

Meet、

一入 IT 深似海，正在学习狗刨中.....

[博客园](#)[首页](#)[新随笔](#)[管理](#)[随笔 - 96](#) [文章 - 59](#) [评论 - 2](#)

python/进程同步锁

python/进程同步锁

python/同步锁

同步锁：通常被用来实现共享资源的同步访问，为每一个共享资源创建一个Lock对象当你需要访问该资源时，调用acquire方法来获取锁对象（如果其他线程已经获得该锁，则当前线程需等待期被释放），待资源访问完后，在调用release方法释放锁

实例如下：

```
1 #同步锁
2 import time    #导入时间模块
0 3 import threading #导入threading模块
   4 num=100      #设置一个全局变量
   5 lock=threading.Lock()
   6 def sudnum():    #定一个函数sudnum'
```

<	2017年12月						>
日	一	二	三	四	五	六	
26	27	28	29	30	1	2	
3	4	5	6	7	8	9	
10	11	12	13	14	15	16	
17	18	19	20	21	22	23	
24	25	26	27	28	29	30	
31	1	2	3	4	5	6	

常用链接

[我的随笔](#)[我的评论](#)[我的参与](#)[最新评论](#)[我的标签](#)

我的标签

[JavaScript\(7\)](#)[Vue框架\(4\)](#)

```
7     global num    #声明全局变量
8     lock.acquire()
9     temp=num      #读取全局变量num
10    time.sleep(0)   #增加一个休眠功能
11    num=temp-1     #把从全局拿来的变量进行减一的操作
12    lock.release()
13    l=[]          #在全局创建一个空了表
14    for i in range(100): #从0到100进行循环
15        t=threading.Thread(target=sudnum) #在循环中创建子线程，共创建100个
16        t.start()    #循环启动子线程
17        l.append(t)  #把循环创建的实例化添加到列表中
18
19    for f in l: #从列表里遍历内容给f：
20        f.join() #循环设置列表的内容结束
21
22    print('Result:',num) #打印通过多次子线程更改过的变量内容
23 运行结果
24 Result: 0
25
26 Process finished with exit code 0
```



死锁：

所谓死锁，就是指俩个或俩个以上的进程或线程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去

实例如下：

0



python目录(1)

随笔分类

Django(20)

JS-HTML-JQ(12)

Liunx(9)

MySQL(2)

Python(41)

积分与排名

积分 - 11461

排名 - 28231

最新评论

1. Re:python/MySQL（索引、执行计划、BDA、分页）

@培伦 共同学习...

--Meet、

2. Re:python/MySQL（索引、执行计划、BDA、分页）

谢谢

--培伦

阅读排行榜

1. Python/练习题(1562)

2. Python/ selectors模块及队列(916)

3. python/基础输出输入用法(544)

4. Python/MySQL（三、pymysql使用）(348)



```

1  #死锁
2  import threading    #导入模块
3  import time         #导入模块
4
5  mutexA = threading.Lock()    #把threading下Lock类赋值给mutexA
6  mutexB = threading.Lock()    #把threading下Lock类赋值给mutexB
7
8  class MyThread(threading.Thread):    #定义MyThread类 并继承threading下的Thread类功能
9
10     def __init__(self):    #初始化实例化
11         threading.Thread.__init__(self)    #初始父类实例化
12
13     def run(self):    #定义run函数 (此函数是固定函数)
14         self.fun1()    #实例化对象引用执行fun1函数
15         self.fun2()    #实例化对象引用执行fun2函数
16
17     def fun1(self):    #定义fun1函数
18
19         mutexA.acquire()    # 获取公共锁, 如果锁被占用, 则阻塞在这里, 等待锁的释放
20
21         print ("I am %s , get res: %s---%s" %(self.name, "ResA", time.time()))
22
23         mutexB.acquire()    # 获取公共锁, 如果锁被占用, 则阻塞在这里, 等待锁的释放
24         print ("I am %s , get res: %s---%s" %(self.name, "ResB", time.time()))
25         mutexB.release()    # 释放公共锁
26
27         mutexA.release()    # 释放公共锁
28
29     ~
0
1     def fun2(self):    #定义fun2函数
2
32         mutexB.acquire()    # 获取公共锁, 如果锁被占用, 则阻塞在这里, 等待锁的释放

