9.11 装饰器为被包装函数增加参数¶

问题¶

你想在装饰器中给被包装函数增加额外的参数，但是不能影响这个函数现有的调用规则。

解决方案¶

可以使用关键字参数来给被包装函数增加额外参数。考虑下面的装饰器：

```
from functools import wraps

def optional_debug(func):
    @wraps(func)
    def wrapper(*args, debug=False, **kwargs):
        if debug:
            print('Calling', func.__name__)
        return func(*args, **kwargs)

    return wrapper
```

```
>>> @optional_debug
... def spam(a,b,c):
...     print(a,b,c)
...
>>> spam(1,2,3)
1 2 3
>>> spam(1,2,3, debug=True)
Calling spam
1 2 3
>>>
```

讨论¶

通过装饰器来给被包装函数增加参数的做法并不常见。 尽管如此，有时候它可以避免一些重复代码。例如，如果你有下面这样的代码：

```
def a(x, debug=False):
    if debug:
        print('Calling a')

def b(x, y, z, debug=False):
    if debug:
        print('Calling b')

def c(x, y, debug=False):
    if debug:
        print('Calling c')
```

那么你可以将其重构成这样：

```
from functools import wraps
import inspect
```

```
def optional_debug(func):
    if 'debug' in inspect.getargspec(func).args:
        raise TypeError('debug argument already defined')

    @wraps(func)
    def wrapper(*args, debug=False, **kwargs):
        if debug:
            print('Calling', func.__name__)
        return func(*args, **kwargs)
    return wrapper

@optional_debug
def a(x):
    pass

@optional_debug
def b(x, y, z):
    pass

@optional_debug
def c(x, y):
    pass
```

这种实现方案之所以行得通，在于强制关键字参数很容易被添加到接受 `*args` 和 `**kwargs` 参数的函数中。 通过使用强制关键字参数，它被作为一个特殊情况被挑选出来， 并且接下来仅仅使用剩余的位置和关键字参数去调用这个函数时，这个特殊参数会被排除在外。 也就是说，它并不会被纳入到 `**kwargs` 中去。

还有一个难点就是如何去处理被添加的参数与被包装函数参数直接的名字冲突。 例如，如果装饰器 `@optional_debug` 作用在一个已经拥有一个 `debug` 参数的函数上时会有问题。 这里我们增加了一步名字检查。

上面的方案还可以更完美一点，因为精明的程序员应该发现了被包装函数的函数签名其实是错误的。例如：

```
>>> @optional_debug
... def add(x,y):
...     return x+y
...
>>> import inspect
>>> print(inspect.signature(add))
(x, y)
>>>
```

通过如下的修改，可以解决这个问题：

```
from functools import wraps
import inspect

def optional_debug(func):
    if 'debug' in inspect.getargspec(func).args:
        raise TypeError('debug argument already defined')

    @wraps(func)
    def wrapper(*args, debug=False, **kwargs):
        if debug:
            print('Calling', func.__name__)
        return func(*args, **kwargs)
```

```
    sig = inspect.signature(func)
    parms = list(sig.parameters.values())
    parms.append(inspect.Parameter('debug',
                inspect.Parameter.KEYWORD_ONLY,
                default=False))
    wrapper.__signature__ = sig.replace(parameters=parms)
    return wrapper
```

通过这样的修改，包装后的函数签名就能正确的显示 `debug` 参数的存在了。例如：

```
>>> @optional_debug
... def add(x,y):
...     return x+y
...
>>> print(inspect.signature(add))
(x, y, *, debug=False)
>>> add(2,3)
5
>>>
```

参考9.16小节获取更多关于函数签名的信息。