

公告

wiki和教程: www.pythonav.com

免费教学视频: B站: 凸头统治地球

高级专题教程: 网易云课堂: 武沛齐



Python技术交流群: 737658057

软件测试开发交流群: 721023555

昵称: 武沛齐
园龄: 8年
粉丝: 9956
关注: 44
+加关注

我的标签

- Python(17)
- ASP.NET MVC(15)
- python之路(7)
- Tornado源码分析(5)
- 每天一道Python面试题(5)
- crm项目(4)
- 面试都在问什么? (2)
- Python开源组件 - Tyrion(1)
- Python面试315题(1)
- Python企业面试题讲解(1)

积分与排名

积分 - 425671
排名 - 780

随笔分类

- JavaScript(1)
- MVC(15)
- Python(17)
- 面试都在问什么系列? 【图】(2)
- 其他(37)

随笔 - 140 文章 - 164 评论 - 891

Python之路【第七篇】：线程、进程和协程

Python线程

Threading用于提供线程相关的操作，线程是应用程序中工作的最小单元。

```
1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3  import threading
4  import time
5
6  def show(arg):
7      time.sleep(1)
8      print 'thread'+str(arg)
9
10 for i in range(10):
11     t = threading.Thread(target=show, args=(i,))
12     t.start()
13
14 print 'main thread stop'
```

上述代码创建了10个“前台”线程，然后控制器就交给了CPU，CPU根据指定算法进行调度，分片执行指令。

更多方法：

- start 线程准备就绪，等待CPU调度
- setName 为线程设置名称
- getName 获取线程名称
- setDaemon 设置为后台线程或前台线程（默认）
如果是后台线程，主线程执行过程中，后台线程也在进行，主线程执行完毕后，后台线程不论成功与否，均停止
如果是前台线程，主线程执行过程中，前台线程也在进行，主线程执行完毕后，等待前台线程也执行完成后，程序停止
- join 逐个执行每个线程，执行完后继续往下执行，该方法使得多线程变得无意义
- run 线程被cpu调度后自动执行线程对象的run方法

```
import threading
import time

class MyThread(threading.Thread):
    def __init__(self, num):
        threading.Thread.__init__(self)
        self.num = num

    def run(self): #定义每个线程要运行的函数

        print("running on number:%s" %self.num)

        time.sleep(3)

if __name__ == '__main__':

    t1 = MyThread(1)
    t2 = MyThread(2)
```

企业面试题及答案(1)

请求响应(6)

设计模式(9)

微软C#(34)

随笔档案

2020年6月(1)

2020年5月(1)

2019年11月(1)

2019年10月(1)

2019年9月(4)

2018年12月(1)

2018年8月(1)

2018年5月(2)

2018年4月(1)

2017年8月(1)

2017年5月(1)

2017年3月(1)

2016年10月(1)

2016年7月(1)

2015年10月(1)

2015年8月(1)

2015年7月(1)

2015年6月(2)

2015年4月(2)

2014年3月(3)

2014年1月(3)

2013年12月(2)

2013年11月(2)

2013年10月(7)

2013年8月(17)

2013年7月(1)

2013年6月(14)

2013年5月(23)

2013年4月(3)

2013年3月(13)

2013年2月(1)

2012年11月(26)

相册

git(14)

最新评论

1. Re:MySQL练习题参考答案

13、查询至少学过学号为“001”同学所有课的其他同学学号和姓名； 需要考虑这种情况，001同学选了1、2、4，002同学选了1、2、3。这种情况，两位同学选课有交集，count(选课数)也一样，然是...

