



马哥教育

IT 人的高薪职业学院

Python装饰器

讲师：Wayne

从业十余载，漫漫求知路

装饰器

□ 需求

- 一个加法函数，想增强它的功能，能够输出被调用过以及调用的参数信息

```
def add(x, y):
```

```
    return x + y
```

增加信息输出功能

```
def add(x, y):
```

```
    print("call add, x + y") # 日志输出到控制台
```

```
    return x + y
```

- 上面的加法函数是完成了需求，但是有以下的缺点

- 打印语句的耦合太高

- 加法函数属于业务功能，而输出信息的功能，属于非业务功能代码，不该放在业务函数加法中



装饰器

- 做到了业务功能分离，但是 fn函数调用传参是个问题

```
def add(x,y):  
    return x + y
```

```
def logger(fn):  
    print('begin') # 增强的输出  
    x = fn(4,5)  
    print('end') # 增强的功能  
    return x
```

```
print(logger(add))
```



装饰器

- 解决了传参的问题，进一步改变

```
def add(x,y):  
    return x + y
```

```
def logger(fn,*args,**kwargs):  
    print('begin')  
    x = fn(*args,**kwargs)  
    print('end')  
    return x
```

```
print(logger(add,5,y=60))
```



装饰器

□ 柯里化

```
def add(x,y):  
    return x + y
```

```
def logger(fn):
```

```
    def wrapper(*args,**kwargs):
```

```
        print('begin')
```

```
        x = fn(*args,**kwargs)
```

```
        print('end')
```

```
        return x
```

```
    return wrapper
```

```
print(logger(add)(5,y=50))
```

换一种写法

```
add = logger(add)
```

```
print(add(x=5, y=10))
```



马哥教育
IT人的高薪职业学院

装饰器

□ 装饰器语法糖

```
def logger(fn):  
    def wrapper(*args,**kwargs):  
        print('begin')  
        x = fn(*args,**kwargs)  
        print('end')  
        return x  
    return wrapper
```

@logger # 等价于 add = logger(add)

```
def add(x,y):  
    return x + y
```

```
print(add(45,40))
```

