

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python？

02 我会怎样带你学 Python？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编程

24 让你的代码更灵活：进程和线程

更新时间：2019-10-21 10:26:35



“

学习知识要善于思考，思考，再思考。

—— 爱因斯坦

”

进程和线程是操作系统所提供的，能让程序在同一时间处理多个任务的方法，让程序能够做到「一心二用」。

关于进程和线程的具体概念，可以参考这篇通俗易懂的文章 [进程与线程的一个简单解释](#)。

进程

当我们运行一个程序时，这个程序的代码会被操作系统加载内存中，并创建一个进程来承载和运行它。简单来说，每一个运行中的程序就是一个进程，这个进程被称为主进程。

在主进程中，我们可以创建子进程来协助处理其它任务，这时主进程和子进程是并行运行的。子进程也可以有它的子进程，从而形成以主进程为根的一棵进程树。

我们可以使用 `multiprocessing.Process()` 方法来创建进程：

```
import multiprocessing

p = multiprocessing.Process(target=目标函数, args=(目标函数的参数,))
```

用 `start()` 方法来启动一个进程：

```
p.start()
```

来看个例子：

<div><div>← 慕课专栏</div><div>≡ 你的第一本Python基础入门书 / 24 让你的代码更灵活：进程和线程</div></div>	
目录	
第 1 章 入门准备	
01 开篇词：你为什么要学 Python ？	
02 我会怎样带你学 Python ？	
03 让 Python 在你的电脑上安家落户	
04 如何运行 Python 代码 ？	<pre>def target_func(): print('子进程运行') print('子进程 pid:', os.getpid()) print('子进程的 ppid:', os.getppid()) p = multiprocessing.Process(target=target_func) p.start() print('主进程运行') print('主进程 pid:', os.getpid())</pre>
	将上述代码拷贝至文件 <code>process.py</code> 中，执行下：
第 2 章 通用语言特性	
05 数据的名字和种类—变量和类型	
06 一串数据怎么存—列表和字符串	
07 不只有一条路—分支和循环	
08 将代码放进盒子—函数	<pre>→ ~ python3 process.py 主进程运行 主进程 pid: 13343 子进程运行 子进程 pid: 13344 子进程的 ppid: 13343</pre>
09 知错能改—错误处理、异常机制	
10 定制一个模子—类	
11 更大的代码盒子—模块和包	
12 练习—密码生成器	
第 3 章 Python 进阶语言特性	
13 这么多的数据结构（一）：列表、元祖、字符串	
14 这么多的数据结构（二）：字典、集合	
15 Python大法初体验：内置函数	
16 深入理解下迭代器和生成器	
17 生成器表达式和列表生成式	
18 把盒子升级为豪宅：函数进阶	
19 让你的模子更好用：类进阶	
20 从小独栋升级为别墅区：函数式编程	<pre>import multiprocessing import os def target_func(): print('子进程运行') print('子进程 pid:', os.getpid()) print('子进程的 ppid:', os.getppid()) p = multiprocessing.Process(target=target_func) p.start() p.join() print('主进程运行') print('主进程 pid:', os.getpid())</pre>

在这里例子中，

- 使用 `multiprocessing.Process()` 来创建进程，并为该进程指定要执行的目标函数 `target_func`，进程启动后将执行该函数
- 使用 `start()` 方法来启动进程
- 使用 `os.getpid()` 获取进程的进程 ID，它是进程的唯一标识，可用于区分进程
- 使用 `os.getppid()` 获取进程的父进程 ID，父进程是创建子进程的进程
- 主进程的 `pid` 和子进程的 `ppid` 相同（因为主进程是该子进程的父进程）

另外可以看到，虽然子进程被创建并启动，但子进程中的 `print()` 函数并未立即执行，反而是主进程中的 `print()` 函数先执行。这说明进程间的执行顺序是不确定的，并非同步执行。

