Python @函数装饰器及用法（超级详细）

< 上一页 下一页 >

前面章节中，我们已经讲解了 Python 内置的 3 种函数装饰器，分别是 @staticmethod、@classmethod 和 @property，其中 staticmethod()、classmethod() 和 property() 都是 Python 的内置函数。

那么，函数装饰器的工作原理是怎样的呢？假设用 funA() 函数装饰器去装饰 funB() 函数，如下所示：

```
#funA 作为装饰器函数
def funA(fn):
    #...
    fn() # 执行传入的fn参数
    #...
    return '...'

@funA
def funB():
    #...
```

实际上，上面程序完全等价于下面的程序：

```
def funA(fn):
    #...
    fn() # 执行传入的fn参数
    #...
    return '...'

def funB():
    #...

funB = funA(funB)
```

通过比对以上 2 段程序不难发现，使用函数装饰器 A() 去装饰另一个函数 B()，其底层执行了如下 2 步操作：

1. 将 B 作为参数传给 A() 函数；
2. 将 A() 函数执行完成的返回值反馈回 B。

举个实例：

```
#funA 作为装饰器函数
def funA(fn):
    print("C语言中文网")
    fn() # 执行传入的fn参数
    print("http://c.biancheng.net")
    return "装饰器函数的返回值"

@funA
def funB():
    print("学习 Python")
```

程序执行流程为：

```
C语言中文网
学习 Python
http://c.biancheng.net
```

在此基础上，如果在程序末尾添加如下语句：

```
print(funB)
```

其输出结果为：

```
装饰器函数的返回值
```

显然，被“@函数”修饰的函数不再是原来的函数，而是被替换成一个新的东西（取决于装饰器的返回值），即如果装饰器函数的返回值为普通变量，那么被修饰的函数名就变成了变量名；同样，如果装饰器返回的是一个函数的名称，那么被修饰的函数名依然表示一个函数。

> 实际上，所谓函数装饰器，就是通过装饰器函数，在不修改原函数的前提下，来对函数的功能进行合理的扩充。

带参数的函数装饰器

在分析 funA() 函数装饰器和 funB() 函数的关系时，细心的读者可能会发现一个问题，即当 funB() 函数无参数时，可以直接将 funB 作为 funA() 的参数传入。但是，如果被修饰的函数本身带有参数，那

应该如何传值呢？

比较简单的解决方法就是在函数装饰器中嵌套一个函数，该函数带有的参数个数和被装饰器修饰的函数相同。例如：

```
def funA(fn):
    # 定义一个嵌套函数
    def say(arc):
        print("Python教程:",arc)
    return say

@funA
def funB(arc):
    print("funB():", a)
funB("http://c.biancheng.net/python")
```

程序执行结果为：

```
Python教程: http://c.biancheng.net/python
```

这里有必要给读者分析一下这个程序，其实，它和如下程序是等价的：

```
def funA(fn):
    # 定义一个嵌套函数
    def say(arc):
        print("Python教程:",arc)
    return say

def funB(arc):
    print("funB():", a)

funB = funA(funB)
funB("http://c.biancheng.net/python")
```

如果运行此程序会发现，它的输出结果和上面程序相同。

显然，通过 funB() 函数被装饰器 funA() 修饰，funB 就被赋值为 say。这意味着，虽然我们在程序显式调用的是 funB() 函数，但其实执行的是装饰器嵌套的 say() 函数。

但还有一个问题需要解决，即如果当前程序中，有多个（≥ 2）函数被同一个装饰器函数修饰，这些函数带有的参数个数并不相等，怎么办呢？

最简单的解决方式是用 *args 和 **kwargs 作为装饰器内部嵌套函数的参数，*args 和 **kwargs 表示接受任意数量和类型的参数。举个例子：

```
def funA(fn):
    # 定义一个嵌套函数
    def say(*args,**kwargs):
        fn(*args,**kwargs)
    return say

@funA
def funB(arc):
    print("C语言中文网：",arc)

@funA
def other_funB(name,arc):
    print(name,arc)
funB("http://c.biancheng.net")
other_funB("Python教程：","http://c.biancheng.net/python")
```

运行结果为：

```
C语言中文网： http://c.biancheng.net
Python教程： http://c.biancheng.net/python
```

函数装饰器可以嵌套

上面示例中，都是使用一个装饰器的情况，但实际上，Python 也支持多个装饰器，比如：

```
@funA
@funB
@funC
def fun():
    #...
```

上面程序的执行顺序是里到外，所以它等效于下面这行代码：

```
fun = funA( funB ( funC (fun) ) )
```

> 这里不再给出具体实例，有兴趣的读者可自行编写程序进行测试。

所有教程

C语言入门 C语言编译器 C语言项目案例 数据结构 C++ STL C++11
socket GCC GDB Makefile OpenCV Qt教程 Unity 3D UE4
游戏引擎 Python Python并发编程 TensorFlow Django NumPy Linux
Shell Java教程 设计模式 Java Swing Servlet JSP教程 Struts2
Maven Spring Spring MVC Spring Boot Spring Cloud Hibernate
Mybatis MySQL教程 MySQL函数 NoSQL Redis MongoDB HBase
Go语言 C# MATLAB JavaScript Bootstrap HTML CSS教程 PHP
汇编语言 TCP/IP vi命令 Android教程 区块链 Docker 大数据
云计算 编程笔记 资源下载 VIP视频 一对一答疑 关于我们

优秀文章

操作系统的计算环境应用
汇编语言CloseHandle函数：关闭一个打开的对象句柄
SpringBoot中使用Scala开发
JSP initParam对象：获取Web应用初始化参数的值
JS！（非运算）详解
JS arguments对象详解（附带多个实例）
暴力破解FTP服务器

Linux系统安全性分析

双向链表实现贪吃蛇游戏

使用建造者模式构建动态SQL语句

↑