北京郵電大學

Python 实验报告

题目：Scrapy 爬虫的编写

姓　　名　王晓晖

学　　院　计算机学院

专　　业　计算机科学与技术

班　　级　2019211301

学　　号　2019213683

班内序号　15

指导教师　杨亚

2021 年 11 月

目　　录

第一章　Scrapy 的安装与项目构建

1.1　Scrapy 的安装

通过 pip 直接安装 Scrapy

代码 1-1 Scrapy 的安装

```
pip install Scrapy
```

如果一切正常的话，就可以在终端中运行 Scrapy 的命令了

如果出现问题可以检查环境变量的设置

1.2　Scrapy 项目构建

在合适的目录下运行 Scrapy 的命令初始化一个项目

代码 1-2 Scrapy 项目构建

```
scrapy startproject Spider
```

这时，scrapy 会自动生成一个名为 Spider 的项目，并且会在当前目录下生成一个 Spider 目录

1.3　Spider 目录结构

代码 1-3 Spider 目录结构

```
Spider
    scrapy.cfg
    Spider
        __init__.py
        items.py
        middlewares.py
        pipelines.py
        settings.py
        spiders
            __init__.py
```

其中，我们自己的爬虫要写在 spiders 目录下。

items.py 是用来表述抓取到数据的结构。

middlewares.py 是用来处理请求的中间件。

pipelines.py 用来处理数据的管道。

settings.py 用来设置 Scrapy 的配置。

第二章　爬取链家

本代码的 github 链接为https://github.com/NekoMio/Python-Spider/tree/lianjia
项目中包含代码和完整结果。

2.1　Item 类的编写

首先我们编辑 items.py 文件，定义一个 LianjiaItem 类。用来描述爬取到的数据的结构。

代码 2-1 Item 类的编写

```
class LianjiaItem(scrapy.Item):
    # define the fields for your item here like:
    name = scrapy.Field()
    price = scrapy.Field()
    area = scrapy.Field()
    unit_price = scrapy.Field()
    place = scrapy.Field()
    pass
```

2.2　spider 的编写

spider 需要写在 spiders 文件夹下
一种方法是直接用 scrapy genspider lianjia bj.lianjia.com 命令生成一个 spider 模板
或者可以手动在 spiders 文件夹下创建一个文件 spider.py 写入以下代码

代码 2-2 spider 的编写

```
import scrapy
from Spider.items import LianjiaItem
class LianjiaSpider(scrapy.Spider):
    name = 'lianjia'
    allowed_domains = ['bj.lianjia.com']
    start_urls = ['https://bj.lianjia.com/ershoufang/dongcheng/',
                  'https://bj.lianjia.com/ershoufang/xicheng/',
                  'https://bj.lianjia.com/ershoufang/chaoyang/',
                  'https://bj.lianjia.com/ershoufang/haidian/',
                 ]
```

这里定义了 Spider 的名字为 lianjia，允许的域名为 bj.lianjia.com，以及起始的 url，分别对于东城、西城、朝阳、海淀这四个区域的房源。

下一步要为它编写 prase 函数。

这里要注意 prase 是 LianjiaSpider 的一个方法。他是 LianjiaSpider 类之下的一部分。

代码 2-3 spider parse 的编写

```
1  def parse(self, response, **kwargs):
2      self.logger.info('A response from %s just arrived!, args: %s', response.url,
           kwargs)
3
4      item = LianjiaItem()
5
6      for each in response.xpath("/html/body/div[4]/div[1]/ul/li/div[1]"):
7          # self.logger.info('each: %s', each)
8          item['name'] = each.xpath("div[2]/div/a[1]/text()").get().strip()
9          # self.logger.info('name: %s', item['name'])
10         item['price'] = each.xpath("div[6]/div[1]/span/text()").get().strip() +
               each.xpath("div[6]/div[1]/i[2]/text()").get().strip()
11         # self.logger.info('price: %s', item['price'])
12         item['area'] = each.xpath("div[3]/div/text()").get().split('|')[1].strip()
13         # self.logger.info('area: %s', item['area'])
14         item['unit_price'] = each.xpath("div[6]/div[2]/span/text()").get().strip()
15         item['place'] = Pinyin_to_City[response.url.split('/')[4]]
16         if (item['name'] and item['price'] and item['area'] and item['unit_price']):
17             # self.logger.info('%s %s %s %s', item['name'], item['price'],
                   item['area'], item['unit_price'])
18             yield item
```

