第4章（一）

本章导学

- 程序中的对象是现实中对象的模拟，具有属性和功能/行为；
- 抽象出同一类对象的共同属性和行为，形成类，对象是类的实例；
- 类将数据和处理数据的函数封装在一起，隐藏内部细节，提供对外访问接口；
- 定义对象时，可以通过构造函数进行初始化；
- 删除对象时，可以通过析构函数释放资源
- 一个类的对象可以由其他类的对象组合而成，即类的成员可以是其他类的对象；
- 在这一章，我们还将学习结构体、联合体和枚举类

面向对象程序设计的基本特点

抽象

- 对同一类对象的共同属性和行为进行概括，形成类。
  - 先注意问题的本质及描述，其次是实现过程或细节。
  - 数据抽象：描述某类对象的属性或状态（对象相互区别的物理量）。
  - 代码抽象：描述某类对象的共有的行为特征或具有的功能。
  - 抽象的实现：类。
- 抽象实例——钟表
  - 数据抽象：
    int hour,int minute,int second
  - 代码抽象：
    setTime(),showTime()

```
class  Clock {
  public:
   void setTime(int newH, int newM, int newS);
   void showTime();
  private:
   int hour, minute, second;
```

```
};
```

封装

- 将抽象出的数据、代码封装在一起，形成类。
  - 目的：增强安全性和简化编程，使用者不必了解具体的实现细节，而只需要通过外部接口，以特定的访问权限，来使用类的成员。
  - 实现封装：类声明中的{}
- 例：

```
class  Clock {
  public: void setTime(int newH, int newM, int newS);
                void showTime();
  private: int hour, minute, second;
};
```

继承

- 在已有类的基础上，进行扩展形成新的类。
- 详见第 7 章

多态

- 多态：同一名称，不同的功能实现方式。
- 目的：达到行为标识统一，减少程序中标识符的个数。
- 实现：重载函数和虚函数——见第 8 章

类和对象的定义

- 对象是现实中的对象在程序中的模拟。
- 类是同一类对象的抽象，对象时类的某一特定实体。
- 定义类的对象，才可以通过对象使用类中定义的功能。

设计类就是设计类型

- 此类型的“合法值”是什么？
- 此类型应该有什么样的函数和操作符？
- 新类型的对象该如何被创建和销毁？
- 如何进行对象的初始化和赋值？
- 对象作为函数的参数如何以值传递？

- 谁将使用此类型的对象成员？

类定义的语法形式

```
class 类名称
{
  public:
            公有成员（外部接口）
  private:
            私有成员
  protected:
            保护型成员
}
```

类内初始值

- 可以为数据成员提供一个类内初始值
- 在创建对象时，类内初始值用于初始化数据成员
- 没有初始值的成员将被默认初始化。
- 类内初始值举例

```
class Clock {
public:
    void setTime(int newH, int newM, int newS);
    void showTime();
private:
    int hour = 0, minute = 0, second = 0;
};
```

类成员的访问控制

- 公有类型成员
  - 在关键字public后面声明，它们是类与外部的接口，任何外部函数都可以访问公有类型数据和函数。
- 私有类型成员
  - 在关键字private后面声明，只允许本类中的函数访问，而类外部的任何函数都不能访问。
  - 如果紧跟在类名称的后面声明私有成员，则关键字private可以省略。

- 保护类型成员
  - 与private类似，其差别表现在继承与派生时对派生类的影响不同，详见第七章。

对象定义的语法

- 类名 对象名；
- 例：Clock myClock;

类成员的访问权限

- 类中成员互相访问
  - 直接使用成员名访问
- 类外访问
  - 使用“对象名.成员名”方式访问 public 属性的成员

类的成员函数

- 在类中说明函数原型；
- 可以在类外给出函数体实现，并在函数名前使用类名加以限定；
- 也可以直接在类中给出函数体，形成内联成员函数；
- 允许声明重载函数和带默认参数值的函数。

内联成员函数

- 为了提高运行时的效率，对于较简单的函数可以声明为内联形式。
- 内联函数体中不要有复杂结构（如循环语句和switch语句）。
- 在类中声明内联成员函数的方式：
  - 将函数体放在类的声明中。
  - 使用inline关键字。

