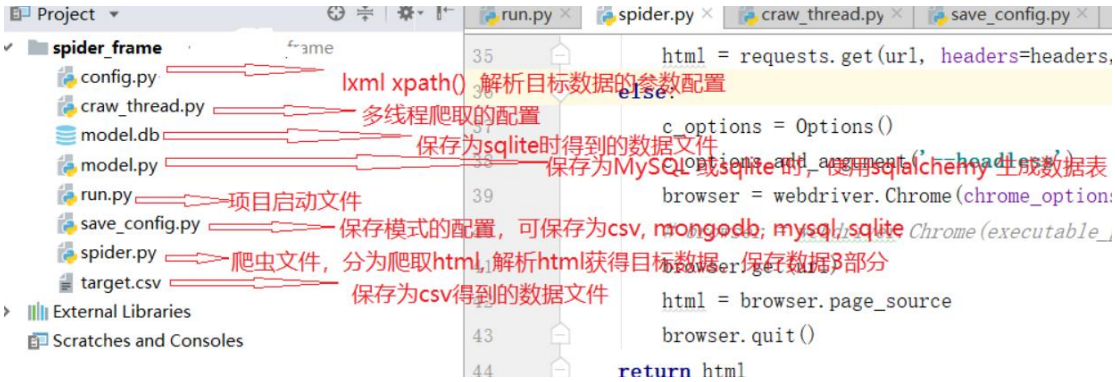
1.简介
2.使用
3.不足
4.总结

简介

先看一下项目整体的目录结构:



run.py: 启动文件，文件的具体参数请仔细看注释内容

spider.py: 具体的爬虫文件，里面主要有三个方法:

1）save_data(data, mode='db')：保存目标数据，data 是 2）中返回的 data，mode='db' 是保存的模式，这里默认保存为 MySQL 或 sqlite，当要保存为 csv 文件时，mode='csv'，要保存为 mongodb 时，mode='mongodb'

2）extract_target(html, dict_arg)：解析目标数据，实际调用时，html 为 3）中返回的 html 源码，dict_arg 为 config.py 文件 配置的 lxml xpath 方法的解析参数

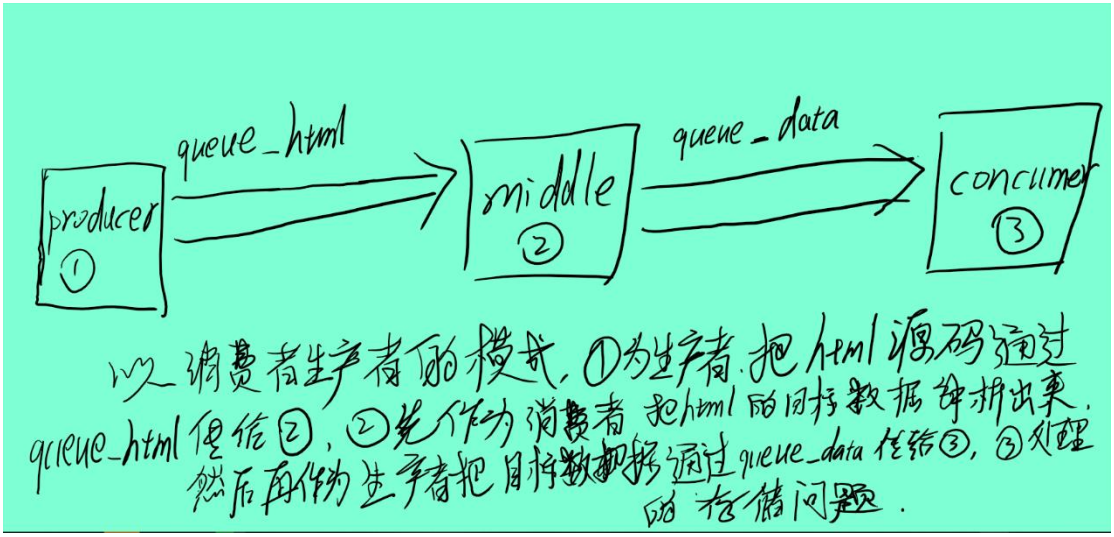
3）craw_html(url, way='')：url：爬取地址，way：使用爬取 html 源码的方式是 requests，或 selenium，这里默认使用 requests，当 way 不为空是则使用 selenium 方式，这里

selenium 用的是 chrome 浏览器的无头模式，当你需要用到这种方式爬取源码时，还需要下载和你的 chrome 浏览器匹配的 chrome 浏览器驱动器。

config.py: 配置的 lxml xpath 方法的解析参数

```
parser_arg = {
            'title': '//div[@class="w"]/div[@class="bt"]/a//text()',
            'img': '//div[@class="w"]/div[@class="pic"]/a/img/@src',
            }
```

craw_thread.py: 多线程爬取(使用的 queue 进行线程间通信，queue 是线程安全的)， 原理如下图:



