C++语言程序设计

第四章 类与对象

本章主要内容

- 面向对象程序设计的基本特点
- 类概念和声明
- 对象
- 构造函数
- 析构函数
- 内联成员函数
- 拷贝构造函数
- 类的组合
- 结构体与联合体
- 小结

面向对象的思想

回顾：面向过程的设计方法

设计思路：

自顶向下，逐步求精——功能分解。

程序结构：

按功能划分为若干个模块，模块本身还可以再做适当分解，形成树状结构；模块是单入口和单出口的，内部都是由顺序、选择和循环三种基本控制结构组成。

缺点：

效率低，程序的可重用性差。

面向对象的思想

面向对象的方法

- 面向对象语言被分为两大阵营
  - 纯粹的面向对象语言：Java
  - 混合型面向对象语言：C++
- 结构化程序设计
  - 将问题进行功能分解（函数）
  - 效率低，程序的可重用性差。
- 面向对象方法
  - 将问题分解为一系列的实体（对象）
  - 可维护性、可扩充性好

抽象

OOP的基本特点

抽象是对具体对象（问题）进行概括，抽出这一类对象的公共性质并加以描述的过程。

- 数据抽象：描述某类对象的属性或状态。
- 行为抽象（又称功能抽象、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：通过类的声明。

抽象实例——钟表

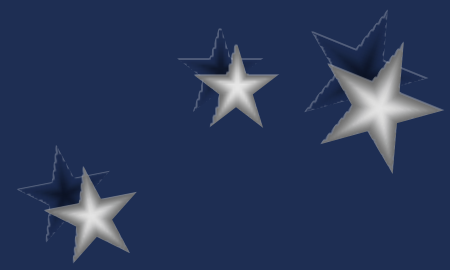
OOP的基本特点

- 数据抽象：

  int hour,int minute,int second

- 代码抽象：

  setTime(),showTime()



抽象实例——钟表类

OOP的基本特点

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

OOP的基本特点

封装

将抽象出的数据成员、代码成员相结合，将它们视为一个整体。

- 目的是曾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{}

封装

OOP的基本特点

- 实例：

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

外部接口

特定的访问权限

边界

继承与派生

OOP的基本特点

是C++中支持层次分类的一种机制，允许程序员在保持原有类特性的基础上，进行更具体的说明。

实现：声明派生类——见第7章

OOP的基本特点

多态性

- 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

c++中的类

类和对象

- 类是具有相同属性和行为的一组对象的集合，其内部包括属性和行为两个主要部分。
- 利用类可以实现数据的封装、隐藏、继承与派生。
- 利用类易于编写大型复杂程序，其模块化程度比C中采用函数更高。

类的声明形式

类是一种用户自定义类型，声明形式：

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

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

在关键字protected后面声明，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，第七章讲。

类的成员

类和对象

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

成员函数

成员数据

成员数据

- 与一般的变量声明相同，但需要将它放在类的声明体中。
- 注 意：

  C++语言不允许在类的主体中定义数据成员时，同时进行初始化。例如以下类的定义是非法的：

```
class clock
      {
        ……
        private:
          int hour=0;    //错误，在类体中不允许对
                         //数据成员进行初始化
       };
```

成员函数的实现

- 在类中说明原型，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。其形式为：

```
返回值类型 类名::成员函数名（参数表）
{
        函数体
}
```

- 也可以直接在类中给出函数体，形成内联成员函数。
- C++类中允许声明重载函数和带默认形参值的函数

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

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：
  - 将函数体放在类体中声明（隐式声明）。
  - 使用inline关键字（显式声明）。

内联成员函数举例(一)

类和对象

```
class Point {
 public:
    void init(int initX, int initY) {
       x = initX;
       y = initY;
    }                          //隐式声明
    int getX() { return x; } //隐式声明
    int getY() { return y; } //隐式声明
 private:
    int x, y;
};
```

内联成员函数举例(二)

