



python3之装饰器

2019.04.23 17:46:26 字数 847 阅读 650

一、装饰器介绍

装饰器也是一个函数，它是让其他函数在不改变变动的前提下增加额外的功能。
装饰器是一个闭包，把一个函数当作参数返回一个替代版的函数，本质是一个返回函数的函数（即返回值为函数对象）。
python3支持用@符号直接将装饰器应用到函数。
装饰器工作场景：插入日志、性能测试、事务处理等等。
函数被装饰器装饰过后，此函数的属性均已发生变化，如名称变为装饰器的名称。

1. 简单的装饰器

1.1. 被装饰的函数不带参数

```
1  """入门装饰器：函数功能不带参数"""
2  def my_decorator(func):
3      def inner():
4          print("*****")
5          print("要添加的功能代码")
6          func()
7      return inner
8
9  # script1()函数调用装饰器的第一种方法
10 def script1():
11     print("测试")
12
13 runScript1 = my_decorator(script1)  # 运行script()函数的同时添加有my_decorator()函数的功能
14 runScript1()
15 # script1()函数调用装饰器的第二种方法：使用@符号，简单明了
16 @my_decorator
17 def script1():
18     print("测试")
19
20 script1()
```

1.2. 被装饰的函数带参数

可变参数args和关键字参数kwargs添加函数通用的装饰器

```
1  """入门装饰器：函数带参数"""
2  def my_decorator(func):
3      def inner(*args, **kwargs):  # 可变参数*args和关键字参数**kwargs
4          print("*****")
5          print("要添加的功能代码")
6          func(*args, **kwargs)
7      return inner
8
9  # script2()函数调用装饰器的第一种方法：了解即可
10 def script2(arg):
11     print("测试: %s" % arg)
12
13 runScript2 = my_decorator(script2)
14 runScript2("aaa")
15 # script2()函数调用装饰器的第二种方法：使用@符号，目前使用此方法
16 @my_decorator
17 def script2(arg):
18     print("测试: %s" % arg)
19
20 script2("aaa")
```

2. 装饰器带参数



python3之发送和读取邮件
阅读 40

python3数据类型之数字
阅读 46

精彩继续

卖不掉的房子，被套在北京的我们
阅读 34247



知乎热门一句话打脸父母，揭开多少人不败...
阅读 19301



python3之装饰器



rr1990

[关注](#)[赞赏支持](#)

```
3     def outer(func):
4         def inner(*args, **kwargs):
5             print("*****")
6             print("添加带装饰器参数%s的功能代码" % self.name)
7             func(*args, **kwargs)
8         return inner
9     return outer
10
11 @my_decorator(name='settings')
12 def script3(arg):
13     print("测试---%s" % arg)
14     script3("bbb")
```

3. 基于类封装的装饰器

`__call__()`方法是将实例成为一个可调用对象（即callable对象），同时不影响实例的构造，但可以改变实例的内部值。

3.1. 基于类封装的不带参数装饰器

通过类封装装饰器的实现方法：先通过构造函数`__init__()`传入函数；再通过`__call__`方法重载，并返回一个函数。

```
1 """基于类封装的不带参数装饰器"""
2 class MyDecorator:
3     def __init__(self, func):
4         self.func = func
5
6     def __call__(self, *args, **kwargs):
7         print("*****")
8         print("要添加的功能代码")
9         return self.func(*args, **kwargs)
10
11 @MyDecorator
12 def script4(arg):
13     print("测试---%s" % arg)
14     script4("ccc")
```

3.2. 基于类封装的带参数装饰器

通过类封装装饰器的实现方法：先通过构造函数`__init__()`传入装饰器参数；再通过`__call__`方法传入被装饰的函数，并返回一个函数。

```
1 """基于类封装的带参数装饰器"""
2 class MyDecorator:
3     def __init__(self, name):
4         self.name = name
5
6     def __call__(self, func):
7         def inner(*args, **kwargs):
8             print("*****")
9             print("添加带装饰器参数%s的功能代码" % self.name)
10            func(*args, **kwargs)
11        return inner
12
13 @MyDecorator(name="settings")
14 def script4(arg):
15     print("测试---%s" % arg)
16     script4("ddd")
```

二、常用的内置装饰器

1. @property装饰器

写下你的评论...

评论1 赞 ...



python3之装饰器

添加装饰器@property时，函数类型是返回值的类型：如，<class 'str'>

- property对象的setter方法：表示给属性添加设置功能，即可修改属性值。
若未添加设置属性，就设置新值，则会引发错误AttributeError: can't set attribute。
- property对象的deleter方法：表示给属性添加删除功能
若添加删除属性，就删除属性则会引发错误AttributeError: can't delete attribute。

```
1  """@property装饰器"""
2  class Test1:
3      def __init__(self, name):
4          self.__name = name
5
6      @property          # 将函数由方法变为属性
7      def get_name(self):
8          return self.__name
9
10     @get_name.setter      # 添加设置属性
11     def get_name(self, value):
12         if not isinstance(value, str):
13             raise TypeError("参数应为字符串类型，但实际是%s类型" % type(value))
14         else:
15             self.__name = value
16
17     @get_name.deleter      # 添加删除属性
18     def get_name(self):
19         del self.__name
20
21 test1 = Test1("launcher")
22 # 获取get_name类型
23 print(type(test1.get_name))    # 结果: <class 'str'>
24 # 获取get_name属性值
25 print(test1.get_name)         # 结果: launcher
26 # 给get_name属性设置新值：添加设置属性需使用装饰器@property的setter函数；
27 test1.get_name = "赋新值"
28 print(test1.get_name)         # 结果: 赋新值
29 # 删除get_name属性：删除属性需使用装饰器@property的deleter函数；
30 del test1.get_name
31 print(test1.get_name)         # 结果:报错(AttributeError: 'Test1' object has no attribute '_Test1')
32
33
34
35 """@property实例：加减法运算"""
36 class Test2:
37     def __init__(self, a, b):
38         self.a = a
39         self.b = b
40
41
42     @property
43     def add(self):
44         return self.a + self.b
45
46     @property
47     def reduce(self):
48         return self.a - self.b
49
50 print(Test2(3, 1).add)        # 结果: 4
51 print(Test2(5, 2).reduce)    # 结果: 3
```