使用 `join()` 方法可以控制子进程的执行顺序：

上述代码中新增了 `p.join()`。相应修改原先的 `process.py` 文件，再来执行下：

<div>← 慕课专栏</div> <div>你的第一本Python基础入门书 / 24 让你的代码更灵活：进程和线程</div>	
目录	
第 1 章 入门准备	
01 开篇词：你为什么要学 Python ？	子进程 pid: 13386 子进程的 ppid: 13385 主进程运行 主进程 pid: 13385
02 我会怎样带你学 Python ？	可以看到，使用 <code>p.join()</code> 后主进程将等待子进程执行完成，然后再向下执行代码。
03 让 Python 在你的电脑上安家落户	线程
04 如何运行 Python 代码？	每一个进程都默认有一个线程，这个线程被称为主线程。我们可以在主线程中创建其它线程来协助处理任务，这些线程也是并行运行的。
第 2 章 通用语言特性	线程是进程的执行单元，CPU 调度进程时，实际上是在进程的线程间作切换。另外线程间共享它们所在进程的内存空间（栈除外）。
05 数据的名字和种类—变量和类型	可以使用 <code>threading.Thread()</code> 方法来创建线程：
06 一串数据怎么存—列表和字符串	<pre>import threading t = threading.Thread(target=目标函数, args=(目标函数的参数,))</pre>
07 不只有一条路—分支和循环	用 <code>start()</code> 方法来启动一个线程：
08 将代码放进盒子—函数	<pre>t.start()</pre>
09 知错能改—错误处理、异常机制	来看个例子：
10 定制一个模子—类	<pre>import threading def target_func(n): for i in range(n): print(i) t = threading.Thread(target=target_func, args=(8,)) t.start() print('主线程结束')</pre>
11 更大的代码盒子—模块和包	将上述代码拷贝至文件 <code>thread.py</code> 中，执行下：
12 练习—密码生成器	<pre>→ ~ python3 thread.py 0 1 主线程结束 2 3 4 5</pre>
第 3 章 Python 进阶语言特性	
13 这么多的数据结构（一）：列表、元祖、字符串	
14 这么多的数据结构（二）：字典、集合	
15 Python大法初体验：内置函数	
16 深入理解下迭代器和生成器	
17 生成器表达式和列表生成式	
18 把盒子升级为豪宅：函数进阶	
19 让你的模子更好用：类进阶	
20 从小独栋升级为别墅区：函数式编程	

<div><div>← 慕课专栏</div><div>≡ 你的第一本Python基础入门书 / 24 让你的代码更灵活：进程和线程</div></div>	
目录	
第 1 章 入门准备	上述子线程和主线程交替执行，可以使用 <code>join()</code> 让主线程等待子线程执行完成：
01 开篇词：你为什么要学 Python ？	<pre>import threading def target_func(n): for i in range(n): print(i) t = threading.Thread(target=target_func, args=(8,)) t.start() t.join() print('主线程结束')</pre>
02 我会怎样带你学 Python ？	
03 让 Python 在你的电脑上安家落户	
04 如何运行 Python 代码 ？	
第 2 章 通用语言特性	上述代码中新增了 <code>t.join()</code> 。相应修改原先的 <code>thread.py</code> 文件，再来执行下：
05 数据的名字和种类—变量和类型	<pre>→ ~ python3 thread.py 0 1 2 3 4 5 6 7 主线程结束</pre>
06 一串数据怎么存—列表和字符串	
07 不只有一条路—分支和循环	
08 将代码放进盒子—函数	
09 知错能改—错误处理、异常机制	
10 定制一个模子—类	
11 更大的代码盒子—模块和包	
12 练习—密码生成器	
第 3 章 Python 进阶语言特性	线程锁
13 这么多的数据结构（一）：列表、元祖、字符串	多个线程间回共享进程的内存空间，如果多个线程同时修改和访问同一个对象，则可能会出现非预期的错误。
14 这么多的数据结构（二）：字典、集合	比如下面这个例子中，我们创建了两个线程，这两个线程分别对 <code>number</code> 变量做一百万次 <code>+1</code> 操作。
15 Python大法初体验：内置函数	<pre>import threading number = 0 def add(): for i in range(1000000): global number number += 1 t_1 = threading.Thread(target=add) t_2 = threading.Thread(target=add) t_1.start() t_2.start() t_1.join() t_2.join()</pre>
16 深入理解下迭代器和生成器	
17 生成器表达式和列表生成式	
18 把盒子升级为豪宅：函数进阶	
19 让你的模子更好用：类进阶	
20 从小独栋升级为别墅区：函数式编程	

