进程

计算机程序都是保存在磁盘里的。只有被加载到内存中被操作系统调用，才有生命周期。进程：表示一个执行中的程序，正如我们打开任务管理器所看到的的一样，运行了很多系统进程。有时候称之为：重量级进程

线程

线程（有时候称为：轻量级进程）他们是在进程下执行的，类似于一条主线程下还能调用其他小线程

多线程概念---（老鸟略过）

假如不用多线程-程序只有一条主线程来跑程序需要使用10秒,而调用两条线程去跑理论上说五秒钟就可以跑完，这大大节约了时间，让计算机的性能最大化，大大提升了效率。在爬虫方面多线程是必不可少的，本是一条主线程爬取一个网页，假如通过主线程同时运行了十条子线程，同时爬取十种不同的网页，而且还是用爬取一次的时间完成，这是不是很爽呢？ 上述废话了很多，一个简单的多线程的概念

> “合理”使用多线程，让程序快人一步，让自己的程序更迷人。

python的多线程使用

python使用兼容POSIX的线程，也就是众所周知的pthread

```
import thread
```

首先导入多线程模块 如果你的python解释器没有将线程支持编译进去，模块导入将失败，无法找到 在`Linux`下你可能需要重新编译你的解释器，window重装一次python环境

我们先来看下如果没有使用多线程的情况:
创建两个时间循环：一个`loop0()`睡眠4秒
一个`loop1()`睡眠2秒 整个执行时间至少会达到六秒以上 再加上一些定义、执行、声明等的开销 可能长达7秒

```
#!/usr/bin/env python
from time import sleep, ctime
def loop0():
    print 'start loop 0 at:', ctime()
    sleep(4)
    print 'loop 0 done at:', ctime()

def loop1():
    print 'start loop 1 at:', ctime()
    sleep(2)
    print 'loop 1 done at:', ctime()

def main():
    print 'starting at:', ctime()
    loop0()
    loop1()
    print 'all DONE at:', ctime()

if __name__ == '__main__':
    main()
```

执行输出以下结果：

```
starting at: Wed Feb 15 19:46:30 2017
start loop 0 at: Wed Feb 15 19:46:30 2017
loop0 done at: Wed Feb 15 19:46:34 2017
start loop 1 at: Wed Feb 15 19:46:34 2017
loop 1 done at: Wed Feb 15 19:46:36 2017
all DONE at: Wed Feb 15 19:46:40 2017
```

假如loop0和loop1是独立的程序 那么我们是不是可以通过多线程一起执行呢节约时间，当然这才两秒钟 如果程序很多呢

使用之前介绍一下thread模块中的方法

函数 / 方法	描述
start_new_thread(function, args, kwargs=None)	生产一个新的线程，使用给定的args和可选的kwargs来执行function
allocate_lock()	分配LockType锁对象(执行 创建一个锁的实例)
exit()	发送退出线程指令给线程

LockType 锁对象的方法	描述
acquire(wait=None)	加锁
loced()	如果获取到加锁返回True，反之False
release()	释放锁(解锁)

```
#!/usr/bin/env python

import thread # 导入thread模块
from time import sleep, ctime
def loop0():
    print 'start loop 0 at:', ctime()
    sleep(4)
    print 'loop 0 done at:', ctime()

def loop1():
    print 'start loop 1 at:', ctime()
    sleep(2)
    print 'loop 1 done at:', ctime()

def main():
    print 'starting at:', ctime()
    thread.start_new_thread( loop0, () ) # 生产一个新的线程
    thread.start_new_thread( loop1, () ) # 生产一个新的线程
    sleep( 6 )
    print 'all DONE at:', ctime()

if __name__ == '__main__':
    main()
```

输出以下结果：

```
starting at: Wed Feb 15 20:27:13 2017
start loop 0 at:start loop 1 at:  Wed Feb 15 20:27:13 2017Wed Feb 15 20:27:13 2017

loop 1 done at: Wed Feb 15 20:27:15 2017
loop 0 done at: Wed Feb 15 20:27:17 2017
all DONE at: Wed Feb 15 20:27:19 2017
```

`start_new_thread()`必须传前两个参数，也就是说第二个参数不传值也需要传递一个空元祖`tuple`
可以通过看出 loop0 和loop1 是并发执行的，节约了两秒执行时间。

- 为什么要加一个`sleep(6)`我们不妨去掉可以尝试下
  他是和执行loop函数一起执行了最后一句代码打印 完成

> 通过以上案例可以看出 生产出来的两个子线程之间相互不影响 并且主线程 还会继续往下执行

sleep(6) 并不靠谱 执行程序你会算时间吗 显然不会 如果不加会使程序提前或过晚退出主线程，这就是引出 锁的原因

修改代码如下：

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

import thread
from time import sleep, ctime

loops = [4, 2]# 第一次延时4秒  第二次延时两秒

# 定义loop函数 为创建多线程 做准备
def loop (nloop, nsec, lock): # 参一：执行线程名字  参二：延时/s  参三：传递"锁"的对象
    # 执行前输出信息  ctime记录当前时间
    print 'start loop', nloop, 'at:', ctime()
    # 延时多少秒后执行下面
    sleep( nsec )
    # 本条线程执行完成 输出文字
    print 'loop', nloop, 'done at:', ctime()
    # 释放锁  传进来的锁，让他解锁
    lock.release()

# 主函数
def main():
    # 输出执行主函数时间
    print 'starting at:', ctime()
    # 定义一个空列表 用来装锁对象
    locks = []
    # 生成0,1的列表 用来迭代
    nloops = range(len(loops))

    # 迭代0 1  生成两个锁对象
    for i in nloops:
        # 分配两个锁对象
        lock = thread.allocate_lock()
        # 给锁对象加上锁  执行加锁方法
        lock.acquire()
        # 把加好锁的对象放进列表 一同传参
        locks.append(lock)

    for i in nloops:# 创建两条子线程相互不影响
        # 创建子线程 把锁对象和延时多少秒传进去
        thread.start_new_thread( loop, (i, loops[i], locks[i]) )

    # 执行死循环 当锁列表里的两个锁对象都被释放(解锁) 的时候跳出循环
    # 为了等待子线程执行 锁释放(解锁)lock.release()  然后再执行主线程
    for i in nloops:
        while locks[i].locked():pass
    # 锁都被释放完 主线程执行到这 打印输出 执行完成
    print 'all Dat:', ctime()

# 主函数入口
if __name__ == '__main__':
    main()
```

执行结果:

```
starting at: Wed Feb 15 22:53:05 2017

start loopstart loop  10  at:at:  Wed Feb 15 22:53:05 2017Wed Feb 15 22:53:05 2017

loop 1 done at: Wed Feb 15 22:53:07 2017

loop 0 done at: Wed Feb 15 22:53:09 2017

all Dat: Wed Feb 15 22:53:09 2017
```

由于`GIL`　在python里的多线程是假的，当然会采用调用`c语言的多线程`

经过以上应该有了一个多线程的概念
这只是`假多线`程中最简单的一个模块

稍加改下代码 就可实现 **同时**爬取两个不同网页，和单线程爬取一个网页时间差不多相同

想要获取源码？想看其他不错的文章？↓↓↓↓↓↓

想学习更牛逼的多线程 或 多线程高级用法以及并发、协程 新技欢迎加入本群：

python交流群：526929231本群是学习群【新手也可以加】广告请绕道谢谢！

还有一些很好的技术文章尽情戳知了课堂官方QQ：2156600937