

INDEX



Python教程

Python简介

安装Python

第一个Python程序

Python基础

函数

高级特性

函数式编程

模块

面向对象编程

面向对象高级编程

使用__slots__

使用@property

多重继承

定制类

使用枚举类

使用元类

错误、调试和测试

IO编程

进程和线程

正则表达式

常用内建模块

常用第三方模块

virtualenv

图形界面

网络编程

电子邮件

访问数据库

Web开发

异步IO

实战

FAQ

期末总结

关于作者



廖雪峰 北京 朝阳区

+ 加关注

使用@property

Reads: 10497642

在绑定属性时，如果我们直接把属性暴露出去，虽然写起来很简单，但是，没办法检查参数，导致可以把成绩随便改：

```
s = Student()
s.score = 9999
```

这显然不合逻辑。为了限制score的范围，可以通过一个set_score()方法来设置成绩，再通过一个get_score()来获取成绩，这样，在set_score()方法里，就可以检查参数：

```
class Student(object):

    def get_score(self):
        return self._score

    def set_score(self, value):
        if not isinstance(value, int):
            raise ValueError('score must be an integer!')
        if value < 0 or value > 100:
            raise ValueError('score must between 0 ~ 100!')
        self._score = value
```

现在，对任意的Student实例进行操作，就不能随心所欲地设置score了：

```
>>> s = Student()
>>> s.set_score(60) # OK!
>>> s.get_score()
60
>>> s.set_score(9999)
Traceback (most recent call last):
...
ValueError: score must between 0 ~ 100!
```

但是，上面的调用方法又略显复杂，没有直接用属性这么直接简单。

有没有既能检查参数，又可以用类似属性这样简单的方式来访问类的变量呢？对于追求完美的Python程序员来说，这是必须要做到的！

还记得装饰器（decorator）可以给函数动态加上功能吗？对于类的方法，装饰器一样起作用。Python内置的@property装饰器就是负责把一个方法变成属性调用的：

```
class Student(object):

    @property
    def score(self):
        return self._score

    @score.setter
    def score(self, value):
        if not isinstance(value, int):
            raise ValueError('score must be an integer!')
        if value < 0 or value > 100:
            raise ValueError('score must between 0 ~ 100!')
        self._score = value
```

@property的实现比较复杂，我们先考察如何使用。把一个getter方法变成属性，只需要加上@property就可以了，此时，@property本身又创建了另一个装饰器@score.setter，负责把一个setter方法变成属性赋值，于是，我们就拥有一个可控的属性操作：

```
>>> s = Student()
>>> s.score = 60 # OK, 实际转化为s.set_score(60)
>>> s.score # OK, 实际转化为s.get_score()
60
>>> s.score = 9999
Traceback (most recent call last):
...
ValueError: score must between 0 ~ 100!
```

注意到这个神奇的@property，我们在对实例属性操作的时候，就知道该属性很可能不是直接暴露的，而是通过getter和setter方法来实现的。

还可以定义只读属性，只定义getter方法，不定义setter方法就是一个只读属性：

```
class Student(object):

    @property
    def birth(self):
        return self._birth

    @birth.setter
    def birth(self, value):
        self._birth = value

    @property
    def age(self):
        return 2015 - self._birth
```

上面的birth是可读写属性，而age就是一个只读属性，因为age可以根据birth和当前时间计算出来。

小结

@property广泛应用在类的定义中，可以让调用者写出简短的代码，同时保证对参数进行必要的检查，这样，程序运行时就减少了出错的可能性。

练习

请利用@property给一个Screen对象加上width和height属性，以及一个只读属性resolution：

```
# -*- coding: utf-8 -*-

class Screen(object):
    pass


# 测试:
s = Screen()
s.width = 1024
s.height = 768
print('resolution =', s.resolution)
if s.resolution == 786432:
    print('测试通过!')
else:
    print('测试失败!')
```

Run

参考源码

use_property.py

读后有收获可以支付宝请作者喝咖啡，读后有疑问请加微信群讨论：



还可以分享给朋友：

分享到微博

Previous Page

Next Page

Comments

Make a comment

Sign in to make a comment

