# 第16章 Python进程

本节所讲内容：

16.1 进程和线程的关系

16.2 进程的使用

16.3 进程中的变量

16.4 进程中的通信

16.5 进程池Pool

## 16.1 进程和线程的关系

### 16.1.1 什么是进程

通俗理解一个运行的程序或者软件,进程是操作系统资源分配的基本单位。

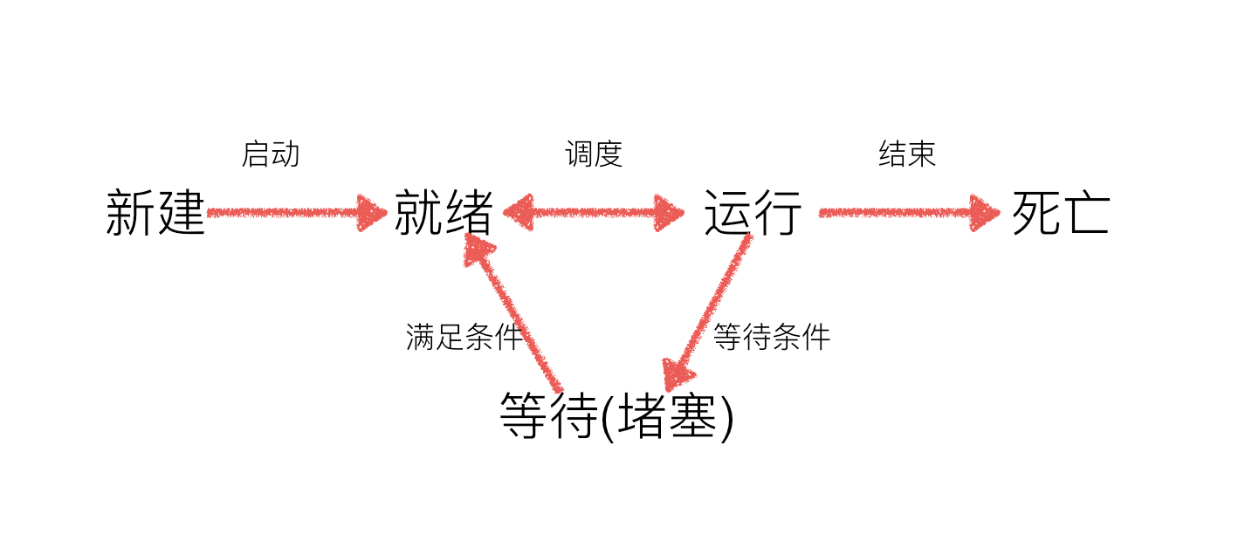


现实生活中的公司可以理解成是一个进程，公司提供办公资源(电脑、办公桌椅等)，真正干活的是员工，员工可以理解成线程。

注意：一个程序至少有一个进程,一个进程至少有一个线程,多进程可以完成多任务.

2.进程的状态

工作中，任务数往往大于cpu的核数，即一定有一些任务正在执行，而另外一些任务在等待cpu进行执行，因此导致了有了不同的状态。



就绪态：运行的条件都已经慢去，正在等在cpu执行；

执行态：cpu正在执行其功能；

等待态：等待某些条件满足，例如一个程序sleep了，此时就处于等待态；

小结：一个进程默认有一个线程，进程里面可以创建线程，线程是依附在进程里面的，没有进程就没有线程。

## 16.2 进程的使用

### 16.2.1 导入进程模块

#导入线程模块

import multiprocessing

Process进程类的语法结构如下：

Process([group [, target [, name [, args [, kwargs]]]]])

group：指定进程组，目前只能使用None

target：执行的目标任务名

name：进程名字

args：以元组方式给执行任务传参

kwargs：以字典方式给执行任务传参

Process创建的实例对象的常用方法：

start()：启动子进程实例（创建子进程）

join([timeout])：是否等待子进程执行结束，或等待多少秒

terminate()：不管任务是否完成，立即终止子进程

Process创建的实例对象的常用属性：

name：当前进程的别名，默认为Process-N，N为从1开始递增的整数

pid：当前进程的pid（进程号）

16.2.3 多进程完成多任务代码

import multiprocessing  
import time  
def run\_proc():  
 *'''子进程要执行的代码'''* while True:  
 print('---2---')  
 time.sleep(1)  
if \_\_name\_\_ == '\_\_main\_\_':  
 sub\_process = multiprocessing.Process(target=run\_proc)  
 sub\_process.start()  
 while True:  
 print('---1---')  
 time.sleep(2)

