

## 22 | 并发编程之Asyncio

2019-06-28 景霄

Python核心技术与实战

[进入课程 >](#)



讲述：冯永吉

时长 09:04 大小 7.28M



你好，我是景霄。

上节课，我们一起学习了 Python 并发编程的一种实现——多线程。今天这节课，我们继续学习 Python 并发编程的另一种实现方式——Asyncio。不同于协程那章，这节课我们更注重原理的理解。

通过上节课的学习，我们知道，在处理 I/O 操作时，使用多线程与普通的单线程相比，效率得到了极大的提高。你可能会想，既然这样，为什么还需要 Asyncio ？

诚然，多线程有诸多优点且应用广泛，但也存在一定的局限性：

比如，多线程运行过程容易被打断，因此有可能出现 race condition 的情况；

再如，线程切换本身存在一定的损耗，线程数不能无限增加，因此，如果你的 I/O 操作非常 heavy，多线程很有可能满足不了高效率、高质量的需求。

正是为了解决这些问题，Asyncio 应运而生。

## 什么是 Asyncio

### Sync VS Async

我们首先来区分一下 Sync（同步）和 Async（异步）的概念。

所谓 Sync，是指操作一个接一个地执行，下一个操作必须等上一个操作完成后才能执行。

而 Async 是指不同操作间可以相互交替执行，如果其中的某个操作被 block 了，程序并不会等待，而是会找出可执行的操作继续执行。

举个简单的例子，你的老板让你做一份这个季度的报表，并且邮件发给他。

如果按照 Sync 的方式，你会先向软件输入这个季度的各项数据，接下来等待 5min，等报表明细生成后，再写邮件发给他。

但如果按照 Async 的方式，再你输完这个季度的各项数据后，便会开始写邮件。等报表明细生成后，你会暂停邮件，先去查看报表，确认后继续写邮件直到发送完毕。

### Asyncio 工作原理

明白了 Sync 和 Async，回到我们今天的主题，到底什么是 Asyncio 呢？

事实上，Asyncio 和其他 Python 程序一样，是单线程的，它只有一个主线程，但是可以进行多个不同的任务（task），这里的任务，就是特殊的 future 对象。这些不同的任务，被一个叫做 event loop 的对象所控制。你可以把这里的任务，类比成多线程版本里的多个线程。

为了简化讲解这个问题，我们可以假设任务只有两个状态：一是预备状态；二是等待状态。所谓的预备状态，是指任务目前空闲，但随时待命准备运行。而等待状态，是指任务已经运行，但正在等待外部的操作完成，比如 I/O 操作。

在这种情况下，event loop 会维护两个任务列表，分别对应这两种状态；并且选取预备状态的一个任务（具体选取哪个任务，和其等待的时间长短、占用的资源等等相关），使其运行，一直到这个任务把控制权交还给 event loop 为止。

当任务把控制权交还给 event loop 时，event loop 会根据其是否完成，把任务放到预备或等待状态的列表，然后遍历等待状态列表的任务，查看他们是否完成。

如果完成，则将其放到预备状态的列表；

如果未完成，则继续放在等待状态的列表。

而原先在预备状态列表的任务位置仍旧不变，因为它们还未运行。

这样，当所有任务被重新放置在合适的列表后，新一轮的循环又开始了：event loop 继续从预备状态的列表中选取一个任务使其执行...如此周而复始，直到所有任务完成。

值得一提的是，对于 Asyncio 来说，它的任务在运行时不会被外部的一些因素打断，因此 Asyncio 内的操作不会出现 race condition 的情况，这样你就不需要担心线程安全的问题了。

## Asyncio 用法

讲完了 Asyncio 的原理，我们结合具体的代码来看一下它的用法。还是以上节课下载网站内容为例，用 Asyncio 的写法我放在了下面代码中（省略了异常处理的一些操作），接下来我们一起来看看：

