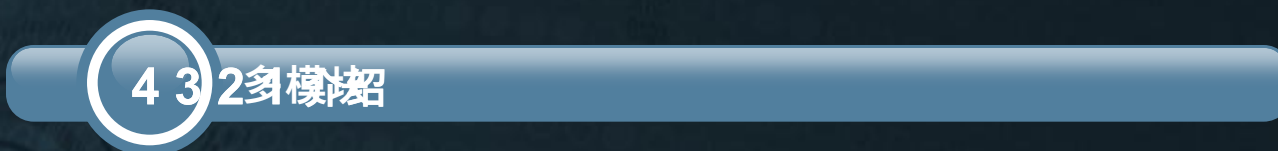
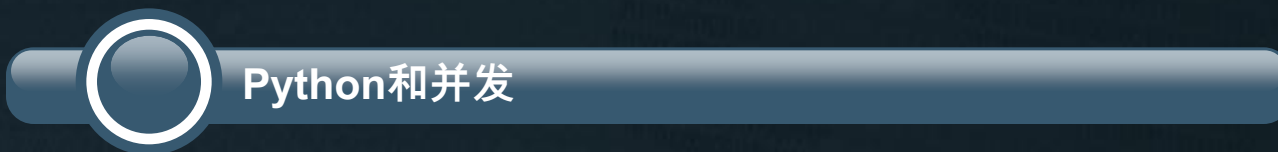
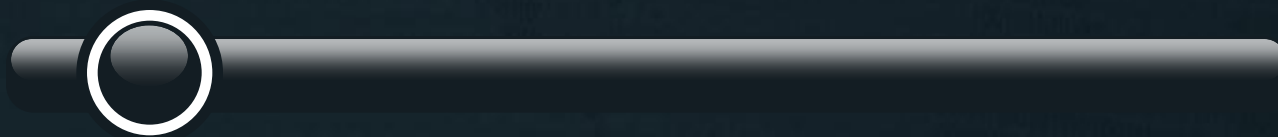
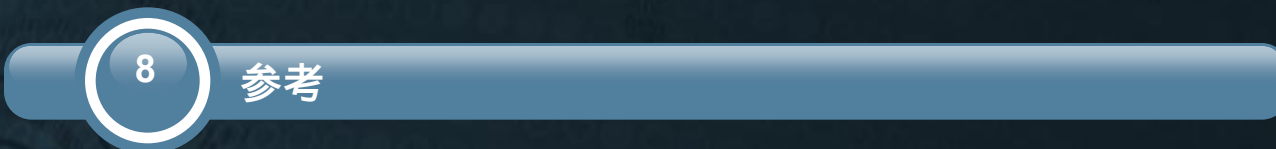
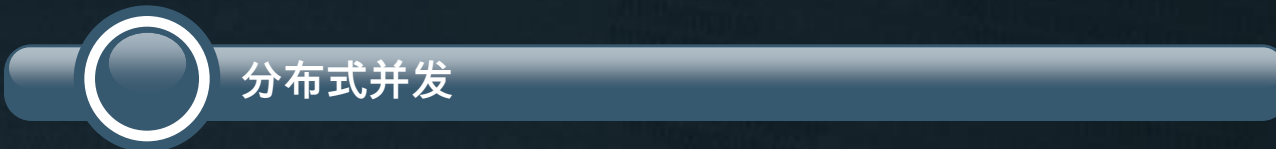



Python的多处理模块

阿里Alzabarah







介绍roduction

- 主题：是执行程序中的一个线程。阿卡，轻量级进程。
- 过程：是正在执行的计算机程序的一个实例。
 -
- 线程共享父，过程无共享的内存和状态。
- 过程中使用的进程间通信进行通信，线程没有。
 -



Python和并发 Concurrency

- Python中有三个并发模块：

线

穿线

多





Python和并发 Concurrency

- 主题：

- 提供了多线程工作的低级原。
- Python的第一个执行线程，它是旧的。
- 不包括在Python 3000。

- 线程：

- 在构建线程模块顶部更高级别的线程接口。



Python和并发 Concurrency

- 多 :

