

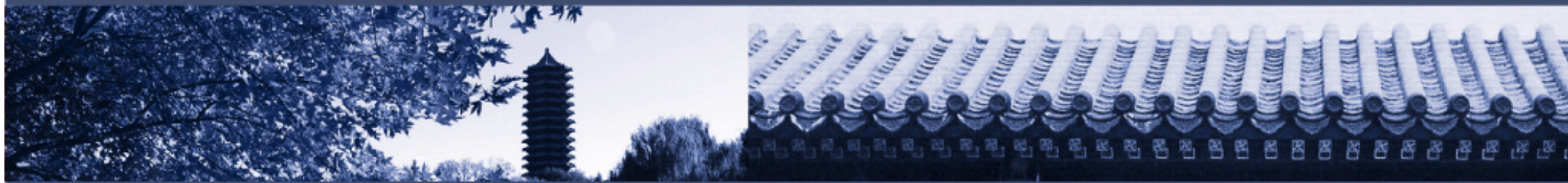
2018年春

程序设计实习 (I): C++程序设计

第二讲 类和对象 (1)

刘家瑛

liujiaying@pku.edu.cn



主要内容

□ 面向对象的基本概念

- 例子—矩形类
- 对象的内存分配 与 运算符
- 三种方式使用
- 引用 & 常引用

□ 构造函数

□ 复制构造函数

□ 析构函数



面向对象的程序设计

□ 面向对象的程序设计方法:

- 将某类客观事物共同特点(属性)归纳出来

- 形成一个数据结构(用多个变量描述事物的属性)

- 将这类事物所能进行的行为也归纳出来

- 形成一个个函数

- 这些函数可以用来操作数据结构

抽象



面向对象的程序设计

□ 通过某种语法形式,

将数据结构和操作该数据结构的函数 "捆绑"
在一起 → 形成一个 "类"

□ 使得数据结构和操作该数据结构的算法呈现出显而易见的紧密关系

封装



从客观事物抽象出类的例子

- 写一个程序, 输入矩形的长和宽, 输出面积和周长
 - 如对于 "矩形" 这种对象, 要用一个类来表示, 该如何做 "抽象" 呢?
 - 矩形的属性就是长和宽
 - 因此需要两个变量, 分别代表长和宽
 - 可以对矩形进行哪些操作
 - 矩形可以有设置长和宽, 计算面积和计算周长这三种行为
 - 这三种行为, 可以各用一个函数来实现, 都需要用到长和宽这两个变量



从客观事物抽象出类的例子

- 将长, 宽变量和设置长, 宽, 求面积, 以及求周长的三个函数封装在一起, 就能形成一个"矩形类"
 - 长和宽变量成为该"矩形类"的"成员变量"
 - 三个函数成为该类的"成员函数"
 - 成员变量和成员函数统称为类的成员
- 类 → 带函数的结构



从客观事物抽象出类的例子

```
class CRectangle
{
    public:
        int w, h;
        int Area() {
            return w * h;
        }
        int Perimeter() {
            return 2 * (w + h);
        }
        void Init(int w_, int h_) {
            w = w_; h = h_;
        }
}; //必须有分号
```



从客观事物抽象出类的例子

```
int main( ) {  
    int w, h;  
    CRectangle r; //r是CRectangle类的一个对象  
    cin >> w >> h;  
    r.Init(w, h);  
    cout << r.Area() << endl << r.Perimeter();  
    return 0;  
}
```

- 通过类, 可以定义变量
- 类定义出来的变量, 也称为类的实例, 即 "对象"
- C++中, 类的名字就是用户自定义的类型的名字
可以像使用基本类型那样来使用它
 - CRectangle 就是一种用户自定义的类型



对象的内存分配

- 和结构变量一样, **对象**所占用的**内存空间的大小**,
= 所有成员变量的大小之和
- 对于上面的**CRectangle**类, **sizeof(CRectangle) = 8**
- 每个对象各有自己的存储空间
- 一个对象的某个成员变量被改变了, 不会影响到另一个对象



对象间的运算

- 和结构变量一样, 对象之间可以用 "=" 进行赋值
- 不能用 "==" "!=" ">" "<" ">=" "<=" 进行比较, 除非这些运算符经过了 "重载"



如何使用类的成员变量和成员函数

用法1: 对象名.成员名

```
CRectangle r1, r2;
```

```
r1.w = 5;
```

```
r2.Init(3,4);
```

- **Init**函数作用在 **r2**上, 即**Init**函数执行期间访问的**w**和**h**是属于**r2**这个对象的, 执行**r2.Init**不会影响到**r1**

用法2: 指针->成员名

```
CRectangle r1, r2;
```

```
CRectangle * p1 = & r1;
```

```
CRectangle * p2 = & r2;
```

```
p1->w = 5;
```

```
p2->Init(3,4); //Init作用在p2指向的对象上
```



如何使用类的成员变量和成员函数

用法3: 引用名.成员名

```
CRectangle r2;  
CRectangle & rr = r2;  
rr.w = 5;  
rr.Init(3, 4); //rr的值变了, r2的值也变
```

```
void PrintRectangle(CRectangle & r)  
{  
    cout << r.Area() << ", " << r.Perimeter();  
}  
CRectangle r3;  
r3.Init(3, 4);  
PrintRectangle(r3);
```



引用的概念

类型名 & 引用名 = 某变量名;

- 定义了一个引用,并将其初始化为引用某个变量
- 某个变量的引用,和这个变量是一回事,相当于该变量的一个别名

```
int n = 4;  
int & r = n;  
r = 4;  
cout << r; //输出 4  
cout << n; //输出 4  
n = 5;  
cout << r; //输出 5
```