在这里我们定义了一个 item，他是我们用来存储数据的结构。

之后我们要打开浏览器，开始寻找我们需要爬取的数据。

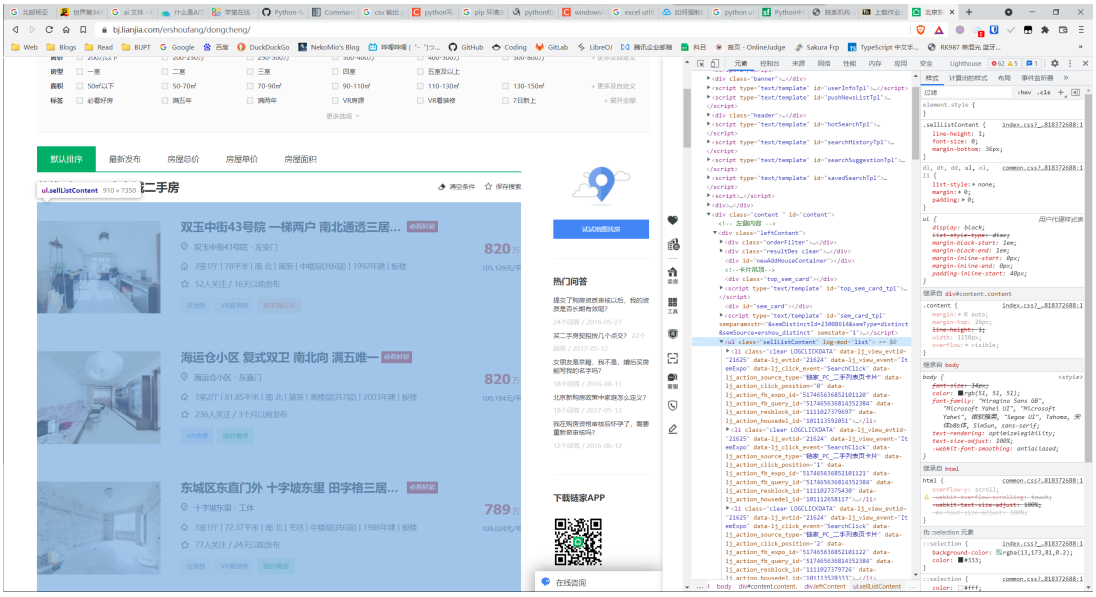


图 2-1 找到链家的数据的父结构

可以在浏览器中直接复制完整 XPath

并将其粘贴到 response.xpath() 中。

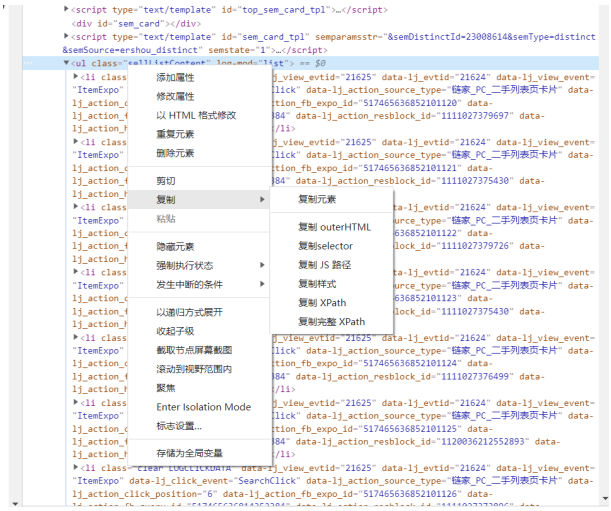


图 2-2 复制链家的数据的父结构的 XPath

代码中的 for each 是对每一个房子进行循环，在 debug 时可以将 each 输出查看结构，便于调试。比如我 self.logger.info('each') 一行就是输出了 each 的结构

在获取 each 之后可以对它在通过子路径找到我们需要的数据。

这里有一个方法是重新复制完整的 Xpath，再寻找和 each 的共同前缀，然后删除掉就能找到相对路径了。

到这里为止，我们就已经找到了数据所在的位置，理论上，我们就可以获取一页的数据了。但是我们的任务不止于此，我们需要获取至少 5 页的数据。所以我们要获取下一页的 url。这里有两种方法