执行结果:

----1----

----2----

----1----

----2----

----1----

----2----

### 16.2.4 获取进程pid

import multiprocessing,time,os  
def work():  
 *#获取当前的进程* current\_process = multiprocessing.current\_process()  
 *#获取当前工作进程* print('目前工作的:',current\_process)  
 *#获取当前进程的编号* print('当前进程编号:',current\_process.pid,os.getpid())  
 *#获取父进程编号* print('父进程的编号:',os.getppid())  
 time.sleep(2)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 *#获取当前进程* current\_process = multiprocessing.current\_process()  
 *#得到目前的进程* print('main:',current\_process)  
 *#得到目前进程编号* print('main进程的编号:',current\_process.pid)  
 *#开启子进程* sub\_process = multiprocessing.Process(target=work)  
 sub\_process.start()  
 *# sub\_process.join()* print('主进程结束')

执行结果:

main: <\_MainProcess(MainProcess, started)>

main进程的编号: 7272

主线程结束

目前工作的: <Process(Process-1, started)>

当前进程编号: 7328 7328

父进程的编号: 72724. 给子进程指定的函数传递参数

带有参数的进程：

import multiprocessing  
*# 显示人员信息*def show\_info(name, age):  
 print(name, age)  
 current\_process = multiprocessing.current\_process()  
 print(current\_process.name)  
if \_\_name\_\_ == '\_\_main\_\_':  
 *# 创建子进程  
 # 1. group:进程组，目前必须使用None，一般不用设置  
 # 2. target:执行目标函数  
 # 3. name: 进程名称  
 # 4. args: 以元组方式给函数传参  
 # 5. kwargs: 以字典方式给函数传参*

*#利用元组的形式*

*#sub\_process=multiprocessing.Process(target=show\_info,name="myprocess",args=("for", 18))  
 # # 启动进程  
 # sub\_process.start()*

*#利用字典的形式* sub\_process = multiprocessing.Process(target=show\_info, name="myprocess",  
 kwargs={"name": "for", "age": 18})  
 *# 启动进程* sub\_process.start()

运行结果:

for 18

## 16.3 进程中的变量

学前提示：

1、进程之间不共享全局变量

2、主进程会等待所有的子进程执行完成程序再退出

16.3.1. 进程之间不共享全局变量

import multiprocessing  
import time  
*# 定义全局变量*my\_list = []  
*# 写入数据*def write\_data():  
 for i in range(5):  
 my\_list.append(i)  
 time.sleep(0.2)  
 print("write\_data:", my\_list)  
*# 读取数据*def read\_data():  
 print('read\_data:',my\_list)  
if \_\_name\_\_ == '\_\_main\_\_':  
 *# 创建写入数据的进程* write\_process = multiprocessing.Process(target=write\_data)  
 read\_process = multiprocessing.Process(target=read\_data)  
 write\_process.start()  
 *# 主进程等待写入进程执行完成以后代码 再继续往下执行* write\_process.join()  
 read\_process.start()

执行结果:

write\_data: [0, 1, 2, 3, 4]

read\_data: []

注意:创建子进程其实是对主进程进行拷贝，进程之间相互独立，访问的全局变量不是同一个，所以进程之间不共享全局变量

### 16.3.2 主进程会等待所有的子进程执行完成程序再退出

import multiprocessing  
import time  
  
*# 测试子进程是否执行完成以后主进程才能退出*def work():  
 for i in range(10):  
 print("工作中...")  
 time.sleep(0.2)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 *# 创建子进程* work\_process = multiprocessing.Process(target=work)  
 work\_process.start()  
 *#如果不加 work\_process.join()  
 # 让主进程等待1秒钟* time.sleep(1)  
 print("主进程执行完成了啦")

执行结果:

工作中...

工作中...

工作中...

工作中...

工作中...

主进程执行完成了啦

工作中...

工作中...

工作中...

工作中...

工作中...

