

21 | Python并发编程之Futures

2019-06-26 景霄

Python核心技术与实战

[进入课程 >](#)



讲述：冯永吉

时长 09:45 大小 7.82M



你好，我是景霄。

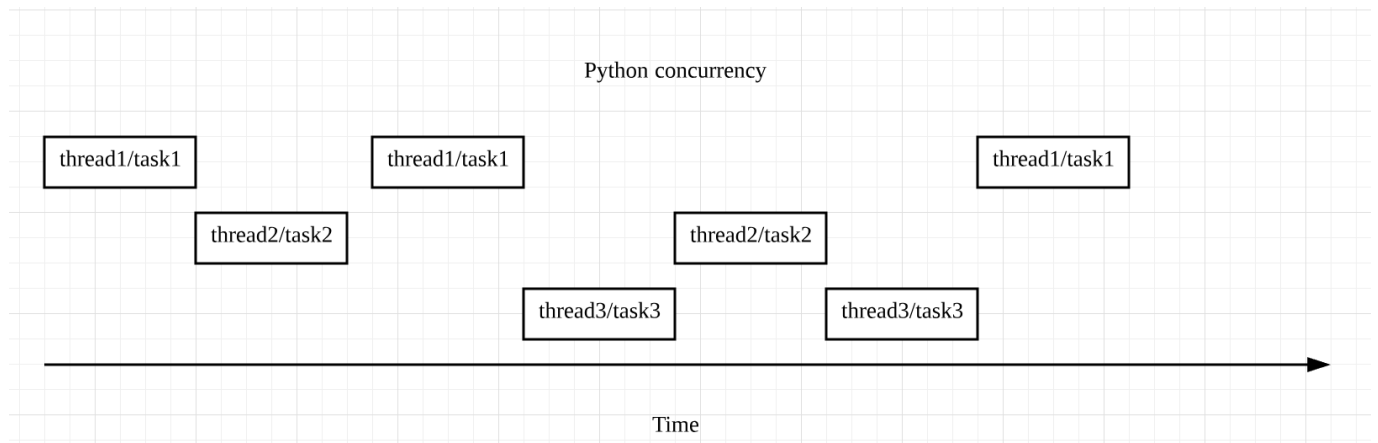
无论对于哪门语言，并发编程都是一项很常用很重要的技巧。比如我们上节课所讲的很常见的爬虫，就被广泛应用在工业界的各个领域。我们每天在各个网站、各个 App 上获取的新闻信息，很大一部分便是通过并发编程版的爬虫获得。

正确合理地使用并发编程，无疑会给我们的程序带来极大的性能提升。今天这节课，我就带你一起来学习理解、运用 Python 中的并发编程——Futures。

区分并发和并行

在我们学习并发编程时，常常同时听到并发（Concurrency）和并行（Parallelism）这两个术语，这两者经常一起使用，导致很多人以为它们是一个意思，其实不然。

首先你要辨别一个误区，在 Python 中，并发并不是指同一时刻有多个操作（thread、task）同时进行。相反，某个特定的时刻，它只允许有一个操作发生，只不过线程 / 任务之间会互相切换，直到完成。我们来看下面这张图：

