第二章 类和对象

written by SCNU第一帅----->孔文康

什么是面向对象设计

对象

面向对象编程，重点显然是对象。

当然，这个对象，并不是你的对象那个对象。

用松本行弘在《代码的未来》中一句话来说：

所谓对象，就是抽象化的数据本身。

什么意思呢？可以这样理解。

用现实中的行为来解析，可以表示为属性+行为

用程序语言来解析，可以表示为数据(变量)+操作(函数)

就拿吃饭这件事情来说，可以分为面向过程吃饭和面向对象吃饭。自己亲自下厨就是面向过程，点外卖就是面向对象。

在面向过程吃饭中，我们想要吃饱，需要自己下厨把这顿饭做出来，那么我们需要先想好吃什么、然后去买菜、洗菜、洗米、煮饭、炒菜等一系列事情。

public void 做饭{制定菜单;买菜;洗菜;洗米;蒸饭;炒菜;}

那么，你说和点外卖相比，自己做饭有哪些缺点啊？
那还用说，麻烦呗。

的确，面向过程编程也一样，由于想要完成做饭这件事，需要自己定义很多个方法。除此之外，还有很多遇到很多其他问题，比如：

我不想吃米饭，我想吃馒头。
上次买的菜家里面还有，不需要去买菜。
中午吃剩下的菜家里面还有，直接热一热就可以吃了。
这次去的一家超市提供洗菜服务，不需要我们自己洗菜了。

以上这些突发事件，在编程中就叫做需求变更或者新的需求，这种事情发生是必然会发生的。

那么，有新的需求了怎么办，上面这种自己动手做饭的场景，就只能重新拼凑

对于程序员来说，就需要通读代码，找出可以复用的方法，然后重新调用，不能复用的就重新写一个。时间久了，方法就会越来越多，系统维护越来越复杂。

面向对象，其实就是我们通过点外卖的方式来“做饭”。

我们知道我们需要一顿饭，我们只需要打开外卖软件，在里面选择我们需要的菜品然后下单就可以了。我们不关心饭店做饭的过程。想吃什么点什么，家里来人了就再下一个订单，不想吃米饭了，想吃馒头了，也可以给饭店打电话，让他们把米饭换成馒头。

外卖软件.点餐（红烧肉，糖醋鱼，可乐一瓶）.送达时间（一小时后）.备注（可乐加冰）所以，通过面向对象的方式“做饭”，就像上面的代码一样。

哦，我明白了，面向对象就是把本来可能需要自己做的事情交给别人来做？对于我来说，外卖软件就是个对象，我再面向他“做饭”，其实是他帮我做的。

你说的也对，也不对。在这个场景中，确实可以把外卖软件当做是「对象」。其实，在面向对象编程中，抛弃了函数，想要实现一个功能不再是通过函数的叠加调用实现的了。而是通过对象。

对象就是对事物的一种抽象描述。现实世界中的事物，都可以用「数据」和「能力」来描述。

比如我要描述一个人，「数据」就是他的年龄、性别、身高体重，「能力」就是他能做什么工作，承担什么样的责任。

描述一个外卖软件，「数据」就是他包含的菜品，而「能力」就是他可以点菜。

消息

一个对象与另一个对象之间的交互称之为消息。对象之间的联系只能通过消息传递来进行。

请注意

面向对象思想有三大要素：封装、继承和多态。

以你是土豪作为例子。

1、封装

不管你中午吃的是窝头还是鲍鱼，你的食物都在你肚皮里，别人看不到你中午吃了什么，除非你自己说给他们听（或者画给他们看，whatever)

2、继承

你个豪，你们全家都是豪。冰冻三尺非一日之寒，你有今天，必定可以从你爸爸爷爷那里追根溯源。

3、多态

哲学家说过，世上不会有两个一模一样的双胞胎。即使你从你父亲那里继承来的土豪气质，也不可能完全是从一个模子里刻出来的，总会有些差别。比如你爸喜欢蹲在门前吃面，你喜欢骑在村口的歪脖子树上吃，或者反过来。

4、抽象

抽象就是对数据共性的归纳和总结。

例如把中国人、日本人、韩国人等所有国籍的人归纳为人，这就是一种抽象。

类是对象的抽象，而对象是类的特例，或者说是类的具体表现。

类的声明和对象的定义

一、声明类的类型

如果我们程序中要用到类，必须自己根据需要进行类型声明，或者使用别人已建立的类。

C++中怎样声明一个类的类型呢？

其方法和声明一个结构差不多，下面是声明结构体类型的方法。

```
// 声明一个名为Student的结构体类型
struct Student {
 int sum;
 char name[20];
 char sex;
};
//定义了两个结构体变量stu1,stu2
Student stu1, stu2;
```