类和对象

```
class Point {
 public:
    void init(int initX, int initY);
    int getX();
    int getY();
 private:
    int x, y;
};
```

```
inline void Point::
            init(int initX,int initY) {
    x = initX;
    y = initY;
}   //显式声明

inline int Point::getX() {
    return x;
}  //显式声明

inline int Point::GetY() {
    return y;
}  //显式声明
```

对象

类和对象

- 类的对象是该类的某一特定实体，即类类型的变量。
- 声明形式：

  类名 对象名;
- 例：

```
Clock  myClock;
```

类中成员的访问方式

类和对象

- 类中成员互访
  - 直接使用成员名
- 类外访问
  - 使用“.”操作符访问 public 属性的成员。
  - 访问数据成员的一般形式：

    对象名. 数据成员名
  - 调用函数成员的一般形式：

    对象名.函数成员名(参数表)

类的应用举例

类和对象

```
#include<iostream>
using namespace std;
class Clock {
        ......//类的声明略
}
//......类的实现略
int main() {
   Clock myClock;
   myClock.setTime(8, 30, 30);
   myClock.showTime();
   return 0;
}
```

构造函数和析构函数

构造函数

- C++语言提供了类的两个成员函数，它们分别是**构造函数**和**析构函数**，可以帮助用户自动完成对象的初始化和清理性工作。
- **构造函数**（constructor）是类所特有的成员函数，对象创建时自动被调用，完成对象的初始化工作。
- 允许为**内联**函数、**重载**函数、**带默认形参值**的函数

构造函数

- 语法形式

  类名(参数表);

说 明

1. 构造函数名与类名相同。
2. 构造函数无返回值，不能把void写在构造函数名的前面，函数体中也不能出现return语句。
3. 构造函数的访问控制属性必须是public。

构造函数举例

构造函数和析构函数

```
class Clock {
public:
   Clock(int newH,int newM,int newS); //构造函数
protected:
   Clock(){       //错，构造函数必须是public
            hour=0;
            minute=0;
            second=0;
         }
   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;
}
```

建立对象时构造函数的作用：

```
int main() {
  Clock c1(8,30,30);//隐含调用构造函数，将初始值作为实参
  c.setTime();
  return 0;
}
```

拷贝构造函数

拷贝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。它的作用是自动用一个已经存在的对象，去初始化另一个新的同类的对象。

```
class 类名 {
public :
    类名（形参）; //构造函数
    类名（类名 &对象名）; //拷贝构造函数
        ...
};
类名::类名（类名 &对象名）//拷贝构造函数的实现
{    函数体    }
```

拷贝构造函数(例4-2)


```
class Point {
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;
};
```

拷贝构造函数的实现

```
Point::Point (Point& p) {
   x = p.x;
   y = p.y;
   cout << "Calling the copy constructor "
      << endl;
}
```

拷贝构造函数

- 当用类的一个对象去初始化该类的另一个对象时，系统自动调用拷贝构造函数实现拷贝赋值。

```
int main() {
    Point a(1,2);
    Point b(a); //用对象a初始化对象b，拷贝构造函数被调用
    Point c=a; //用对象a初始化对象c，拷贝构造函数被调用
    cout << b.getX() << endl;
}
```

拷贝构造函数

- 若函数的形参为类对象，调用函数时，实参赋值给形参，系统自动调用拷贝构造函数。例如：

```
void fun1(Point p) {
    cout << p.getX() << endl;
}
int main() {
    Point a(1, 2);
    fun1(a); //调用拷贝构造函数
    return 0;
}
```

拷贝构造函数(例4-2)

- 当函数的返回值是类对象时，系统自动调用拷贝构造函数。例如：

```
Point fun2() {
  Point a(1, 2);
  return a; //函数的返回值是类对象,调用拷贝构造函数
}
int main() {
  Point b;
  b = fun2();
  return 0;
}
```

隐含的拷贝构造函数

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拷贝构造函数。

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。

构造函数和析构函数

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隐含的析构函数。

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：https://d.book118.com/487055012043006161