第5章 面向对象程序设计

导学

面向对象程序设计(Object Oriented Programming,OOP)是一种将数据和对数据的操作封装成为“对象”来处理的程序设计方法。这种设计思想使得软件设计更加灵活,提高了代码的可读性和可扩展性。OOP可以在对象的基础上进行再抽象,根据不同对象之间的共同特征进行分类、抽象,形成类。OOP的关键就在于如何合理地定义和组织类以及类与类之间的关系。

了解:面向对象的程序设计思想。

掌握:对象、类的概念;面向对象程序设计的封装、继承和多态。

前面的章节介绍了语法、数据类型和程序的控制结构等内容。本章将主要介绍Python中另一个非常核心的概念,即对象。和C++、Java等其他语言一样,Python被称为面向对象的语言。在对象之上,可以抽象出类的概念,类具有封装性、继承性和多态性。

5.1 面向对象程序设计基础

5.1.1 面向对象程序设计的基本概念

1. 对象

对象(Object)是面向对象程序设计的核心,是程序的主要组成部分。在面向对象的程序设计方法中,一个程序可以看作一组对象的集合。

在现实世界中,我们可以广义地认为,对象就是客观世界里存在着的任何事物。它可以是有生命的,也可以是无生命的;可以是具体的,也可以是抽象的。比如,我们每一个人、我们身边的花草树木、桌椅板凳,都可以称为一个对象。任何对象都不仅包含它本身,同时还包含其所具有的特性和行为。

在面向对象的程序设计思想中,对象可以被看成是数据以及其具有的属性和存取、操作该数据的方法所构成的集合。所以,在这种设计方法中,只要将程序中包含的每一个对象设计完成,也就完成了对整个程序的设计。

2. 类

类(Class)是同一类型对象,也就是具有相似特征或者行为的对象的集合和抽象。在面向对象程序设计中,对象是程序的基本单位,是一种复合的数据类型,是程序的基本要素。而类是将具有相同状态、行为和访问机制的多个对象抽象形成一个整体。简言之,类是对象的集合,对象是类的实例。在定义了一个类之后,符合类特点的对象称为类实例或类对象。

类代表一般，而类中的一个对象代表具体。例如：如果将每一个人看成一个对象，那么一类人群就可以称为一个“类”，比如，男人是一类，女人是一类。

其实，不必将类的概念想象得那么复杂难以理解，我们可以把类简单地理解为类型，即数据类型。就像 Python 等语言允许用户创建除自带函数之外的自定义函数一样，Python 也允许用户创建自定义的满足特殊条件的数据类型，与 Python 定义好的 string、list、dict 等数据类型一样。

3. 属性和方法

属性(Properties)是类的特征的描述，也就是类中的对象所具有的一致的数据结构。在实际编程过程中，属性就是定义在类中的变量。

方法(Method)是类的行为的总称，是允许作用在对象上的所有操作。在实际编程过程中，方法可以理解为定义在类中的函数。

5.1.2 面向对象程序设计的 3 个基本特性

面向对象程序设计是一种计算机编程架构。面向对象程序设计方法具有以下 3 个基本特性。

1. 封装性

封装性(Encapsulation)就是将数据和数据的属性及对其可能进行的操作集合起来，形成一个整体，即对象。一方面，用户不必知道对象行为的实现细节，只需根据对象提供的外部特性接口对对象进行访问。这样，就可以实现将对象的用户和设计者分开，用户在使用对象时，不必知道对象行为的细节，只需调用设计者提供的协议命令去做即可。另一方面，面向对象方法的封装性有效地使对象以外的事物不能随意获取对象的内部属性(公有属性除外)，避免了外部错误对其产生的影响，大幅减少软件开发过程中查错的工作量，降低排错难度；同时隐蔽了程序设计的复杂性，提高了代码的重用性。

2. 继承性

提到继承性(Inheritance)，就要先给出两个在类的继承过程中出现的新概念：父类(基类)和子类(派生类)。父类和子类是相对而言的，如果我们将已定义好的一个类称为父类，则从这个类派生出来的类就被称为这个类的子类。子类可以继承父类的所有属性。这种从父类派生出子类的现象称为类的继承机制，也就是继承性。

