第 5 章

面向对象编程

Python 语言是一种解释型语言，它既支持面向过程编程，也支持面向对象编程。也就是说，利用 Python 语言既可以设计与运行面向过程（结构化）程序，也可以设计与运行面向对象程序。

Python 语言中的类提供了面向对象编程的所有功能，如继承机制允许多个基类，派生类可以覆盖基类中的任何方法，并且可以调用基类中的同名方法等。

5.1 程序设计的重要里程碑

面向过程与面向对象是程序设计发展史上的两个重要里程碑。

5.1.1 面向过程编程

面向过程编程是一种以过程为中心的编程思维方式，是以什么正在发生为主要目标进行编程的。在面向过程编程中，数据与计算分离。当需要计算时，将数据传给计算系统，计算完成之后，再传回计算的结果。这种编程方法存在的一个问题是变量容易被修改，一旦出现错误却很难查找出来。另一个问题是，当程序较大时，相互间的关系复杂，如

果要增加或修改一段程序代码，有可能会影响另一些程序段，出现了副作用，这就增加了程序维护的困难。

由于面向过程的设计是自上而下、逐步求精，所以其最重要的方法是模块化。首先进行总体设计，将程序按功能划分成若干模块，然后通过详细设计完成模块内的设计。当程序规模不大时，面向过程方法的优势是程序的流程清楚，可以按照模块与函数的方法进行组织，将一个问题分解成若干小问题，再对每个小问题进行分解，直到每个小问题能够解决为止。然后再把这些已解决的小问题进行合并，就解决了大问题。显然，这使用了还原论的方法。

5.1.2 面向对象编程

面向对象程序设计(Orient Object Programming，OOP)是程序设计发展史上的一个重要里程碑，其主要特点是提高了程序设计的效率与软件的重用率，进而显著降低了维护成本。面向对象程序设计与面向过程程序设计的不同之处是将程序作为一组对象的集合，每个对象都可以接收其他对象发过来的消息并处理它们，致使程序的执行过程转化成一系列消息在各个对象之间传递的过程。在 Python 语言中，所有的数据类型都可以作为对象，程序设计者也可以自定义对象，面向对象中的类就是自定义对象数据类型。

在当下最流行的四种程序设计语言中，只有 C 语言是面向过程程序设计语言，而 Python 语言、Java 语言、C++ 语言都是面向对象程序设计语言，具有类似的面向对象程序设计的概念与过程。与面向过程程序设计语言相比，面向对象的方法主要是将事物对象化，对象包括属性与方法，但方法也包含了面向过程的思想。

1. 对象

在面向对象程序设计中，将数据和对数据的操作方法都绑定在同一个实体中，将这个实体称为对象。对象具有状态和行为两个特征，例如，一个人的身高或体重可以作为状态，而唱歌、打球、骑摩托车、开汽车等作为其行为。在面向对象程序设计中，将对象状态保存在变量或数据字段中，而行为则由方法来实现，如图 5-1 所示。

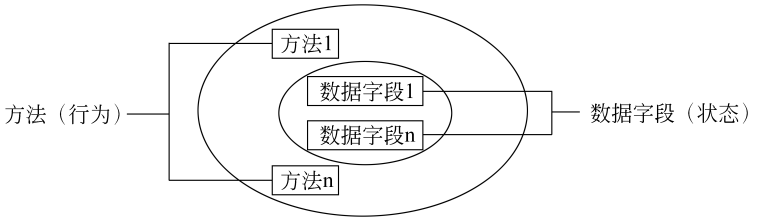

图 5-1 对象的内容

以汽车对象为例，其状态可定义为颜色、排档数、排气量和轮胎类型，而方法是换挡、刹车、开大灯和开冷气等。这样的例子不胜枚举。

2. 类

在客观世界中，有许多相同种类的对象，可将这些对象归纳为一个类。例如，可将世界上所有的汽车归纳为汽车类，如图 5-2 所示。

汽车类有一些共同的状态，如气缸排气量、排档数、颜色、轮胎数等，以及一些共同的行为，如开灯、开冷气等，但是，每一辆汽车都有个别的状态及方法可能不同于其他汽车。

3. 实例化

类的实例化是创建一个类的实例，即类的具体对象。

汽车只是世界上众多汽车中的一辆，汽车对象是汽车类中的一个实例，如图 5-3 所示。关于类与实例的概念，需要进一步来说明。为了制造汽车，不用为每一辆车设计一种蓝图，可以使用同一个蓝图来制造许多相同的汽车，或者只要稍加修改，即可制造出不同型号的汽车，这样就可以显著地提高生产效率。

