python_多线程

每一个应用就是一个进程，比如我们电脑上的qq等，进程有多个线程组成，线程和线程之间是独立的

python运行进程里面本身就有一个线程，这个线程叫主线程

多线程主要用threading模块就够了，通过声明对象，通过start启动线程

备注：一个start就是一个线程

例子如下：

```
import threading
import time


def lajifenlei():
    time.sleep(2)
    print(threading.current_thread().name+'haha haha') #打印线程名称及字符串


#非多线程方式运行，运行时间为20秒多一些
start_time=time.time()
for i in range(10):
    lajifenlei()
print(time.time()-start_time)


#多线程统计时间却是零点几，究其原因是由于主线程没有等子线程运行结束导致，即每个py文件运行都有一个主线程
#我们声明的是子线程
s_time=time.time()
for i in range(10):
    th=threading.Thread(target=lajifenlei) #声明线程，注意写方法名，不需要带括号
    th.start() #启动线程
print(time.time()-s_time)



#第一种方式等待子线程
s_time2=time.time()
threads=[]

for i in range(10):
    th2=threading.Thread(target=lajifenlei) #声明线程，注意写方法名，不需要带括号
    th2.start() #启动线程
    threads.append(th2) #把每个线程对象加入list
```

```
for t in threads:
    t.join() #等待子线程
print(time.time()-s_time2)



#第二种方法，主线程等待子线程
s_time3=time.time()

for j in range(10):
    th3=threading.Thread(target=lajifenlei) #声明线程，注意写方法名，不需要带括号
    th3.start() #启动线程

while threading.active_count()!=1: #当活跃的线程仅剩1的时候，结束等待，说明所有的子线程均已运行完成，只剩最后一个线程在运行
    pass

print(time.time()-s_time3)
```

多线程执行的方法的带参数，下面以下载文件为例，如下：

```
import requests,threading
import faker

f=faker.Faker(locale="zh-CN")
def down_load_file(url):
    r=requests.get(url) #get请求
    m=f.name() #随机生成文件名字
    with open('./pic/%s.jpg'%m,'wb') as fw: #图片二进制写入文件，保存为jpg格式
        fw.write(r.content)

url_list=[
            'http://www.nnzhp.cn/wp-content/uploads/2019/02/jiami.jpeg',
            'http://www.nnzhp.cn/wp-content/uploads/2019/03/js.png',
            'http://www.nnzhp.cn/wp-content/uploads/2018/08/ab389f04cb5b57344ef9655428bccaec.png'
        ]

for url in url_list:
    th=threading.Thread(target=down_load_file,args=(url,))#声明线程且执行的方法带参数，参数后面的逗号不可省略
    th.start() #启动线程

while threading.activeCount()!=1: #等待子线程
    pass
```

备注：因为Python的线程虽然是真正的线程，但解释器执行代码时，有一个GIL锁：Global Interpreter Lock，任何Python线程执行前，必须先获得GIL锁，然后，每执行100条字节码，解释器就自动释放GIL锁，让别的线程有机会执行。这个GIL全局锁实际上把所有线程的执行代码都给上了锁，所以，多线程在Python中只能交替执行，即使100个线程跑在100核CPU上，也只能用到1个核

不过，也不用过于担心，Python虽然不能利用多线程实现多核任务，但可以通过多进程实现多核任务。多个Python进程有各自独立的GIL锁，互不影响。

结论：多线程用在 io 密集，多进程用在 cpu 密集