

## 线程多任务编程

笔记本：多任务编程

创建时间：2018/5/16 17:33

更新时间：2018/5/16 22:07

作者：ly

标签：threading, threading.Thread, time, time.sleep(1)

**python的thread**模块是比较底层的模块，python的threading模块是对thread做了一些包装的，可以更加方便的被使用。

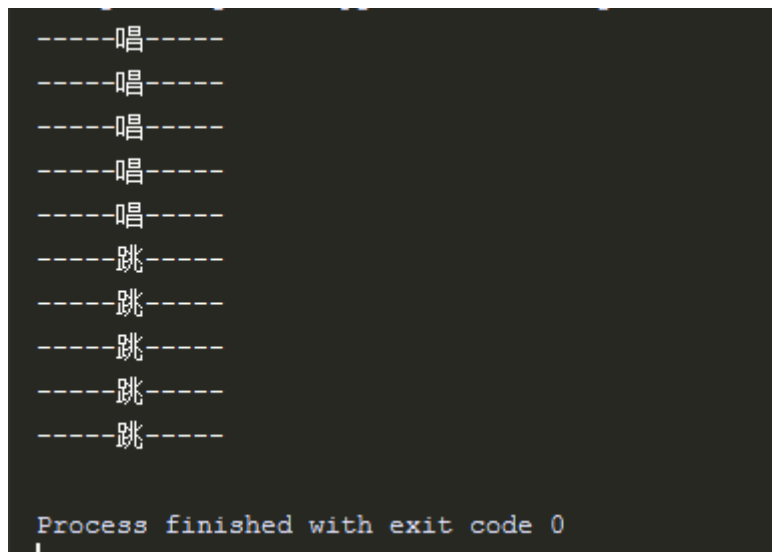
### 单任务基础

```
import time

def sing():
    for i in range(5):
        print("-----唱-----")
        time.sleep(1)

def dance():
    for i in range(5):
        print("-----跳-----")
        time.sleep(1)

sing()
dance()
```



```
-----唱-----
-----唱-----
-----唱-----
-----唱-----
-----唱-----
-----跳-----
-----跳-----
-----跳-----
-----跳-----
-----跳-----

Process finished with exit code 0
```

唱歌和跳舞没有同时进行，有很明显的先后顺序

### 多任务编程基础实现

```
import time    # 时间模块
import threading # 线程模块

def sing():
    for i in range(5):
        print("-----唱-----")
        time.sleep(1)    # 休眠1秒钟 sleep睡
```

```
def dance():
    for i in range(5):
        print("-----跳-----")
        time.sleep(1)

# 指定线程处理函数
t1 = threading.Thread(target=sing)
t2 = threading.Thread(target=dance)

# 当调用start方法的时候，才会去真正创建一个线程，并立马开始执行
# 在创建这个t1指向的Thread对象时target指定的是那个函数名，那么这个新的线程就会到哪个函数中执行代码
t1.start()
t2.start()
# t1和t2同时进行
```

```
-----跳-----
-----唱-----
-----唱-----
-----跳-----
-----跳-----
-----唱-----
-----跳-----
-----唱-----
-----唱-----
-----跳-----

Process finished with exit code 0
```

线程数量查看 enumerate()查看线程号

```
import threading
import time

def dance():
    for i in range(3):
        print("-----唱-----")
        time.sleep(2)

def sing():
    for i in range(3):
        print("-----跳-----")
        time.sleep(1)

print("1,在t1创建前的线程号：", threading.enumerate())
t1 = threading.Thread(target=dance)
print("2,在t1创建后，t2创建前的线程号：", threading.enumerate()) # enumerate()查看线程号

t2 = threading.Thread(target=sing)
print("3，在t2创建后的线程号：", threading.enumerate())

# 当带调用start方法的时候才会真真的创建一个线程
```

```
t1.start()
print("4, 在t1启动后, t2启动前线程号:", threading.enumerate())

t2.start()
print("5, 所有线程都启动后线程号:", threading.enumerate())
```

```
1, 在t1创建前的线程号: [<_MainThread(MainThread, started 18932)>]
2, 在t1创建后, t2创建前的线程号: [<_MainThread(MainThread, started 18932)>]
3, 在t2创建后的线程号: [<_MainThread(MainThread, started 18932)>]
----唱-----
4, 在t1启动后, t2启动前线程号: [<_MainThread(MainThread, started 18932)>, <Thread(Thread-1, started 20100)>]
----跳-----
5, 所有线程都启动后线程号: [<_MainThread(MainThread, started 18932)>, <Thread(Thread-1, started 20100)>, <Thread(Thread-2, started 20856)>]
----跳-----
----唱-----
----唱-----

Process finished with exit code 0
```

主线程号      t1子线程号      t2子线程号

## 主线程等待子线程

- # 当程序启动时, 即在一个单独的线程中运行, 叫主程序。
- # 在主线程中新创建的线程叫子线程
- # 主线程会等待所有子线程先结束才会运行
- # 两个子进程是【随机】运行 但是第一次主程序一定等子程序运行完 才会运行

```
import threading
import time

def sing():
    for i in range(3):
        print("-----")
        time.sleep(1)

def dance():
    for i in range(3):
        print("+++++")
        time.sleep(0.5)

t1 = threading.Thread(target=sing)
t2 = threading.Thread(target=dance)

t2.start()
t1.start()

def main():
    for i in range(3):
        print("*****")

main()
```