子类因为继承了父类的属性，所以在使用时无须重新定义父类中已经定义好的属性和行为，而是自动地拥有其父类的全部属性和行为。

3. 多态性

面向对象程序设计的多态性(Polymorphism)是指子类在继承了父类中定义的属性或行为的基础上，可以再定义出其他个性化的属性或行为。

Python 完全采用了 OOP 的思想，是真正面向对象的高级动态编程语言，完全支持面向对象诸如封装、继承、多态以及派生类对基类方法的覆盖和重写等基本功能。

与其他面向对象程序设计语言不同的是，Python 中对象的概念更加宽泛。在 Python 中，一切内容都可以称为对象，包括字符串、列表、字典和元组等内置数据类型都具有和类完全相似的语法和用法。也可以说，为了保持语句简约的风格，Python 是在尽可能不增加新语法和语义的情况下加入了类机制。

5.2 定义和访问类与对象

5.2.1 定义类和对象

对象是类的实例，是由类创建的。所以在定义一个对象时，应该首先存在一个类。下面我们就来分别定义类和对象。

1. 定义类

创建类时，对象的属性用变量形式表现，称为数据成员或成员变量；对象的方法用函数形式表示，称为成员函数或成员方法。成员变量和成员方法统称为类的成员。

Python 使用 class 关键字来定义类，具体的书写格式是这样的：class 关键字之后是一个空格，在空格的后面是用户定义的类名以及一个冒号，然后换行开始定义类的内部实现方法。注意，类名的首字母一般需要大写，虽然也可以按照自己的习惯风格来定义类名，但是一般还是推荐参考惯例来命名，并在整个系统的设计和实现中保持风格的一致，以此增强程序的可读性，也有利于团队合作。

下面给出定义类的最简单的格式：

```
class 类名:
    属性(成员变量)
    属性
    …
    …
    成员函数(成员方法)
```

【例 5-1】 定义一个类，取名为 SWMU，实现输出字符串“Welcome to Southwest Medical University!”的作用。程序代码如下：

```
class SWMU:
    #定义成员变量
    xm = "Southwest Medical University"
    #定义成员函数 Welcome()
    def Welcome(self):
        print("Welcome to " + SWMU.xm + "!")
```

例 5-1 在 CMU 类中定义一个成员变量 xm，变量的值为字符串“Southwest Medical University”和一个成员函数 Welcome (self)，在成员函数中调用成员变量 xm，并实现输出字符串“Welcome to Southwest Medical University!”的目的。

需要说明的是，在 Python 中，函数和成员函数是有区别的。成员函数一般指与特定实例绑定的函数，当用户通过对象调用成员函数时，对象本身将作为第一个参数被传递过去，这是普通函数所不具备的特点。如例 5-1 中成员函数 Welcome (self)中的参数 self 是代表类对象本身的参数。在类的成员函数中访问实例属性时，需要以 self 为前缀。

本例只实现了定义一个类的作用，没有具体的类实例，也就是对象，所以程序没有运行结果。

2. 定义对象

前面介绍过，类就是数据类型。那么，面向对象程序设计方法中，在类的基础上定义的变量就是“类对象”“实例对象”“对象”“实例”，这些各种称呼来源于不同的程序设计语言，其中“类对象”和“实例对象”比较严格些，建议大家多采用。

既然对象是类的实例，那么接着上面的例子，我们就可以创建一个基于类“SWMU”的对象了。在定义了具体的对象之后，即可通过使用“对象名.成员”的方式来访问类的成员函数和成员变量。

首先给出 Python 创建对象的语法格式：

```
对象名 = 类名()
```

然后在例 5-1 的基础上，进一步定义 SWMU 类的一个对象 xm，并通过类对象来调用成员函数以实现打印欢迎词的功能，具体代码如下：

```
class SWMU:
    #定义成员变量
    xm = "Southwest Medical University"
    #定义成员函数 Welcome()
    def Welcome(self):
        print("Welcome to " + SWMU.xm + "!")
xm = SWMU()                                  #定义类对象 xm
xm.Welcome ()                                #通过类对象调用成员函数
print(xm.xm)                                 #通过类对象调用成员变量
```

程序运行结果如下：

```
Welcome to Southwest Medical University!
Southwest Medical University
```

5.2.2 利用构造函数定义类

定义类时，可以使用一些特殊的方法，如__ init__() 和__new __()，它们被称为构造函数或者构造器。这里，我们只为大家介绍最常用的构造函数__ init__()。大家需要注意构造函数的写法，构造函数分别以两个下画线“__”作为开头和结尾。

当定义一个类时，系统会自动建立一个没有任何操作的默认的__ init__()方法。此时，如果用户自己建立了一个新的特殊的__ init__()方法，那么系统给出的默认__ init__()方法将被用户自定义的__ init__()方法所覆盖。