- 支持产卵过程。
- 提供本地和远程并发
- 新的Python 2.6。
- 解决了线程模块中的问题。



Python和并发 Concurrency

- 为什么新的模块？
 - Python的全局解释器锁，GIL，限制防止在多处理器机器上真正的并行。
- 什么是GIL？
 - 锁，必须获得一个线程进入翻译“的太空。
 - 锁确保只有一个线程在CPython的虚拟机在同一时间执行。



Python和并发 Concurrency

- GIL是如何工作的？
 - 它控制线程之间的控制的转移。Python解释器确定多久丝线“轮到运行，而不是硬件定时器。
 - Python使用OS线程为基础，但Python本身控制控制线程之间的传输。
- 基于上述原因，真正的并行荣获“与线程模块”发生。
- 于是，他们想出了多处理来解决这个问题。



Python和并发 Concurrency

- “不过，你的“再右边的GIL并不那么糟糕，因为你会觉得开始：你只需要撤消

洗脑从Windows和Java支持者谁似乎认为线程数接近同期活动的唯一途径“，吉多·范罗苏姆了。



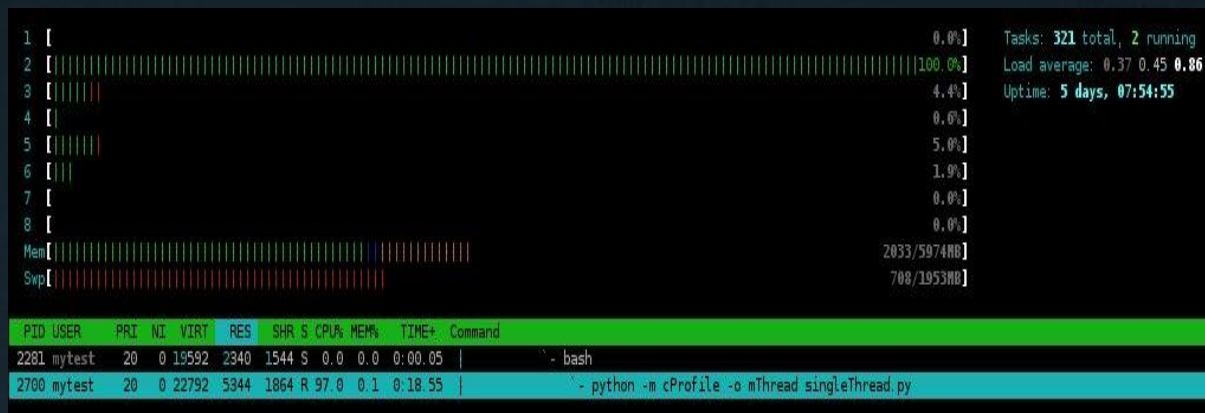
多线程 multiprocessing VS Threading

- 让“看看工作中的问题。我分析了在深入的书面杰西洛勒尔的代码。我用CPROFILE和pstats模块获得的代码是如何被处理的Python的想法。
- I“M在四核机测试程序，8 CPU”秒。



多线程 vs multiprocessing VS Threading

- 单线程：



- 该项目采取了52.810 CPU秒。



Multiprocessing VS Threading

s功能。多线程VS

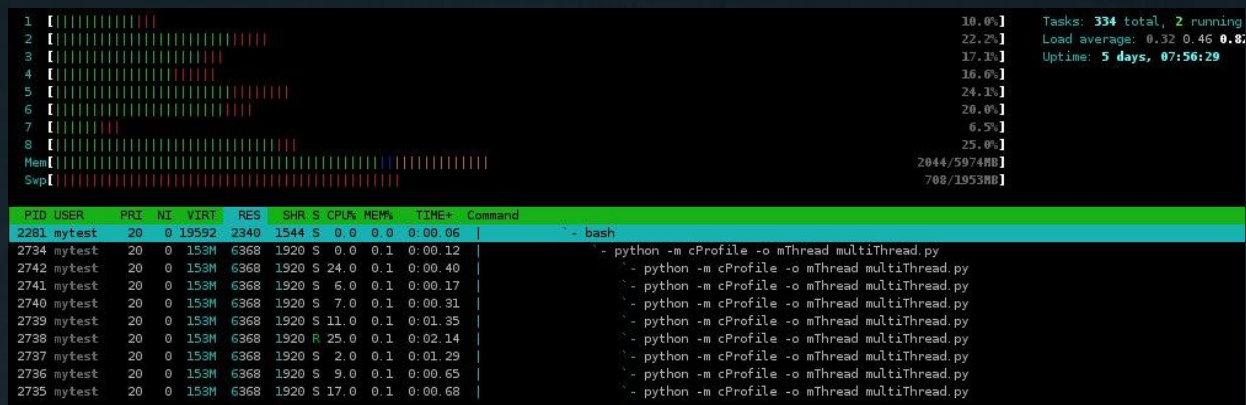
- 在大多数情况下，它被执行isPrime，sum_prime

```
cumtime  percall  filename:lineno(function)
52.810    52.810  <string>:1(<module>)
52.810    52.810  {execfile}
52.809    52.809  singleThread.py:1(<module>)
52.809     0.264  singleThread.py:17(sum_primes)
50.002     0.000  singleThread.py:3(isPrime)
 1.545     0.000  {math.sqrt}
 1.309     0.000  {math.ceil}
 0.010     0.000  {sum}
 0.000     0.000  {method 'disable' of '_lsprof.
```



