第5章

爬取网络中的小说和购物评论

视频讲解

本章将选取两个实用且有趣的主题作为爬虫实践的内容，分别是抓取网络小说的内容和抓取购物评论，对象网站分别是逐浪小说网和京东网。这是两个非常贴近生活的示例，有兴趣的读者可以在本章的基础上实现自己的个人爬虫，为之增添更多的功能。

5.1 下载网络小说

网络文学是新世纪我国流行文化中的重要领域，年轻人对网络小说更是有着广泛的喜爱。前面已经学习了使用 Selenium 自动化浏览器抓取信息的基础，接下来以抓取网络小说正文为例编写一个简单、实用的爬虫脚本。

5.1.1 分析网页

很多人在阅读网络小说时都喜欢本地阅读，换句话说就是把小说下载到手机或者其他移动设备上阅读，这样不仅不受网络的限制，还能够使用阅读 App 调整出自己喜欢的显示风格。但遗憾的是，各大网站很少会提供整部小说的下载功能，只有部分网站会给 VIP 会员开放下载多个章节内容的功能。对于普通读者而言，虽然 VIP 章节需要购买阅读，但是至少还是希望能够把大量的免费章节一口气看完。用户完全可以使用爬虫程序来帮助自己把一个小说的所有免费章节下载到 TXT 文件中，以方便在其他设备上阅读(这里也要提示大家支持正版，远离盗版，提高知识产权意识)。

以逐浪小说网为例，从排行榜中选取一个比较流行的小说(或者是读者感兴趣的)进行分析，首先是小说的主页，其中包括了各种各样的信息(如小说简介、最新章节、读者评论等)，其次是一个章节列表页面(有的网站也称为“最新章节”页面)，而小说的每一章有着单独的页面。很显然，如果用户能够利用章节列表页面来采集所有章节的 URL 地址，那么我们只要用程序分别抓取这些章节的内容，并将内容写入本地 TXT 文件，即可完成小说的抓取。

在查看章节页面之后，用户十分遗憾地发现，小说的章节内容使用 JS 加载，并且整个页面使用了大量的 CSS 和 JS 生成的效果，这给用户的抓取增加了一点难度。使用 requests 或者 urllib 库直接请求章节页面的 URL 是不现实的，但用户可以用 Selenium 来轻松搞定这个问题，对于一个规模不大的任务而言，在性能和时间上的代价还是可以接受的。

接下来分析一下如何定位正文元素。使用开发者模式查看元素(见图 5-1)，用户发现可以使用 read-content 这个 ID 的值定位到正文。不过 class 的值也是 read-content，在理论上似乎可以使用 class 名定位，但 Selenium 目前还不支持复合类名的直接定位，所以使用 class 来

定位的想法只能先作罢。

图 5-1　开发者模式下的小说章节内容

【提示】 虽然 Selenium 目前只支持对简单类名的定位，但是用户可以使用 CSS 选择的方式对复合类名进行定位，有兴趣的读者可以了解 Selenium 中的 find_element_by_css_selector()方法。

5.1.2　编写爬虫

使用 Selenium 配合 Chrome 进行本次抓取，除了用 pip 安装 Selenium 之外，首先需要安装 ChromeDriver，可访问以下地址将其下载到本地：

https://sites.google.com/a/chromium.org/chromedriver/downloads

进入下载页面后(见图 5-2)，根据自己系统的版本进行下载即可。

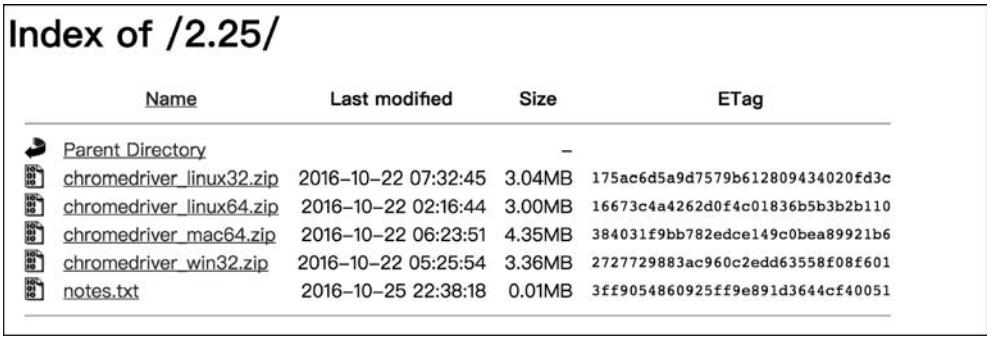

图 5-2　ChromeDriver 的下载页面

之后，使用 selenium.webdriver.Chrome(path_of_chromedriver)语句可创建 Chrome 浏览器对象，其中 path_of_chromedriver 就是下载的 ChromeDriver 的路径。

在脚本中，用户可以定义一个名为 NovelSpider 的爬虫类，使用小说的"全部章节"页面 URL 进行初始化(类似于 C 语言中的"构造")，同时它还拥有一个 list 属性，其中将会存放各个章节的 URL。类方法如下。

- get_page_urls()：从全部章节页面抓取各个章节的 URL。

- get_novel_name()：从全部章节页面抓取当前小说的书名。
- text_to_txt()：将各个章节中的文字内容保存到 TXT 文件中。
- looping_crawl()：循环抓取。