图 5-2　汽车类

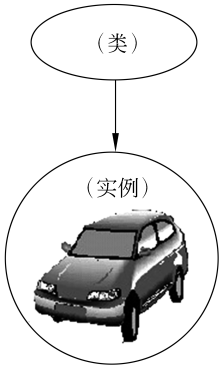

图 5-3　实例

类是用来描述具有相同属性和方法的对象的集合，它定义了该集合中每个对象所共有的属性和方法，对象是类的实例。

可以将一个类看作一个蓝图，那么实例就是从一个类中所产生的具有此类的状态（变量）与行为（方法）的对象。在 Python 中可以利用已定义的类来产生 n 个对象，其关系如图 5-4 所示。

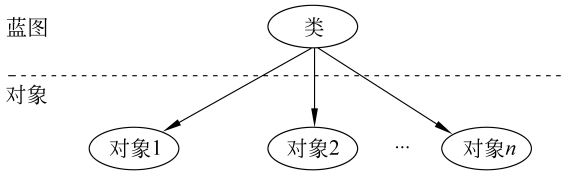

图 5-4　类与对象的关系

在图 5-4 中，描述了类与对象的关系。例如，汽车厂中的蓝图是不能拿来开车的，而照此蓝图所设计生产出来的车则是真正可以使用的对象。每当需要实例化一个对象时，就需要先从类来产生对象。例如，下述一行语句：

```
hongqi=SmallCar()
```

在上述语句中，如果 hongqi 小汽车是属于 SmallCar 类，而要使用 SmallCar 对象时，就需要实例化，使它成为属于 SmallCar 类的一个实例对象，hongqi＝SmallCar()完成了

SmallCar 类的实例化。

4. 属性和成员函数

在一个类中包含两种成员，分别称为属性和成员函数。属性是类的数据，而成员函数完成对属性的操作，在类中定义的函数就是类的方法。

5.2 Python 类

类是一种用户自定义的数据类型，是对具有共同属性和行为的一类事物的抽象描述。共同属性又称为类属性，类属性是类所特有的，类属性经常定义在类的开头和方法的外面。类属性是类中的属性，共同行为是类中的成员函数，称为成员方法。类是对象的抽象，而类的具体实例是实例对象，在应用过程中，需要先定义类，然后才能够用它定义和使用对象。

Python 的类具有所有面向对象程序设计语言的标准特征，而且具备 Python 特有的动态特点，这种动态特点表现在类是在程序运行中创建，类生成后也可以进行修改。Python 所有的类与其包含的成员都是公有的，使用时也不用声明该类的类型。

继承的最大特点是可以实现代码重用。当定义一个类时，可以继承现有的类，并将新定义的类称为子类，而将被继承的类称为基类、父类或者超类。

在 Python 中，使用关键字 class 定义类，类定义的语法格式如下：

```
class 类名(Objegt):              #定义一个类,派生自 Objegt 类,即 Objegt 代表父类或基类
    '类的帮助信息'               #类文档字符串
    语句 1
    语句 2                       类体
    …
    语句 n
```

其中，class 为类定义的关键字，class 之后为空格，接下来是所定义的类名。如果派生自其他类，则需要将所有基类放到一对圆括号中，并使用逗号相隔开，然后是一个冒号结尾，最后换行并定义类的内部实现，即类体。

类名使用大写开头的单词，最好与所描述的事务有关。

类体由属性与方法定义组成，还包括该类的被继承的类。如果没有被继承的类，就使用本类名所代表的类，这个没有被继承的类也是所有类都可以继承的类。

5.2.1 创建类

1. 类定义

例如 Person 类定义如下。当创建实例时，自动调用__init__()方法。

```
class Person(object):
    x=100

    def __init__(self, name, age, sex):
```

```
        self.name = name
        self.age = age
        self.sex = sex

    def ------
```

2. 创建实例

```
student_1 = Person('xiaoming',24,'male')
student_2 = Person('xiaohong',20,'female')
```

3. 实例属性

```
print(student_1.name)
print(student_1.age)
print(student_1.sex)
print(student_2.name)
print(student_2.age)
print(student_2.sex)
```