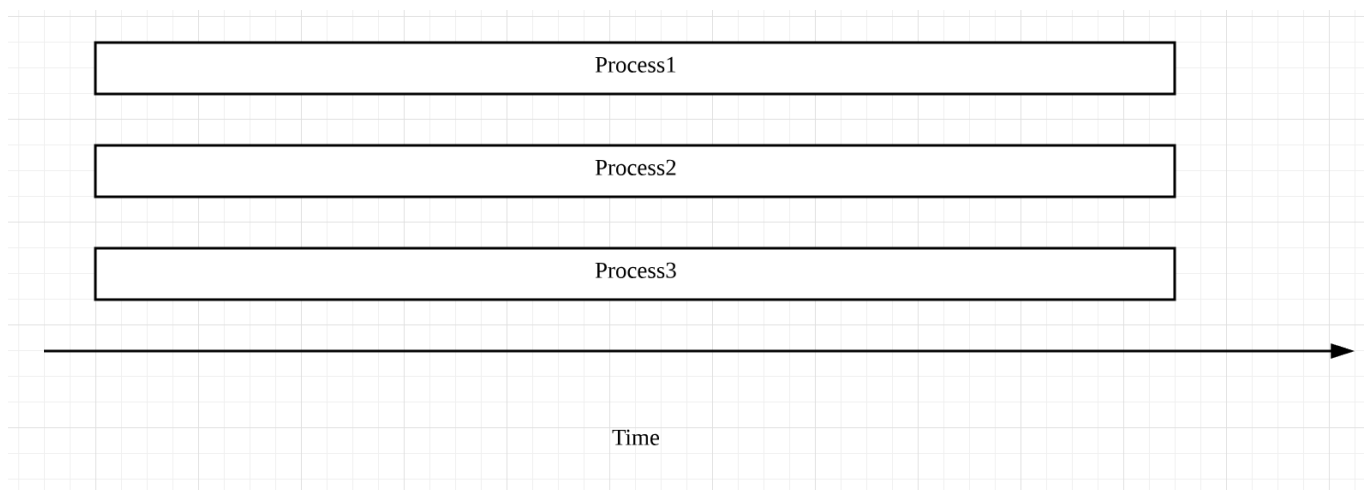


图中出现了 thread 和 task 两种切换顺序的不同方式，分别对应 Python 中并发的两种形式——threading 和 asyncio。

对于 threading，操作系统知道每个线程的所有信息，因此它会做主在适当的时候做线程切换。很显然，这样的好处是代码容易书写，因为程序员不需要做任何切换操作的处理；但是切换线程的操作，也有可能出现在一个语句执行的过程中（比如 $x += 1$ ），这样就容易出现 race condition 的情况。

而对于 asyncio，主程序想要切换任务时，必须得到此任务可以被切换的通知，这样一来也就可以避免刚刚提到的 race condition 的情况。

至于所谓的并行，指的才是同一时刻、同时发生。Python 中的 multi-processing 便是这个意思，对于 multi-processing，你可以简单地这么理解：比如你的电脑是 6 核处理器，那么在运行程序时，就可以强制 Python 开 6 个进程，同时执行，以加快运行速度，它的原理示意图如下：



