Python Decorator 装饰器

刘鹏

2020 年 6 月 25 日

1 Python 装饰器

装饰器可以让一个 Python 函数拥有原本没有的功能，也就是你可以通过装饰器，让一个平淡无奇的函数变的强大，变的漂亮。举几个现实中的例子

1. 你一个男的程序员，穿上女装，戴上假发，你就有了女人的外表（穿女装、戴假发的过程就是新的特效，你拥有了女人的外表，你原来的小 jj 还在，没有消失）
2. 你新买的毛坯房，装修，买家具后变好看了（装修、家具就是新的特效）
3. 孙悟空被放进炼丹炉装饰了一下，出来后，学会了火眼金睛，以前的本领都还在

作者：python 教程链接：https://www.zhihu.com/question/271201015/answer/360295993

[1]:
```
def 炼丹炉(func): # func 就是‘孙悟空’这个函数

    #*args, **kwargs 就是‘孙悟空’的参数列表，这里的‘孙悟空’函数没有传参数，我们写上也不影响，建议都写上
    def 变身(*args, **kwargs):

        # 加特效，增加新功能，比如孙悟空的进了炼丹炉后，有了火眼金睛技能
        print('有火眼金睛了')

        # 保留原来的功能，原来孙悟空的技能，如吃桃子
        return func(*args, **kwargs)
    # 炼丹成功，更强大的，有了火眼金睛技能的孙悟空出世
    return 变身

@ 炼丹炉
def 孙悟空():
    print('吃桃子')
```

```
孙悟空 ()
# 输出:
# 有火眼金睛了
# 吃桃子
```

有火眼金睛了
吃桃子

[2]:
```
#
def AddNewFeature(func):
    def NewFeature(*args, **kwargs):
        if (args[0] > args[1]):
            func(*args, **argv)
        else:
            print(args[0] - args[1])
    return NewFeature

@AddNewFeature
def calculate(a, b):
    print(a + b)

calculate(1, 2)
```

-1

1.1 多装饰器的执行流

接下来展示一个很简单的多装饰器执行流。

[3]:
```
def 炼丹炉(func):
    def 变身(*args, **kwargs):
        print('3 有火眼金睛了')
        return func(*args, **kwargs)
    return 变身

def 龙宫走一趟(func):
    def 你好(*args, **kwargs):
        print('2 有金箍棒了')
```

```
        return func(*args, **kwargs)
    return 你好

def 拜师学艺(func):
    def 师傅(*args, **kwargs):
        print('1 学会飞、72 变了')
        return func(*args, **kwargs)
    return 师傅

@ 拜师学艺
@ 龙宫走一趟
@ 炼丹炉
def 孙悟空():
    print('吃桃子')

孙悟空 ()
# 输出
# 学会飞、72 变了
# 有金箍棒了
# 有火眼金睛了
# 吃桃子
```

```
1 学会飞、72 变了
2 有金箍棒了
3 有火眼金睛了
吃桃子
```

接下来展示一个简单的装饰器实例。

当函数被装饰器修饰之后，运行该函数时，解释器会**直接把控制流交给装饰器函数**。

- 为了保证装饰器函数的行为与原函数差不太多，应该要保证**装饰器函数的返回值与原函数一致**
- 注意传递参数，否则装饰器无法调用原函数

  参考视频：https://www.bilibili.com/video/BV11s411V7Dt

[4]:
```
import time
```

```python
# 装饰器函数
def displayTime(func):

    # wrapper 是装饰器的具体内容
    def wrapper(*args):
        t1 = time.time()
        # *args 是一个列表，直接用就可以
        result = func(*args)
        t2 = time.time()
        print("总计运行时间为: {:.4} s".format(t2 - t1))
        return result
    return wrapper

def isPrime(num):
    if (num < 2):
        return False
    elif (num == 2):
        return True
    else:
        for i in range(2, num):
            if (num % i == 0):
                return False
            else:
                return True

@displayTime
def countPrimeNums(maxNum):
    count = 0
    for i in range(2, maxNum):
        if (isPrime(i)):
            count = count + 1
    return count

countPrimeNums(5000)
```

```
总计运行时间为: 0.00299 s
```

[4]: 2500

装饰器其实就是函数的继承？至少从形式上看起来很像。

[5]:
```
import time

# 装饰器函数
def displayTime(func):
        # wrapper 是装饰器的具体内容
    def wrapper(*args):
        t1 = time.time()
        # *args 是一个列表，直接用就可以
        result = func(*args)
        t2 = time.time()
        print("总计运行时间为: {:.4} s".format(t2 - t1))
        return result

    def wrapper2(args):
        print(args)

    # 这个返回值是与某个函数相对应的，该返回值将决定这个装饰器具体选择哪个函数作为装饰器
    return wrapper



def isPrime(num):
    if (num < 2):
        return False
    elif (num == 2):
        return True
    else:
        for i in range(2, num):
            if (num % i == 0):
                return False
            else:
                return True
```

```
@displayTime
def countPrimeNums(maxNum):
    count = 0
    for i in range(2, maxNum):
        if (isPrime(i)):
            count = count + 1
    return count

countPrimeNums(5000)
```

总计运行时间为: 0.001993 s

[5]: 2500