创建进程

```
'''
Process类的构造函数参数说明:
Target:表示调用对象,一般为函数,也可以是类
Args:表示调用对象的位置参数元组
Kwargs:表示调用对象的字典
Name:为进程的别名
Group:参数不使用,可忽略

Process类常用方法:
is_alive():返回进程是否是激活的
join([timeout]):阻塞进程,直到进程执行完成或超时或进程被终止
run():代表进程执行的任务函数,可被重写
start():激活进程
terminate():终止进程

Process的属性:
daemon:父进程终止后自动终止，且不能产生新进程，必须在start(之前设置
authkey:字节码，进程的准密钥。
exitcode:退出码，进程在运行时为None，如果为-N，就表示被信号N结束。
name:获取进程名称。
pid:进程id。
'''

Daemon属性：
daemon:父进程终止后自动终止，且不能产生新进程，必须在start()之前设置

父进程就是运行代码的哪个进程

没有设置daemon和有设置daemon的区别
1.不设置daemon
#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====

from multiprocessing import Process
import os
import time

#子进程要执行的任务函数
def task_func(d):
    print(f'{time.strftime("%Y-%m-%d %H:%M:%S")}子进程开始执行')
    print(f"sleep {d} s")
    time.sleep(d)
    print(f'{time.strftime("%Y-%m-%d %H:%M:%S")}子进程结束执行')
```

```
49
50
51  if __name__=='__main__':
52      print(f'{time.strftime("%Y-%m-%d %H:%M:%S")}父进程开始执行')
53      p=Process(target=task_func,args=(3,))
54      p.start()
55      print(f'{time.strftime("%Y-%m-%d %H:%M:%S")}父进程结束执行')
56  父进程没有等待子进程,父进程代码直接运行到尾部,但程序还在等子进程结束
57
58  2.设置daemon：
59  #!/usr/bin/env python
60  # -*- coding:utf-8 -*-
61  #====#====#====#====
62  #Author:
63  #CreatDate:
64  #Version:
65  #====#====#====#====
66
67  from multiprocessing import Process
68  import os
69  import time
70
71  #子进程要执行的任务函数
72  def task_func(d):
73      print(f'{time.strftime("%Y-%m-%d %H:%M:%S")}子进程开始执行')
74      print(f"sleep {d} s")
75      time.sleep(d)
76      print(f'{time.strftime("%Y-%m-%d %H:%M:%S")}子进程结束执行')
77
78
79  if __name__=='__main__':
80      print(f'{time.strftime("%Y-%m-%d %H:%M:%S")}父进程开始执行')
81      p=Process(target=task_func,args=(3,))
82      p.daemon=True#让程序不等待子进程,直接结束
83      p.start()
84      print(f'{time.strftime("%Y-%m-%d %H:%M:%S")}父进程结束执行')
85  如果设置了daemon为True,那么父进程执行完代码,程序直接结束
```

进程并发控制

```
1  Semaphore是控制同一时刻并发的进程数
2
3  有时候如果很多进程都去访问共享资源，可能导致资源压力过大，比如100个进程都去访问数据库，那么数据库压
   力会很大，这个时候就可以使用控制进程，让一些进程去访问，一些进程后面再访问
4
5  #!/usr/bin/env python
6  # -*- coding:utf-8 -*-
7  #====#====#====#====
8  #Author:
9  #CreatDate:
```

```
#Version:
#====#====#====#====
from multiprocessing import Process,Semaphore,current_process
import time

def mytest(se,i):
    se.acquire()#获取许可,可以使用公共资源,其他进程不能使用
    print(time.strftime("%H:%M:%S"),current_process().name+"开始运行")
    time.sleep(i)
    print(time.strftime("%H:%M:%S"),current_process().name+"结束运行")
    se.release()#放弃许可,其他进程可以使用

if __name__=="__main__":
    #创建控制进程的对象
    se=Semaphore(2)
    #循环创建子进程
    for i in range(6):
        p=Process(target=mytest,args=(se,2))
        p.start()

```

进程同步-Lock

如果有多个进程同时运行，都去访问资源，那么可能导致混乱

这时需要使用锁(Lock)来控制同一时刻仅有一个进程在访问资源

没有Lock时:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====
from multiprocessing import Process
import time

def task1():
    n=5
    while n>1:
        print(time.strftime("%H:%M:%S")+" task1 输出信息")
        time.sleep(1)
        n-=1


def task2():
    n=5
    while n>1:
        print(time.strftime("%H:%M:%S")+" task2 输出信息")
        time.sleep(1)
        n-=1
```

