类和对象

- 我们编写程序的目的是为了解决现实中的问题，而这些问题的构成都是由各种事物组成，我们在计算机中要解决这种问题，首先要做就是要将这个问题的参与者：事和物抽象到计算机程序中，也就是用程序语言表示现实的事物。
- 那么现在问题是如何用程序语言来表示现实事物？现实世界的事物所具有的共性就是每个事物都具有自身的属性，一些自身具有的行为，所以如果我们能把事物的属性和行为表示出来，那么就可以抽象出来这个事物。
- 比如，一个Circle类，属性包括半径r，以及方法getC()、getS()，获取周长和面积。
- 封装
  - 把变量（属性）和函数（操作）合成一个整体，封装在一个类中
  - 对变量和函数进行访问控制

class和struct区别

- struct默认成员权限public
- class默认成员权限private

```
Fruits apple;
apple.price = 6;
apple.print();

Fruits banana;
banana.price = 8;
banana.print();
```

```
    Fruits apple;
    apple.price = 6;
0082189F  mov         dword ptr [apple],6
    apple.print();
008218A6  lea         ecx,[apple]
008218A9  call        Fruits::print (08211DBh)

    Fruits banana;
    banana.price = 8;
008218AE  mov         dword ptr [banana],8
    banana.print();
008218B5  lea         ecx,[banana]
008218B8  call        Fruits::print (08211DBh)
```

```
cout << sizeof(apple) << endl;
cout << sizeof(Fruits) << endl;
```

```
Microsoft Visual Studio 调试控制台
4
4
```

黑猫编程 blackcat1995.com 访问权限

```
class Circle {
private:
    double m_r;
public:
    void setR(double r) {
        m_r = r;
    }

    double getC() {
        return 2 * PI * m_r;
    }

    double getS() {
        return PI * m_r * m_r;
    }
};
```

- 访问权限
  - 在类的内部(作用域范围内)，没有访问权限之分，所有成员可以相互访问
  - 在类的外部(作用域范围外)，访问权限才有意义：public，private，protected
  - 在类的外部，只有public修饰的成员才能被访问，在没有涉及继承与派生时，private和protected是同等级的，外部不允许访问

黑猫编程 blackcat1995.com 构造函数和析构函数

- 构造函数函数名和类名相同，没有返回值，不能有void，但可以有参数。ClassName(){}
- 析构函数函数名是在类名前面加"~"组成,没有返回值，不能有void,不能有参数，不能重载。~ClassName(){}

```
private:
    char* m_name;
    int m_tall;
    int m_money;
```

```
public:
    Person() {
        cout << "构造函数开始调用" << endl;
        m_name = new char[20];
        strcpy(m_name, "cat");
        m_tall = 180;
        m_money = 100;
    }

    ~Person() {
        if (m_name != NULL) {
            delete[] m_name;
            m_name = NULL;
        }
        cout << "析构函数调用" << endl;
    }
```


构造函数分类

- 参数类型：分为无参构造函数和有参构造函数
- 类型分类：普通构造函数和拷贝构造函数

```
public:
    // 无参数构造函数
    Person() {
        cout << "无参数构造函数" << endl;
        m_name = "cat";
        m_age = 26;
    }

    // 有参数构造函数
    Person(string name, int age) {
        cout << "有参数构造函数" << endl;
        m_name = name;
        m_age = age;
    }
```

```
// 拷贝构造函数 使用另一个对象初始化本对象
Person(const Person& person) {
    cout << "拷贝构造函数" << endl;
    m_name = person.m_name;
    m_age = person.m_age;
}
```

列表初始化

```
class Person {
private:
    string m_name;
    int m_age;

public:

    Person(string name, int age) : m_name(name), m_age(age) {}

    void print() {
        cout << m_name << " " << m_age << endl;
    }
};
```