

Py西游攻关之多进程(multiprocessing模块)

多进程

一 多进程的概念

`multiprocessing` is a package that supports spawning processes using an API similar to the `threading` module. The `multiprocessing` package offers both local and remote concurrency, effectively side-stepping the `Global Interpreter Lock` by using subprocesses instead of threads. Due to this, the `multiprocessing` module allows the programmer to fully leverage multiple processors on a given machine. It runs on both Unix and Windows.

由于GIL的存在，python中的多线程其实并不是真正的多线程，如果想要充分地使用多核CPU的资源，在python中大部分情况需要使用多进程。Python提供了非常好用的多进程包multiprocessing，只需要定义一个函数，Python会完成其他所有事情。借助这个包，可以轻松完成从单进程到**并发执行**的转换。multiprocessing支持子进程、通信和共享数据、执行不同形式的同步，提供了Process、Queue、Pipe、Lock等组件。

multiprocessing包是Python中的多进程管理包。与threading.Thread类似，它可以利用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对象的使用方法相同，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，其用法与threading包中的同名类一致。所以，multiprocessing的很大一部份与threading使用同一套API，只不过换到了多进程的情境。

但在使用这些共享API的时候，我们要注意以下几点：

- 在UNIX平台上，当某个进程终结之后，该进程需要被其父进程调用wait，否则进程成为僵尸进程(Zombie)。所以，有必要对每个Process对象调用join()方法（实际上等同于wait）。对于多线程来说，由于只有一个进程，所以不存在此必要性。
- multiprocessing提供了threading包中没有的IPC(比如Pipe和Queue)，效率上更高。应优先考虑Pipe和Queue，避免使用Lock/Event/Semaphore/Condition等同步方式（因为它们占据的不是用户进程的资源）。
- 多进程应该避免共享资源。在多线程中，我们可以比较容易地共享资源，比如使用全局变量或者传递参数。在多进程情况下，由于每个进程有自己独立的内存空间，以上方法并不合适。此时我们可以通过共享内存和Manager的方法来共享资源。但这样做提高了程序的复杂度，并因为同步的需要而降低了程序的效率。

Process.PID中保存有PID，如果进程还没有start()，则PID为None。

window系统下，需要注意的是要想启动一个子进程，必须加上那句if __name__ == "main"，进程相关的要写在这句下面。

实例：

```
from multiprocessing import Process
import time
def f(name):
    time.sleep(1)
    print('hello', name, time.ctime())

if __name__ == '__main__':
    p_list=[]
    for i in range(3):
        p = Process(target=f, args=('alvin',))
        p_list.append(p)
        p.start()
    for i in p_list:
        p.join()
    print('end')
```

类式调用

```

from multiprocessing import Process
import time

class MyProcess(Process):
    def __init__(self):
        super(MyProcess, self).__init__()
        #self.name = name

    def run(self):
        time.sleep(1)
        print ('hello', self.name,time.ctime())

if __name__ == '__main__':
    p_list=[]
    for i in range(3):
        p = MyProcess()
        p.start()
        p_list.append(p)

    for p in p_list:
        p.join()

    print('end')

```

To show the individual process IDs involved, here is an expanded example:

```

from multiprocessing import Process
import os
import time

def info(title):
    print(title)
    print('module name:', __name__)
    print('parent process:', os.getppid())
    print('process id:', os.getpid())

def f(name):
    info('\033[31;1mfunction f\033[0m')
    print('hello', name)

if __name__ == '__main__':
    info('\033[32;1mmain process line\033[0m')
    time.sleep(100)
    p = Process(target=info, args=('bob',))
    p.start()
    p.join()

```

二 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/kwars: 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]): 阻塞当前上下文环境的进程程，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(): start()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:

authkey

daemon: 和线程的setDaemon功能一样

exitcode(进程在运行时为None、如果为-N, 表示被信号N结束)

name: 进程名字。

pid: 进程号。

```
import time
from multiprocessing import Process

def foo(i):
    time.sleep(1)
    print (p.is_alive(), i, p.pid)
    time.sleep(1)

if __name__ == '__main__':
    p_list=[]
    for i in range(10):
        p = Process(target=foo, args=(i,))
        #p.daemon=True
        p_list.append(p)

    for p in p_list:
        p.start()
    # for p in p_list:
    #     p.join()

    print('main process end')
```