程序运行结果如下：

```
xiaoming
24
male
xiaohong
20
female
```

5.2.2　类的属性与方法

类的属性存储了类的各种数据。类的方法定义了类的行为特征，包括这个类的各种操作。类属性定义在类的内部和类方法的外部。

1. 类属性

类属性用以描述类的某些特征。例如：

```
class A():
    a = xx                              #类属性 a
    ...
```

(1) 类的属性可以通过(类名.属性)调用。a 是类的属性，可以通过(类名.属性)调用。例如：

```
A.a = xx                                #类的属性
```

例如，通过类访问与修改类属性。

```
>>>class Group():
      number1=30
```

```
>>> Group().number1
30
>>> Group().number1=15
>>> Group().number1
15
```

(2) 类的属性可以通过(类对象.属性)调用。对象是类的实例,对象的创建过程也是类的实例化过程。实例化是创建一个实例对象,即类的具体对象。类对象建立之后,就可以使用点运算符“.”访问对象的属性与方法。对象创建的语法格式如下:

```
对象名=类名(参数列表)
```

例如:

```
maclass1=MyClass()
```

maclass1 = MyClass( ) 的作用是完成类 MyClass 的实例化,即创建一个对象 Maclass1。一个类的对象可以有多个。

例如,通过类对象访问与修改类属性。

```
>>>group1 = Group()
>>>group1.number1                    #访问 number1 属性
30
>>>group1.number1=35                 #修改 number1 属性
>>>group1.number1
35
```

2. 类的方法

类的方法定义了类的行为特征,包括这个类的各种操作。类的方法可以通过“对象名.对象方法“的形式调用,其调用格式如下:

```
对象名.对象方法
```

例如:

```
class A():
    a = 59                             #类的属性 a
    b = 32                             #类的属性 b

    def ff(self):
        return 'Python'

print(A.a)                             #类的属性 a
print(A.b)
ob_01=A()                              #ob_01 是 A()的实例对象
print(ob_01.a)
print(ob_01.ff())
```

程序运行结果如下：

```
59
32
59
Python
```

在上面的程序中，在类中定义的 ff()方法中的第 1 个参数必须是 self，除了这一点之外，类方法与函数无区别。

方法是在类中定义的，并以关键字 self 作为第一个参数。self 参数代表调用这个方法的对象本身。在方法调用时，可以不用传递这个参数，系统将自动调用方法的对象作为 self 参数传入。

3. 实例属性

实例属性是描述对象特征的数据属性，实例属性只能由对象调用，不能由类调用。属性的调用格式如下：

```
对象名.属性
```

例如：

```
xiaoli = TesClass()                    #实例化对象
print(xiaoli.x)
```

实例属性设置可以在类定义中添加，也可以在调用实例的代码中添加。

(1) 在类定义的对象方法中设置实例属性。

在类定义的对象方法中设置的实例属性，只有在调用了该方法后才能使用这个属性，否则会出现错误。特别注意，实例属性只能由对象调用，不能由类来调用。

(2) 在调用对象时，可以动态地给对象添加属性。

(3) 删除实例属性。

删除实例属性是 Python 特有的语法，例如删除 myclass1 对象的 z 属性：

```
>>>del myclass1.z
```

如果属性不存在，就出现错误提示。

【例 5-1】 实例属性及调用。

```
class MyClass:                         #类定义
    a=123456                           #类的属性
    b=789                              #类的属性
    c=888                              #类的属性

    def myMethod(self,x,y,z):          #方法
        self.x=x                       #定义实例属性
        self.y=y                       #定义实例属性
        self.z=z                       #定义实例属性
        return x+y+z
```

```
    def f(self):                          #方法
        return 'Hello Python'
myclass1= MyClass()                       #类 MyClass 实例化,对象变量为 myclass1
print(myclass1.a)                         #使用对象访问属性 a
print(myclass1.b)                         #使用对象访问属性 b
print(myclass1.f())                       #使用对象调用方法 f()
myclass1.d=350                            #添加了实例属性 d
print('d=', myclass1.d)                   #使用对象访问实例属性 d
print(myclass1.myMethod(1,2,3))           #使用对象调用方法 myMethod()
print(myclass1.z)                         #使用对象访问实例属性 z
del myclass1.z                            #删除实例属性 z
print(myclass1.z)                         #使用对象访问实例属性 z
```