 复制代码

```
1 import asyncio
2 import aiohttp
3 import time
4
5 async def download_one(url):
6     async with aiohttp.ClientSession() as session:
7         async with session.get(url) as resp:
8             print('Read {} from {}'.format(resp.content_length, url))
9
10 async def download_all(sites):
11     tasks = [asyncio.create_task(download_one(site)) for site in sites]
12     await asyncio.gather(*tasks)
13
14 def main():
```


```

15     sites = [
16         'https://en.wikipedia.org/wiki/Portal:Arts',
17         'https://en.wikipedia.org/wiki/Portal:History',
18         'https://en.wikipedia.org/wiki/Portal:Society',
19         'https://en.wikipedia.org/wiki/Portal:Biography',
20         'https://en.wikipedia.org/wiki/Portal:Mathematics',
21         'https://en.wikipedia.org/wiki/Portal:Technology',
22         'https://en.wikipedia.org/wiki/Portal:Geography',
23         'https://en.wikipedia.org/wiki/Portal:Science',
24         'https://en.wikipedia.org/wiki/Computer_science',
25         'https://en.wikipedia.org/wiki/Python_(programming_language)',
26         'https://en.wikipedia.org/wiki/Java_(programming_language)',
27         'https://en.wikipedia.org/wiki/PHP',
28         'https://en.wikipedia.org/wiki/Node.js',
29         'https://en.wikipedia.org/wiki/The_C_Programming_Language',
30         'https://en.wikipedia.org/wiki/Go_(programming_language)'
31     ]
32     start_time = time.perf_counter()
33     asyncio.run(download_all(sites))
34     end_time = time.perf_counter()
35     print('Download {} sites in {} seconds'.format(len(sites), end_time - start_time))
36
37 if __name__ == '__main__':
38     main()
39
40 ## 输出
41 Read 63153 from https://en.wikipedia.org/wiki/Java_(programming_language)
42 Read 31461 from https://en.wikipedia.org/wiki/Portal:Society
43 Read 23965 from https://en.wikipedia.org/wiki/Portal:Biography
44 Read 36312 from https://en.wikipedia.org/wiki/Portal:History
45 Read 25203 from https://en.wikipedia.org/wiki/Portal:Arts
46 Read 15160 from https://en.wikipedia.org/wiki/The_C_Programming_Language
47 Read 28749 from https://en.wikipedia.org/wiki/Portal:Mathematics
48 Read 29587 from https://en.wikipedia.org/wiki/Portal:Technology
49 Read 79318 from https://en.wikipedia.org/wiki/PHP
50 Read 30298 from https://en.wikipedia.org/wiki/Portal:Geography
51 Read 73914 from https://en.wikipedia.org/wiki/Python_(programming_language)
52 Read 62218 from https://en.wikipedia.org/wiki/Go_(programming_language)
53 Read 22318 from https://en.wikipedia.org/wiki/Portal:Science
54 Read 36800 from https://en.wikipedia.org/wiki/Node.js
55 Read 67028 from https://en.wikipedia.org/wiki/Computer_science
56 Download 15 sites in 0.062144195078872144 seconds

```


这里的 Async 和 await 关键字是 Asyncio 的最新写法，表示这个语句 / 函数是 non-block 的，正好对应前面所讲的 event loop 的概念。如果任务执行的过程需要等待，则将其放入等待状态的列表中，然后继续执行预备状态列表里的任务。

主函数里的 `asyncio.run(coro)` 是 Asyncio 的 root call，表示拿到 event loop，运行输入的 `coro`，直到它结束，最后关闭这个 event loop。事实上，`asyncio.run()` 是 Python3.7+ 才引入的，相当于老版本的以下语句：

 复制代码

```
1 loop = asyncio.get_event_loop()
2 try:
3     loop.run_until_complete(coro)
4 finally:
5     loop.close()
```

至于 Asyncio 版本的函数 `download_all()`，和之前多线程版本有很大的区别：

 复制代码

```
1 tasks = [asyncio.create_task(download_one(site)) for site in sites]
2 await asyncio.gather(*tasks)
```