- 第一种是直接观察地址链接的规律，手动修改 url 来获取下一页的地址
- 第二种是找到网页中下一页按钮的位置，获取它所指向的地址

这里我用第二种方法来编写

代码 2-4 spider parse 的编写

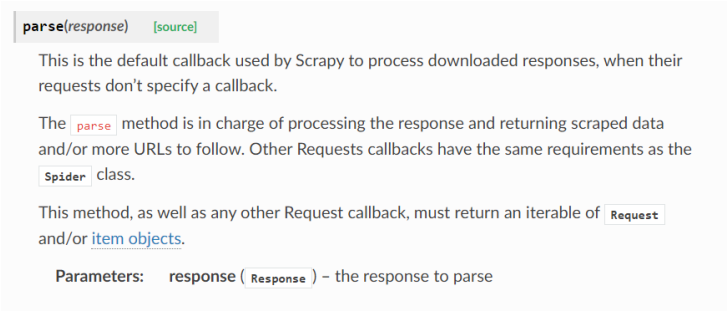
```
1 next_page =
    response.xpath("/html/body/div[4]/div[1]/div[7]/div[2]/div/a[last()]/@href").get()
```

这种方法看起来简单，但是实际上会更加的复杂。

如果我们以静态方式获取链家的网页，我们无法获取到这个值，因为链家的下一页是通过 js 生成的链接，所以我们需要使用 selenium 来获取。这一部分的内容在 middlewares 中会进行详细的描述。

我们现在假设我们已经获取到了这个值，那么我们如何去请求下一页呢？

这里我们查看 Scrapy 的文档。可以看到 parse 方法可以返回一个 request 在这里返



parse(*response*) [source]

This is the default callback used by Scrapy to process downloaded responses, when their requests don't specify a callback.

The `parse` method is in charge of processing the response and returning scraped data and/or more URLs to follow. Other Requests callbacks have the same requirements as the `Spider` class.

This method, as well as any other Request callback, must return an iterable of `Request` and/or item objects.

Parameters: **response** (`Response`) – the response to parse

图 2-3 Scrapy-parse

回的 Request 会被重新执行 parse 方法。

这是，只剩下最后一个问题，如何知道现在是第几页？

在 Request 中可以有一个 *cb_kwargs*，这个参数是一个字典，它将传递给用来处理它的函数，比如我们这里的 parse 函数，我们可以把第几页的值存储在里面。

所以最后我们的代码如下：

代码 2-5 spider parse 的编写

```
1    class LianjiaSpider(scrapy.Spider):
2    name = 'lianjia'
3    allowed_domains = ['bj.lianjia.com']
4    start_urls = ['https://bj.lianjia.com/ershoufang/dongcheng/',
5                  'https://bj.lianjia.com/ershoufang/xicheng/',
6                  'https://bj.lianjia.com/ershoufang/chaoyang/',
7                  'https://bj.lianjia.com/ershoufang/haidian/',
8                 ]
9
10   def start_requests(self):
11       self.logger.info('start_requests')
12       for url in self.start_urls:
13           yield scrapy.Request(url, callback=self.parse, cb_kwargs={'page': 1})
14
15   def parse(self, response, **kwargs):
16       self.logger.info('A response from %s just arrived!, args: %s', response.url,
         kwargs)
17
18       # base_url = get_base_url(response)
19       item = LianjiaItem()
20
21       for each in response.xpath("/html/body/div[4]/div[1]/ul/li/div[1]"):
22           # self.logger.info('each: %s', each)
23           item['name'] = each.xpath("div[2]/div/a[1]/text()").get().strip()
24           # self.logger.info('name: %s', item['name'])
25           item['price'] = each.xpath("div[6]/div[1]/span/text()").get().strip() +
```

```
                each.xpath("div[6]/div[1]/i[2]/text()").get().strip()
26
27              item['area'] = each.xpath("div[3]/div/text()").get().split('|')[1].strip()
28              # self.logger.info('area: %s', item['area'])
29              item['unit_price'] = each.xpath("div[6]/div[2]/span/text()").get().strip()
30              item['place'] = Pinyin_to_City[response.url.split('/')[4]]
31              if (item['name'] and item['price'] and item['area'] and
                    item['unit_price']):
32                  self.logger.info('%s %s %s %s', item['name'], item['price'],
                        item['area'], item['unit_price'])
33                  yield item
34
35          next_page =
                response.xpath("/html/body/div[4]/div[1]/div[7]/div[2]/div/a[last()]/@href").get()
36
37          self.logger.info('next_page: %s', next_page)
38
39
40          if (kwargs.get('page') == 5):
41              pass
42          else:
43              yield response.follow(next_page, callback=self.parse, cb_kwargs={'page':
                    kwargs.get('page')+1})
```