1. 销毁子进程的代码

import multiprocessing  
import time  
*# 测试子进程是否执行完成以后主进程才能退出*def work():  
 for i in range(10):  
 print("工作中...")  
 time.sleep(0.2)  
if \_\_name\_\_ == '\_\_main\_\_':  
 *# 创建子进程* work\_process = multiprocessing.Process(target=work)  
 *# 1、设置守护主进程，主进程退出后子进程直接销毁，不再执行子进程中的代码  
 # work\_process.daemon = True* work\_process.start()  
 *# 让主进程等待1秒钟* time.sleep(1)  
 print("主进程执行完成了啦")  
 *#/2、让子进程直接销毁，表示终止执行， 主进程退出之前，把所有的子进程直接销毁就可以了* work\_process.terminate()

执行结果:

工作中...

工作中...

工作中...

工作中...

工作中...

主进程执行完成了啦

小结：

进程之间不共享全局变量

主进程会等待所有的子进程执行完成程序再退出

## 16.4 进程间通信-Queue

在这里我们要学习知道消息队列queue的放入值和获取值得操作

### 16.4.1. Queue的使用

可以使用multiprocessing模块的Queue实现多进程之间的数据传递，Queue本身是一个消息列队程序，首先用一个小实例来演示一下Queue的工作原理：

import multiprocessing  
import time  
if \_\_name\_\_ == '\_\_main\_\_':  
 *# 创建消息队列, 3:表示队列中最大消息个数* queue = multiprocessing.Queue(3)  
 *# 放入数据* queue.put(1)  
 queue.put("hello")  
 queue.put([3,5])  
 *# 总结: 队列可以放入任意数据类型  
 # 提示： 如果队列满了，需要等待队列有空闲位置才能放入数据，否则一直等待  
 # queue.put((5,6))  
 # 提示： 如果队列满了，不等待队列有空闲位置，如果放入不成功直接崩溃  
 # queue.put\_nowait((5,6))*

*# 建议： 向队列放入数据统一使用put  
 # 查看队列是否满了  
 # print(queue.full())  
 # 注意点：queue.empty()判断队列是否空了不可靠  
 # 查看队列是否空了  
 # print(queue.empty())  
 # 解决办法: 1. 加延时操作 2. 使用判断队列的个数,不使用empty  
 # time.sleep(0.01)* if queue.qsize() == 0:  
 print("队列为空")  
 else:  
 print("队列不为空")  
 *# 获取队列的个数* size = queue.qsize()  
 print(size)  
 *# 获取数据* value = queue.get()  
 print(value)  
 *# 获取队列的个数* size = queue.qsize()  
 print(size)  
 *# 获取数据* value = queue.get()  
 print(value)  
 *# 获取数据* value = queue.get()  
 print(value)  
 *# 获取队列的个数* size = queue.qsize()  
 print(size)  
 *# 提示：如果队列空了，再取值需要等待，只有队列有值以后才能获取队列中数据  
 # value = queue.get()  
 # print(value)  
 # 提示： 如果队列空了 ，不需要等待队列有值，但是如果取值的时候发现队列空了直接崩溃  
 # 建议大家: 向队列取值使用get  
 # value = queue.get\_nowait()  
 # print(value)*

运行结果:

队列不为空

3

1

2

hello

[3, 5]

0

说明

初始化Queue()对象时（例如：q=Queue()），若括号中没有指定最大可接收的消息数量，或数量为负值，那么就代表可接受的消息数量没有上限（直到内存的尽头）；

Queue.qsize()：返回当前队列包含的消息数量；

Queue.empty()：如果队列为空，返回True，反之False , 注意这个操作是不可靠的。

Queue.full()：如果队列满了，返回True,反之False；

Queue.get([block[, timeout]])：获取队列中的一条消息，然后将其从列队中移除，block默认值为True；

1）如果block使用默认值，且没有设置timeout（单位秒），消息列队如果为空，此时程序将被阻塞（停在读取状态），直到从消息列队读到消息为止，如果设置了timeout，则会等待timeout秒，若还没读取到任何消息，则抛出"Queue.Empty"异常；

2）如果block值为False，消息列队如果为空，则会立刻抛出"Queue.Empty"异常；

Queue.get\_nowait()：相当Queue.get(False)；

Queue.put(item,[block[, timeout]])：将item消息写入队列，block默认值为True；

1）如果block使用默认值，且没有设置timeout（单位秒），消息列队如果已经没有空间可写入，此时程序将被阻塞（停在写入状态），直到从消息列队腾出空间为止，如果设置了timeout，则会等待timeout秒，若还没空间，则抛出"Queue.Full"异常；

