**python学习笔记6.3-类的属性函数(@property)**

2016年12月18日 13:04:51

阅读数：30317

**属性函数(@property)**   
在对象中两个很重要的元素就是属性和方法，在调用的时候两者是有区别的。

class People:

def \_\_init\_\_(self,first\_name,last\_name):

self.first\_name = first\_name

self.last\_name = last\_name

def get\_first\_name(self):

return self.first\_name

a = People('lala','ouyang')

print(a.get\_first\_name())

print(a.first\_name)

从例子中我们可以发现，一样的结果，但是调用的过程不一样（虽然其实也就是多一个括号而已），那么有没有一种办法，使得我们调用属性的时候就会自动调用相应的方法，也就是增加一些额外的处理过程（例如类型检查或者验证）。这时候属性函数（@property）就能给我们提供很好的解决方案。   
首先是最简单的例子，自动调用get，set，del函数对属性的处理。

class People:

def \_\_init\_\_(self,name):

self.name = name

#getter function

@property #属性函数

def name(self):

return self.\_name

#setter function

@name.setter

def name(self,name):

self.\_name = name

#deleter function

@name.deleter

def name(self):

raise AttributeError('Can not delete the name')

a = People('leida')

print(a.name) #calling the getter function automatic when we get the attribute

a.name = 'libai' #Calling the setter function automatic when we set the attribute

print(a.name)

del a.name #Calling the deleter function automatic when we delete the attribute

leida

Traceback (most recent call last):

libai

File "D:/home/WX/test\_clsaa.py", line 65, in <module>

del a.name #Calling the deleter function automatic when we delete the attribute

File "D:/home/WX/test\_clsaa.py", line 59, in name

raise AttributeError('Can not delete the name')

AttributeError: Can not delete the name

正如例子中这样。要定义对属性的访问，一种最简单的方法就是将其定义为property。比如说，增加对属性的类型检查：

class People:

def \_\_init\_\_(self,name):

self.name = name

#getter function

@property #属性函数

def name(self):

return self.\_name

#setter function

@name.setter

def name(self,name):

if not isinstance(name,str):

raise TypeError('name must is string type')

self.\_name = name

#deleter function

@name.deleter

def name(self):

raise AttributeError('Can not delete the name')

a = People(12) #calling the setter function automatic when create the instance

Traceback (most recent call last):

File "D:/home/WX/test\_clsaa.py", line 63, in <module>

a = People(12) #calling the setter function automatic when create the instance

File "D:/home/WX/test\_clsaa.py", line 45, in \_\_init\_\_

self.name = name

File "D:/home/WX/test\_clsaa.py", line 55, in name

raise TypeError('name must is string type')

TypeError: name must is string type

当我们实现一个property时，底层数据仍然需要保存在某个地方，因而在get和set的方法中，可以看到直接对\_name操作的，这就是数据实际保存的地方。但是，也发现在**init**()方法中任然是对self.name操作的。但是实际情况是我初始化的时候程序仍旧是对self.\_name操作的。(这点我也还不理解，应该不是这样的啊.但是必须这么写，不然会报错)。   
对于已经存在的get，set，del方法，同样也可以定义为property：

class People:

def \_\_init\_\_(self,name):

self.name = name

def get\_name(self):

print('calling the get function')

return self.\_name

def set\_name(self,name):

print('calling the set function')

self.\_name = name

def del\_name(self):

del self.\_name

name = property(get\_name,set\_name,del\_name)

a = People('libai') #calling the setter function automatic when create the instance

* 1同时，在set\_name 函数中做了打印标记，发现在**init**()方法中确实调用了set\_name()函数。

Property属性实际上就是把一系列的方法绑定到一起。如果检查类的property属性，就会发现property自身所持有的属性fget,fset,fdel所代表的原始方法。

print(People.name.fget)

print(People.name.fset)

print(People.name.fdel)

一般来说，我们不会直接去调用fset或者fget，但是当我们调用property属性时会自动触发对这些方法的调用。   
上面例子中的两种写法，一般倾向于第二种写法，特别是如果需要对某个普通的属性额外增加处理步骤时，可以在不修改已有代码的情况下将这个属性提升为一个property。   
Property也可以用来定义需要计算的属性，这类属性并不会实际被保存起来，而是根据需要完成计算。

import math

class Circle:

def \_\_init\_\_(self,radius):

self.radius = radius

@property

def area(self):

return math.pi \* self.radius\*\*2

@property

def perimeter(self):

return 2\*math.pi\*self.radius

c = Circle(5)

print(c.area) #do not need (): c.area instead of c.area()

print(c.perimeter)

78.53981633974483

31.41592653589793

这样的写法就会使得实例的接口变得统一，本来用方法实现的计算调用的时候用属性就可以，很好的避免了方法、属性傻傻分不清的情况了。   
个人建议，不要在代码中不断重复使用@property，这样会使得代码变得臃肿，而且难以阅读，容易出错。同样的任务，**利用描述符或者闭包也能够很好的完成**（以后会详细解释）。