链滴

Python 中类的三大特性

作者：branda2019wj
原文链接：https://ld246.com/article/1554255363898
来源网站：链滴

类的三大特性

1 封装

第一层面的封装:创建类和对象时，分别创建两者的名称空间。只能通过类名加“.”或者obj.的方式访里面的名字
第二层面的封装:类中把某些属性和方法隐藏起来，或者定义为私有，只在类的内部使用，在类的外部法访问，或者留下少量的接口(函数)供外部访问

2 继承

继承分为单继承和多继承
单继承就是子类继承父类的属性和方法，子类也可以对父类的属性和方法进行重构
类的多继承是子类可以同时继承父类1和父类2的属性和方法，但如果父类1和父类2有相同名字的属性方法，子类会按从左到右的顺序执行，若父类1中的属性和方法和父类2中的有重叠，那重叠部分就只承父类1的属性和方法

继承顺序：
Python2中：继承策略是深度优先（经典类）
Pyhton3即以上版本中：继承策略是广度优先（新式类）

3 多态

特性：同一种接口，多种实现
比如动物都会叫，定义一个父类动物，子类1狗，子类2猫，分别定义了叫的方法，通常情况下，如果想让猫叫，调用猫叫的方法就行，如果我现在想只调用一个接口，让狗狗和猫咪都叫这就是一种接口多种实现

构造函数

__init__()函数也叫构造函数，其作用是在实例化的时候做一些类的初始化工作

析构函数

含义：在实例释放、销毁的时候自动执行的，通常用于做一些收尾工作。比如关闭一些数据库连接、闭打开的临时文件等等（方便程序结束的时候回收内存）

案例解析1：类和构造函数

```
# 主要介绍类和构造函数
# 第一步：先定义一个类，比如student,初始化中定义student的属性，比如name、sex、grade等
# 第二步： 生成一个对象，比如stu1
# 第三步： 定义一个函数，score大于90分，就进入精英班（jyb）
class student(object):
    n = 123 # 类变量
    name = "我是类的name"
    def __init__(self,name,sex,grade,age,score):
        self.name = name # 实例变量（也称为静态属性），其作用域就是实例本身
        self.sex = sex
```

```
        self.grade = grade
        self.age = age
        self.score = score

    def jyb(self):   # 类方法，也称为动态属性
        if self.score > 90:
            print("congratulation,you have be a student of jyb")
        else:
            print("sorry")


stu1 = student('candy','female','grade6',13,91)# 把一个类变成一个具体对象的过程叫实例化
stu2 = student('sandy','female','grade5',12,89)
stu1.name = 'alice' # 可以修改实例上stu1的name值
stu1.hobby = 'swimming'
stu1.n = "修改类变量" # 在实例中，可以修改类变量,但只对实例stu1生效，对stu2无影响
# stu1.jyb()
print(stu1.hobby)
print(stu1.n)  # 在实例中可以查类变量，比如此处的n
print(stu2.n)
```

案例解析2：析构函数、私有属性和私有方法

```
# 主要介绍析构函数、私有属性和私有方法
# 私有属性：一旦定义了私有属性，外界就无法访问到score了（例子中的score），若想查看,必须定
私有方法，   然后通过私有方法调用和查看
class student(object):
    n = 123 # 类变量
    name = "我是类的name"
    def __init__(self,name,sex,grade,age,score):
        self.name = name # 实例变量（也称为静态属性），其作用域就是实例本身
        self.sex = sex
        self.grade = grade
        self.age = age
        self.__score = score # 在score前面加两个_,此变量就是私有变量，也称为私有属性，只有加了
有方法才能访问

    # 现在有一个场景是我不想让人随便改变我的分数score,那就要定义一个私有方法，私有属性可以在
      私有方法内部被调用，在外部只能查看
    def show_status(self):
        self.__score -= 2 # 在原有的分数基础上减去2分
        print("name:%s grade:%s score:%s" %(self.name,self.grade,self.__score))


    # def __del__(self): # 析构函数
      # print("%s 分数非常高哈" %self.name)



stu1 = student('candy','female','grade6',13,91)
print(stu1.show_status()) # 查看
```

