第3章 数据抓取

我们在浏览网页资源时，其实也是获取网页数据的一个过程。但对于有大量数据的网页，可以使用编程的方式来高效地抓取数据，以方便我们对数据做统计、分析。通过前面对Python、HTML、HTTP 及谷歌开发者工具的学习，我们可以正式开始对网页数据进行抓取了。本章将偏重实战，通过各种比较典型的实例来详细讲解网页抓取的过程。前面的工具在本章中将会一起工作，组成一套强大的数据抓取工具。

3.1 工具准备

我们需要通过代码来模拟浏览器行为，帮助执行 HTTP 的请求，获取 HTTP 响应等。Python 本身的标准库里带有相关库可以完成相关的任务，但我们有更简单易用的第三方库：requests（http://cn.python-requests.org/zh_CN/latest/）。它声称让 HTTP 为人类服务，我们先来看看是不是方便人类使用。

前面安装 Anaconda 开发工具套件里已经包含了 requests 库，所以可以直接在 Jupyter 里使用。

代码 3-1

```
1.  import requests
2.
3.  http_response = requests.get("https://movie.python.org")
4.  print(http_response.text)
```

直接在 Jupyter 里运行代码 3-1，可以看到下面的输出：

```
<!DOCTYPE html>
<!--[if lt IE 7]>   <html class="no-js ie6 lt-ie7 lt-ie8 lt-ie9">   <![endif]-->
<!--[if IE 7]>      <html class="no-js ie7 lt-ie8 lt-ie9">         <![endif]-->
<!--[if IE 8]>      <html class="no-js ie8 lt-ie9">               <![endif]-->
<!--[if gt IE 8]><!--><html class="no-js" lang="en" dir="ltr">  <!--<![endif]-->

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <link rel="prefetch" href="//ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js">

    <meta name="application-name" content="Python.org">
    <meta name="msapplication-tooltip" content="The official home of the Python Programming Language">
    <meta name="apple-mobile-web-app-title" content="Python.org">
```

在上面的代码中，使用 requests 库发送了一个 HTTP 的 GET 请求。HTTP 的响应直接保存在了 http_response 变量里，我们可以通过该变量来获取需要的信息，如通过 http_response.text 获取 HTML 代码的文本。我们还可以通过 http_reponse.headers 和 http_response.status_code 来获取 HTTP

响应的头部和状态码，headers 返回的是一个 Python 字典，而 status_code 返回的是一个整数。

```
print(http_response.headers)
print(http_response.status_code)
```

下面再来看看 requests 是如何发送 POST 请求的。POST 请求需要向网站发送数据，我们找到一个专门用来测试 HTTP 的网站：https://httpbin.org。

代码 3-2

```
import requests

http_response = requests.post('https://httpbin.org/post',
                              data = {'hello':'world'})
print(http_response.text)
```

代码 3-2 中使用了 requests.post()方法，把一个 Python 字典{"hello":"world"}赋值给 post()方法。字典里的数据将会作为 POST 请求的消息体传递给 httpbin.org/post 这个 URL，该 URL 会进行如下响应。

```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "hello": "world"
  },
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Content-Length": "11",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.21.0"
  },
  "json": null,
  "origin": "104.168.211.122, 104.168.211.122",
  "url": "https://httpbin.org/post"
}
```

httpbin.org/bin 这个 URL 会向我们响应一些测试信息：通过 post()方法传递给 URL 数据，requests.post()方法的 HTTP 请求头部等。从头部信息的 User-Agent 可以看出，我们使用的是 python-requests/2.21.0 这个库来发出的 HTTP 请求。服务器可以很容易地识别这个不是正常的浏览器请求，有些严格的服务器检测到这种类型的 User-Agent 会直接拒绝这个请求。因此，在需要的情况下，我们可以伪造 HTTP 请求头部的 User-Agent 信息。

代码 3-3

```
import requests

myheaders = {"User-
Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.
```

```
131 Safari/537.36"}
 4.  http_response = requests.post("https://httpbin.org/post", headers=myheaders, data = {'hello':'world'})
 5.  print(http_response.text)
```

