

# day14

---

## 今日内容

---

- 带参数的装饰器： flask框架 + django缓存 + 写装饰器实现被装饰的函数要执行N次
- 模块
  - os
  - sys
  - time (三种类型)
  - datetime 和 timezone 【了解】

## 内容回顾 & 补充

---

### 1.函数

写代码的方式：面向过程 --> 函数式编程（多） --> 面向对象编程。

#### 1.1 函数基础

```
def func(a1,a2):  
    pass  
  
result = func(1,2)
```

#### 1.2 参数

补充：对于函数的默认值慎用可变类型。

```
# 如果想给value设置默认是空列表  
  
# 不推荐(坑)  
def func(data,value=[]):  
    pass  
  
# 推荐  
def func(data,value=None):  
    if not value:  
        value = []
```

```
def func(data,value=[]):
    value.append(data)
    return value

v1 = func(1) # [1,]
v2 = func(1,[11,22,33]) # [11,22,33,1]
```

面试题:

- def func(a,b=[]) 有什么陷阱?
- 看代码写结果

```
def func(a,b=[]):
    b.append(a)
    return b

l1 = func(1)
l2 = func(2,[11,22])
l3 = func(3)

# [1,3]    [11,22,2]    [1,3]
print(l1,l2,l3)
```

- 看代码写结果

```
def func(a,b=[]):
    b.append(a)
    print(b)

func(1)
func(2,[11,22,33])
func(3)

# [1]    [11,22,33,2]    [1,3]
```

### 1.3 返回值

分析函数执行的内存

```
def func(name):
    def inner():
        print(name)
        return 123
    return inner

v1 = func('alex')
v2 = func('eric')

v1()
v2()
```

## 闭包

```
# 不是闭包
def func1(name):
    def inner():
        return 123
    return inner

# 是闭包: 封装值 + 内层函数需要使用。
def func2(name):
    def inner():
        print(name)
        return 123
    return inner
```

## 1.4 作用域

## 1.5 递归

函数自己调用自己。（效率低）

```
def func():
    print(1)
    func()
```

func()

```
def func(i):
    print(i)
    func(i+1)
```

func(1)

```
def func(a,b):
    # 1
    # 1
    # 2
    # 3
    # 5
    print(b)
    func(b,a+b)
```

func(0,1)

```
v = func(1)
```



- ```
# 递归的返回值

def func(a):
    if a == 5:
        return 100000
    result = func(a+1) + 10

v = func(1)

name = 'alex'

def func():
    def inner():
        print(name)
    return inner

v = func()
```

- hashlib

- random
- getpass
- time

## 内容详细

---

### 1. 装饰器

#### 1.1 基本格式

```
def x(func):  
    def inner():  
        return func()  
    return inner  
  
@x  
def index():  
    pass
```

#### 1.2 关于参数

```
def x(func):  
    def inner(a1):  
        return func(a1)  
    return inner  
  
@x  
def index(a1):  
    pass
```

```
def x(func):  
    def inner(a1,a2):  
        return func(a1,a2)  
    return inner  
  
@x  
def index(a1,a2):  
    pass  
  
# index = inner  
index(1,2)  
  
# ##### 参数统一的目的是为了原来的index函数传参  
def x(func):  
    def inner(a1,a2):  
        return func()  
    return inner  
  
@x
```

```
def index():  
    pass  
# func = 原来的index函数u  
# index = inner  
index(1,2)
```

如果给好几个函数写一个统一的装饰器，怎么办？

```
def x1(func):  
    def inner(*args,**kwargs):  
        return func(*args,**kwargs)  
    return inner  
  
@x1  
def f1():  
    pass  
  
@x1  
def f2(a1):  
    pass  
  
@x1  
def f3(a1,a2):  
    pass
```