我这里还进行了一些处理，用来区分它所在的地区。

`Pinyin_to_City[response.url.split('/')[4]]` 就是把dongcheng,xicheng等转化为文字

2.3 pipelines 的编写

pipeline 中我们接收 Item 并把它保存下来

代码 2-6 pipeline 的编写

```
1  class SpiderPipeline:
2      def open_spider(self, spider):
3          try:
4              self.file = open('data.json', 'w', encoding='utf-8')
5              self.result = {'海淀': [], '东城': [], '西城': [], '朝阳': []}
6          except Exception as e:
7              print(e)
8
9      def process_item(self, item, spider):
```

```
        dict_item = ItemAdapter(item).asdict()
        # self.logger.info(dict_item)
        place = dict_item['place']
        # print(place)
        dict_item.pop('place')
        self.result[place].append(dict_item)
        # self.file.write(json_str)
        return item

    def close_spider(self, spider):
        self.file.write(json.dumps(self.result, ensure_ascii=False, indent=4,
            separators=(',', ': ')))
        self.file.close()
```

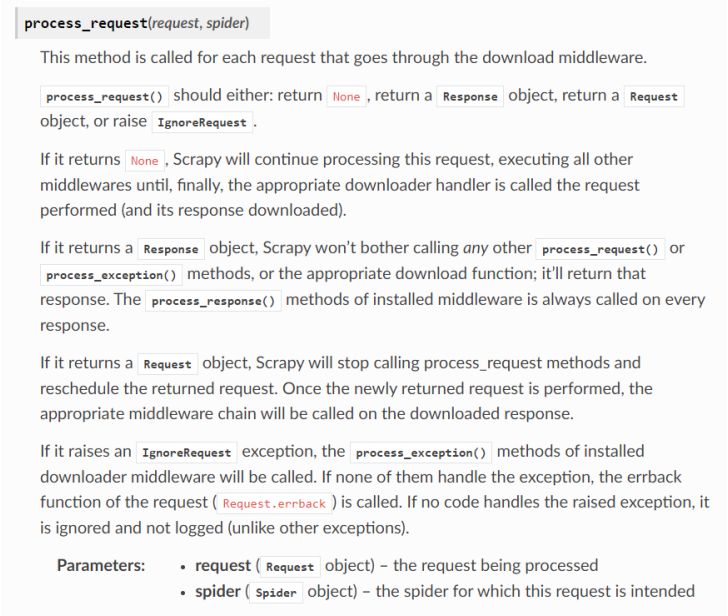
我这里是整体保存的，将所有数据放入 self.result 中, 在`close_spider`中写入 json 文件。

这里有一些要注意的点, `json.dumps()` 中的 `indent=4, separators=(',', ': ')` 中的 indent 是缩进，separators 是分隔符。这两个是用来格式化输出

`ensure_ascii=False`这个可以让中文直接以中文显示而不是用编码格式显示。

2.4 middlewares 的编写

由于之前我提到过我要爬取动态网页，所有我要执行 js 代码，这里我编写了 DownloaderMiddleware 中的 `process_request`。用于处理 request。



process_request(*request*, *spider*)

This method is called for each request that goes through the download middleware.

`process_request()` should either: return `None`, return a `Response` object, return a `Request` object, or raise `IgnoreRequest`.

If it returns `None`, Scrapy will continue processing this request, executing all other middlewares until, finally, the appropriate downloader handler is called the request performed (and its response downloaded).

If it returns a `Response` object, Scrapy won't bother calling *any* other `process_request()` or `process_exception()` methods, or the appropriate download function; it'll return that response. The `process_response()` methods of installed middleware is always called on every response.

If it returns a `Request` object, Scrapy will stop calling process_request methods and reschedule the returned request. Once the newly returned request is performed, the appropriate middleware chain will be called on the downloaded response.

If it raises an `IgnoreRequest` exception, the `process_exception()` methods of installed downloader middleware will be called. If none of them handle the exception, the errback function of the request (`Request.errback`) is called. If no code handles the raised exception, it is ignored and not logged (unlike other exceptions).

