**python 多线程**

**单线程**

from time import ctime,sleep

def music():

for i in range(2):

print "I was listening to music. %s" %ctime()

sleep(1)

def move():

for i in range(2):

print "I was at the movies! %s" %ctime()

sleep(5)

if \_\_name\_\_ == '\_\_main\_\_':

music()

move()

print "all over %s" %ctime()

运行结果：

>>=========================== RESTART ================================

>>>

I was listening to music. Thu Apr 17 10:47:08 2014

I was listening to music. Thu Apr 17 10:47:09 2014

I was at the movies! Thu Apr 17 10:47:10 2014

I was at the movies! Thu Apr 17 10:47:15 2014

all over Thu Apr 17 10:47:20 2014

　　其实，music()和move()更应该被看作是音乐和视频播放器，至于要播放什么歌曲和视频应该由我们使用时决定。所以，我们对上面代码做了改造：

#coding=utf-8

import threading

from time import ctime,sleep

def music(func):

for i in range(2):

print "I was listening to %s. %s" %(func,ctime())

sleep(1)

def move(func):

for i in range(2):

print "I was at the %s! %s" %(func,ctime())

sleep(5)

if \_\_name\_\_ == '\_\_main\_\_':

music(u'爱情买卖')

move(u'阿凡达')

print "all over %s" %ctime()

　　对music()和move()进行了传参处理。运行结果：

>>> ======================== RESTART ================================

>>>

I was listening to 爱情买卖. Thu Apr 17 11:48:59 2014

I was listening to 爱情买卖. Thu Apr 17 11:49:00 2014

I was at the 阿凡达! Thu Apr 17 11:49:01 2014

I was at the 阿凡达! Thu Apr 17 11:49:06 2014

all over Thu Apr 17 11:49:11 2014

**多线程**

python提供了两个模块来实现多线程thread 和threading ，thread 有一些缺点，在threading 得到了弥补，为了不浪费你和时间，所以我们直接学习threading 就可以了。

#coding=utf-8

import threading

from time import ctime,sleep

def music(func):

for i in range(2):

print "I was listening to %s. %s" %(func,ctime())

sleep(1)

def move(func):

for i in range(2):

print "I was at the %s! %s" %(func,ctime())

sleep(5)

threads = []

t1 = threading.Thread(target=music,args=(u'爱情买卖',))

threads.append(t1)

t2 = threading.Thread(target=move,args=(u'阿凡达',))

threads.append(t2)

if \_\_name\_\_ == '\_\_main\_\_':

for t in threads:

t.setDaemon(True)

t.start()

print "all over %s" %ctime()

import threading

首先导入threading 模块，这是使用多线程的前提。

threads = []

t1 = threading.Thread(target=music,args=(u'爱情买卖',))

threads.append(t1)

　　创建了threads数组，创建线程t1,使用threading.Thread()方法，在这个方法中调用music方法target=music，args方法对music进行传参。 把创

建好的线程t1装到threads数组中。

　　接着以同样的方式创建线程t2，并把t2也装到threads数组。

for t in threads:

　　t.setDaemon(True)

　　t.start()

最后通过for循环遍历数组。（数组被装载了t1和t2两个线程）

setDaemon()

　　setDaemon(True)将线程声明为守护线程，必须在start() 方法调用之前设置，如果不设置为守护线程程序会被无限挂起。子线程启动后，父线程也继续执行下去，当父线程执行完最后一条语句print "all over %s" %ctime()后，没有等待子线程，直接就退出了，同时子线程也一同结束。

start()

开始线程活动。

运行结果：

>>> ========================= RESTART ================================

>>>

I was listening to 爱情买卖. Thu Apr 17 12:51:45 2014 I was at the 阿凡达! Thu Apr 17 12:51:45 2014 all over Thu Apr 17 12:51:45 2014

　　从执行结果来看，子线程（muisc 、move ）和主线程（print "all over %s" %ctime()）都是同一时间启动，但由于主线程执行完结束，所以导致子线程也终止。

继续调整程序：

...

if \_\_name\_\_ == '\_\_main\_\_':

for t in threads:

t.setDaemon(True)

t.start()

t.join()

print "all over %s" %ctime()

　　我们只对上面的程序加了个join()方法，用于等待线程终止。join（）的作用是，在子线程完成运行之前，这个子线程的父线程将一直被阻塞。

　　注意:  join()方法的位置是在for循环外的，也就是说必须等待for循环里的两个进程都结束后，才去执行主进程。运行结果：

>>> ========================= RESTART ================================

>>>

I was listening to 爱情买卖. Thu Apr 17 13:04:11 2014

I was at the 阿凡达! Thu Apr 17 13:04:11 2014

I was listening to 爱情买卖. Thu Apr 17 13:04:12 2014

I was at the 阿凡达! Thu Apr 17 13:04:16 2014

all over Thu Apr 17 13:04:21 2014

　　从执行结果可看到，music 和move 是同时启动的。

　　开始时间4分11秒，直到调用主进程为4分22秒，总耗时为10秒。从单线程时减少了2秒，我们可以把music的sleep()的时间调整为4秒。

...

def music(func):

for i in range(2):

print "I was listening to %s. %s" %(func,ctime())

sleep(4)

...

执行结果：

>>> ====================== RESTART ================================

>>>

I was listening to 爱情买卖. Thu Apr 17 13:11:27 2014

I was at the 阿凡达! Thu Apr 17 13:11:27 2014

I was listening to 爱情买卖. Thu Apr 17 13:11:31 2014

I was at the 阿凡达! Thu Apr 17 13:11:32 2014

all over Thu Apr 17 13:11:37 2014

　　子线程启动11分27秒，主线程运行11分37秒。

　　虽然music每首歌曲从1秒延长到了4 ，但通多程线的方式运行脚本，总的时间没变化。