随笔- 33  文章- 127  评论- 280

[**python之路——装饰器函数**](https://www.cnblogs.com/Eva-J/articles/7194277.html)

**阅读目录**

* **[楔子](https://www.cnblogs.com/Eva-J/articles/7194277.html" \l "_label1)**
* [**装饰器的形成过程**](https://www.cnblogs.com/Eva-J/articles/7194277.html#_label2)
* [**开放封闭原则**](https://www.cnblogs.com/Eva-J/articles/7194277.html#_label3)
* [**谈装饰器主要功能和装饰器固定结构**](https://www.cnblogs.com/Eva-J/articles/7194277.html#_label4)
* [**带参数的装饰器**](https://www.cnblogs.com/Eva-J/articles/7194277.html#_label5)
* [**多个装饰器装饰一个函数**](https://www.cnblogs.com/Eva-J/articles/7194277.html#_label6)

**楔子**

作为一个会写函数的python开发，我们从今天开始要去公司上班了。写了一个函数，就交给其他开发用了。

def func1():

print('in func1')

季度末，公司的领导要给大家发绩效奖金了，就提议对这段日子所有人开发的成果进行审核，审核的标准是什么呢？就是统计每个函数的执行时间。

这个时候你要怎么做呀？

你一想，这好办，把函数一改：

[复制代码](javascript:void(0);)

import time

def func1():

start = time.time()

print('in func1')

print(time.time() - start)

func1()

[复制代码](javascript:void(0);)

来公司半年，写了2000+函数，挨个改一遍，1个礼拜过去了，等领导审核完，再挨个给删了。。。又1个礼拜过去了。。。这是不是很闹心？

你觉得不行，不能让自己费劲儿，告诉所有开发，现在你们都在自己原本的代码上加上一句计算时间的语句？

[复制代码](javascript:void(0);)

import time

def func1():

print('in func1')

start = time.time()

func1()

print(time.time() - start)

[复制代码](javascript:void(0);)

还是不行，因为这样对于开发同事来讲实在是太麻烦了。

那怎么办呢？你灵机一动，写了一个timer函数。。。

[复制代码](javascript:void(0);)

import time

def timer(func):

start = time.time()

func()

print(time.time() - start)

def func1():

print('in func1')

def func2():

print('in func2')

timer(func1)

timer(func2)

[复制代码](javascript:void(0);)

这样看起来是不是简单多啦？不管我们写了多少个函数都可以调用这个计时函数来计算函数的执行时间了。。。尽管现在修改成本已经变得很小很小了，但是对于同事来说还是改变了这个函数的调用方式，假如某同事因为相信你，在他的代码里用你的方法用了2w多次，那他修改完代码你们友谊的小船也就彻底地翻了。

你要做的就是，让你的同事依然调用func1，但是能实现调用timer方法的效果。

[复制代码](javascript:void(0);)

import time

def timer(func):

start = time.time()

func()

print(time.time() - start)

def func1():

print('in func1')

func1 =timer #要是能这样的就完美了。。。可惜报错

func1()

[复制代码](javascript:void(0);)

非常可惜，上面这段代码是会报错的，因为timer方法需要传递一个func参数，我们不能在赋值的时候传参，因为只要执行func1 = timer(func1)，timer方法就直接执行了，下面的那句func1根本就没有意义。到这里，我们的思路好像陷入了僵局。。。

[返回顶部](https://www.cnblogs.com/Eva-J/articles/7194277.html" \l "_labelTop)

**装饰器的形成过程**

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

import time

def func1():

print('in func1')

def timer(func):

def inner():

start = time.time()

func()

print(time.time() - start)

return inner

func1 = timer(func1)

func1()

[复制代码](javascript:void(0);)

忙活了这么半天，终于初具规模了！现在已经基本上完美了，唯一碍眼的那句话就是还要在做一次赋值调用。。。

你觉得碍眼，python的开发者也觉得碍眼，所以就为我们提供了一句语法糖来解决这个问题！

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

import time

def timer(func):

def inner():

start = time.time()

func()

print(time.time() - start)

return inner

@timer #==> func1 = timer(func1)

def func1():

print('in func1')

func1()

[复制代码](javascript:void(0);)

到这里，我们可以简单的总结一下：

　　装饰器的本质：一个闭包函数

　　装饰器的功能：在不修改原函数及其调用方式的情况下对原函数功能进行扩展

还有最后一个问题要解决，刚刚我们讨论的装饰器都是装饰不带参数的函数，现在要装饰一个带参数的函数怎么办呢？

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

def timer(func):

def inner(a):

start = time.time()

func(a)

print(time.time() - start)

return inner

@timer

def func1(a):

print(a)

func1(1)

[复制代码](javascript:void(0);)