实际上，调用类时，传递的任何参数都是赋给__ init__()方法的，可以理解为创建实例时，对类的调用实际上就是对构造器方法的调用。通过构造器调用可以更新类的数据属性。具体情况可以在例 5-2 中体现出来。

【例 5-2】 使用构造器方法__ init__()。程序代码如下：

```
class my_class:
    x = 100
    y = 200
#使用构造函数为类定义除类对象本身之外的另外两个参数：a、b
```

```
    def __init__(self,a,b):
        self.x = a
        self.y = b
#分别使用类和类对象输出参数,体会两者的区别
object_test = my_class(10,20)                    #定义类对象 object_test,给参数 a 和 b 传值
print(object_test.x,object_test.y)               #输出类对象参数 x、y
print(my_class.x,my_class.y)                     #输出类参数 x、y
my_class.x = 1                                   #重新定义类参数 x
my_class.y = 2                                   #重新定义类参数 y
print(my_class.x,my_class.y)                     #输出类参数 x、y
```

程序运行结果如下：

```
10 20
100 200
1 2
```

5.2.3 定义和访问实例属性与类属性

属性(成员变量)一共有两种：一种是实例属性；另一种是类属性(类变量)。实例属性是在构造函数__ init__()中定义的，定义时以 self 作为前缀；类属性是在类中方法之外定义的属性。在主程序中，也就是在类的外部，实例属性属于实例(对象)，只能通过对象名访问；类属性属于类，可以通过类名访问，也可以通过对象名访问，为类的所有实例共享。

【例 5-3】 定义一个含有实例属性(课程名 kcm、学时 xs)和类属性(学科总数 num)的 Course“课程”类。程序代码如下：

```
class Course:
    num = 0                                  #成员变量(属性)
    def __init__ (self,s,n):                 #构造函数,定义两个参数 s、n
        self.kcm = s                         #定义实例属性 self.kcm,并将参数 s 的值赋予它
        self.xs = n                          #定义实例属性 self.xs,并将参数 n 的值赋予它
    def PrintName(self):                     #定义成员函数,用以输出课程名和学时
        print("课程名: ",self.kcm,"学时: ",self.xs)
    def PrintNum(self):                      #定义成员函数,用以统计输出学科总数
        Course.num = Course.num + 1
        print(Course.num)
#主程序
P1 = Course("人工智能",48)                    #定义类实例 P1
P2 = Course("医学大数据",36)                  #定义类实例 P2
P1.PrintName()                               #通过类实例调用成员函数
P2.PrintName()                               #通过类实例调用成员函数
P1.PrintNum()                                #通过类实例调用成员函数
P2.PrintNum()                                #通过类实例调用成员函数
```

程序运行结果如下：

```
课程名: 人工智能 学时: 48
课程名: 医学大数据 学时: 36
```

```
1
2
```

5.3 类的继承与多态

前面介绍过，Python 作为面向对象的程序设计语言具有继承性。继承性体现在父类和子类之间、类和对象之间。Python 中，子类在继承父类时还满足多态性。这一节将主要介绍如何在代码中体现出类的继承性和多态性。

5.3.1 类的继承

继承是为了代码复用和设计复用而设计的，是面向对象程序设计的重要特性之一。当设计一个新类时，如果可以继承另一个已有的设计良好的且特性相似的类，然后进行二次开发，则会大幅地减少程序开发的工作量，提高设计效率。

