Chap02 面向对象程序设计方法

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

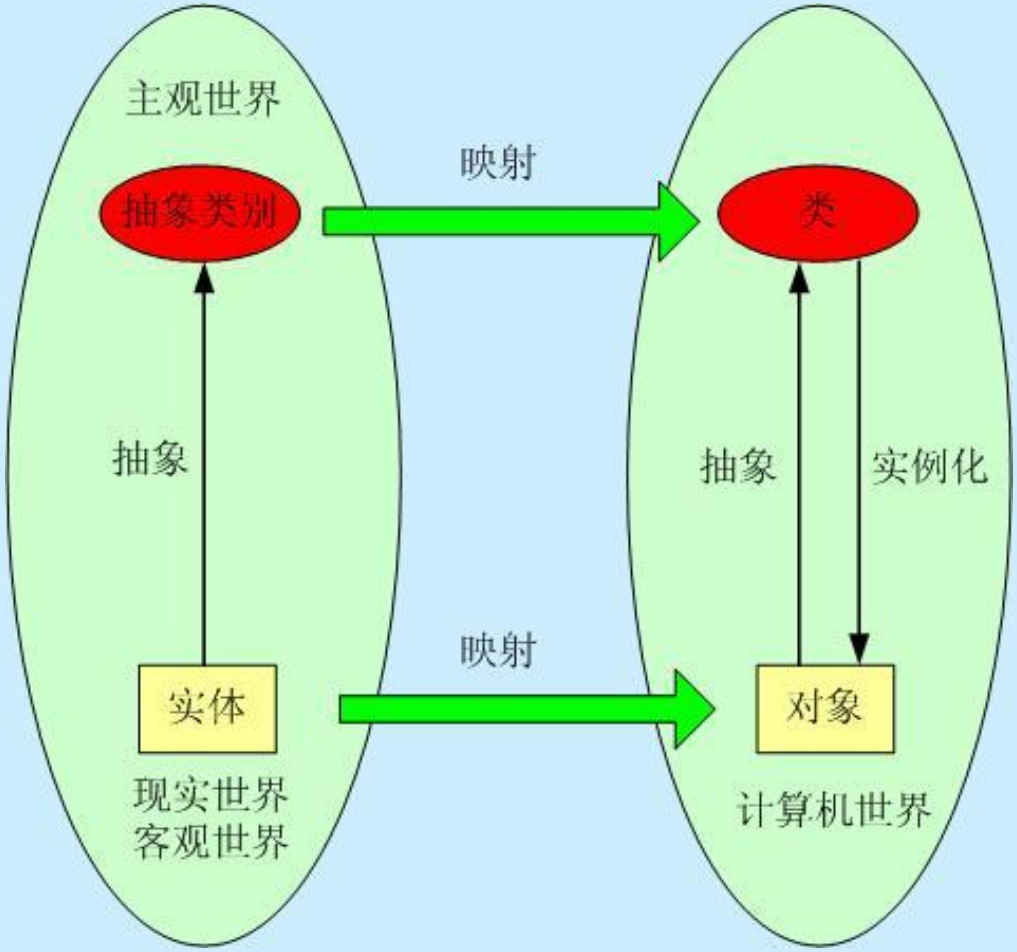
- 面向对象编程思想
  - 类和对象概念
  - 封装性/继承性/多态性
- 类的定义
  - 成员变量
  - 成员函数
  - 访问权限
- 对象的定义
  - 构造函数
  - 析构函数
  - 调用顺序
- this指针
- 对象成员的使用
- 静态成员
- 对象数组
- 友元

面向过程编程

- 基本思想
  - 自顶向下，逐步求精(Divide and Conquer, Stepwise Refinement)；
- 开发方法
  - 结构化分析(SA)：用数据流程图(DFD)描述系统中信息的变换和传递过程，并辅以其他形式进行说明；
  - 结构化设计(SD)：将系统划分为一个个模块，模块是按功能来划分的基本单位。在C++中实现为函数，一个函数实现一个功能或一个操作，解决一个子问题；
  - 结构化编程(SP)：如C语言等；
- 主要缺点
  - 面向过程的结构化程序设计的方法与现实世界(包括主观世界和客观世界)往往都不一致；
  - 当软件规模过大，开发和维护就越来越难控制；