□ @logger 是什么？这就是装饰器语法

装饰器

□ 装饰器（无参）

- 它是一个函数
- 函数作为它的形参
- 返回值也是一个函数
- 可以使用@functionname方式，简化调用

□ 装饰器和高阶函数

- 装饰器是高阶函数，但装饰器是对传入函数的功能的装饰（功能增强）

装饰器

```
import datetime
import time
```

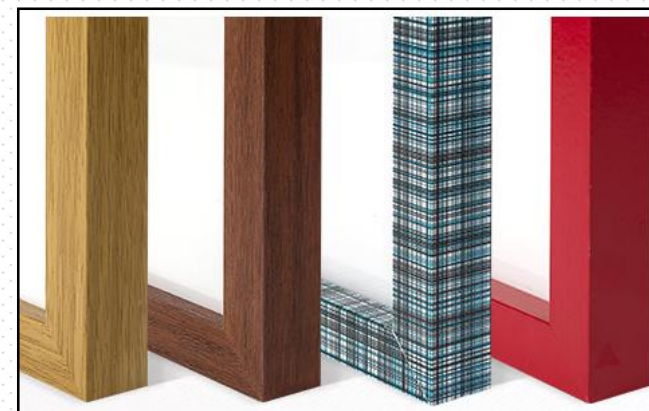
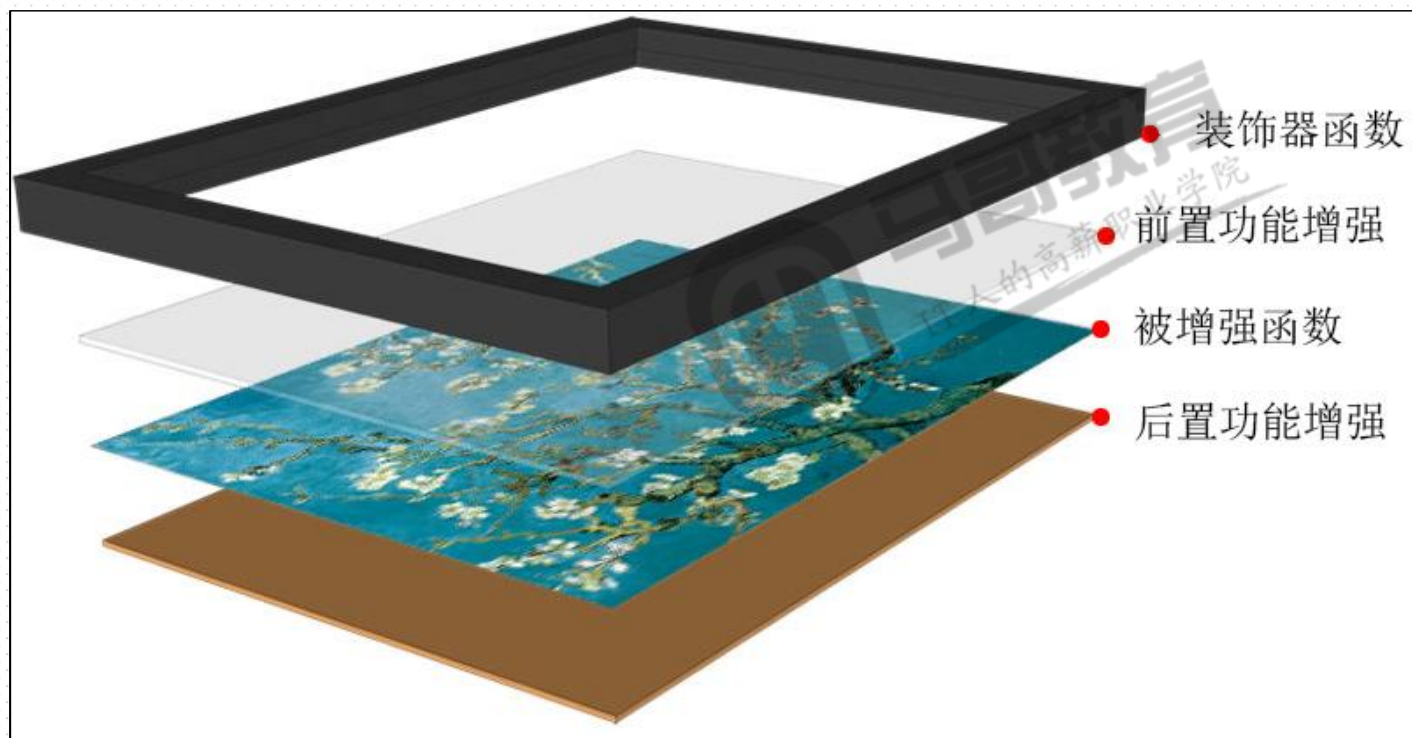
```
def logger(fn):
    def wrap(*args, **kwargs):
        # before 功能增强
        print("args={}, kwargs={}".format(args,kwargs))
        start = datetime.datetime.now()
        ret = fn(*args, **kwargs)
        # after 功能增强
        duration = datetime.datetime.now() - start
        print("function {} took {}s.".format(fn.__name__, duration.total_seconds()))
        return ret
    return wrap
```

```
@logger # 相当于 add = logger(add)
def add(x, y):
    print("===call add=====")
    time.sleep(2)
    return x + y
```

```
print(add(4, y=7))
```


装饰器

□ 怎么理解装饰器呢？



马哥教育
IT 人的高薪职业学院

文档字符串

□ Python的文档

- Python是文档字符串Documentation Strings
- 在函数语句块的第一行，且习惯是多行的文本，所以多使用三引号
- 惯例是首字母大写，第一行写概述，空一行，第三行写详细描述
- 可以使用特殊属性__doc__访问这个文档

```
def add(x,y):  
    """This is a function of addition"""  
  
    a = x+y  
  
    return x + y
```

```
print("name={} \ndoc={}".format(add.__name__, add.__doc__))  
print(help(add))
```