多线程 vs multiprocessing VS Threading

- 多线程技术：



- 该项目采取了59.337 CPU秒。这是比它采取了同样的程序的单一版本的更多！



多线程 multiprocessing VS Threading

- 大部分时间是使用内置的方法获取！

```
cumtime  percall  filename:lineno(function)
59.313    0.125 {built-in method acquire}
54.303    6.788 threading.py:622(join)
0.001     0.000 threading.py:179(__init__)
0.001     0.001 threading.py:1(<module>)
59.337    59.337 multiThread.py:1(<module>)
59.337    59.337 {execfile}
0.001     0.000 Queue.py:107(put)
0.000     0.000 heapq.py:31(<module>)
0.000     0.000 {thread.start_new_thread}
0.000     0.000 threading.py:270(notify)
```



多线程 vs Processing VS Threading

- 等一下！内置的方法？线程获取方法是不是在代码
-

```
def isPrime(n):
    if n < 2:
        return False
    if n == 2:
        return True
    max = int(math.ceil(math.sqrt(n)))
    i = 2
    while i <= max :
        if n % i == 0:
            return False
        i += 1
    return True

def sum_primes(n):
    return sum([x for x in xrange(2,n) if isPrime(x)])

def do_work(q):
    while True:
        try:
            x = q.get(block=False)
            sum_primes(x)
        except Empty:
            break

if __name__ == "__main__":

    work_queue = Queue()
    for i in xrange(0,100000,500):
        work_queue.put(i)

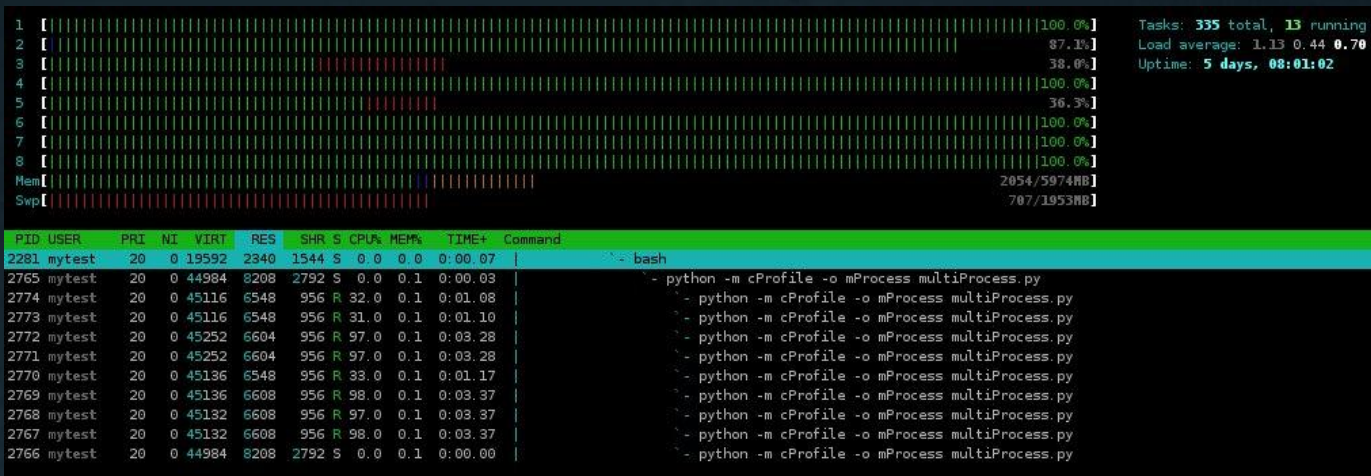
    threads = [ Thread(target=do_work, args=(work_queue,)) for i in range(8) ]
    for t in threads:
        t.start()
    for t in threads:
        t.join()
```