```

5. Python/MySQL (二、表操作以及连接) (312)
6. python/MySQL (索引、执行计划、BDA、分页) (248)
7. python目录(170)
8. python Josnp(跨域)(166)
9. python/Djangof分页与自定义分页(138)
10. python Saltstack(131)

评论排行榜

1. python/MySQL (索引、执行计划、BDA、分页) (2)



```

33     print ("I am %s , get res: %s---%s" %(self.name, "ResB",time.time()))
34     time.sleep(0.2)
35
36     mutexA.acquire() # 获取公共锁, 如果锁被占用, 则阻塞在这里, 等待锁的释放
37     print ("I am %s , get res: %s---%s" %(self.name, "ResA",time.time()))
38     mutexA.release() # 释放公共锁
39
40     mutexB.release() # 释放公共锁
41
42 if __name__ == "__main__":
43
44     print("start-----%s"%time.time())
45
46     for i in range(0, 10):
47         my_thread = MyThread()
48         my_thread.start()
49
50 运行结果
51 start-----1494320240.1851542
52 I am Thread-1 , get res: ResA---1494320240.1856549
53 I am Thread-1 , get res: ResB---1494320240.1861556
54 I am Thread-1 , get res: ResB---1494320240.1861556
55 I am Thread-2 , get res: ResA---1494320240.186656

```



实际for循环10次，就是创建10个子线程，但是执行结果就运行到第二个子线程和第一子线程就出现了死锁的现象，第一个子线程把A锁释放掉时第二个子线程获取到A锁。第一个子线程释放了B锁，然后又获取了B锁，现在第二个子线程获得了A锁，第一个子线程获得了B锁，第二个子线程想要获取B锁，但是第一个子线程没有释放掉。第一个子线程想要获取到A锁 第二个子线程没有释放。就出现俩个子线程都相等对方释放获取的锁。

递归锁：



```
1 #递归锁
2 import threading    #导入模块
3 import time         #导入模块
4
5 RLock = threading.RLock()    #把threading下RLock类赋值给RLock
6
7
8 class MyThread(threading.Thread):    #定义MyThread类 并继承threading下的Thread类功能
9
10     def __init__(self):        #初始化实例化
11         threading.Thread.__init__(self)    #初始父类实例化
12
13     def run(self):            #定义run函数 (此函数是固定函数)
14         self.fun1()          #实例化对象引用执行fun1函数
15         self.fun2()          #实例化对象引用执行fun2函数
16
17     def fun1(self):          #定义fun1函数
18
19         RLock.acquire()    # 获取公共锁, 如果锁被占用, 则阻塞在这里, 等待锁的释放
20
21         print ("I am %s , na res: %s---%s" %(self.name, "ResA",time.time()))
22
23         RLock.acquire()    # 获取公共锁, 如果锁被占用, 则阻塞在这里, 等待锁的释放
24         print ("I am %s , na res: %s---%s" %(self.name, "ResB",time.time()))
25         RLock.release()    # 释放公共锁
26
27     def fun2(self):
28         RLock.release()    # 释放公共锁
29
30 ;
```



```
30     def fun2(self):    #定义fun2函数
31
32         RLock.acquire() # 获取公共锁, 如果锁被占用, 则阻塞在这里, 等待锁的释放
33         print ("I am %s , na res: %s---%s" %(self.name, "ResB",time.time()))
34         time.sleep(0.2)
35
36         RLock.acquire() # 获取公共锁, 如果锁被占用, 则阻塞在这里, 等待锁的释放
37         print ("I am %s , na res: %s---%s" %(self.name, "ResA",time.time()))
38         RLock.release() # 释放公共锁
39
40         RLock.release() # 释放公共锁
41
42 if __name__ == "__main__":
43
44     print("start-----%s"%time.time())
45
46     for i in range(0, 10):
47         my_thread = MyThread()
48         my_thread.start()
49
```

