

## 20 | 揭秘 Python 协程

2019-06-24 景霄

Python核心技术与实战

[进入课程 >](#)



**讲述：冯永吉**

时长 10:50 大小 8.70M



你好，我是景霄。

上一节课的最后，我们留下一个小小的悬念：生成器在 Python 2 中还扮演了一个重要角色，就是用来实现 Python 协程。

那么首先你要明白，什么是协程？

协程是实现并发编程的一种方式。一说并发，你肯定想到了多线程 / 多进程模型，没错，多线程 / 多进程，正是解决并发问题的经典模型之一。最初的互联网世界，多线程 / 多进程在服务器并发中，起到举足轻重的作用。

随着互联网的快速发展，你逐渐遇到了 C10K 瓶颈，也就是同时连接到服务器的客户达到了一万个。于是很多代码跑崩了，进程上下文切换占用了大量的资源，线程也顶不住如此巨大的压力，这时，NGINX 带着事件循环出来拯救世界了。

如果将多进程 / 多线程类比为起源于唐朝的藩镇割据，那么事件循环，就是宋朝加强的中央集权制。事件循环启动一个统一的调度器，让调度器来决定一个时刻去运行哪个任务，于是省却了多线程中启动线程、管理线程、同步锁等各种开销。同一时期的 NGINX，在高并发下能保持低资源低消耗高性能，相比 Apache 也支持更多的并发连接。

再到后来，出现了一个很有名的名词，叫做回调地狱（callback hell），手撸过 JavaScript 的朋友肯定知道我在说什么。我们大家惊喜地发现，这种工具完美地继承了事件循环的优越性，同时还能提供 async / await 语法糖，解决

了执行性和可读性共存的难题。于是，协程逐渐被更多人发现并看好，也有越来越多的人尝试用 Node.js 做起了后端开发。（讲个笑话，JavaScript 是一门编程语言。）

回到我们的 Python。使用生成器，是 Python 2 开头的时代实现协程的老方法了，Python 3.7 提供了新的基于 `asyncio` 和 `async / await` 的方法。我们这节课，同样的，跟随时代，抛弃掉不容易理解、也不容易写的旧的基于生成器的方法，直接来讲新方法。

我们先从一个爬虫实例出发，用清晰的讲解思路，带你结合实战来搞懂这个不算特别容易理解的概念。之后，我们再由浅入深，直击协程的核心。

## 从一个爬虫说起

爬虫，就是互联网的蜘蛛，在搜索引擎诞生之时，与其一同来到世上。爬虫每秒钟都会爬取大量的网页，提取关键信息后存储在数据库中，以便日后分析。爬虫有非常简单的 Python 十行代码实现，也有 Google 那样的全球分布式爬虫的上百万行代码，分布在内部上万台服务器上，对全世界的信息进行嗅探。

话不多说，我们先看一个简单的爬虫例子：


```
1 import time
2
3 def crawl_page(url):
4     print('crawling {}'.format(url))
5     sleep_time = int(url.split('_')[-1])
6     time.sleep(sleep_time)
7     print('OK {}'.format(url))
8
9 def main(urls):
10     for url in urls:
11         crawl_page(url)
12
13 %time main(['url_1', 'url_2', 'url_3', 'url_4'])
14
15 ##### 输出 #####
16
17 crawling url_1
18 OK url_1
19 crawling url_2
20 OK url_2
21 crawling url_3
22 OK url_3
23 crawling url_4
24 OK url_4
25 Wall time: 10 s
```

(注意：本节的主要目的是协程的基础概念，因此我们简化爬虫的 `scrawl_page` 函数为休眠数秒，休眠时间取决于 url 最后的那个数字。)

这是一个很简单的爬虫，main() 函数执行时，调取 crawl\_page() 函数进行网络通信，经过若干秒等待后收到结果，然后执行下一个。

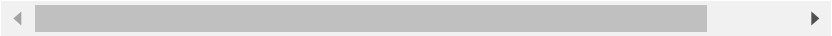
看起来很简单，但你仔细一算，它也占用了不少时间，五个页面分别用了 1 秒到 4 秒的时间，加起来一共用了 10 秒。这显然效率低下，该怎么优化呢？

