Python面向对象编程-多态

在OOP程序设计中，当我们定义一个class的时候，可以从某个现有的class继承，新的class称为子类（Subclass，而被继承的class称为基类、父类或超类（Base class、Super class）。

比如，我们已经编写了一个名为Animal的class，有一个run()方法可以直接打印：

In [2]:
```
class Animal(object):
    def run(self):
        print('Animal is running...')

#当我们需要编写Dog和Cat类时，就可以直接从Animal类继承：

class Dog(Animal):
    pass

class Cat(Animal):
    pass

#对于Dog来说，Animal就是它的父类；对于Animal来说，Dog Cat就是它的子类.
```

继承有什么好处?

最大的好处是子类获得了父类的全部功能, 由于Animial实现了run()方法, 因此Dog和Cat作为它的子类, 什么事也没干, 就自动拥有了run()方法.

In [3]:
```
#示例代码1.
dog = Dog()
dog.run()

cat = Cat()
cat.run()
```

```
Animal is running...
Animal is running...
```

无论是Dog还是Cat, 它们run()的时候，显示的都是Animal is running..., 符合逻辑的做法是分别显示Dog is running...和 Cat is running..., 因此, 对Dog和Cat类改进如下:

In [6]:
```
#示例代码2.
class Dog(Animal):
    def run(self):
        print('Dog is running...')

    def eat(self):
        print('Eating meat...')


class Cat(Animal):
    def run(self):
        print('Cat is running...')

dog = Dog()
dog.run()

cat = Cat()
cat.run()
```

```
Dog is running...
Cat is running...
```

当子类和父类都存在相同的run()方法时, 子类的run()会覆盖了父类的run(), 在代码运行的时候, 总是会调用子类的run(). 这样, 我们就获得了继承的另一个好处: 多态.

要理解什么是多态, 我们首先要对数据类型再作一点说明; 当我们定义一个class的时候, 我们实际上就定义了一种数据类型, 我们定义的数据类型和Python自带的数据类型, 比如str、list、dict没什么两样:

In [8]:
```
#示例代码3.
x = list()
y = Animal()
z = Dog()
w = Cat()

print type(x)
print type(y)
print type(z)
print type(w)
```

```
<type 'list'>
<class '__main__.Animal'>
<class '__main__.Dog'>
<class '__main__.Cat'>
```

看来x y z w确实对应着list Animal Dog Cat这4种类型.

In [10]:
```
#示例代码4.
print isinstance(z, Dog)
print isinstance(z, Animal)
```

```
True
True
```

看来z不仅仅是Dog, z还是Animal; 这是有道理的, 因为Dog是从Animal继承下来的, 当我们创建了一个Dog的实例z时, 我们认为z的数据类型是Dog没错, 但z同时也是Animal也没错, Dog本来就是Animal的一种.

所以在继承关系中, 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做是父类. 但是,反过来就不行.

In [12]:
```
print isinstance(y, Animal)
print isinstance(y, Dog)
```

```
True
False
```

要理解多态的好处, 我们还需要再编写一个函数, 这个函数接受一个Animal类型的变量

In [13]:
```
#示例代码5.

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

run_twice(Animal())
```

```
Animal is running...
Animal is running...
```

In [14]:
```
run_twice(Dog())
```

```
Dog is running...
Dog is running...
```

In [15]:
```
run_twice(Cat())
```

```
Cat is running...
Cat is running...
```

run_twice()做任何修改, 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时, 我们只需要接收Animal类型就可以了, 因为Dog、Cat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.对于一个变量, 我们只需要知道它是Animal类型, 无需确切地知道它的子类型, 就可以放心地调用run()方法, 而具体调用的run()方法是作用在Animal、Dog、还是Cat上, 由运行时该对象的确切类型决定, 这就是多态真正的威力.

多态在静态语言与动态语言中的区别:

对于静态语言(例如Java)来说, 如果需要传入Animal类型, 则传入的对象必须是Animal类型或者它的子类, 否则将无法调用run()方法.

对于Python这样的动态语言来说, 则不一定需要传入Animal类型. 我们只需要保证传入的对象有一个run()方法就可以了.

In [20]:
```
class Timer(object):
    def run(self):
        print('Start...')

run_twice(Timer())
```

```
Start...
Start...
```

这就是动态语言的'鸭子类型',即它并不要求严格的继承体系, 一个对象只要"看起来像鸭子，走起路来像鸭子", 那它就可以被看做是鸭子.