这里的 `asyncio.create_task(coro)`，表示对输入的协程 `coro` 创建一个任务，安排它的执行，并返回此任务对象。这个函数也是 Python 3.7+ 新增的，如果是之前的版本，你可以用 `asyncio.ensure_future(coro)` 等效替代。可以看到，这里我们对每一个网站的下载，都创建了一个对应的任务。

再往下看，`asyncio.gather(*aws, loop=None, return_exception=False)`，则表示在 event loop 中运行 `aws` 序列的所有任务。当然，除了例子中用到的这几个函数，Asyncio 还提供了很多其他的用法，你可以查看 [相应文档](#) 进行了解。

最后，我们再来看一下最后的输出结果——用时只有 0.06s，效率比起之前的多线程版本，可以说是更上一层楼，充分体现其优势。

## Asyncio 有缺陷吗？

学了这么多内容，我们认识到了 Asyncio 的强大，但你要清楚，任何一种方案都不是完美的，都存在一定的局限性，Asyncio 同样如此。

实际工作中，想用好 Asyncio，特别是发挥其强大的功能，很多情况下必须得有相应的 Python 库支持。你可能注意到了，上节课的多线程编程中，我们使用的是 requests 库，但今天我们并没有使用，而是用了 aiohttp 库，原因就是 requests 库并不兼容 Asyncio，但是 aiohttp 库兼容。

Asyncio 软件库的兼容性问题，在 Python3 的早期一直是个大问题，但是随着技术的发展，这个问题正逐步得到解决。


另外，使用 Asyncio 时，因为你在任务的调度方面有了更大的自主权，写代码时就得更加注意，不然很容易出错。

举个例子，如果你需要 await 一系列的操作，就得使用 `asyncio.gather()`；如果只是单个的 future，或许只用 `asyncio.wait()` 就可以了。那么，对于你的 future，你是想要让它 `run_until_complete()` 还是 `run_forever()` 呢？诸如此类，都是你在面对具体问题时需要考虑的。

## 多线程还是 Asyncio

不知不觉，我们已经把并发编程的两种方式都给学习完了。不过，遇到实际问题时，多线程和 Asyncio 到底如何选择呢？

总的来说，你可以遵循以下伪代码的规范：

 复制代码

```
1 if io_bound:
2     if io_slow:
3         print('Use Asyncio')
4     else:
5         print('Use multi-threading')
6 else if cpu_bound:
7     print('Use multi-processing')
```

如果是 I/O bound，并且 I/O 操作很慢，需要很多任务 / 线程协同实现，那么使用 Asyncio 更合适。

如果是 I/O bound，但是 I/O 操作很快，只需要有限数量的任务 / 线程，那么使用多线程就可以了。



如果是 CPU bound，则需要使用多进程来提高程序运行效率。

## 总结

今天这节课，我们一起学习了 Asyncio 的原理和用法，并比较了 Asyncio 和多线程各自的优缺点。

不同于多线程，Asyncio 是单线程的，但其内部 event loop 的机制，可以让它并发地运行多个不同的任务，并且比多线程享有更大的自主控制权。

Asyncio 中的任务，在运行过程中不会被打断，因此不会出现 race condition 的情况。尤其是在 I/O 操作 heavy 的场景下，Asyncio 比多线程的运行效率更高。因为 Asyncio 内部任务切换的损耗，远比线程切换的损耗要小；并且 Asyncio 可以开启的任务数量，也比多线程中的线程数量多得多。

但需要注意的是，很多情况下，使用 Asyncio 需要特定第三方库的支持，比如前面示例中的 aiohttp。而如果 I/O 操作很快，并不 heavy，那么运用多线程，也能很有效地解决问题。

