类时，基类的公有和保护成员将成为派生类的私有成员。

1.3 访问控制和继承

派生类可以访问基类中所有的非私有成员。因此基类成员如果不想被派生类的成员函数访问，则应在基类中声明为 private。

我们可以根据访问权限总结出不同的访问类型，如表 1 所示：

访问	public	protected	private
同一个类	Yes	Yes	Yes
派生类	Yes	Yes	No
外部的类	Yes	No	No

表 1: 上图演示了不同的访问类型。

派生类无法继承的情况：

1. 基类的构造函数、析构函数和拷贝构造函数。
2. 基类的重载运算符。
3. 基类的友元函数。

1.4 基类和派生类构建过程

在 C++ Primer Plus 6th 中讲解如下：

派生类构造函数：

1. 首先创建基类对象
2. 派生类构造函数应通过成员初始化列表传递给基类构造函数
3. 派生类构造函数应初始化派生类新增数据成员

2 C++ 多态

2.1 什么是多态

多态按字面的意思就是多种形态。当类之间存在层次结构，并且类之间是通过继承关联时，就会用到多态。也可解释为派生类和基类的行为是不同的，使用同一个方法行为要根据上下文确定。

2.2 如何实现多态公有继承

有两种方法可以实现多态公有继承：

1. 在派生类重新定义基类
2. 使用虚方法

一般我们使用第二种，虚方法，又叫虚函数。

2.2.1 纯虚函数以及派生类

纯虚函数：纯虚函数或纯虚方法是一个虚拟函数，需要由不是抽象的派生类来实现。

派生类：在编程语言中，抽象类型是一种不能直接实例化的主导类型系统中的类型; 一个不抽象的类型可以被实例化被称为具体类型。

2.2.2 如何通过纯虚函数设计派生类

设计抽象类（通常称为 ABC）的目的，是为了给其他类提供一个可以继承的适当的基类。抽象类不能被用于实例化对象，它只能作为接口使用。如果试图实例化一个抽象类的对象，会导致编译错误。因此，如果一个 ABC 的子类需要被实例化，则必须实现每个虚函数，这也意味着 C++ 支持使用 ABC 声明接口。如果没有在派生类中重载纯虚函数，就尝试实例化该类的对象，会导致编译错误。可用于实例化对象的类被称为具体类。如果类中至少有一个函数被声明为纯虚函数，则这个类就是抽象类。纯虚函数是通过在声明中使用”= 0” 来指定的。

3 实验目的

3.1 简介

通过学习简单的类定义和类继承，了解 C++ 的多态，抽象类，虚函数；派生类。实现数据和程序的方便可视的处理和计算。

此外我是使用了 C++11 标准的成员初始化器列表 `member initializer` ，以及基于范围的 `for` 循环 for range 用来熟悉新版 C++ 特性，学习更多，跟随潮流。

具体请看代码赏析。

这是我们最后一个 C++ 实验，也是我用 LaTeX 写的第二篇实验报告，之所以用 LaTeX 写实验报告，是因为我想认真完成，也想试试新的东西。我觉得自己喜欢的事业，就应该应该的认真完成，虽然十分花费时间，但是我觉得很值得，年轻就应该多闯闯。

最后谢谢老师的帮助和教导，老师您幸苦了。

4 附录

4.1 源码

4.1.1 main.cpp

```cpp
//Sources:main.cpp
#include <iostream>
#include "sportCar.h"
#include "Truck.h"
#include "Car.h"

using namespace std;

int main()
{
    //Car mycar();
    //The class car is abstract,so it can not be instantiated.

    //sportCar e1();
    //e1.print();
    //this is error,i do not know why.

    sportCar car1("White");
    car1.print();
    std::cout<<std::endl;
    sportCar car2("Red",4,4);
    car2.print();
    std::cout<<std::endl;
    //Truck e2();
    //e2.print();
    //this also is error,i do not know why.

    Truck car3("Gray");
    car3.print();
    std::cout<<std::endl;
```

```
31
32      Truck car4("Black",6,"Gasoline");
33      car4.print();
34      std::cout<<std::endl;
35
36      Car *ptr[4];
37      ptr[0]=&car1;
38      ptr[1]=&car2;
39      ptr[2]=&car3;
40      ptr[3]=&car4;
41
42      //C++11 for range
43      for( const auto &y : ptr )
44      {   // Type inference by const reference.
45          // Observes in-place. Preferred when no modify is needed.
46          // 参考: https://docs.microsoft.com/zh-cn/cpp/cpp/range-based-for-statement-cpp
47          y->print();
48      }
49      /*
50          以上代码等价:
51          for (int i = 0; i < 5; i++)
52              ptr[i]->print();
53      */
54
55      cout << endl;
56
57      return 0;
58  }
59  #endif
```