思路梳理完毕后就可以着手编写程序了，最终的爬虫代码见例 5-1。

【例 5-1】 网络小说的爬取程序。

```
# NovelSpider.py
import selenium.webdriver, time, re
from selenium.common.exceptions import WebDriverException


class NovelSpider():
  def __init__(self, url):
    self.homepage = url
    self.driver = selenium.webdriver.Chrome(path_of_chromedriver)
    self.page_list = []

  def __del__(self):
    self.driver.quit()
  def get_page_urls(self):
    homepage = self.homepage
    self.driver.get(homepage)
    self.driver.save_screenshot('screenshot.png')

    self.driver.implicitly_wait(5)
    elements = self.driver.find_elements_by_tag_name('a')

    for one in elements:
      page_url = one.get_attribute('href')

      pattern = '^http:\/\/book\.zhulang\.com\/\d{6}\/\d+\.html'
      if re.match(pattern, page_url):
        print(page_url)
        self.page_list.append(page_url)
  def looping_crawl(self):
    homepage = self.homepage
    filename = self.get_novel_name(homepage) + '.txt'
    self.get_page_urls()
    pages = self.page_list
    # print(pages)

    for page in pages:
      self.driver.get(page)
      print('Next page:')

      self.driver.implicitly_wait(3)
      title = self.driver.find_element_by_tag_name('h2').text
      res = self.driver.find_element_by_id('read-content')
      text = '\n' + title + '\n'
      for one in res.find_elements_by_xpath('./p'):
        text += one.text
        text += '\n'

      self.text_to_txt(text, filename)
```

```
            time.sleep(1)
            print(page + '\t\t\tis Done!')

        def get_novel_name(self, homepage):

            self.driver.get(homepage)
            self.driver.implicitly_wait(2)

            res = self.driver.find_element_by_tag_name('strong').find_element_by_xpath('./a')
            if res is not None and len(res.text) > 0:
                return res.text
            else:
                return 'novel'

        def text_to_txt(self, text, filename):
            if filename[-4:]!= '.txt':
                print('Error, incorrect filename')
            else:
                with open(filename, 'a') as fp:
                    fp.write(text)
                    fp.write('\n')


    if __name__ == '__main__':
        hp_url = input('输入小说"全部章节"页面: ')

        path_of_chromedriver = 'your_path_of_chrome_driver'
        try:
            sp1 = NovelSpider(hp_url)
            sp1.looping_crawl()
            del sp1
        except WebDriverException as e:
            print(e.msg)
```

__init__()和__del__()方法可以视为构造函数和析构函数，分别在对象被创建和被销毁时执行。在__init__()中使用一个URL字符串进行了初始化，而在__del__()方法中退出了Selenium浏览器。try-except语句执行主体部分并尝试捕获WebDriverException异常（这也是Selenium运行时最常见的异常类型）。在lopping_crawl()方法中则分别调用了上述其他几个方法。

driver.save_screenshot()方法是selenium.webdriver中保存浏览器当前窗口截图的方法。

driver.implicitly_wait()方法是Selenium中的隐式等待，它设置了一个最长等待时间，如果在规定的时间内网页加载完成，则执行下一步，否则一直等到时间截止，然后再执行下一步。

【提示】 显式等待会等待一个确定的条件触发然后才进行下一步，可以结合ExpectedCondition共同使用，支持自定义各种判定条件。隐式等待在编写时只需要一行，所以编写十分方便，其作用范围是WebDriver对象实例的整个生命周期，会让一个正常响应的应用的测试变慢，导致整个测试执行的时间变长。

driver.find_elements_by_tag_name()是Selenium用来定位元素的诸多方法之一，所有定位单个元素的方法如下。

- find_element_by_id()：根据元素的id属性来定位，返回第一个id属性匹配的元素；

如果没有元素匹配，会抛出 NoSuchElementException 异常。

- find_element_by_name()：根据元素的 name 属性来定位，返回第一个 name 属性匹配的元素；如果没有元素匹配，则抛出 NoSuchElementException 异常。
- find_element_by_xpath()：根据 XPath 表达式定位。
- find_element_by_link_text()：用链接文本定位超链接。该方法还有子串匹配版本 find_element_by_partial_link_text()。
- find_element_by_tag_name()：使用 HTML 标签名来定位。
- find_element_by_class_name()：使用 class 定位。
- find_element_by_css_selector()：根据 CSS 选择器定位。

寻找多个元素的方法名只是将 element 变为复数 elements，并返回一个寻找的结果（列表），其余和上述方法一致。在定位到元素之后，可以使用 text()和 get_attribute()方法获取其中的文本或各个属性。

```
page_url = one.get_attribute('href')
```

这行代码使用 get_attribute()方法来获取定位到的各章节的 URL 地址。在以上程序中还使用了 re（Python 的正则模块）中的 re.match()方法，根据正则表达式来匹配 page_url。形如：

```
'^http:\/\/book\.zhulang\.com\/\d{6}\/\d+\.html'
```

