《面向对象程序设计》朋辈课程 · 第五辑

类与对象

信息学院 赵家宇

PART 01

初识：类

进入类的世界

- 在之前C++的学习中，我们已经基本掌握了面向过程的编程和使用标准库类型；
- 而C++功能强大最为显著的表现在于提供了类编程机制，允许用户自己定义自己的类类型，这需要回顾之前的一些内容（内联、重载等）。
- 掌握如何定义类只是基础；更重要的是如何从实际问题出发，将问题域用类来表现，也即通过定义合适类型来对应所要解决的问题中的各种概念。

从结构体到类

- 结构体：将不同类型的或是相关联的数据元素组合在一起成为一个整体。
- 例如以下结构体类型：

1、学生结构类型

```
struct Student {
  int id;
  char name[10];  //string name;
  float score;
  char sex;
};
```

2、日期类型

```
struct DATE {
  int year;
  int month;
  int day;
};
```

从结构体到类

- 如果需要使用结构体，还需要我们进行许多类型的操作。

（1）设计结构体；

```
struct DATE {
  int year;
  int month;
  int day;
};
```

（2）声明与修改结构体变量；

```
struct DATE d1,d2,d3;
d1.year=“2024”;
```

在使用上：视为一个变量，main函数中直接访问成员；
存在风险：重要的属性值可以被外界任何人随意访问和修改。

（3）结构体的应用；

```
bool IsLeapyear(struct DATE x)
{
    …
};
```

从结构体到类

- C语言中的结构体一般只有数据成员，无成员函数。结构体是一个没有封装的数据类型，在缺省情况下，结构体中的数据成员都是公有的，因此无法对外界隐藏自己的重要信息和私密信息。同时，外界可随意修改结构体变量中的数据，这样对数据的操作是很不安全的，不能通过结构体对数据进行保护和控制；
- 为了保护数据不被随意修改和访问，C++语言中引入类类型，其包括数据成员和函数成员；在类的声明中，成员默认均为私有的，这样保证类这一封装的相对安全。

类类型的声明

- 类的声明与结构体声明类似，仅仅是定义了类型（不占空间）；
- private和public称为成员访问限定符，其顺序没有要求，可以分别出现多次，按照归类处理；缺省下默认为private。

```
class 类名 {
    private:
        私有的数据和成员函数;
    public:
        公用的数据和成员函数;
};
```

类：前向声明

- 类应该先声明后使用；如果需要在某个类声明之前引用，则应进行前向引用声明。
  - 先暂时声明一个类名但没有具体定义类中的各个成员；
  - 声明一个数据成员类型时，如果未见到相应的类型定义或相应的类型未定义完，则该数据成员的类型一般是类型的指针或引用。

```
class B;  //前向引用声明
class A{
   public:
      void f(B *b);
};
class B{
   public:
       void g(A a);
   };//类的前向声明一般用来编写互相依赖的类
```

```
class A; //前向引用声明
class B{
  A a;  //Error
  B b;  //Error
  A *p;   //OK
  B *q;    //OK
  A &aa;   //OK
  B &bb;   //OK
};
```

类：数据对象

- 类的数据成员说明对象的内部本质特征；
- 在类定义中，需要对数据成员的名字和类型进行说明；在类中，描述数据成员一般不能给他们赋初值。

```
class A {
    int x; //普通成员,不能赋值
    const double y;//const成员，不能赋值
    …
};//类似结构体的数据成员的描述一样
```

类：成员函数

- 类的数据成员说明对象的特征，而成员函数决定对象的操作行为；成员函数是对封装的数据进行操作的唯一途径。
- 成员函数本质上是一类特殊的函数，用法和作用和一般函数基本上是一致的，也具有返回值和函数类型；但其与一般函数的区别是：
  - 它属于类的成员，出现在类体中，并且可以指定成员访问限定符(private,public等)；
  - 在使用时，要注意调用它的权限以及它的作用域；

类：成员函数

- 类的成员函数有两种定义位置，分别为内联定义（类内定义）和类外定义。
- 在类体中定义的成员函数，程序规模比较小且不包括循环等控制，C++会自动将它们作为内联函数来处理，声明时可以省略inline；
- 如果成员函数在类体外定义，必须在函数名前加类名，即类名::；成员函数必须先在类体中作原型声明，然后在类外定义；系统并不把它默认为内联函数，若想将成员函数指定为内联函数，应当用inline作显式声明。

类：成员函数

- 在声明和定义类的成员函数时，应综合使用类内定义和类外定义。
- 使用类内定义，能够隐式声明内联函数，这些成员函数一般为简单的、直接的、与类的创生、赋值、消亡有直接关系的；
- 使用类外定义，不直接在类体中定义一些复杂的成员函数，能够减少类体长度，使类体清晰，而且有助于把类的接口和类的实现细节分离。

PART 02

对象：类的实例化

类与对象

- 类对象是类类型的变量，先有类再有对象；一个对象就是一个实际问题域中的一个实体。它包含了数据结构和所提供的相关操作功能，即，对象是属性和服务的封装体。
- 对象的属性用于描述对象的静态数据特征。对象的属性可用系统的或用户自定义的数据类型来表示。
- 对象的服务用于描述对象的动态特征，它是定义在对象属性基础上的一组操作方法（method）的集合。

对象的定义：静态对象

- 静态对象定义可以通过以下几种方式进行：

①先声明类类型，然后再定义对象；

②在声明类类型的同时定义对象；

③不出现类名，直接定义对象。

```
class Student{
    public:
           void display() {…};
        private:
            ….
};
Student stud1,stud2;
```

```
class Student{
    public:
           void display() {…};
        private:
            ….
}stud1,stud2;
```

```
class {
    public:
           void display() {…};
        private:
            ….
}stud1,stud2;
```