类和对象程序举例

例 4-1 钟表类

类的定义

```
#include<iostream>
using namespace std;
class Clock{
public:
    void setTime(int newH = 0, int newM = 0, int newS = 0);
    void showTime();
```

```
  private:
      int hour, minute, second;
  }
```

成员函数的实现

```
  void Clock::setTime(int newH, int newM, int newS) {
    hour = newH;
    minute = newM;
    second = newS;
  }
  void Clock::showTime() {
    cout << hour << ":" << minute << ":" << second;
  }
```

对象的使用

```
  int main() {
      Clock myClock;
      myClock.setTime(8, 30, 30);
      myClock.showTime();
      return 0;
  }
```

构造函数

构造函数的作用

- 在对象被创建时使用特定的值构造对象，将对象初始化为一个特定的初始状态。
- 例如：
  - 希望在构造一个Clock类对象时，将初试时间设为0:0:0，就可以通过构造函数来设置。

构造函数的形式

- 函数名与类名相同；
- 不能定义返回值类型，也不能有return语句；
- 可以有形式参数，也可以没有形式参数；
- 可以是内联函数；

- 可以重载；
- 可以带默认参数值。

构造函数的调用时机

- 在对象创建时被自动调用
- 例如：

```
Clock myClock ( 0,0,0 ) ;
```

默认构造函数

- 调用时可以不需要实参的构造函数
  - 参数表为空的构造函数
  - 全部参数都有默认值的构造函数
- 下面两个都是默认构造函数，如在类中同时出现，将产生编译错误：

```
Clock();
Clock(int newH=0,int newM=0,int newS=0);
```

隐含生成的构造函数

- 如果程序中未定义构造函数，编译器将在需要时自动生成一个默认构造函数
  - 参数列表为空，不为数据成员设置初始值；
  - 如果类内定义了成员的初始值，则使用内类定义的初始值；
  - 如果没有定义类内的初始值，则以默认方式初始化；
  - 基本类型的数据默认初始化的值是不确定的。

“=default”

- 如果程序中未定义构造函数，默认情况下编译器就不再隐含生成默认构造函数。如果此时依然希望编译器隐含生成默认构造函数，可以使用“=default”。
- 例如

```
class Clock {
public:
 Clock() =default; //指示编译器提供默认构造函数
 Clock(int newH, int newM, int newS);  //构造函数
private:
 int hour, minute, second;
};
```

构造函数例题（1）

例 4_1 修改版 1

```
//类定义
class Clock {
public:
    Clock(int newH,int newM,int newS);//构造函数
    void setTime(int newH, int newM, int newS);
    void showTime();
private:
    int hour, minute, second;
};

//构造函数的实现：
Clock::Clock(int newH,int newM,int newS): hour(newH),minute(newM),
    second(newS) {
    }
//其它函数实现同例4_1

int main() {
  Clock c(0,0,0); //自动调用构造函数
  c.showTime();
    return 0;
}
```

构造函数例题（2）

例 4_1 修改版 2

```
class Clock {
public:
      Clock(int newH, int newM, int newS);  //构造函数
      Clock(); //默认构造函数
```

```
        void setTime(int newH, int newM, int newS);
        void showTime();
private:
        int hour, minute, second;
};
Clock::Clock(): hour(0),minute(0),second(0){ }//默认构造函数
//其它函数实现同前

int main() {
    Clock c1(0, 0, 0);        //调用有参数的构造函数
    Clock c2;          //调用无参数的构造函数
    ……
}
```

委托构造函数

类中往往有多个构造函数，只是参数表和初始化列表不同，其初始化算法都是相同的，这时，为了避免代码重复，可以使用委托构造函数。

回顾

Clock类的两个构造函数：

```
Clock(int newH, int newM, int newS) : hour(newH),minute(newM),
second(newS)  {      //构造函数
}
Clock::Clock(): hour(0),minute(0),second(0) { }//默认构造函数
```

委托构造函数

- 委托构造函数使用类的其他构造函数执行初始化过程

- 例如：

```
Clock(int newH, int newM, int newS):  hour(newH),minute(newM),
second(newS){
}
Clock(): Clock(0, 0, 0) { }
```