对比来看，

并发通常应用于 I/O 操作频繁的场景，比如你要从网站上下载多个文件，I/O 操作的时间可能会比 CPU 运行处理的时间长得多。


而并行则更多应用于 CPU heavy 的场景，比如 MapReduce 中的并行计算，为了加快运行速度，一般会用多台机器、多个处理器来完成。

并发编程之 Futures

单线程与多线程性能比较

接下来，我们一起通过具体的实例，从代码的角度来理解并发编程中的 Futures，并进一步来比较其与单线程的性能区别。

假设我们有一个任务，是下载一些网站的内容并打印。如果用单线程的方式，它的代码实现如下所示（为了简化代码，突出主题，此处我忽略了异常处理）：

 复制代码

```
1 import requests
2 import time
3
4 def download_one(url):
5     resp = requests.get(url)
6     print('Read {} from {}'.format(len(resp.content), url))
7
8 def download_all(sites):
9     for site in sites:
10         download_one(site)
11
12 def main():
```

```

13     sites = [
14         'https://en.wikipedia.org/wiki/Portal:Arts',
15         'https://en.wikipedia.org/wiki/Portal:History',
16         'https://en.wikipedia.org/wiki/Portal:Society',
17         'https://en.wikipedia.org/wiki/Portal:Biography',
18         'https://en.wikipedia.org/wiki/Portal:Mathematics',
19         'https://en.wikipedia.org/wiki/Portal:Technology',
20         'https://en.wikipedia.org/wiki/Portal:Geography',
21         'https://en.wikipedia.org/wiki/Portal:Science',
22         'https://en.wikipedia.org/wiki/Computer_science',
23         'https://en.wikipedia.org/wiki/Python_(programming_language)',
24         'https://en.wikipedia.org/wiki/Java_(programming_language)',
25         'https://en.wikipedia.org/wiki/PHP',
26         'https://en.wikipedia.org/wiki/Node.js',
27         'https://en.wikipedia.org/wiki/The_C_Programming_Language',
28         'https://en.wikipedia.org/wiki/Go_(programming_language)'
29     ]
30     start_time = time.perf_counter()
31     download_all(sites)
32     end_time = time.perf_counter()
33     print('Download {} sites in {} seconds'.format(len(sites), end_time - start_time))
34
35 if __name__ == '__main__':
36     main()
37
38 # 输出
39 Read 129886 from https://en.wikipedia.org/wiki/Portal:Arts
40 Read 184343 from https://en.wikipedia.org/wiki/Portal:History
41 Read 224118 from https://en.wikipedia.org/wiki/Portal:Society
42 Read 107637 from https://en.wikipedia.org/wiki/Portal:Biography
43 Read 151021 from https://en.wikipedia.org/wiki/Portal:Mathematics
44 Read 157811 from https://en.wikipedia.org/wiki/Portal:Technology
45 Read 167923 from https://en.wikipedia.org/wiki/Portal:Geography
46 Read 93347 from https://en.wikipedia.org/wiki/Portal:Science
47 Read 321352 from https://en.wikipedia.org/wiki/Computer_science
48 Read 391905 from https://en.wikipedia.org/wiki/Python_(programming_language)
49 Read 321417 from https://en.wikipedia.org/wiki/Java_(programming_language)
50 Read 468461 from https://en.wikipedia.org/wiki/PHP
51 Read 180298 from https://en.wikipedia.org/wiki/Node.js
52 Read 56765 from https://en.wikipedia.org/wiki/The_C_Programming_Language
53 Read 324039 from https://en.wikipedia.org/wiki/Go_(programming_language)
54 Download 15 sites in 2.464231112999869 seconds

```

这种方式应该是最直接也最简单的：


先是遍历存储网站的列表；

然后对当前网站执行下载操作；

等到当前操作完成后，再对下一个网站进行同样的操作，一直到结束。

我们可以看到总共耗时约 2.4s。单线程的优点是简单明了，但是明显效率低下，因为上述程序的绝大多数时间，都浪费在了 I/O 等待上。程序每次对一个网站执行下载操作，都必须等到前一个网站下载完成后才能开始。如果放在实际生产环境中，我们需要下载的网站数量至少是以万为单位的，不难想象，这种方案根本行不通。