```
30
31  def task3():
32      n=5
33      while n>1:
34          print(time.strftime("%H:%M:%S")+" task3 输出信息")
35          time.sleep(1)
36          n-=1
37
38
39  if __name__=="__main__":
40      p1=Process(target=task1)
41      p2=Process(target=task2)
42      p3=Process(target=task3)
43
44      p1.start()
45      p2.start()
46      p3.start()
47
48  #同一时刻,有3个进程在运行,打印的或实现的信息会混乱
49
50
51  有Lock的时候
52  #!/usr/bin/env python
53  # -*- coding:utf-8 -*-
54  #====#====#====#====
55  #Author:
56  #CreatDate:
57  #Version:
58  #====#====#====#====
59  from multiprocessing import Process,Lock
60  import time
61
62  def task1(lock):
63      #加锁
64      lock.acquire()
65      n=5
66      while n>1:
67          print(time.strftime("%H:%M:%S")+" task1 输出信息")
68          time.sleep(1)
69          n-=1
70      #解锁
71      lock.release()
72
73  def task2(lock):
74      with lock:
75          n=5
76          while n>1:
77              print(time.strftime("%H:%M:%S")+" task2 输出信息")
78              time.sleep(1)
79              n-=1
80
81  def task3(lock):
82      with lock:
```

```
        n=5
        while n>1:
            print(time.strftime("%H:%M:%S")+" task3 输出信息")
            time.sleep(1)
            n-=1


if __name__=="__main__":
    #创建Lock对象
    lock=Lock()
    p1=Process(target=task1,args=(lock,))
    p2=Process(target=task2,args=(lock,))
    p3=Process(target=task3,args=(lock,))

    p1.start()
    p2.start()
    p3.start()
```

进程之间通信-Event

```
需要使用Event来挂起进程或唤醒进程

1.  Event().wait()    插入在进程中插入一个标记（flag）  flag默认为 false  然后flag为false时,程序会停止运行  进入阻塞状态

2.  Event().set()     使flag为Ture  然后程序会停止运行 进入运行状态

3.  Event().clear()      使flag为false  然后程序会停止运行 进入阻塞状态

4.  Event().is_set()   判断flag  是否为True  是的话 返回True  不是 返回false

Set能唤醒进程，并让flag为True.
Clear能让flag为false
Wait在flag为false的时候可以挂起进程，flag默认是false

#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====

import multiprocessing
import time

def mytest1(e):
    #让子进程暂停,同时有一个flag
    e.wait()
```

```
    time.sleep(1)
    print("flag的值是:",e.is_set())
    e.clear()#让flag的值为False,是因为下面的wait要有效果的话,flag必须为False
    print(f"{time.strftime('%H:%M:%S')} 子进程1:我们是兄弟,我等你")
    e.wait()
    print(f"{time.strftime('%H:%M:%S')} 子进程1:好,我们一起走吧")


def mytest2(e):
    e.wait()
    time.sleep(1)
    print("flag的值是:",e.is_set())
    e.clear()
    print(f"{time.strftime('%H:%M:%S')} 子进程2:好吧,但我只等你5秒")
    e.wait(5)#表示只暂停5秒
    print(f"{time.strftime('%H:%M:%S')} 子进程2:时间到了,我继续走了")



if __name__=='__main__':
    #创建事件对象
    e=multiprocessing.Event()
    p1=multiprocessing.Process(target=mytest1,args=(e,))
    p2=multiprocessing.Process(target=mytest2,args=(e,))
    p1.start()
    p2.start()

    print(f"{time.strftime('%H:%M:%S')} 主进程:谁等我一下,我需要8秒时间")
    e.set()#唤醒进程
    time.sleep(8)
    print(f"{time.strftime('%H:%M:%S')} 主进程:好,感谢兄弟,我赶上")
    e.set()
    p1.join()
    p2.join()
    print(f"{time.strftime('%H:%M:%S')} 主进程:好,大家都登顶了")


```

进程优先级队列-Queue

```
Queue 是多进程安全的队列，可以使用 Queue实现多进程之间的数据传递。
put 方法用以插入数据到队列中，put 方法还有两个可选参数:blocked 和 timeout。如果blocked为 True(默认值)，并且 timeout 为正值，则该方法会阻塞timeout 指定的时间，直到该队列有剩余的空间。如果超时，则会抛出 Queue.Full 异常。如果blocked 为 False，但该 Queue 已满，则会立即抛出 Queue.Full 异常。