- 定义引用时一定要将其初始化成引用某个变量,不初始化编译不过
- 引用只能引用变量,不能引用常量和表达式



引用的作用

□ C语言中, 交换两个整型变量值的函数, 只能通过指针

```
void swap( int * a, int * b )  
{  
    int tmp;  
    tmp = * a; * a = * b; * b = tmp;  
}  
int n1, n2;  
swap(&n1, &n2) ;    // n1,n2的值被交换
```



引用的作用

□ 使用引用：

```
void swap( int & a, int & b )  
{  
    int tmp;  
    tmp = a; a = b; b = tmp;  
}  
int n1, n2;  
swap(n1, n2) ; // n1,n2的值被交换
```



引用作为函数的返回值

□ 函数的返回值可以是引用, 如:

```
#include <iostream>
using namespace std;
int n = 4;
int & SetValue() { return n; }
int main()
{
    SetValue() = 40;
    cout << n;
    return 0;
}
```

程序输出结果是 40



常引用

□ 定义引用时,在前面加 **const**关键字 -- 该引用即为"常引用"

```
int n;
```

```
const int & r = n;
```

不能通过常引用去修改其引用的内容

```
int n = 100;
```

```
const int & r = n;
```

```
r = 200; //编译出错,不能通过常引用修改其引用的内容
```

```
n = 300; //没问题,n的值变为300
```



常引用

- 请注意, **const T &** 和 **T &** 是不同的类型
- **T &**类型的引用或 **T**类型的变量可以用来初始化

const T &类型的引用

- **const T** 类型的常变量和**const T &**类型的引用则不能用来初始化**T &**类型的引用, 除非进行强制类型转换



类的成员函数的另一种写法

- 成员函数体和类的定义分开写

```
class CRectangle
{
    public:
        int w, h;
        int Area(); //成员函数仅在此处声明
        int Perimeter();
        void Init( int w_, int h_ );
};
```



类的成员函数的另一种写法

```
int CRectangle::Area() {  
    return w * h;  
}  
int CRectangle::Perimeter() {  
    return 2 * ( w + h );  
}  
void CRectangle::Init( int w_, int h_ )  
{  
    w = w_;    h = h_;  
}
```

CRectangle::说明后面的函数是CRectangle类的成员函数,而非普通函数

一定要通过对象或对象的指针或对象的引用才能调用



对象成员的访问权限

在类的定义中,

用下列权限关键字——说明对象成员的访问权限:

- **private**: 私有成员, 只能在成员函数内访问
- **public** : 公有成员, 可以在任何地方访问
- **protected**: 保护成员 [以后再说]

以上三种关键字出现的次数和先后次序都没有限制



对象成员的访问权限

- 定义一个类

```
class className {
```

```
    private:
```

私有属性和函数

```
    public:
```

公有属性和函数

```
    protected:
```

保护属性和函数

```
};
```

说明类成员
的可见性



对象成员的访问权限

- 如过某个成员前面没有上述关键字，
则缺省认为是私有成员

```
class Man {  
    int nAge;           //私有成员  
    char szName[20];    // 私有成员  
public:  
    void SetName(char * szName) {  
        strcpy(Man::szName, szName);  
    }  
};
```



对象成员的访问权限

- 在类的成员函数内部, 能够访问:
 - 当前对象的全部属性, 函数
 - 同类其它对象的全部属性, 函数
- 在类的成员函数以外的地方, 只能够访问该
类对象的公有成员



```
class CEmployee {  
    private:  
        char szName[30]; //名字  
    public:  
        int salary; //工资  
        void setName(char * name);  
        void getName(char * name);  
        void averageSalary(CEmployee e1,  
CEmployee e2);  
};  
void CEmployee::setName(char * name) {  
    strcpy( szName, name); //ok  
}  
void CEmployee::getName(char * name) {  
    strcpy( name, szName); //ok  
}
```



```
void CEmployee::averageSalary(CEmployee e1,  
CEmployee e2)  
{  
    salary = (e1.salary + e2.salary)/2;  
}  
int main()  
{  
    CEmployee e;  
    strcpy(e.szName, "Tom1234567889"); // 编译错,  
                                         // 不能访问私有成员  
    e.setName("Tom"); // ok  
    e.salary = 5000; //ok  
    return 0;  
}
```




```
int main() {  
    CEmployee e;  
    strcpy(e.szName, "Tom1234567889"); // 编译错, 不能访问私有成员  
    e.setName( "Tom"); // ok  
    e.salary = 5000;    // ok  
    return 0;  
}
```

□ 设置私有成员的目的

- ① 强制对成员变量的访问一定要通过成员函数进行
- ② 方便修改成员变量的类型等属性 → 只需更改成员函数

Vs. 否则, 所有直接访问成员变量的语句都需要修改

□ 设置私有成员的机制叫隐藏



□ 例如,如将上述程序 → 内存空间紧张的手持设备上
将 `szName` 改为 `char szName[5]`,
若 `szName` 不是私有,那么就要找出所有类似

```
strcpy(man1.szName, "Tom1234567889");
```

这样的语句进行修改,以防止数组越界 → 太麻烦!

□ 如果将 `szName` 变为私有,

那么程序中就不会出现 (除非在类的内部)

```
strcpy(man1.szName, "Tom1234567889");
```



所有对**szName**的访问都是通过成员函数来进行,

例如:

```
man1.setName( "Tom12345678909887" );
```

□ 如需将**szName**改短,也不需将上面的语句找出来修改,
只要改setName成员函数,在里面确保不越界即可

