8.3 类的继承和多态

目录

CONTENTS

8.3.1 类的继承

类的继承语法：

```
class 派生类名（基类名）：      #基类名写在括号里
        派生类成员
```

在继承关系中，已有的、设计好的类称为父类或基类，新设计的类称为子类或派生类。派生类可以继承父类的公有成员，但是不能继承其私有成员。

Python中继承的一些特点：

- 在继承中基类的构造函数（__init__()方法）不会被自动调用，需要在其派生类的构造中亲自专门调用。
- 如果需要在派生类中调用基类的方法时，通过“基类名.方法名()”的方式来实现，需要加上基类的类名前缀，且需要带上self参数变量。区别于在类中调用普通函数时并不需要带上self参数。也可以使用内置函数super()实现这一目的。
- Python总是首先查找对应类型的方法，如果不能在派生类中找到对应的方法，会再到基类中逐个查找。即先在本类中查找调用的方法，找不到才去基类中找。

例：类的继承应用。

```
1   class Parent:        # 定义父类
2       parentAttr = 100
3       def __init__(self):
4           print ("调用父类构造函数")
5       def parentMethod(self):
6           print("调用父类方法")
7       def setAttr(self, attr):
8           Parent.parentAttr = attr
9       def getAttr(self):
10          print( "父类属性 :", Parent.parentAttr)
11
12  class Child(Parent): # 定义子类
13      def __init__(self):
14          print( "调用子类构造函数")
15      def childMethod(self):
16          print("调用子类方法 child method")
17  #主程序
18  c = Child()          # 实例化子类
19  c.childMethod()      # 调用子类的方法
20  c.parentMethod()     # 调用父类方法
21  c.setAttr(200)       # 再次调用父类的方法
22  c.getAttr()          # 再次调用父类的方法
```

```
调用子类构造函数
调用子类方法 child method
调用父类方法
父类属性 : 200
```

在 Python 中，可以使用 isinstance() 或者 issubclass()方法来检测判断类之间关系或者某个对象实例是哪个类的对象。

- issubclass(sub,sup)布尔函数，判断一个类sub是另一个类sup的子类或者子孙类，是则返回true.
- isinstance(obj,Class) 布尔函数，如果obj是Class类或者是Class子类的实例对象，则返回true。

下面的代码演示了isinstance()和issubclass()的用法。

```
class Foo(object):
    pass
class Bar(Foo):
    pass
a=Foo()
b=Bar()
print(type(a)==Foo)
print(type(b)==Foo)
print(isinstance(b,Foo))
print(issubclass(Bar,Foo))
```

```
True
False
True
True
```

8.3.2 类的多继承

Python的类可以继承多个基类。继承的基类列表跟在类名之后。类的多继承语法：

```
class SubClassName (ParentClass1[, ParentClass2, ...]):
    派生类成员
```

例如定义C类继承A,B两个基类如下：

```
class A:          # 定义类 A
.....
class B:          # 定义类 B
.....
class C(A, B):    # 派生类C继承类 A 和 B
.....
```

8.3.3 方法重写

重写必须出现在继承中。它是指当派生类继承了基类的方法之后，如果基类方法的功能不能满足需求，需要对基类中的某些方法进行修改，可以在派生类重写基类的方法，这就是重写。

例：重写父类（基类）的方法。

```
1  class Animal:                # 定义父类
2      def run(self):
3          print ('调用父类方法')
4  class Cat (Animal):          # 定义子类
5      def run (self):
6          print ('调用子类方法')
7  class Dog (Animal):          # 定义子类
8      def run (self):
9          print ('调用子类方法')
10
11   c = Dog()                  # 子类实例
12   c. run ()                  # 子类调用重写方法
```

运行结果如：

```
调用子类方法
```

8.3.4 多态

要理解什么是多态，首先要对数据类型再作一点说明。当定义一个class的时候，实际上就定义了一种数据类型。定义的数据类型和Python自带的数据类型，如string、list、dict没什么区别。

```
a=list()                    #a是list类型
b=Animal()                  #b是Animal类型
c=Dog()                     #c是Dog类型
```

在继承关系中，如果一个实例的数据类型是某个子类，那它的数据类型也可以被看作是父类。但是，反过来就不行：

```
>>> b=Animal()
>>> isinstance(b,Dog)
False
```

Dog可以看成是Animal,但Animal不可以看成Dog.

要理解多态的好处，还需要再编写一个函数，这个函数接收一个Animal类型的变量：

```
>>> def run_twice(animal):
        animal.run()
        animal.run()
```

当传入Animal 实例时，run_twice()就打印出：

```
>>> run_twice(Animal())
调用父类方法
调用父类方法
```

当传入Cat的实例时，run_twice()就打印出：

```
>>> run_twice(Cat())
调用子类方法
调用子类方法
```

现在如果再定义一个Tortoise类型，也从Animal派生：

```
>>> class Tortoise(Animal):
        def run(self):
            print("Tortoise is running slower...")
```

当调用run_twice()时，传入Tortoise的实例：

```
>>> run_twice(Tortoise())
Tortoise is running slower...
Tortoise is running slower...
```

我们会发现新增一个Animal的子类，不必对run_twice()做任何修改。实际上，任何依赖Animal作为参数的函数或者方法都可以不加修改地正常运行，原因就在于多态。

多态的好处就是，当我们需要传入Dog、Cat、Tortoise……时，我们只需要接收Animal类型就可以了，因为Dog、Cat、Tortoise……都是Animal类型，然后，按照Animal类型进行操作即可。由于Animal类型有run()方法，因此，传入的任意类型，只要是Animal类或者子类，就会自动调用实际类型的run()方法，这就是多态的意思。

- 多态真正的威力

调用方只管调用，不管细节，而当新增一种Animal的子类时，只要确保run()方法编写正确，不用管原来的代码是如何调用的。这就是著名的“开闭”原则：对扩展开放，允许新增Animal子类；对修改封闭，不需要修改依赖Animal类型的run_twice()等函数。

8.3.5 运算符重载

在Python中可以通过运算符重载来实现对象之间的运算。Python把运算符与类的方法关联起来，每个运算符对应一个函数，因此重载运算符就是实现函数。常用的运算符与函数方法的对应关系如下表：

Python中运算符与函数方法的对应关系表

函数方法	重载的运算符	说明	调用举例
add	+	加法	Z=X+Y,X+=Y
sub	-	减法	Z=X-Y,X-=Y
mul	*	乘法	Z=X*Y,X*=Y
div	/	除法	Z=X/Y,X/=Y
_lt_	<	小于	X<Y
_eq_	=	等于	X=Y
len	长度	对象长度	len(X)
str	输出	输出对象时调用	print(X),str(X)
_or_	或	或运算	X|Y,X|=Y

例：对Vector类重载运算符。

```
class Vector:
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def __str__(self):                  #重写print()方法，打印Vector对象实例信息
        return 'Vector (%d, %d)' % (self.a, self.b)
    def __add__(self,other):            #重载加法+运算符
        return Vector(self.a + other.a, self.b + other.b)
    def __sub__(self,other):            #重载减法-运算符
        return Vector(self.a - other.a, self.b - other.b)
#主程序
v1 = Vector(2,10)
v2 = Vector(5,-2)
print (v1 + v2)
```

运行结果为：

```
Vector (7, 8)
```

总结

总结

类的继承、类的多继承、方法重写、多态、运算符重载。

THANKS