在代码 3-3 中，我们定义了名为 myheaders 的 Python 字典，设置 User-Agent 为正常的浏览器值，再运行代码时，HTTP 的响应里就会检测到伪造的 User-Agent 值：

```
{
   "args": {},
   "data": "",
   "files": {},
   "form": {
     "hello": "world"
   },
   "headers": {
     "Accept": "*/*",
     "Accept-Encoding": "gzip, deflate",
     "Content-Length": "11",
     "Content-Type": "application/x-www-form-urlencoded",
     "Host": "httpbin.org",
     "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
chrome/74.0.3729.131 s afari/537.36"
   },
   "json": null,
   "origin": "104.168.211.122, 104.168.211.122",
   "url": "https://httpbin.org/post"
}
```

笔记栏

https://developers.whatismybrowser.com/useragents/explore/ 这个网站提供了各种浏览器的 User-Agent，可以根据自己的需要来进行选择。如果出现有些网页不能抓取的情况，可以利用 requests.get()或 requests.post()方法的 headers 参数进行伪装。

关于 requests 库的用户，我们先介绍到这里，在后面的数据抓取实例中，还会进一步讲解 requests 库的使用。

3.2 Xpath 和 lxml.html

现在已经找到在代码中可以当作浏览器使用的 requests 库，但我们还需要类似像谷歌开发者工具那样能够通过代码来分析网页。而 Xpath 和 lxml.html 就是我们要找的分析工具。

3.2.1 网页分析利器——lxml

lxml 是用来处理 XML 和 HTML 的一个 Python 第三方工具库，通常我们会使用它里面的 html 工具，也就是 lxml.html。而在 lxml.html 库里有一个常用的方法 lxml.html.fromstring()，这个方法会把我们通过 requests 库获取的 HTML 文本转换成可以分析的 HTML 对象。

关于这个 HTML 对象，可以把它当作我们用谷歌浏览器的开发者工具来分析的网页，可以

通过鼠标来定位元素。但在代码里，没有图形界面不能使用鼠标，而 lxml.html 神奇的地方就在于，它提供了一个叫作 XPah 的方法来帮助我们定位元素。

笔记栏

XML 是 eXtensible Markup Language（可扩展标记语言）的缩写，它与 HTML 在结构上非常接近，因此我们可以用 XPath 来分析 XML 和 HTML。

3.2.2 XPath

XPath 是 XML Path Language 的缩写，它可以用来查找 HTML 元素及元素属性。XPath 利用自己特有类似路径的标记方法来定位 HTML 中嵌套的元素关系，如表 3-1 所示列出了 XPath 的基本表达式。

表 3-1

表 达 式	功 能
/	选择元素，但必须从 HTML 的根目录中选择。如/html/body/div 会选择 HTML 元素下面的 body 元素，body 元素下面的所有 div 元素
//	选择所有元素，不管元素在 HTML 中的位置。如//div 会选择 HTML 中所有的 div 元素
.	选择当前元素。如.//li，会在选择当前元素下面的所有 li 元素
@	选择 HTML 元素的属性。如//li[@class]会选择所有带有 class 属性的 li 元素。//li[@class="fruit"]会选择所有 class="fruit"的 li 元素。//div[@attr1="a"][@attr2="b"]选择带有 attr1="a"和 attr2="b"属性的元素
//element[n]	选择所有 element 元素的第 n 个元素
//*[@attr="abc"]	选择所有带 attr="a"属性的任何元素

下面就是一个 XPath 表达式，用来定位 HTML 中的 li 元素：

```
//*[@id="screening"]/div[2]/ul/li[7]/ul/li[1]
```

可以看出，不同的元素之间使用“/”号分隔，有点像操作系统中的路径一样，一层一层从最外层的元素，定位到目标元素。对于 XPath 表达式，最好的学习途径就是通过实例来进行。先来看下面一段网页的 HTML 代码。

代码 3-4

```
<!DOCTYPE html>
<html>
  <head>
    <title>XPath 示例</title>
  </head>
  <body>
    <div class="fruit">
      水果列表:
      <ul>
        <li id="apple">苹果</li>
        <li>香蕉</li>
        <li class="special">西瓜</li>
      </ul>
```

```
14.     </div>
15.
16.     <div class="vehicle">
17.       交通工具列表:
18.       <ul>
19.         <li>汽车</li>
20.         <li>火车</li>
21.         <li class="special">飞车</li>
22.       </ul>
23.     </div>
24.   </body>
25. </html>
```

上面这个示例 HTML 代码的<body>元素里有 2 个 元素，每个 元素里面有 1 个<ul>列表，每个<ul>列表里有 3 个<li>元素。我们专门为有些元素设置了一些属性。