二 进程间通讯

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, 可以用以下方法:

Queues

使用方法跟threading里的queue类似:

```
from multiprocessing import Process, Queue

def f(q,n):
    q.put([42, n, 'hello'])

if __name__ == '__main__':
    q = Queue()
    p_list=[]
    for i in range(3):
        p = Process(target=f, args=(q,i))
        p_list.append(p)
        p.start()
    print(q.get())
    print(q.get())
    print(q.get())
    for i in p_list:
        i.join()
```

Pipes

The `Pipe()` function returns a pair of connection objects connected by a pipe which by default is duplex (two-way). For example:

```
from multiprocessing import Process, Pipe

def f(conn):
    conn.send([42, None, 'hello'])
    conn.close()

if __name__ == '__main__':
    parent_conn, child_conn = Pipe()
    p = Process(target=f, args=(child_conn,))
    p.start()
    print(parent_conn.recv())  # prints "[42, None, 'hello']"
    p.join()
```

The two connection objects returned by `Pipe()` represent the two ends of the pipe. Each connection object has `send()` and `recv()` methods (among others). Note that data in a pipe may become corrupted if two processes (or threads) try to read from or write to the *same* end of the pipe at the same time. Of course there is no risk of corruption from processes using different ends of the pipe at the same time.

Managers

A manager object returned by `Manager()` controls a server process which holds Python objects and allows other processes to manipulate them using proxies.

A manager returned by `Manager()` will support types `list`, `dict`, `Namespace`, `Lock`, `RLock`, `Semaphore`, `BoundedSemaphore`, `Condition`, `Event`, `Barrier`, `Queue`, `Value` and `Array`. For example,

```
from multiprocessing import Process, Manager

def f(d, l, n):
    d[n] = '1'
    d['2'] = 2
    d[0.25] = None
    l.append(n)
    print(l)

if __name__ == '__main__':
    with Manager() as manager:
        d = manager.dict()

        l = manager.list(range(5))
        p_list = []
        for i in range(10):
            p = Process(target=f, args=(d, l, i))
            p.start()
            p_list.append(p)
        for res in p_list:
            res.join()

        print(d)
        print(l)
```

三 进程同步

Without using the lock output from the different processes is liable to get all mixed up.

```
from multiprocessing import Process, Lock

def f(l, i):
    l.acquire()
    try:
```

```
print('hello world', i)
finally:
    l.release()

if __name__ == '__main__':
    lock = Lock()

    for num in range(10):
        Process(target=f, args=(lock, num)).start()
```



四 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

```
from multiprocessing import Process, Pool
import time

def Foo(i):
    time.sleep(2)
    return i+100

def Bar(arg):
    print('-->exec done:',arg)

pool = Pool(5)

for i in range(10):
    pool.apply_async(func=Foo, args=(i,), callback=Bar)
    #pool.apply(func=Foo, args=(i,))

print('end')
pool.close()
pool.join()
```



协程

[好文要顶](#)[已关注](#)[收藏该文](#)[Yuan先生](#)[关注 - 1](#)[粉丝 - 3951](#)[我在关注他](#) [取消关注](#)

4

1

posted @ 2016-08-07 12:50 Yuan先生 阅读(3512) 评论(0) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)

【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】开放下载 | 多场景多实战《阿里云AIoT造物秘籍》，值得收藏！

相关博文：

- Py西游攻关之Socket网络编程
 - Python学习之多进程并发模块(multiprocessing)
 - Py西游攻关之RabbitMQ、Memcache、Redis
 - python之多进程multiprocessing模块
 - Py西游攻关之Socket网络编程
- » 更多推荐...

最新 IT 新闻：

- 黎巴嫩首都爆炸能量有多大？物理学家看视频计算：300吨TNT！
 - 马斯克到底从特斯拉赚了多少钱？他是最富的穷光蛋
 - 特朗普封禁微信，张小龙至少有两张牌可打
 - “抖音点赞员”月入1万+，靠谱吗？
 - 减持500亿元、死磕马斯克，贝索斯在下一盘大棋？
- » 更多新闻...

Copyright © 2020 Yuan先生
Powered by .NET Core on Kubernetes