案例解析3：类的单继承

```
# 主要介绍类的继承
# 第一步，先定义一个类，此处定义的是人类，设置一个初始化，初始化值是姓名和年龄
# 第二步 定义人类都可以做的事情，比如都需要eat，都需要sleep，都可以talk
# 第三步 定义子类，子类可以继承父类，可以对父类进行重构，但子类与子类之间不可以调用
# 第四步 创建实例
# class People: # 经典类
class People(object): # 新式类，新式类和经典类在多继承的方式上有很多不同
    def __init__(self,name,age):
        self.name = name
        self.age = age

    def eat(self):
        print("%s is eating" %self.name)

    def sleep(self):
        print("%s is sleepping" %self.name)

    def talk(self):
        print("%s is talking" %self.name)

# 接下来定义子类,在括号中传入People,就代表继承父类了,但子类定义的方法不可以互相调用
class Man(People):
# 接下来想给子类新增加一个属性，比如男人有胡子这个属性(如果不想影响其他子类的调用
# 必须按照如下的写法，先初始化，初始化过程中加上子类自己的属性，然后还需要调用父类
    def __init__(self,name,age,mustache):
        # People.__init__(self,name,age) # 调用父类的方法一，经典类的写法
        super(Man,self)._init_(name,age) # 调用父类的方法二，此方法比较常用，是新式类的写法，
处可以不用重复输入People，因为如果父类名称改变，方法一就要跟着修改，比较麻烦
        self.mustache = mustache
        print("%s 一出生就有%s " %(self.name,self.mustache))

# 子类可以定义一些新的方法，是子类Man独有的,比如男人都有胡子,所以都需要刮胡子
    def shave(self):
        print("%s is shaving" %self.name)

# 子类还可以重构父类的方法，比如添加一个新的功能
    def eat(self):
        People.eat(self) # 父类的功能
        print("man is eating now") # 在父类功能的基础上新添加的功能

# 再定义一个子类Woman，Woman独有的功能是可以生宝宝，英文叫give birth
class Woman(People):
    def give_birth(self):
        print("%s is giving birth" % self.name)


# 创建实例
m1 = Man("hou",18,"a litter mustache")
w1 = Woman("wj",29)
w1.give_birth()
m1.talk()
```

案例解析4：类的多继承

```
# 主要介绍类的多继承
# 在多继承的过程中，从左到右依次执行，如果父类1里面没有构造函数，父类2里有，就会直接继承类2的
# 继承顺序：1 Python2中的经典类的继承顺序是深度优先，新式类是广度优先 2 Python3即以上版的中的经典类和新式类的继承顺序都是广度优先
class People(object): # 括号中添加了object即为新式类，新式类和经典类在多继承的方式上有很多同
    def __init__(self,name,age):
        self.name = name
        self.age = age

        def eat(self):
            print("%s is eating" % self.name)

        def sleep(self):
            print("%s is sleepping" % self.name)

        def talk(self):
            print("%s is talking" % self.name)

# 接下来，再创建一个父类，比如说人类需要进行社交relationship
class Relationship(object):
    def make_friends(self,someone):
        print("%s is making friends with %s" %(self.name,someone.name))

# 子类Man继承父类People和父类Relationship的属性和方法
class Man(People,Relationship):
    def _init_(self,name,age,mustache): # 覆盖父类的构造函数
        # People.__init__(self,name,age) # 继承父类的方法之一，经典类的写法，现在基本不用了
        super(Man,self)._init_(name,age) # 继承父类的属性和方法，此方法比较常用，是新式类的写法
        self.mustache = mustache # 新增子类独有的属性
        print("%s 一出生就有%s " %(self.name,self.mustache))

# 子类Woman继承父类People和Relationship
class Woman(People,Relationship):
    def give_birth(self):
        print("%s is giving birth" % self.name)
# 创建实例
m1 = Man("hou",18)
w1 = Woman("wj",29)
m1.make_friends(w1) # 运行结果就是hou is making friends with wj
```

案例分析5：继承实例

```
# 本实例主要描述学院、讲师、学员之间的关系
# 首先定义一个学校的类，即School
class School(object):
    def __init__(self,sname,addr):
```

```
        self.sname = sname
        self.addr = addr
        self.students = [] # 初始化学生空列表
        self.staffs = [] # 初始化老师空列表

    # 学校为学生提供了注册的功能
    def enroll(self,stu_object):
        print("为学员%s 办理注册手续" %stu_object.name)
        self.students.append(stu_object)

    # 学校为老师提供了雇佣功能
    def hire(self,staff_object):
        self.staffs.append(staff_object)
        print("雇佣新员工%s" %staff_object.name)

# 定义一个类，叫学校成员（讲师和学生都属于学校成员）
class SchoolMember(object):
    def __init__(self,name,age,sex):
        self.name = name
        self.age = age
        self.sex = sex
    def tell(self):     # 定义一个打印出成员信息的功能
        pass

# 定义一个教师类，它可以继承父类SchoolMember
class Teacher(SchoolMember):
    def __init__(self,name,age,sex,salary,course): # 覆盖父类
        super(Teacher,self).__init__(name,age,sex) # 继承父类已经实现了的
        self.salary = salary # 新增新的属性salary
        self.course = course # 新增新的属性course

    def tell(self): # 定义了一个打印出老师基本信息的方法
        print('''--------info of Teacher %s--------
        Name:%s
        Age:%s
        Sex:%s
        Salary:%s
        Course:%s
        '''%(self.name,self.name,self.age,self.sex,self.salary,self.course))

    # 给老师添加一个教课功能
    def teach(self):
        print("%s is teaching [%s]" %(self.name,self.course))

# 定义一个student类，继承父类SchoolMember
class Student(SchoolMember):
    def __init__(self,name,age,sex,stu_id,grade):
        super(Student,self).__init__(name,age,sex)
        self.stu_id = stu_id
        self.grade = grade

    def tell(self): # 定义了一个打印出学生基本信息的方法
        print('''--------info of Student %s--------
        Name:%s
```

```
        Age:%s
        Sex:%s
        Stu_id:%s
        Grade:%s
        '''%(self.name,self.name,self.age,self.sex,self.stu_id,self.grade))

    # 定义一个交学费的功能,amount代表的是金额数量
    def pay_tuition(self,amount):
        print("%s has paid tuition for $%s " %(self.name,amount))

# 实例化
# 实例化一个学校
school = School("新东方烹饪学校","南京路1号")

# 实例化2个老师
t1 = Teacher("branda",32,"F",9000,"cooking course")
t2 = Teacher("alice",30,"M",6000,"dessert course")

# 实例化2个学生
s1 = Student("jack",18,"M",1001,"grade2")
s2 = Student("sandy",20,"F",1002,"grade3")

s1.tell() # 打印学生1的基础信息
school.enroll(s1)  # 为学生1办理注册手续
school.enroll(s2) # 为学生2办理注册手续
school.hire(t1)  # 雇佣老师1
school.staffs[0].teach() # 某个老师正在上什么课，调用的是teach()方法
```