50 运行结果

```
51 start-----1494324391.4339159
52 I am Thread-1 , na res: ResA---1494324391.4344165
53 I am Thread-1 , na res: ResB---1494324391.4344165
54 I am Thread-1 , na res: ResB---1494324391.4344165
55 I am Thread-1 , na res: ResA---1494324391.63575
56 I am Thread-2 , na res: ResA---1494324391.63575
57 I am Thread-2 , na res: ResB---1494324391.63575
58 I am Thread-2 , na res: ResB---1494324391.63575
59 I am Thread-2 , na res: ResA---1494324391.836299
60 I am Thread-4 , na res: ResA---1494324391.836299
61 I am Thread-4 , na res: ResB---1494324391.8367958
```



```
62 I am Thread-4 , na res: ResB---1494324391.8367958
63 I am Thread-4 , na res: ResA---1494324392.040432
64 I am Thread-6 , na res: ResA---1494324392.040432
65 I am Thread-6 , na res: ResB---1494324392.040432
66 I am Thread-7 , na res: ResA---1494324392.040432
67 I am Thread-7 , na res: ResB---1494324392.040432
68 I am Thread-7 , na res: ResB---1494324392.040432
69 I am Thread-7 , na res: ResA---1494324392.2415655
70 I am Thread-9 , na res: ResA---1494324392.2415655
71 I am Thread-9 , na res: ResB---1494324392.2420657
72 I am Thread-9 , na res: ResB---1494324392.2420657
73 I am Thread-9 , na res: ResA---1494324392.4427023
74 I am Thread-3 , na res: ResA---1494324392.4427023
75 I am Thread-3 , na res: ResB---1494324392.4427023
76 I am Thread-3 , na res: ResB---1494324392.4427023
77 I am Thread-3 , na res: ResA---1494324392.643367
78 I am Thread-6 , na res: ResB---1494324392.643367
79 I am Thread-6 , na res: ResA---1494324392.8445525
80 I am Thread-8 , na res: ResA---1494324392.8445525
81 I am Thread-8 , na res: ResB---1494324392.8445525
82 I am Thread-8 , na res: ResB---1494324392.8445525
83 I am Thread-8 , na res: ResA---1494324393.0449915
84 I am Thread-5 , na res: ResA---1494324393.0449915
85 I am Thread-5 , na res: ResB---1494324393.0449915
86 I am Thread-5 , na res: ResB---1494324393.0449915
87 I am Thread-5 , na res: ResA---1494324393.2456653
88 I am Thread-10 , na res: ResA---1494324393.2456653
89 I am Thread-10 , na res: ResB---1494324393.2456653
^ I am Thread-10 , na res: ResB---1494324393.2456653
0 . I am Thread-10 , na res: ResA---1494324393.446061
!
93 Process finished with exit code 0
```