Parameters:
- **request** (`Request` object) – the request being processed
- **spider** (`Spider` object) – the spider for which this request is intended

图 2-4 编写 DownloaderMiddleware

根据文档可以得知如果`process_request`返回一个 response 那么就能跳过其他的 request 和异常处理，这样就直接把 response 返回给 Spider

为了调用浏览器，需要安装 Chrome 浏览器，chromedriver，并用 pip 安装 selenium。这里要注意 chromedriver 版本要和 Chrome 版本一致。

我这里定义了一系列的 option, 其中

- `headless` 可以让浏览器不显示。
- `disable-gpu` 可以让浏览器不使用 GPU。
- `no-sandbox` 可以让浏览器不使用沙盒模式。
- `disable-blink-features=AutomationControlled` 可以使浏览器移除掉 webdriver 的痕迹。
- `add_experimental_option('excludeSwitches', ['enable-automation'])`也是移除掉 webdriver 的痕迹。

`driver = webdriver.Chrome(chrome_options=option)`是使用我们的 option 打开了一个浏览器。`driver.execute_cdp_cmd` 这一句是在浏览器执行前插入 js，目的也是为了避免检测。

这些都配置完之后，我用 driver 请求了 request.url，等待了 5s 后获取其内容保存到 html 中，关闭 driver 并且返回 response。

代码 2-7 DownloaderMiddleware 的编写

```
def process_request(self, request, spider):
    # Called for each request that goes through the downloader
    # middleware.
    option = webdriver.ChromeOptions()
    option.add_argument('--headless')
    option.add_argument('--disable-gpu')
    option.add_argument('no-sandbox')
    option.add_argument("disable-blink-features=AutomationControlled")
    option.add_experimental_option('excludeSwitches', ['enable-automation'])
    driver = webdriver.Chrome(chrome_options=option)
    driver.execute_cdp_cmd("Page.addScriptToEvaluateOnNewDocument", {
        "source": """
            Object.defineProperty(navigator, 'webdriver', {
                get: () => undefined
            })
        """
    })
    driver.get(request.url)
    driver.implicitly_wait(5)
    html = driver.page_source
```

```
    driver.quit()
    return HtmlResponse(url=request.url, body=html, request=request, encoding='utf-8')
```

2.5 settings 的配置

最后，我们只需要在 settings 中启用我们的 DownloaderMiddleware 和 ItemPipeline。

代码 2-8 settings 的配置

```
DOWNLOADER_MIDDLEWARES = {
    'Spider.middlewares.SpiderDownloaderMiddleware': 543,
}
ITEM_PIPELINES = {
   'Spider.pipelines.SpiderPipeline': 300,
}
```

这里要注意名字一定要和自己写的对应

2.6 运行与结果

我们可以直接在项目目录下运行`scrapy crawl lianjia`，这样就可以得到结果。当然为了简便也可以写一个 python 脚本用于启动。

代码 2-9 运行与结果

```
from scrapy import cmdline
cmdline.execute("scrapy crawl lianjia".split()) # 启动爬虫
```

