并发下载

多线程和多进程回顾

在前面的《进程和线程》一文中，我们已经对在Python中使用多进程和多线程实现并发编程进行了简明的讲解，在此我们补充几个知识点。

threading.local类

使用线程时最不愿意遇到的情况就是多个线程竞争资源，在这种情况下为了保证资源状态的正确性，我们可能需要对资源进行加锁保护的处理，这一方面会导致程序失去并发性，另外如果多个线程竞争多个资源时，还有可能因为加锁方式的不当导致死锁。要解决多个线程竞争资源的问题，其中一个方案就是让每个线程都持有资源的副本（拷贝），这样每个线程可以操作自己所持有的资源，从而规避对资源的竞争。

要实现将资源和持有资源的线程进行绑定的操作，最简单的做法就是使用threading模块的local类，在网络爬虫开发中，就可以使用local类为每个线程绑定一个MySQL数据库连接或Redis客户端对象，这样通过线程可以直接获得这些资源，既解决了资源竞争的问题，又避免了在函数和方法调用时传递这些资源。具体的请参考本章多线程爬取“手机搜狐网”（Redis版）的实例代码。

concurrent.futures模块

Python3.2带来了`concurrent.futures`模块，这个模块包含了线程池和进程池、管理并行编程任务、处理非确定性的执行流程、进程/线程同步等功能。关于这部分的内容推荐大家阅读《Python并行编程》。

分布式进程

使用多进程的时候，可以将进程部署在多个主机节点上，Python的`multiprocessing`模块不但支持多进程，其中`managers`子模块还支持把多进程部署到多个节点上。当然，要部署分布式进程，首先需要一个服务进程作为调度者，进程之间通过网络进行通信来实现对进程的控制和调度，由于`managers`模块已经对这些做出了很好的封装，因此在无需了解网络通信细节的前提下，就可以编写分布式多进程应用。具体的请参照本章分布式多进程爬取“手机搜狐网”的实例代码。

协程和异步I/O

协程的概念

协程（coroutine）通常又称之为微线程或纤程，它是相互协作的一组子程序（函数）。所谓相互协作指的是在执行函数A时，可以随时中断去执行函数B，然后又中断继续执行函数A。注意，这一过程并不是函数调用（因为没有调用语句），整个过程看似像多线程，然而协程只有一个线程执行。协程通过`yield`关键字和`send()`操作来转移执行权，协程之间不是调用者与被调用者的关系。

协程的优势在于以下两点：

1. 执行效率极高，因为子程序（函数）切换不是线程切换，由程序自身控制，没有切换线程的开销。
2. 不需要多线程的锁机制，因为只有一个线程，也不存在竞争资源的问题，当然也就不需要对资源加锁保护，因此执行效率高很多。

> 说明：协程适合处理的是I/O密集型任务，处理CPU密集型任务并不是它的长处，如果要提升CPU的利用率可以考虑“多进程+协程”的模式。

历史回顾

1. Python 2.2：第一次提出了生成器（最初称之为迭代器）的概念（PEP 255）。
2. Python 2.5：引入了将对象发送回暂停了的生成器这一特性即生成器的`send()`方法（PEP 342）。
3. Python 3.3：添加了`yield from`特性，允许从迭代器中返回任何值（注意生成器本身也是迭代器），这样我们就可以串联生成器并且重构出更好的生成器。
4. Python 3.4：引入`asyncio.coroutine`装饰器用来标记作为协程的函数，协程函数和`asyncio`及其事件循环一起使用，来实现异步I/O操作。
5. Python 3.5：引入了`async`和`await`，可以使用`async def`来定义一个协程函数，这个函数中不能包含任何形式的`yield`语句，但是可以使用`return`或`await`从协程中返回值。