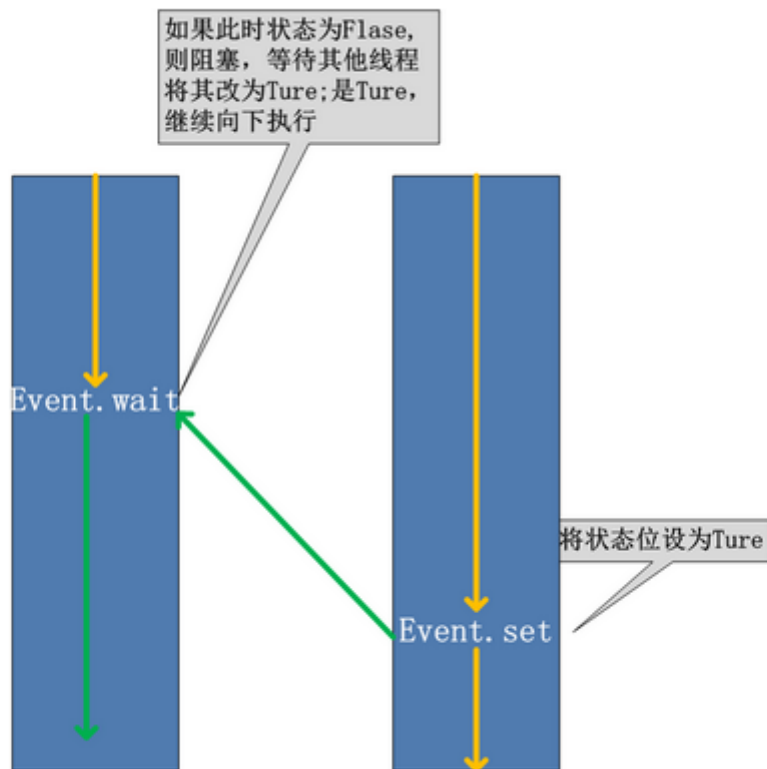
递归锁就是调用threading下的RLock类功能实现的，RLock它自带有计数功能，每让线程获取到以后就会就进行自加一的功能（RLock默认数值是0，只要RLock不是0线程就不能进行获取），只要进行一进行释放功能RLock就会进行自减一的功能直到为0时。

Event对象：

线程的一个关键特性是每个线程都是独立运行且状态不可预测。如果程序中的其他线程需要通过判断某个线程的状态来确定自己下一步的操作，这时线程同步问题就会变非常棘手。为了解决这些问题，我们需要使用threading库中的Event对象。对象包含一个可有线程设置的信号标志，它允许线程等待某些事情的发生。**在初始情况下，Event对象的标志为假**，name这个线程将会被一直阻塞至该标志为真。一个线程如果讲义个Event对象的信号标志设置为真，他将唤醒所有等待这个Event对象的线程。如果一个线程



等待一个已经被设置为真的Event对象，那么它将忽略这个事情，继续执行




```
1 1 event.isSet()返回event的状态值
2 2
3 3 event.wait() 如果event.isSet() == False将阻塞线程
4 4
5 5 event.set() 设置event的状态值为True, 所有阻塞池的线程激活进入就绪状态, 等待操作系统调度
6
7 event.clear() 恢复event的状态值为False
```



可以考虑一种应用场景，例如，我们有多少个线程从Redis队列中读取数据来处理，这些线程都要尝试去连接Redis的服务，一般情况下，如果Redis连接不成功，在各个线程的代码中，都会去尝试重新连接。如果我们想要再启动是确保Redis服务正常，才让那些工作线程去连接Redis服务器，那么我们就可以采用threading.Event机制来协调各个工作线程的连接操作：主线程中回去尝试连接Redis服务，如果正常的话，触发事件，各工作线程会尝试连接Redis服务。

实例如下：



```
1 import threading
2 import time
3 import logging
4
5 logging.basicConfig(level=logging.DEBUG, format='%(threadName)-10s) %(message)s',)
6
7 def worker(event):
8     logging.debug('Waiting for redis ready...')
9     event.wait()
10    logging.debug('redis ready, and connect to redis server and do some work [%s]',
11                  time.ctime())
12    time.sleep(1)
13
14 def main():
15     readis_ready = threading.Event()
16     t1 = threading.Thread(target=worker, args=(readis_ready,), name='t1')
17     t1.start()
18
19     t2 = threading.Thread(target=worker, args=(readis_ready,), name='t2')
20     t2.start()
21
22    logging.debug('first of all, check redis server, make sure it is OK, and then trigger
```



```

the redis ready event')
22     time.sleep(3)
23     readis_ready.set()
24
25 if __name__=="__main__":
26     main()
27 运行结果
28 (t1          ) Waiting for redis ready...
29 (t2          ) Waiting for redis ready...
30 (MainThread) first of all, check redis server, make sure it is OK, and then trigger the
redis ready event
31 (t1          ) redis ready, and connect to redis server and do some work [Tue May 9
19:10:09 2017]
32 (t2          ) redis ready, and connect to redis server and do some work [Tue May 9
19:10:09 2017]
33
34 Process finished with exit code 0