接着我们再来看，多线程版本的代码实现：


 复制代码

```
1 import concurrent.futures
2 import requests
3 import threading
4 import time
5
6 def download_one(url):
7     resp = requests.get(url)
8     print('Read {} from {}'.format(len(resp.content), url))
9
10
11 def download_all(sites):
12     with concurrent.futures.ThreadPoolExecutor(max_workers=5) as executor:
13         executor.map(download_one, sites)
14
15 def main():
16     sites = [
17         'https://en.wikipedia.org/wiki/Portal:Arts',
18         'https://en.wikipedia.org/wiki/Portal:History',
19         'https://en.wikipedia.org/wiki/Portal:Society',
20         'https://en.wikipedia.org/wiki/Portal:Biography',
21         'https://en.wikipedia.org/wiki/Portal:Mathematics',
22         'https://en.wikipedia.org/wiki/Portal:Technology',
23         'https://en.wikipedia.org/wiki/Portal:Geography',
24         'https://en.wikipedia.org/wiki/Portal:Science',
25         'https://en.wikipedia.org/wiki/Computer_science',
26         'https://en.wikipedia.org/wiki/Python_(programming_language)',
27         'https://en.wikipedia.org/wiki/Java_(programming_language)',
28         'https://en.wikipedia.org/wiki/PHP',
29         'https://en.wikipedia.org/wiki/Node.js',
30         'https://en.wikipedia.org/wiki/The_C_Programming_Language',
31         'https://en.wikipedia.org/wiki/Go_(programming_language)'
32     ]
33     start_time = time.perf_counter()
34     download_all(sites)
35     end_time = time.perf_counter()
36     print('Download {} sites in {} seconds'.format(len(sites), end_time - start_time))
37
```

```
38 if __name__ == '__main__':
39     main()
40
41 ## 输出
42 Read 151021 from https://en.wikipedia.org/wiki/Portal:Mathematics
43 Read 129886 from https://en.wikipedia.org/wiki/Portal:Arts
44 Read 107637 from https://en.wikipedia.org/wiki/Portal:Biography
45 Read 224118 from https://en.wikipedia.org/wiki/Portal:Society
46 Read 184343 from https://en.wikipedia.org/wiki/Portal:History
47 Read 167923 from https://en.wikipedia.org/wiki/Portal:Geography
48 Read 157811 from https://en.wikipedia.org/wiki/Portal:Technology
49 Read 91533 from https://en.wikipedia.org/wiki/Portal:Science
50 Read 321352 from https://en.wikipedia.org/wiki/Computer_science
51 Read 391905 from https://en.wikipedia.org/wiki/Python_(programming_language)
52 Read 180298 from https://en.wikipedia.org/wiki/Node.js
53 Read 56765 from https://en.wikipedia.org/wiki/The_C_Programming_Language
54 Read 468461 from https://en.wikipedia.org/wiki/PHP
55 Read 321417 from https://en.wikipedia.org/wiki/Java_(programming_language)
56 Read 324039 from https://en.wikipedia.org/wiki/Go_(programming_language)
57 Download 15 sites in 0.1993663580002023 seconds
```

非常明显，总耗时是 0.2s 左右，效率一下子提升了 10 倍多。

我们具体来看这段代码，它是多线程版本和单线程版的主要区别所在：

 复制代码

```
1 with concurrent.futures.ThreadPoolExecutor(max_workers=5) as executor:
2     executor.map(download_one, sites)
```


这里我们创建了一个线程池，总共有 5 个线程可以分配使用。executor.map() 与前面所讲的 Python 内置的 map() 函数类似，表示对 sites 中的每一个元素，并发地调用函数 download_one()。

顺便提一下，在 download_one() 函数中，我们使用的 requests.get() 方法是线程安全的（thread-safe），因此在多线程的环境下，它也可以安全使用，并不会出现 race condition 的情况。

另外，虽然线程的数量可以自己定义，但是线程数并不是越多越好，因为线程的创建、维护和删除也会有一定的开销。所以如果你设置的很大，反而可能会导致速度变慢。我们往往需

要根据实际的需求做一些测试，来寻找最优的线程数量。

当然，我们也可以用并行的方式去提高程序运行效率。你只需要在 `download_all()` 函数中，做出下面的变化即可：

 复制代码

```
1 with futures.ThreadPoolExecutor(workers) as executor
2 =>
3 with futures.ProcessPoolExecutor() as executor:
```

在需要修改的这部分代码中，函数 `ProcessPoolExecutor()` 表示创建进程池，使用多个进程并行的执行程序。不过，这里我们通常省略参数 `workers`，因为系统会自动返回 CPU 的数量作为可以调用的进程数。

我刚刚提到过，并行的方式一般用在 CPU heavy 的场景中，因为对于 I/O heavy 的操作，多数时间都会用于等待，相比于多线程，使用多进程并不会提升效率。反而很多时候，因为 CPU 数量的限制，会导致其执行效率不如多线程版本。

到底什么是 Futures ？

Python 中的 Futures 模块，位于 `concurrent.futures` 和 `asyncio` 中，它们都表示带有延迟的操作。Futures 会将处于等待状态的操作包裹起来放到队列中，这些操作的状态随时可以查询，当然，它们的结果或是异常，也能够在操作完成后被获取。


通常来说，作为用户，我们不用考虑如何去创建 Futures，这些 Futures 底层都会帮我们处理好。我们要做的，实际上是去 schedule 这些 Futures 的执行。