3.2.3 XPath 使用实例

当我们获取了代码 3-1 中的 HTML 代码文本时，就可以利用 XPath 来进行元素的选择了。

实例一：选择所有的 元素

```
//div
```

上面的 XPath 会选择 HTML 中的 2 个 元素。

实例二：选择“水果列表”的 元素

```
//div[@class="fruit"]
```

这个 XPath 会选中带有 class="fruit"属性的“水果列表”的 ，根据代码 3-1 会返回一个 元素。

实例三：选择所有的<li>元素

```
//li
```

这个 XPath 会选择所有的<li>元素，因为“//”并不会考虑元素的位置，所以返回 6 个<li>元素。

实例四：先选择“水果列表” 再选择下面的<li>

```
//div[@class="fruit"]
.//li
```

第一个 XPath 会选择“水果列表”的 元素，第二个 XPath 在“//”前面使用了“.”，表示以第一个 XPath 选出来的 元素为基准，选择这个 元素下面所有的<li>元素。这里的结果应该是 3 个<li>元素。

如果第二个 XPath 之前没有使用“.”，那么这个 XPath 就会无视元素的位置，选择所有的<li>元素，效果跟实例三中的 XPath 一样，会选出页面中所有的 6 个<li>元素。

实例五：选择“苹果”元素

```
//li[@id="apple"]
```

这个 XPath 使用了“苹果”列表的“id”属性来选择这个元素，前面我们讲 HTML 属性

时提到过，id 属性是唯一的，所以这个 XPath 会选出这个网页上唯一的 id 为“apple”的<li>元素。

实例六：选择“交通工具列表”下面带有 class="special"属性的交通工具

```
//div[@class="vehicle"]/li[@class="special"]
```

这个 XPath 使用了“/”符号，表示后面的<li>元素是在前面的 元素下面，即<li>是 的子元素。

实例七：选择第一个 元素

```
//div[1]
```

这里使用了[1]类似 Python 列表的访问方法。注意，Python 列表的第一个元素序号是从 0 开始的，而 XPath 的第二个元素是从 1 开始的。

实例八：选择所有带 class="special"的元素

```
//*[@class="special"]
```

这个 XPath 将会选择两个带 class="special"的<li>元素。

实例九：不使用“//”的选择方式

```
/html/body/div/ul/li
```

因为“//”语法无视元素所在的位置，如果不使用这个语法，单纯使用“/”来选择元素的话，就需要使用完整的路径。这种 XPath 将会选择所有的 6 个<li>元素。

3.2.4 XPath 演示

接下来，我们使用 lxml.html 库来演示 3.2.3 节中的 XPath 使用实例。lxml.html 库中有一个 fromstring()方法，可以把 HTML 代码文本转换成一种特殊的 Python 对象。有了这个对象之后，就可以对它使用 xpath()方法。把使用 XPath 语法编写的字符串作为该方法的参数传递给它，xpath()就可以进行元素选择了。

代码 3-5

```
1.  import lxml.html
2.
3.  with open('xpath.html', 'r', encoding='utf-8') as f:
4.      # 通过 lxml.html.fromstring()方法
5.      # 将保存在 xpath.html 中的 HTML 代码转换成 HTML 对象
6.      html = lxml.html.fromstring(f.read())
7.
8.  print('HTML 对象: {}'.format(html))
9.
10. print('\n 实例一: 选择所有的<div>元素(2 个):')
11. all_div = html.xpath('//div')
12. print(all_div)
13.
14. print('\n 实例二: 选择“水果列表”的<div>元素(1 个):')
15. fruit_div = html.xpath('//div[@class="fruit"]')
```

```
print(fruit_div)

print('\n 实例三：选择所有的<li>元素(6 个)：')
all_li = html.xpath('//li')
print(all_li)

print('\n 实例四：先选择“水果列表”<div>再选择下面的<li>')
# 由于 xpath()方法会返回一个列表，而且这个列表只有一个元素，
# 所以使用序号 0 来选择列表中的元素
fruit_div = html.xpath('//div[@class="fruit"]')[0]

# XPath 使用"."表示从当前<div>开始选择
fruit_li = fruit_div.xpath('.//li')

# XPath 不使用"."
wrong_fruit_li = fruit_div.xpath('//li')

print('使用了"."的选择结果(3 个):')
print(fruit_li)
print('未使用"."的选择结果(6 个):')
print(wrong_fruit_li)

print('\n 实例五：选择“苹果”元素(1 个):')
apple_li = html.xpath('//li[@id="apple"]')
print(apple_li)

print('\n 实例六：选择“交通工具列表”下面带有 class="special"属性的交通工具(1 个):')
# 这种方式就需要提供完整的嵌套路径，如/div/ul/li 的上一级
vehicle_special_li = html.xpath('//div[@class="vehicle"]/ul/li[@class="special"]')
print(vehicle_special_li)

print('\n 实例七：选择第一个<div>元素(1 个):')
first_div = html.xpath('//div[1]')
print(first_div)

print('\n 实例八：选择所有带有 class="special"的元素(2 个):')
special_li = html.xpath('//*[@class="special"]')
print(special_li)

print('\n 实例九：不使用“//”的选择方式(6 个):')
full_path = html.xpath('/html/body/div/ul/li')
print(full_path)
```