示例代码

1. 生成器 - 数据的生产者。

```
from time import sleep


# 倒计数生成器
def countdown(n):
    while n > 0:
        yield n
        n -= 1


def main():
    for num in countdown(5):
        print(f'Countdown: {num}')
        sleep(1)
    print('Countdown Over!')


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

生成器还可以叠加来组成生成器管道，代码如下所示。

```
# Fibonacci数生成器
def fib():
    a, b = 0, 1
    while True:
        a, b = b, a + b
        yield a


# 偶数生成器
def even(gen):
    for val in gen:
        if val % 2 == 0:
            yield val


def main():
    gen = even(fib())
    for _ in range(10):
        print(next(gen))
```

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

2. 协程 - 数据的消费者。

```
from time import sleep


# 生成器 - 数据生产者
def countdown_gen(n, consumer):
    consumer.send(None)
    while n > 0:
        consumer.send(n)
        n -= 1
    consumer.send(None)


# 协程 - 数据消费者
def countdown_con():
    while True:
        n = yield
        if n:
            print(f'Countdown {n}')
            sleep(1)
        else:
            print('Countdown Over!')


def main():
    countdown_gen(5, countdown_con())


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

> 说明：上面代码中countdown_gen函数中的第1行consumer.send(None)是为了激活生成器，通俗的说就是让生成器执行到有yield关键字的地方挂起，当然也可以通过next(consumer)来达到同样的效果。如果不愿意每次都用这样的代码来“预激”生成器，可以写一个包装器来完成该操作，代码如下所示。

```
from functools import wraps


def coroutine(fn):

    @wraps(fn)
    def wrapper(*args, **kwargs):
        gen = fn(*args, **kwargs)
        next(gen)
        return gen

    return wrapper
```

这样就可以使用`@coroutine`装饰器对协程进行预激操作，不需要再写重复代码来激活协程。

3. 异步I/O - 非阻塞式I/O操作。

```
import asyncio


@asyncio.coroutine
def countdown(name, n):
    while n > 0:
        print(f'Countdown[{name}]: {n}')
        yield from asyncio.sleep(1)
        n -= 1


def main():
    loop = asyncio.get_event_loop()
    tasks = [
        countdown("A", 10), countdown("B", 5),
    ]
    loop.run_until_complete(asyncio.wait(tasks))
    loop.close()


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

4. `async`和`await`。

```
import asyncio
import aiohttp


async def download(url):
    print('Fetch:', url)
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as resp:
            print(url, '--->', resp.status)
            print(url, '--->', resp.cookies)
            print('\n\n', await resp.text())


def main():
    loop = asyncio.get_event_loop()
    urls = [
        'https://www.baidu.com',
        'http://www.sohu.com/',
        'http://www.sina.com.cn/',
        'https://www.taobao.com/',
        'https://www.jd.com/'
    ]
    tasks = [download(url) for url in urls]
    loop.run_until_complete(asyncio.wait(tasks))
    loop.close()