比如，Futures 中的 `Executor` 类，当我们执行 `executor.submit(func)` 时，它便会安排里面的 `func()` 函数执行，并返回创建好的 future 实例，以便你之后查询调用。

这里再介绍一些常用的函数。Futures 中的方法 `done()`，表示相对应的操作是否完成——`True` 表示完成，`False` 表示没有完成。不过，要注意，`done()` 是 non-blocking 的，会立即返回结果。相对应的 `add_done_callback(fn)`，则表示 Futures 完成后，相对应的参数函数 `fn`，会被通知并执行调用。

Futures 中还有一个重要的函数 `result()`，它表示当 future 完成后，返回其对应的结果或异常。而 `as_completed(fs)`，则是针对给定的 future 迭代器 `fs`，在其完成后，返回完成后的迭代器。


所以，上述例子也可以写成下面的形式：

 复制代码

```
1 import concurrent.futures
2 import requests
3 import time
4
5 def download_one(url):
6     resp = requests.get(url)
7     print('Read {} from {}'.format(len(resp.content), url))
8
9 def download_all(sites):
10     with concurrent.futures.ThreadPoolExecutor(max_workers=5) as executor:
11         to_do = []
12         for site in sites:
13             future = executor.submit(download_one, site)
14             to_do.append(future)
15
16         for future in concurrent.futures.as_completed(to_do):
17             future.result()
18
19 def main():
20     sites = [
21         'https://en.wikipedia.org/wiki/Portal:Arts',
22         'https://en.wikipedia.org/wiki/Portal:History',
23         'https://en.wikipedia.org/wiki/Portal:Society',
24         'https://en.wikipedia.org/wiki/Portal:Biography',
25         'https://en.wikipedia.org/wiki/Portal:Mathematics',
26         'https://en.wikipedia.org/wiki/Portal:Technology',
27         'https://en.wikipedia.org/wiki/Portal:Geography',
28         'https://en.wikipedia.org/wiki/Portal:Science',
29         'https://en.wikipedia.org/wiki/Computer_science',
30         'https://en.wikipedia.org/wiki/Python_(programming_language)',
31         'https://en.wikipedia.org/wiki/Java_(programming_language)',
32         'https://en.wikipedia.org/wiki/PHP',
33         'https://en.wikipedia.org/wiki/Node.js',
34         'https://en.wikipedia.org/wiki/The_C_Programming_Language',
35         'https://en.wikipedia.org/wiki/Go_(programming_language)'
36     ]
37     start_time = time.perf_counter()
38     download_all(sites)
39     end_time = time.perf_counter()
40     print('Download {} sites in {} seconds'.format(len(sites), end_time - start_time))
41
42 if __name__ == '__main__':
```



```
42     main()
43
44 # 输出
45 Read 129886 from https://en.wikipedia.org/wiki/Portal:Arts
46 Read 107634 from https://en.wikipedia.org/wiki/Portal:Biography
47 Read 224118 from https://en.wikipedia.org/wiki/Portal:Society
48 Read 158984 from https://en.wikipedia.org/wiki/Portal:Mathematics
49 Read 184343 from https://en.wikipedia.org/wiki/Portal:History
50 Read 157949 from https://en.wikipedia.org/wiki/Portal:Technology
51 Read 167923 from https://en.wikipedia.org/wiki/Portal:Geography
52 Read 94228 from https://en.wikipedia.org/wiki/Portal:Science
53 Read 391905 from https://en.wikipedia.org/wiki/Python_(programming_language)
54 Read 321352 from https://en.wikipedia.org/wiki/Computer_science
55 Read 180298 from https://en.wikipedia.org/wiki/Node.js
56 Read 321417 from https://en.wikipedia.org/wiki/Java_(programming_language)
57 Read 468421 from https://en.wikipedia.org/wiki/PHP
58 Read 56765 from https://en.wikipedia.org/wiki/The_C_Programming_Language
59 Read 324039 from https://en.wikipedia.org/wiki/Go_(programming_language)
60 Download 15 sites in 0.21698231499976828 seconds
```