装饰器

□ 副作用

```
def logger(fn):  
    def wrapper(*args,**kwargs):  
        'I am wrapper'  
        print('begin')  
        x = fn(*args,**kwargs)  
        print('end')  
        return x  
    return wrapper
```

```
@logger #add = logger(add)  
def add(x,y):  
    """This is a function for add"""  
    return x + y
```

```
print("name={}, doc={}".format(add.__name__, add.__doc__))
```

□ 原函数对象的属性都被替换了，而使用装饰器，我们的需求是查看被封装函数的属性，如何解决？



装饰器

□ 提供一个函数，被封装函数属性 ==copy==> 包装函数属性

```
def copy_properties(src, dst): # 可以改造成装饰器
    dst.__name__ = src.__name__
    dst.__doc__ = src.__doc__
```

```
def logger(fn):
    def wrapper(*args, **kwargs):
        'I am wrapper'
        print('begin')
        x = fn(*args, **kwargs)
        print('end')
        return x
    copy_properties(fn, wrapper)
    return wrapper
```

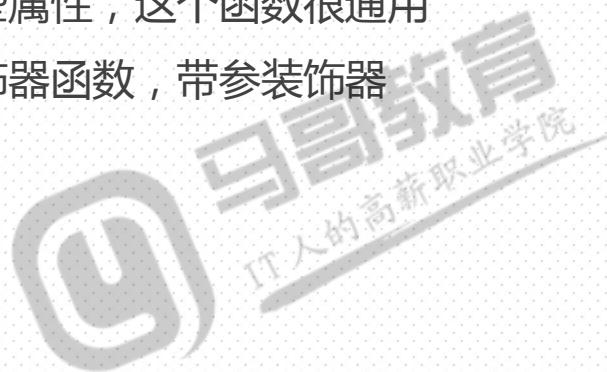
```
@logger #add = logger(add)
def add(x,y):
    """This is a function for add"""
    return x + y
```

```
print("name={}, doc={}".format(add.__name__, add.__doc__))
```



装饰器

- 通过copy_properties函数将被包装函数的属性覆盖掉包装函数
- 凡是被装饰的函数都需要复制这些属性，这个函数很通用
- 可以将复制属性的函数构建装饰器函数，带参装饰器



装饰器

□ 提供一个函数，被封装函数属性 ==copy==> 包装函数属性，改造成带参装饰器

```
def copy_properties(src): # 柯里化
    def _copy(dst):
        dst.__name__ = src.__name__
        dst.__doc__ = src.__doc__
        return dst
    return _copy
def logger(fn):
    @copy_properties(fn) # wrapper = copy_properties(fn)(wrapper)
    def wrapper(*args, **kwargs):
        'I am wrapper'
        print('begin')
        x = fn(*args, **kwargs)
        print('end')
        return x
    return wrapper

@logger #add = logger(add)
def add(x, y):
    '''This is a function for add'''
    return x + y
print("name={}, doc={}".format(add.__name__, add.__doc__))
```


带参装饰器

□ 需求

- 获取函数的执行时长，对时长超过阈值的函数记录一下

```
def logger(duration):  
    def _logger(fn):  
        @copy_properties(fn) # wrapper = wrapper(fn)(wrapper)  
        def wrapper(*args,**kwargs):  
            start = datetime.datetime.now()  
            ret = fn(*args,**kwargs)  
            delta = (datetime.datetime.now() - start).total_seconds()  
            print('so slow') if delta > duration else print('so fast')  
            return ret  
        return wrapper  
    return _logger
```

```
@logger(5) # add = logger(5)(add)  
def add(x,y):  
    time.sleep(3)  
    return x + y  
  
print(add(5, 6))
```



马哥教育
IT人的高薪职业学院

带参装饰器

□ 带参装饰器

- 它是一个函数
- 函数作为它的形参
- 返回值是一个不带参的装饰器函数
- 使用@functionname(参数列表)方式调用
- 可以看做在装饰器外层又加了一层函数