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

上面的代码使用了AIOHTTP这个非常著名的第三方库，它实现了HTTP客户端和HTTP服务器的功能，对异步操作提供了非常好的支持，有兴趣可以阅读它的官方文档。

实例 - 多线程爬取“手机搜狐网”所有页面

下面我们把之间讲的所有知识结合起来，用面向对象的方式实现一个爬取“手机搜狐网”的多线程爬虫。

```
import pickle
import zlib
from enum import Enum, unique
from hashlib import sha1
from random import random
from threading import Thread, current_thread, local
from time import sleep
from urllib.parse import urlparse

import pymongo
import redis
import requests
from bs4 import BeautifulSoup
from bson import Binary


@unique
class SpiderStatus(Enum):
    IDLE = 0
    WORKING = 1


def decode_page(page_bytes, charsets=('utf-8',)):
    page_html = None
    for charset in charsets:
        try:
            page_html = page_bytes.decode(charset)
            break
        except UnicodeDecodeError:
            pass
    return page_html


class Retry(object):

    def __init__(self, *, retry_times=3,
                 wait_secs=5, errors=(Exception, )):
        self.retry_times = retry_times
        self.wait_secs = wait_secs
        self.errors = errors

    def __call__(self, fn):

        def wrapper(*args, **kwargs):
            for _ in range(self.retry_times):
                try:
                    return fn(*args, **kwargs)
                except self.errors as e:
                    print(e)
                    sleep((random() + 1) * self.wait_secs)
```

```
            return None

        return wrapper


class Spider(object):

    def __init__(self):
        self.status = SpiderStatus.IDLE

    @Retry()
    def fetch(self, current_url, *, charsets=('utf-8', ),
              user_agent=None, proxies=None):
        thread_name = current_thread().name
        print(f'[{thread_name}]: {current_url}')
        headers = {'user-agent': user_agent} if user_agent else {}
        resp = requests.get(current_url,
                            headers=headers, proxies=proxies)
        return decode_page(resp.content, charsets) \
            if resp.status_code == 200 else None

    def parse(self, html_page, *, domain='m.sohu.com'):
        soup = BeautifulSoup(html_page, 'lxml')
        for a_tag in soup.body.select('a[href]'):
            parser = urlparse(a_tag.attrs['href'])
            scheme = parser.scheme or 'http'
            netloc = parser.netloc or domain
            if scheme != 'javascript' and netloc == domain:
                path = parser.path
                query = '?' + parser.query if parser.query else ''
                full_url = f'{scheme}://{netloc}{path}{query}'
                redis_client = thread_local.redis_client
                if not redis_client.sismember('visited_urls', full_url):
                    redis_client.rpush('m_sohu_task', full_url)

    def extract(self, html_page):
        pass

    def store(self, data_dict):
        # redis_client = thread_local.redis_client
        # mongo_db = thread_local.mongo_db
        pass


class SpiderThread(Thread):

    def __init__(self, name, spider):
        super().__init__(name=name, daemon=True)
        self.spider = spider

    def run(self):
        redis_client = redis.Redis(host='1.2.3.4', port=6379,
password='1qaz2wsx')
        mongo_client = pymongo.MongoClient(host='1.2.3.4', port=27017)
        thread_local.redis_client = redis_client
        thread_local.mongo_db = mongo_client.msohu
        while True:
            current_url = redis_client.lpop('m_sohu_task')
```

```
            while not current_url:
                current_url = redis_client.lpop('m_sohu_task')
            self.spider.status = SpiderStatus.WORKING
            current_url = current_url.decode('utf-8')
            if not redis_client.sismember('visited_urls', current_url):
                redis_client.sadd('visited_urls', current_url)
                html_page = self.spider.fetch(current_url)
                if html_page not in [None, '']:
                    hasher = hasher_proto.copy()
                    hasher.update(current_url.encode('utf-8'))
                    doc_id = hasher.hexdigest()
                    sohu_data_coll = mongo_client.msohu.webpages
                    if not sohu_data_coll.find_one({'_id': doc_id}):
                        sohu_data_coll.insert_one({
                            '_id': doc_id,
                            'url': current_url,
                            'page':
Binary(zlib.compress(pickle.dumps(html_page)))
                        })
                    self.spider.parse(html_page)
            self.spider.status = SpiderStatus.IDLE


def is_any_alive(spider_threads):
    return any([spider_thread.spider.status == SpiderStatus.WORKING
                for spider_thread in spider_threads])


thread_local = local()
hasher_proto = sha1()


def main():
    redis_client = redis.Redis(host='1.2.3.4', port=6379, password='1qaz2wsx')
    if not redis_client.exists('m_sohu_task'):
        redis_client.rpush('m_sohu_task', 'http://m.sohu.com/')

    spider_threads = [SpiderThread('thread-%d' % i, Spider())
                      for i in range(10)]
    for spider_thread in spider_threads:
        spider_thread.start()

    while redis_client.exists('m_sohu_task') or is_any_alive(spider_threads):
        sleep(5)

    print('Over!')


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