Python程序设计

第七讲 面向对象程序设计

类与对象

张 华
WHU

类与对象

类的定义

class 类名[(基类表)]:
[类变量名=初始值]
[def 方法名(self[,参数表]):
方法体]

示例

```
class Dog:                              #定义Dog类
    def bark(self):                     #定义方法bark()
        print ("汪!汪!汪!")
```

类与对象

对象的创建和使用

- 定义了类之后，就可以用来实例化对象，并通过“对象名.成员”的方式来访问其中的数据成员或成员方法。
- 示例

```
dog1 = Dog()                    #创建对象dog1
dog1.bark()                     #调用对象dog1的bark()方法
```

- 在Python中，可以使用内置函数isinstance()来测试一个对象是否为某个类的实例，或者使用内置函数type()查看对象类型。

```
>>> isinstance(dog1, Dog)
True
```

类的成员

私有成员

- 在类的外部不能直接访问，一般是在类的内部进行访问和操作，或者在类的外部通过调用对象的公有成员方法来访问。

公有成员

- 是可以公开使用的，既可以在类的内部进行访问，也可以在外部程序中使用。

类的成员

- 在Python中，以下划线开头的变量名和方法名有特殊的含义，尤其是在类的定义中。
  - _xxx：受保护成员；
  - __xxx__：系统定义的特殊成员；
  - __xxx：私有成员，只有类对象自己能访问，子类对象不能直接访问到这个成员。
    - 但是，在对象外部可以通过“对象名._类名__xxx”这样的特殊方式来访问。
    - Python中不存在严格意义上的私有成员。

类的成员

实验：类的私有成员和保护成员

```
>>> class class1:
        __pri_m = 123
        _pro_m = 456


>>> o1 = class1()
>>> o1.__pri_m
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    o1.__pri_m
AttributeError: 'class1' object has no attribute '__pri_m'
>>> o1._class1__pri_m
123
>>> o1._pro_m
456
```

类的成员

构造函数

- __init__()，在创建对象时被自动调用和执行，一般用来为数据成员设置初值或进行其他必要的初始化工作。
  - 如果用户没有定义构造函数，Python将提供一个默认的构造函数用来进行必要的初始化工作。

析构函数

- __del__()，在删除对象和收回对象空间时被自动调用和执行，一般用来释放对象占用的资源。
  - 如果用户没有定义析构函数，Python将提供一个默认的析构函数进行必要的清理工作。

类的成员变量（数据成员或属性）

- 数据成员可以大致分为两类
  - 实例属性（属于对象的数据成员）
    - 一般在构造方法__init__()中定义，当然也可以在其他成员方法中定义，在定义和在实例方法中访问数据成员时以self作为前缀，同一个类的不同对象（实例）的数据成员之间互不影响。
    - 在主程序中或类的外部，对象数据成员属于实例（对象），只能通过对象名访问。
  - 类属性（属于类的数据成员）
    - 是该类所有对象共享的，不属于任何一个对象，在定义类时这类数据成员一般不在任何一个成员方法的定义中。
    - 类数据成员属于类，可以通过类名或对象名访问。

类的成员变量

实例属性

- 通过“self.变量名”定义的属性
- 示例

```
class Dog:
    def __init__(self, name):
        self.name = name

    def bark(self):
        print ("汪!汪!汪!", self.name+"在这里。")

dog1 = Dog("阿黄")
dog1.bark()
```

```
汪!汪!汪! 阿黄在这里。
```

面向对象编程案例

练习1

- 定义一个类表示球体，包括以下属性和方法：
  - 属性：半径 **radius**
  - 构造函数：获取半径值
  - 计算表面积：**SurfaceArea = 4*pi*radius**2**
  - 计算体积：**Volume = 4/3*pi*radius**3**
- 然后，编写程序让用户输入一个球体的半径值，然后计算其表面积和体积。

面向对象编程案例

练习1

```
# Ball.py
import math

class Ball:
    def __init__(self, radius):
        self.radius = radius

    def surfaceArea(self):
        return 4*math.pi*self.radius**2

    def volume(self):
        return 4/3*math.pi*self.radius**3

def main():
    r = float(input('Enter the radius of a ball: '))
    ball = Ball(r)

    print('Surface area: {0:.2f}'.format(ball.surfaceArea()))
    print('Volume: {0:.2f}'.format(ball.volume()))

main()
```

类的成员变量

类属性

- 类属性是类本身的成员变量，在类中成员函数外部定义。
- 示例

```
class Dog:
    name = "狗"

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

    def bark(self):
        print ("汪!汪!汪!", self.name+"在这里。")

dog1 = Dog("阿黄")
dog1.bark()
print(dog1.name+"是一条"+Dog.name)
```

```
汪!汪!汪! 阿黄在这里。
阿黄是一条狗
```