1. 如果block值为False，消息列队如果没有空间可写入，则会立刻抛出"Queue.Full"异常；

Queue.put\_nowait(item)：相当Queue.put(item, False)；

### 16.4.2. 消息队列Queue完成进程间通信的演练

我们以Queue为例，在父进程中创建两个子进程，一个往Queue里写数据，一个从Queue里读数据：

import multiprocessing  
import time  
*# 写入数据*def write\_data(queue):  
 for i in range(10):  
 if queue.full():  
 print("队列满了")  
 break  
 queue.put(i)  
 time.sleep(0.2)  
 print('写入数据为:',i)  
*# 读取数据*def read\_data(queue):  
 while True:  
 *# 加入数据从队列取完了，那么跳出循环* if queue.qsize() == 0:  
 print("队列空了")  
 break  
 value = queue.get()  
 print('得到的数据:',value)  
if \_\_name\_\_ == '\_\_main\_\_':  
 *# 创建消息队列* queue = multiprocessing.Queue(5)  
  
 *# 创建写入数据的进程* write\_process = multiprocessing.Process(target=write\_data, args=(queue,))  
 *# 创建读取数据的进程* read\_process = multiprocessing.Process(target=read\_data, args=(queue,))  
  
 *# 启动进程* write\_process.start()  
 *# 主进程等待写入进程执行完成以后代码再继续往下执行* write\_process.join()  
 read\_process.start()

运行结果：

写入数据为: 0

写入数据为: 1

写入数据为: 2

写入数据为: 3

写入数据为: 4

队列满了

得到的数据: 0

得到的数据: 1

得到的数据: 2

得到的数据: 3

得到的数据: 4

队列空了

小结：

从队列取值使用get方法，向队列放入值使用put方法

消息队列判断队列是否为空不可靠，可以使用延时和根据个数进行判断

## 16.5 进程池Pool

### 16.5.1 进程池的概念

池子里面放的是进程，进程池会根据任务执行情况自动创建进程，而且尽量少创建进程，合理利用进程池中的进程完成多任务

当需要创建的子进程数量不多时，可以直接利用multiprocessing中的Process动态成生多个进程，但如果是上百甚至上千个目标，手动的去创建进程的工作量巨大，此时就可以用到multiprocessing模块提供的Pool方法。

初始化Pool时，可以指定一个最大进程数，当有新的请求提交到Pool中时，如果池还没有满，那么就会创建一个新的进程用来执行该请求；但如果池中的进程数已经达到指定的最大值，那么该请求就会等待，直到池中有进程结束，才会用之前的进程来执行新的任务.

### 16.5.2 进程池同步执行任务

进程池同步执行任务表示进程池中的进程在执行任务的时候一个执行完成另外一个才能执行，如果没有执行完会等待上一个进程执行

进程池同步实例代码

import multiprocessing  
import time  
*# 拷贝任务*def work():  
 print("复制中...", multiprocessing.current\_process().pid)  
 time.sleep(0.5)  
if \_\_name\_\_ == '\_\_main\_\_':  
 *# 创建进程池  
 # 3:进程池中进程的最大个数* pool = multiprocessing.Pool(3)  
 *# 模拟大批量的任务，让进程池去执行* for i in range(5):  
 *# 循环让进程池执行对应的work任务  
 # 同步执行任务，一个任务执行完成以后另外一个任务才能执行* pool.apply(work)

运行结果:

复制中... 100512

复制中... 68128

复制中... 98924

复制中... 100512

复制中... 68128

### 16.5.3 进程池异步执行任务

进程池异步执行任务表示进程池中的进程同时执行任务，进程之间不会等待

进程池异步实例代码

# 进程池:池子里面放的进程，进程池会根据任务执行情况自动创建进程，而且尽量少创建进程，合理利用进程池中的进程完成多任务

import multiprocessing  
import time  
*# 拷贝任务*def work():  
 print("复制中...", multiprocessing.current\_process().pid)  
 *# 获取当前进程的守护状态  
 # 提示：使用进程池创建的进程是守护主进程的状态，默认自己通过Process创建的进程是不是守住主进程的状态* print(multiprocessing.current\_process().daemon)  
 time.sleep(0.5)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 *# 创建进程池  
 # 3:进程池中进程的最大个数* pool = multiprocessing.Pool(3)  
 *# 模拟大批量的任务，让进程池去执行* for i in range(5):  
 *# 循环让进程池执行对应的work任务  
 # 同步执行任务，一个任务执行完成以后另外一个任务才能执行  
 # pool.apply(work)  
 # 异步执行，任务执行不会等待，多个任务一起执行* pool.apply\_async(work)  
 *# 关闭进程池，意思告诉主进程以后不会有新的任务添加进来* pool.close()  
 *# 主进程等待进程池执行完成以后程序再退出* pool.join()