于是，一个很简单的思路出现了——我们这种爬取操作，完全可以并发化。我们就来看看使用协程怎么写。

 复制代码

```
1 import asyncio
2
3 async def crawl_page(url):
4     print('crawling {}'.format(url))
5     sleep_time = int(url.split('_')[-1])
6     await asyncio.sleep(sleep_time)
7     print('OK {}'.format(url))
8
9 async def main(urls):
10     for url in urls:
11         await crawl_page(url)
12
13 %time asyncio.run(main(['url_1', 'url_2', 'url_3', 'url_4']))
14
15 ##### 输出 #####
16
17 crawling url_1
18 OK url_1
```

```
19 crawling url_2
20 OK url_2
21 crawling url_3
22 OK url_3
23 crawling url_4
24 OK url_4
25 Wall time: 10 s
```



看到这段代码，你应该发现了，在 Python 3.7 以上版本中，使用协程写异步程序非常简单。

首先来看 `import asyncio`，这个库包含了大部分我们实现协程所需的魔法工具。

`async` 修饰词声明异步函数，于是，这里的 `crawl_page` 和 `main` 都变成了异步函数。而调用异步函数，我们便可得到一个协程对象（coroutine object）。

举个例子，如果你 `print(crawl_page(''))`，便会输出 `<coroutine object crawl_page at 0x000002BEDF141148>`，提示你这是一个 Python 的协程对象，而并不会真正执行这个函数。

再来说说协程的执行。执行协程有多种方法，这里我介绍一下常用的三种。

首先，我们可以通过 `await` 来调用。`await` 执行的效果，和 Python 正常执行是一样的，也就是说程序会阻塞在这里，进入被调用的协程函数，执行完毕返回后再继续，而这也是 `await` 的字面意思。代码中 `await asyncio.sleep(sleep_time)` 会在这里休息若干秒，`await crawl_page(url)` 则会执行 `crawl_page()` 函数。

其次，我们可以通过 `asyncio.create_task()` 来创建任务，这个我们下节课会详细讲一下，你先简单知道即可。


最后，我们需要 `asyncio.run` 来触发运行。`asyncio.run` 这个函数是 Python 3.7 之后才有的特性，可以让 Python 的协程接口变得非常简单，你不用去理会事件循环怎么定义和怎么使用的问题（我们会在下面讲）。一个非常好的编程规范是，`asyncio.run(main())` 作为主程序的入口函数，在程序运行周期内，只调用一次 `asyncio.run`。

这样，你就大概看懂了协程是怎么用的吧。不妨试着跑一下代码，欸，怎么还是 10 秒？

10 秒就对了，还记得上面所说的，await 是同步调用，因此，crawl\_page(url) 在当前的调用结束之前，是不会触发下一次调用的。于是，这个代码效果就和上面完全一样了，相当于我们用异步接口写了个同步代码。

现在又该怎么办呢？

其实很简单，也正是我接下来要讲的协程中的一个重要概念，任务（Task）。老规矩，先看代码。

 复制代码

```
1 import asyncio
2
3 async def crawl_page(url):
4     print('crawling {}'.format(url))
5     sleep_time = int(url.split('_')[-1])
6     await asyncio.sleep(sleep_time)
7     print('OK {}'.format(url))
8
9 async def main(urls):
10     tasks = [asyncio.create_task(crawl_page(url)) for u
11     for task in tasks:
12         await task
13
14 %time asyncio.run(main(['url_1', 'url_2', 'url_3', 'url
15
16 ##### 输出 #####
17
18 crawling url_1
19 crawling url_2
```



```
20 crawling url_3
21 crawling url_4
22 OK url_1
23 OK url_2
24 OK url_3
25 OK url_4
26 Wall time: 3.99 s
```




你可以看到，我们有了协程对象后，便可以通过 `asyncio.create_task` 来创建任务。任务创建后很快就会被调度执行，这样，我们的代码也不会阻塞在任务这里。所以，我们要等所有任务都结束才行，用 `for task in tasks: await task` 即可。