复制构造函数

复制构造函数定义

复制构造函数是一种特殊的构造函数，其形参为本类的对象引用。作用是用一个已存在的对象去初始化同类型的新对象。

```
class 类名 {
public :
   类名（形参）; //构造函数
   类名（const  类名 &对象名）; //复制构造函数
   //       ...
} ;
类名::类（  const  类名 &对象名）//复制构造函数的实现
{    函数体    }
```

隐含的复制构造函数

- 如果程序员没有为类声明拷贝初始化构造函数，则编译器自己生成一个隐含的复制构造函数。
- 这个构造函数执行的功能是：用作为初始值的对象的每个数据成员的值，初始化将要建立的对象的对应数据成员。

“=delete”

- 如果不希望对象被复制构造
  - C++98做法：将复制构造函数声明为private，并且不提供函数的实现。
  - C++11做法：用“=delete”指示编译器不生成默认复制构造函数。
- 例：

```
class Point {   //Point 类的定义
public:
    Point(int xx=0, int yy=0) { x = xx; y = yy; }    //构造函数，内联
    Point(const Point& p) =delete;  //指示编译器不生成默认复制构造函数
private:
    int x, y; //私有数据
};
```

复制构造函数被调用的三种情况

- 定义一个对象时，以本类另一个对象作为初始值，发生复制构造；

- 如果函数的形参是类的对象，调用函数时，将使用实参对象初始化形参对象，发生复制构造；
- 如果函数的返回值是类的对象，函数执行完成返回主调函数时，将使用return语句中的对象初始化一个临时无名对象，传递给主调函数，此时发生复制构造。
  - 这种情况也可以通过移动构造避免不必要的复制（第6章介绍）

例 4-2 Point 类的完整程序

```
class Point {   //Point 类的定义
public:
    Point(int xx=0, int yy=0) { x = xx; y = yy; }   //构造函数，内联
    Point(const Point& p); //复制构造函数
  void setX(int xx) {x=xx;}
  void setY(int yy) {y=yy;}
    int getX() const { return x; } //常函数（第5章）
    int getY() const { return y; } //常函数（第5章）
private:
    int x, y; //私有数据
};

//复制构造函数的实现
Point::Point (const Point& p) {
  x = p.x;
  y = p.y;
  cout << "Calling the copy constructor " << endl;
}

//形参为Point类对象void fun1(Point p) {
    cout << p.getX() << endl;
}

//返回值为Point类对象Point fun2() {
    Point a(1, 2);
    return a;
}
```

```
int main() {
    Point a(4, 5);
    Point b(a);   //用a初始化b。
    cout << b.getX() << endl;
    fun1(b);              //对象b作为fun1的实参
    b = fun2();  //函数的返回值是类对象
    cout << b.getX() << endl;
    return 0;
}
```

析构函数

- 完成对象被删除前的一些清理工作。
- 在对象的生存期结束的时刻系统自动调用它，然后再释放此对象所属的空间。
- 如果程序中未声明析构函数，编译器将自动产生一个默认的析构函数，其函数体为空。
- 构造函数和析构函数举例

```
#include <iostream>
using namespace std;
class Point {
public:
  Point(int xx,int yy);
  ~Point();
  //...其他函数原型
private:
  int x, y;
};
```

类的组合

组合的概念

- 类中的成员是另一个类的对象。
- 可以在已有抽象的基础上实现更复杂的抽象。

类组合的构造函数设计

- 原则：不仅要负责对本类中的基本类型成员数据初始化，也要对对象成员初始化。
- 声明形式：

类名::类名(对象成员所需的形参，本类成员形参)
:对象1(参数)，对象2(参数)，.....
{
//函数体其他语句
}

构造组合类对象时的初始化次序

- 首先对构造函数初始化列表中列出的成员（包括基本类型成员和对象成员）进行初始化，初始化次序是成员在类体中定义的次序。
  - 成员对象构造函数调用顺序：按对象成员的声明顺序，先声明者先构造。
  - 初始化列表中未出现的成员对象，调用用默认构造函数（即无形参的）初始化
- 处理完初始化列表之后，再执行构造函数的函数体。