由于我进行了格式化输出，使得我的结果比较长，这里我只列出来一小部分。更多的部分可以在https://github.com/NekoMio/Python-Spider/blob/lianjia/data.json中看到

```
"海淀": [
    {
        "name": "韦伯豪家园",
        "price": "2500万",
        "area": "231.2平米",
        "unit_price": "108,132元/平"
    },
    {
        "name": "复兴路79号院",
        "price": "540万",
        "area": "66.4平米",
        "unit_price": "81,326元/平"
    },
    {
```

```
        "name": "金隅山墅",
        "price": "1600万",
        "area": "224.81平米",
        "unit_price": "71,172元/平"
    },
    {
        "name": "中关村南大街乙8号",
        "price": "1314万",
        "area": "95.54平米",
        "unit_price": "137,535元/平"
    },
    {
        "name": "会城门小区",
        "price": "585万",
        "area": "62平米",
        "unit_price": "94,355元/平"
    },
    {
        "name": "水科院",
        "price": "850万",
        "area": "73.2平米",
        "unit_price": "116,121元/平"
    },
    {
        "name": "翠微南里",
        "price": "580万",
        "area": "54.29平米",
        "unit_price": "106,834元/平"
    },
    {
        "name": "金庄1号院",
        "price": "698万",
        "area": "77.7平米",
        "unit_price": "89,833元/平"
    },
    {
        "name": "昌运宫",
        "price": "680万",
        "area": "72.6平米",
        "unit_price": "93,664元/平"
    },
    {
        "name": "恩济里小区",
```

```
        "price": "585万",
        "area": "63.4平米",
        "unit_price": "92,272元/平"
    },
    {
        "name": "万泉河62号院",
        "price": "1320万",
        "area": "104.5平米",
        "unit_price": "126,316元/平"
    },
    {
        "name": "学知园",
        "price": "938万",
        "area": "111.9平米",
        "unit_price": "83,825元/平"
    },
    {
        "name": "复兴路77号院",
        "price": "468万",
        "area": "52.5平米",
        "unit_price": "89,143元/平"
    },
    {
        "name": "金沟河2号院",
        "price": "475万",
        "area": "56.4平米",
        "unit_price": "84,220元/平"
    },
    {
        "name": "恩济里小区",
        "price": "690万",
        "area": "76.78平米",
        "unit_price": "89,868元/平"
    },
    {
        "name": "玉海园三里",
        "price": "593万",
        "area": "72.65平米",
        "unit_price": "81,625元/平"
    },
    {
        "name": "梅园甲",
        "price": "820万",
```

```
        "area": "105.32平米",
        "unit_price": "77,858元/平"
    },
    {
        "name": "满庭芳园",
        "price": "1420万",
        "area": "132.71平米",
        "unit_price": "107,001元/平"
    },
    {
        "name": "明光北里",
        "price": "570万",
        "area": "61.4平米",
        "unit_price": "92,834元/平"
    },
    {
        "name": "门头馨村北二区",
        "price": "520万",
        "area": "92.89平米",
        "unit_price": "55,981元/平"
    },
    {
        "name": "金雅园南区",
        "price": "1360万",
        "area": "141.85平米",
        "unit_price": "95,876元/平"
    },
    {
        "name": "北太平庄路2号院",
        "price": "930万",
        "area": "97.65平米",
        "unit_price": "95,239元/平"
    },
    {
        "name": "明光村小区",
        "price": "830万",
        "area": "79.5平米",
        "unit_price": "104,403元/平"
    },
    {
        "name": "怡美家园",
        "price": "1080万",
        "area": "122.39平米",
```

```
        "unit_price": "88,243元/平"
    },
    {
        "name": "清缘里中区",
        "price": "598万",
        "area": "76.43平米",
        "unit_price": "78,242元/平"
    },
    {
        "name": "西三环北路27号院",
        "price": "705万",
        "area": "67.4平米",
        "unit_price": "104,600元/平"
    },
    {
        "name": "燕北园",
        "price": "820万",
        "area": "81.6平米",
        "unit_price": "100,491元/平"
    },
    {
        "name": "西单宿舍",
        "price": "680万",
        "area": "61.3平米",
        "unit_price": "110,930元/平"
    },
    {
        "name": "强佑•府学上院",
        "price": "1165万",
        "area": "139.58平米",
        "unit_price": "83,465元/平"
    },
]
```

第三章　爬取学堂在线

本代码的 github 链接为https://github.com/NekoMio/Python-Spider/tree/xuetang 项目中包含代码和完整结果。

3.1　Item 类的编写

与上一章中的 Item 类几乎完全一样，针对要求编写即可。

代码 3-1 Item 类的编写

```
class XuetangItem(scrapy.Item):
    # define the fields for your item here like:
    name = scrapy.Field()
    teacher = scrapy.Field()
    school = scrapy.Field()
    student_num = scrapy.Field()
    pass
```