这次，你就看到效果了吧，结果显示，运行总时长等于运行时间最长的爬虫。

当然，你也可以想一想，这里用多线程应该怎么写？而如果需要爬取的页面有上万个又该怎么办呢？再对比下协程的写法，谁更清晰自是一目了然。

其实，对于执行 `tasks`，还有另一种做法：

 复制代码

```
1 import asyncio
```


```
2
3 async def crawl_page(url):
4     print('crawling {}'.format(url))
5     sleep_time = int(url.split('_')[-1])
6     await asyncio.sleep(sleep_time)
7     print('OK {}'.format(url))
8
9 async def main(urls):
10     tasks = [asyncio.create_task(crawl_page(url)) for u
11     await asyncio.gather(*tasks)
12
13 %time asyncio.run(main(['url_1', 'url_2', 'url_3', 'url
14
15 ##### 输出 #####
16
17 crawling url_1
18 crawling url_2
19 crawling url_3
20 crawling url_4
21 OK url_1
22 OK url_2
23 OK url_3
24 OK url_4
25 Wall time: 4.01 s
```

这里的代码也很好理解。唯一要注意的是，`*tasks` 解包列表，将列表变成了函数的参数；与之对应的是，`** dict` 将字典变成了函数的参数。

另外, `asyncio.create_task`, `asyncio.run` 这些函数都是 Python 3.7 以上的版本才提供的, 自然, 相比于旧接口它们也更容易理解和阅读。

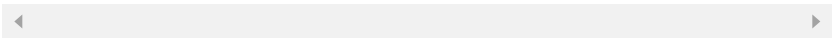
## 解密协程运行时


说了这么多, 现在, 我们不妨来深入代码底层看看。有了前面的知识做基础, 你应该很容易理解这两段代码。

 复制代码

```
1 import asyncio
2
3 async def worker_1():
4     print('worker_1 start')
5     await asyncio.sleep(1)
6     print('worker_1 done')
7
8 async def worker_2():
9     print('worker_2 start')
10    await asyncio.sleep(2)
11    print('worker_2 done')
12
13 async def main():
14    print('before await')
15    await worker_1()
16    print('awaited worker_1')
17    await worker_2()
18    print('awaited worker_2')
19
20 %time asyncio.run(main())
21
```

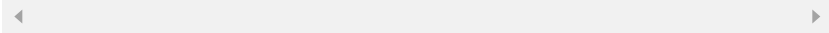
```
22 ##### 输出 #####
23
24 before await
25 worker_1 start
26 worker_1 done
27 awaited worker_1
28 worker_2 start
29 worker_2 done
30 awaited worker_2
31 Wall time: 3 s
```



 复制代码

```
1 import asyncio
2
3 async def worker_1():
4     print('worker_1 start')
5     await asyncio.sleep(1)
6     print('worker_1 done')
7
8 async def worker_2():
9     print('worker_2 start')
10    await asyncio.sleep(2)
11    print('worker_2 done')
12
13 async def main():
14     task1 = asyncio.create_task(worker_1())
15     task2 = asyncio.create_task(worker_2())
16     print('before await')
17     await task1
18     print('awaited worker_1')
19     await task2
20     print('awaited worker_2')
```

```
21
22 %time asyncio.run(main())
23
24 ##### 输出 #####
25
26 before await
27 worker_1 start
28 worker_2 start
29 worker_1 done
30 awaited worker_1
31 worker_2 done
32 awaited worker_2
33 Wall time: 2.01 s
```



不过，第二个代码，到底发生了什么呢？为了让你更详细了解到协程和线程的具体区别，这里我详细地分析了整个过程。步骤有点多，别着急，我们慢慢来看。

1. `asyncio.run(main())`，程序进入 `main()` 函数，事件循环开启；
2. `task1` 和 `task2` 任务被创建，并进入事件循环等待运行；运行到 `print`，输出 `'before await'`；
3. `await task1` 执行，用户选择从当前的主任务中切出，事件调度器开始调度 `worker_1`；
4. `worker_1` 开始运行，运行 `print` 输出 `'worker_1 start'`，然后运行到 `await asyncio.sleep(1)`，