--BobAylN

2. Re:人生没有白走的路，每一步都算数
后续来了吗

--小鱼仔。

3. Re:MySQL练习题参考答案

```
t1.start()
t2.start()
```



线程锁 (Lock、RLock)

由于线程之间是进行随机调度，并且每个线程可能只执行n条执行之后，当多个线程同时修改同一条数据时可能会出现脏数据，所以，出现了线程锁 - 同一时刻允许一个线程执行操作。



```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
import threading
import time

gl_num = 0

def show(arg):
    global gl_num
    time.sleep(1)
    gl_num += 1
    print gl_num

for i in range(10):
    t = threading.Thread(target=show, args=(i,))
    t.start()

print 'main thread stop'
```



```
1  #!/usr/bin/env python
2  #coding:utf-8
3
4  import threading
5  import time
6
7  gl_num = 0
8
9  lock = threading.RLock()
10
11 def Func():
12     lock.acquire()
13     global gl_num
14     gl_num += 1
15     time.sleep(1)
16     print gl_num
17     lock.release()
18
19 for i in range(10):
20     t = threading.Thread(target=Func)
21     t.start()
```

信号量 (Semaphore)

互斥锁 同时只允许一个线程更改数据，而Semaphore是同时允许一定数量的线程更改数据，比如厕所所有3个坑，那最多只允许3个人上厕所，后面的人只能等里面有人出来了才能再进去。

```
1  import threading,time
2
3  def run(n):
4      semaphore.acquire()
```

11、查询没有学全所有课的同学的学号、姓名；
 select sid,sname from student
 where sid not in (select student_id
 from score ...

--BobAylIn

4. Re:1. 路过面了个试就拿到2个offer。是运气吗?

听了大王的讲课，觉得大王真是厉害，年轻有为！

--Xiyue666

5. Re: Celery

s3.py中 使用async为变量是关键字，会报错吧？

--killer-147

```

5     time.sleep(1)
6     print("run the thread: %s" %n)
7     semaphore.release()
8
9     if __name__ == '__main__':
10
11         num= 0
12         semaphore = threading.BoundedSemaphore(5) #最多允许5个线程同时运行
13         for i in range(20):
14             t = threading.Thread(target=run,args=(i,))
15             t.start()

```

事件 (event)

python线程的事件用于主线程控制其他线程的执行，事件主要提供了三个方法 set、wait、clear。

事件处理的机制：全局定义了一个“Flag”，如果“Flag”值为 False，那么当程序执行 event.wait 方法时就会阻塞，如果“Flag”值为True，那么event.wait 方法时便不再阻塞。

- clear: 将“Flag”设置为False
- set: 将“Flag”设置为True

```

1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3
4  import threading
5
6
7  def do(event):
8      print 'start'
9      event.wait()
10     print 'execute'
11
12
13     event_obj = threading.Event()
14     for i in range(10):
15         t = threading.Thread(target=do, args=(event_obj,))
16         t.start()
17
18     event_obj.clear()
19     inp = raw_input('input:')
20     if inp == 'true':
21         event_obj.set()

```

条件 (Condition)

使得线程等待，只有满足某条件时，才释放n个线程

```

1  import threading
2
3  def run(n):
4      con.acquire()
5      con.wait()
6      print("run the thread: %s" %n)
7      con.release()
8
9  if __name__ == '__main__':
10
11     con = threading.Condition()
12     for i in range(10):
13         t = threading.Thread(target=run, args=(i,))
14         t.start()
15
16     while True:
17         inp = input('>>>')

```

```

18         if inp == 'q':
19             break
20         con.acquire()
21         con.notify(int(inp))
22         con.release()

```



```

def condition_func():

    ret = False
    inp = input('>>>')
    if inp == '1':
        ret = True

    return ret

def run(n):
    con.acquire()
    con.wait_for(condition_func)
    print("run the thread: %s" %n)
    con.release()

if __name__ == '__main__':

    con = threading.Condition()
    for i in range(10):
        t = threading.Thread(target=run, args=(i,))
        t.start()

```



Timer

定时器，指定n秒后执行某操作

```

1  from threading import Timer
2
3
4  def hello():
5      print("hello, world")
6
7  t = Timer(1, hello)
8  t.start() # after 1 seconds, "hello, world" will be printed

```

Python 进程

```

1  from multiprocessing import Process
2  import threading
3  import time
4
5  def foo(i):
6      print 'say hi',i
7
8  for i in range(10):
9      p = Process(target=foo,args=(i,))
10     p.start()