3.2　spider 的编写

spider 的内容也与上一章中的 spider 类几乎完全一样。但是需要针对它重新分析 XPath

这里有一个问题是某些课程没有教师和学校，不能简单的通过 xpath 获取，需要进行判断。我这里使用了一个循环然后根据 class 的类型来判断是什么类型。

同时，对于信息不全的，我们也应该保存下来进行处理。

代码 3-2 spider 的编写

```
def parse(self, response, **kwargs):

    # base_url = get_base_url(response)
    item = XuetangItem()

    for each in
        response.xpath("/html/body/div[1]/div/div[2]/div[1]/div[1]/div[2]/div[1]/div"):
        # self.logger.info('each: %s', each)
        item['name'] = each.xpath("div[2]/p[1]/span[1]/text()")
        item['teacher'] = ""
        item['school'] = ""
        if (item['name']):
            item['name'] = item['name'].get().strip()
            self.logger.info('name: %s', item['name'])
```

```
14          for each_span in each.xpath("div[2]/p[2]/*"):
15              self.logger.info('each_span: %s', each_span)
16              if each_span.xpath("@class").get() == "teacher_con":
17                  teacher_string = ""
18                  for teacher_each in each_span.xpath("span"):
19                      teacher_string = teacher_string +
                            teacher_each.xpath("text()").get()
20                  item['teacher'] = teacher_string
21                  self.logger.info('teacher: %s', item['teacher'])
22              elif each_span.xpath("@class").get() == "org_con":
23                  item['school'] = each_span.xpath("span/text()").get()
24                  self.logger.info('school: %s', item['school'])
25              else:
26                  item['student_num'] = each_span.xpath("text()").get().strip()
27                  self.logger.info('student_num: %s', item['student_num'])
28          if (item['name']):
29              yield item
30
31      if response.url != None:
32          yield scrapy.Request(response.url, callback=self.parse)
```

同时，这里还有一个小问题，那就是教师可能不止一个，还需要一层循环相加即可。

那么，对于这个网站，如何获取下一页的值呢？

我认为这个是和上一个最大的区别，这个页面中下一页的地址并不是通过 a 标签实现的，而且对按钮绑定了 js 事件，虽然我们仍然可以通过直接更改地址来获取下一页的值，但是我感觉这样不是很好，降低了通用性。

可以看到我这里获取下一页的时候直接使用了 response.url，这个是因为我在中间件中直接将下一页的地址存到了 response.url 中。这个是我唯一能找到的用来传递链接的地方。而我在中间件中如何获取下一页地址，可以看 middlewares 的编写 一节。

3.3 pipelines 的编写

这里的 pipelines 类也与上一章中的 pipelines 类也很相似。只不过这里需要引用 csv 的包并以 csv 的格式进行输出。

不过这里有一个神奇的地方，我这里的编码格式的`utf_8_sig`而不是上一章节的 utf-8。这里使用了 UTF-8 With BOM。这个是为了让微软的 excel 能够正确识别中文。

代码 3-3 pipelines 的编写

```
1   class SpiderPipeline:
2       def open_spider(self, spider):
3           try:
```

```
            self.file = open('data.csv', 'w', encoding='utf_8_sig')
            self.result = csv.writer(self.file)
            self.result.writerow(['name', 'teacher', 'school', 'student_num'])
        except Exception as e:
            print(e)

    def process_item(self, item, spider):
        dict_item = ItemAdapter(item).asdict()
        self.result.writerow(dict_item.values())
        return item

    def close_spider(self, spider):
        self.file.close()
```

3.4 middlewares 的编写

中间件的处理逻辑与上一章也是相似的，但是这里我又遇到了一点问题，首先，我这里的环境是纯英文的环境，学堂在线识别到我的语言是英文，所以默认切换到了英文的网页。如果是本地又英文环境的话，是没有这个问题的。

为了解决这个问题，我添加了`intl.accept_languages`这个参数，用来指明我的语言，但是很遗憾的是 Chrome 并不支持这个参数，所以我被迫切换到了 Firefox 进行配置。

同时。在这里，我需要为 Spider 找到下一页的地址，我是通过 XPath 获取到按钮，然后进行模拟点击，点击后，浏览器将进入下一页，这时候，我获取他的地址，就是我下一页的地址，并将其传入 response.url 中，返回给 Spider。

这个方法虽然我认为依然不过优美，但是它相对满足了功能隔离，并且通过简单的方法实现了我的需求。对整体代码的修改量也很小，所以我使用了这样的方法。

代码 3-4 middlewares 的编写

```
def process_request(self, request, spider):
    # Called for each request that goes through the downloader
    # middleware.
    option = webdriver.FirefoxOptions()
    ffpf = webdriver.FirefoxProfile()
    ffpf.set_preference("intl.accept_languages", "zh-CN")
    option.add_argument('--headless')
    option.add_argument('--disable-gpu')
    driver = webdriver.Firefox(firefox_profile=ffpf, options=option)
    driver.get(request.url)
    driver.implicitly_wait(5)
```

```
12      html = driver.page_source
13      try:
14          next_page = driver.find_element_by_xpath(
15              '/html/body/div[1]/div/div[2]/div[1]/div[1]/div[2]/div[2]/button[2]'
16          )
17          next_page.click()
18          driver.implicitly_wait(2)
19          next_url = driver.current_url
20          spider.logger.info(next_url)
21      except:
22          next_url = None
23
24      driver.quit()
25      return HtmlResponse(url=next_url, body=html, request=request, encoding='utf-8')
```

