Python并发编程-多线程 threading

threading 下的 Thread类
函数
start() #开始线程执行
join() ##程序挂起直到线程结束
run() #定义线程的功能函数
setName() #设置线程名字
getName() #返回线程名字
isAlive() #布尔标志 表示这个线程是否在运行
isDaemon() # 返回线程是不是守护线程
setDaemon() #把线程设置为守护线程 一定要在start()之前调用

'''

实例1.threading模块应用

```
import threading
import time
print time.time()
def loop(nloop, nsec):
    print 'start loop', nloop, 'at: ', time.ctime()
    time.sleep(nsec)
    print 'loop', nloop, 'done at: ', time.ctime()
loops = [4 ,2]  ##定义两个函数一个执行4S一个执行2S
def main():
    print 'start at: ', time.ctime()
    threads = []
    nloops = len(loops)

    for i in range(nloops):
        t = threading.Thread(target=loop, args=(i, loops[i])) #创建了一个线程执行对象
        threads.append(t)  ## 将创建的对象放入到了列表 threads中
    for i in range(nloops):
        threads[i].start()   ###开始线程的执行
    for i in range(nloops):
        threads[i].join()   ##程序挂起 直到线程结束
    print 'ALL DONE at: ', time.ctime()
main()

print time.time()
```

执行结果:

```
1436673174.34
start at:  Sat Jul 11 23:52:54 2015
start loop 0 at:  Sat Jul 11 23:52:54 2015
start loop 1 at:  Sat Jul 11 23:52:54 2015
loop 1 done at:  Sat Jul 11 23:52:56 2015
loop 0 done at:  Sat Jul 11 23:52:58 2015
ALL DONE at:  Sat Jul 11 23:52:58 2015
1436673178.35
```

有4s可以执行完这个脚本

所有的线程都创建了之后 再一起调用start() 函数启动，而且不用理会锁的问题(分配锁获得锁 释放锁) 只要简单的对每个线程调用join()函数就可以了

实例2： 并发执行脚本中的函数

```
import threading
import time

print time.time()
def thread_a():
    time.sleep(5)
    print 'this is function a'

def thread_b():
    time.sleep(2)
    print 'this is function b'

def thread_c():
    time.sleep(3)
    print 'this is function c'

def thread_d():
    time.sleep(4)
    print 'this is function d'
func_list = filter((lambda x: x.startswith('thread_')), globals())

def main():
    threads = []
    for i in range(len(func_list)):
        t = threading.Thread(target=globals().get(func_list[i]), args=())
        threads.append(t)
    for i in range(len(func_list)):
        threads[i].start()
    for i in range(len(func_list)):
        threads[i].join()
main()
print time.time()
```

实例3. 并发指定数量的线程

```
import threading
import time

print time.time()
def thread_a():
    time.sleep(5)
    print 'this is function a'

def thread_b():
    time.sleep(2)
    print 'this is function b'

def thread_c():
    time.sleep(3)
    print 'this is function c'

def thread_d():
    time.sleep(4)
    print 'this is function d'
func_list = filter((lambda x: x.startswith('thread_')), globals())

def main():
    thread_num = 2
    while True:
        thread_count = min([thread_num, len(func_list)])
        threads = []
        if not func_list:
            break
        for i in range(thread_count):
            t = threading.Thread(target=globals().get(func_list.pop()), args=())
            threads.append(t)
        for i in range(thread_count):
            threads[i].start()
        for i in range(thread_count):
            threads[i].join()
main()
print time.time()
```

4.通过多线程去读日志文件无法实现节约时间的作用(计算密集型)

```
#coding:utf8
import time
import threading

def f1(filename):
    result = {}
    with open(str(filename) + '.log') as file:
        for line in file:
            sp = line.strip().split()
            if sp[6] == 'ios':
                result['ios'] = result.get('ios', 0) + 1
    return result


l = [20160915, 20160916]
def main():
    threads = []
    for j in range(len(l)):
        t = threading.Thread(target=f1, args=(l[j], ))
        threads.append(t)

    for j in range(len(l)):
        threads[j].start()
    for j in range(len(l)):
        threads[j].join()


beg = time.time()
main()
end = time.time()
print end - beg


beg = time.time()
for j in l:
    f1(j)
end = time.time()
print end - beg
```

执行结果
0.0584738254547 ##多线程
0.0507390499115 ## 直接顺次执行

##通过上面结果可以看到 反而不用多线程的方式计算会快一些 因为这种计算密集型的 Python多线程并不适用

5.通过多线程去获取页面html代码可以起到节约时间作用 (I/O密集型)

```
#coding:utf8
import urllib2
import time
import threading

def f1(url):

    request = urllib2.Request(url)
    beg = time.time()
    response = urllib2.urlopen(request)
    print  len(response.read())
    end = time.time()
    print url, end - beg



beg = time.time()
f1('http://www.baidu.com')
f1('http://www.qq.com')
f1('http://www.taobao.com')
end = time.time()
print end - beg



l = ['http://www.baidu.com', 'http://www.qq.com', 'http://www.taobao.com']
def main():
    threads = []
    for j in range(len(l)):
        t = threading.Thread(target=f1, args=(l[j], ))
        threads.append(t)
    for j in range(len(l)):
        threads[j].start()
    for j in range(len(l)):
        threads[j].join()

beg = time.time()
main()
end = time.time()
print end - beg
```

```
102637
http://www.baidu.com 0.859601020813
52180
http://www.qq.com 0.436904907227
130220
http://www.taobao.com 1.34308409691
2.63988304138   ##为所有时间相加
```

```
101908
http://www.baidu.com 2.83915400505
130220
http://www.taobao.com 3.80751895905
249295
http://www.qq.com 4.05713915825
4.05779409409  ##为最长的那个时间
```

可见处理I/O密集型的程序使用多线程可以起到节约时间的效果

4. 线程锁 Lock

当多个进程需要访问共享资源的时候，Lock可以用来避免访问的冲突。

1) 没有锁的情况下得到错误结果

```python
import time
import threading

def add():
    global count
    temp = count
    time.sleep(0.00001)
    count = temp + 1


if __name__ == "__main__":
    count = 0
    threading_list = []
    for _ in range(10):
        t = threading.Thread(target=add)
        t.start()
        threading_list.append(t)
    [ t.join() for t in threading_list ]
    print count
```

2）有锁的情况下得到正确结果

```
import time
import threading

def add():
    with lock:
        global count
        temp = count
        time.sleep(0.0001)
        count = temp + 1


if __name__ == "__main__":
    lock = threading.Lock()
    count = 0
    threading_list = []
    for _ in range(10):
        t = threading.Thread(target=add)
        t.start()
        threading_list.append(t)
    [ t.join() for t in threading_list ]
    print count
```