装饰器

面向切面的编程范式
(Aspect-Oriented Programming - AOP)

在运行时，动态地将代码切入到类的指定方法、指定位置上的编程思想就是面向切面的编程，更通俗一点就是通过在现有代码中添加额外行为而不修改代码本身

```
In : def func1():
...:     print('inside func1()')
...:     return 1
...:

In : def func2():
...:     print('inside func2()')
...:     return 2
...:
```

```
In : from datetime import datetime

In : def func1():
...:     print('inside func1()')
...:     print(datetime.now())
...:     return 1

In : func1()
inside func1()
2018-03-31 17:54:10.255646
Out: 1
```

```
In : def do(func):
...:     rs = func()
...:     print(datetime.now())
...:     return rs
...:

In : do(func1)
inside func1()
2018-03-31 18:09:10.053601
Out: 1
```

```
In : def do(func):
...:     def wrapper():
...:         rs = func()
...:         print(datetime.now())
...:         return rs
...:     return wrapper
...:

In : func1 = do(func1)

In : func1()
inside func1()
2018-03-31 18:21:48.191821
Out: 1

In : func1
Out: <function __main__.do.<locals>.wrapper>
```

```
In : @do
...: def func1():
...:     print('inside func1()')
...:     return 1
...:

# 等于 func1 = do(func1)
In : func1
Out: <function __main__.do.<locals>.wrapper>

In : func1()
inside func1()
2018-03-31 18:27:40.513106
Out: 1
```

@符号是装饰器的语法糖，语法糖指计算机语言中添加的某种语法，这种语法对语言的功能没有影响，但是更方便程序员使用。语法糖让程序更加简洁，有更高的可读性。

装饰器应用场景

1. 记录函数行为 (日志统计、缓存、计时)
2. 预处理 / 后处理 (配置上下文、参数字段检查、统一返回格式)
3. 注入 / 移除参数
4. 修改调用时的上下文 (实现异步或者并行)

使用装饰器有如下好处

1. 降低模块的耦合度
2. 使系统容易扩展
3. 更好的代码复用性

functools.wraps

```
In : def func2():
...:     '''func2 doc'''
...:     print('inside func2()')
...:     return 2
...:

In : func2.__name__
Out: 'func2'

In : func2.__module__
Out: '__main__'

In : func2.__doc__
Out: 'func2 doc'

In : func2 = do(func2)

In : func2.__name__
Out: 'wrapper'

In : func2.__module__
Out: '__main__'

In : func2.__doc__
```

```
In : def func2():
...:     '''func2 doc'''
...:     print('inside func2()')
...:     return 2
...:

In : def do(func):
...:     @wraps(func)
...:     def wrapper():
...:         rs = func()
...:         print(datetime.now())
...:         return rs
...:     return wrapper
...:

In : func2 = do(func2)

In : func2.__name__
Out: 'func2'

In : func2.__doc__
Out: 'func2 doc'
```

给函数的类装饰器

```
In : class Common:
...:     def __init__(self, func):
...:         self.func = func
...:     def __call__(self, *args, **kwargs):
...:         print(f'args: {args}')
...:         return self.func(*args, **kwargs)
...:

In : @Common
...: def test(num):
...:     print(f'Number: {num}')
...:

In : test(10)  # 也就是 Common(test)(10)
args: (10,)
Number: 10
```

```
In : def common(func):
...:     def wrapper(*args, **kwargs):
...:         print(f'args: {args}')
...:         return func(*args, **kwargs)
...:     return wrapper
...:

In : common(test)(10)
args: (10,)
Number: 10

In : common(test)
Out: <function __main__.common.<locals>.wrapper>
```

给类用的函数装饰器

```
In : def borg(cls):
...:     cls._state = {}
...:     orig_init = cls.__init__
...:     def new_init(self, *args, **kwargs):
...:         self.__dict__ = cls._state
...:         orig_init(self, *args, **kwargs)
...:     cls.__init__ = new_init
...:     return cls
...:

In : @borg
...: class A:
...:     def common(self):
...:         print(hex(id(self)))
...:

In : a, b = A(), A()

In : b.d
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-104-1dbeb93aa9bb> in <module>()
----> 1 b.d

AttributeError: 'A' object has no attribute 'd'

In : b.d = 1

In : a.d
Out: 1

In : a.common()
0x104f0c198
```

```
# 延伸阅读4

import attr


@attr.s(hash=True)
class Product(object):
    id = attr.ib()
    author_id = attr.ib()
    ...
```

带参数的装饰器

```
In : def common(*args, **kw):
...:     a = args
...:     def _common(func):
...:         def _deco(*args, **kwargs):
...:             print(f'args: {args} {a}')
...:             return func(*args, **kwargs)
...:         return _deco
...:     return _common
...:

In : @common('abc')
...: def test(num):
...:     print(f'Number: {num}')
...:

In : test(10)  # 相当于 common('abc')(test)(10)
args: (10,) ('abc',)
Number: 10
```

延伸阅读

1. https://wiki.python.org/moin/PythonDecoratorLibrary
2. https://github.com/madisonmay/Tomorrow
3. http://dongweiming.github.io/Expert-Python/#36
4. https://zhuanlan.zhihu.com/p/34963159