多线程 vs multiprocessing VS Threading

- 内置采集！这一定是GIL。

- 多进程：



- 只用了11.968秒。～5倍的速度！



Multiprocessing VS Threading

- 大部分的时间是在等待其他进程完成。多线程VS

```
cumtime  percall  filename:lineno(function)
 11.786    0.327  {posix.waitpid}
   0.166    0.021  {posix.fork}
   0.003    0.003  socket.py:44(<module>)
   0.002    0.001  sre_parse.py:385(_parse)
   0.004    0.004  __init__.py:43(<module>)
   0.001    0.001  socket.py:174(_socketobject)
   0.001    0.001  random.py:40(<module>)
   0.198    0.025  process.py:90(start)
   0.003    0.003  util.py:9(<module>)
   0.002    0.000  queues.py:73(put)
```



多线程 vs multiprocessing VS Threading

- 那么，如何多模块解决这个问题？
 - 它使用的子进程而不是线程。
 - 因此，它允许程序员充分利用给定的机器上的多个处理器。



多线程 vs multiprocessing VS Threading

- 线程/多语法之间的差异？差不多一样。
 - 线程：
 - 线 (目标= do_work , ARGS = (work_queue ,))
 - 多：
 - 处理 (目标= do_work , ARGS = (work_queue ,))
- 我“不会去覆盖所有的多处理模块提供的功能，但我将讨论什么是新的。该线程模块提供的任何功能也是多模块中。



并行处理模块 multiprocessing Module

- 请记住：
 - 进程共享什么。
 - 处理过的进程间通信信道进行通信。
- 这不符合线程模块的问题。
- Python开发者不得不寻找进程进行通信和共享日期的方式。否则，该模块将无法做到高效，因为它是。
 -



进程之间的交换对象 Object between Processes

- 多处理模块具有两个通信信道：





进程之间的交换对象 Object between Processes

- 队列：
 - 返回进程共享队列。
 - 任何咸菜，能够对象可以通过它。
 - 线程和进程安全的。
- 管道：
 - 返回一对连接对象的通过管连接。
 - 每个对象已经发送了在进程之间的通信中使用/ REC V方法。



进程之间的交换对象 Object between Processes

- 让“看看一个例子：
 - 排队简单的例子：
 - 该程序创建两个队列：
 - 任务：队列为int的范围。
 - 结果：队列为空。它是用来存储结果。
 - 然后创建n个工人，每个工人得到一个数据，数字，从共享队列，乘以2，并把它存储在结果队列。



进程之间的交换对象 Object between Processes

```
from multiprocessing import Queue, Process, current_process

def worker(tasks, results):
    # get a tasks
    t = tasks.get()
    # do operation
    result = t * 2
    # put the result in results queue
    results.put([current_process().name, t, "*", 20, "=", result])

if __name__ == '__main__':
    n = 100
    # create my tasks and results Queues.
    myTasks = Queue()
    myResults = Queue()
    # create n process :
    Workers = [ Process(target=worker, args=(myTasks, myResults)) for i in range(n) ]
    # start processes
    for each in Workers:
        each.start()

    # create tasks
    for each in range(n):
        myTasks.put(each)

    # get the results :
    while n :
        result = myResults.get()
        print "Res : " , result
        n -= 1
```



