【目的及要求】

目的：

开发一个 Scrapy 爬虫，结合 Selenium，用于从豆瓣电影网站抓取电影详情、演员信息以及用户评论数据。实现对演员个人页面的深度访问，以获取更详细的演员信息。获取并解析用户评论，包括评论者的用户名、评论标题和内容。

要求：

使用 XPath 选择器定位网页元素。处理中文字符，确保数据正确解析和存储。实现翻页功能，能够连续抓取多页评论。使用正则表达式进行文本清洗。结构化输出数据，使用自定义的 Scrapy Item 来存储演员和评论信息。

【原理】

Python：编程语言，提供强大的数据处理和网络请求能力，结合 Scrapy 和 Selenium 实现自动化爬虫。

Scrapy：一个快速、高级的 web 爬虫框架，用于抓取网页数据，提供了一种简单的方式来提取结构化的数据。

Selenium：用于自动化浏览器操作，帮助解决 JavaScript 渲染的内容抓取问题，可以驱动真实的浏览器实例。

XPath：用于在 XML 文档中查询和选择元素，这里用于在 HTML 文档中定位元素位置。

正则表达式：用于字符串模式匹配，本实验中用于提取和清洗数据中的特定模式。

【环境】

PyCharm 2024.1.1 (Python 3.11)

【设置】

环境准备：安装 Python 环境，配置 Scrapy 和 Selenium 库，以及相应的 WebDriver。

爬虫设计：定义爬虫类 MovieSpider，设置起始 URL 和允许的域名。

数据提取：通过 XPath 和正则表达式从网页中提取基本信息、演员和评论数据。

数据处理：利用 Python 函数和正则表达式清洗数据，如去除多余空格、括号等。

结果存储：使用自定义的 Items 存储数据，便于后期分析或导入数据库。

异常处理：添加错误处理逻辑，确保爬虫稳定运行。

测试与优化：运行爬虫，检查数据完整性，并对性能进行优化。

资源释放：在爬虫结束时，关闭 Selenium 驱动的浏览器实例。

【过程】

本实验的要求是从豆瓣电影“选电影”选项卡中挑选出电影。经观察发现，当处于未登录状态（左图）时，该选项卡所对应的页面中仅有 6 个电影，而当登录豆瓣账号（右图）后，该选项卡所对应的 Xpath 会发生改变（因为在其前方会出现“我看”选项卡，导致“选电影”选项卡对应的 Xpath 由 li[2]变为 li[3]）。本实验基于未登录状态，且以其中第一个电影为例展示实验过程。

在正式开始爬虫前，还按照惯例对爬虫进行了一些设置和初始化。例如在 setting.py 文件中，修改了 USER_AGENT 以绕开豆瓣网对爬虫的限制，得到正确的结果，设置 LOG_LEVEL='ERROR'，使得只在出错时输出日志信息，提升可读性。另外，由于要进行动态网页爬取，解除了设置中对 DOWNLOADER_MIDDLEWARES 的注释，以便 middlewares.py 中的代码能够正确实现。

```
USER_AGENT = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36 Edg/125.0.0.0"
LOG_LEVEL = 'ERROR'
```

```
DOWNLOADER_MIDDLEWARES = {
   "douban.middlewares.DoubanDownloaderMiddleware": 543,
}
```

接下来，正式开始爬虫部分代码的编写。

首先，初始化爬虫类，定义 MovieSpider 类继承自 scrapy.Spider。设置爬虫名称“movie”、起始 URL：https://movie.douban.com/。基于实验中的具体需要，又分别定义了 MovieSpider 类的三个实例变量 movielink，page 和 link1。其中，movielink 被用来存储从豆瓣电影首页抓取到的第一个电影的链接，page 在 reviewdetail 方法中被用来构建下一页评论的 URL 并控制翻页逻辑，link1 用于存储从电影页面抓取到的第一个演员或评论列表的链接。

```
class MovieSpider(scrapy.Spider):

    name = "movie"

    start_urls = ["https://movie.douban.com/"]

    movielink = ''

    page = 1

    link1 = ''
```