get 方法可以从队列读取并删除一个元素。同样，get方法有两个可选参数:blocked 和 timeout。如果 blocked 为 True 默认值)，并且 timeout 为正值，在等待时间内没有取到任何元素，则会抛出 Queue.Empty 天异常。如果 blocked 为False，那么将会有两种情况存在: Queue 有一个值可用，立即返回该值，否则队列为空，立即抛出 Queue.Empty异常。

```

下面的代码定义类生产者和消费者函数，设置其队列最大容量是5，生产者不停的生产冷饮，消费者不停的消费冷饮，当队列满时，生产者等待，当队列空时，消费者等待。他们的放入和取出速度可能不一致，但使用Queue可以让生产者和消费者有条不絮地一直处理下去

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====

from multiprocessing import Process,Queue
import time

#生产者
def mytest(q):
    n=1
    while True:
        q.put(f'冷饮{n}')
        print(f"{time.strftime('%H:%M:%S')}A进程 放入冷饮{n}")
        n+=1
        time.sleep(1)


#消费者
def mytest2(q):
    while True:
        print(f"{time.strftime('%H:%M:%S')}B进程 取出冷饮{q.get()}")
        time.sleep(5)

if __name__=='__main__':
    #定义队列,容量为5
    q=Queue(maxsize=5)
    p1=Process(target=mytest,args=(q,))
    p2=Process(target=mytest2,args=(q,))
    p1.start()
    p2.start()
    p1.join()
    p2.join()


```

进程池Pool

在使用 Python 进行系统管理的时候，特别是是同时操作多个文件目录，或者远程控制多台主机并行操作，可以节约大量的时间。当被操作对象数目不大时，可以直接利用 multiprocessing中的Process 动态生成多个进程，十几个还好，但但如果是上百个，上千个目标，手动限制进程数量又太过烦琐，此时就可以发挥进程池的功效了。

Pool 可以提供指定数量的进程供用户调用，当有新的请求提交到 pool 中时，如果池还没 有满，就会创建一个新的进程用于执行该请求； 如果池中的进程数量已经达到规定的最大值，该请求就会等待，直到池中有进程结束才会创建新的进程。

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====

import multiprocessing
import time

def mytest(name):
    print(f"{time.strftime('%H:%M:%S')}:{name} 开始执行 ")
    time.sleep(3)

if __name__=='__main__':
    #创建进程池,并设定有多少个进程
    pool=multiprocessing.Pool(processes=3)
    for i in range(10):
        pool.apply_async(func=mytest,args=(i,))

    #关闭进程池
    pool.close()

    pool.join()
```

数据交换Pipe

我们在类 Unix 系统中经常使用管道(Pipe) 命令来让一条命令的输出(STDOUT)作为另一条命令的输入(STDIN)获取最终的结果。在 Python 多进程编程中也有一个 Pipe 方法可以帮忙我们实现多进程之前的数据传输。我们可以将 Unix系统中的一个命令比作一个进程，一个进程的输出可以作为另一个进程的输入

multiprocessing.Pipe()方法返回一个管道的两个端口，如 Command1的STDOUT 和 Command2 的 STDIN，这样 Command1 的输出就作为 Command2的输入。如果反过来，让 Command2 的输出也可以作为 Command1的输入，这就是全双工管道，默认全双工管道。如果想设置半双工管道,只需要给Pipe()方法传递参数 duplex=False 就可以,即 Pipe(duplex=False)。

Pipe()方法返回的对象具有发送消息 send()方法和接收消息 recv()方法，可以调用 Command1.send(msg) 发送消息，调用 Command2.recv()接收消息。如果没有消息可接收，recv()方法会一直阻塞。如果管道已经被关闭，recv()方法就会抛出异常 EOFError。

```
#!/usr/bin/env python
```

```
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====
import multiprocessing
import time

def mytest(pp):
    for i in range(5):
        mystr=f"mytest-{i}"
        print(f"{time.strftime('%H:%M:%S')} mytest 发送{mystr}")
        pp.send(mystr)#发送

    time.sleep(2)
    for i in range(5):
        print(f"{time.strftime('%H:%M:%S')} mytest 接收:{pp.recv()}")

def mytestO2(pp):
    for i in range(5):
        print(f"{time.strftime('%H:%M:%S')} mytestO2 接收:{pp.recv()}")

    time.sleep(1)
    for i in range(5):
        mystr=f"mytestO2-{i}"
        print(f"{time.strftime('%H:%M:%S')} mytestO2 发送{mystr}")
        pp.send(mystr)#发送
if __name__=='__main__':
    #创建Pipe对象
    pi=multiprocessing.Pipe()
    #Pipe返回的是2个数据,一个是发送端,一个是接收端,但默认是双全工,所以发送和接收端都可以发送数据和接收数据
    p1=multiprocessing.Process(target=mytest,args=(pi[0],))
    p2=multiprocessing.Process(target=mytest,args=(pi[1],))

    p1.start()
    p2.start()
    p1.join()
    p2.join()



```

;