类的成员变量

类属性

- 如果类属性名和实例属性名不冲突，也可以通过对象名来引用类属性。
- 示例

```
class Dog:
    animalname = "狗"

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

    def bark(self):
        print ("汪!汪!汪!", self.name+"在这里。")

dog1 = Dog("阿黄")
dog1.bark()
print(dog1.name+"是一条"+dog1.animalname)
```

类的成员变量

类属性

- 类属性是被类的所有对象共享的，实例属性只被自己的对象所拥有。
- 示例

```
Dog.animalname = "中华田园犬"

dog2 = Dog("阿黑")
print(dog2.name+"是一条"+dog2.animalname)

print(dog1.name+"是一条"+dog1.animalname)
```

```
阿黑是一条中华田园犬
阿黄是一条中华田园犬
```

类的成员变量

类属性

- 利用类属性的共享性，可以实时获得该类的对象数量，并且可以控制该类可以创建的对象最大数量。
- 示例

```
class Dog:
    animalname = "狗"
    dogcount = 0

    def __new__(cls, *args, **kwargs): # 该方法在__init__()之前被调用
        if cls.dogcount>=3:
            raise Exception("最多只能有3条"+cls.animalname)
        else:
            return object.__new__(cls)

    def __init__(self, name):
        self.name = name
        Dog.dogcount += 1

    def bark(self):
        print ("汪!汪!汪!", self.name+"在这里。")
```

类的成员变量

类属性

示例

```
dog1 = Dog("阿黄")
dog2 = Dog("阿黑")
dog3 = Dog("白雪")
print("现在有", Dog.dogcount, "条"+Dog.animalname)

dog4 = Dog("不可能")
```

```
现在有 3 条狗
…
Exception: 最多只能有3条狗
```

类的成员方法

- Python类的成员方法大致可以分为
  - 特殊方法
    - 新建函数__new__
    - 构造函数__init__
    - 析构函数__del__
  - 实例方法
  - 类方法
  - 静态方法
  - 抽象方法

类的成员方法

实例方法

- 所有实例方法都必须至少有一个名为**self**的参数，并且必须是方法的第一个形参（如果有多个形参的话），**self**参数代表当前对象。
- 在实例方法中访问实例成员时需要以**self**为前缀，但在外部通过对象名调用对象方法时并不需要传递这个参数。
- 如果在外部通过类名调用属于对象的公有方法，需要显式为该方法的**self**参数传递一个对象名，用来明确指定访问哪个对象的成员。

类的成员方法

静态方法和类方法

- 静态方法和类方法都可以通过类名和对象名调用，但不能直接访问属于对象的成员，只能访问属于类的成员。
- 静态方法和类方法不属于任何实例，不会绑定到任何实例，当然也不依赖于任何实例的状态，与实例方法相比能够减少很多开销。
- 类方法一般以cls作为类方法的第一个参数表示该类自身，在调用类方法时不需要为该参数传递值，静态方法则可以不接收任何参数。

类的成员方法

类的成员方法示例

```
class Dog:
    __name = "狗" # 类属性
    __count = 0          # 类属性

    def __init__(self, name): # 构造函数
        self.name = name
        Dog.__count += 1

    def bark(self): # 一般实例方法
        print ("汪!汪!汪!", self.name+"在这里。")

    @classmethod                # 定义类方法的装饰器
    def animal_name(cls):       # 类方法
        return cls.__name

    @staticmethod               # 定义静态方法的装饰器
    def dog_count():            # 静态方法
        return Dog.__count
```

类的成员方法

类的成员方法示例

```
dog1 = Dog("阿黄")
dog1.bark()
print(dog1.name+"是一条"+dog1.animal_name())
print("现在有", dog1.dog_count(), "条"+dog1.animal_name())

dog2 = Dog("阿黑")
Dog.bark(dog2) # 通过类名来调用实例方法，需要把对象名作为self参数的实参
print(dog2.name+"是一条"+dog2.animal_name())
print("现在有", Dog.dog_count(), "条"+Dog.animal_name())
```

```
汪!汪!汪! 阿黄在这里。
阿黄是一条狗
现在有 1 条狗
汪!汪!汪! 阿黑在这里。
阿黑是一条狗
现在有 2 条狗
```

面向对象编程案例

练习2

- 定义一个类SavingsAccount描述银行账户
  - 定义以下私有的数据成员：
    - name，存储储户姓名
    - balance，存储账户的当前余额
  - 定义类数据成员annualInterestRate，用来存储年利率。
  - 定义静态成员函数setAnnualInterestRate，可以重设利率。
  - 定义成员函数calculateMonthlyInterest，用来计算月利息（余额X年利率/12）。
  - 定义执行如下操作的成员函数：
    - 创建对象，并将其初始化。
    - 显示储户姓名、余额、年利率和月利息。
    - 存入参数指定的金额（存款）。
    - 取出参数指定的金额（取款）。

面向对象编程案例

练习2

编写测试程序

- 定义两个账户对象account1，account2
- 余额分别为3000元，5000元
- 设置年利率为3%
- 显示每个账户的月利息
- 储户进行存款和取款操作后显示账户的余额
- 修改年利率为2.5%，然后再显示账户的余额和月利息

