7.8 减少可调用对象的参数个数¶

问题¶

你有一个被其他python代码使用的callable对象，可能是一个回调函数或者是一个处理器， 但是它的参数太多了，导致调用时出错。

解决方案¶

如果需要减少某个函数的参数个数，你可以使用 `functools.partial()` 。 `partial()` 函数允许你给一个或多个参数设置固定的值，减少接下来被调用时的参数个数。 为了演示清楚，假设你有下面这样的函数：

```
def spam(a, b, c, d):
    print(a, b, c, d)
```

现在我们使用 `partial()` 函数来固定某些参数值：

```
>>> from functools import partial
>>> s1 = partial(spam, 1) # a = 1
>>> s1(2, 3, 4)
1 2 3 4
>>> s1(4, 5, 6)
1 4 5 6
>>> s2 = partial(spam, d=42) # d = 42
>>> s2(1, 2, 3)
1 2 3 42
>>> s2(4, 5, 5)
4 5 5 42
>>> s3 = partial(spam, 1, 2, d=42) # a = 1, b = 2, d = 42
>>> s3(3)
1 2 3 42
>>> s3(4)
1 2 4 42
>>> s3(5)
1 2 5 42
>>>
```

可以看出 `partial()` 固定某些参数并返回一个新的callable对象。这个新的callable接受未赋值的参数， 然后跟之前已经赋值过的参数合并起来，最后将所有参数传递给原始函数。

讨论¶

本节要解决的问题是让原本不兼容的代码可以一起工作。下面我会列举一系列的例子。

第一个例子是，假设你有一个点的列表来表示(x,y)坐标元组。 你可以使用下面的函数来计算两点之间的距离：

```
points = [ (1, 2), (3, 4), (5, 6), (7, 8) ]

import math
def distance(p1, p2):
    x1, y1 = p1
    x2, y2 = p2
    return math.hypot(x2 - x1, y2 - y1)
```

现在假设你想以某个点为基点，根据点和基点之间的距离来排序所有的这些点。 列表的 `sort()` 方法接受一个关键字参数来自定义排序逻辑， 但是它只能接受一个单个参数的函数(distance()很明显是不符合条件的)。 现在我们可以通过使用 `partial()` 来解决这个问题：

```
>>> pt = (4, 3)
>>> points.sort(key=partial(distance,pt))
>>> points
[(3, 4), (1, 2), (5, 6), (7, 8)]
>>>
```

更进一步，`partial()` 通常被用来微调其他库函数所使用的回调函数的参数。 例如，下面是一段代码，使用 `multiprocessing` 来异步计算一个结果值， 然后这个值被传递给一个接受一个result值和一个可选logging参数的回调函数：

```
def output_result(result, log=None):
    if log is not None:
        log.debug('Got: %r', result)

# A sample function
def add(x, y):
    return x + y

if __name__ == '__main__':
    import logging
    from multiprocessing import Pool
    from functools import partial

    logging.basicConfig(level=logging.DEBUG)
    log = logging.getLogger('test')

    p = Pool()
    p.apply_async(add, (3, 4), callback=partial(output_result, log=log))
    p.close()
    p.join()
```

当给 `apply_async()` 提供回调函数时，通过使用 `partial()` 传递额外的 `logging` 参数。 而 `multiprocessing` 对这些一无所知——它仅仅只是使用单个值来调用回调函数。

作为一个类似的例子，考虑下编写网络服务器的问题，`socketserver` 模块让它变得很容易。 下面是个简单的echo服务器：

```
from socketserver import StreamRequestHandler, TCPServer

class EchoHandler(StreamRequestHandler):
    def handle(self):
        for line in self.rfile:
            self.wfile.write(b'GOT:' + line)

serv = TCPServer(('', 15000), EchoHandler)
serv.serve_forever()
```

不过，假设你想给EchoHandler增加一个可以接受其他配置选项的 `__init__` 方法。比如：

```
class EchoHandler(StreamRequestHandler):
    # ack is added keyword-only argument. *args, **kwargs are
    # any normal parameters supplied (which are passed on)
    def __init__(self, *args, ack, **kwargs):
        self.ack = ack
        super().__init__(*args, **kwargs)

    def handle(self):
        for line in self.rfile:
            self.wfile.write(self.ack + line)
```

这么修改后，我们就不需要显式地在TCPServer类中添加前缀了。 但是你再次运行程序后会报类似下面的错误：

```
Exception happened during processing of request from ('127.0.0.1', 59834)
Traceback (most recent call last):
...
TypeError: __init__() missing 1 required keyword-only argument: 'ack'
```

初看起来好像很难修正这个错误，除了修改 `socketserver` 模块源代码或者使用某些奇怪的方法之外。 但是，如果使用 `partial()` 就能很轻松的解决——给它传递 `ack` 参数的值来初始化即可，如下：

```
from functools import partial
serv = TCPServer(('', 15000), partial(EchoHandler, ack=b'RECEIVED:'))
serv.serve_forever()
```

在这个例子中， `__init__()` 方法中的ack参数声明方式看上去很有趣，其实就是声明ack为一个强制关键字参数。 关于强制关键字参数问题我们在7.2小节我们已经讨论过了，读者可以再去回顾一下。

很多时候 `partial()` 能实现的效果，lambda表达式也能实现。比如，之前的几个例子可以使用下面这样的表达式：

```
points.sort(key=lambda p: distance(pt, p))
p.apply_async(add, (3, 4), callback=lambda result: output_result(result,log))
serv = TCPServer(('', 15000),
        lambda *args, **kwargs: EchoHandler(*args, ack=b'RECEIVED:', **kwargs))
```

这样写也能实现同样的效果，不过相比而已会显得比较臃肿，对于阅读代码的人来讲也更加难懂。 这时候使用 `partial()` 可以更加直观的表达你的意图(给某些参数预先赋值)。