这样的正则表达式所匹配的是下面这样一种字符串：

```
http://book.zhulang.com/A/B/.html
```

其中，A 部分必须是 6 个数字，B 部分必须是一个以上的数字。这也正好是小说各个章节页面的 URL 形式，只有符合这个形式的 URL 链接才会被加入 page_list 中。

re 模块的常用函数如下。

- compile()：编译正则表达式，生成一个 Pattern 对象。之后就可以利用 Pattern 的一系列方法对文本进行匹配/查找（当然，匹配/查找函数也支持直接将 Pattern 表达式作为参数）。
- match()：用于查找字符串的头部（也可以指定起始位置），它是一次匹配，只要找到了一个匹配的结果就返回。
- search()：用于查找字符串的任何位置，只要找到了一个匹配的结果就返回。
- findall()：以列表形式返回能匹配的全部子串，如果没有匹配，则返回一个空列表。
- finditer()：搜索整个字符串，获得所有匹配的结果。与 findall()的一大区别是，它返回一个顺序访问每一个匹配结果（Match 对象）的迭代器。
- split()：按照能够匹配的子串将字符串分割后返回一个结果列表。
- sub()：用于替换，将母串中被匹配的部分使用特定的字符串替换掉。

【提示】 *正则表达式在计算机领域中应用广泛，读者有必要好好了解一下它的语法。*

在 looping_crawl()方法中分别使用了 get_novel_name()获取书名并转换为 TXT 文件名，get_page_urls()获取章节页面的列表，text_to_txt()保存抓取到的正文内容。在这之间还大量使用了各类元素定位方法（如上文所述）。

5.1.3 运行并查看 TXT 文件

这里选取一个小说——逐浪小说网的《绝世神通》，运行脚本并输入其章节列表页面的

URL，可以看到控制台中程序成功运行时的输出，如图 5-3 所示。

```
Next page:
http://book.zhulang.com/344033/298426.html          is Done!
Next page:
http://book.zhulang.com/344033/218044.html          is Done!
Next page:
http://book.zhulang.com/344033/219747.html          is Done!
Next page:
http://book.zhulang.com/344033/220347.html          is Done!
Next page:
http://book.zhulang.com/344033/221904.html          is Done!
Next page:
http://book.zhulang.com/344033/221907.html          is Done!
Next page:
http://book.zhulang.com/344033/223892.html          is Done!
Next page:
http://book.zhulang.com/344033/223893.html          is Done!
Next page:
http://book.zhulang.com/344033/225854.html          is Done!
Next page:
http://book.zhulang.com/344033/225856.html          is Done!
Next page:
```

图 5-3　小说爬虫的输出

抓取结束后，用户可以发现目录下多出一个名为“screenshot.png”的图片（见图 5-4）和一个“绝世神通.txt”文件（见图 5-5），小说《绝世神通》的正文内容（按章节顺序）已经成功保存。

图 5-4　逐浪小说网的屏幕截图

绝世神通第三十六章　轩炎的杀意
“这是，下品灵器。”秦萧微一愣。
灵器亦也是下中上极品之分，极品之上还有王品，跟武技功法的等级划分是一样的。
不过下品的灵器可比下品的武技要珍贵上不少，以至于诺大的秦家都只有一件下品灵器，就是他父亲秦鼎天的剑。
下品之下，都称为凡品，皆是不入流之物。
武器的作用，在战斗之中还是很能够体现出来的。就比如刚才，如果秦萧手中的是一柄下品的灵器剑，那刚才一剑不说百分百能够将地头剑齿虎杀死的话，至少也能让它基本失去反抗能力。再补一剑，自然就可以杀死了。而不是，还会让剑齿虎逃走的份。
白剑求道：“送给你吧，一件对我无用之物，却能赚你一份人情，哈哈。一件好的兵器相助，会让你更加得心应手，尤其是对付荒兽。不过你的剑，我总感觉有些特殊吧？”
“特殊？”秦萧眉头微一皱，其实他也感觉有些特殊，因为上次跟严公子一战，他明显感觉从自己的剑中有一股诡异的力量涌了出来。
只是，他后来也好好地研究了一下，实在是看不出来有什么特殊之处。
这剑跟了他这么多年了，也就是那次有些诡异。
“或许吧，不过我也不知道。这剑是我母亲留给我的遗物，不对付荒兽的话，我还是习惯用这剑的。白哥，你的剑我就先收下了，欠你的几分人情，以后再还你。”秦萧道。
白剑求一笑：“我就跟你开个玩笑，别当真。好了，我们走吧。再往里面，就危险了，你可得小心一点。甚至有可能，连我都难照顾到你。”

图 5-5　小说的部分内容

程序圆满地完成了下载小说的任务，缺点是耗时有些久，而且 Chrome 占用了大量的硬件资源。对于动态网页，其实不一定必须使用浏览器模拟的方式来爬取，可以尝试用浏览器开发

者工具分析网页的请求，获取到接口后通过请求接口的方式请求数据，不再需要 Selenium 作为“中介”。另外，对于获得的屏幕截图而言，图片是窗口截图，而不是整个页面的截图（长图），为了获得整个页面的截图或者部分页面元素的截图，用户需要使用其他方法，如注入 JavaScript 脚本等，本节就不再展开介绍。

5.2 下载购物评论

现今，在线购物平台已经成为人们生活中不可或缺的一部分，从淘宝、天猫、京东到当当，很难想象离开了这些网购平台人们的生活会缺失多少便利。无论对于普通消费者还是商家而言，商品评论都是十分有用的信息，消费者可以从他人的评论中分析出商品的质量，商家也可以根据评论调整生产与商业策略。本节以著名的网购平台——京东为例，看看如何抓取特定商品的评论信息。