上面声明了一个名为Student的结构体类型并定义了两个结构体变量stu1和stu2。

可以看到它只包括数据(变量)，没有包括操作。

现在我们声明一个类。

```
//以class开头，类名为Student
class Student {
 int num;
 char name[20];
 char sex;
 // 以上三行是数据成员
 // 下面是成员函数
 void participate() {
 // 下面是函数中的操作语句
  cout<<"num:"<<num<<endl;
  cout<<"name:"<<name<<endl;
  cout<<"sex:"<<sex<<endl;
  }
}
// 定义了两个Student类的对象stu1和stu2
Student stu1, stu2;
```

从上面我们可以看到，类体中除了有变量，还有函数。

这就体现了把数据和操作封装在一起。

public和private

怎么体现类的封装性和安全性呢?

现在这个声明仅仅把变量和函数隐蔽起来，但是没有防止无关的外界调用。

在C++中，有成员访问限定符public和private。

用他们来声明各成员的访问属性。

被声明为private的成员，只能被本类中的成员函数调用，类外不能引用（友元类除外）。

被声明为public的成员，既可以被本类中的成员函数所引用，也可以被类的作用域内的其他函数引用。

```
//以class开头，类名为Student
class Student {
 // 声明以下部分是私有的
 private:
 int num;
 char name[20];
 char sex;
 // 以上三行是数据成员
 // 下面是成员函数
  // 声明以下部分是公有的
  public:
  void participate() {
  // 下面是函数中的操作语句
  cout<<"num:"<<num<<endl;
  cout<<"name:"<<name<<endl;
  cout<<"sex:"<<sex<<endl;
  }
}
```

请注意

如果类的定义中既不指定private，也不指定public，则系统默认为私有。

除了private和public外，还有一种成员访问限定符protected。

被声明为protected的成员，它不能被类外访问，但可以被派生类的成员函数访问。

二、定义对象方法

1、先声明再定义

上述例子。

2、声明的同时定义

```
//以class开头，类名为Student
class Student {
 // 声明以下部分是私有的
 private:
 int num;
 char name[20];
 char sex;
 // 以上三行是数据成员
 // 下面是成员函数
  // 声明以下部分是公有的
  public:
 void participate() {
 // 下面是函数中的操作语句
 cout<<"num:"<<num<<endl;
 cout<<"name:"<<name<<endl;
 cout<<"sex:"<<sex<<endl;
 }
}stu1,stu2; //定义了两个Student类的对象
```

3、不出现类名直接定义

```
class {
 // 声明以下部分是私有的
 private:
 int num;
```

```
 char name[20];
 char sex;
 // 以上三行是数据成员
 // 下面是成员函数
 // 声明以下部分是公有的
  public:
  void participate() {
  // 下面是函数中的操作语句
  cout<<"num:"<<num<<endl;
  cout<<"name:"<<name<<endl;
  cout<<"sex:"<<sex<<endl;
  }
}stu1,stu2; //定义了两个无类名类对象
```

4、结构体声明类

```
//以struct开头
struct Student {
 // 声明以下部分是私有的
 private:
 int num;
 char name[20];
 char sex;
 // 以上三行是数据成员
 // 下面是成员函数
  // 声明以下部分是公有的
  public:
  void participate() {
  // 下面是函数中的操作语句
  cout<<"num:"<<num<<endl;
  cout<<"name:"<<name<<endl;
  cout<<"sex:"<<sex<<endl;
  }
}
Student stu1,stu2;
```

请注意

用struct声明的类，如果对其成员不作private和public声明，系统默认为public。

而用class声明的类，如果对其成员不作private和public声明，系统默认为private。

三、类的成员函数

一般的做法是将需要被外界调用的成员函数指定为public,他们是类的对外接口。不需要被调用的，指定为private。

1、在类外定义成员函数

类的成员函数既可以在类中定义，也可以在类外定义。

```
class Student {
 // 声明以下部分是私有的
 private:
 int num;
 char name[20];
 char sex;
 // 以上三行是数据成员
 // 下面是成员函数
 // 声明以下部分是公有的
  public:
  void participate();
 }
void Student::participate() {
  cout<<"num:"<<num<<endl;
  cout<<"name:"<<name<<endl;
  cout<<"sex:"<<sex<<endl;
 }
```

请注意

如果在作用域运算：：的前面没有类名，或者函数名前面既无类名也没有作用域运算符如

```
::participate()
participate()
```

则表示participate()函数不属于任何类，这个函数不是成员函数，而是全局函数，即一般函数。

类函数必须现在类体作原型声明，然后才能在类外定义。

2、内置成员函数

C++中，在类体定义的函数规模一般很小，而系统调用函数的过程所花费的时间开销相对是比较大的。调用一个函数的时间远远大于小规模函数体中全部语句的执行时间。

为了减少时间开销，如果在类体中定义的成员函数中不包括循环等控制结构，C++系统会自动地对它们作为内置函数处理。

也就是说，在程序调用这些成员函数时，并不是真正地执行函数的调用过程（如保留返回地址等处理），而是把函数代码嵌入程序的调用点。这样大大减少调用成员函数的时间开销。

请注意

如果成员函数不在类体内定义，而是在类体外定义，系统并不把他们默认为内置函数，调用这些函数的过程跟调用一般函数是相同的。如果想把这些函数指定为内置函数，应当作inline显式声明。

