线程创建

Python里面经常会用到多线程，即所有的方法在同一时间开始运行，而不是按顺序一个一个运行。
相对于 thread 包，threading 包提供了更多的功能。该包的用法基本分成两步：

- 第一步是构造一个 threading.Thread 实例对象，这时该对象对应的线程就处于“新建”状态；
- 第二步是操作该对象，如调用 start() 来将该线程转换到“就绪”状态。

Thread类的构造函数定义如下

```
class threading.Thread(group=None, target=None, name=None, args=(), kwargs={})
```

- group：留作ThreadGroup扩展使用，一般没什么用
- target：新线程的任务函数名
- name：线程名，一般也没什么用
- args： tuple参数
- kwargs：dictionary参数

Thread类的成员变量和函数如下

- start() ：启动一个线程
- run()：线程执行体，也是一般要重写的内容
- join([timeout]) ：等待线程结束
- name：线程名
- ident ：线程ID
- daemon：是否守护线程
- isAlive()、is_alive()：线程是否存活
- getName()、setName()：Name的get&set方法
- isDaemon()、setDaemon()：daemon的get&set方法

创建 Thread 对象有 2 种方法

1.直接创建 Thread 对象

Thread 的构造方法中，最重要的参数是 target，所以我们需要将一个 callable 对象赋值给它，线程才能正常运行。
如果要让一个 Thread 对象启动，调用它的 start() 方法就好了。

```
import threading
import time

def test():
    for i in range(5):
```

```
        print('test ',i)
        time.sleep(1)
thread = threading.Thread(target=test,name='TestThread')
 #Thread 对象创建时，可在构造方法里面赋值。
thread.start()
thread.join() #阻塞自身所在的线程

for i in range(5):
    print('main ', i)
    print(threading.current_thread().name+' main ', i) #name 属性，代表线程的名字，thread.current_thread() 方法可以返回线程本身
    print(thread.name+' is alive ', thread.isAlive()) #is_alive() 方法查询线程是否还在运行
    time.sleep(1)
```

上面的 callable 没有参数，如果需要传递参数的话，args 是固定参数，kwargs 是可变参数。

2.自定义类继承 Thread

编写一个自定义类继承 Thread，然后复写 run() 方法，在 run() 方法中编写任务处理代码，然后创建这个 Thread 的子类。

```
import threading
import time

class TestThread(threading.Thread):

    def __init__(self,name=None):
        threading.Thread.__init__(self,name=name)

    def run(self):
        for i in range(5):
            print(threading.current_thread().name + ' test ', i)
            time.sleep(1)

thread = TestThread(name='TestThread')
thread.start()

for i in range(5):
    print(threading.current_thread().name+' main ', i)
    print(thread.name+' is alive ', thread.isAlive())
    time.sleep(1)
```

Queue的用法详解

多个线程之间的数据是共享的，多个线程进行数据交换时，不能够保证数据的安全性和一致性，所以当多个线程需要进行数据交换时，队列可以完美解决线程间的数据交换，保证线程间数据的安全性和一致性。

queue模块有三种队列及构造函数

- Python queue模块的FIFO队列先进先出。 queue.Queue(maxsize)
- LIFO类似于堆，即先进后出。 queue.LifoQueue(maxsize)
- 还有一种是优先级队列级别越低越先出来。queue.PriorityQueue(maxsize)

queue模块中的常用方法

- queue.qsize() 返回队列的大小
- queue.empty() 如果队列为空，返回True,反之False
- queue.full() 如果队列满了，返回True,反之False
- queue.full 与 maxsize 大小对应
- queue.get([block[, timeout]])获取队列，立即取出一个元素， timeout超时时间
- queue.put(item[, timeout]]) 写入队列，立即放入一个元素， timeout超时时间
- queue.get_nowait() 相当于queue.get(False)
- queue.put_nowait(item) 相当于queue.put(item, False)
- queue.join() 阻塞调用线程，直到队列中的所有任务被处理掉, 实际上意味着等到队列为空，再执行别的操作
- queue.task_done() 在完成一项工作之后，queue.task_done()函数向任务已经完成的队列发送一个信号