5.2.1 查看网络数据

首先进入京东官网，单击并进入一个感兴趣的商品页面。这里以图书《解忧杂货店》的页面为例，在浏览器中查看（见图 5-6）。

图 5-6 京东商品页面

单击“商品评价”标签，可以查看以一页一页的文字形式所呈现的评价内容。既然想编写程序把这些评价内容抓取下来，那么就应该考虑这次使用什么手段和工具。在之前的小说内容抓取中使用了 Selenium 浏览器自动化的方式，通过加载每一章节对应页面的内容来抓取，对于商品评论而言，这个策略看起来应该是没有问题的，毕竟 Selenium 的特色就是可以执行对页面的交互。不过，这次不妨从更深层的角度思考，仅以简单的 requests 来搞定这个任务。

一般来说，在网购平台的页面中会大量使用 AJAX，因为这样就可以实现网页数据的局部刷新，避免了加载整个页面的负担，对于商品评论这种变动频繁、时常刷新的内容而言尤其如此。用户可以尝试直接使用 requests 请求页面并使用 lxml 的 XPath 定位来抓取一条评论。

首先使用 Chrome 的开发者模式检查元素并获得其 XPath，见图 5-7。

然后用几行代码检查一下是否能直接用 requests 请求页面并获得这条评论，代码如下（不要忘了在.py 文件开头使用 import 导入相关的包）：

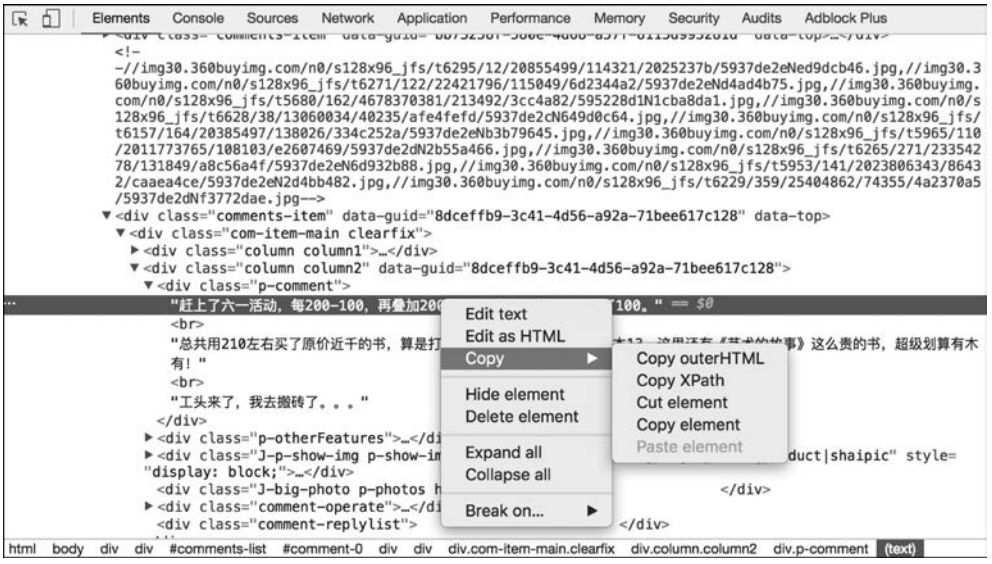

图 5-7　Chrome 检查评论内容

```
if__name__ == '__main__':
  xpath_raw = '//*[@id="comment-0"]/div[1]/div[2]/div/div[2]/div[1]/text()[1]'
  url = input("输入商品链接: ")
  response = requests.get(url)
  ht1 = lxml.html.fromstring(response.text)
  print(ht1.xpath(xpath_raw))
```

输入商品链接“https://item.jd.com/11452840.html#comment”后，果不其然，获得的结果是“[]”。换句话说，这个简单粗暴的策略并不能抓取到评论内容。为保险起见，观察一下 requests 请求到的页面内容，在代码的最后加上如下两行：

```
with open('jd_item.html','w') as fp:
  fp.write(response.text)
```

这样就可以把 response 的 text 内容直接写入 jd_item.html 文件，再次运行后，使用编辑器打开文件，找到商品评论区域，只看到了几个大大的“加载中”：

```
...
<div id="comment-0" class="mc ui-switchable-panel comments-table">
    <div class="loading-style1"><b></b>加载中，请稍候...</div>
</div>
<div id="comment-1" class="mc none ui-switchable-panel comments-table">
    <div class="loading-style1"><b></b>加载中，请稍候...</div>
</div>
<div id="comment-2" class="mc none ui-switchable-panel comments-table">
    <div class="loading-style1"><b></b>加载中，请稍候...</div>
</div>
<div id="comment-3" class="mc none ui-switchable-panel comments-table">
    <div class="loading-style1"><b></b>加载中，请稍候...</div>
</div>
<div id="comment-4" class="mc none ui-switchable-panel comments-table">
    <div class="loading-style1"><b></b>加载中，请稍候...</div>
</div>
...
```

看来商品的评论属于动态内容，直接请求 HTML 页面是抓取不到的，用户只能另寻他法。之前提到可以使用 Chrome 的 Network 工具来查看与网站的数据交互，所谓的数据交互，当然也包括 AJAX 内容。