类的继承的语法规则如下：

```
class 子类名(父类名):
    子类成员
```

【例 5-4】 设计 Person 类，根据 Person 派生 Student 类，并分别创建 Person 类和 Student 类的对象。程序代码如下：

```
#主程序
wangwu = Person('李楠',19,'男')
wangwu.show()
zhaoliu = Student('赵想',21,'男','20180333')
zhaoliu.show()
zhaoliu.setAge(20)                              #通过父类成员函数更新实例属性
zhaoliu.show()
#定义父类:Person 类
class Person:                                   #创建类 Person
    def __init__(self,name = '',age = 20,sex = '女'):
        self.setName(name)                      #通过 self 调用成员函数,并传递参数
        self.setAge(age)                        #通过 self 调用成员函数,并传递参数
        self.setSex(sex)                        #通过 self 调用成员函数,并传递参数
    def setName(self,name):                     #定义成员函数
        if type(name)!= str:                    #内置函数 type()返回被测对象的数据类型
            print('姓名必须是字符串!')
            return
        self.__name = name                      #变量符合类型要求,则赋值
    def setAge(self,age):                       #定义成员函数
        if type(age)!= int:                     #内置函数 type()返回被测对象的数据类型
            print('年龄必须是整型!')
            return
        self.__age = age                        #变量符合类型要求,则赋值
    def setSex(self,sex):                       #定义成员函数
        if sex!= '男' and sex!= '女': #内置函数 type()返回被测对象的数据类型
            print('请输入正确的性别(男或女)!')
```

```
            return
        self.__sex = sex #变量符合类型要求,则赋值
    def show(self): #定义成员函数,用来输出对象属性
        print('姓名:',self.__name,'年龄:',self.__age,'性别:',self.__sex)
#定义子类(student 类),其中增加一个私有属性(学号)
class Student(Person):
    #定义子类构造函数,增加私有属性 studentID
    def __init__(self,name = '',age = 20,sex = 'man',studentID = '20180101'):
    #在子类中调用父类构造函数,并初始化父类私有数据成员
        Person.__init__(self,name,age,sex)
        #通过 self 调用成员函数,并传递参数
        self.setStudentID(studentID)
    def setStudentID(self,studentID):
        self.__studentID = studentID
    def show(self):
        Person.show(self)
        print('学号:',self.__studentID)
```

程序运行结果如下：

```
姓名: 李楠 年龄: 19 性别: 男
姓名: 赵想 年龄: 21 性别: 男
学号: 20180333
姓名: 赵想 年龄: 20 性别: 男
学号: 20180333
```

5.3.2 类的多继承

Python 的类可以继承多个父类。书写时将被继承的父类列表写在子类名后面，具体规则如下：

```
class SubClassName(ParentClass1{,ParentClass2,ParentClass3,…}):
    子类成员
```

例如，定义 C 类同时继承 A、B 两个基类，写法如下：

```
class A:                    #定义类 A
…
class B:                    #定义类 B
…
class C(A,B):               #定义子类 C,继承父类 A 和 B
…
```

5.3.3 类的多态

多态，即多种数据类型，也就是说对象可以满足不止一种数据类型。在这里先介绍一个用来测试变量类型的函数 isinstance()。判断一个变量是否是某个类型，可以用 isinstance()函数，书写的基本格式如下：

```
isinstance(object,class)
```

isinstance(object,class)是布尔函数,返回值为布尔值“True”或布尔值“False”,当object是class类或class子类的实例对象时,返回值就为“True”。

【例5-5】 类的多态,用函数isinstance()测试变量的类型,程序代码如下:

```
class Animal:                          #定义父类 Animal
    def run(self):
        print("Animal is running...")
class Cat(Animal):                     #定义子类 Cat
    def run(self):
        print("Cat is running...")
class Dog(Animal):                     #定义子类 Dog
    def run(self):
        print("Dog is running...")
a = Animal()                           #定义父类 Animal 的实例对象 a
b = Cat()                              #定义子类 Cat 的实例对象 b
c = Dog()                              #定义子类 Dog 的实例对象 c
print(isinstance(a, Animal))           #判断 a 是否为父类 Animal 的实例对象
print(isinstance(b, Cat))              #判断 b 是否为子类 Cat 的实例对象
print(isinstance(c, Dog))              #判断 c 是否为子类 Dog 的实例对象
print(isinstance(c, Animal))           #判断 c 是否为父类 Animal 的实例对象
print(isinstance(a, Dog))              #判断 a 是否为子类 Dog 的实例对象
```

程序运行结果如下:

```
True
True
True
True
False
```

从例5-5可以看到,类Dog是类Animal的子类,所以对象c作为类Dog的实例,既具有Dog的数据类型,同时又具有Animal的数据类型。因为类Dog本身就继承了父类Animal的属性。

本章小结

本章介绍了面向对象程序设计的基本概念,Python中创建类和对象的语法规则,实例属性和类属性以及类的继承、多继承和多态。

在本章中,读者应该重点掌握对象和类的概念以及面向对象程序设计方法的3个主要特征,即封装性、继承性和多态性。

值得注意的是,Python中存在着一些与其他程序设计语言比较明显的不同。其中最为重要的是:在其他面向对象程序设计语言中,类是没有属性的,而在Python中,类和对象一样,是具有属性的。

另外,面向对象程序设计方法与面向过程程序设计方法存在较大区别,需要读者在不断的学习实践中慢慢体会。