案例实例6：多态实例

```
# 多态的特性：同一种接口，多种实现
# 多态指的是一类事物多种形态，比如动物有多种形态，有人、猪、狗等
class Animal(object):
    def __init__(self, name):
        self.name = name

    def talk(self):  # 抽象方法，仅由约定定义
        print(self.name, '叫')  # 当子类没有重写talk方法的时候调用

    def animal_talk(obj):  # 多态
        obj.talk()


class Cat(Animal):
    def talk(self):
        print('%s: 喵喵喵!' % self.name)  # 重写talk方法


class Dog(Animal):
    def talk(self):
```

```
        print('%s: 汪！汪！汪！' % self.name)


a = Dog('a')
b = Cat('b')
Animal.animal_talk(b)  # 多态调用
Animal.animal_talk(a)
```

案例分析7：静态方法、类方法和属性方法

```
主要介绍静态方法(staticmethod)、类方法（classmethod）和属性方法（property）
class Dog(object):
    # name = "alice"
    def __init__(self,name):
        self.name = name
        self.__food = None

    #@staticmethod # 静态方法：只是名义上归类管理，实际上静态方法是不可以访问实例变量
    或者类变量的，只能通过类名来调用这个方法
    #@classmethod # 类方法，只能调用类变量；类方法只能访问类变量，不能访问实例变量
    @property # 属性方法，即把一个方法变成一个静态属性，作用是隐藏细节
    def eat(self):
        print("%s is eating %s " %(self.name,self.__food)) # 此处的name是类变量name，即name=
alice"

    @eat.setter  # 只要赋值（比如d.eat = "noddles"）,就会触发此方法
    def eat(self,food):
        print("set to food: ",food)
        self.__food = food

    # 删除自由属性self.__food
    @eat.deleter
    def eat(self):
        del self.__food
        print("删完了")

d = Dog("candy")
# d.eat(d)  # 静态方法调用方式，需要通过类名来调用
d.eat # 属性方法的调用方式，不可以传参数
d.eat = "noddles"
```

案例分析8：属性方法实例

```
# 属性方法举例
# 属性方法就是把一个方法变成一个静态属性，作用是隐藏实现细节
# 定义航班类
class Flight(object):
    def __init__(self,name):
        self.flight_name = name
```

```
    # 定义查看航班状态的功能,此处是伪代码，比如某个网站和航空公司约定好
    # return 1 代表航班已经到达
    # return 0 代表航班已经取消
    # return 2 代表航班已经起飞
    # 其他返回值代表无法确认该状态
    def checking_status(self):
        print("checking flight %s status" %self.flight_name)
        return 2

    @property
    def flight_status(self):
        status = self.checking_status()
        if status == 0:
            print("flight got canceled....")
        elif status == 1:
            print("flight had arrived...")
        elif status == 2:
            print("flight has already left....")
        else:
            print("sorry,can not confirm the status...")

    # 修改航班状态
    @flight_status.setter
    def flight_status(self,status):
        print("flight %s has changed status to %s " %(self.flight_name,status))


f = Flight("CA980")
f.flight_status
f.flight_status = 0
```

案例分析九 @classmethond的用法

描述

classmethod 修饰符对应的函数不需要实例化，不需要 self 参数，但第一个参数需要是表示自身类的 cls 参数，可以来调用类的属性，类的方法，实例化对象等。

```
class A(object):
    # 属性默认类属性，可以被类本身直接调用
    num = '类属性'

    # 实例化方法，必须实例化类之后才能被调用
    def talk(self):
        print('who is talking...')

    # 类方法，不需要实例化类就可以被类本身调用
    @classmethod
    def eat(cls):
        print('branda is eating....')
```

```
A.talk()  # 运行报错，因为调用talk方法必须实例化类
A.eat()  # 成功运行，输出‘ branda is eating.....'
```