

python装饰器以及最终写法

```
1 """
2 装饰器形成 与 固定模式
3 """
4 import time
5
6 # 比如
7 # def func1():
8 #     print('我在编写代码')
9 #
10 # func1()
11
12 # 现在你要计算代码的时间，那可能第一次编写的时候，就嵌入到函数中 这个代码就被入侵了
13 # def func1():
14 #     start = time.time()
15 #     print('我在编写代码')
16 #     time.sleep(0.01)
17 #     end = time.time()
18 #     print(end-start)
19 #
20 # func1()
21
22 #装饰器的本质
23
24 #本身方法
25 def func():
26     time.sleep(1)
27     print('hahaha')
28
29 #装饰器
30 def wrapper(f):
31     def inner():
32         start = time.time()
33         f()
34         end = time.time()
35         print(end-start)
36     return inner
37
38 #等于具体实现 可以看到本体执行了，然后装饰对象所附加的功能也做了
39 func = wrapper(func)
40 func()
41
42
43
44
45
46
47
48
49
```

50
51
52

```
-----  
1 # 装饰器的作用  
2 # 额外的为本体添加一些附加的功能  
3 #python 使用语法糖  
4 import time  
5  
6 #装饰器  
7 def wrapper(f):  
8     def inner():  
9         start = time.time()  
10        f()  
11        end = time.time()  
12        print(end-start)  
13    return inner  
14  
15  
16 #本身方法  
17 @wrapper  
18 def func():  
19     time.sleep(1)  
20     print('hahaha')  
21  
22  
23  
24 func()  
25  
26 """  
27     1.申明装饰器  
28     2.标记需要修饰的函数  
29     3.直接使用函数调用  
30  
31 """
```

```
-----  
1 # 装饰器完整  
2 import time  
3  
4 """  
5 *args: (表示的就是将实参中按照位置传值, 多出来的值都给args, 且以元祖的方式呈现)  
6  
7 **kwargs: (表示的就是形参中按照关键字传值把多余的传值以字典的方式呈现)  
8  
9 """  
10  
11 # 1.定义装饰器 带参数, 待返回值  
12  
13 def wrapper(f):  
14     def innner(*args,**kwargs):  
15         print("妈耶")  
16         ret = f(*args,**kwargs)  
17         print("天耶")
```

```
18         return ret
19     return inner
20
21
22 # 定义本体方法 并且使用装饰器
23 @wrapper
24 def abc(n1,n2):
25     return n1+n2
26
27
28 print(abc(1,2))
```