面向对象编程

- 实体、类与对象



面向对象编程(续)

- 基本思想
  - 面向对象程序设计模拟自然界认识和处理事物的方法，将数据和对数据的操作封装为一个相对独立的整体－对象(Object)，同类对象还可抽象出共性，形成类(Class)。一个类中的数据通常只能通过本类提供的方法进行处理，这些方法成为该类与外部的接口，对象之间通过消息(Message)进行通信；
- 核心概念
  - 类(Class)

    类是具有相同属性和行为的一组对象的集合，它为属于该类的全部对象提供了统一的抽象描述，其内部包括属性和行为两个主要部分；
  - 对象(Object)

    类的对象是该类的某一特定实体，即类类型的变量；
  - 消息(Message)

    对象之间产生相互作用所传递的信息称做消息；

面向对象编程(续)

- 主要特性
  - 封装性(Encapsulation)

    封装是面向对象程序设计最基本的特性，也就是把数据(属性)和函数(操作)合成一个整体，这是用类与对象实现的；
  - 继承性(Inheritance)

    继承是根据一个类型定义另一个类型的能力。它是面向对象程序设计使代码可以复用的最重要的手段，它允许程序员在保持原有类特性的基础上进行扩展，增加功能，从而在已有类的基础上产生新的类；
  - 多态性(Polymorphism)

    多态表示在不同的时刻有不同的形态；即同一个消息被不同对象接收时，产生不同结果，即实现同一接口，不同方法。

类定义

- 类是用户定义的数据类型
- 类的例子

```
class Box {
public:
    Box(double lengthval=1.0, double widthval=1.0,double heightval=1.0);
    double volume()  const;
private:
    double length;
    double width;
    double height;
};
```

  - 关键字class是数据类型说明符，Box是所定义的类型的名称；
  - public、private是访问限定符；
  - 最后的分号不可少；

类定义(续)

- 访问限定符
  - public
    类的公共成员可以直接在类外部访问，因此这些成员是不隐藏的；
  - protected
    类的受保护成员不能直接在类外部访问；
  - private
    类的私有成员仅可由类的成员函数访问；
  - 每种访问说明符可在类体中使用多次；
  - 默认情况下，类对象的成员是私有的；
- 注意事项
  - 通常数据成员被说明成私有的，函数成员被说明成公有的。从外部对数据成员进行操作，只能通过公有函数来完成，数据受到了良好的保护；
  - 类是一种数据类型，定义时系统不为类分配存储空间，故不能对类的数据成员初始化。类中的任何数据成员也不能使用关键字extern、auto或register限定其存储类型；

构造函数

- 类的构造函数是一种特殊的函数，主要用于创建新对象并进行初始化
- 注意事项
  - 构造函数的函数名必须与类名相同；
  - 构造函数不需要也不允许有返回类型说明；
  - 构造函数可以重载，系统调用时按一般函数重载的规则选一个构造函数执行；
  - 构造函数可以在类中定义，也可以在类外定义；
- 默认构造函数
  - 如果类说明中没有给出构造函数，则C++编译器自动给出一个默认的构造函数，但一旦定义了一个构造函数，系统就不会自动生成默认的构造函数；
  - 只要构造函数是无参的或各参数均有默认值的，C++编译器都认为是默认的构造函数，并且默认的构造函数只能有一个；

构造函数(续)

- 默认构造函数的一个主要特性是调用时不需要指定参数列表，甚至不需要括号；

- 默认初始化值
  - 如果在类定义中包含了函数的声明，则默认的参数值应该放在声明中，而不是在函数定义中；
  - 有默认参数值的构造函数也可以作为默认构造函数；