### 1.3 关于返回值

```
def x1(func):  
    def inner(*args,**kwargs):  
        data = func(*args,**kwargs)  
        return data  
    return inner  
  
@x1  
def f1():  
    print(123)  
  
v1 = f1()  
print(v1)
```

```
def x1(func):
    def inner(*args, **kwargs):
        data = func(*args, **kwargs)
        return data
    return inner

@x1
def f1():
    print(123)
    return 666

v1 = f1()
print(v1)
```

```
def x1(func):
    def inner(*args, **kwargs):
        data = func(*args, **kwargs)
        return inner

@x1
def f1():
    print(123)
    return 666

v1 = f1()
print(v1)
```

装饰器建议写法：

```
def x1(func):
    def inner(*args, **kwargs):
        data = func(*args, **kwargs)
        return data
    return inner
```

## 1.4 关于前后

```
def x1(func):
    def inner(*args, **kwargs):
        print('调用原函数之前')
        data = func(*args, **kwargs) # 执行原函数并获取返回值
        print('调用员函数之后')
        return data
    return inner

@x1
def index():
    print(123)

index()
```

## 1.5 带参数的装饰器

```
# 第一步: 执行 ret = xxx(index)
# 第二步: 将返回值赋值给 index = ret
@xxx
def index():
    pass

# 第一步: 执行 v1 = uuu(9)
# 第二步: ret = v1(index)
# 第三步: index = ret
@uuu(9)
def index():
    pass
```

```
# ##### 普通装饰器 #####
def wrapper(func):
    def inner(*args, **kwargs):
        print('调用原函数之前')
        data = func(*args, **kwargs) # 执行原函数并获取返回值
        print('调用原函数之后')
        return data
    return inner

@wrapper
def index():
    pass

# ##### 带参数装饰器 #####
def x(counter):
    def wrapper(func):
        def inner(*args, **kwargs):
            data = func(*args, **kwargs) # 执行原函数并获取返回值
            return data
        return inner
    return wrapper

@x(9)
def index():
    pass
```

### 练习题

```
# 写一个带参数的装饰器, 实现: 参数是多少, 被装饰的函数就要执行多少次, 把每次结果添加到列表中, 最终返回列表。
def xxx(counter):
    print('x函数')
    def wrapper(func):
        print('wrapper函数')
        def inner(*args, **kwargs):
            v = []
            for i in range(counter):
```



```

        data = func(*args,**kwargs) # 执行原函数并获取返回值
        v.append(data)
    return v
    return inner
return wrapper

@xxx(5)
def index():
    return 8

v = index()
print(v)

# 写一个带参数的装饰器，实现：参数是多少，被装饰的函数就要执行多少次，并返回最后一次执行的结果【面试题】
def xxx(counter):
    print('x函数')
    def wrapper(func):
        print('wrapper函数')
        def inner(*args,**kwargs):
            for i in range(counter):
                data = func(*args,**kwargs) # 执行原函数并获取返回值
            return data
        return inner
    return wrapper

@xxx(5)
def index():
    return 8

v = index()
print(v)

# 写一个带参数的装饰器，实现：参数是多少，被装饰的函数就要执行多少次，并返回执行结果中最大的值。
def xxx(counter):
    print('x函数')
    def wrapper(func):
        print('wrapper函数')
        def inner(*args,**kwargs):
            value = 0
            for i in range(counter):
                data = func(*args,**kwargs) # 执行原函数并获取返回值
                if data > value:
                    value = data
            return value
        return inner
    return wrapper

@xxx(5)
def index():
    return 8

v = index()
print(v)

```

```
def x(counter):
    print('x函数')
    def wrapper(func):
        print('wrapper函数')
        def inner(*args,**kwargs):
            if counter:
                return 123
            return func(*args,**kwargs)
        return inner
    return wrapper

@x(True)
def fun990():
    pass

@x(False)
def func10():
    pass
```