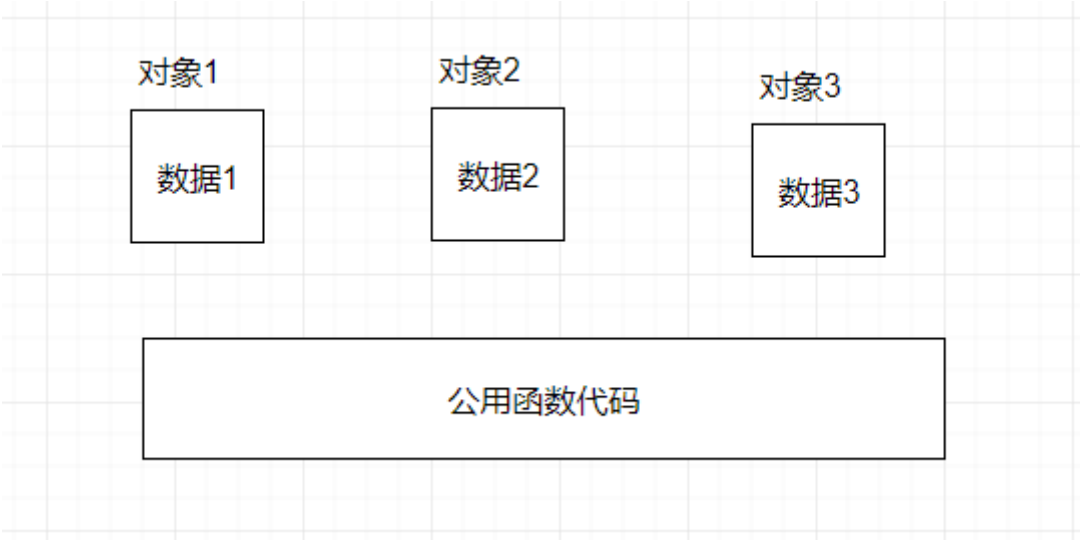
```
class Student {
```

```
// 声明以下部分是私有的
private:
int num;
char name[20];
char sex;
// 以上三行是数据成员
// 下面是成员函数
// 声明以下部分是公有的
 public:
 inline void participate();
}
inline void Student::participate() {
 cout<<"num:"<<num<<endl;
 cout<<"name:"<<name<<endl;
 cout<<"sex:"<<sex<<endl;
}
```

3、成员函数的存储方式

同一类的不同对象中的数据成员的值一般是不相同的，而不同对象的函数代码是相同的，不论调用哪一个对象的函数代码，其实调用的都是同样内容的代码。

存储结构如下图所示



因此，每个对象所占用的存储空间只是该对象的数据成员所占用的存储空间，而不包括函数代码所占用的存储空间，如果声明了如下一个类。

```
class Time {
 int hour;
 int minute;
 int sec;
 void setTime () {
 cin>>hour>>minute>>sec;
 }
}

cout<<sizeof(Time)<<endl;
```

这个值输出来是12。这就证明了一个对象所占的空间大小只取决于该对象中的数据成员所占的空间，而与成员函数无关。

函数的目标代码是存储在对象空间之外的。

那么问题来了

不同的对象使用的是同一个函数代码片段，它怎么能够分辨不同对象中的数据进行操作呢?

C++为此专门设立了一个名为this的指针，用来指向不同的对象，当调用对象为stu1时指向stu1，当调用对象stu2时，指向stu2。

请注意

- 不论成员函数在类内定义还是在类外定义，成员函数的代码段存储方式是相同的，都不占用对象的存储空间。

- 不要将成员函数的这种存储方式和inline函数的概念混淆。
- 不论是否使用inline函数声明，成员函数的代码都不占用对象的存储空间
- inline函数只影响程序的执行效率，而与成员函数是否占用对象的存储空间无关。
- 虽然成员函数没有放在对象的存储空间中，但从逻辑的角度来说，成员函数是和数据一起封装在一个对象中的，只允许本对象中的成员函数访问同一对象中的私密数据。

