**使用functools模块wrap方法装饰函数**

2016年06月13日 08:47:53

阅读数：2931

**使用functools模块wrap方法装饰函数**

今天无意间看到Python中的functools模块，发现使用这个模块的wraps()可以实现一些类似拦截器的功能，比如：包装异常，隐藏异常，打印日志，统计函数使用时间等。下面就通过几个代码片段来看看具体使用方法：

**包装异常**

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

import functools

def wrap\_exception(func):

@functools.wraps(func)

def wrapper(self, \*args, \*\*kwargs):

try:

return func(self, \*args, \*\*kwargs)

except BaseException as ex:

raise MyException(ex.message)

return wrapper

class MyException(Exception):

def \_\_init\_\_(self, msg):

self.msg = msg

def \_\_str\_\_(self):

return self.msg

def \_\_repr\_\_(self):

return self.msg

class Test:

def \_\_init\_\_(self):

pass

@wrap\_exception

def test(self):

raise Exception("hello")

t = Test()

t.test()

**代码说明：**

* wrap\_exception函数用来装饰我们最终的函数，主要功能就是捕捉异常然后转换成别的异常并抛出。
* 定义了一个自定义异常MyException，我们会把捕捉到的异常转换成这个异常。
* 一个测试类，测试方法使用@wrap\_exception说明使用装饰函数，这个方法会raise一个异常。这个异常最终会被转换成MyException异常。

**应用场景：**

* 这段代码可以用在我们期望某一类的函数对任何异常都抛出成我们期望的异常类型，最典型的case就是我们在处理数据库操作的时候期望任何异常都被转换成DataAccessException。

**隐藏异常**

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

import functools

def wrap\_exception(func):

@functools.wraps(func)

def wrapper(self, \*args, \*\*kwargs):

try:

return func(self, \*args, \*\*kwargs)

except:

pass

return wrapper

class Test:

def \_\_init\_\_(self):

pass

@wrap\_exception

def test(self):

raise Exception("hello")

t = Test()

t.test()

**代码说明：**

* 和上一个列子类似，只是我们在捕捉到异常后什么都不做，有点想让函数静默执行的意思。

**应用场景：**

* 可以用在关闭文件流或者数据库连接的时候。

**打印日志**

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

def wrap\_logger(func):

@functools.wraps(func)

def wrapper(self, \*args, \*\*kwargs):

print ("%s(%s, %s)" % (func, args, kwargs))

print "before execute"

result = func(self, \*args, \*\*kwargs)

print "after execute"

return result

return wrapper

class Test:

def \_\_init\_\_(self):

pass

@wrap\_logger

def test(self, a, b, c):

print a, b, c

t = Test()

t.test(1, 2, 3)

**代码说明：**

* 代码比较简单，主要是wrapper方法内部在调用函数前和函数后加了一些日志。

**应用场景：**

* 有时候我们为了调试方便，期望在进入函数和退出函数的时候打印一些日志信息，此时可以使用这段代码。

**统计函数使用时间**

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

import functools

import time

def wrap\_performance(func):

@functools.wraps(func)

def wrapper(self, \*args, \*\*kwargs):

t\_begin = time.time()

result = func(self, \*args, \*\*kwargs)

t\_end = time.time()

print("Time: %f " % (t\_end - t\_begin))

return result

return wrapper

class Test:

def \_\_init\_\_(self):

pass

@wrap\_performance

def test(self):

time.sleep(1)

t = Test()

t.test()

**代码说明：**

* 有了上一个列子，我们自然想到能不能统计一下函数执行时间，这个例子就此诞生。实现还是比较简单，真实情况下我们可能会需要一个数据结构来保存统计每个方法的最小/最大/平均执行时间。

**应用场景：**

* 用在需要统计函数执行时间的情况下。