- 初始化列表

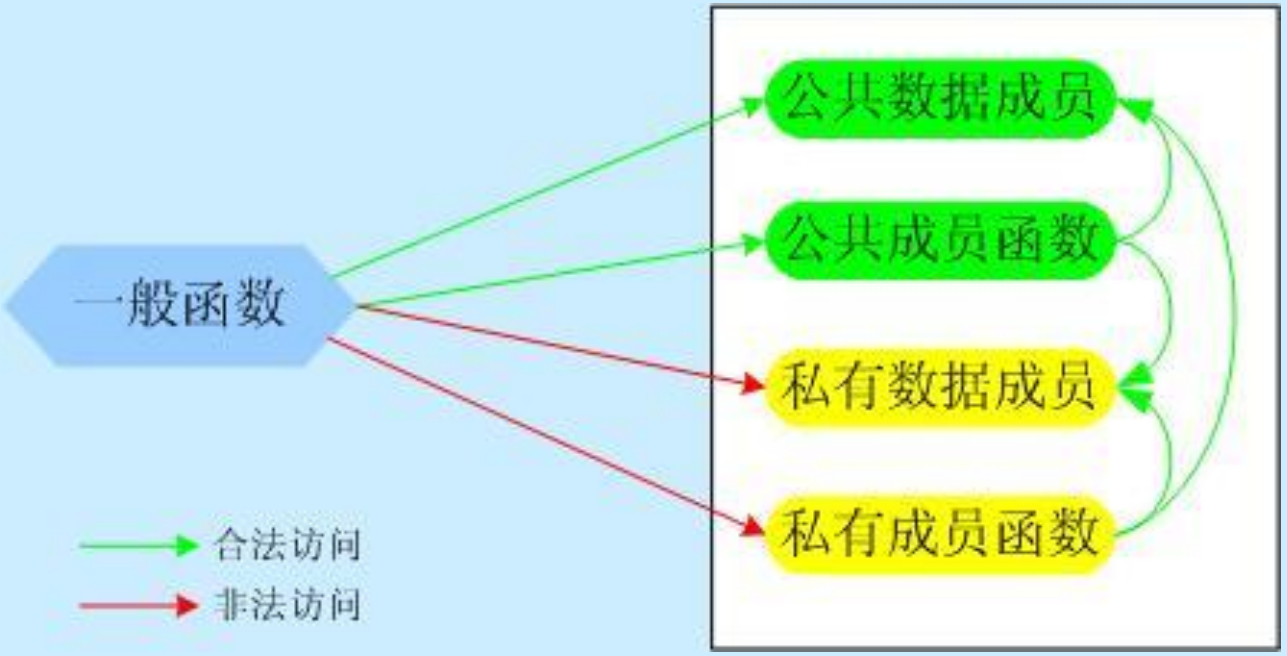
  数据成员的值用函数表示法指定为初始值，并显示在初始化列表中，作为函数头的一部分；

```
Box::Box(double lengthval, double widthval, double  heightval) :
                    length(lengthval), width(widthval), height(heightval){
      cout <<“Box constructor called.” << endl;
}
```

- explicit关键字
  - 对只有一个参数的构造函数，使用explicit关键字可以避免进行隐式转换；

私有成员

- 类的私有成员
  - 由private访问限定符进行说明；
  - 私有成员只能由类的成员函数进行访问，如果某个函数不是类的成员函数，就不能直接访问该类的私有成员；
  - 一般情况下，推荐将数据成员设置为私有成员；



友元

- 可以使用friend关键字将某个函数或类指定为某个类的友元，友元可以访问类对象的任意成员，无论这些成员的访问指定符是什么
- 友元函数

```
friend double boxsurface(const Box& thebox);  // 声明有元函数
```

- 友元类

```
friend class carton;   // 声明友元类
```

- 注意事项
  - 友元可以是一个全局函数，也可以是另一个类的成员；
  - 访问指定符不能应用与类的友元；
  - 友元是类接口的一部分，但较好的编程方式是只要有可能，就应该根据需要的成员函数来定义类的接口，惟一需要友元的情形是当需要访问两个不同类的非共有成员时；

this指针

- 在执行任何类成员函数时，该函数都会自动包含一个隐藏的指针，称为this指针，该指针包含了调用该函数的对象的地址
- 常见用法
  - 用this区分数据成员与形参

```
Box::setlength(double length) {
    this->length=length;
    this->volume();              //通过this指针访问成员函数
}
```