3.5 settings 的配置

这个 Settings 类与上一章中的 Settings 没有任何区别

3.6 运行与结果

我们可以直接在项目目录下运行`scrapy crawl xuetang`，这样就可以得到结果。

这里我只列出来一小部分。更多的内容可以在https://github.com/NekoMio/Python-Spider/blob/xuetang/data.csv中看到

```
1  name,teacher,school,student_num
2  C++语言程序设计基础,郑莉 李超 等,清华大学,466951 人
3  数据结构(上),邓俊辉,清华大学,456263 人
4  数据结构（下）,邓俊辉,清华大学,381166 人
5  操作系统,向勇 陈渝,清华大学,218933 人
6  Java程序设计,郑莉,清华大学,213250 人
7  网络技术与应用,沈鑫剡 俞海英 等,中国人民解放军陆军工程大学,199250 人
8  C++语言程序设计进阶,郑莉 李超 等,清华大学,135116 人
9  C程序设计案例教程（基础）,张莉,中国农业大学,117462 人
10 C程序设计案例教程（进阶）,张莉,中国农业大学,113001 人
11 大数据技术与应用,李军,清华大学,104735 人
12 软件工程,刘强 刘璘,清华大学,103448 人
13 计算机文化基础,李秀 姚瑞霞 等,清华大学,91346 人
14 程序设计基础,徐明星 王瑀屏 等,清华大学,89997 人
15 人工智能原理,王文敏,北京大学,79374 人
16 组合数学,马昱春,清华大学,78963 人
17 大数据系统基础,王建民 徐葳 等,清华大学,78906 人
```

```
Office办公软件应用,史巧硕 朱怀忠 等,河北工业大学,75107 人
算法设计与分析,王振波,清华大学,73184 人
VC++面向对象与可视化程序设计（上）: Windows编程基础,黄维通,清华大学,71679 人
大数据平台核心技术,武永卫 姚文辉 等,清华大学,69720 人
大学计算机基础,徐红云 刘欣欣 等,华南理工大学,52765 人
JAVA程序设计进阶,许斌,清华大学,52683 人
大学计算机——计算思维的视角,郝兴伟,山东大学,52342 人
面向对象程序设计（C++）,黄震春 徐明星,清华大学,51819 人
计算几何,邓俊辉,清华大学,47964 人
R语言数据分析,艾新波,北京邮电大学,45129 人
Python程序设计基础,许志良,深圳信息职业技术学院,44732 人
大数据机器学习,袁春,清华大学,42248 人
数据库技术与程序设计,高裴裴 康介恢 等,南开大学,41801 人
程序设计基础（上）,赵宏 闫晓玉 等,南开大学,41224 人
单片机原理及应用,杨居义 王颖丽 等,绵阳职业技术学院,40851 人
Web开发技术,王成良 陈静 等,重庆大学,39558 人
ARM微控制器与嵌入式系统,曾鸣 薛涛 等,清华大学,39450 人
Linux 内核分析与应用,陈莉君 谢宝友 等,西安邮电大学,39233 人
区块链和加密数字货币,罗玫,清华大学,38384 人
程序设计基础（下）,赵宏 李妍 等,南开大学,37441 人
大学计算机基础,李敏 高裴裴 等,南开大学,36834 人
C语言程序设计（上）,李凤霞 薛静锋 等,北京理工大学,33580 人
数据结构与算法设计,张小艳 史晓楠 等,西安科技大学,33457 人
高级大数据系统,王智,清华大学,33089 人
物联网概论,何源,清华大学,32500 人
网络安全概述,纪平,学堂在线,32461 人
大数据治理与政策,孟庆国 张楠 等,清华大学,32141 人
Android应用开发基础,赖红 李钦 等,深圳信息职业技术学院,31402 人
物联网工程导论,普园媛 何乐生 等,云南大学,31060 人
大学计算机,李凤霞 陈宇峰 等,北京理工大学,30091 人
移动快速应用开发,唐贤传 盛鸿宇 等,芜湖职业技术学院,29811 人
人工智能,罗会兰,江西理工大学,28934 人
大学物理,侯岩雪 张素红 等,燕山大学,28409 人
科学计算与MATLAB语言,刘卫国 蔡旭晖 等,中南大学,27954 人
```