```



threading.Event的wait方法还接受一个超时参数，默认情况下如果事情一致没有发生，wait方法会一直阻塞下去，而加入这个超时参数之后，如果阻塞时间超过这个参数设定的值之后，wait方法会返回。对应于上面的应用场景，如果Redis服务器一致没有启动，我们希望子线程能够打印一些日志来不断地提醒我们当前没有一个可以连接的Redis服务，我们就可以通过设置这个超时参数来表达成这样的目的：

semaphore(信号量)

Semaphore管理一个内置的计算器

0 当调用acquire () 时内置计数器-1

用release () 时内置计算器+1

计算器不能小于0，当计数器为0时，acquire () 将阻塞线程直到其他线程调用release ()



实例：（同时只有5个线程可以获得semaphore，即可以限制最大连接数为5）



```
1 import threading
2 import time
3
4 semaphore=threading.Semaphore(5)  #最大一次性进行次数
5
6 def func():
7     semaphore.acquire()
8     print(threading.currentThread().getName()+ 'grt semaphore')
9     time.sleep(2)
10    semaphore.release()
11 for i in range(20):
12     t1=threading.Thread(target=func)
13     t1.start()
14
15 运行结果
16 Thread-1grt semaphore
17 Thread-2grt semaphore
18 Thread-3grt semaphore
19 Thread-4grt semaphore
20 Thread-5grt semaphore
21 Thread-6grt semaphore
22 Thread-7grt semaphore
23 Thread-8grt semaphore
24 Thread-9grt semaphore
25 Thread-10grt semaphore
0  ; Thread-12grt semaphore
   ' Thread-13grt semaphore
   ; Thread-14grt semaphore
29 Thread-15grt semaphore
```



```
30 Thread-11grt semaphore
31 Thread-17grt semaphore
32 Thread-18grt semaphore
33 Thread-19grt semaphore
34 Thread-20grt semaphore
35 Thread-16grt semaphore
36
37 Process finished with exit code 0
```



multiprocessing模块:

multiprocessing包是Python中多进程管包。与threading.Thread类似，他可以利用multiprocessing.Process对象来创建一个进程。该进程可以运行在python程序内部编写的函数。该Process对象与Thread的用法相同，也有start() run() join()的方法。此外multiprocessing包中也有Lock/Event/Semaphore/Condition类（这些对象可以像多线程那样，通过参数传递给各个进程），用以同步进程，器用法与threading包中的同名类一致。所以，

multiprocessing的很大一部分与threading使用同一套API（接口），只不过换到了多进程的情境。

python的进程调用

方法一：



```
1 ##Process类调用
2 from multiprocessing import Process
3 import time
4 def f(name):
5
6     print('hello',name,time.ctime())
7     time.sleep(1)
8
9 if __name__ == '__main__':
```



```
10 l=[]
11 for i in range(3):
12     p=Process(target=('alvin:%s'%i))
13     l.append(p)
14     p.start()
15 for i in l:
16     i.join()
17 print('ending')
```