- 从当前任务切出，事件调度器开始调度 `worker_2`;
5. `worker_2` 开始运行，运行 `print` 输出 `'worker_2 start'`，然后运行 `await asyncio.sleep(2)` 从当前任务切出；
  6. 以上所有事件的运行时间，都应该在 1ms 到 10ms 之间，甚至可能更短，事件调度器从这个时候开始暂停调度；
  7. 一秒钟后，`worker_1` 的 `sleep` 完成，事件调度器将控制权重新传给 `task_1`，输出 `'worker_1 done'`，`task_1` 完成任务，从事件循环中退出；
  8. `await task1` 完成，事件调度器将控制器传给主任务，输出 `'awaited worker_1'`，然后在 `await task2` 处继续等待；
  9. 两秒钟后，`worker_2` 的 `sleep` 完成，事件调度器将控制权重新传给 `task_2`，输出 `'worker_2 done'`，`task_2` 完成任务，从事件循环中退出；
  10. 主任务输出 `'awaited worker_2'`，协程全任务结束，事件循环结束。


接下来，我们进阶一下。如果我们想给某些协程任务限定运行时间，一旦超时就取消，又该怎么做呢？再进一步，如果某些协程运行时出现错误，又该怎么处理呢？同样的，来看代码。

```
1 import asyncio
2
3 async def worker_1():
4     await asyncio.sleep(1)
5     return 1
6
7 async def worker_2():
8     await asyncio.sleep(2)
9     return 2 / 0
10
11 async def worker_3():
12     await asyncio.sleep(3)
13     return 3
14
15 async def main():
16     task_1 = asyncio.create_task(worker_1())
17     task_2 = asyncio.create_task(worker_2())
18     task_3 = asyncio.create_task(worker_3())
19
20     await asyncio.sleep(2)
21     task_3.cancel()
22
23     res = await asyncio.gather(task_1, task_2, task_3,
24                               print(res))
25
26 %time asyncio.run(main())
27
28 ##### 输出 #####
29
30 [1, ZeroDivisionError('division by zero'), CancelledErr
31 Wall time: 2 s
```

你可以看到，worker\_1 正常运行，worker\_2 运行中出现错误，worker\_3 执行时间过长被我们 cancel 掉了，这些信息会全部体现在最终的返回结果 res 中。

不过要注意 `return_exceptions=True` 这行代码。如果不设置这个参数，错误就会完整地 throw 到我们这个执行层，从而需要 try except 来捕捉，这也就意味着其他还没被执行的任务会被全部取消掉。为了避免这个局面，我们将 `return_exceptions` 设置为 True 即可。

到这里，发现了没，线程能实现的，协程都能做到。那就让我们温习一下这些知识点，用协程来实现一个经典的生产者消费者模型吧。

 复制代码

```
1 import asyncio
2 import random
3
4 async def consumer(queue, id):
5     while True:
6         val = await queue.get()
7         print('{} get a val: {}'.format(id, val))
8         await asyncio.sleep(1)
9
10 async def producer(queue, id):
11     for i in range(5):
12         val = random.randint(1, 10)
13         await queue.put(val)
```



```
14         print('{} put a val: {}'.format(id, val))
15         await asyncio.sleep(1)
16
17 async def main():
18     queue = asyncio.Queue()
19
20     consumer_1 = asyncio.create_task(consumer(queue, 'c
21     consumer_2 = asyncio.create_task(consumer(queue, 'c
22
23     producer_1 = asyncio.create_task(producer(queue, 'p
24     producer_2 = asyncio.create_task(producer(queue, 'p
25
26     await asyncio.sleep(10)
27     consumer_1.cancel()
28     consumer_2.cancel()
29
30     await asyncio.gather(consumer_1, consumer_2, produc
31
32 %time asyncio.run(main())
33
34 ##### 输出 #####
35
36 producer_1 put a val: 5
37 producer_2 put a val: 3
38 consumer_1 get a val: 5
39 consumer_2 get a val: 3
40 producer_1 put a val: 1
41 producer_2 put a val: 3
42 consumer_2 get a val: 1
43 consumer_1 get a val: 3
44 producer_1 put a val: 6
45 producer_2 put a val: 10
46 consumer_1 get a val: 6
47 consumer_2 get a val: 10
48 producer_1 put a val: 4
```

```
49 producer_2 put a val: 5
50 consumer_2 get a val: 4
51 consumer_1 get a val: 5
52 producer_1 put a val: 2
53 producer_2 put a val: 8
54 consumer_1 get a val: 2
55 consumer_2 get a val: 8
56 Wall time: 10 s
```