  - 使用*this获取整个对象

```
Box Box::getbox() {
    return *this;
}
```

静态成员

- 静态数据成员
  - 类的数据成员和成员函数都可以声明为static，类的静态数据成员可以在类的范围内存储数据，而这种数据独立于类的任何对象；
  - 将类的数据成员声明为static，则该静态数据成员只定义一次，且即使没有创建对象实例，该静态数据成员依然存在；
- 静态成员函数
  - 将成员函数声明为static就可以使它独立于类的对象；
- 注意事项
  - 对静态数据成员进行初始化需要在类的外部进行，即使类的静态成员指定为私有的，仍然可以用这种方式进行初始化；
  - 静态数据成员不是类种各对象的一部分，所以它可以与类具有相同的类型；
  - 类的静态成员函数不能声明为const，因为它与类的对象无关，它没有this指针，所以不能使用this关键字；

常对象和常成员函数

- 常对象
  - 如果将一个对象指定为const，即告诉编译器不要修改它；
- 常成员函数
  - 对于声明为const的对象，只能调用也声明为const的函数，const成员函数不会修改调用它的对象；
  - 如果要将成员函数声明为const，则需要在类定义时在函数声明的最后加上关键字const；

    double volume() const;
  - 在函数定义时，也必须在后面加上const；
  - 一般访问器函数都声明为const；
- 注意事项
  - 下面两个函数是重载函数！

```
int Box::compareVolume(const Box& otherbox);
int Box::compareVolume(const Box& otherbox) const;
```

对象数组

- 声明类的对象数组的方式与声明其他类型的数组的方式完全相同，类对象的每个元素都是独立创建的，编译器要为每个元素调用默认构造函数
- 示例

```
Box boxes[5];
for(int i = 0; i < count; i++){
        cout << boxes[i].volume() << endl;
}
```

- 类对象的大小
  - 类对象的大小一般来说是类中数据成员大小的总和，但在某些机器上类对象的大小要比所有数据成员字节总和还要大，这是因为边界对齐的需要；

    sizeof(boxes);

类对象的指针和引用

- 对类对象使用指针
  - 使用指针成员访问运算符->来调用函数
  - 作为函数的参数
  - 作为类的数据成员
- 主要优点
  - 使多态调用函数成为可能
  - 对象指针作为函数参数可以避免隐含的复制操作
  - 对象指针作为类成员可以实现链表等数据结构
  - 对象引用作为函数参数是实现副本构造函数的基础
- 示例：TruckLoad类
- 存在的问题
  - Package可以有任何对象访问
  - TruckLoad对象的复制问题
  - 内存管理问题，存在内存泄漏

副本构造函数

- 副本构造函数
  - 用于创建一个与已有对象完全相同的新对象；
  - 如果类中没有副本构造函数，编译器会在需要的时候提供默认的副本构造函数；
  - 默认副本构造函数通过依次复制每个数据成员实现的；
- 定义副本构造函数
  - 如果对象中动态分配了内存，副本构造函数有可能会导致多个对象共享一个链表；
  - 类与对象的安全性和处理的正确性要求提供特殊的副本构造函数；
  - 副本构造函数的参数类型总是对同一个类对象的const引用；
- 副本构造函数格式

```
Type:: Type(const Type& object){
        ……;   // 副本构造函数体
}
```

析构函数

- 类的析构函数是一种特殊的函数，当对象完成其使命需要销毁时调用，以释放对象占用的资源
- 如果在对象中动态分配了内存，就必须实现该类的析构函数
- 示例

```
Box::~Box(){
        ……
        delete phead;
}
```

- 注意事项
  - 析构函数是类的公共成员函数，与类同名但前面有符号~；
  - 类的析构函数没有参数，也没有返回值；
  - 一个类只能有一个析构函数；
  - 如果没有显示地定义析构函数，则编译器会生成一个默认的析构函数；

类的引用

- 引用主要优点
  - 避免在按值传递过程中的隐式复制过程；
- 引用作为类的成员

```
class Package{
public:
        Box& rbox;
        Package(Box& rnewbox);
}
```

  - 引用是另一个名称的别名，不能在构造函数中用赋值语句来初始化，必须使用初始化列表；
  - 在使用动态对象的引用时，对象有可能被删除，所以引用就成为不存在的对象的别名；
  - 把引用作为类的成员一般不会改善类的功能，故指针应该是首选方案；

本章小结

- 面向对象编程思想
- 类的定义
- 对象的定义
- this指针
- 对象成员的使用
- 静态成员
- 对象数组
- 友元

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