



面向对象习题课

2022-10-2 南工骁鹰视觉组



知识回顾

理解类最好的方法就是把一个类当成真实世界中的一类物品, 然后不断做类比



知识回顾

理解类最好的方法就是把一个类当成真实世界中的一类物品, 然后不断做类比

以“狗”为例, 假设我们只关注这条狗的体重/名字/年龄

狗:

- unsigned int 体重
- string 名字
- unsigned int 年龄

知识回顾

理解类最好的方法就是把一个类当成真实世界中的一类物品, 然后不断做类比
以“狗”为例, 假设我们只关注这条狗的体重/名字/年龄

狗:

- unsigned int 体重
- string 名字
- unsigned int 年龄

```
5
6
7  class Dog // 类的名称首字母大写比较好
8  {
9  public:
10     unsigned int weight;
11     std::string name;
12     unsigned int age;
13 };
14
15
```



知识回顾

接下来,我们希望我们的狗能发出点声音,并且还能做个自我介绍(假设我们的狗具备这样的智能)

知识回顾

接下来,我们希望我们的狗能发出点声音,并且还能做个自我介绍(假设我们的狗具备这样的智能)

```
6
7
8  class Dog // 类的名称首字母大写比较好
9  {
10 public:
11     unsigned int weight;
12     std::string name;
13     unsigned int age;
14
15     void bark() // 🐶叫
16     {
17         cout << "my name is " << name
18             << ", woof! woof!" << endl;
19     }
20 };
21
22
```

知识回顾

但是我们只是像一个协议一样说明了一条狗应该具备什么素质

我们需要**创建**一条活蹦乱跳的狗

```
6
7
8  class Dog // 类的名称首字母大写比较好
9  {
10 public:
11     unsigned int weight;
12     std::string name;
13     unsigned int age;
14
15     void bark() // 🐶叫
16     {
17         cout << "my name is " << name
18             << ", woof! woof!" << endl;
19     }
20 };
21
22
```



知识回顾

但是我们只是像一个协议一样说明了一条狗应该具备什么素质

我们需要创建一条活蹦乱跳的狗(实例化)

类比:

```
int main()
{
    int x = 0;
    Dog dog = {12, "Poly", 12};
}
```




知识回顾

区别: C的结构体 vs C++的类

C结构体: 所见即所得

C++的类: 多了很多额外的功能, 成员函数

要用一种思想: 类的成员构成了一个对象, 而不是一堆毫无意义绑定在一起的数据

成员函数是类里面十分强大的组成, 没有成员函数就没有类存在的意义



知识回顾

区别: C的结构体 vs C++的类


C结构体: 所见即所得

C++的类: 多了很多额外的功能, 成员函数

要用一种思想: 类的成员构成了一个对象, 而不是一堆毫无意义绑定在一起的数据

成员函数是类里面十分强大的组成, 没有成员函数就没有类存在的意义

我们回顾一下如果不使用类怎么实现刚才的方法




练习题1 (7 min)

构造一个三角形的类Triangle

类里面有三条边长的数据

以及一个成员方法IsValid(), 如果三条边可以构成三角形, 返回true, 否则false, 然后在main函数内验证自己的类是否工作正常




练习题1 (7 min)

计算三角形面积Area() [海伦公式] $s = (a+b+c)/2$

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$$

同样，使用main验证是否正确



练习题1 (7 min)

构造一个三角形的类Triangle

类里面有三条边长的数据

1. 成员方法IsValid(), 如果三条边可以构成三角形, 返回true, 否则false
2. 计算三角形面积Area() [海伦公式] $s = (a+b+c)/2$
3. 使用main验证是否正确

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$$



知识回顾2

- 每个类的实例构造的时候可能都需要进行重复的劳动

→ 构造函数

析构函数：成员超出作用域的时候需要执行的操作（释放资源，处理后事[划掉]）

构造函数和析构函数都不需要注明函数类型



知识回顾2

简单的案例：检查三角形是否有效，如果无效，直接生成一个边长为1的三角形

知识回顾2

简单的案例：检查三角形是否有效，如果无效，直接生成一个边长为1的三角形

```
class Triangle
{
public:
    double a, b, c;
    bool IsValid()
    {
        if ((a + b > c) && (a + c > b) && (b + c > a))
        {
            return true;
        }
        else return false;
    }

    Triangle(double _a, double _b, double _c)
    {
        a = _a;
        b = _b;
        c = _c;


        if (!IsValid()) // 可以调用成员函数
        {
            a = b = c = 1;
        }
    }
};
```




知识回顾2

动态内存分配，C++的臭名昭著的new和delete/delete[]

```
int main()
{
    Dog* d = new Dog();
    d->name = "Hellen";
    d->age = 12;
    d->bark();
    delete d;
}
```



练习题3 (15 min)

一个有意思的(奇怪的)练习题

步兵击杀升级

机器人击杀获得经验，然后被击杀的机器人被delete，又再次被new出来，再次被杀，一直到某个机器人升到3级

给出了main函数，自己来实现类，使代码能够运行，让master连续击杀敌方机器人



静态成员

使用了static关键字

函数中static → 不会随着每次调用而重新生成

类里的static → 不会随着每次实例化而重新赋值



静态成员

使用了static关键字

函数中static → 不会随着每次调用而重新生成

类里的static → 不会随着每次实例化而重新赋值


统计一些与类相关，但是是存储在每个对象身上的特性



静态方法

不是作用在每个对象身上的，而是像工具一样能够“客观地”完成任务


例子： `Dog::Compare` / `Dog::CompareTo()`



练习题4 (15 min)

实现一个自己的`std::vector<int>`

当然... 简化版本




练习题4 (15 min)

实现一个自己的`std::vector<int>`

当然... 简化版本

首先我们要看一下vector的使用的效果是如何的



练习题4 (15 min)

实现一个自己的 `std::vector<int>`

- 实现 `push_back`, 并且能在其中 检测并分配内存
- 实现 `pop_back`, 返回最后一个 值并且将其 删除
- 构造函数
- 析构函数
- 内部计数器, 记录内部大小
- `capacity()` / `size()`
- `at`方法



继承/多态性

继承的语法

```
class A : public BaseClass
{
    // blahblah
}
```