4.1.2 car.cpp

```
1  //Sources:Car.cpp
2  #include "Car.h"
3
4  void Car::setColor(std::string color)
5  {
6      Color=color;
7  }
8
```

```
9  void Car::setTire(int tire)
10 {
11     Tire=tire;
12 }
13
14 std::string Car::getColor() const
15 {
16     return Color;
17 }
18
19 int Car::getTire() const
20 {
21     return Tire;
22 }
23
24 void Car::print()
25 {
26     std::cout<<"The car's color is "<<Color<<" ,It has "<<Tire<<"
           tire(s)."<<std::endl;
27 }
```

4.1.3 car.h

```
1  //Headers:Car.h
2  #ifndef CAR_H
3  #define CAR_H
4
5  #include <iostream>
6  #include <string>
7
8  /*
9      设计抽象类（通常称为 ABC）的目的，是为了给其他类提供一个可以继承的适当的基类。抽象类不
       能被用于实例化对象，它只能作为接口使用。如果试图实例化一个抽象类的对象，会导致编译错误。
10     因此，如果一个 ABC 的子类需要被实例化，则必须实现每个虚函数，这也意味着 C++ 支持使用
       ABC 声明接口。如果没有在派生类中重载纯虚函数，就尝试实例化该类的对象，会导致编译错误。
11     可用于实例化对象的类被称为具体类。
12     如果类中至少有一个函数被声明为纯虚函数，则这个类就是抽象类。纯虚函数是通过在声明中使用
       "= 0" 来指定的
13     摘自:http://www.runoob.com/cplusplus/cpp-interfaces.html
14 */
```

```
15
16
17 /*
18     从一个类派生出一个类，原始类成为基类，继承类称为派生类
19     多态：派生类和基类的行为是不同的，同一个方法行为根据上下文，有两种方法可以实现多态公有
       继承：
20     1. 在派生类重新定义基类
21     2. 使用虚方法
22     - C++ Primer Plus 6th
23 */
24
25 class Car
26 {
27     //这是个抽象类
28 public:
29     Car(std::string color="White",int tire=4) : Color(color),Tire(tire){};
           //constuctor 构造函数
30     /*
31        member initializer 成员初始化器列表
32        等价于:
33        Car(std::string color="White",int tire=4)
34        {
35             Color=color;
36             Tire=tire;
37        }
38        - https://www.geeksforgeeks.org/when-do-we-use-initializer-list-in-c/
39     */
40     ~Car(){};//destructors
41     void setColor(std::string);
42     std::string getColor() const;
43     void setTire(int);
44     int getTire() const;
45     virtual void print()=0;//纯虚函数
46     /*
47         A pure virtual function or pure virtual method is a virtual function that is
       required to be implemented by a derived class that is not abstract" - Wikipedia
48          “纯虚函数或纯虚方法是一个虚拟函数，需要由不是抽象的派生类来实现“ - 维基百科
```

```
49          In programming languages, an abstract type is a type in a nominative type
        system that cannot be instantiated directly; a type that is not abstract - which
        can be instantiated - is called a concrete type. Every instance of an abstract
        type is an instance of some concrete subtype. Abstract types are also known as
        existential types.
50          “在编程语言中，抽象类型是一种不能直接实例化的主导类型系统中的类型；一个不抽象的类
        型可以被实例化被称为具体类型。” - 维基百科
51          C++ Virtual/Pure Virtual Explained:
52          https://stackoverflow.com/questions/1306778/c-virtual-pure-virtual-explained
53          Virtual function:
54          https://en.wikipedia.org/wiki/Virtual_function
55      */
56  private:
57      std::string Color;
58      int Tire;
59  };//do not forget ;
60
61  #endif
```

4.1.4 sportcar.cpp

```
1   //Sources:sportCar.cpp
2   #include "sportCar.h"
3
4   void sportCar::setDoor(int door)
5   {
6       Door=door;
7   }
8
9   int sportCar::getDoor() const
10  {
11      return Door;
12  }
13
14  void sportCar::print()
15  {
16      //std::cout<<"The sportCar's color is "<<Color<<" ,It has "<<Tire<<" tire(s) and
            "<<Door<<" Door(s)."<<std::endl;
17      //This is error,the complier will tell it is private,but we can use like this
18      int tire=Car::getTire();
```

```
19      std::string color=Car::getColor();
20      std::cout<<"The sportCar's color is "<<color<<" ,It has "<<tire<<" tire(s) and
            "<<Door<<" door(s)."<<std::endl;
21      //or like this:
22      Car::print();
23      std::cout<<"It also has "<<Door<<" door(s)."<<std::endl;
24  }
```