## 1.6 欠

- 元数据：flask框架
- 多个装饰器:: Flask框架

```
@x1
@x2
def func():
    pass
```

## 总结

基本装饰器（更重要）

```
def x1(func):
    def inner(*args,**kwargs):
        data = func(*args,**kwargs) # 执行原函数并获取返回值
        return data
    return inner

@x1
def index():
    print(123)

index()
```

带参数的装饰器

```
def x(counter):
    def wrapper(func):
        def inner(*args,**kwargs):
            data = func(*args,**kwargs) # 执行原函数并获取返回值
            return data
        return inner
    return wrapper

@x(9)
def index():
    pass
```

## 2.模块

### 2.1 sys

python解释器相关的数据。

- sys.getrefcount , 获取一个值的应用计数

```
a = [11,22,33]
b = a
print(sys.getrefcount(a))
```

- sys.getrecursionlimit , python默认支持的递归数量
- sys.stdout.write --> print (进度)

```
import time
for i in range(1,101):
    msg = "%s%%\r" %i
    print(msg,end='')
    time.sleep(0.05)
```

```
import os

# 1. 读取文件大小 (字节)
file_size = os.stat('20190409_192149.mp4').st_size

# 2. 一点一点的读取文件
read_size = 0
with open('20190409_192149.mp4',mode='rb') as f1,open('a.mp4',mode='wb') as f2:
    while read_size < file_size:
        chunk = f1.read(1024) # 每次最多去读取1024字节
        f2.write(chunk)
        read_size += len(chunk)
        val = int(read_size / file_size * 100)
        print('%s%%\r' %val ,end='')
```

- sys.argv

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
"""
让用户执行脚本传入要删除的文件路径，在内部帮助用将目录删除。
C:\Python36\python36.exe D:/code/s21day14/7.模块传参.py D:/test
C:\Python36\python36.exe D:/code/s21day14/7.模块传参.py

"""
import sys

# 获取用户执行脚本时，传入的参数。
# C:\Python36\python36.exe D:/code/s21day14/7.模块传参.py D:/test
# sys.argv = [D:/code/s21day14/7.模块传参.py, D:/test]
path = sys.argv[1]

# 删除目录
import shutil
shutil.rmtree(path)
```

- sys.path --- 欠

## 2.2 os

和操作系统相关的数据。

- os.path.exists(path)，如果path存在，返回True；如果path不存在，返回False
- os.stat('20190409\_192149.mp4').st\_size，获取文件大小
- os.path.abspath()，获取一个文件的绝对路径

```
path = '20190409_192149.mp4' # D:\code\s21day14\20190409_192149.mp4

import os
v1 = os.path.abspath(path)
print(v1)
```

- os.path.dirname，获取路径的上级目录

```
import os
v = r"D:\code\s21day14\20190409_192149.mp4"

print(os.path.dirname(v))
```

- os.path.join，路径的拼接

```
import os
path = "D:\code\s21day14" # user/index/inx/fasd/
v = 'n.txt'

result = os.path.join(path,v)
print(result)
result = os.path.join(path,'n1','n2','n3')
print(result)
```

- os.listdir , 查看一个目录下所有的文件【第一层】

```
import os

result = os.listdir(r'D:\code\s21day14')
for path in result:
    print(path)
```

- os.walk , 查看一个目录下所有的文件【所有层】

```
import os

result = os.walk(r'D:\code\s21day14')
for a,b,c in result:
    # a,正在查看的目录 b,此目录下的文件夹 c,此目录下的文件
    for item in c:
        path = os.path.join(a,item)
        print(path)
```

- 补充:
  - 转义

```
v1 = r"D:\code\s21day14\n1.mp4" (推荐)
print(v1)

v2 = "D:\\code\\s21day14\\n1.mp4"
print(v2)
```

## 2.3 shutil

```
import shutil
shutil.rmtree(path)
```

## 总结

- 普通装饰器 5\*

- 参数
  - 返回值
  - 前后
- 带参数 4\*
- 模块：
  - random
  - hashlib
  - getpass
  - time
  - os
  - sys
  - shutil