8.3 让对象支持上下文管理协议¶

问题¶

你想让你的对象支持上下文管理协议(with语句)。

解决方案¶

为了让一个对象兼容 `with` 语句，你需要实现 `__enter__()` 和 `__exit__()` 方法。 例如，考虑如下的一个类，它能为我们创建一个网络连接：

```
from socket import socket, AF_INET, SOCK_STREAM

class LazyConnection:
    def __init__(self, address, family=AF_INET, type=SOCK_STREAM):
        self.address = address
        self.family = family
        self.type = type
        self.sock = None

    def __enter__(self):
        if self.sock is not None:
            raise RuntimeError('Already connected')
        self.sock = socket(self.family, self.type)
        self.sock.connect(self.address)
        return self.sock

    def __exit__(self, exc_ty, exc_val, tb):
        self.sock.close()
        self.sock = None
```

这个类的关键特点在于它表示了一个网络连接，但是初始化的时候并不会做任何事情(比如它并没有建立一个连接)。 连接的建立和关闭是使用 `with` 语句自动完成的，例如：

```
from functools import partial

conn = LazyConnection(('www.python.org', 80))
# Connection closed
with conn as s:
    # conn.__enter__() executes: connection open
    s.send(b'GET /index.html HTTP/1.0\r\n')
    s.send(b'Host: www.python.org\r\n')
    s.send(b'\r\n')
    resp = b''.join(iter(partial(s.recv, 8192), b''))
    # conn.__exit__() executes: connection closed
```

讨论¶

编写上下文管理器的主要原理是你的代码会放到 `with` 语句块中执行。 当出现 `with` 语句的时候，对象的 `__enter__()` 方法被触发， 它返回的值(如果有的话)会被赋值给 `as` 声明的变量。然后， `with` 语句块里面的代码开始执行。 最后， `__exit__()` 方法被触发进行清理工作。

不管 `with` 代码块中发生什么，上面的控制流都会执行完，就算代码块中发生了异常也是一样的。 事实上，`__exit__()` 方

不管 with 代码块中发生什么，上面的控制流都会执行完，就算代码块中发生了异常也是一样的。事实上，__exit__() 方法的第三个参数包含了异常类型、异常值和追溯信息(如果有的话)。 __exit__() 方法能自己决定怎样利用这个异常信息，或者忽略它并返回一个None值。 如果 __exit__() 返回 True ，那么异常会被清空，就好像什么都没发生一样， with 语句后面的程序继续在正常执行。

还有一个细节问题就是 LazyConnection 类是否允许多个 with 语句来嵌套使用连接。 很显然，上面的定义中一次只能允许一个socket连接，如果正在使用一个socket的时候又重复使用 with 语句， 就会产生一个异常了。不过你可以像下面这样修改下上面的实现来解决这个问题：

```
from socket import socket, AF_INET, SOCK_STREAM

class LazyConnection:
    def __init__(self, address, family=AF_INET, type=SOCK_STREAM):
        self.address = address
        self.family = family
        self.type = type
        self.connections = []

    def __enter__(self):
        sock = socket(self.family, self.type)
        sock.connect(self.address)
        self.connections.append(sock)
        return sock

    def __exit__(self, exc_ty, exc_val, tb):
        self.connections.pop().close()

# Example use
from functools import partial

conn = LazyConnection(('www.python.org', 80))
with conn as s1:
    pass
    with conn as s2:
        pass
        # s1 and s2 are independent sockets
```

在第二个版本中， LazyConnection 类可以被看做是某个连接工厂。在内部，一个列表被用来构造一个栈。 每次 __enter__() 方法执行的时候，它复制创建一个新的连接并将其加入到栈里面。 __exit__() 方法简单的从栈中弹出最后一个连接并关闭它。 这里稍微有点难理解，不过它能允许嵌套使用 with 语句创建多个连接，就如上面演示的那样。

在需要管理一些资源比如文件、网络连接和锁的编程环境中，使用上下文管理器是很普遍的。 这些资源的一个主要特征是它们必须被手动的关闭或释放来确保程序的正确运行。 例如，如果你请求了一个锁，那么你必须确保之后释放了它，否则就可能产生死锁。 通过实现 __enter__() 和 __exit__() 方法并使用 with 语句可以很容易的避免这些问题， 因为 __exit__() 方法可以让你无需担心这些了。

在 contextmanager 模块中有一个标准的上下文管理方案模板，可参考9.22小节。 同时在12.6小节中还有一个对本节示例程序的线程安全的修改版。