首先单击页面中的“商品评价”按钮，之后打开 Network 工具。鉴于用户并不关心 JS 数据之外的其他繁杂信息，为了保持简洁，可以使用过滤器工具并选中 JS 选项。不过，可能会有读者发现这时并没有在显示结果中看到对应的信息条目，这种情况可能是因为在 Network 工具开始记录信息之前评论数据就已经加载完毕。碰到这种情况，直接单击“下一页”查看第 2 页的商品评论即可，这时可以直观地看到有一条 JS 数据加载信息被展示出来，如图 5-8 所示。

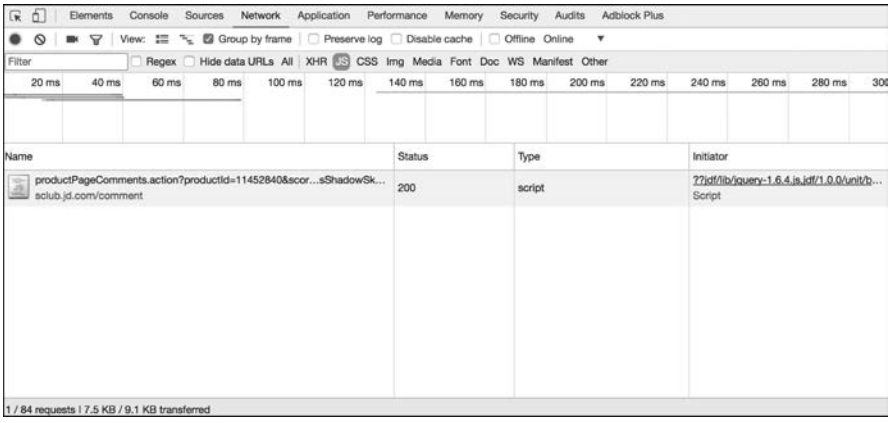

图 5-8　Network 工具查看 JS 请求信息

单击图 5-9 中的这条记录，在它的 Headers 选项卡中便是有关其请求的具体信息，用户可以看到它请求的 URL 为 https://sclub.jd.com/comment/productPageComments.action?productId=11452840&score=0&sortType=3&page=1&pageSize=10&isShadowSku=0&callback=fetchJSON_comment98vv110378，状态为 200（即请求成功，没有任何问题）。在右侧的 Preview 选项卡中可以预览其中所包含的评论信息。不妨分析一下这个 URL 地址，显然，“?”之后的内容都是参数，访问这个 API 会使得对应的后台函数返回相关的 JSON 数据。其中，productId 的值正好就是商品页面 URL 中的编号，可见这是一个确定商品的 ID 值。如果将其中一个参数进行修改，如将 page 改为 5，并在浏览器中访问，得到了不一样的信息（见图 5-9），说明大家的猜测是正确的，在接下来的爬虫编写中只需要更改对应的参数即可。

图 5-9　更改参数后访问 URL 的效果

5.2.2　编写爬虫

在动手编写爬虫之前可以先设想一下.py 脚本的结构，为方便起见，使用一个类作为商品

评论页面的抽象表示，其属性应该包括商品页面的链接和抓取到的所有评论文本（作为一个字符串）。为了输出和调试方便，还应该加入日志功能，编写类方法 get_comment_from_item_url()作为访问数据并抓取的主体，同时还应该有一个类方法用来处理抓取到的数据，不如称为 content_process()（意为“内容处理”）。还可以将评论信息中的几项关键内容（如评论文字、日期时间、用户名、用户客户端等）保存到 CSV 文件中以备日后查看和使用。出于以上考虑，爬虫类可以编写为例 5-2 中的代码。

【例 5-2】 JDComment 类的雏形。

```
class JDComment():
  _itemurl = ''

  def __init__(self, url):
    self._itemurl = url
    logging.basicConfig(
      level = logging.INFO,
    )
    self.content_sentences = ''

  def get_comment_from_item_url(self):

    comment_json_url = 'https://sclub.jd.com/comment/productPageComments.action'
    p_data = {
      'callback': 'fetchJSON_comment98vv110378',
      'score': 0,
      'sortType': 3,
      'page': 0,
      'pageSize': 10,
      'isShadowSku': 0,
    }
    p_data['productId'] = self.item_id_extracter_from_url(self._itemurl)

    ses = requests.session()

    while True:
      response = ses.get(comment_json_url, params = p_data)
      logging.info('-' * 10 + 'Next page!' + '-' * 10)
      if response.ok:
        r_text = response.text
        r_text = r_text[r_text.find('({') + 1:)
        r_text = r_text[:r_text.find(');'])
        js1 = json.loads(r_text)

        for comment in js1['comments']:
          logging.info('{}\t{}\t{}\t{}'.format(comment['content'], comment['referenceTime'],
comment['nickname'], comment['userClientShow']))

          self.content_process(comment)
          self.content_sentences += comment['content']
      else:
        logging.error('Status NOT OK')
        break
      p_data['page'] += 1
      if p_data['page'] > 50:
        logging.warning('We have reached at 50th page')
```

```
            break

    def item_id_extracter_from_url(self, url):
        item_id = 0

        prefix = 'item.jd.com/'
        index = str(url).find(prefix)
        if index!=-1:
          item_id = url[index + len(prefix): url.find('.html')]

        if item_id!= 0:
          return item_id

      def content_process(self, comment):
        with open('jd-comments-res.csv','a') as csvfile:
          writer = csv.writer(csvfile,delimiter = ',')
          writer.writerow([comment['content'],comment['referenceTime'],
                           comment['nickname'],comment['userClientShow']])
```

