python装饰器以及最终写法

```python
"""
装饰器形成 与  固定模式
"""
import  time

# 比如
# def func1():
#     print('我在编写代码')
#
# func1()

# 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
# def func1():
#     start = time.time()
#     print('我在编写代码')
#     time.sleep(0.01)
#     end = time.time()
#     print(end-start)
#
# func1()

#装饰器的本质

#本身方法
def func():
    time.sleep(1)
    print('hahaha')

#装饰器
def wrapper(f):
    def inner():
        start = time.time()
        f()
        end = time.time()
        print(end-start)
    return inner

#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
func = wrapper(func)
func()
```

```

```

```
# 装饰器的作用
# 额外的为本体添加一些附加的功能
#python 使用语法糖
import  time

#装饰器
def wrapper(f):
    def inner():
        start = time.time()
        f()
        end = time.time()
        print(end-start)
    return inner


#本身方法
@wrapper
def func():
    time.sleep(1)
    print('hahaha')



func()

"""
    1.申明装饰器
    2.标记需要修饰的函数
    3.直接使用函数调用

"""
```

```
#  装饰器完整
import  time

"""
*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)

**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)

"""

#  1.定义装饰器 带参数，待返回值

def wrapper(f):
    def innner(*args,**kwargs):
        print("妈耶")
        ret = f(*args,**kwargs)
        print("天耶")
```

```
        return ret
    return innner


# 定义本体方法 并且使用装饰器
@wrapper
def abc(n1,n2):
    return n1+n2


print(abc(1,2))
```