另外，由于该 URL 对应的网页由 JavaScript 渲染，需要采用动态网页爬取的技术，故初始化 Selenium 了一个 Edge 浏览器的 WebDriver。

```
def __init__(self):

    self.driver = webdriver.Edge()
```

为了实现整个流程，还在 Scrapy 的中间件（middlewares）中定义的 process_response 方法中实现了针对特定 URL（spider.movielink）的动态页面加载功能，利用 Selenium 的 WebDriver（spider.driver）来获取完整的页面源码。

```
def process_response(self, request, response, spider):

    driver = spider.driver

    if request.url == spider.movielink:

        driver.get(spider.movielink)

        pagetext = driver.page_source

        new_response = HtmlResponse(url=request.url, body=pagetext,
encoding='utf-8', request=request)

        return new_response

    else:

        return response
```

然后，使用 parse 方法从主页提取电影详情页面的链接，通过使用浏览器中的“检查”功能获取所要爬取的电影对应的 XPath，并发起新的 Request 到电影详情页面。这里

使用 yield 发送请求到电影详情页面，并利用 callback 参数指定了响应的处理函数，正确过度到 movieparse——电影详情页面解析的主体部分。

```
def parse(self, response):

  self.movielink = response.xpath('//*[@id="db-nav-
movie"]/div[2]/div/ul/li[2]/a/@href').extract_first()

  yield scrapy.Request(url=self.movielink, callback=self.movieparse)
```

在 movieparse 方法中，分别调用了 actorparse 和 reviewparse 方法进行进一步的数据抓取演员列表和评论列表中的内容。此处，由于需要多次访问同一个页面来获取不同的数据，需要禁用 Request 的过滤去重机制。于是在代码中添加了 dont_filter=True 部分，避免在初步实验中出现的只能回去演员信息而评论信息被跳过，返回一个空值的错误。

```
def movieparse(self, response):

   self.link1 =
response.xpath('//*[@id="app"]/div/div[2]/ul/li[1]/a/@href').extract_first()

   yield scrapy.Request(url=self.link1, callback=self.actorparse,
dont_filter=True)

   yield scrapy.Request(url=self.link1, callback=self.reviewparse,
dont_filter=True)
```

在 actorparse 方法中，遍历了电影详情页面呈现出来的所有演员信息，使用 XPath 定位到演员名字和角色，构造 ActorItem 实例，并使用 yield scrapy.Request 发送请求到演员个人详情页面。其中，选择从 litags[1:]开始遍历，这是因为列表中的第一个元素是导演信息，与我们的需求不符。另外，meta={'item': item}将当前的 ActorItem 实例作为额外的元数据传递给下一个请求。

```
def actorparse(self, response):

   litags = response.xpath('//*[@id="celebrities"]/ul/li')

   for li in litags[1:]:

      actor = li.xpath('./div/span[1]/a/text()').extract_first()

      role = li.xpath('./div/span[2]/text()').extract_first()



      item = ActorItem()

      item['actor'] = actor

      item['role'] = role



      actorlink = li.xpath('./div/span[1]/a/@href').extract_first()
```

```
        yield scrapy.Request(url=actorlink, callback=self.actordetail,
meta={'item': item})
```

在 ActorItem 实例中，分别定义了 actor、role、span1 和 span2 四个信息，其中 span1、span2 改变了原本设计中的 gender 和 birthday。这样改变的原因是豆瓣网的演员详情页对不同演员的信息介绍差异较大，并不是每个演员的介绍中第一、二个列表的信息都是性别和生日，将它们记为“span”这样通用的名字（改名字与其 Xpath 对应）更加合理。

```
class ActorItem(scrapy.Item):

    actor = scrapy.Field()

    role = scrapy.Field()

    span1 = scrapy.Field()

    span2 = scrapy.Field()
```