在上面的代码中使用 requests.session()来保存会话信息，这样会比单纯的 requests.get()更接近一个真实的浏览器。当然，用户还应该定制 User-Agent 信息，不过由于爬虫程序规模不大，被 ban(封禁)的可能性很低，所以不妨先专注于其他具体功能。

```
logging.basicConfig(
      level = logging.INFO,
    )
```

这几行代码设置了日志功能并将级别设为 INFO，如果想把日志输出到文件而不是控制台，可以在 level 下面加一行“filename='app.log'”，这样日志就会被保存到“app.log”这个文件之中。

p_data 是将要在 requests 请求中发送的参数(params)，这正是在之前的 URL 分析中得到的结果。以后用户只需要更改 page 的值即可，其他参数保持不变。

```
p_data['productId'] = self.item_id_extracter_from_url(self._itemurl)
```

这行代码为 p_data(本身是一个 Python 字典结构)新插入了一项，键为'productId'，值为 item_id_extracter_from_url()方法的返回值。item_id_extracter_from_url()方法接收商品页面的 URL(注意，不是请求商品评论的 URL)并抽取出其中的 productId，而_itemurl(即商品页面 URL)在 JDComment 类的实例创建时被赋值。

```
response = ses.get(comment_json_url, params = p_data)
```

这行代码会向 comment_json_url 请求评论信息的 JSON 数据，接下来大家看到了一个 while 循环，当页码数突破一个上限(这里为 50)时停止循环。在循环中会对请求到的 fetchJSON 数据做少许处理，将它转换成可编码为 JSON 的文本并使用。

```
js1 = json.loads(r_text)
```

这行代码会创建一个名为 js1 的 JSON 对象，然后用户就可以用类似于字典结构的操作来获取其中的信息了。在每次 for 循环中，不仅在 log 中输出一些信息，还使用

```
self.content_process(comment)
```

调用 content_process()方法对每条 comment 信息进行操作，具体就是将其保存到 CSV 文件中。

```
self.content_sentences += comment['content']
```

这样会把每条文字评论加入当前的 content_sentences 中，这个字符串中存放了所有文字评论。不过，在正式运行爬虫之前，用户不妨再多想一步。对于频繁的 JSON 数据请求，最好能够保持一个随机的时间间隔，这样不易被反爬虫机制(如果有的话)ban 掉，编写一个 random_sleep()函数来实现这一点，每次请求结束后调用该函数。另外，使用页码最大值来中断爬虫的做法恐怕还不够合理，既然抓取的评论信息中就有日期信息，完全可以使用一个日期检查函数来共同控制循环抓取的结束——当评论的日期已经早于设定的日期或者页码已经超出最大限制时立刻停止抓取。在变量 content_sentences 中存放着所有评论的文字内容，可以使用简单的自然语言处理技术来分析其中的一些信息，比如抓取关键词。在实现这些功能以后，最终的爬虫程序就完成了，见例 5-3。

【例 5-3】 京东商品评论的爬虫。

```
# JDComment.py
import requests, json, time, logging, random, csv, lxml.html, jieba.analyse
from pprint import pprint
from datetime import datetime


# 京东评论 JS
class JDComment():
  _itemurl = ''

  def __init__(self, url, page):
    self._itemurl = url
    self._checkdate = None
    logging.basicConfig(
      # filename = 'app.log',
      level = logging.INFO,
    )
    self.content_sentences = ''
    self.max_page = page

  def go_on_check(self, date, page):
    go_on = self.date_check(date) and page <= self.max_page
    return go_on

  def set_checkdate(self, date):
    self._checkdate = datetime.strptime(date, '%Y-%m-%d')

  def get_comment_from_item_url(self):

    comment_json_url = 'https://sclub.jd.com/comment/productPageComments.action'
    p_data = {
      'callback': 'fetchJSON_comment98vv242411',
      'score': 0,
      'sortType': 3,
      'page': 0,
```

```
        'pageSize': 10,
        'isShadowSku': 0,
      }

      p_data['productId'] = self.item_id_extracter_from_url(self._itemurl)

      ses = requests.session()

      go_on = True
      while go_on:
        response = ses.get(comment_json_url, params = p_data)
        logging.info('-' * 10 + 'Next page!' + '-' * 10)
        if response.ok:

          r_text = response.text
          r_text = r_text[r_text.find('({') + 1:]
          r_text = r_text[:r_text.find(');')]
          js1 = json.loads(r_text)

          for comment in js1['comments']:
            go_on = self.go_on_check(comment['referenceTime'], p_data['page'])
            logging.info('{}\t{}\t{}\t{}'.format(comment['content'], comment
['referenceTime'],comment['nickname'], comment['userClientShow']))

            self.content_process(comment)
            self.content_sentences += comment['content']

        else:
          logging.error('Status NOT OK')
          break

        p_data['page'] += 1
        self.random_sleep()  # delay

    def item_id_extracter_from_url(self, url):
      item_id = 0

      prefix = 'item.jd.com/'
      index = str(url).find(prefix)
      if index!=-1:
        item_id = url[index + len(prefix): url.find('.html')]

      if item_id != 0:
        return item_id

    def date_check(self, date_here):
      if self._checkdate is None:
        logging.warning('You have not set the checkdate')
        return True
      else:
        dt_tocheck = datetime.strptime(date_here, '%Y-%m-%d %H:%M:%S')
        if dt_tocheck > self._checkdate:
          return True
        else:
          logging.error('Date overflow')
          return False
```

```
    def content_process(self, comment):
        with open('jd-comments-res.csv', 'a') as csvfile:
            writer = csv.writer(csvfile, delimiter=',')
            writer.writerow([comment['content'], comment['referenceTime'],
                             comment['nickname'], comment['userClientShow']])

    def random_sleep(self, gap=1.0):
        # gap=1.0
        bias = random.randint(-20, 20)
        gap += float(bias) / 100
        time.sleep(gap)

    def get_keywords(self):
        content = self.content_sentences
        kws = jieba.analyse.extract_tags(content, topK=20)
        return kws
if __name__ == '__main__':

    url = input("输入商品链接: ")
    date_str = input("输入限定日期: ")
    page_num = int(input("输入最大爬取页数: "))
    jd1 = JDComment(url, page_num)
    jd1.set_checkdate(date_str)
    print(jd1.get_comment_from_item_url())
    print(jd1.get_keywords())
```