```

注意：由于进程之间的数据需要各自持有一份，所以创建进程需要的非常大的开销。

进程数据共享

进程各自持有一份数据，默认无法共享数据



```
#!/usr/bin/env python
#coding:utf-8

from multiprocessing import Process
from multiprocessing import Manager

import time

li = []

def foo(i):
    li.append(i)
    print 'say hi',li

for i in range(10):
    p = Process(target=foo,args=(i,))
    p.start()

print 'ending',li
```



```
1 #方法一, Array
2 from multiprocessing import Process,Array
3 temp = Array('i', [11,22,33,44])
4
5 def Foo(i):
6     temp[i] = 100+i
7     for item in temp:
8         print i,'----->',item
9
10 for i in range(2):
11     p = Process(target=Foo,args=(i,))
12     p.start()
13
14 #方法二: manage.dict()共享数据
15 from multiprocessing import Process,Manager
16
17 manage = Manager()
18 dic = manage.dict()
19
20 def Foo(i):
21     dic[i] = 100+i
22     print dic.values()
23
24 for i in range(2):
25     p = Process(target=Foo,args=(i,))
26     p.start()
27     p.join()
```



```
'c': ctypes.c_char, 'u': ctypes.c_wchar,
'b': ctypes.c_byte, 'B': ctypes.c_ubyte,
'h': ctypes.c_short, 'H': ctypes.c_ushort,
'i': ctypes.c_int, 'I': ctypes.c_uint,
'l': ctypes.c_long, 'L': ctypes.c_ulong,
'f': ctypes.c_float, 'd': ctypes.c_double
```



```
from multiprocessing import Process, Queue

def f(i,q):
```

```

print(i,q.get())

if __name__ == '__main__':
    q = Queue()

    q.put("h1")
    q.put("h2")
    q.put("h3")

    for i in range(10):
        p = Process(target=f, args=(i,q,))
        p.start()

```

当创建进程时（非使用时），共享数据会被拿到子进程中，当进程中执行完毕后，再赋值给原值。

```

#!/usr/bin/env python
# -*- coding:utf-8 -*-

from multiprocessing import Process, Array, RLock

def Foo(lock,temp,i):
    """
    将第0个数加100
    """
    lock.acquire()
    temp[0] = 100+i
    for item in temp:
        print i,'----->',item
    lock.release()

lock = RLock()
temp = Array('i', [11, 22, 33, 44])

for i in range(20):
    p = Process(target=Foo,args=(lock,temp,i,))
    p.start()

```

进程池

进程池内部维护一个进程序列，当使用时，则去进程池中获取一个进程，如果进程池序列中没有可供使用的进程，那么程序就会等待，直到进程池中有可用进程为止。

进程池中有两个方法：

- apply
- apply_async

```

1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3  from multiprocessing import Process,Pool
4  import time
5
6  def Foo(i):
7      time.sleep(2)
8      return i+100
9
10 def Bar(arg):
11     print arg
12
13 pool = Pool(5)

```

```

14 #print pool.apply(Foo,(1,))
15 #print pool.apply_async(func =Foo, args=(1,)).get()
16
17 for i in range(10):
18     pool.apply_async(func=Foo, args=(i,),callback=Bar)
19
20 print 'end'
21 pool.close()
22 pool.join()#进程池中进程执行完毕后再关闭，如果注释，那么程序直接关闭。

```

协程

线程和进程的操作是由程序触发系统接口，最后的执行者是系统；协程的操作则是程序员。

协程存在的意义：对于多线程应用，CPU通过切片的方式来切换线程间的执行，线程切换时需要耗时（保存状态，下次继续）。协程，则只使用一个线程，在一个线程中规定某个代码块执行顺序。

协程的适用场景：当程序中存在大量不需要CPU的操作时（IO），适用于协程；

greenlet

```

1  #!/usr/bin/env python
2  # -*- coding:utf-8 -*-
3
4
5  from greenlet import greenlet
6
7
8  def test1():
9      print 12
10     gr2.switch()
11     print 34
12     gr2.switch()
13
14
15  def test2():
16     print 56
17     gr1.switch()
18     print 78
19
20  gr1 = greenlet(test1)
21  gr2 = greenlet(test2)
22  gr1.switch()