## 实战：豆瓣近日推荐电影爬虫


最后，进入今天的实战环节——实现一个完整的协程爬虫。

任务描述：

<https://movie.douban.com/cinema/later/beijing/> 这个页面描述了北京最近上映的电影，你能否通过 Python 得到这些电影的名称、上映时间和海报呢？这个页面的海报是缩小版的，我希望能从具体的电影描述页面中抓取到海报。

听起来难度不是很大吧？我在下面给出了同步版本的代码和协程版本的代码，通过运行时间和代码写法的对比，希望你能对协程有更深入的了解。（注意：为了突出重点、简化代码，这里我省略了异常处理。）


不过，在参考我给出的代码之前，你是不是可以自己先动手写一下、跑一下呢？

 复制代码

```
1 import requests
2 from bs4 import BeautifulSoup
3
4 def main():
5     url = "https://movie.douban.com/cinema/later/beijir
6     init_page = requests.get(url).content
7     init_soup = BeautifulSoup(init_page, 'lxml')
8
9     all_movies = init_soup.find('div', id="showing-soor
10    for each_movie in all_movies.find_all('div', class_
11        all_a_tag = each_movie.find_all('a')
12        all_li_tag = each_movie.find_all('li')
13
14        movie_name = all_a_tag[1].text
15        url_to_fetch = all_a_tag[1]['href']
16        movie_date = all_li_tag[0].text
17
18        response_item = requests.get(url_to_fetch).cont
19        soup_item = BeautifulSoup(response_item, 'lxml'
20        img_tag = soup_item.find('img')
21
22        print('{} {} {}'.format(movie_name, movie_date,
23
24 %time main()
25
26 ##### 输出 #####
27
28 阿拉丁 05 月 24 日 https://img3.doubanio.com/view/photo/
29 龙珠超：布罗利 05 月 24 日 https://img3.doubanio.com/vie
```

```
30 五月天人生有限公司 05 月 24 日 https://img3.doubanio.com/
31 ... ...
32 直播攻略 06 月 04 日 https://img3.doubanio.com/view/phot
33 Wall time: 56.6 s
```



 复制代码

```
1 import asyncio
2 import aiohttp
3
4 from bs4 import BeautifulSoup
5
6 async def fetch_content(url):
7     async with aiohttp.ClientSession(
8         headers=header, connector=aiohttp.TCPConnector(
9     ) as session:
10         async with session.get(url) as response:
11             return await response.text()
12
13 async def main():
14     url = "https://movie.douban.com/cinema/later/beijir
15     init_page = await fetch_content(url)
16     init_soup = BeautifulSoup(init_page, 'lxml')
17
18     movie_names, urls_to_fetch, movie_dates = [], [], [
19
20     all_movies = init_soup.find('div', id="showing-soor
21     for each_movie in all_movies.find_all('div', class_
22         all_a_tag = each_movie.find_all('a')
23         all_li_tag = each_movie.find_all('li')
24
25         movie_names.append(all_a_tag[1].text)
26         urls_to_fetch.append(all_a_tag[1]['href'])
```

```

27         movie_dates.append(all_li_tag[0].text)
28
29     tasks = [fetch_content(url) for url in urls_to_fetch]
30     pages = await asyncio.gather(*tasks)
31
32     for movie_name, movie_date, page in zip(movie_names,
33                                             soup_items,
34                                             pages):
35         soup_item = BeautifulSoup(page, 'lxml')
36         img_tag = soup_item.find('img')
37
38         print('{} {} {}'.format(movie_name, movie_date,
39                                 img_tag.get('src')))
40
41 %time asyncio.run(main())
42 ##### 输出 #####
43
44 阿拉丁 05 月 24 日 https://img3.doubanio.com/view/photo/
45 龙珠超：布罗利 05 月 24 日 https://img3.doubanio.com/view/photo/
46 五月天人生无限公司 05 月 24 日 https://img3.doubanio.com/view/photo/
47 ... ...
48 直播攻略 06 月 04 日 https://img3.doubanio.com/view/photo/
49 Wall time: 4.98 s

