9.22 定义上下文管理器的简单方法¶

问题¶

你想自己去实现一个新的上下文管理器，以便使用with语句。

解决方案¶

实现一个新的上下文管理器的最简单的方法就是使用 `contexlib` 模块中的 `@contextmanager` 装饰器。 下面是一个实现了代码块计时功能的上下文管理器例子：

```
import time
from contextlib import contextmanager

@contextmanager
def timethis(label):
    start = time.time()
    try:
        yield
    finally:
        end = time.time()
        print('{}: {}'.format(label, end - start))

# Example use
with timethis('counting'):
    n = 10000000
    while n > 0:
        n -= 1
```

在函数 `timethis()` 中，`yield` 之前的代码会在上下文管理器中作为 `__enter__()` 方法执行， 所有在 `yield` 之后的代码会作为 `__exit__()` 方法执行。 如果出现了异常，异常会在yield语句那里抛出。

下面是一个更加高级一点的上下文管理器，实现了列表对象上的某种事务：

```
@contextmanager
def list_transaction(orig_list):
    working = list(orig_list)
    yield working
    orig_list[:] = working
```

这段代码的作用是任何对列表的修改只有当所有代码运行完成并且不出现异常的情况下才会生效。 下面我们来演示一下：

```
>>> items = [1, 2, 3]
>>> with list_transaction(items) as working:
...     working.append(4)
...     working.append(5)
...
>>> items
[1, 2, 3, 4, 5]
>>> with list_transaction(items) as working:
...     working.append(6)
...     working.append(7)
...     raise RuntimeError('oops')
...
Traceback (most recent call last):
    File "<stdin>", line 4, in <module>
RuntimeError: oops
>>> items
[1, 2, 3, 4, 5]
>>>
```

讨论¶

通常情况下，如果要写一个上下文管理器，你需要定义一个类，里面包含一个 `__enter__()` 和一个 `__exit__()` 方法，如下所示：

```
import time

class timethis:
    def __init__(self, label):
        self.label = label

    def __enter__(self):
        self.start = time.time()

    def __exit__(self, exc_ty, exc_val, exc_tb):
        end = time.time()
        print('{}: {}'.format(self.label, end - self.start))
```

尽管这个也不难写，但是相比较写一个简单的使用 `@contextmanager` 注解的函数而言还是稍显乏味。

`@contextmanager` 应该仅仅用来写自包含的上下文管理函数。 如果你有一些对象(比如一个文件、网络连接或锁)，需要支持 `with` 语句，那么你就需要单独实现 `__enter__()` 方法和 `__exit__()` 方法。