python多进程multiprocessing使用

1. 为什么需要multiprocessing

如果你想在python中使用线程来实现并发以提高效率，大多数情况下你得到的结果是比串行执行的效率还要慢；这主要是python中GIL（全局解释锁）的缘故，通常情况下线程比较适合高IO低CPU的任务，否则创建线程的耗时可能比串行的还要多。GIL是历史问题，和C解释器有关系。

为了解决这个问题，python中提供了多进程的方式来处理需要并发的任务，可以有效的利用多核cpu达到并行的目的。

但是需要注意的是，进程之间的资源是相互独立的，特别是内存；如果你需要在进程之间交换数据，共享信息，你需要有别于单进程的方式来创建你的代码。

好在`multiprocessing`包针对这些问题都提供了良好的解决方案。接下来就让我们一睹真容。

2. multiprocessing使用

2.1 Process

```
from multiprocessing import Process
import os

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

def f(name):
    info('function f')
    print('hello', name)


info('main line')
p_list = []
for i in range(5):
    p = Process(target=f, args=('the {}-th bob'.format(i),))
    p_list.append(p)
    p.start()

for i in range(5):
    p_list.get(i).join
```

2.2 进程池Pool

进程占用的是系统的资源，一般不会创建很多，否则资源不足的进程还是要等待获取资源。比如1000个任务，不会移动1000个进程，一是进程启动需要耗时，一是进程之间抢占资源。multiprocessing提供了进程池的方式，共享进程的使用。

```
from multiprocessing import Pool
import time

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

def end_call(arg):
    print("end_call",arg)

p = Pool(5)
for i in range(10):
    p.apply_async(func=add_by_input, args=(i,), callback=end_call)

print("end")
p.close()
p.join()
```

3. multiprocessing通信

multiprocess.Queue是跨进程通信队列。但是不能用于multiprocessing.Pool多进程的通信。

multiprocessing.Manager 进程池multiprocessing.Pool()的多进程之间的通信要用multiprocessing.Manager().Queue()

一个multiprocessing.Manager对象会控制一个服务器进程，其他进程可以通过代理的方式来访问这个服务器进程。从而达到多进程间数据通信且安全。

Manager支持的类型有list,dict,Namespace,Lock,RLock,Semaphore,BoundedSemaphore,Condition,Event,Queue,Value和Array。

3.1 进程间同步锁

进程之间如果存在资源互斥的情况，可以通过Lock进行加锁。`acquire`获得独占锁，`release`释放锁。

```
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()
```

3.2 共享数据

多进程之间最好避免数据共享，进程之间要共享数据可以通过Manager来启动一个服务进程，来协调进程之间数据的共享。

Manager() 返回的管理器支持类型： list 、dict 、Namespace 、Lock 、RLock 、Semaphore 、BoundedSemaphore 、Condition 、Event 、Barrier 、Queue 、Value 和 Array

```
from multiprocessing import Process, Manager

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

if __name__ == '__main__':
    with Manager() as manager:
        d = manager.dict()
        l = manager.list(range(10))

        p = Process(target=f, args=(d, l))
        p.start()
        p.join()

        print(d)
        print(l)
# {0.25: None, 1: '1', '2': 2}
# [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

4. 总结

本文分享了multiprocessing来进行python多进程操作，希望对你有帮助。总结如下：

- 利用好multiprocessing好Pool，共享资源池
- 进程间共享锁用Lock
- 进程间数据共享，通过Manager做中间进程服务，速度慢点