多线程的使用

创建一个子线程

```
import threading
t1 = threading.Thread(target=run_thread, args=(5,)) #run_thread是线程需要运
#启动线程
t1.start()
#主线程等待子线程完成
t1.join()
```

创建线程锁

```
lock = threading.Lock()

def run_thread(n):
    for i in range(100000):
        # 先要获取锁:
        lock.acquire()
        try:
            # 放心地改吧:
            change_it(n)
        finally:
            # 改完了一定要释放锁:
            lock.release()
```

ThreadLocal

```
import threading

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

```
5
6  def process_student():
7      print 'Hello, %s (in %s)' % (local_school.student, threading.current_t
8
9  def process_thread(name):
10     # 绑定ThreadLocal的student:
11     local_school.student = name
12     process_student()
13
14 t1 = threading.Thread(target= process_thread, args=('Alice',), name='Threa
15 t2 = threading.Thread(target= process_thread, args=('Bob',), name='Thread-
16 t1.start()
17 t2.start()
18 t1.join()
19 t2.join()
```

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。

多线程的问题

Python的线程虽然是真正的线程，但解释器执行代码时，有一个GIL锁：Global Interpreter Lock，任何Python线程执行前，必须先获得GIL锁，然后，每执行100条字节码，解释器就自动释放GIL锁，让别的线程有机会执行。这个GIL全局锁实际上把所有线程的执行代码都给上了锁，所以，多线程在Python中只能交替执行，即使100个线程跑在100核CPU上，也只能用到1个核。

多进程的使用

创建子进程

方式一

在非windows平台下可以直接使用os模块的fork命令拷贝一个子进程

```
import os
print 'Process (%s) start...' % os.getpid()
pid = os.fork()
if pid==0:
    print 'I am child process (%s) and my parent is %s.' % (os.getpid(), c
else:
    print 'I (%s) just created a child process (%s).' % (os.getpid(), pid)
```

方式二

使用multiprocessing模块创建一个Process对象

```
from multiprocessing import Process
import os

# 子进程要执行的代码
def run_proc(name):
    print 'Run child process %s (%s)...' % (name, os.getpid())

if __name__=='__main__':
    print 'Parent process %s.' % os.getpid()
    p = Process(target=run_proc, args=('test',))
    print 'Process will start.'
    p.start()
    p.join()
    print 'Process end.'
```

使用multiprocessing的进程池

```
from multiprocessing import Pool
import os, time, random
```

```
3
4  def long_time_task(name):
5      print 'Run task %s (%s)...' % (name, os.getpid())
6      start = time.time()
7      time.sleep(random.random() * 3)
8      end = time.time()
9      print 'Task %s runs %0.2f seconds.' % (name, (end - start))
10
11 if __name__=='__main__':
12     print 'Parent process %s.' % os.getpid()
13     p = Pool(5)#表示创建一个大小为5的进程池
14     for i in range(5):
15         p.apply_async(long_time_task, args=(i,))
16     print 'Waiting for all subprocesses done...'
17     p.close()#调用close之后无法向进程池添加新的进程
18     p.join()#等待所有进程结束
19     print 'All subprocesses done.'
```

进程间通信

```
1  from multiprocessing import Process, Queue
2  import os, time, random
3
4  # 写数据进程执行的代码:
5  def write(q):
6      for value in ['A', 'B', 'C']:
7          print 'Put %s to queue...' % value
8          q.put(value)
9          time.sleep(random.random())
10
11 # 读数据进程执行的代码:
12 def read(q):
13     while True:
14         value = q.get(True)
15         print 'Get %s from queue.' % value
16
17 if __name__=='__main__':
18     # 父进程创建Queue，并传给各个子进程:
```

```
    q = Queue()
    pw = Process(target=write, args=(q,))
    pr = Process(target=read, args=(q,))
    # 启动子进程pw，写入:
    pw.start()
    # 启动子进程pr，读取:
    pr.start()
    # 等待pw结束:
    pw.join()
    # pr进程里是死循环，无法等待其结束，只能强行终止:
    pr.terminate()
```

通过multiprocessing.managers模块还能实现分布式进程。