multiprocessing

--p = Process(target=func,args=(,...))

类似于threading中Thread类

--p.start()

--p.join()

join()方法可以等待子进程结束后再继续往下运行，通常用于进程间的同步。

--p = Pool(processes=4)

process的大小最好取多核的N倍

--apply_async(target=func,args=(,..))

向进程池里添加进程

--p.get()

返回函数return后的值

--p.close()

对Pool对象调用join()方法会等待所有子进程执行完毕，调用join()之前必须先调用close()，调用close()之后就不能继续添加新的Process了。

FE：

```
if __name__==’__main__’:
    p = Pool(4)
    for i in range(5);
        p.apply_async(long_time_task, args=(i,))
    print('Waiting for all subprocesses done...')
    p.close()
    p.join()
    print('All subprocesses done.')
```

FE：

```
import multiprocessing
import time
def func(msg):
  for i in xrange(3):
    print msg
    time.sleep(1)
  return "done " + msg
if __name__ == "__main__":
  pool = multiprocessing.Pool(processes=4)
  result = []
  for i in xrange(10):
    msg = "hello %d" %(i)
    result.append(pool.apply_async(func, (msg, )))
  pool.close()
  pool.join()
  for res in result:
    print res.get()
  print "Sub-process(es) done."
```

--q = Queue()

Queue()用于进程之间的信息交流

FE：

```
from multiprocessing import Process, Queue
import os, time, random

# 写数据进程执行的代码:
def write(q):
    print('Process to write: %s' % os.getpid())
    for value in ['A', 'B', 'C']:
        print('Put %s to queue...' % value)
        q.put(value)
        time.sleep(random.random())
```

```
# 读数据进程执行的代码:
def read(q):
    print('Process to read: %s' % os.getpid())
    while True:
        value = q.get(True)
        print('Get %s from queue.' % value)

if __name__=='__main__':
    # 父进程创建Queue，并传给各个子进程：
    q = Queue()
    pw = Process(target=write, args=(q,))
    pr = Process(target=read, args=(q,))
    # 启动子进程pw，写入:
    pw.start()
    # 启动子进程pr，读取:
    pr.start()
    # 等待pw结束:
    pw.join()
    # pr进程里是死循环，无法等待其结束，只能强行终止:
    pr.terminate()
```