其实装饰带参的函数并不是什么难事，但假如你有两个函数，需要传递的参数不一样呢？

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

import time

def timer(func):

def inner(\*args,\*\*kwargs):

start = time.time()

re = func(\*args,\*\*kwargs)

print(time.time() - start)

return re

return inner

@timer #==> func1 = timer(func1)

def func1(a,b):

print('in func1')

@timer #==> func2 = timer(func2)

def func2(a):

print('in func2 and get a:%s'%(a))

return 'fun2 over'

func1('aaaaaa','bbbbbb')

print(func2('aaaaaa'))

[复制代码](javascript:void(0);)

现在参数的问题已经完美的解决了，可是如果你的函数是有返回值的呢？

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

import time

def timer(func):

def inner(\*args,\*\*kwargs):

start = time.time()

re = func(\*args,\*\*kwargs)

print(time.time() - start)

return re

return inner

@timer #==> func2 = timer(func2)

def func2(a):

print('in func2 and get a:%s'%(a))

return 'fun2 over'

func2('aaaaaa')

print(func2('aaaaaa'))

[复制代码](javascript:void(0);)

刚刚那个装饰器已经非常完美了，但是正常我们情况下查看函数的一些信息的方法在此处都会失效

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

def index():

'''这是一个主页信息'''

print('from index')

print(index.\_\_doc\_\_) #查看函数注释的方法

print(index.\_\_name\_\_) #查看函数名的方法

为了不让他们失效，我们还要在装饰器上加上一点来完善它：

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

from functools import wraps

def deco(func):

@wraps(func) #加在最内层函数正上方

def wrapper(\*args,\*\*kwargs):

return func(\*args,\*\*kwargs)

return wrapper

@deco

def index():

'''哈哈哈哈'''

print('from index')

print(index.\_\_doc\_\_)

print(index.\_\_name\_\_)

[复制代码](javascript:void(0);)

[返回顶部](https://www.cnblogs.com/Eva-J/articles/7194277.html" \l "_labelTop)

**开放封闭原则**

　　1.对扩展是开放的

　　　　为什么要对扩展开放呢？

　　　　我们说，任何一个程序，不可能在设计之初就已经想好了所有的功能并且未来不做任何更新和修改。所以我们必须允许代码扩展、添加新功能。

　　2.对修改是封闭的

　　　　为什么要对修改封闭呢？

　　　　就像我们刚刚提到的，因为我们写的一个函数，很有可能已经交付给其他人使用了，如果这个时候我们对其进行了修改，很有可能影响其他已经在使用该函数的用户。

装饰器完美的遵循了这个开放封闭原则。

[返回顶部](https://www.cnblogs.com/Eva-J/articles/7194277.html" \l "_labelTop)

**装饰器的主要功能和装饰器的固定结构**

**装饰器的主要功能：**

在不改变函数调用方式的基础上在函数的前、后添加功能。

**装饰器的固定格式：**

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

def timer(func):

def inner(\*args,\*\*kwargs):

'''执行函数之前要做的'''

re = func(\*args,\*\*kwargs)

'''执行函数之后要做的'''

return re

return inner

[复制代码](javascript:void(0);)

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

from functools import wraps

def deco(func):

@wraps(func) #加在最内层函数正上方

def wrapper(\*args,\*\*kwargs):

return func(\*args,\*\*kwargs)

return wrapper

[复制代码](javascript:void(0);)

[返回顶部](https://www.cnblogs.com/Eva-J/articles/7194277.html" \l "_labelTop)

**带参数的装饰器**

假如你有成千上万个函数使用了一个装饰器，现在你想把这些装饰器都取消掉，你要怎么做？

一个一个的取消掉？ 没日没夜忙活3天。。。

过两天你领导想通了，再让你加上。。。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

def outer(flag):

def timer(func):

def inner(\*args,\*\*kwargs):

if flag:

print('''执行函数之前要做的''')

re = func(\*args,\*\*kwargs)

if flag:

print('''执行函数之后要做的''')

return re

return inner

return timer

@outer(False)

def func():

print(111)

func()

[复制代码](javascript:void(0);)

[返回顶部](https://www.cnblogs.com/Eva-J/articles/7194277.html" \l "_labelTop)

**多个装饰器装饰同一个函数**

有些时候，我们也会用到多个装饰器装饰同一个函数的情况。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

def wrapper1(func):

def inner():

print('wrapper1 ,before func')

func()

print('wrapper1 ,after func')

return inner

def wrapper2(func):

def inner():

print('wrapper2 ,before func')

func()

print('wrapper2 ,after func')

return inner

@wrapper2

@wrapper1

def f():

print('in f')

f()

[复制代码](javascript:void(0);)