链滴

Python3 查缺补漏：2、装饰器

作者：zhaolixiang

原文链接：https://ld246.com/article/1566875364115

来源网站：链滴

装饰器功能：

1. 引入日志
2. 函数执行时间统计
3. 执行函数前预备处理
4. 执行函数后清理功能
5. 权限校验
6. 缓存

1、无参数函数的装饰器

实例：

```
from time import ctime,sleep
def time_fun(func):
    #内部包裹函数
    def wrapped_fun():
        #ctime():打印当前时间
        print("%s 在 %s  时被调用"%(func.__name__,ctime()))
        #执行函数
        func()
    #把内部嵌套函数作为对象返回
    return wrapped_fun

@time_fun
def test():
    print("test 执行了")

test()
#休眠3秒
sleep(3)
test()
```

结果：

```
test 在 Wed Aug 15 22:19:51 2018 时被调用
test 执行了
test 在 Wed Aug 15 22:19:53 2018 时被调用
test 执行了
```

2、有参数函数的装饰器

实例：

```
from time import ctime,sleep
def time_fun(func):
    #内部包裹函数
    def wrapped_fun(a,b):
        #ctime():打印当前时间
        print("%s 在 %s  时被调用"%(func.__name__,ctime()))
```

```
        #执行函数执行
        func(a,b)
    #把内部嵌套函数作为对象返回
    return wrapped_fun

@time_fun
def test(a,b):
    print(a+b)

test(1,2)
#休眠3秒
sleep(3)
test(3,4)
```

结果：

```
test 在 Wed Aug 15 22:23:07 2018 时被调用
3
test 在 Wed Aug 15 22:23:10 2018 时被调用
7
```

3、不定长函数的装饰器

实例：

```
from time import ctime,sleep
def time_fun(func):
    #内部包裹函数
    def wrapped_fun(*args,**kwargs):
        #ctime():打印当前时间
        print("%s 在 %s  时被调用"%(func.__name__,ctime()))
        #执行函数执行
        func(*args,**kwargs)
    #把内部嵌套函数作为对象返回
    return wrapped_fun

@time_fun
def test(a,b,c):
    print(a+b+c)

test(1,2,3)
#休眠3秒
sleep(3)
test(3,4,5)
```

结果：

```
test 在 Wed Aug 15 22:26:36 2018 时被调用
6
test 在 Wed Aug 15 22:26:39 2018 时被调用
12
```

###4、带返回值函数的装饰器

实例：

```
from time import ctime,sleep
def time_fun(func):
    #内部包裹函数
    def wrapped_fun(*args,**kwargs):
        #ctime():打印当前时间
        print("%s 在 %s  时被调用"%(func.__name__,ctime()))
        #执行函数执行
        return func(*args,**kwargs)
    #把内部嵌套函数作为对象返回
    return wrapped_fun

@time_fun
def test(a,b,c):
    print("test--",a+b+c)

@time_fun
def test2(a,b,c):
    return a+b+c

test(1,2,3)
print(test2(1,2,3))
#休眠3秒
sleep(3)
test(1,2,3)
print(test2(3,4,5))
```

结果：

```
test 在 Wed Aug 15 22:31:14 2018 时被调用
test-- 6
test2 在 Wed Aug 15 22:31:14 2018 时被调用
6
test 在 Wed Aug 15 22:31:17 2018 时被调用
test-- 6
test2 在 Wed Aug 15 22:31:17 2018 时被调用
12
```

5、装饰器带有参数

实例：

```
from time import ctime,sleep
def time_fun_pre(pre="hello"):
    def time_fun(func):
        # 内部包裹函数
        def wrapped_fun(*args, **kwargs):
            # ctime():打印当前时间
            print("%s 在 %s  时被调用,pre参数为：  %s" % (func.__name__, ctime(),pre))
            # 执行函数执行
            return func(*args, **kwargs)

        # 把内部嵌套函数作为对象返回
```

```
        return wrapped_fun
    return time_fun

@time_fun_pre("mark_test")
def test(a,b,c):
    print("test--",a+b+c)

@time_fun_pre("mark_test2")
def test2(a,b,c):
    return a+b+c

test(1,2,3)
print(test2(1,2,3))
#休眠3秒
sleep(3)
test(1,2,3)
print(test2(3,4,5))
```

结果：

```
test 在 Wed Aug 15 22:43:27 2018 时被调用,pre参数为：mark_test
test-- 6
test2 在 Wed Aug 15 22:43:27 2018 时被调用,pre参数为：mark_test2
6
test 在 Wed Aug 15 22:43:30 2018 时被调用,pre参数为：mark_test
test-- 6
test2 在 Wed Aug 15 22:43:30 2018 时被调用,pre参数为：mark_test2
12
```

6、类装饰器

python类装饰性必须要接受一个callable对象作为参数，然后返回一个callable对象，在python中，般callable对象都是函数，

只要对象重写了__call__()方法，那么这个对象就是callable对象。

实例：

```
class Test():
    def __init__(self,func):
        print("test初始化：",func.__name__)
        self.func=func

    def __call__(self, *args, **kwargs):
        print("我调用了")
        self.func
@Test
def test():
    print("--test--")
test()
```

结果：

```
test初始化： test
我调用了
```