ThreadLocal

754次阅读

在多线程环境下，每个线程都有自己的数据。一个线程使用自己的局部变量比使用全局变量好，因为局部变量只有线程自己能看见，不会影响其他线程，而全局变量的修改必须加锁。

但是局部变量也有问题，就是在函数调用的时候，传递起来很麻烦：

```
def process_student(name):
    std = Student(name)
    # std是局部变量，但是每个函数都要用它，因此必须传进去：
    do_task_1(std)
    do_task_2(std)

def do_task_1(std):
    do_subtask_1(std)
    do_subtask_2(std)

def do_task_2(std):
    do_subtask_2(std)
    do_subtask_2(std)
```

每个函数一层一层调用都这么传参数那还得了？用全局变量？也不行，因为每个线程处理不同的Student对象，不能共享。

如果用一个全局dict存放所有的Student对象，然后以thread自身作为key获得线程对应的Student对象如何？

```
global_dict = {}

def std_thread(name):
    std = Student(name)
    # 把std放到全局变量global_dict中：
    global_dict[threading.current_thread()] = std
    do_task_1()
    do_task_2()

def do_task_1():
    # 不传入std，而是根据当前线程查找：
    std = global_dict[threading.current_thread()]
    ...

def do_task_2():
    # 任何函数都可以查找出当前线程的std变量：
    std = global_dict[threading.current_thread()]
    ...
```

这种方式理论上是可行的，它最大的优点是消除了std对象在每层函数中的传递问题，但是，每个函数获取std的代码有点丑。

有没有更简单的方式？

ThreadLocal应运而生，不用查找dict，ThreadLocal帮你自动做这件事：

```
import threading

# 创建全局ThreadLocal对象:
local_school = threading.local()
```

```
def process_student():
    print 'Hello, %s (in %s)' % (local_school.student, threading.current_thread().name)

def process_thread(name):
    # 绑定ThreadLocal的student:
    local_school.student = name
    process_student()

t1 = threading.Thread(target= process_thread, args=('Alice',), name='Thread-A')
t2 = threading.Thread(target= process_thread, args=('Bob',), name='Thread-B')
t1.start()
t2.start()
t1.join()
t2.join()
```

执行结果：

```
Hello, Alice (in Thread-A)
Hello, Bob (in Thread-B)
```

全局变量local_school就是一个ThreadLocal对象，每个Thread对它都可以读写student属性，但互不影响。你可以把local_school看成全局变量，但每个属性如local_school.student都是线程的局部变量，可以任意读写而互不干扰，也不用管理锁的问题，ThreadLocal内部会处理。

可以理解为全局变量local_school是一个dict，不但可以用local_school.student，还可以绑定其他变量，如local_school.teacher等等。

ThreadLocal最常用的地方就是为每个线程绑定一个数据库连接，HTTP请求，用户身份信息等，这样一个线程的所有调用到的处理函数都可以非常方便地访问这些资源。