在 actordetail 方法中，对于每一个列表项，首先提取出信息标题并使用正则表达式清洗标题，仅保留中文部分，这是因为标题可能包含冒号或英文单词等非中文字符。接着，提取出与标题相对应的具体内容，例如出生地的具体地点或生日的具体日期。最后，使用.strip()去除内容两边的空白字符，保持数据的整洁。通过遍历列表项，将提取到的标题和内容填充到 ActorItem 实例中，键名为 actorkey 列表中的元素，即'span1'和'span2'，值为处理后的标题和内容组合。

```
def actordetail(self, response):

    item = response.meta['item']

    litags = response.xpath('//ul[@class="subject-property"]/li')

    actorkey = ['span1', 'span2']

    for litag in litags[:2]:

        name = litag.xpath('./span[1]/text()').extract_first()

        name = re.search(r'[\u4e00-\u9fa5]+', name).group()



        content = litag.xpath('./span[2]/text()').extract_first().strip()

        i = litags.index(litag)

        item[actorkey[i]] = name + ": " + content

    yield item
```

在 reviewparse 方法中，首先使用正则表达式来从 self.link1 中提取电影的 ID。这个 ID 位于 URL 的尾部，是一个数字序列，用于唯一标识电影。然后，使用 XPath 来获取评论列表页面的 URL 后缀，指向电影评论的列表页面。最后，将电影 ID 和评论列表的 URL 后缀拼接到基础 URL "https://movie.douban.com/subject/" 上，形成完整的评论列表页面的 URL，存储在 self.url 中。这样的设计经过了深思熟虑。传统的 self.url =

self.start_urls[0] + urltail 构造方法中，self.start_urls 即为某个电影的详情页面，因此是可以采用的。但是本实验的电影详情页并非直接输入，而是通过一步步的传递得到，经验证并不是 https://movie.douban.com/subject/，而是通过一个自动跳转得到。这就造成了如果直接在后面加 ID 和“reviews”得到的 URL 并不存在。根据实际操作验证发现又会自动跳转回电影详情页，而不能正确返回评论页面。按照修改后的代码中这种进一步细化拆解的方法才能实现正确的跳转，得到想要的结果。

```
def reviewparse(self, response):
    movieid = re.search(r'/(\d+)$', self.link1).group(1)
    urltail = response.xpath('//*[@id="reviews-
wrapper"]/p/a/@href').extract_first()
    start = ["https://movie.douban.com/subject/"]
    self.url = start[0] + movieid + '/' + urltail
    yield scrapy.Request(url=self.url, callback=self.reviewdetail)
```

在 reviewdetail 方法中，使用 XPath 来定位到拥有特定的类名"review-list"，包含评论信息的 元素集合。对每一个 div 元素，即每个评论块，执行以下操作：提取评论者 ID，提取评论标题，提取短评内容。在处理短评内容时，为了应对豆瓣中有的评论会标记“这篇影评可能有剧透”的问题，做了进一步的设计：如果 shortcontent 的长度大于 3，选择第 3 个元素作为内容；否则选择第一个元素。之后，使用正则表达式来清除内容中的括号，并去除首尾空白。

```
def reviewdetail(self, response):
    divtags = response.xpath('//div[@class="review-list  "]/div')
    for div in divtags:
        idname = div.xpath('./div/header/a[2]/text()').extract_first()
        title = div.xpath('./div/div/h2/a/text()').extract_first()


        shortcontent = div.xpath('.//div[@class="short-content"]//text()')
        if (len(shortcontent)>3):
            shortcontent = shortcontent.extract()[2]
        else:
            shortcontent = shortcontent.extract_first()
        shortcontent = re.sub(r'\(',  '', shortcontent).strip()
```

创建一个新的 ReviewItem 类（items.py 中）及对象（movie.py 中），并将提取到的评论者 ID、评论标题和短评内容填充进去。

```
class ReviewItem(scrapy.Item):
    idname = scrapy.Field()
    title = scrapy.Field()
    shortcontent = scrapy.Field()
```

```
item = ReviewItem()
item['idname'] = idname
item['title'] = title
item['shortcontent'] = shortcontent
  yield item
```

