装饰器

笔记本:	git基础		
创建时间:	2018/5/14 8:54	**更新时间:**	2018/5/14 8:55
作者:	王伟		

装饰器：在不修改原函数的代码的情况下，添加新的功能。可以在执行原函数之前加，也可以在执行原函数之后添加。
所有的函数名，类名都是变量名。
开放封闭的原则：就是原函数不要动，添加一些新的功能；可以通过装饰器来实现。

```
def set_func(func):
    def call_func():
        print("这是新添加的 权限验证功能")
        func()
        print("这是新添加的 log日志功能")
    return call_func


@set_func  # 等价于 test = set_func(test)
def test():
    print("------test-----")


# class T(object):
#     @staticmethod  # test = staticmethod(test)
#     def test():
#         pass


# a = set_func(test)
# a()
# test = set_func(test)
# test()


test()

# 1. 装饰器只能在调用原函数之前 或者之后 添加功能，而不能在函数的中间添加功能
# 2. 只要用装饰器装饰了的函数，那么不管被调用多少次，都是装饰之后的效果


def set_func(func):
    def call_func(*args, **kwargs):
        print("这是新添加的 权限验证功能")
        ret = func(*args, **kwargs)  # 想一想：如果func(args, kwargs)意味着什么?
        print("这是新添加的 log日志功能")
```

```
        return ret
    return call_func


@set_func
def test1():
    """没有参数、没有返回值"""
    print("hello python-----100---")


@set_func
def test2():
    """没有参数、有返回值"""
    print("hello python____200-----")
    return 200


@set_func
def test3(num1, num2):
    """有参数、没有返回值"""
    print("num1=%d, num2=%d" % (num1, num2))


@set_func
def test4(num):
    """有参数、有返回值"""
    print("num=%d" % num)
    return 400
```

装饰器不会根据你调用不调用执不执行，只要有装饰器@#装饰器其实已经开始执行了。调用是执行功能，装饰是看代码@#。

多个装饰器对同一个函数进行装饰

装饰的时候：先装饰set_log然后再装饰set_pro。(通俗的说：谁离被装饰的函数跃进，那么就先装谁。)
调用的时候：谁离的远，就先调用谁。

```
def set_pro(func):
    print("装饰器开始装饰(权限的功能).....")
    def call_func(*args, **kwargs):
        print("这是新添加的 权限验证功能")
        ret = func(*args, **kwargs)
        return ret
    return call_func
```

```
def set_log(func):
    print("装饰器开始装饰(log日志的功能).....")
    def call_func(*args, **kwargs):
        print("这是新添加的 log日志的功能")
        ret = func(*args, **kwargs)
        return ret
    return call_func


@set_pro  # 添加权限的装饰器
@set_log  # 添加log日志的装饰器
def test1():
    """没有参数、没有返回值"""
    print("hello python-----100---")


print("-"*30)

test1()
```

带有参数的装饰器

```
import time


def set_log(log_level):  # 最外边的函数 用来接收 装饰器在装饰时候的参数
    def set_func(func):  # 中间的函数，用来接收 需要装饰的原函数的引用
        def call_func(*args, **kwargs):  # 最里面的函数，用来调用被装饰的原函数

            log_level_dict = {
                1: "warning",
                2: "error"
            }

            # print("要添加的新的功能--------")
            with open("./log.txt", "a") as f:
                f.write("--(%s)--%s---调用了(%s)函数\n" % (log_level_dict[log_level], time.ctime(),
func.__name__))
            return func()
        return call_func
    return set_func

# 带有参数的装饰器，装饰的过程如下
# 1. 先去调用set_log函数，并且把1当做实参进行传递到log_level这个变量中
# 2. 用set_log返回的函数 对 test进行装饰，装饰的过程与之前的方式一样
@set_log(1)
```

```
def test():
    print("-------test----")


@set_log(2)
def test2():
    print("-------test2----")


test()
test2()
```

```
>>>def xxyy():
    pass
>>>xxyy.__name__
>>>"xxyy"
```

```
>>>def test(temp):
    print(temp.__name__)
>>>test(xxyy)
>>>xxyy
```

这个变量指向某个函数时，这个变量.__name__得出的结果就是指向的函数名。

类当做装饰器：

```
class Log(object):
    def __init__(self, func):
        self.func = func

    def __call__(self):
        print("----call----被调用了....")
        self.func()


@Log  # 等价于test = Log(test)  # 想一想：如果是@Log(1) 怎样理解这个装饰器呢？提示：按照带有参数的装饰的流程来看
def test():
    print("-------test----")


test()  # 调用__call__方法
```