装饰器拓展

装饰器

装饰器的存在是为了实现开放封闭原则：
· 封闭： 已实现的功能代码块不应该被修改；
· 开放： 对现有功能的扩展开放。
理解装饰器的三要素：
· 函数的作用域
· 高阶函数
· 闭包

1. 闭包

闭包定义：如果在一个内部函数里，对在外部作用域（但不是在全局作用域）的变量进行引用，那么内部函数就被认为是闭包（closure）

```
def test_1(x = 20):
    def test_2():                                          # 条件一：test_2就是内部函数
        print(x)                                           # 条件二： 对外部函数变量的引用
    retrn test_2()                                        # 结论： 内部函数test_2就是一个闭包
test_1()
```

2. 装饰器的应用

现在有如下一段代码，需要测算出代码的执行时间。

```
def foo():
    print('foo...')
foo()
```

对功能进行扩展：
由于代码执行太快，使用sleep暂停2s，此次修改没有执行开放封闭原则，修改了原代码。

```
import timedef foo():
```

```
    start = time.time()
    print('foo...')
    time.sleep(2)
    end = time.time()
    print(end - start)

foo()
```

此次修改改变了调用方式，差评！

```
import timedef foo():
    print('foo...')
def show_time(func):
    start = time.time()
    func()
    time.sleep(3)
    end = time.time()
    print('spend %s' % (end - start))

show_time(foo)
```

完美实现！五星好评~~

```
import timedef show_time(f):
    def func():
        start = time.time()
        f()
        time.sleep(3)
        end = time.time()
        print('spend %s' % (end - start))
    return func

@show_time                        # @show_time 等价于 “foo =
show_time(foo)”def foo():
    print('foo...')
foo()
```

装饰器之功能函数的参数：

前边写了功能函数中没有带参数，如果功能功能函数中有参数呢?

```
import timedef show_time(f):
    def func(*a):
        start = time.time()
        f(*a)
```

```
        time.sleep(3)
        end = time.time()
        print('spend %s' % (end - start))
    return func

@show_timedef add(*a):
    num = 0
    for i in a:
        num = num +i
    print(num)
add(2, 2, 3, 4, 7)
```

装饰器之装饰器函数的参数：

有这样一个需求，在代码执行的过程中有的需要记录日志，有的不需要记录日志，怎么办呢？给原本装饰器外再套一个装饰器，功能函数执行过程中再进行判断。

```
import time
def logger(flag = 'True'):
    def show_time(x):
        def func():
            start = time.time()
            x()
            time.sleep(3)
            end = time.time()
            print('spend %s' % (end - start))
            if flag == 'True':
                with open('装饰器测试文件', 'w', encoding='utf8') as f:
                    f.write('添加日志测试！！！')
        return func
    return show_time

@logger()def foo():
    print('foo...')
foo()
```