threading&thread

threading是thread的高级模块，使用threading

--t = threading.Thread(target = func,args = (参数,)[,name = "..."])

args参数形式必须是元组

主程序是MainThread

--run()

第二种方法，重写run()+类实例化

FE：

```
class myThread(threading.Thread):
 def __init__(self,func,args):
  #threading.Thread.__init__(self)
  super().__init__(func,args)
  self._func = func
  self._args = args
 def run(self):
  Sum(*self._args)

n = [100,100]
for i in range(len(n)):
 t = myThread(Sum,(n[i],))
 my.append(t)
......
```

--t.start()

开始线程执行

--t.join(timeout = None)

程序挂起，直到线程结束

--t.getName()

获得线程名字

--t.setName(name)

设置线程名字

--t.isAlive()

布尔标志，表示线程是否在进行

--t.isDaemon()

返回daemon标志

--t.setDaemon(daemonic)

把daemon标志设置为daemonic，一定在调用start()前设置

拓：

Daemon守护线程

当主线程崩溃，子线程会执行

--threading.current_thread().name

获得主线程名字

--threading.activeCount()

当前活动的线程对象的数量

--threading.currentThread()
返回当前线程对象

--threading.enumerate()
返回当前活动线程的列表

--lock = threading.Lock()
锁标志，一般用于函数输出语句之前（print之前）

--lock.acquire()
获得锁标志
```
if lock.acquire():
    ...
```

--lock.release()
释放锁标志
FE：
```
if lock.acquire():
    ...
    lock.release()
```

--t = threading.RLock()
可重入锁对象。让单线程获得已经获得了的锁

--t = threading.Condition()
条件变量对象。让一个线程pause。等待其他线程满足条件后，再执行

--t=threading.Event()
通用条件变量。多个线程等待时间发生后，一起被激活。