多进程爬虫

目标：掌握多进程在爬虫中的应用

进程：资源分配和调度的基本单位

> cpu密集型计算：多进程
>
> I/o密集型计算：爬虫，多线程可以

多进程知识点梳理(对比多线程threading)

语法	多线程	多进程
导入模块	from threading import Thread import threding	from multiprocessing import Process

语法	多线程	多进程
新建启动等待结束	t = Thread(target=func, args=(100,)) t.start() t.join()	p = Process(target=func, args=('100',)) p.start() p.join()
数据通信	import queue q = queue.Queue() q.put(data) data = q.get()	from multiprocessing import Queue q = Queue() q.put(data) data = q.get()
加锁	mutex = threading.Lock() mutex.acquire() mutex.release()	lock = multiprocessing.Lock() lock.acquire() lock.release()

语法	多线程	多进程
池化	from concurrent.futures import ThreadPoolExecutor with ThreadPoolExecutor() as executor: result1 = executor.map(func, [1,2,3]) result2 = executor.submit(func, 1) data = result2.result()	from concurrent.futures import ProcessPoolExecutor with ProcessPoolExecutor() as executor: result1 = executor.map(func, [1,2,3]) result2 = executor.submit(func, 1) data = result2.result()

1. 多进程方法

```
from multiprocessing import Process
import time


def func(x):
    start_time = time.time()
    for i in range(x):
```

```
            print('结果是', i)
    end_time = time.time()
    print(start_time - end_time)

if __name__ == '__main__':
    # 进程创建
    # target=func  func代表进程执行的函数，做什么任务
    # args=(3,)  kwargs={'x': 4}  参数传递的两种方式
    # start()  进程开启，执行
    Process(target=func, args=(3,)).start()
    Process(target=func, kwargs={'x': 4}).start()
```

2. 进程案例

lol皮肤图片爬虫，进程版实现

```
from multiprocessing import Process
from requests_html import HTMLSession
from fake_useragent import UserAgent
import jsonpath, os
session = HTMLSession()
ua = UserAgent()


class Lol(object):
    def __init__(self):
```

```
        self.start_url =
'https://game.gtimg.cn/images/lol/act/img/j
s/heroList/hero_list.js'
        self.p_url =
'https://game.gtimg.cn/images/lol/act/img/j
s/hero/{}.js'

    def parse_url(self):
        """
        提取c，提取英雄的名称
        :return: item_id_list, name_list
        """
        response_json =
session.get(url=self.start_url).json()
        # 提取英雄的id
        item_id_list =
jsonpath.jsonpath(response_json,
'$..heroId')
        return item_id_list

    def parse_item(self, item_id_list):
        """
        json数据的解析
        :param item_id_list: 英雄的id列表
        """
        for id in item_id_list:
            referer =
'https://lol.qq.com/data/info-defail.shtml?
id={}'.format(id)
            headers = {
```

```
                "user-agent": ua.chrome,
                "referer": referer
            }
            name_json =
session.get(url=self.p_url.format(id),
headers=headers).json()
            # print(name_json)
            # 电脑版图片
            win_img_list =
jsonpath.jsonpath(name_json, '$..mainImg')
            # 手机版的图片
            phone_img_list =
jsonpath.jsonpath(name_json,
'$..loadingImg')
            # print(win_img_list)
            # print(phone_img_list)
            # 创建文件夹的名称    英雄的名称
            name =
jsonpath.jsonpath(name_json, '$..name')[0]
            # print(name)
            print('文件夹创建中-----logging--
---')
            os_path = os.getcwd() + '/' +
name + '/'
            if not os.path.exists(os_path):
                os.mkdir(os_path)
            # 皮肤图片列表
            # 列表下标操作，过滤第一个数据
```

```
            name_list_data =
jsonpath.jsonpath(name_json, '$..name')
[1::]

 self.process_run(phone_img_list,
win_img_list, os_path, name_list_data)

    def save_data(self, img_list, os_path,
name_list_data, num):
        """
        保存
        :param img_list: 图片列表
        :param os_path: 保存的路径
        :param name_list_data: 英雄皮肤的名称
列表
        :return:
        """
        if num == 1:
            # 循环同下标的两个列表的元素
            for url, title in zip(img_list,
name_list_data):
                if url != '':
                    data =
session.get(url=url).content
                    with open(os_path +
title + '电脑版' + '.jpg', 'wb')as f:
                        f.write(data)
                        print('电脑版图片下载
完毕------logging-------{}'.format(title))
        if num == 2:
```

```
            # 循环同下标的两个列表的元素
            for url, title in zip(img_list,
name_list_data):
                if url != '':
                    data =
session.get(url=url).content
                    with open(os_path +
title + '手机版' + '.jpg', 'wb')as f:
                        f.write(data)
                        print('手机版图片下载
完毕------logging-------{}'.format(title))

    def process_run(self, phone_img_list,
win_img_list, os_path, name_list_data):
        Process(target=self.save_data,
args=(win_img_list, os_path,
name_list_data, 1)).start()
        Process(target=self.save_data,
args=(phone_img_list, os_path,
name_list_data, 2)).start()

    def run(self):
        item_id_list = self.parse_url()
        self.parse_item(item_id_list)


if __name__ == '__main__':
    lol = Lol()
    lol.run()
```

