

继承和多态

[2.7旧版教程](#)

阅读: 127282

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：

```
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` 就是它的子类。`Cat` 和 `Dog` 类似。

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 `Animal` 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：

```
dog = Dog()
dog.run()

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

运行结果如下：

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

当然，也可以对子类增加一些方法，比如 `Dog` 类：

```
class Dog(Animal):

    def run(self):
        print('Dog is running...')

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

继承的第二个好处需要我们对代码做一点改进。你看到了，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` 类进行如下修改：

```
class Dog(Animal):

    def run(self):
        print('Dog is running...')

class Cat(Animal):

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

再次运行，结果如下：

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

当子类 and 父类都存在相同的 `run()` 方法时，我们说，子类的 `run()` 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：

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

判断一个变量是否是某个类型可以用 `isinstance()` 判断：

```
>>> isinstance(a, list)
True
>>> isinstance(b, Animal)
True
>>> isinstance(c, Dog)
True
```

看来 `a`、`b`、`c` 确实对应着 `list`、`Animal`、`Dog` 这3种类型。

但是等等，试试：

```
>>> isinstance(c, Animal)
True
```

看来 `c` 不仅仅是 `Dog`，`c` 还是 `Animal`！

不过仔细想想，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 `c` 时，我们认为 `c` 的数据类型是 `Dog` 没错，但 `c` 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：

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

False



`Dog` 可以看成 `Animal`，但 `Animal` 不可以看成 `Dog`。

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：

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

当我们传入 `Animal` 的实例时，`run_twice()` 就打印出：

```
>>> run_twice(Animal())  
Animal is running...  
Animal is running...
```

当我们传入 `Dog` 的实例时，`run_twice()` 就打印出：

```
>>> run_twice(Dog())  
Dog is running...  
Dog is running...
```

当我们传入 `Cat` 的实例时，`run_twice()` 就打印出：

```
>>> run_twice(Cat())  
Cat is running...  
Cat is running...
```

看上去没啥意思，但是仔细想想，现在，如果我们再定义一个 `Tortoise` 类型，也从 `Animal` 派生：

```
class Tortoise(Animal):  
    def run(self):  
        print('Tortoise is running slowly...')
```

当我们调用 `run_twice()` 时，传入 `Tortoise` 的实例：

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

你会发现，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。

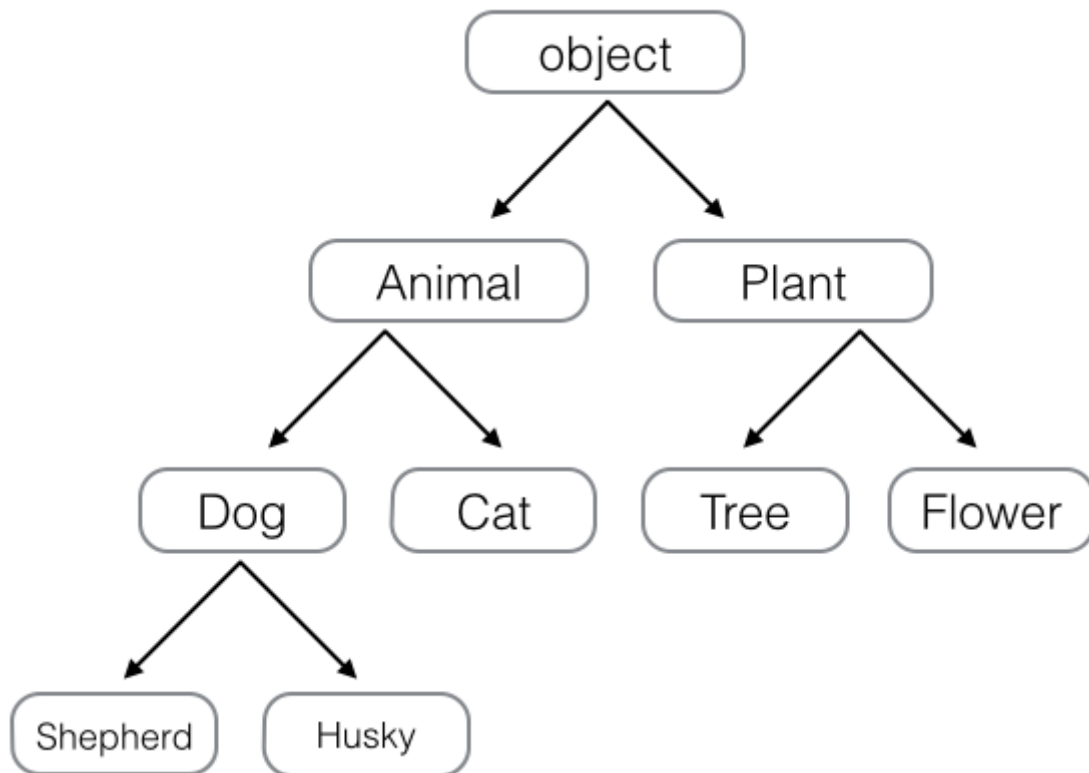
多态的好处就是，当我们需要传入 `Dog`、`Cat`、`Tortoise`时，我们只需要接收 `Animal` 类型就可以了，因为 `Dog`、`Cat`、`Tortoise`都是 `Animal` 类型，然后，按照 `Animal` 类型进行操作即可。由于 `Animal` 类型有 `run()` 方法，因此，传入的任意类型，只要是 `Animal` 类或者子类，就会自动调用实际类型的 `run()` 方法，这就是多态的意思：

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`、`Cat` 还是 `Tortoise` 对象上，由运行时该对象的确切类型决定，这就是多态真正的威力：调用方只管调用，不管细节，而当我们新增一种 `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。

继承还可以一级一级地继承下来，就好比从爷爷到爸爸、再到儿子这样的关系。而任何类，最终都可以追溯到根类 `object`，这些继承关系看上去就像一颗倒着的树。比如如下的继承树：



静态语言 vs 动态语言

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：

```
class Timer(object):
    def run(self):
        print('Start...')
```

这就是动态语言的“鸭子类型”，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。

Python的“file-like object”就是一种鸭子类型。对真正的文件对象，它有一个 `read()` 方法，返回其内容。但是，许多对象，只要有 `read()` 方法，都被视为“file-like object”。许多函数接收的参数就是“file-like object”，你不一定要传入真正的文件对象，完全可以传入任何实现了 `read()` 方法的对象。

小结

继承可以把父类的所有功能都直接拿过来，这样就不必重零做起，子类只需要新增自己特有的方法，也可以把父类不适合的方法覆盖重写。

动态语言的鸭子类型特点决定了继承不像静态语言那样是必须的。

感觉本站内容不错，读后有收获？


¥ 我要小额赞助，鼓励作者写出更好的教程

还可以分享给朋友

分享 赵岩、 、 蓝人可 等5人分享过



珠峰NODE.JS全栈开发



技术陪伴成长社区 珠峰培训

麦子学院 www.maiziedu.com

百万级python导师亲身指导

保你120天
变身python大牛

有时候，你需要的只是一句点拨

立即咨询

掘金

一个只有高手分享
的技术社区

立即加入

深度学习在线课程
通向无人驾驶的必经之路

评论

发表评论

Sign In to Make a Comment

