引用计数

深拷贝的缺点：多次申请堆内存的时间开销，内存中存在多个拷贝开销
利用写时复制，可以有效解决深拷贝所带来的性能与内存上的消耗——用一个引用计数器来对资源区进行引用计数，当引用其对象增加，引用计数器也随之增加，反之减少亦然
当引用计数器的值为0时，则会释放资源区

```
class Testclass {
    ...
    char *buf;          // 资源区
    int *ref_count;     // 引用计数器
    ...
};
```

封装

```
class Clock {
public:
    void set_time(...);
private:
    void set_hour(...);
    void set_minute(...);
    void set_second(...);
private:
    int _hours;
    int _minutes;
    int _seconds;
}
```

封装的意义：

- 数据可以以想要的方式被修改，通过私有权限的方式隐藏数据，通过公有成员函数的方式提供堆数据的访问
- 关注事物提供的功能，而不关注事物功能背后的运作细节
- 对于对象，关注它的接口，而不关注背后实现的细节，实现细节会通过私有权限方式隐藏
- 提高代码的重用性

const成员函数

```
class Testclass {
public:
    ...
    char *get_buffer() const    // const成员函数
    {
        return _buf;
    }
    ...
private:
    ...
    char *_buf;
    ...
};
```

- 不允许修改数据成员
- 不允许调用一般的成员函数，只能调用`const成员函数`
- `const`成员函数的`this`指针的类型为`const type * const`，而一般的成员函数的`this`指针类型为`type * const`
- 若在数据成员前以`mutable`关键字修饰，则在`const`成员函数中可以被修改——即被修饰的成员永远不会为`const`
- `const`对象不可以调用一般成员函数，可以调用`const`成员函数

初始化列表

```
class Testclass {
public:
    Testclass() : a(0), b(3.14)     // 初始化列表
    {
        ...
    }
    ~Testclass()
    {
        ...
    }
private:
    int a;
    double b;
};
```

C++11标准

C++11标准允许直接在数据成员后赋值

```
class Testclass {
public:
    Testclass()
    {
        ...
    }
```

```
    ~Testclass()
    {
        ...
    }
private:
    int a = 0;
    double b = 3,14;
};
```

静态成员

静态数据成员

- 在类内部声明，在类外部定义

```
class Testclass {
private:
    static int a;       // 类内部声明
    ...
};

static Testclass::a;    // 类外部定义
...
```

- 静态数据成员是所有对象共享——即属于类的，一般的数据成员属于对象
- 静态数据成员可以使用类名来访问，也可以使用对象来访问（依然受访问权限限制）
- 成员函数可以使用静态数据成员

静态成员函数

```
class Testclass {
public:
    static void foo()
    {
        ...
    }
};
```

- 可以定义在类内部，也可以定义在内外部
- 静态成员函数只能访问静态的成员，不能访问一般的成员
- 静态成员函数没有传入`this`指针
- 若想访问一般的成员，则需要把对象的引用（指针）显式传参
- 静态成员函数可以使用类名来调用