Web Crawler Example

This lesson discusses a text-book example of a concurrent program - the web crawler.

Web Crawler Example

From what we have learnt so far, asyncio is an excellent choice for blocking operations. Usually, there are two kinds of blocking operations:

- network I/O
- disk I/O

Let's start with implementing a simple web crawler. A web crawler is a program that systematically browses the world wide web, typically with the intent to index it. For our purposes, we'll dumb down our crawler and limit its capability to fetch the HTML for a list of URLs. The downloaded HTML is passed onto a consumer which then performs the indexing but we'll not implement that part.

The meat of the problem lies in asynchronously downloading the given URLs. We'll be using the **`aiohttp`** module for asynchronous REST GET calls. If we were to serially download the URLs, we'll unnecessarily be wasting CPU cycles as a network request is usually the slowest form of I/O. Let's start with the serial version of the code.

Serial Download

In our example, we'll pass three URLs to be downloaded. The serial version of the code uses the following snippet to make the network requests:

```
    # url is a string such as www.cnn.com
    html = urlopen(url)
    text = html.read()
```

The complete code appears in the code widget below. Run the code and observe the total time taken to download the URLs in the output.

```
import time
from urllib.request import urlopen


def get_urls_to_crawl():
    urls_list = list()
    urls_list.append('http://www.cnn.com/')
    urls_list.append('https://www.foxnews.com/')
    urls_list.append('https://www.bbc.com/')
    urls_list.append('https://www.dawn.com')
    urls_list.append('https://www.cnbc.com')
    urls_list.append('https://www.twitter.com')

    return urls_list


if __name__ == "__main__":

    urls_to_crawl = get_urls_to_crawl()
    start = time.time()

    for url in get_urls_to_crawl():
        html = urlopen(url)
        text = html.read()

    elapsed = time.time() - start
    print("\n{} URLS downloaded in {:.2f}s".format(len(urls_to_crawl), elapsed))
```




Asynchronous Download

Next we'll code an asynchronous version of the crawler. The snippet of code that does the actual download is shown below:

```
async def crawl_one_url(url, session):

    # make a HTTP GET request for the URL. Note that
    # get_request is a coroutine object
    get_request = session.get(url)

    # await for the request to complete
    res = await get_request

    # await to read the response payload
    txt = await res.text()

    # remember to clean up
    get_request.close()

    return txt
```

The trick is to create a coroutine object for each URL and then submit all the coroutines to the event loop for execution as follows:

```
async def crawl_urls(urls_to_crawl):
    # create a single session to download all URLs
    session = aiohttp.ClientSession()

    # create a coroutine
    work_to_do = list()
    for url in urls_to_crawl:
        work_to_do.append(crawl_one_url(url, session))

    # res variable will contain the HTML for each URL
    res = await asyncio.gather(*work_to_do)

    # remember to clean up
    await session.close()
    return res
```

Complete Code Python 3.7+

The complete code for Python 3.7+ appears below. The code widget runs Python 3.5 and therefore can't execute the newer code:

```
import asyncio
import aiohttp
import time


async def crawl_one_url(url, session):
    get_request = session.get(url)

    res = await get_request
    txt = await res.text()

    get_request.close()

    return txt


async def crawl_urls(urls_to_crawl):
    session = aiohttp.ClientSession()

    work_to_do = list()
    for url in urls_to_crawl:
        work_to_do.append(crawl_one_url(url, session))

    res = await asyncio.gather(*work_to_do)

    await session.close()
    return res


def main():
    t0 = time.time()

    urls_to_crawl = get_urls_to_crawl()

    asyncio.run(crawl_urls(urls_to_crawl))
    elapsed = time.time() - t0
    print("\n{} URLS downloaded in {:.2f}s".format(len(urls_to
_crawl), elapsed))
```

```
def get_urls_to_crawl():
    urls_list = list()
    urls_list.append('http://www.cnn.com/')
    urls_list.append('https://www.foxnews.com/')
    urls_list.append('https://www.bbc.com/')
    urls_list.append('https://www.dawn.com')
    urls_list.append('https://www.cnbc.com')
    urls_list.append('https://www.twitter.com')

    return urls_list


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

Generator-based Coroutines

For the sake of completeness, we also present the crawler implementation using generator-based coroutines. The complete code appears in the widget below:

```
import asyncio
import aiohttp
import time


@asyncio.coroutine
def crawl_one_url(url, session):
    get_request = session.get(url)

    res = yield from get_request
    txt = yield from res.text()

    get_request.close()
    return txt


@asyncio.coroutine
def crawl_urls(urls_to_crawl):
    session = aiohttp.ClientSession()

    work_to_do = list()
    for url in urls_to_crawl:
        work_to_do.append(crawl_one_url(url, session))

    completed, pending = yield from asyncio.wait(work_to_do, return_when=asyncio.AL

    # uncomment to retrieve the downloaded HTML
    # for task in completed:
    #     print(task.result())

    # remember to clean up
    yield from session.close()


def main():
    urls_to_crawl = get_urls_to_crawl()
    start = time.time()

    loop = asyncio.get_event_loop()
    loop.run_until_complete(crawl_urls(urls_to_crawl))
    elapsed = time.time() - start
    print("\n{} URLS downloaded in {:.2f}s".format(len(urls_to_crawl), elapsed))


def get_urls_to_crawl():
    urls_list = list()
    urls_list.append('http://www.cnn.com/')
    urls_list.append('https://www.foxnews.com/')
    urls_list.append('https://www.bbc.com/')
    urls_list.append('https://www.dawn.com')
    urls_list.append('https://www.cnbc.com')
```

```
    urls_list.append('https://www.twitter.com')


    return urls_list


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

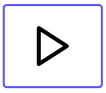
Multithreaded Implementation

We can also convert our serial implementation into a multithreaded one. The trick is to assign one thread to download each URL. The loop from our serial implementation would change as follows:

```
    threads = list()
    for url in get_urls_to_crawl():
        threads.append(Thread(target=crawl_one_url, args=(url
,)))
```

The complete implementation appears in the code widget below.

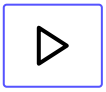

```
import time
from urllib.request import urlopen
from threading import Thread


def get_urls_to_crawl():
    urls_list = list()
    urls_list.append('http://www.cnn.com/')
    urls_list.append('https://www.foxnews.com/')
    urls_list.append('https://www.bbc.com/')
    urls_list.append('https://www.dawn.com')
    urls_list.append('https://www.cnbc.com')
    urls_list.append('https://www.twitter.com')
    return urls_list


def crawl_one_url(url):
    html = urlopen(url)
    text = html.read()


if __name__ == "__main__":

    urls_to_crawl = get_urls_to_crawl()
    start = time.time()

    threads = list()
    for url in get_urls_to_crawl():
        threads.append(Thread(target=crawl_one_url, args=(url,)))

    for thread in threads:
        thread.start()

    for thread in threads:
        thread.join()

    elapsed = time.time() - start
    print("\n{} URLS downloaded in {:.2f}s".format(len(urls_to_crawl), elapsed))
```

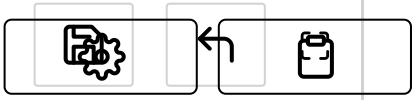



Multiprocess Implementation

Last but not the least, we can also write a program that delegates downloading URLs to different processes instead of threads by using the **`multiprocessing`** module. Notice how similar the multiprocess and multithreaded code snippets are. The APIs in the two modules mirror each other and converting from one model to the other becomes trivial.

```
import time
from urllib.request import urlopen
from multiprocessing import Process


def get_urls_to_crawl():
    urls_list = list()
    urls_list.append('http://www.cnn.com/')
    urls_list.append('https://www.foxnews.com/')
    urls_list.append('https://www.bbc.com/')
    urls_list.append('https://www.dawn.com')
    urls_list.append('https://www.cnbc.com')
    urls_list.append('https://www.twitter.com')
    return urls_list


def crawl_one_url(url):
    html = urlopen(url)
    text = html.read()


if __name__ == "__main__":

    urls_to_crawl = get_urls_to_crawl()
    start = time.time()

    processes = list()
    for url in get_urls_to_crawl():
        processes.append(Process(target=crawl_one_url, args=(url,)))

    for process in processes:
        process.start()

    for process in processes:
        process.join()

    elapsed = time.time() - start
    print("\n{} URLS downloaded in {:.2f}s".format(len(urls_to_crawl), elapsed))
```





Back

Next

Future & Tasks

Async Sleep Problem

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/web-crawler-example__asyncio__python-concurrency-for-senior-engineering-interviews)