这里，我们首先调用 `executor.submit()`，将下载每一个网站的内容都放进 `future` 队列 `to_do`，等待执行。然后是 `as_completed()` 函数，在 `future` 完成后，便输出结果。

不过，这里要注意，`future` 列表中每个 `future` 完成的顺序，和它在列表中的顺序并不一定完全一致。到底哪个先完成、哪个后完成，取决于系统的调度和每个 `future` 的执行时间。

为什么多线程每次只能有一个线程执行？

前面我说过，同一时刻，Python 主程序只允许有一个线程执行，所以 Python 的并发，是通过多线程的切换完成的。你可能会疑惑这到底是为什么呢？

这里我简单提一下全局解释器锁的概念，具体内容后面会讲到。

事实上，Python 的解释器并不是线程安全的，为了解决由此带来的 `race condition` 等问题，Python 便引入了全局解释器锁，也就是同一时刻，只允许一个线程执行。当然，在执行 I/O 操作时，如果一个线程被 `block` 了，全局解释器锁便会被释放，从而让另一个线程能够继续执行。

总结

这节课，我们首先学习了 Python 中并发和并行的概念与区别。

并发，通过线程和任务之间互相切换的方式实现，但同一时刻，只允许有一个线程或任务执行。

而并行，则是指多个进程完全同步同时的执行。

并发通常用于 I/O 操作频繁的场景，而并行则适用于 CPU heavy 的场景。

随后，我们通过下载网站内容的例子，比较了单线程和运用 Futures 的多线程版本的性能差异。显而易见，合理地运用多线程，能够极大地提高程序运行效率。

我们还一起学习了 Futures 的具体原理，介绍了一些常用函数比如 `done()`、`result()`、`as_completed()` 等的用法，并辅以实例加以理解。

要注意，Python 中之所以同一时刻只允许一个线程运行，其实是由于全局解释器锁的存在。但是对 I/O 操作而言，当其被 block 的时候，全局解释器锁便会被释放，使其他线程继续执行。

思考题

最后给你留一道思考题。你能否通过查阅相关文档，为今天所讲的这个下载网站内容的例子，加上合理的异常处理，让程序更加稳定健壮呢？欢迎在留言区写下你的思考和答案，也欢迎你把今天的内容分享给你的同事朋友，我们一起交流、一起进步。

Python 核心技术与实战

系统提升你的 Python 能力

景霄

Facebook 资深工程师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 20 | 揭秘 Python 协程

下一篇 22 | 并发编程之Asyncio

精选留言 (17)

写留言



KaitoShy

2019-06-26

思考题：

1. request.get 会触发：ConnectionError, TimeOut, HTTPError等，所有显示抛出的异常都是继承requests.exceptions.RequestException
2. executor.map(download_one, urls) 会触发concurrent.futures.TimeoutError
3. result() 会触发Timeout，CancelledError...

作者回复：回答的很对



7



SCAR

2019-06-26

future之与中文理解起来其实挺微妙，不过这与生活中大家熟知的期物在底层逻辑上是一致的，future英文词义中就有期货的意思，都是封存一个东西，平常你该干嘛就干嘛，可以不用去理会，在未来的某个时候去看结果就行，只是python中那个物是对象而已。而关键词是延迟，异步。

思考题：添加异常处理...



4



Fergus

2019-06-30

老师好，看到文中为了使用.as_complete()作的修改似乎做了重复的工作，我对比了使用.as_complete()和.submit()后直接result()，得到的是相同的结果。

-- 问1：这里所做的修改只是为了展示.as_complete的功能么？我查看了文档也没想明白。

-- 问2：.as_complete()可能会在什么场景下使用得比较多？...



