9.2 创建装饰器时保留函数元信息¶

问题¶

你写了一个装饰器作用在某个函数上，但是这个函数的重要的元信息比如名字、文档字符串、注解和参数签名都丢失了。

解决方案¶

任何时候你定义装饰器的时候，都应该使用 `functools` 库中的 `@wraps` 装饰器来注解底层包装函数。例如：

```
import time
from functools import wraps
def timethis(func):
    '''
    Decorator that reports the execution time.
    '''
    @wraps(func)
    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        print(func.__name__, end-start)
        return result
    return wrapper
```

下面我们使用这个被包装后的函数并检查它的元信息：

```
>>> @timethis
... def countdown(n):
...     '''
...     Counts down
...     '''
...     while n > 0:
...         n -= 1
...
>>> countdown(100000)
countdown 0.008917808532714844
>>> countdown.__name__
'countdown'
>>> countdown.__doc__
'\n\tCounts down\n\t'
>>> countdown.__annotations__
{'n': <class 'int'>}
>>>
```

讨论¶

在编写装饰器的时候复制元信息是一个非常重要的部分。如果你忘记了使用 `@wraps`， 那么你会发现被装饰函数丢失了所有有用的信息。比如如果忽略 `@wraps` 后的效果是下面这样的：

```
>>> countdown.__name__
'wrapper'
>>> countdown.__doc__
>>> countdown.__annotations__
{}
>>>
```

`@wraps` 有一个重要特征是它能让你通过属性 `__wrapped__` 直接访问被包装函数。例如:

```
>>> countdown.__wrapped__(100000)
>>>
```

`__wrapped__` 属性还能让被装饰函数正确暴露底层的参数签名信息。例如：

```
>>> from inspect import signature
>>> print(signature(countdown))
(n:int)
>>>
```

一个很普遍的问题是怎样让装饰器去直接复制原始函数的参数签名信息， 如果想自己手动实现的话需要做大量的工作，最好就简单的使用 `@wraps` 装饰器。 通过底层的 `__wrapped__` 属性访问到函数签名信息。更多关于签名的内容可以参考9.16小节。