在该爬虫程序中使用的模块有 requests、json、time、random、csv、lxml. html、jieba. analyse、logging、datetime 等。后面将会对其中的一些模块做简要说明。接下来先运行爬虫试一试，打开另外一个商品页面来测试爬虫的可用性，URL 为“http://item. jd. com/1027746845. html”(这是图书《白夜行》的页面)，运行爬虫，效果如图 5-10 所示。

图 5-10　运行 JDComment 爬虫后的效果

“ERROR:root:Date overflow”信息说明由于日期限制，爬虫自动停止了，在后续的输出中用户可以看到评论关键词信息如下：

```
['京东', '正版', '不错', '好评', '快递', '本书', '包装', '超快', '东野', '速度', '质量', '价钱', '物流', '便宜', '喜欢', '白夜', '满意', '好看', '很快', '很棒']
```

同时，在爬虫程序目录下生成了“jd-comments-res. csv”文件，说明爬虫运行成功。

5.2.3　数据下载结果与爬虫分析

使用软件打开 CSV 文件，可以看到抓取到的所有评论及相关信息(见图 5-11)，如果以后

还需要对这些内容进行进一步的分析，就不需要再运行爬虫了。当然，对于大规模的数据分析要求而言，保存结果到数据库中可能是更好的选择。

2	618跨店买了600多的书，觉得超值，买上装备我的书房	2017-05-31 01:14:06	苗***_	来自京东iPhone客户端
3	感觉不错，印刷不错，服务不错，书两好，如果印刷不好，纸张不好，让你看着心里也不舒服。买书的过程也很重要，如果服务不好，速度太慢，等的心急，	2017-04-23 16:58:29	广***q	来自京东Android客户端
4	书很棒，东野圭吾大作！ 是正品，阅读中，下次还来你这家店，包装也严实	2017-02-22 18:23:34	—***d	来自京东iPhone客户端
5	买了两本！还不错！可以继续看下去！	2017-09-10 09:00:31	糖***士	来自京东iPhone客户端
6	非常可以，纸张还可以！	2016-12-27 18:39:35	中***红	来自京东Android客户端
7	塑封包装完好，字迹清晰，送货速度也很快！	2017-02-26 01:15:18	青***地	来自京东Android客户端
8	心仪已久，终于买到了，感谢京东开学季的活动，让我以平均每本15元的价钱买到了正版书籍，静下来读读书还是不错的，不能当电子产品的奴隶。虽然买	2017-09-05 11:02:27	松***日	来自微信购物
9	一次性买了很多，和朋友一起交换来看，之前也在京东上买过书，一般没什么问题，书也是正版，就是这次有一本书的膜是散开的，不过因为只有一本书也	2017-01-05 21:51:23	广***e	来自京东Android客户端
10	那么多书，感觉得看一段儿了，书都很好，就是快递包装真的是“不堪入目”啊	2017-06-03 10:31:35	荔***飞	来自京东Android客户端
11	非常好，快递很快，包装完好，没有破损，棒棒棒棒	2017-07-05 18:43:05	c***D	来自京东Android客户端
12	正版，书外观完好，先放着有时间慢慢来读。	2017-04-04 23:30:37	广***0	来自京东Android客户端
13	这是我第一次在文轩网买书，品质不错，但包装上有些瑕疵以至于书有一点点的损坏。建议：包装时在书的四个角上垫点东西，还有就是多错几圈胶带	2017-06-25 11:04:59	阿Hodor	来自京东iPad客户端
14	很期待的一本书！活动时买的，价格优惠，一下子买了十多本，每本书都有薄膜包着，都是正版，非常满意！够看一阵子的了，还会继续来买的！！	2017-04-23 21:22:25	小***y	来自京东Android客户端
15	就是这样的一个包装，外面连个盒子都没有，看四个角直接变形，省事，不想退换。	2017-05-31 23:43:08	d***n	来自京东iPhone客户端
16	可以可以。我觉得我双十一之前都不会再买书了。包装完好，价格划算，快递态度好	2017-05-31 00:58:56	小***糖	来自京东Android客户端
17	在这买了好几次了，每次都很满意，物流快，书质量很好，没有详细了解书的内容，看看封面的感觉买的。刚开始看，是推理小说，前面的文字描述还是有	2016-12-25 10:47:48	广***h	来自京东iPhone客户端
18	一本好书 闲暇时阅读 看看精装的书籍 翻过拥有质感的书页 走进书中的故事	2017-08-12 23:25:37	广***1	来自微信购物
19	买家印象经典推理京东正版热卖畅销好评最多读者推荐	2017-03-06 10:47:01	赤***智	
20	没想到是硬壳的。，软皮的比较结实，618大促很给力9本书才100多点。	2017-06-18 15:01:17	H***_	来自京东iPhone客户端
21	拿到之后其实的喜欢他的封面设计，有质感，很不一样，另外还有个不一样的地方，整本书没有序言和目录。哈哈哈，看来作者很直接～	2016-12-19 22:54:15	p***4	来自京东Android客户端
22	三天到的货，精装书。质量还可以，到货需要两三天。还没拆，应该不错吧。	2017-05-31 01:13:35	陈***冰	来自京东iPhone客户端
23	好评 有精心的包装 打开了书页质量很好 京东真的好啊大年初二送货上门 赞赞赞	2017-01-23 18:28:35	M***t	来自京东iPhone客户端