进程之间的交换对象 Object between Processes

- 观察：
 - 结果不是为了即使我们的任务的队列是为了。这是因为该程序并行运行。
 - Queue.get () 将数据返回给工人，并删除它。

- 输出部分：

```
Res : ['Process-16', 14, '*', 2, '=', 28]
Res : ['Process-12', 15, '*', 2, '=', 30]
Res : ['Process-17', 16, '*', 2, '=', 32]
Res : ['Process-18', 17, '*', 2, '=', 34]
Res : ['Process-20', 18, '*', 2, '=', 36]
Res : ['Process-21', 19, '*', 2, '=', 38]
Res : ['Process-22', 20, '*', 2, '=', 40]
Res : ['Process-19', 22, '*', 2, '=', 44]
Res : ['Process-24', 23, '*', 2, '=', 46]
Res : ['Process-25', 24, '*', 2, '=', 48]
Res : ['Process-26', 25, '*', 2, '=', 50]
Res : ['Process-27', 26, '*', 2, '=', 52]
Res : ['Process-28', 27, '*', 2, '=', 54]
Res : ['Process-58', 56, '*', 2, '=', 112]
Res : ['Process-59', 57, '*', 2, '=', 114]
Res : ['Process-60', 59, '*', 2, '=', 118]
Res : ['Process-63', 61, '*', 2, '=', 122]
Res : ['Process-65', 63, '*', 2, '=', 126]
Res : ['Process-61', 64, '*', 2, '=', 128]
```




Sharing state between 进程 processes

- 多模块有两种方式进程间共享状态：





Sharing state between processes

- 共享内存：
 - 蟒提供要被存储在共享存储器中的地图数据两种方法：
 - 价值：
 - 返回值是该对象的同步包装。
 - 阵：
 - 返回值是用于阵列的同步包装。



Sharing state between processes

- 服务器进程：
 - 经理对象控制保存Python对象服务器进程，并允许其他程序来操作它们。
- 什么是管理？
 - 控制它管理的共享对象服务器进程。
 - 它确保当有人修改它的共享对象获得的所有进程更新。



进程间共享状态 sharing state between processes

- 让“看看进程之间的共享状态的例子：
 - 该程序创建一个管理器列表，分享工人之间的n个，每一个工人都更新的索引。
 - 毕竟工人完成，新的清单打印到标准输出。



Sharing state between processes

- 服务器进程简单的例子：进程间共享状态

```
from multiprocessing import Manager, Process, current_process

# function that takes a list, every worker access on index. The index
# is actually the worker number. The worker will update the value of the index
# to be the old value to the power2.

def worker(aList):
    aList[ int(current_process().name.split("-")[1]) -1 ] = (int(current_process().name.split("-")[1]) -1) * \
        (int(current_process().name.split("-")[1]) -1)

if __name__ == '__main__':
    n = 100
    # create a Manager
    manager = Manager()
    # create a list that is managed by the manager
    l = manager.list()
    # fill the list with number from 0 to n
    l.extend(range(n))
    # create processes
    p = [ Process(target=worker, args=(l,)) for i in range(n -1)]

    # start the processes
    for each in p :
        each.start()
    # join the processes
    for each in p :
        each.join()
    # print the final result
    print "Final result : ", l
```



Sharing state between processes

- 观察：
 - 我们不用担心同步访问列表。经理采取的照顾。
 - 所有进程看到相同的列表，并充当一个共享列表上。
- 结果，当 $n = 10000$ ：进程之间共享状态

```
Final result : [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121,
144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529,
576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156,
1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764,
1849, 1936, 2025, 2116, 2209, 2304, 2401, 2500, 2601, 2704,
2809, 2916, 3025, 3136, 3249, 3364, 3481, 3600, 3721, 3844,
3969, 4096, 4225, 4356, 4489, 4624, 4761, 4900, 5041, 5184,
5329, 5476, 5625, 5776, 5929, 6084, 6241, 6400, 6561, 6724,
6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 8464,
8649, 8836, 9025, 9216, 9409, 9604, 9801]
```




模块 processing module

- 在过去10张幻灯片的摘要：
 - 沟通渠道：
 - 队列
 - 管道
 - 共享数据：
 - 共享内存：
 - 值
 - 阵列
 - 服务器：
 - 经理
- 让“S发现我们的模块中的其他很酷的功能：



Pool的游泳池worker

- 多处理器模块具有池类：
 - 分发工人之间的工作。
 - 收集返回值的列表。
- 您不必担心管理队列，进程，共享日期/统计自己。
- 它可以很容易地实现快速/简单的并发程序。
- 让我们来看一个例子：