## 思考题

这两节课，我们学习了并发编程的两种实现方式，也多次提到了并行编程（multi-processing），其适用于 CPU heavy 的场景。

现在有这么一个需求：输入一个列表，对于列表中的每个元素，我想计算 0 到这个元素的所有整数的平方和。

我把常规版本的写法放在了下面，你能通过查阅资料，写出它的多进程版本，并且比较程序的耗时吗？

 复制代码

```
1 import time
2 def cpu_bound(number):
3     print(sum(i * i for i in range(number)))
4
5 def calculate_sums(numbers):
6     for number in numbers:
7         cpu_bound(number)
```

```
8
9 def main():
10     start_time = time.perf_counter()
11     numbers = [10000000 + x for x in range(20)]
12     calculate_sums(numbers)
13     end_time = time.perf_counter()
14     print('Calculation takes {} seconds'.format(end_time - start_time))
15
16 if __name__ == '__main__':
17     main()
```

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 21 | Python并发编程之Futures

精选留言 (17)

 写留言





**transformation**

2019-06-28

import time  
from concurrent import futures

def cpu\_bound(number):...



5



**HelloWorld**

2019-06-28

总结多线程和协程之间的共同点和区别：

共同点：

都是并发操作，多线程同一时间点只能有一个线程在执行，协程同一时间点只能有一个任务在执行；

不同点：...



4



**Geek\_59f23e**

2019-06-28

import time  
from multiprocessing import Pool

def square(number):...



1



**Destroy\_**

2019-06-28

race condition 是什么？



1



**天凉好个秋**

2019-06-28

如果完成，则将其放到预备状态的列表；  
如果未完成，则继续放在等待状态的列表。

这里是不是写的有问题？

PS:想问一下，完成之后为什么还要放队列里？难道不应该从队列里移除吗？

💬 1

👍 1



**TopolInside**

2019-06-29

老师，能否麻烦把音频的内容解释得再多一些，而不是现在的“我们来看下面这段代码”。网上的资源这么多，我之所以看重我们的专栏，实际上就是看重音频，可以通勤时间或者不方便看屏幕时间学习。这里推荐一下王争老师的数据结构与算法之美，对音频用户相对友好。这里一些小建议：1.音频内容涉及到代码的时候，尽量详细说出代码主要都做了什么，有哪些变量哪些函数哪些类等。2.可以在此基础上说明代码的逻辑，为何要这...

💬

👍



**SZC**

2019-06-29

能否举一些例子，哪些场景是IO密集型中的IOheavy，那些是IO很快

作者回复: 这个得看具体场景。比如大公司里相应业务爬虫的规模非常大，要抓取百万级的视频新闻信息流，这种就属于IO heavy。但是如果你只需要抓取10个网站的信息，并且网络连接良好，那么IO就很快



💬

👍



**响雨**

2019-06-29

```
from multiprocessing import Pool  
import time
```

```
def square(number):...
```

💬

👍



**hlz-123**

2019-06-28

1、单进程，老师的原程序，运行时间

Calculation takes 15.305913339 seconds

2、CPU并行方式，运行时间：

Calculation takes 3.457259904 seconds

```
def calculate_sums(numbers): ...
```



**HelloWorld**

2019-06-28

```
import concurrent.futures
import multiprocessing
import time
```

```
def cpu_bound(number):...
```



**Redevil**

2019-06-28

学到了：想启动多进程python程序

1. 不能用jupyter notebook
2. 必须使用if `__name__ == '__main__':` 方式



**广州最优惠**

2019-06-28

```
import time
import concurrent.futures
```

```
def cpu_bound(number):...
```



**程序员人生**

2019-06-28

多进程修改版本：

```
import time
import multiprocessing
```

...



**舒服**

2019-06-28

async 和 await有什么作用啊



**farFlight**

2019-06-28

请问在运行下载网站内容时出现以下问题怎么解决啊？

SSL error in data received

protocol: <asyncio.sslproto.SSLProtocol object at 0x7f3a241e4b00>

transport: <\_SelectorSocketTransport fd=58 read=polling write=<idle, bufsize=0>>



**方向**

2019-06-28

如果完成，则放到预备状态列表，这句话不理解。这样一来，预备状态列表同时拥有两种形式的任务啊



**大牛凯**

2019-06-28

老师好，想请问一下Asyncio和go function的区别是什么？感觉好像都是用户级线程...谢谢老师