第一次运行

```
+++++++  
-----  
*****  
*****  
*****  
+++++++  
+++++++  
-----  
-----  
  
Process finished with exit code 0
```

上面两个子线程随机抢占运行  
主程序等待子进程运行完，才会执行

主进程无线循环：等待子进程

```
def sing():  
    for i in range(10):  
        print("-----")  
        time.sleep(0.5)  
  
def dance():  
    for i in range(10):  
        print("+++++")  
        time.sleep(0.5)  
  
t1 = threading.Thread(target=sing)  
t2 = threading.Thread(target=dance)  
  
t1.start()  
t2.start()  
  
while True:  
    print(threading.enumerate())  
    time.sleep(0.5)
```

主进程会等待所有子线程先结束  
多线程的执行顺序是不确定

使用Thread类的子类创建线程

为了让每个线程的封装性更完美，在使用threading模块时，往往会定义一个新的子类class，只要继承threading.Thread就可以了，然后重写run方法。

```
import threading  
import time
```

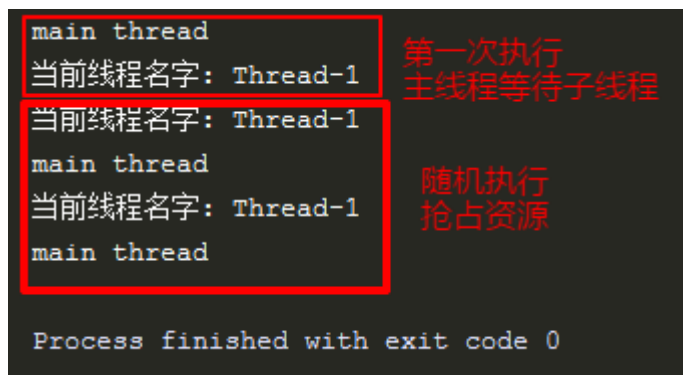
```

class myThread(threading.Thread):
    """定义创建线程类"""
    def run(self):    # 重写父类Thread的run方法
        for i range(3):
            print("当前线程名字: ", self.name)

t = myThread()    # 通过类创建一个子线程==>t1 = threading.Thread(target=run)
t.start()    # 底层优先调用了run方法，但是由于重写了，所以先调自己写的。

for i range(3):
    time.sleep(1)
    print("main thread")

```



```

main thread
当前线程名字: Thread-1
main thread
当前线程名字: Thread-1
main thread
当前线程名字: Thread-1
main thread

Process finished with exit code 0

```

第一次执行  
主线程等待子线程

随机执行  
抢占资源

## 多线程共享全局变量

### 多线程普通方法共享全局变量实现

```

import threading
import time

g_unm = 100    # 定义全局变量

def test1():
    global g_unm    # 因为下面要修改变量，所以要声明变量为函数外的全局变量
    g_unm -= 100
    print(g_unm)

def test2():
    print(g_unm)    # 只用打印

t1 = threading.Thread(target=test1)
t1.start()

time.sleep(1)    # 保证t1先执行完

t2 = threading.Thread(target=test2)
t2.start()

```

输出：

```

0
0

```

### 元组小结

```

a = (1)

```

```
b = (1,)

print(type(a))    ==> int
print(type(b))    ==> tuple
```

元组中只有一个元素时，会自动推到类型，如果需要它继续保持元组类型，就在元素后面添加逗号，

```
In [1]: a = (1)

In [2]: b = (1,)
        ]

In [3]: type(a)
Out[3]: int

In [4]: type(b)
Out[4]: tuple
```

列表当做实参传递到线程中

```
import time
import threading

g_nums = [11, 22]

def test1(temp):
    temp.append(30)    # 增加元组中的数据 没有改变本身 不需要global
    print(g_nums)

def test2():
    print(g_nums)

t1 = threading.Thread(target=test1, args=(g_nums,))    # 以元组的方式传参
t1.start()

time.sleep(1)    # 保证test1先执行完

t2 = threading.Thread(target=test2)
t2.start()
```

```
[11, 22, 33]
[11, 22, 33]

Process finished with exit code 0
```

资源共享带来的资源竞争问题

```
import threading
```

```
import time

g_num = 0

def test1(num):
    global g_num
    for i in range(num):
        g_num += 1

    print(g_num)

def test2(num):
    global g_num
    for i in range(num):
        g_num += 1

    print(g_num)

t1 = threading.Thread(target=test1, args=(10000000,))
t2 = threading.Thread(target=test2, args=(10000000,))

t1.start()
t2.start()

time.sleep(3) # 延时：让线程都执行完毕
print(g_num)
```

```
5620621
11795352
12044355

Process finished with exit code 0
```

资源竞争 CPU资源竞争

资源竞争问题解决办法：互斥锁

同步与互斥

互斥：一个公共资源同一时刻只能被一个进程或线程使用，多ge