图 5-11　京东商品评论 CSV 文件的内容

在例 5-3 的爬虫程序中使用了 json 库来操作 JSON 数据，json 库是 Python 自带的模块，这个模块为 JSON 数据的编码和解码提供了十分方便的解决策略，其中最重要的两个函数是 json.dumps()和 json.loads()。json.dumps()函数可以把一个 Python 字典数据结构转换为 JSON；json.loads()函数则会将一个 JSON 编码的字符串转换回 Python 数据结构，在上述的爬虫代码中就使用了 json.loads()。

【提示】 json 模块中的 dumps 与 dump、load 与 loads 非常容易混淆，用一句话来说，函数名里的“s”代表的不是单数第三人称动词形式，而是“string”。因此虽然都是“解码”，load 用于解码 JSON 文件流，而 loads 用于解码 JSON 字符串。dumps 和 dump 的关系同理。

此外还使用了 csv 模块来存储数据（写入 CSV），在 Python 中 csv 模块可以胜任绝大部分 CSV 相关操作。为了写入 CSV 数据，首先创建一个 writer 对象，writerow()方法接收一个列表作为参数并逐个写入列中（一行数据）。类似地，writerows()方法则会写入多行。下面是一个例子：

```
import csv

headers = ['姓名','性别','学号','专业']
rows = [('王小明', '男', '10007', '计算机科学与技术'),
        ('赵小蕾', '女', '10008', '汉语言文学'),
       ]

with open('stu_info.csv','w') as f:
    f_csv = csv.writer(f)
    f_csv.writerow(headers)
    f_csv.writerows(rows)
```

之后就可以看到 stu_info.csv 文件中被写入的信息了。使用 csv 读取的过程类似：

```
with open('stu_info.csv') as f:
  f_csv = csv.reader(f)
  for row in f_csv:
    print(row)
```

运行上面的代码后就能在终端/控制台看到被打印出的 CSV 内容信息。

在 get_keywords()函数中还使用了 jieba 中文分词来分析评论文本中的关键词，jieba.analyse.extract_tags()的使用方法是 jieba.analyse.extract_tags(sentence, topK = 20,

withWeight=False, allowPOS=())，其中各参数的意义分别如下。

- sentence：待提取的文本。
- topK：返回几个 TF/IDF 权重最大的关键词，默认值为 20。
- withWeight：是否一并返回关键词权重值，默认值为 False。
- allowPOS：仅包括指定词性的词，默认值为空，即不筛选。

该函数使用 TF/IDF 方法来确定关键词，所谓的 TF/IDF 方法，主要思路是认为字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。也就是说，如果某个词或短语在一篇文章中出现的频率高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类，也就可以作为文本的关键词。

最后，在检查日期时（和初始化限定日期时）使用了 datetime.strptime()，可以将时间字符串根据指定的时间格式转换成时间对象。运行下面的代码就可以看到：

```
import datetime
dt1 = datetime.datetime.strptime('2017-01-01','%Y-%m-%d')
print(dt1)
print(type(dt1))
```

其输出结果为：

```
2017-01-01 00:00:00
<class 'datetime.datetime'>
```

【提示】 上述代码中的“%Y-%m-%d”为字符串格式，strptime()函数使用 C 语言库实现，格式信息有严格规定，见“http://pubs.opengroup.org/onlinepubs/009695399/functions/strptime.html”。另外，作为 strptime()函数的“另一面”，还存在一个 strftime()函数，它的功能是 strptime()函数的反面，即将一个日期（时间）对象格式化为一个字符串。

5.3　本章小结

本章使用了 Selenium 与 ChromDriver 的组合来抓取网络小说，还使用了 requests 模块展示如何分析并获取购物网站后台 JSON 数据，同时对爬虫程序中用到的功能及其对应的模块做了一些简单的讨论。本章中出现的 Python 库大多都是编写爬虫时的常用工具，在 Python 学习中掌握这些常用模块的基本用法是很有必要的。