```

gevent


```

1  import gevent
2
3  def foo():
4      print('Running in foo')
5      gevent.sleep(0)
6      print('Explicit context switch to foo again')
7
8  def bar():
9      print('Explicit context to bar')
10     gevent.sleep(0)
11     print('Implicit context switch back to bar')
12
13  gevent.joinall([
14     gevent.spawn(foo),
15     gevent.spawn(bar),
16 ])

```

遇到IO操作自动切换：





```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

from gevent import monkey


monkey.patch_all()

import threading
import gevent
import time

def eat():
    print(threading.current_thread().getName())
    print('eat food 1')
    time.sleep(20)
    print('eat food 2')

def play():
    print(threading.current_thread().getName())
    print('play 1')
    time.sleep(20)
    print('play 2')

g1 = gevent.spawn(eat)
g2 = gevent.spawn(play)
gevent.joinall([g1, g2])
print('主')
```



```
from gevent import monkey; monkey.patch_all()
import gevent

import urllib2

def f(url):
    print('GET: %s' % url)
    resp = urllib2.urlopen(url)
    data = resp.read()
    print('%d bytes received from %s.' % (len(data), url))

gevent.joinall([
    gevent.spawn(f, 'https://www.python.org/'),
    gevent.spawn(f, 'https://www.yahoo.com/'),
    gevent.spawn(f, 'https://github.com/'),
])
```



作者：武沛齐

出处：<http://www.cnblogs.com/wupeiqi/>

本文版权归作者和博客园共有，欢迎转载，但未经作者同意必须保留此段声明，且在文章页面明显位置给出原文连接。

好文要顶

关注我

收藏该文





武沛齐

关注 - 44

粉丝 - 9956

+加关注

70

posted @ 2015-12-12 10:03 武沛齐 阅读(17691) 评论(2) 编辑 收藏

评论列表

#1楼 2016-05-31 23:46 NoSomking 回复 引用

引用

join 逐个执行每个线程，执行完毕后继续往下执行，该方法使得多线程变得无意义
关于join的方法有问题.如果用的不当会变成线程的顺序执行；合理使用时，是让线程顺序的退出.
比如：
from threading import Thread
def f(i):
 print i
 time.sleep(1)

def main1():
 print "Main1 start run"
 for i in range(3):
 t = Thread(target=f, args=(i,))
 t.start()
 t.join()
 print "Main1 END run"

def main2():
 print "Main2 start run"
 threads = []
 for i in range(3):
 t = Thread(target=f, args=(i,))
 threads.append(t)
 t.start()

 for i in threads:
 t.join()

 print "Main2 END run"

main1()
#main2()
上面的main1()是阻塞成顺执行线程. 而main2是先启动3线程后，当线程退出时才会阻塞顺序的退出，这时不会影响线程运行.

join我是这么理解的.

支持(3) 反对(0)

#2楼 2017-03-28 17:56 23云恋49枫 回复 引用





+1

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

发表评论

编辑 预览

B    

支持 Markdown

提交评论 退出 订阅评论

[Ctrl+Enter快捷键提交]

【推荐】超50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库

【推荐】了不起的开发者，挡不住的华为，园子里的品牌专区

【推荐】有道智云周年庆，API服务大放送，注册即送100元体验金！

【推荐】对阿里云庞大的技术产品一知半解？这里有16大领域超4000个技术问题答



相关博文：

- Python之路【第七篇续】：进程、线程、协程
 - 【python之路】进程线程
 - Python之路：进程、线程
 - Python之路【第七篇续】：进程、线程、协程
 - python之路-进程、线程
- » 更多推荐...

最新 IT 新闻：

- 美国强迫收购TikTok、限45天、还要中间费！李开复：做法不可思议
 - 联发科回应与华为签订采购大单：不评论单一客户讯息
 - 苹果回应Siri专利案：对诉讼感到失望，最高法院认证机构表示未侵权
 - 互联网免费时代终结 全面付费用户被套路
 - 2020胡润全球独角兽榜上的企业，有多少能留到明年
- » 更多新闻...