```



## 总结

到这里，今天的主要内容就讲完了。今天我用了较长的篇幅，从一个简单的爬虫开始，到一个真正的爬虫结束，在中间穿插讲解了 Python 协程最新的基本概念和用法。这里带你简单复习一下。

协程和多线程的区别，主要在于两点，一是协程为单线程；二是协程由用户决定，在哪些地方交出控制权，切换到下一个任务。

协程的写法更加简洁清晰，把 `async / await` 语法和 `create_task` 结合来用，对于中小级别的并发需求已经毫无压力。

写协程程序的时候，你的脑海中要有清晰的事件循环概念，知道程序在什么时候需要暂停、等待 I/O，什么时候需要一并执行到底。

最后的最后，请一定不要轻易炫技。多线程模型也一定有其优点，一个真正牛逼的程序员，应该懂得，在什么时候用什么模型能达到工程上的最优，而不是自觉某个技术非常牛逼，所有项目创造条件也要上。技术是工程，而工程则是时间、资源、人力等纷繁复杂的事情的折衷。

## 思考题

最后给你留一个思考题。协程怎么实现回调函数呢？欢迎留言和我讨论，也欢迎你把这篇文章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 19 | 深入理解迭代器和生成器

## 精选留言 (15)

写留言



程序员人生

2019-06-24

关于思考题，这样写不知道对不对？

```
import asyncio
```

```
async def computer(a,b,func):
```

```
res = func(a,b)...
```



**佛系学python**

2019-06-24

老师，你那代码我一个都运行不出来。。%time那里会报错



**佛系学python**

2019-06-24

老师，课程的代码是基于py3的吗？



**enjoylearning**

2019-06-24

并发一直想用future executor 来着，不知道这两个有什么区别没



**csn**

2019-06-24

为什么在spyder下运行协程报错，



提示 " File

"C:\ProgramData\Anaconda3\lib\asyncio\runners.py"  
line 34, in run

"asyncio.run() cannot be called from a running...



**catshitfive**

2019-06-24

用adaconda升级了到python version 3.7.3,发现新的api  
asyncio.run()在调用的时候报错: asyncio.run() cannot  
be called from a running event loop,在网上看了下解  
决方法(不清楚为什么这么做):pip thirdparty模块:nest-  
asyncio,然后调用其模块内函数apply就可以正常使用了



**tt**

2019-06-24

学习笔记:异步和阻塞。

阻塞主要是同步编程中的概念:执行一个系统调用,如果暂  
时没有返回结果,这个调用就不会返回,那这个系统调用  
后面的应用代码也不会执行,整个应用被“阻塞”了。...





阿西吧

2019-06-24

这个代码是不是只有在linux环境才能运行成功



阿西吧

2019-06-24

%time 是什么语法糖，直接输出时间，要怎么写？



阿西吧

2019-06-24

协程的第一个例子，一运行就报错：

RuntimeError: asyncio.run() cannot be called from a running event loop

win7 64位 python3.7.3



cotter

2019-06-24

受教了，第一次听说这个高级功能！

我在工作中遇到一个需要并发的問題，用python在后台并发执行shell ,并发数量用时间范围控制，要不停的改时

间分多次串行，方法比较笨拙。协程可以简化我的代码。  
老师，并发很多事件应该也是需要消耗很多资源，协程...



李

2019-06-24

代码中我敲的time那一块一直报invalid syntax?  
`%time asyncio.run`



zx钟

2019-06-24

官方文档看了好久 还是一脸懵逼。



Hoo-Ah

2019-06-24

使用asyncio获取事件循环，将执行的函数使用loop创建一个任务。`add_done_callback`将回调函数传进去。



敏杰

2019-06-24

创建loop事件循环asyncio.get\_event\_loop