程序运行结果如下：

```
123456
789
Hello Python
d= 350
6
3
Traceback (most recent call last):
  File "D:\Python\Python38\例 5-1.py", line 23, in <module>
    print(myclass1.z)
AttributeError: 'MyClass' object has no attribute 'z'
```

4. __init__()方法

使用__init__()方法，可以在创建类的实例时，自动调用这个方法，完成实例的属性初始化。例如，下述程序：

```
class TestClass (object):
    def __init__(self, name, gender,age):
        self.Name = name
        self.Gender = gender
        self.Age=age
        print ('hello')

testman = TestClass ('xiaowang', 'male','25')
print (testman.Name)
print (testman.Gender)
print (testman.Age)
```

程序运行结果如下：

```
hello
```

```
xiaowang
male
25
```

这里__init__()方法有三个参数，这个 self 指的是在创建类的实例时被创建的实例本身，即 self.Name＝name。

通常写成 self.name＝name，这是为了区分前后两个 name 是不同的含义而将前面那个 name 的首字母大写了，等于号左边的那个 Name(或 name)是实例属性，后面那个是方法__init__()的参数。通常将 self.Gender＝gender 写成 self.gender＝gender。print('hello')是说明在创建类的实例时__init__()方法就被调用了。

testman ＝ TestClass('xiaowang','male')创建了类 TestClass 的一个实例 testman，类中有__init__()这个方法，在创建类的实例时，就必须要有和方法__init__()匹配的参数了，由于 self 指的就是创建的实例本身，self 不用传入，因此传入两个参数。这条语句初始化实例 testman 的两个属性 Name 和 Gender，其中 Name 是 xiaowang，Gender 是 male。

实例属性和具体某个实例对象有关系，并且各个实例对象之间不共享实例属性，实例属性值仅在自己的实例对象中使用，其他的实例对象不能直接使用，因为 self 值就属于该实例对象。在类外面，可以通过实例对象调用实例属性。在类中则通过 self.实例属性调用它们。实例属性就相当于局部变量，在这个类或者这个类的实例对象之外，就不起作用。

5. 类属性和实例属性比较

类属性用以描述类的某些特征，是直接在类中创建的属性。实例属性是描述对象特征的数据属性。实例属性由每个实例各自拥有，相互独立；而类属性有且只有 1 份，创建的实例都将继承自唯一的类属性。这就是说，绑定在一个实例上的属性不会影响到其他的实例。如果在类上绑定一个属性，那么所有的实例都可以访问类属性，且访问的类属性是同一个，一旦类属性改变就将影响到所有的实例，这种情况不是我们想要的。例如：

```
class Person(object):
      address = 'Earth'          #类属性 address

      def __init__(self, name):
          self.name = name

print(Person.address)            #类属性直接绑定在类上，可以直接通过类名调用类属性
p1 = Person(xiaoliu')
p2 = Person('xiaowang')
print(p1.address)                #通过实例调用类属性
print(p2.address)
```

程序运行结果如下：

```
Earth
Earth
Earth
```

【例 5-2】 类属性和实例属性的比较。

在本例中,通过类属性和实例属性的比较来说明概念的区别。

```
#创建类
>>>class Test(object):
          class_attr = 10                #类属性

          def __init__(self):
              self.example= 10           #实例属性

          def func(self):
              print('类.类属性的值:',Test.class_attr) #调用类属性
              print('self.类属性的值:',self.class_att#将类属性变成实例属性
              print('self.实例属性的值:', self.example#调用实例属性 example
>>>a = Test()                           #创建 a 对象
>>>a.func()
类.类属性的值: 10
self.类属性的值: 10
self.实例属性的值: 10

>>>b = Test()                           #创建 b 对象
>>>b.func()
类.类属性的值: 10
self.类属性的值: 10
self.实例属性的值: 10

>>>a.class_attr = 20                    #通过 "实例对象.类属性"修改类属性
>>>a.example = 20                       #通过"实例对象.实例属性"修改实例属性的值
>>>a.func()
类.类属性的值: 10
self.类属性的值: 20
self.实例属性的值: 20

>>>b.func()                             #再次运行 b 对象
类.类属性的值: 10
self.类属性的值: 10
self.实例属性的值: 10

>>>Test.class_attr = 30                 #通过"类.类属性"修改属性值
>>>a.func()
类.类属性的值: 30
self.类属性的值: 20
self.实例属性的值: 20
```