**二、进程**

multiprocessing模块

进程调用

[复制代码](javascript:void(0);)

#!/usr/bin/env python

# -\*- coding:utf-8 -\*-

from multiprocessing import Process

import time

def start(name):

time.sleep(1)

print('hello', name)

if \_\_name\_\_ == '\_\_main\_\_':

p = Process(target=start, args=('zhangsan',))

p1 = Process(target=start, args=('lisi',))

p.start()

p1.start()

p.join()

[复制代码](javascript:void(0);)

进程间通讯

每个进程都拥有自己的内存空间，因此不同进程间内存是不共享的，要想实现两个进程间的数据交换，有几种方法

Queue（队列）

[复制代码](javascript:void(0);)

#!/usr/bin/env python

# -\*- coding:utf-8 -\*-

from multiprocessing import Process, Queue

def start(q):

q.put( 'hello')

if \_\_name\_\_ == '\_\_main\_\_':

q = Queue()

p = Process(target=start, args=(q,))

p.start()

print(q.get())

p.join()

[复制代码](javascript:void(0);)

Pipe（管道，不常用）

　　把管道的两头分别赋给两个进程，实现两个进程的互相通信

[复制代码](javascript:void(0);)

#!/usr/bin/env python

# -\*- coding:utf-8 -\*-

from multiprocessing import Process, Pipe

def start(conn):

conn.send('hello')#发送

print(conn.recv())#接收

conn.close()

if \_\_name\_\_ == '\_\_main\_\_':

parent\_conn, child\_conn = Pipe() #生成一个管道

p = Process(target=start, args=(child\_conn,))

p.start()

print(parent\_conn.recv())#接收

parent\_conn.send("11111")#发送

p.join()

[复制代码](javascript:void(0);)

Manager（实现了进程间真正的数据共享）

[复制代码](javascript:void(0);)

#!/usr/bin/env python

from multiprocessing import Process, Manager

def f(dic, list,i):

dic['1'] = 1

dic['2'] = 2

dic['3'] = 3

list.append(i)

if \_\_name\_\_ == '\_\_main\_\_':

manager = Manager()

dic = manager.dict()#通过manager生成一个字典

list = manager.list(range(5))#通过manager生成一个列表

p\_list = []

for i in range(10):

p = Process(target=f, args=(dic, list,i))

p.start()

p\_list.append(p)

for res in p\_list:

res.join()

print(dic)

print(list)

#执行结果

'''

{'2': 2, '3': 3, '1': 1}

[0, 1, 2, 3, 4, 1, 9, 2, 5, 3, 7, 6, 0, 8, 4]

'''

[复制代码](javascript:void(0);)

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：

1、apply（同步）

2、apply\_async（异步）

from multiprocessing import Process,Pool

import time

def Foo(i):

time.sleep(1)

return i+100

def Bar(arg):

print('number::',arg)

if \_\_name\_\_ == "\_\_main\_\_":

pool = Pool(3)#定义一个进程池，里面有3个进程

for i in range(10):

pool.apply\_async(func=Foo, args=(i,),callback=Bar)

#pool.apply(func=Foo, args=(i,))

pool.close()#关闭进程池

pool.join()#进程池中进程执行完毕后再关闭,(必须先close在join)

[复制代码](javascript:void(0);)

callback是回调函数，就是在执行完Foo方法后会自动执行Bar函数，并且自动把Foo函数的返回值作为参数传入Bar函数

**三、协程**

　　协程，又称微线程，是一种用户态的轻量级线程。协程能保留上一次调用时的状态，每次过程重入时，就相当于进入上一次调用的状态，换种说法：进入上一次离开时所处逻辑流的位置，当程序中存在大量不需要CPU的操作时（IO），适用于协程。

协程有极高的执行效率，因为子程序切换不是线程切换，而是由程序自身控制，因此，没有线程切换的开销。

不需要多线程的锁机制，因为只有一个线程，也不存在同时写变量冲突，在协程中控制共享资源不加锁，只需要判断状态就好了，所以执行效率比多线程高很多。

因为协程是一个线程执行，所以想要利用多核CPU，最简单的方法是多进程+协程，这样既充分利用多核，又充分发挥协程的高效率。

那符合什么条件就能称之为协程：1、必须在只有一个单线程里实现并发 2、修改共享数据不需加锁 3、用户程序里自己保存多个控制流的上下文栈 4、一个协程遇到IO操作自动切换到其它协程

python中对于协程有两个模块，greenlet和gevent。

Greenlet（greenlet的执行顺序需要我们手动控制）

[复制代码](javascript:void(0);)

#!/usr/bin/env python

# -\*- coding:utf-8 -\*-

from greenlet import greenlet

def test1():

print (11)

gr2.switch() #手动切换

print (22)

gr2.switch()

def test2():

print (33)

gr1.switch()

print (44)

gr1 = greenlet(test1)

gr2 = greenlet(test2)

gr1.switch()

[复制代码](javascript:void(0);)

gevent（自动切换，由于切换是在IO操作时自动完成，所以gevent需要修改Python自带的一些标准库，这一过程在启动时通过monkey patch完成）

[复制代码](javascript:void(0);)

from gevent import monkey; monkey.patch\_all()

import gevent

import time

def foo():

print('11')

time.sleep(3)

print('22')

def bar():

print('33')

print('44')

gevent.joinall([

gevent.spawn(foo),

gevent.spawn(bar),

])

[复制代码](javascript:void(0);)

运行结果：（从结果可以看出，它们是并发执行的）

11

33

44

22