默认各使用 3 个线程（3 个爬取 html 源码的线程即 producer，3 个解析获取目标数据的线程即 middle，3 个存储目标数据线程即 concumer）来处理，producer，middle，concumer 的实现使用的是 spider.py 文件中对应的 craw_html(), extract_target(), save_data()方法

model.py：主要使用 sqlalchemy 来生成保存模式为 mysql，或 sqlite 的数据表。如果保存为 sqlite，会在当前目录生产一个

model.db 的数据库文件，该文件保存的即为目标数据，可以用 navicat，sqlitebrowser 等工具打开，如果保存为 mysql，请把#

```
engine = create_engine('mysql://root:root@localhost/xzj?charset=utf8', pool_size=100)
# 保存至 mysql
```

该行注释放开，填写你本地 mysql 数据库的用户名和密码，并把上一行保存为 sqllite 的代码注释

save_config.py:

该文件主要处理保存为 mysql, sqlite, csv, mongodb 等保存模式的实现，使用该框架一般不需要修改该文件

使用

1）修改 run.py 文件的 main()函数的相关参数，具体参数值请看注释内容，修改目标 url

2）修改 config.py 中的参数，该参数是一个字典， 字典的键会与保存的目标字段值一致（当保存为 mysql，或 sqlite 时数据断要根据此处的参数名进行相应修改，3）会详细说明），字典的值需要符合 lxml xpath() 解析语法。 爬虫所要用到的解析规则可以统一放在此处配置，只需在调用 spider.py 文件的 extract_target() 时传入相对应的参数即可。

```
parser_arg = {
            'title': '//div[@class="w"]/div[@class="bt"]/a//text()',
            'img': '//div[@class="w"]/div[@class="pic"]/a/img/@src',
            }
```

3）修改 save_config.py 文件:

1）当保存为 sqllite 或 mysql 时，请仔细阅读下图代码的注释

```
def save_as_db(data):
    Model.metadata.create_all(engine)    # 创建   Model 数据库，该名称与model.py 中的class Model 是一致的
    for obj in get_obj_list(data):       # 将所有数据循环存储到表中
        mx = Model(                      # sqlalchemy 的 orm 的存储方式
            title=obj['title'],          # title, img 与config.py 文件中的参数的键名是一致的
            img=obj['img'],
        )
        sess.add(mx)
        sess.commit()
    print('----------db-------------')
```

2）当保存为 csv，数据库文件名是 target.csv，如果该文件名已存在，会往该文件追加数据， 如果想重新存放所有数据，请把已存在的 target.csv 文件重命名或删除。

3）当保存的模式是 mongodb 时，默认的数据库名是 Target， 数据表名是 target，注意你本机的 mongodb 是否存在重名的数据库。

4）修改 model.py 文件：

该文件主要处理生成的数据表字段，通过 sqlalchemy 生成数据表

不足

1） 参数封装的不够彻底：run.py 的 main 函数的参数和爬取的 url 可以直接放到命令行中传入，如 scrapy 一样通过命令启动项目时传入一些必要的参数。启动项目时需要修改的地方过于分散，如果能集中到一处，同时修改会更好

2） 当启用多线程爬取时，数据已经爬取完成时项目不会自动停止，需要手动停止。原来是通过判断 queue_detail_url，queue_html，queue_data 队列中的数据是否为空来决定是否退出循环爬取，但考虑到爬取 html，解析 html，保存目标数据三部分所用到的

时间是不一样的，用时短的部分会让其中某个队列先为空，造成程序提前退出，所以这里用的是一个 while True 的死循环。为了让项目能自动停止，我现在能想到的解决方法是通过判断 queue_detail_url 队列为空时，让 producer 通过多线程的 notify() 通知处于 wait() 状态的 middle 然后 producer 部分跳出死循环。middle 接到通知后，又 notify() 处于 wait() 状态的 consumer 然后跳出死循。consumer 接到 notify() 后直接跳出死循环。

总结

关于我自己封装的爬虫框架已经全部介绍完毕了，如果你有什么不了解或对不足之处有更好的解决方案可以给我发邮件，我的邮箱：xiezhaojun1010@163.com，

所有的代码的 github 地址：https://github.com/JackXie1010/spider_frame