2. 类对象中的方法

类对象中的方法：实例方法、类方法和静态方法

- 实例方法：函数中的第一个参数为self的方法
- 静态方法：使用@staticmethod装饰器来将类中的函数定义为静态方法。
类中创建的一些方法，但该方法并不需要引用类或实例。静态方法通过类直接调用，无需创建对象，也无需传递self。
- 类方法：使用@classmethod装饰器来装饰类中的函数定义为类方法

写下你的评论...

评论1 赞 ...

python3之装饰器



rr1990

关注

赞赏支持

```

1  """实例方法、静态方法@staticmethod、类方法@classmethod"""
2  class Student:
3      description = "学员统计信息"
4
5      def __init__(self, name, age, sex):
6          self.name = name
7          self.age = age
8          self.sex = sex
9
10     def function(self, fun_type):
11         return fun_type
12
13     def instance_method(self):          # 实例方法
14         use_type = self.function("实例方法")
15         print("-----%s-----" % use_type)
16         print(Student.description)
17         print(self.name + '_' + str(self.age) + '_' + self.sex)
18
19     @staticmethod                      # 静态方法
20     def static_method():
21         student_info = Student("xiaoxiao", 20, "female")
22         use_type = student_info.function("静态方法")
23         print("-----%s-----" % use_type)
24         print(Student.description)
25         print(student_info.name + '_' + str(student_info.age) + '_' + student_info.sex)
26
27     @classmethod                        # 类方法
28     def class_method(cls):
29         student_info = cls("xiaoming", 23, "male")
30         use_type = student_info.function("类方法")
31         print("-----%s-----" % use_type)
32         print(Student.description)
33         print(student_info.name + '_' + str(student_info.age) + '_' + student_info.sex)
34
35     def call_different_method(self):
36         print("-----同一类对象中调用实例方法、静态方法、类方法-----")
37         self.instance_method()
38         self.static_method()
39         self.class_method()
40
41 # 实例方法
42 Student("xiaohong", 19, "female").instance_method()
43 # 静态方法
44 Student.static_method()
45 # 类方法
46 Student.class_method()
47 # 同一类对象中某个函数调用实例/静态/类方法
48 Student("xiaohong", 19, "female").call_different_method()

```

三、使用三方已封装的装饰器

• 三方模块decorator

先安装decorator模块，再导入from decorator import decorator

• 三方模块wrapt

先安装wrapt模块，再导入import wrapt

```

1  # decorator三方模块
2  from decorator import decorator
3  @decorator
4  def My_decorator(func, *args, **kwargs):
5      print("*****")
6      print("添加封装的功能内容")
7      return func(*args, **kwargs)
8
9  @My_decorator
10 def testScript2():
11     print("待装饰的函数")
12 testScript2()
13

```

写下你的评论...

评论1 赞 ...

python3之装饰器



rr1990

关注

赞赏支持

```
19 @wrapt.decorator
20 def My_decorator(wrapped, instance, args, kwargs):
21     # instance参数即使用不使用也必须保留
22     print("*****")
23     print("添加封装的功能内容")
24     return wrapped(*args, **kwargs)
25
26
27 @My_decorator
28 def testScript1():
29     print("待装饰的函数")
30 testScript1()
31
32 """装饰器带参数"""
33 def My_decorator(name):
34     @wrapt.decorator
35     def inner(wrapped, instance, args, kwargs):
36         print("*****")
37         print("添加封装的功能内容，且装饰器参数为%s" % name)
38         return wrapped(*args, **kwargs)
39     return inner
40
41 @My_decorator(set)
42 def testScript1():
43     print("待装饰的函数")
44 testScript1()
```

0人点赞 >

python基础学习



rr1990
拥有2钻 (约0.43元)

关注

"小礼物走一走，来简书关注我"

赞赏

写下你的评论...

全部评论 1 只看作者

按时间倒序 按时间正序



QSC三个
2楼 07.19 16:26
2. 装饰器带参数
里的self.name应该去掉self

赞 回复

推荐阅读

更多精彩内容 >

2016年5月Swift 2 学习 --- 117个注意事项与要点

这是16年5月份编辑的一份比较杂乱适合自己观看的学习记录文档，今天18年5月份再次想写文章，发现简书还为我保存起的...

Jenaral

Python学习笔记[2]

要点：函数式编程：注意不是“函数编程”，多了一个“式” 模块：如何使用模块 面向对象编程：面向对象的概念、属性、...

victorsung0

写下你的评论... 评论1 赞 ...