9.3 解除一个装饰器¶

问题¶

一个装饰器已经作用在一个函数上，你想撤销它，直接访问原始的未包装的那个函数。

解决方案¶

假设装饰器是通过 `@wraps` (参考9.2小节)来实现的，那么你可以通过访问 `__wrapped__` 属性来访问原始函数：

```
>>> @somedecorator
>>> def add(x, y):
...     return x + y
...
>>> orig_add = add.__wrapped__
>>> orig_add(3, 4)
7
>>>
```

讨论¶

直接访问未包装的原始函数在调试、内省和其他函数操作时是很有用的。 但是我们这里的方案仅仅适用于在包装器中正确使用了 `@wraps` 或者直接设置了 `__wrapped__` 属性的情况。

如果有多个包装器，那么访问 `__wrapped__` 属性的行为是不可预知的，应该避免这样做。 在Python3.3中，它会略过所有的包装层，比如，假如你有如下的代码：

```
from functools import wraps

def decorator1(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        print('Decorator 1')
        return func(*args, **kwargs)
    return wrapper

def decorator2(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        print('Decorator 2')
        return func(*args, **kwargs)
    return wrapper

@decorator1
@decorator2
def add(x, y):
    return x + y
```

下面我们在Python3.3下测试：

```
>>> add(2, 3)
Decorator 1
Decorator 2
5
>>> add.__wrapped__(2, 3)
5
>>>
```

下面我们在Python3.4下测试：

```
>>> add(2, 3)
Decorator 1
Decorator 2
5
```

```
>>> add.__wrapped__(2, 3)
Decorator 2
5
>>>
```

最后要说的是，并不是所有的装饰器都使用了 `@wraps`，因此这里的方案并不全部适用。特别的，内置的装饰器 `@staticmethod` 和 `@classmethod` 就没有遵循这个约定 (它们把原始函数存储在属性 `__func__` 中)。