4.1.5 sportcar.h

```
1   //Headers:sportCar.h
2   #ifndef SPORTCAR_H
3   #define SPORTCAR_H
4
5   #include "Car.h"
6
7   class sportCar : public Car
8   {
9   //继承 Car 类
10  public:
11      sportCar(std::string color="Black", int tire=4, int door=2):
            Car(color,tire),Door(door) {};
12      //将参数从派生类构造函数传递给基类构造函数
13      /*
14          派生类构造函数
15          首先创建基类对象
16          派生类构造函数应通过成员初始化列表传递给基类构造函数
17          派生类构造函数应初始化派生类新增数据成员
18          - C++ Primer Plus 6th
19      */
20      ~sportCar(){};
21      void setDoor(int);
22      int getDoor() const;
23      void virtual print();
24  private:
25      int Door;
26  };
27
28
29  #endif
```

4.1.6 Truck.cpp

```cpp
//Sources:Truck.cpp
#include "Truck.h"

Truck::Truck(std::string color,int tire,std::string engine)
    : Car(color,tire)
{
    setEngine(engine);
    // Engine=engine;
}
Truck::~Truck()
{

}
void Truck::setEngine(std::string engine)
{
    Engine=engine;
}
std::string Truck::getEngine()
{
    return Engine;
}
void Truck::print()
{
    int tire=Car::getTire();
    std::string color=Car::getColor();
    std::cout<<"The truck's color is "<<color<<" ,It has "<<tire<<" tire(s) and
        "<<Engine<<" engine."<<std::endl;
    //or like this:
    Car::print();
    std::cout<<"It also has "<<Engine<<" engine."<<std::endl;
}
```

4.1.7 Truck.h

```cpp
//Headers:Truck.h
#ifndef TRUCK_H
```

```
#define TRUCK_H

#include "Car.h"

class Truck : public Car
{
public:
    Truck(std::string="",int=4,std::string="Diesel");
    ~Truck();
    void setEngine(std::string);
    std::string getEngine();
    void virtual print();
private:
    std::string Engine;
};



#endif
```

4.2 UML 图

请依据下列 UML 编写基类 Car 以及卡車类(Truck)、跑車类(sportCar)两个派生类，其中的成员包括一个虚函數(virtual)

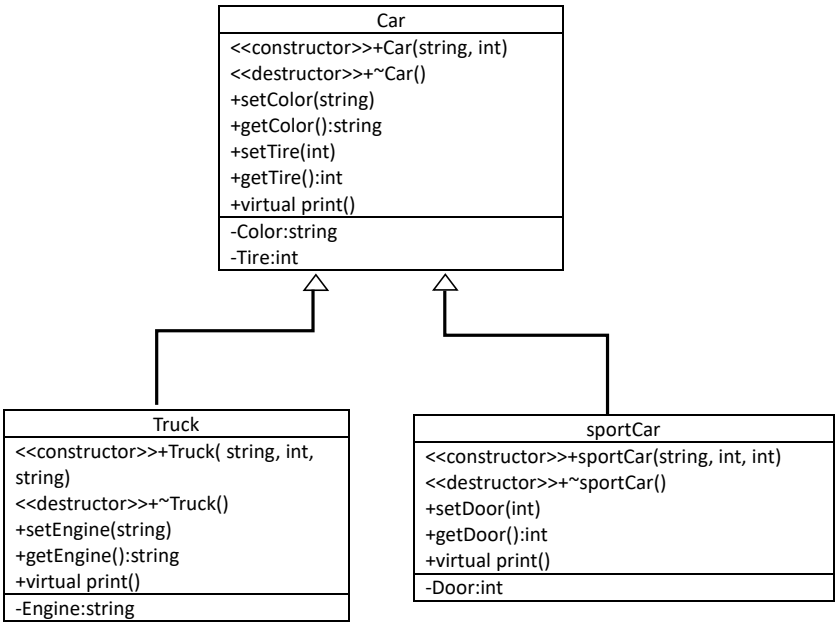

图 1: *UML* 图

参考文献

[1] *C++ Primer Plus*（第六版）中文版. 人民邮电出版社, 2012.

[2] cppreference. 基于范围的 for 循环. http://zh.cppreference.com/w/cpp/language/range-for.

[3] 刘海洋. *LaTeX* 入门. 电子工业出版社, 2013.

[4] 维基百科. 虚函数. https://zh.wikipedia.org/wiki/%E8%99%9A%E5%87%BD%E6%95%B0_(%E7%A8%8B%E5%BA%8F%E8%AF%AD%E8%A8%80).

[5] 菜鸟教程. C++ 接口（抽象类）. http://www.runoob.com/cplusplus/cpp-interfaces.html.

[6] 菜鸟教程. C++ 继承. http://www.runoob.com/cplusplus/cpp-inheritance.html.