协程爬虫(拓展)

目标：1.掌握 协程池模块的使用

2.掌握 使用线程池实现爬虫的流程

协程, 又称为微线程,是一种用户态的轻量级线程。协程不像线程和进程那样,需要进行系统内核上的上下文切换,协程的上下文切换是由程序员决定的

1.协程yield

```
def work1():
    while True:
        print("----work1---")
        yield
        time.sleep(0.5)


def work2():
    while True:
        print("----work2---")
        yield
        time.sleep(0.5)


def main():
    w1 = work1()
```

```
    w2 = work2()
    for i in range(5):
        next(w1)
        next(w2)


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

2.协程greenlet

```
"""协程：greenlet   安装：pip install greenlet"""
from greenlet import greenlet
import time


def test1():
    while True:
        print("---A--")
        gr2.switch()
        time.sleep(0.5)


def test2():
    for i in range(5):
        print("---B--")
        gr1.switch()
```

```
        time.sleep(0.5)


gr1 = greenlet(test1)
gr2 = greenlet(test2)

# 切换到gr1中运行
gr1.switch()
```

3.协程gevent

```
"""协程：gevent  操作一"""
import gevent


def f(n):
    for i in range(n):
        print(gevent.getcurrent(), i)


g1 = gevent.spawn(f, 5)
g2 = gevent.spawn(f, 5)
g3 = gevent.spawn(f, 5)
g1.join()
g2.join()
g3.join()
```

```
"""协程：gevent  操作二"""
```

```python
import gevent
import time
from gevent import monkey
# 只要有延迟，先执行下一步
monkey.patch_all()


def kang():
    for i in range(5):
        print(i)
        # time.sleep(1)


def dong(url):
    print(url)
    time.sleep(2)


if __name__ == "__main__":
    for i in range(5):
        # 给协程下达任务
        g1 = gevent.spawn(kang)
        g2 = gevent.spawn(dong, 5)
        print("kangdong")
```

4. 协程池使用介绍

```python
# 导入所需的库
```

```
import asyncio
import aiohttp
from lxml import etree
import csv
import time

# 定义异步函数 fetch，用于发送异步 HTTP 请求
async def fetch(session, url, semaphore):
    # 使用信号量限制并发数量
    async with semaphore, session.get(url)
as response:
        # 返回响应文本
        return await response.text()

# 定义异步函数 parse，用于解析 HTML 并提取数据
async def parse(html):
    result = etree.HTML(html)  # 使用 lxml
解析 HTML
    divs =
result.xpath('//div[@class="info"]')
    lst = []
    for div in divs:
        dic = {}
        # 提取电影标题、类型、评分和引言
        # ...解析逻辑...
        lst.append(dic)
    return lst

# 定义主异步函数 main
async def main():
```

```
    semaphore = asyncio.Semaphore(10)  # 限制并发数量为10
    async with aiohttp.ClientSession() as session:
        tasks = []
        # 为每个页面的 URL 创建一个异步任务
        for page in range(1, 11):
            url = f'https://movie.douban.com/top250?start={(page-1)*25}&filter='

 tasks.append(asyncio.create_task(fetch(session, url, semaphore)))
        # 并发执行所有任务并获取响应
        responses = await asyncio.gather(*tasks)

        all_movies = []
        # 解析每个页面的响应内容
        for response in responses:
            all_movies.extend(await parse(response))

    # 将收集到的电影信息写入 CSV 文件
    with open('douban.csv', 'w', encoding='utf-8-sig', newline='') as f:
        writer = csv.DictWriter(f, fieldnames=('titles', 'types', 'star', 'quote'))
        writer.writeheader()
```

```
        writer.writerows(all_movies)

# 程序入口
if __name__ == '__main__':
    start = time.time()  # 开始计时
    loop = asyncio.get_event_loop()  # 获取当前线程的事件循环
    loop.run_until_complete(main())  # 运行 main 协程直至完成
    end = time.time()  # 结束计时
    print(f'Total time: {end - start} seconds')  # 打印程序执行时间
```

