线程与进程的联系

- 两者都是多任务编程方式,都能使用计算机的多核资源
- 一个进程可以创建多个线程分支,两者之间存在包含关系
- 一个进程中可以包含多个线程, 每条线程执行不同的任务
- 一个进程内至少有一个线程

线程与进程的区别

- 进程的创建所消耗的计算机资源比线程要多
- 进程空间独立,拥有自己的内存空间及数据变量,相互不干扰,线程存在进程中,所以同一个进程里的线程共享当前进程内的数据(进程间数据不共享,线程间共享同一进程内的数据)
- 由于线程比进程更小，基本上不拥有系统资源，故对它的调度所付出的开销就会小得多，能更高效的提高系统内多个程序间并发执行的程度
- 线程不能单独执行，它必须组成进程才能被执行。一个进程可以有多个线程，但是一个线程同时只能被一个进程所拥有

创建线程

```
import threading
from threading import Thread
```

函数式创建

```
import threading
from time import sleep, time

def work():
    print('子线程开始')
    sleep(3)
    print('子线程结束')

if __name__ == '__main__':
    st_time = time()
    t = Thread(target=work)
    t1 = Thread(target=work)
    t2 = Thread(target=work)
```

```
    t.start()
    t1.start()
    t2.start()
    t.join()
    t1.join()
    t2.join()
    end_time = time() - st_time
    print(end_time)
```

函数式创建带有参数的线程 并区分主线程

```
def listen(music, **kwargs):
    print('我正在听:{}, 编号是:{} 线程是:{}'.format(music, kwargs['name'],
threading.current_thread()))
    sleep(3)
    print('听歌结束')


if __name__ == '__main__':
    st_time = time()
    thread_list = []
    # 主线程的线程编号和名字
    # print(threading.main_thread())
    print('主进程是:{} 主线程是:{}'.format(os.getpid(), threading.currentThread()))
    for i in range(3):
        t = Thread(target=listen, args=('爱情买卖',), kwargs={'name':i})
        t.start()
        thread_list.append(t)
    print(p.is_alive())
    print(threading.active_count())
    # 输出线程的名字  同i.getName()
    print([i.name for i in thread_list])

    print([i.is_alive()for i in thread_list])

    [i.join() for i in thread_list]

    print('_'*20)
    print(threading.activeCount())

    print([i.is_alive()for i in thread_list])

    end_time = time() - st_time
    print('共耗时:{}'.format(end_time))
    print(os.getpid())
    print(threading.activeCount())
```

```
threading.current_thread()  #打印当前执行的线程名
threading.active_count()    #打印当前存活的线程数量
os.getpid()  #验证当前进程号
```

继承式创建

```
class mythread(threading.Thread):

    # 改写父类的run方法
    def run(self):
        print ('子线程开始')
        print('task:{}'.format(self.n))
        sleep(2)
        print ('子线程结束')

t = mythread()
t1 = mythread()
t.start()
t1.start()
t.join()
t1.join()
```