方法二：



```
1 ##继承Process类调用
2 from multiprocessing import Process
3 import time
4 class MyProcess(Process):
5     def __init__(self):
6         super(MyProcess, self).__init__()
7
8     def run(self):
9         print('hello',self.name,time.ctime())
10        time.sleep(1)
11
12 if __name__ == '__main__':
13     l=[]
14     for i in range(3):
15         p=MyProcess()
16         p.start()
17         l.append(p)
18     for i in l:
19         i.join()
```



```
20  
21     print('engding')
```



process类

构造方法：

Process([group [, target [, name [, args [, kwargs]]]])

group:线程组，目前还没有实现，库引用中提示必须是None

target：要执行的方法

name：进程名

args/kwarges：要传入方法的参数。

实例方法：

is_alive（）返回进程是否在运行

join（[timeout]）阻塞当期那上下文环境的进程，直到调用此方法的进程终止或到达指定的timeout（可选参数）

start（）进程准备就绪，等待CPU调度

run（）stat（）调用run方法，如果实例进程时未制定传入target，这star执行t默认run（）方法

terminate（）不管任务是否完成，立即停止工作进程


属性：

daemon 和线程的setDeanon功能一样

name 进程名字

0 | 进程号





```
1 from multiprocessing import Process
2 import os
3 import time
4 def info(name):
5
6
7     print("name:", name)
8     print('parent process:', os.getppid())
9     print('process id:', os.getpid())
10    print("-----")
11    time.sleep(1)
12
13 def foo(name):
14
15     info(name)
16
17 if __name__ == '__main__':
18
19     info('main process line')
20
21
22     p1 = Process(target=info, args=('alvin',))
23     p2 = Process(target=foo, args=('egon',))
24     p1.start()
25     p2.start()
26
27     p1.join()
0  ;   p2.join()
    ;
30     print("ending")
```





通过tasklist (Win) 或者ps-elf|grep (linux) 命令检测每一个进程号 (PID) 对应的进程名

协程 :

yield与协程



```
import time
def consumer():
    r=''
    while True:
        n=yield r
        if not n:
            return
        print('[CONSUMER]---Consuming %s...'%n)
        time.sleep(1)
        r='200 OK'

def prduce(c):
    next(c)
    n=0
    while n<5:
        n+=1
        print('[CONSUMER]---Consuming %s...' % n)
        cr=c.send(n)
        print('[CONSUMER]---Consuming %s...'%n)
    c.close()
if __name__ == '__main__':
    c=consumer()
    prduce(c)
```

0

运行结果



```
[CONSUMER]---Consuming 1...
[CONSUMER]---Consuming 1...
[CONSUMER]---Consuming 1...
[CONSUMER]---Consuming 2...
[CONSUMER]---Consuming 2...
[CONSUMER]---Consuming 2...
[CONSUMER]---Consuming 3...
[CONSUMER]---Consuming 3...
[CONSUMER]---Consuming 3...
[CONSUMER]---Consuming 4...
[CONSUMER]---Consuming 4...
[CONSUMER]---Consuming 4...
[CONSUMER]---Consuming 5...
[CONSUMER]---Consuming 5...
[CONSUMER]---Consuming 5...
```

Process finished with exit code 0



分类: Python

好文要顶

关注我

收藏该文



0



Meet_

关注 - 27

粉丝 - 15

[+加关注](#)



« 上一篇: [Python/进程和线程](#)

» 下一篇: [Python/IO模型](#)

posted @ 2017-05-10 08:29 Meet、 阅读(679) 评论(0) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库

【促销】腾讯云技术升级10大核心产品年终让利

【推荐】高性能云服务器2折起，0.73元/日节省80%运维成本

【新闻】H3 BPM体验平台全面上线



最新IT新闻:

- 比特币这么贵的原因可能是机器人，你会成为接盘侠吗
 - 这可能是微软秘密研发的Surface Notebook
 - 还花一两个小时挑房源？Airbnb将推新技术让你全方位无死角了解房源
 - 方舟子：贾跃亭以老赖第一人身份登上纽约时报
 - JS开发者：最喜欢React，Vue.js比Angular值得尝试
- » 更多新闻...



0



最新知识库文章:

- 以操作系统的角度述说线程与进程
- 软件测试转型之路
- 门内门外看招聘
- 大道至简，职场上做人做事做管理
- 关于编程，你的练习是不是有效的？
- » 更多知识库文章...

Copyright ©2017 Meet、