类组合程序举例

- 例4-4 类的组合，线段（Line）类

```
//4_4.cpp
#include <iostream>
#include <cmath>
using namespace std;
class Point {        //Point类定义
public:
    Point(int xx = 0, int yy = 0) {
      x = xx;
      y = yy;
    }
    Point(Point &p);
    int getX() { return x; }
    int getY() { return y; }
private:
```

```
    int x, y;
};

Point::Point(Point &p) { //复制构造函数的实现
    x = p.x;
    y = p.y;
    cout << "Calling the copy constructor of Point" << endl;
}

//类的组合
class Line {//Line类的定义
public:      //外部接口
    Line(Point xp1, Point xp2);
    Line(Line &l);
    double getLen() { return len; }
private:     //私有数据成员
    Point p1, p2;  //Point类的对象p1,p2
    double len;
};

//组合类的构造函数
Line::Line(Point xp1, Point xp2) : p1(xp1), p2(xp2) {
    cout << "Calling constructor of Line" << endl;
    double x = static_cast<double>(p1.getX() - p2.getX());
    double y = static_cast<double>(p1.getY() - p2.getY());
    len = sqrt(x * x + y * y);
}
Line::Line (Line &l): p1(l.p1), p2(l.p2) {//组合类的复制构造函数
    cout << "Calling the copy constructor of Line" << endl;
    len = l.len;
}

//主函数
```

```
int main() {
    Point myp1(1, 1), myp2(4, 5);      //建立Point类的对象
    Line line(myp1, myp2);      //建立Line类的对象
    Line line2(line);       //利用复制构造函数建立一个新对象
    cout << "The length of the line is: ";
    cout << line.getLen() << endl;
    cout << "The length of the line2 is: ";
    cout << line2.getLen() << endl;
    return 0;
}
```

前向引用声明

- 类应该先声明，后使用
- 如果需要在某个类的声明之前，引用该类，则应进行前向引用声明。
- 前向引用声明只为程序引入一个标识符，但具体声明在其他地方。
- 例：

```
class B;  //前向引用声明
class A {
public:
  void f(B b);
};
class B {
public:
  void g(A a);
};
```

前向引用声明注意事项

- 使用前向引用声明虽然可以解决一些问题，但它并不是万能的。
- 在提供一个完整的类声明之前，不能声明该类的对象，也不能在内联成员函数中使用该类的对象。
- 当使用前向引用声明时，只能使用被声明的符号，而不能涉及类的任何细节。
- 例

```
class Fred; //前向引用声明
```

```
class Barney {
   Fred x; //错误：类Fred的声明尚不完善
};
class Fred {
   Barney y;
};
```

结构体

- 结构体是一种特殊形态的类
  - 与类的唯一区别：类的缺省访问权限是private，结构体的缺省访问权限是public
  - 结构体存在的主要原因：与C语言保持兼容
- 什么时候用结构体而不用类
  - 定义主要用来保存数据、而没有什么操作的类型
  - 人们习惯将结构体的数据成员设为公有，因此这时用结构体更方便

结构体的定义

```
struct 结构体名称 {
      公有成员
protected:
    保护型成员
private:
     私有成员
};
```

结构体的初始化

- 如果一个结构体的全部数据成员都是公共成员，并且没有用户定义的构造函数，没有基类和虚函数（基类和虚函数将在后面的章节中介绍），这个结构体的变量可以用下面的语法形式赋初值

类型名 变量名 = { 成员数据1初值, 成员数据2初值, ... };

例 4-7 用结构体表示学生的基本信息

```
#include <iostream>
#include <iomanip>
```

```
#include <string>
using namespace std;

struct Student {   //学生信息结构体
    int num;              //学号
    string name;  //姓名，字符串对象，将在第6章详细介绍
    char sex;             //性别
    int age;        //年龄
};

int main() {
    Student stu = { 97001, "Lin Lin", 'F', 19 };
    cout << "Num:  " << stu.num << endl;
    cout << "Name: " << stu.name << endl;
    cout << "Sex:  " << stu.sex << endl;
    cout << "Age:  " << stu.age << endl;
    return 0;
}
```