3. 多进程中队列的使用

多进程中使用普通的队列模块会发生阻塞，对应的需要使用multiprocessing提供的JoinableQueue模块，其使用过程和在线程中使用的queue方法相同

```
from queue import Queue
q = Queue(maxsize=100)
item = {}
q.put_nowait(item)  # 不等待，直接放
q.put(item)  # 放入数据，队列满的时候回等待
q.get_nowait()  # 不等待直接取，队列空的时候会报错
q.get()  # 取出数据，队列为空的时候会等待
q.qsize()  # 获取队列中现存数据的个数
q.join()  # 队列中维持了一个计数，计数不为0时候让主线程阻塞等待，队列计数为0的时候才会继续往后执行
q.task_done()
# put的时候计数+1，get不会-1，get需要和task_done一起使用才会-1
```

4. 进程队列案例

进程队列，场景解析

```
from multiprocessing import Process
```

```python
from multiprocessing import JoinableQueue
as Queue


class Love(object):
    def __init__(self):
        # 队列容量,队列创建 ， 【】， {}
        self.q = Queue()

    def parse_data(self):
        """功能：往队列添加数据"""
        data = "第{}天----我爱你----"
        for i in range(1, 100):
            # 将数据放入队列，put的时候计数+1，
get不会-1，get需要和task_done一起使用才会-1
            self.q.put(data.format(i))
        self.q.join()

    def func2(self):
        """功能：从队列中获取数据"""
        while True:
            # 循环从队列中获取，取出数据，队列为
空的时候会等待
            result = self.q.get()
            print(result)
            # 使队列计数-1
            self.q.task_done()

    def run(self):
        # 进程创建
```

```
            """进程：功能：往队列中添加数据"""
            m1 =
Process(target=self.parse_data)
            """进程：功能：从队列里面获取数据"""
            m2 = Process(target=self.func2)
            m1.start()
            # 将m2设置成守护进程 因为m2一直是死循环，
设置成守护进程之后当主程序代码运行完毕,m2就会结束，
不会成为僵尸进程
            # 即只在需要的时候才启动，完成任务后就自动
结束
            m2.daemon = True
            m2.start()
            # 队列中维持了一个计数，计数不为0时候让主
线程阻塞等待，队列计数为0的时候才会继续往后执行
            # 等待task_done()返回的信号量和put进去
的数量一直才会往下执行
            m1.join()


if __name__ == '__main__':
    love = Love()
    love.run()
```

上述多进程实现的代码中，multiprocessing提供的JoinableQueue可以创建可连接的共享进程队列。和普通的Queue对象一样，队列允许项目的使用者通知生产者项目已经被成功处理。通知进程是使用共享的信号和条件变量来实现的。 对应的该队列能够和普通队列一样能够调用task_done和join方法

##

5.进程池代码解析

```
from multiprocessing import Pool
import os, time, random


def worker(msg):
    t_start = time.time()
    print("%s开始执行,进程号为%d" % (msg,
os.getpid()))
    # random.random()随机生成0~1之间的浮点数
    time.sleep(random.random() * 2)
    t_stop = time.time()
    print(msg, "执行完毕，耗时%0.2f" %
(t_stop - t_start))


if __name__ == '__main__':
    po = Pool(3)  # 定义一个进程池，最大进程数3
    for i in range(0, 10):
        # Pool().apply_async(要调用的目标,(传
递给目标的参数元祖,))
        # 每次循环将会用空闲出来的子进程去调用目标
        po.apply_async(worker, args=(i,))
    """
    apply_async(func[, args[, kwds]]) ：使用
非阻塞方式调用func，并行执行（堵塞方式必须等待上一
个进程退出才能执行下一个进程）
```

```
    args为传递给func的参数列表，kwds为传递给func的关键字参数列表；
    """


    print("----start----")
    po.close()  # 关闭进程池，关闭后po不再接收新的请求
    po.join()  # 等待po中所有子进程执行完成，必须放在close语句之后
    print("-----end-----")
```

6.总结

```
"""进程"""
# 1.导入模块
# 2.定义，进程执行的函数()
# 3.创建进程，并启动
# 注意点：参数的传递    target=func   func代表进程执行的函数，做什么任务

"""进程池"""
# 1.导入进程池模块
# 2.定义，进程执行的函数()
# 3.定义进程池运行的进程数量
# 4.apply_async(func[, args[, kwds]]) ：使用非阻塞方式调用func，并行执行（堵塞方式必须等待上一个进程退出才能执行下一个进程）
# 5.关闭进程池，join一下
```

```
"""进程队列"""
# 1.导入进程队列模块，multiprocessing提供的JoinableQueue可以创建可连接的共享进程队列
# 2.创建容器,(创建进程队列对象)
# 3.往队列里面添加数据，put
# 4.从队列里面获取数据，get和task_done()结合使用
# 5.创建进程
# 6.设置守护进程  m2.daemon(True)
# 7.join一下
```