在 reviewdetail 方法中还通过循环处理了翻页问题。代码检查了是否达到想要抓取的评论页数（本实验设置为 5 页）。如果是，构造新的 URL，通过将?start=%d 格式化字符串与当前页数相乘的结果追加到基本 URL 上，实现翻页效果。self.page 的值会在每次翻页后递增，直到达到设定的上限。在满足翻页条件时，构造一个新的请求，指向下一页评论的 URL，callback=self.reviewdetail 指定这个响应将继续由 reviewdetail 方法处理，实现连续抓取多页评论的功能。

```
newurltail = '?start=%d'
  if self.page < 5:
    newurl = self.url + format(newurltail%(20*self.page))
    self.page += 1
    yield scrapy.Request(url=newurl, callback=self.reviewdetail)
```

在数据存储与导出的过程中，pipelines.py 中的 DoubanPipeline 类扮演着核心的角色，它负责处理从爬虫传递过来的 ActorItem 和 ReviewItem 实例，并将它们保存到本地文件中。在 open_spider 方法中，创建了两个文件句柄，一个用于存储演员信息(actor_file)，另一个用于存储评论信息(review_file)。这两个文件分别命名为 actor.txt 和 review.txt，并且使用 UTF-8 编码以正确处理中文字符。

```
class DoubanPipeline:
    def __init__(self):
        self.actor_file = None
        self.review_file = None
    def open_spider(self, spider):
        self.actor_file = open('./actor.txt', 'w', encoding='utf-8')
```

```
        self.review_file = open('./review.txt', 'w', encoding='utf-8')
```

process_item 方法接收爬虫传递过来的 Item 实例，并根据 Item 的类型进行不同的处理。这里使用 isinstance 函数来检查 Item 的类型。如果 Item 是 ActorItem 类型，将演员的名字(actor)、角色(role)以及从演员个人页面抓取的例如出生地和生日等(span1 和 span2)写入 actor_file，每条记录以换行符分隔，便于阅读和后续处理；如果 Item 是 ReviewItem 类型，将评论者的 ID(idname)、评论标题(title)和评论内容(shortcontent)写入 review_file，每条评论以换行符分隔，保证文件的整洁和数据的清晰。

```
def process_item(self, item, spider):
```

```
    adapter = ItemAdapter(item)

    if isinstance(item, ActorItem):

        actor = adapter['actor']

        role = adapter['role']

        span1 = adapter['span1']

        span2 = adapter['span2']

        self.actor_file.write(actor + ' ' + role +
'\n\t'+span1+'\n\t'+span2+'\n')



    elif isinstance(item, ReviewItem):

        idname = adapter['idname']

        title = adapter['title']

        shortcontent = adapter['shortcontent']

        self.review_file.write(idname + '\n\t' + title + '\n\t' +
shortcontent + '\n\n')
```

在 close_spider 方法中，确保在爬虫结束时关闭所有打开的文件句柄。这通过检查 self.actor_file 和 self.review_file 是否为真（即是否已打开），并调用 close()方法来实现。

```
def close_spider(self, spider):
```

```
    if self.actor_file:

        self.actor_file.close()

    elif self.review_file:

        self.review_file.close()
```

【结论】

经过上述的步骤，得到了电影的基本信息和两个文本文档：actor.txt 和 review.txt。

下面将具体的内容加以展示。其中评论根据要求仅展示了前 10 条（事实上由于豆瓣“选电影”的推荐逻辑较为难以理解，该片较新和小众，实验时仅有 8 条长评，因此只能展示这 8 条），实际的文本文档中按照代码设计为前 5 页的所有评论。

电影名称：弗里蒙特 Fremont
上映年份：2023
导演：巴巴克·贾拉利
编剧：卡洛琳娜·卡瓦利
主演（仅取前三个）：
阿奈塔·瓦利·扎达
尼尔·汉博格
杰瑞米·艾伦·怀特
剧情简介：

年轻美丽的阿富汗女孩多尼娅的日子过得并不轻松，她曾为美军担任翻译，如今饱受失眠困扰。她独自居住在弗里蒙特的一栋公寓楼中，邻居大多是和她一样的阿富汗移民。她常常一个人去一家生意萧条的餐厅，一边吃饭一边看肥皂剧。多尼娅在城里的一家幸运饼干工厂工作。一天，老板给她升了职，让她负责在幸运饼干的签纸上写签语，她的生活也随之改变。随着越来越多湾区的陌生人读到她的签语，多尼娅心中有了一个愈发难以克制的念头:向世界传递一条特别的信息——虽然她并不确定这条信息将去往何方。