运行结果：

```
Num:  97001
Name: Lin Lin
Sex:  F
Age:  19
```

联合体

声明形式

```
union 联合体名称 {
    公有成员
protected:
    保护型成员
private:
```

私有成员
};

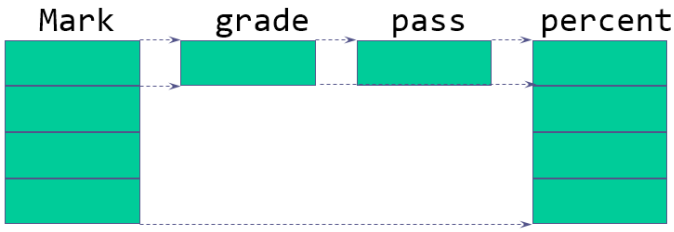
特点：

- 成员共用同一组内存单元
- 任何两个成员不会同时有效

联合体的内存分配

- 举例说明：

```
union Mark {      //表示成绩的联合体
    char grade;   //等级制的成绩
    bool pass;    //只记是否通过课程的成绩
    int percent;  //百分制的成绩
};
```



无名联合

- 例：

```
union {
  int i;
  float f;
}
```

在程序中可以这样使用：

```
i = 10;
f = 2.2;
```

下面我们看一个联合体的例题

例 4-8 使用联合体保存成绩信息，并且输出。

```
#include <string>
#include <iostream>
using namespace std;
class ExamInfo {
```

```
private:
    string name;  //课程名称
    enum { GRADE, PASS, PERCENTAGE } mode;//计分方式
    union {
      char grade;  //等级制的成绩
      bool pass;   //只记是否通过课程的成绩
      int percent;  //百分制的成绩
    };
public:
    //三种构造函数，分别用等级、是否通过和百分初始化
    ExamInfo(string name, char grade)
      : name(name), mode(GRADE), grade(grade) { }
    ExamInfo(string name, bool pass)
      : name(name), mode(PASS), pass(pass) { }
    ExamInfo(string name, int percent)
      : name(name), mode(PERCENTAGE), percent(percent) { }
    void show();
}

void ExamInfo::show() {
    cout << name << ": ";
    switch (mode) {
      case GRADE: cout << grade;  break;
      case PASS: cout << (pass ? "PASS" : "FAIL"); break;
      case PERCENTAGE: cout << percent; break;
    }
    cout << endl;
}

int main() {
    ExamInfo course1("English", 'B');
    ExamInfo course2("Calculus", true);
    ExamInfo course3("C++ Programming", 85);
```

```
    course1.show();
    course2.show();
    course3.show();
    return 0;
}
```

运行结果：

English: B

Calculus: PASS

C++ Programming: 85

枚举类

枚举类定义

- 语法形式

enum class 枚举类型名: 底层类型 {枚举值列表};

- 例：

```
enum class Type { General, Light, Medium, Heavy};
enum class Type: char { General, Light, Medium, Heavy};
enum class Category { General=1, Pistol, MachineGun, Cannon};
```

枚举类的优势

- 强作用域，其作用域限制在枚举类中。
  - 例：使用Type的枚举值General：

  Type::General
- 转换限制，枚举类对象不可以与整型隐式地互相转换。
- 可以指定底层类型
  - 例：

  enum class Type: char { General, Light, Medium, Heavy};

例 4-9 枚举类举例

```
#include<iostream>
using namespace std;
enum class Side{ Right, Left };
enum class Thing{ Wrong, Right };  //不冲突
```

```
int main()
{
Side s = Side::Right;
Thing w = Thing::Wrong;
cout << (s == w) << endl;  //编译错误，无法直接比较不同枚举类
return 0;
}
```

小结

- 主要内容
  - 面向对象的基本概念、类和对象的声明、构造函数、析构函数、内联成员函数、复制构造函数、类的组合
- 达到的目标
  - 掌握面向对象的基本概念；
  - 掌握类设计的思想、类和对象声明的语法；
  - 理解构造函数、复制构造函数和析构函数的作用和调用过程，掌握相关的语法；
  - 理解内联成员函数的作用，掌握相关语法；
  - 理解类的组合在面向对象设计中的意义，掌握类组合的语法。
  - 了解枚举类