```
import gevent.monkey
gevent.monkey.path_all()
from gevent.pool import Pool
```

5. 代码实现

```
"""协程池操作"""
'''
由于gevent的Pool的没有close方法，也没有异常回调参数
引出需要对gevent的Pool进行一些处理，实现与线程池一样接口，实现线程和协程的无缝转换
'''
```

```python
import random
num = random.randint(1,100)
running = True

while running:
    answer = int(input('guess(1-100):'))
    if answer > num:
        print("猜大了")
    elif answer < num:
         print("猜小了")
    else:
        print("猜对了")
        running =False


import gevent.monkey
gevent.monkey.patch_all()    # 打补丁，替换内置的模块
from gevent.pool import Pool


import requests
from lxml import etree
from queue import Queue
import time


class QiubaiSpider:
    def __init__(self):
```

```
        self.url_temp =
"https://www.qiushibaike.com/text/page/{}/"
        self.headers = {"User-Agent":
"Mozilla/5.0 (Macintosh; Intel Mac OS X \
        10_13_3) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/64.0.3282.186
Safari/537.36"}
        # 创建队列容器
        self.queue = Queue()
        # 创建任务池(同一时间，同步并发量) 默认大
小是cpu的个数
        self.pool = Pool(5)
        # 与break的效果相似
        self.is_running = True
        # 创建计数
        self.total_requests_num = 0
        self.total_response_num = 0

    def get_url_list(self):
        """获取url列表, 往队列中添加"""
        for i in range(1, 14):

 self.queue.put(self.url_temp.format(i))
            # url计数累加
            self.total_requests_num += 1

    def parse_url(self, url):
        """发送请求，获取响应"""
        return requests.get(url,
headers=self.headers).content.decode()
```

```
    def get_content_list(self, html_str):
        """
        提取段子
        :param html_str: 响应的源码
        :return: 段子列表
        """
        html = etree.HTML(html_str)
        # 获取段子内容
        content_list =
html.xpath('//div[@class="content"]/span/te
xt()')
        return content_list

    def save_content_list(self,
content_list):  # 保存数据
        print(content_list)
        pass

    def exetute_requests_item_save(self):
        # 从队列中获取url
        url = self.queue.get()
        # 将url传入定义的解析函数中，获取源码
        html_str = self.parse_url(url)
        # 将获取的源码传入定义的数据提取函数中，获
取段子内容列表
        content_list =
self.get_content_list(html_str)
        # 将段子内容列表，传入定义的保存的函数中
```

```
 self.save_content_list(content_list)
        # 响应计数累加
        self.total_response_num += 1

    def _callback(self, temp):
        # 递归退出条件
        if self.is_running:
            # 控制并发
            # 合理的利用cpu性能，提高并发数。

 self.pool.apply_async(self.exetute_request
s_item_save, callback=self._callback)

    def run(self):
        """程序启动运行"""
        self.get_url_list()

        for i in range(2):
            # 控制并发
            # 通过apply_async的方法让函数异步执
行，但是只能执行一次，为了让其能够被反复执行，通过添
加回调函数的方式能够让_callback递归的
            # 调用自己，同时需要指定退出条件   注
意：先做完执行，在回调

 self.pool.apply_async(self.exetute_request
s_item_save, callback=self._callback)

        # while True:
```

```
            #       # 防止主线程结束
            #       time.sleep(0.0001)  # 避免cpu空转，浪费资源
            #       # 当响应计数大于或等于url计数时，程序终止
            #       if self.total_response_num >= self.total_requests_num:
            #           self.is_running = False
            #           break


if __name__ == '__main__':
    """用户警告:libuv只支持毫秒定时器解决;所有时间更少将设置为1毫秒"""
    qiubai = QiubaiSpider()
    qiubai.run()
```