阿奈塔·瓦利·扎达 饰 Donya
性别: 女
编号: nm14282085
Hilda Schmelling 饰 Joanna
编号: nm4903509
职业: 演员
杰瑞米·艾伦·怀特 饰 Daniel
性别: 男
出生日期: 1991 年 2 月 17 日
Avis See-tho 饰 Fan
编号: nm14282086
职业: 演员
尼尔·汉博格 饰 Dr. Anthony
性别: 男
出生日期: 1967 年 11 月 25 日

言筱易
翻译丨《弗里蒙特》影评：绝妙的贾木许式的作品
原文链接： https://www.theguardian.com/film/2023/sep/17/fremont-review-utterly-delightful-jarmuschian-drama-babak-jalali-anaita-wali-zada-jeremy-allen-white 作者 Wendy Ide 在巴巴克·贾拉利的这部充满魅力的黑白影片中，一位阿富汗翻译在加利福尼亚州的单人床上寻...
余颀
诗化的孤独，真是无比迷人
这是一首聆听孤独回响的散文诗 捕捉着寂寞定格的光影， 讲述着小城孤独的人。 我陶醉于它 迷人的黑白光影，精致的比例构图 还有那悠扬惆怅的低沉独奏 影像成了编码，勾勒出银幕内外 我们各自内心最寂寞的角落。 其实 无论我们是 I 人，还是 E 人， 孤独何尝不与我们为伴， 它在生命...

NatNatNat
一个关于孤独的小故事
这是一个有关孤独和战后应激的小故事,不长的记录里囊括了关于工作生活的诸多细节,从这些对话和小事里能看到一个有点特立独行的女生从如何在对抗自己的睡眠障碍开始,和心理医生对话和朋友交流以及用工作传递能量和信息,慢慢治愈自己。 也许是因为议题有点大,所以导演反而选...
大老师
《弗里蒙特》:你要咖啡吗?
港湾里的船是安全的,但这不是我们造船的原因 ——心理医生送给女主角的签语 一个年轻的阿富汗女孩在美国的生活,非常美式的视角和趣味。导演巴巴克·贾拉利是伊朗裔英国人,显然更善于把握移民题材。 一个讲述“孤独”的故事,“政治正确”的碎片镶嵌其中,但抛去令现代人敏感...
追影人 MC
孤独感爆棚!你能看完这部电影?必是情绪稳定之人!
关注我,看文字电影 摘要|孤独 这几天,我试着去理解女主角的孤独, 无法言喻... 某天加班回家的路上,我抬头看了看夜空, 觉着她... 孤独就像漆黑的夜,而她对爱情的渴望, 就像是漆黑夜空中,瞬间消逝的烂漫烟花。 什么是孤独? 对此,我不愿有一个固定的定义, 最好是有一...
crazYSL
弗里蒙特
影片通过唐雅与心理医生的对话交代了她的过往与内心。女主是阿富汗人,96 年生于阿富汗,曾在阿富汗的美军基地当翻译,如今来到美国加州弗里蒙特已有 8 个月,在唐人街一家华裔开的幸运饼干(在包装内写上祝福语并随机发给顾客,类似答案之书?)工厂做工,有心理问题。 为了钱唐...
jenny
小而美的一部片子
年底好多好电影出来,感觉一下子都看不完。 这两天看的,弗里蒙特, 一个阿富汗女孩,曾经为美军当翻译,所以幸运地跟着一起撤回美国,在一家工厂打工,失眠。 她那个心理医生刚出场的时候,低眉耷眼的,我心想,“就你这状态,还做心理医生呢?”后来他给弗里蒙特读 White fang...
在看奥
初次观感
就移民身份问题看的 构图画面不错,导演对角色捕捉相当细腻。 导演喜欢通过人物停顿展现构图跟孤独感,但感觉像剪辑凑时长,略显冗长,镜头手法单一保守,但效果不错。 剧情整体平淡塑造,也没针对移民问题核心讨论。 女主的翻译身份并没有着太多笔墨,在影片中更多作为区分女...