12.6 保存线程的状态信息¶

问题¶

你需要保存正在运行线程的状态，这个状态对于其他的线程是不可见的。

解决方案¶

有时在多线程编程中，你需要只保存当前运行线程的状态。 要这么做，可使用 `thread.local()` 创建一个本地线程存储对象。 对这个对象的属性的保存和读取操作都只会对执行线程可见，而其他线程并不可见。

作为使用本地存储的一个有趣的实际例子， 考虑在8.3小节定义过的 `LazyConnection` 上下文管理器类。 下面我们对它进行一些小的修改使得它可以适用于多线程：

```
from socket import socket, AF_INET, SOCK_STREAM
import threading

class LazyConnection:
    def __init__(self, address, family=AF_INET, type=SOCK_STREAM):
        self.address = address
        self.family = AF_INET
        self.type = SOCK_STREAM
        self.local = threading.local()

    def __enter__(self):
        if hasattr(self.local, 'sock'):
            raise RuntimeError('Already connected')
        self.local.sock = socket(self.family, self.type)
        self.local.sock.connect(self.address)
        return self.local.sock

    def __exit__(self, exc_ty, exc_val, tb):
        self.local.sock.close()
        del self.local.sock
```

代码中，自己观察对于 `self.local` 属性的使用。 它被初始化为一个 `threading.local()` 实例。 其他方法操作被存储为 `self.local.sock` 的套接字对象。 有了这些就可以在多线程中安全的使用 `LazyConnection` 实例了。例如：

```
from functools import partial
def test(conn):
    with conn as s:
        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''))

    print('Got {} bytes'.format(len(resp)))

if __name__ == '__main__':
    conn = LazyConnection(('www.python.org', 80))

    t1 = threading.Thread(target=test, args=(conn,))
    t2 = threading.Thread(target=test, args=(conn,))
    t1.start()
    t2.start()
    t1.join()
    t2.join()
```

它之所以行得通的原因是每个线程会创建一个自己专属的套接字连接（存储为self.local.sock）。 因此，当不同的线程执行套接字操作时，由于操作的是不同的套接字，因此它们不会相互影响。

讨论¶

在大部分程序中创建和操作线程特定状态并不会有什么问题。 不过，当出了问题的时候，通常是因为某个对象被多个线程使用到，用来操作一些专用的系统资源， 比如一个套接字或文件。你不能让所有线程共享一个单独对象， 因为多个线程同时读和写的时候会产生混乱。 本地线程存储通过让这些资源只能在被使用的线程中可见来解决这个问题。

本节中，使用 `thread.local()` 可以让 `LazyConnection` 类支持一个线程一个连接， 而不是对于所有的进程都只有一个连接。

其原理是，每个 `threading.local()` 实例为每个线程维护着一个单独的实例字典。 所有普通实例操作比如获取、修改和删除值仅仅操作这个字典。 每个线程使用一个独立的字典就可以保证数据的隔离了。