四、对象成员的引用

在程序中经常需要访问对象中的成员。可以有以下3种方法

- 通过对象名和成员运算符访问对象中的成员
- 通过指向对象的指针访问对象中的成员
- 通过对象的引用访问对象中的成员

1、通过对象名和成员运算符访问

例如在程序中可以写出以下语句：

```
stu1.num = 100; //假设num已定义为公用数据成员
```

表示将整数100赋值给对象stu1中的数据成员num。

其中 . 是成员运算符，用来对成员进行限定，指明访问的是哪一个对象中的成员。

对象名.成员名

不仅可以在类外引用对象的公用数据成员，而且还可以调用对象的公有成员函数，但同样必须指出对象名，如

```
stu1.participate(); //调用对象stu1的公用成员函数
```

请注意

- 在类外只能调用公用的成员变量和成员函数。
- 一个类中应当最少有一个公用成员函数，作为对外的接口，否则无法对对象进行任何操作。

2、通过指向对象的指针访问对象中的成员

```
class Time {
 int hour;
 int minute;
 int sec;
 void setTime () {
 cin>>hour>>minute>>sec;
 }
}
Time t, *p; // 定义对象t和指针变量p
 p = &t; // 使p指向对象t
 cout<<p->hour; //输出p指向的对象中的成员hour
```

p->hour表示p当前的指向对象t中的成员hour。
(*p).hour也是对象t中的成员hour，因为(*p)就是对象t

在p指向t的前提下
p->hour，(*p).hour，t.hour三者相等

3、通过对象的引用来访问对象中的成员

```
class Time {
 int hour;
 int minute;
 int sec;
 void setTime () {
 cin>>hour>>minute>>sec;
 }
}
Time t1; //定义对象t1
 Time &t2 = t1; //定义Time类引用t2，并使之初始化为t1
 cout<<t2.hour; //输入对象t1中的成员hour
```

由于t2和t1占用同一段存储单元（t2是t1的别名）

因此 t2.hour等于t1.hour。

五、类的封装性和信息隐蔽

1、公用接口与私有实现的分离

在声明了一个类后，用户的主要工作就是通过调用其公用的成员函数来实现类提供的功能（对数据成员设置值，显示数据成员值，对数据成员加工）。

这些功能用户可以使用他们，而不能改变他们。

用户不需要知道具体细节，只知道调用哪个函数，会能到什么结果。

一切于用户操作无关的部分都封装好。

这就是接口与实现分离。

通过成员函数对数据成员进行操作称为类的功能的实现。

类中被操作的数据是私有的，类的功能实现细节对用户是隐蔽的，这种实现成为私有实现。

这种类的公用接口与私有实现的分离形成了信息隐蔽。

软件工程的一个最基本的原则是将接口与实现分离。他的好处有

- 如果想修改或扩充类的功能，只需要修改该类中有关的数据成员和与他有关的成员函数，程序中类以外的部分不用修改。
- 当接口与实现分离时，只要类的接口没有改变，对私有实现的修改不会引起程序的其他部分的修改。
- 如果在编译时发现类中的数据读写有错，不必检查整个程序，只需要检查本类中访问这些数据的成员函数。

2、类声明和成员函数定义的分离

如果一个类只被一个程序使用，那么类的声明和成员函数的定义可以直接写在程序的开头，但是如果一个类被多个程序使用，这样做的重复工作太大了，效率太低。

这时候往往把类的声明放在指定的头文件中，用户如果想使用该类，只要把有关的头文件包含进来即可。

因此可以认为类声明头文件是用户使用类库的公用接口。

为了 实现信息隐蔽，不让用户看到函数执行细节，对类成员函数的定义一般不和类的声明放在头文件中，而放在另外一个文件中。

请注意

- 在系统提供的头文件中只包括对成员函数的声明，而不包括成员函数的定义。类声明和函数定义是分别放在两个文件中的。
- 实际上C++程序由3个部分组成， 1、类声明头文件（后缀为.h或无后缀），2、类实现文件（后缀为.cpp），包括类成员函数定义，3、类的使用文件（后缀为.cpp），即为主文件

例如，可以先分别写两个文件：

（1）类声明头文件

```
// student.h 这是名为student.h的头文件，此文件进行类的声明
#include<string>
using namespace std;
class Student {
 public:
 void show_data(); //公用函数原型声明
 private:
 int num;
 string name;
  char sex;
};
```

（2）包含类成员函数的定义文件（实现文件）

```
// student.cpp （此文件进行函数定义）
#include<iostream>
#include "student.h" // 引入类声明
// 类外定义函数
void Student::show_data() {
 cout<<"num:"<<num<<endl;
 cout<<"name:"<<name<<endl;
```

```
 cout<<"sex:"<<sex<<endl;
}
```

（3）主文件

```
// main.cpp (主函数模块)
#include<iostream>
#include "student.h" //将类声明头文件包含进来
using namespace std;
int main () {
 Student stu; //定义对象stu
 stu.show_data(); //执行stu对象的show_data函数
 return 0;
}
```