类的定义

只读属性

示例

```
class Test:
    def __init__(self, value):
        self.__value = value

    @property
    def value(self):      #只读，无法修改和删除
        return self.__value
```

类的定义

只读属性

示例

```
>>> t = Test(3)
>>> t.value
3
>>> t.value = 5                                  #只读属性不允许修改值
AttributeError: can't set attribute
>>> t.v=5                                        #动态增加新成员
>>> t.v
5
>>> del t.v                                      #动态删除成员
>>> del t.value                                  #试图删除对象属性，失败
AttributeError: can't delete attribute
>>> t.value
3
```

类的定义

可读可写的属性

示例

```
class Test:
    def __init__(self, value):
        self.__value = value

    def __getValue(self):
        return self.__value

    def __setValue(self, v):
        self.__value = v

    value = property(__getValue, __setValue)

    def show(self):
        print(self.__value)
```

类的定义

可读可写的属性

示例

```
>>> t = Test(3)
>>> t.value        #允许读取属性值
3
>>> t.value = 5    #允许修改属性值
>>> t.value
5
>>> t.show()       #属性对应的私有变量也得到了相应的修改
5
>>> del t.value    #试图删除属性，失败
AttributeError: can't delete attribute
```

类的定义

可读可写可删除的属性

示例

```
class Test:
    def __init__(self, value):
        self.__value = value

    def __getValue(self):
        return self.__value

    def __setValue(self, v):
        self.__value = v

    def __delValue(self):
        del self.__value

    value = property(__getValue, __setValue, __delValue)

    def show(self):
        print(self.__value)
```

类的定义

可读可写可删除的属性

示例

```
>>> t = Test(3)
>>> t.show()
3
>>> t.value
3
>>> t.value = 5
>>> t.show()
5
>>> t.value
5
>>> del t.value                    #删除属性
>>> t.value                        #对应的私有数据成员已删除
AttributeError: 'Test' object has no attribute '_Test__value'
>>> t.show()
AttributeError: 'Test' object has no attribute '_Test__value'
>>> t.value = 1                    #为对象动态增加属性和对应的私有数据成员
>>> t.show()
1
>>> t.value
1
```

类与对象的动态性和混入机制

动态性

- 在Python中比较特殊的是，可以动态地为自定义类和对象增加或删除成员，这一点是和很多面向对象程序设计语言不同的，也是Python动态类型特点的一种重要体现。

混入机制

- Python类型的动态性使得我们可以动态地为自定义类及其对象增加新的属性和方法，俗称混入机制，这在大型项目开发中会非常方便和实用。
  - 例如，系统中的所有用户分类非常复杂，不同用户组具有不同的行为和权限，并且可能会经常改变。
  - 这时候我们可以独立地定义一些行为，然后根据需要来为不同的用户设置相应的行为能力。

类与对象的动态性和混入机制

动态性和混入示例

```
class Car:
    price = 100000                          #定义类属性
    def __init__(self, c):
        self.color = c                      #定义实例属性

car1 = Car("Red")                           #实例化对象
car2 = Car("Blue")
print(car1.color, Car.price)                #查看实例属性和类属性的值
Car.price = 110000                          #修改类属性
Car.name = 'QQ'                             #动态增加类属性
car1.color = "Yellow"                       #修改实例属性
print(car2.color, Car.price, Car.name)
print(car1.color, Car.price, Car.name)
```

```
Red 100000
Blue 110000 QQ
Yellow 110000 QQ
```

类与对象的动态性和混入机制

动态性和混入示例

```
import types

def setSpeed(self, s):
    self.speed = s

car1.setSpeed = types.MethodType(setSpeed, car1)  #动态增加成员方法
car1.setSpeed(50)                                  #调用成员方法
print(car1.speed)
```

类与对象的动态性和混入机制

动态性和混入示例

```
>>> import types
>>> class Person(object):
    def __init__(self, name):
        assert isinstance(name, str), 'name must be string'
        self.name = name

>>> def sing(self):
    print(self.name+' can sing.')

>>> def walk(self):
    print(self.name+' can walk.')

>>> def eat(self):
    print(self.name+' can eat.')
```

类与对象的动态性和混入机制

动态性和混入示例

```
>>> zhang = Person('zhang')
>>> zhang.sing()                                    #用户不具有该行为
AttributeError: 'Person' object has no attribute 'sing'
>>> zhang.sing = types.MethodType(sing, zhang) #动态增加一个新行为
>>> zhang.sing()
zhang can sing.
>>> zhang.walk()
AttributeError: 'Person' object has no attribute 'walk'
>>> zhang.walk = types.MethodType(walk, zhang)
>>> zhang.walk()
zhang can walk.
>>> del zhang.walk                                  #删除用户行为
>>> zhang.walk()
AttributeError: 'Person' object has no attribute 'walk'
```