带参装饰器

- 将记录的功能提取出来，这样就可以通过外部提供的函数来灵活的控制输出

```
def logger(duration, func=lambda name, duration: print('{} took {}'.format(name, duration))):  
    def _logger(fn):  
        @copy_properties(fn) # wrapper = wrapper(fn)(wrapper)  
        def wrapper(*args,**kwargs):  
            start = datetime.datetime.now()  
            ret = fn(*args,**kwargs)  
            delta = (datetime.datetime.now() - start).total_seconds()  
            if delta > duration:  
                func(fn.__name__, duration)  
            return ret  
        return wrapper  
    return _logger
```


functools模块

❑ `functools.update_wrapper(wrapper, wrapped, assigned=WRAPPER_ASSIGNMENTS, updated=WRAPPER_UPDATES)`

❑ 类似`copy_properties`功能

❑ `wrapper` 包装函数、被更新者，`wrapped` 被包装函数、数据源

❑ 元组`WRAPPER_ASSIGNMENTS`中是要被覆盖的属性

`'__module__', '__name__', '__qualname__', '__doc__', '__annotations__'`

模块名、名称、限定名、文档、参数注解

❑ 元组`WRAPPER_UPDATES`中是要被更新的属性，`__dict__`属性字典

❑ 增加一个`__wrapped__`属性，保留着`wrapped`函数



马哥教育

IT人的高薪职业学院

functools模块

```
import datetime, time, functools
```

```
def logger(duration, func=lambda name, duration: print('{} took {}'.format(name, duration))):
```

```
    def _logger(fn):
```

```
        def wrapper(*args,**kwargs):
```

```
            start = datetime.datetime.now()
```

```
            ret = fn(*args,**kwargs)
```

```
            delta = (datetime.datetime.now() - start).total_seconds()
```

```
            if delta > duration:
```

```
                func(fn.__name__, duration)
```

```
            return ret
```

```
        return functools.update_wrapper(wrapper, fn)
```

```
    return _logger
```

```
@logger(5) # add = logger(5)(add)
```

```
def add(x,y):
```

```
    time.sleep(1)
```

```
    return x + y
```

```
print(add(5, 6), add.__name__, add.__wrapped__, add.__dict__, sep='\n')
```

functools模块

- ❑ `@functools.wraps(wrapped, assigned=WRAPPER_ASSIGNMENTS, updated=WRAPPER_UPDATES)`
 - ❑ 类似`copy_properties`功能
 - ❑ `wrapped` 被包装函数
 - ❑ 元组`WRAPPER_ASSIGNMENTS`中是要被覆盖的属性
`'__module__', '__name__', '__qualname__', '__doc__', '__annotations__'`
模块名、名称、限定名、文档、参数注解
 - ❑ 元组`WRAPPER_UPDATES`中是要被更新的属性，`__dict__`属性字典
 - ❑ 增加一个`__wrapped__`属性，保留着`wrapped`函数

functools模块

```
import datetime, time, functools
```

```
def logger(duration, func=lambda name, duration: print('{} took {}'.format(name, duration))):
```

```
    def _logger(fn):
```

```
        @functools.wraps(fn)
```

```
        def wrapper(*args, **kwargs):
```

```
            start = datetime.datetime.now()
```

```
            ret = fn(*args, **kwargs)
```

```
            delta = (datetime.datetime.now() - start).total_seconds()
```

```
            if delta > duration:
```

```
                func(fn.__name__, duration)
```

```
            return ret
```

```
        return wrapper
```

```
    return _logger
```

```
@logger(5) # add = logger(5)(add)
```

```
def add(x,y):
```

```
    time.sleep(1)
```

```
    return x + y
```

```
print(add(5, 6), add.__name__, add.__wrapped__, add.__dict__, sep='\n')
```

马哥教育
IT人的高薪职业学院

谢谢

咨询热线 400-080-6560