1



Geek_5bb182

2019-06-27

老师你好，concurrent.futures 和 asyncio 中的Future 的区别是什么，在携程编程中

作者回复: 可以参考<https://stackoverflow.com/questions/29902908/what-is-the-difference-between-concurrent-futures-and-asyncio-futures>



1



Luke Zhang

2019-06-26

关于concurrent写过一篇学习笔记：

<https://www.zhangqibot.com/post/python-concurrent-futures/>

Python实现多线程/多进程，大家常常会用到标准库中的threading和multiprocessing模块。

但从Python3.2开始，标准库为我们提供了concurrent.futures模块，它提供了...



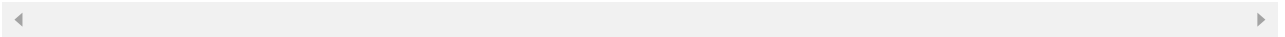
1



广州最优惠
2019-06-26

请问老师，future任务是调用submit后就开始执行，还是在调用as_completed之后才开始执行呢？

作者回复: submit之后



💬 2

👍 1



无才不肖生
2019-06-26

在submit()后只是放入队列而并未真正开始执行，as_completed时才真正去执行，对吗？as_completed会不会有个别future并执行完而没有输出结果，还是说就一定都会完成

💬

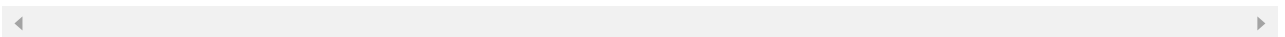
👍 1



简传宝
2019-06-27

老师好，请问是否可以理解为计算密集型任务用多进程，io密集型用多线程

作者回复: 没错。CPU-bound的任务主要是multi-processing，IO-bound的话，如果IO比较快，用多线程，如果IO比较慢，用asyncio，因为效率更加高



💬

👍

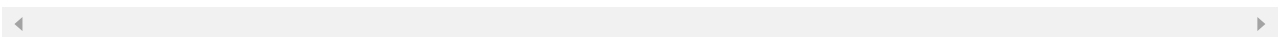


HelloWorld
2019-06-26

总结下并发和并行的概念：

并发，是指遇到I/O阻塞时（一般是网络I/O或磁盘I/O），通过多个线程之间切换执行多个任务（多线程）或单线程内多个任务之间切换执行的方式来最大化利用CPU时间，但同一时刻，只允许有一个线程或任务执行。适合I/O阻塞频繁的业务场景。...

作者回复: 没错



💬

👍



程序员人生

2019-06-26

根据老师的代码，我对download_one函数做了如下修改，

```
def download_one(url):
```

```
    try:
```

```
        resp = requests.get(url, timeout=2)
```

```
        print('Read {} from {}'.format(len(resp.content), url))...
```



阿西吧

2019-06-26

改成多进程后以下print为什么不输出了？

```
print('Read {} from {}'.format(len(resp.content), url))
```



阿西吧

2019-06-26

如果提示：SSLError: HTTPSConnectionPool(host='en.wikipedia.org', port=443): Max retries exceeded with url: /wiki/Portal:Arts (Caused by SSLError(SSLError("bad handshake: SysCallError(10054, 'WSAECONNRESET')")))

请先翻墙



广州最优惠

2019-06-26

```
try:
```

```
    data = future.result()
```

```
except Exception as exc:
```

```
    print('%r generated an exception: %s' % (url, exc))
```

```
else:...
```



_stuView

2019-06-26

老师，请问什么是线程安全，什么是race condition呢？

作者回复: 可以参考https://en.wikipedia.org/wiki/Thread_safety



tt

2019-06-26

学习笔记:和c++中std::future的比较。

std::future的主要目的是用来在线程间传递数据。消费者调用std::future.get获取数据时，如果数据没有准备好，则线程会阻塞直到std::future变为ready。注意，对std::future的访问也需要用锁来保护。...



Hoo-Ah

2019-06-26

加异常判断的话应该在获取result的时候加上。



大牛凯

2019-06-26

老师好，请问一下在python存在GIL的情况下，多进程是不是还是无法并发运行？谢谢老师

作者回复: 如果是多进程，则无所谓，可以并发运行。GIL是作用在线程上的，是不允许进程中的多线程同时运行