对象的定义：动态对象

- 在声明类的基础上，动态对象采用new和delete来创建和删除对象；
- 单个动态对象的创建与撤销：

```
A *p;    //p为一个指针变量
p=new A; //创建一个A类的动态对象，p指向该对象
delete p; //销毁p指向的这个对象
```

- 动态对象数组的创建与撤销：

```
A *p;  //p为一个指针变量
p=new A[100];  //创建连续的A类型的空间，p指向首元素
delete [ ]p;
```

对象成员的引用

- 在类的声明和实现之外，只能访问public成员，而不能访问private成员。(private成员只能通过该类的公有成员函数来访问它们)
- 对象成员的引用形式：
  - 通过对象名和成员运算符访问对象中的成员；
  - 通过指向对象的指针访问对象中的成员；
  - 通过对象的引用访问对象中的成员。

对象成员的引用方法

```
class A {
    private:
        int x;
        void g( ) {   //允许访问:x,f,g
        }
    pulbic:
        void f() {   //类成员函数允许访问:x,f,g
        }
};//类外允许访问:f
```

```
A a;
a.f();   //True
a.x=1;   //False
a.g();   //False
```

对象成员的引用

- 通过对象名和成员运算符访问对象中的成员的一般形式：对象名.成员名
- “.”是成员运算符，用来对成员进行限定，指明所访问的是哪一个对象中的成员（可以是数据成员或是成员函数）；成员函数中访问数据成员时，不需要加点操作符（这是因为这些变量属于当前对象的数据成员）。

对象成员的引用

```
class Student {
  private:
      int num;
      int score;
   public:
      void setdata(){
        cin>>num>>score;
      }
      void display(){
        cout<<“num=“<<num<<endl;
        cout<<“score=“<<score<<endl;
      };
}stud1,stud2;
```

```
stud1.setdata();   //OK
stud2.setdata();   //OK
stud1.display();   //OK
stud2.display();   //OK
stud2.score=6;     //ERROR
```

对象成员的引用

- 通过指向对象的指针访问对象中的成员：

```
Tdate t,*p;    //定义对象t和指向Time类的指针变量p
p=&t;          //建立指向关系
p->set(11,12,2016);  //或是(*p).set(11,12,2016);
```

- 通过对象的引用来访问对象中的成员：

```
Time t1;                    //定义对象t1
Time &t2=t1;          //声明Time类引用
t1.setTime(11,25,30);  //直接通过对象名调用成员函数
t2.setTime(15,25,30);  //通过引用来调用成员函数，与对象名一致
```

对象：this指针

- 在每一个类成员函数的形参表中都有一个隐含的指针变量this，该指针变量的类型就是成员函数所属类的类型；
- 当程序中调用成员函数时，this指针变量被自动初始化为发出函数调用的对象的地址；
- this指针变量是隐含的，但是在成员函数的函数体内可以使用this指针变量。

```
class Example{
    private:
        int m;
    public:
        void setvalue(int arg1) {m=arg1;}
}s;
s.setvalue(100);
```

通过调用setvalue成员函数，将对象s的数据成员m的值赋成100，而不会应用于其他对象。其原因是成员函数的原型：

```
void setvalue(Example *this,int arg1){this->m=arg1; }
```

对象：this指针

- 关于this指针变量的参数传递，都是由编译系统自动实现的，程序员无须人为在形参中增加this指针或是将对象的地址传递给this指针；
- 在需要时，可以显式地使用this指针。即：使用*this表示当前的对象，表示被调用的成员函数所在的对象；
- 通过使用this指针，保证了每个对象可以拥有不同的数据成员，但处理这些数据成员的代码可以被所有的对象共享。

```
bool same_isbn(const Sales_item &rhs) const{
    return this->isbn==rhs.isbn;    }
```

PART 03

实践：设计自己的类

如何声明一个类

- 以声明一个矩形类为例：
  - 先考虑：问题域中有什么？
  - 会分析出：哪些是数据，有哪些操作？
  - 特征：哪些是想隐藏的，哪些是公开的？为何隐藏/公开？
  - 延伸到：如何去获取/应用私有成员？

如何声明一个类

静态特征：

```
float width;
float length;
```

动态特征：

```
SetData( );
CalculateArea( );
getWidth( );
getLength( );
```

容易忽视
（如何输出？）

```
void SetData(float f1, float f2){
    width = f1;
    length = f2;
}
float CalculateArea( ){
    return getWidth()*getLength();
}
float getWidth(){
    return width;
}
float getLength(){
    return length;
}
```

如何声明一个类

```
class Tdate // 定义日期类
{
   public:
        void Set(int m,int d,int y);
        int IsLeapYear();
       void Print();
   private:
        int month;
        int day;
        int year;
};
```

- 以自行设计实现这三个成员函数。

*请设计为类外函数

如何声明一个类

```
class Tdate // 定义日期类
{
   public:
        void Set(int m,int d,int y);
        int IsLeapYear();
       void Print();
   private:
        int month;
        int day;
        int year;
};
```

```
void Tdate::Set(int m,int d,int y){
    month=m; day=d; year=y;
}
int Tdate::IsLeapYear(){
    return (year%4==0&&year%100!=0)||(year%400==0);
}
void Tdate::Print(){
    cout<<month<<"/"<<day <<"/"<<year<<endl;
}
```

感谢倾听

给个好评！