Pool的游泳池

- 计划从文件中读取单词列表，返回列表的列表，每一个列表中包含一个词，它的长度。

```
from multiprocessing import Pool

def aFunction(x):
    return [ [len(z),z] for z in x ]

if __name__ == '__main__':
    # create a pool that has 8 workers
    pool = Pool(processes=8)
    # read the file in
    f = open('stopWord.txt','r')
    l = f.readlines()
    f.close()
    # remove new line character
    d = [ w.strip() for w in l ]
    # get the result asynchronously
    result = pool.apply_async(aFunction, [d])
    # print out the result
    print result.get()
```



Pool的游泳worker

- 观察：
 - 我们没有做到我们讲多少工人要在池中的程序旁边的任何工作。

- 结果的一部分：

```
[3, 'any'], [3, 'app'], [11, 'application'], [9, 'available'], [4, 'back'],  
[7, 'because'], [6, 'before'], [5, 'being'], [5, 'below'], [4, 'best'], [6, 'better'],  
[7, 'between'], [4, 'both'], [3, 'box'], [4, 'buch'], [3, 'but'], [6, 'called'], [3, 'can'],  
[7, 'certain'], [6, 'change'], [5, 'check'], [5, 'click'], [5, 'could'], [6, 'create'],  
[7, 'created'], [8, 'creating'], [7, 'current'], [9, 'currently'], [3, 'did'],  
[9, 'different'], [4, 'does'], [5, 'doesn'], [3, 'don'], [4, 'down'], [4, 'each'],  
[4, 'easy'], [4, 'edit'], [3, 'end'], [3, 'etc'], [4, 'even'], [5, 'first'], [5, 'fixme'],  
[9, 'following'], [5, 'found'], [4, 'full'], [4, 'full'], [7, 'general'], [8, 'generate'],  
[3, 'get'], [4, 'give'], [5, 'going'], [4, 'good'], [3, 'got'], [3, 'had'], [3, 'has'],  
[4, 'have'], [4, 'help'], [4, 'here'], [9, 'highlight'], [4, 'home'], [7, 'instead'],  
[3, 'its'], [4, 'just'], [3, 'key'], [4, 'know'], [4, 'like'], [4, 'line'], [4, 'link'],  
[5, 'links'], [4, 'list'], [4, 'look'], [4, 'many'], [4, 'menu'], [5, 'might'], [4, 'more'],
```

- 它不能比这更容易 •



分布式并发 concurrency

- 回想一下，在多处理器模块的管理控制管理共享对象的服务器进程。
- 该服务器可以远程访问和共享对象可以被分发到多个客户端。每当客户端更新共享的对象，其它客户将会看到的变化。



Distributed concurrency

- 要创建服务器：
 - 创建继承BaseManager类的类。
 - 调用类方法“注册”的名称分配给你想要分享的内容。
 - 定义您的服务器侦听的地址。
 - 调用函数get_server和serve_forever运行服务器。
- 要创建客户端：



受限并发 (limited concurrency)

- 注册该服务器共享对象的名称。
 - 连接到服务器的地址。
 - 调用共享对象的名称。
- 让我们看看一个实施例：

```
# Server :

# create class that inherit the BaseManager class
class QueueManager(BaseManager):
    pass
# register our queue
QueueManager.register('get_queue', callable=lambda:queue)
# define the address of the server
m = QueueManager(address=('', 50000))
# get the server
s = m.get_server()
# run for ever
s.serve_forever()
#####

# client :
from multiprocessing.managers import BaseManager
class QueueManager(BaseManager):
    pass
QueueManager.register('get_queue')
m = QueueManager(address=('127.0.0.1', 50000))
m.connect()
queue = m.get_queue()
queue.put("Hi There")
```



Conclusion

- 多处理模块是一个强大的除了蟒蛇。它解决了GIL问题，并介绍了简单的方法来实现真正的并行性。
- 实现并发程序是不容易的，但与此模块的工作方式，我认为它使程序员工作更容易。



信用使用是由于credit is due

- 在滑板10的例子是基于对PyWorks，亚特兰大2008呈现由Jesse洛勒尔一个例子。
- 我的GIL问题的解释和解决方案均杰西洛勒尔介绍和也由诺曼·马特夫和徐诚斌的教程。



References

- Python 2.6中的文件，
<http://docs.python.org/library/multiprocessing.html>
- PyMOTW由Doug赫尔曼，
<http://www.doughellmann.com/PyMOTW/>
- “教程上的线程与Python编程”，由诺曼·马特夫和徐诚，美国加州大学戴维斯分校。

Thank You!