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python？

02 我会怎样带你学 Python？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元组、字符串

14 这么多的数据结构（二）：字典、集合

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编程

`number` 的预期结果应该是 2000000（两百万）。将上述代码保存至文件 `thread_add.py` 中，来看下实际运行结果：

```
→ ~ python3 thread_add.py
1584627
→ ~ python3 thread_add.py
1413399
→ ~ python3 thread_add.py
1541521
```

可以看到，每次运行的结果并不一致，并且均小于 2000000。

这是因为，`number += 1` 其实是两个操作——首先获取 `number`，然后对获取到的值 `+1`。这两个操作并不是原子的（也就是说，这两个操作并不一定会被 CPU 连续执行，执行第一个操作时，CPU 有可能被中断去执行其它任务，之后又回到这里执行第二个操作）。这个例子中有一种可能情形是，执行到某一时刻时，第一个线程获取到 `number` 值为 100，紧接着第二次线程也获取到 `number` 值为 100，第一个线程在 100 的基础上 `+1` 并将 101 赋值给 `number`，第二线程也在 100 的基础上 `+1` 并将 101 赋值给 `number`，由于两个线程是并行运行的，它们彼此间并不知情，这样就浪费了一次 `+1` 操作，最终的 `number` 结果也会变小。

在这种情况下想要得到正确的结果，应该对 `number += 1` 操作加锁。如下：

```
import threading

number = 0
lock = threading.Lock()

def add():
    for i in range(1000000):
        global number

        lock.acquire()
        number += 1
        lock.release()

t_1 = threading.Thread(target=add)
t_2 = threading.Thread(target=add)
t_1.start()
t_2.start()
t_1.join()
t_2.join()

print(number)
```

更新 `thread_add.py` 文件，来看下运行结果：

```
→ ~ python3 thread_add.py
2000000
```

<div><div>← 慕课专栏</div><div>☰ 你的第一本Python基础入门书 / 24 让你的代码更灵活：进程和线程</div></div>	
目录	<div>→ ~ python3 thread_add.py</div> <div>2000000</div>
第 1 章 入门准备	
01 开篇词：你为什么要学 Python ？	可以看到，这次结果完全正确。但同时我们也能感受到，程序的执行速度变慢了，是的，锁会带来性能上的损耗，这就需要在正确性和性能间做取舍了。
02 我会怎样带你学 Python ？	
03 让 Python 在你的电脑上安家落户	<div>← 23 不简单的输入输出：IO 操作</div> <div>25 Python的影分身之术：虚拟环境 →</div>
04 如何运行 Python 代码 ？	
第 2 章 通用语言特性	精选留言 0
05 数据的名字和种类—变量和类型	欢迎在这里发表留言，作者筛选后可公开显示
06 一串数据怎么存—列表和字符串	
07 不只有一条路—分支和循环	
08 将代码放进盒子—函数	<div>!</div> <div>目前暂无任何讨论</div>
09 知错能改—错误处理、异常机制	
10 定制一个模子—类	
11 更大的代码盒子—模块和包	
12 练习—密码生成器	
第 3 章 Python 进阶语言特性	
13 这么多的数据结构（一）：列表、元祖、字符串	
14 这么多的数据结构（二）：字典、集合	
15 Python大法初体验：内置函数	
16 深入理解下迭代器和生成器	
17 生成器表达式和列表生成式	
18 把盒子升级为豪宅：函数进阶	
19 让你的模子更好用：类进阶	
20 从小独栋升级为别墅区：函数式编程	