执行结果:

复制中... 122872

复制中... 61772

复制中... 114636

复制中... 122872

复制中... 114636

小结:

multiprocessing.Pool常用函数解析：

* apply(func[, args[, kwds]]): 阻塞方式调用函数，args表示以元组方式给函数传参，kwds表示以字典方式给函数传参
* apply\_async(func[, args[, kwds]]) ：使用非阻塞方式调用函数，args表示以元组方式给函数传参，kwds表示以字典方式给函数传参
* close()：关闭Pool，使其不再接受新的任务；
* terminate()：不管任务是否完成，立即终止；
* join()：主进程阻塞，等待子进程的退出， 必须在close或terminate之后使用；

实战：文件夹拷贝器-多任务

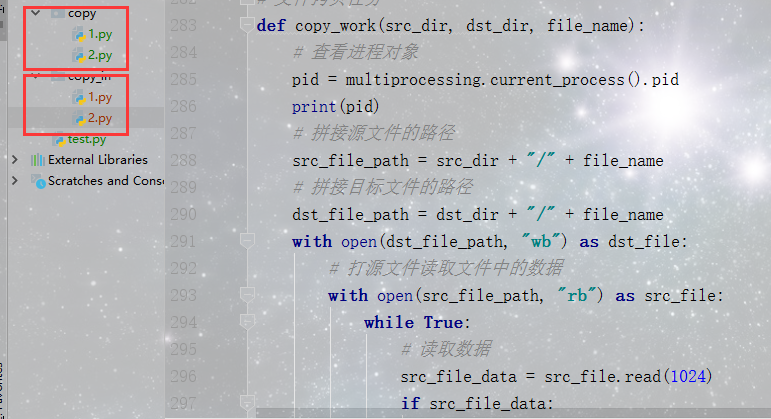
目标

使用进程池完成文件夹拷贝器，文件夹里面的文件能够实现多个文件一起拷贝

功能要求：

使用进程池完成多任务文件夹的拷贝

import os  
import shutil  
import multiprocessing  
*# import time  
# 文件拷贝任务*def copy\_work(src\_dir, dst\_dir, file\_name):  
 *# 查看进程对象* pid = multiprocessing.current\_process().pid  
 print(pid)  
 *# 拼接源文件的路径* src\_file\_path = src\_dir + "/" + file\_name  
 *# 拼接目标文件的路径* dst\_file\_path = dst\_dir + "/" + file\_name  
 with open(dst\_file\_path, "wb") as dst\_file:  
 *# 打源文件读取文件中的数据* with open(src\_file\_path, "rb") as src\_file:  
 while True:  
 *# 读取数据* src\_file\_data = src\_file.read(1024)  
 if src\_file\_data:  
 *# 写入到目标文件里面* dst\_file.write(src\_file\_data)  
 else:  
 break  
if \_\_name\_\_ == '\_\_main\_\_':  
 *# 源目录* src\_dir = "./copy/"  
 *# 目标目录* dst\_dir = "./copy\_in/"  
 *# 判断文件夹是否存在* if os.path.exists(dst\_dir):  
 *# 存在则删除文件夹及文件夹里面的所有文件* shutil.rmtree(dst\_dir)  
 *# 创建目标文件夹* os.mkdir(dst\_dir)  
 *# 获取源目录里面文件的列表* file\_name\_list = os.listdir(src\_dir)  
 *# 创建进程池* pool = multiprocessing.Pool(3)  
 *# 遍历文件里面获取文件名* for file\_name in file\_name\_list:  
 *# 使用进程池执行拷贝任务* pool.apply\_async(copy\_work, (src\_dir, dst\_dir, file\_name))  
 *# 关闭进程池* pool.close()  
 *# 主进程等待进程池执行完成以后程序再退出* pool.join()

小结

进程池在执行任务的时候会尽量少创建进程，合理利用现有进程完成多任务，这样可以减少资源开销

总结：

16.1 进程和线程的关系

16.2 进程的使用

16.3 进程中的变量

16.4 进程中的通信

16.5 进程池Pool