读者可以运行代码 3-5 来验证并体会上面的 XPath 使用实例。

3.3 关于 robots.txt

在我们开始抓取网页数据之前，还需要单独了解 robots.txt 这个文件。一般情况下，在大多

数网站的域名下面都有这个文件，如 https://movie.douban.com/robots.txt。这个文件里通常包含的就是爬虫协议，也叫网络机器人排除协议（Robots Exclusion Protocol）。

笔记栏

Web Robot 也称为网络机器人，通常是搜索引擎用来索引互联网内容的程序。一般我们所说的爬虫就属于网络机器人的一种。

网站的所有者通过这个协议来表明自己网站上的哪些内容可以抓取，哪些内容禁止抓取。

当然这并不能从技术上来限制别人对网站数据的抓取。但我们在抓取别人网站的内容时，为了避免给自己或别人造成不必要的麻烦，还是需要遵守这个协议的。

下面是一个普通 robots.txt 的内容。

```
User-Agent: Sogouspider
Allow:  /article
Allow:  /oshtml
Allow:  /product
Allow:  /ershou
Disallow:   /

User-Agent:  Googlebot
Allow:  /article
Allow:  /oshtml
Allow:  /product
Allow:  /spu
Allow:  /dianpu
Allow:  /oversea
Allow:  /list
Allow:  /ershou
Allow: /$
Disallow:  /

User-Agent:  *
Disallow:  /
```

这个文本里通常有 User-Agent、Allow 和 Disallow 几个关键字。User-Agent 就是第 3 章中 HTTP 头部的 User-Agent。Allow 和 Disallow 的后面通常指定特定的 URL 路径前缀，表示允许或禁止访问的页面。

比如：

```
User-Agent: Sogouspider
Allow:  /article
Allow:  /oshtml
Allow:  /product
Allow:  /ershou
Disallow:   /
```

指定 User-Agent 为 Sogouspider 的 robot 只能访问/article、/oshtml、/product 和/ershou 这几

个为前缀的页面，而不允许访问除这几个页面之外在页面。

再比如：

```
1.  User-Agent:  *
2.  Disallow:  /
```

它表示这个域名下的所有网页都是禁止任何 robot 的，也就是不允许数据抓取。User-Agent 中的*符号，表示任何的 robot。

一般我们在进行数据抓取前，如抓取 http://www.example.com 时，会看一下该域名下面是否有 robots.txt。通过浏览器打开 http://www.example.com/robots.txt 来检查协议的内容，确定要抓取的数据是否为协议所允许的。

除这个协议之外，对于那些我们可以抓取的内容，也不能无限制地抓取。为了避免给对方的网站服务器造成过大的压力，在数据抓取时需要限制程序的抓取频率，这通常在 Python 中使用 time.sleep()方法来实现。

3.4 小试牛刀

前面铺垫了这么多，现在终于进入正题了。我们先抓取一个简单的数据作为入门吧。在豆瓣电影（movie.douban.com）上有一个“一周口碑榜”电影排行（见图 3-1），现在我们需要把排在第一名的电影标题抓取下来。

图 3-1

3.4.1 过程分析

首先打开谷歌浏览器，进入豆瓣电影的网页。调出谷歌开发者工具，直接使用元素查看定位到排名第一的电影标题，如图 3-2 所示。

图 3-2

在右边的 HTML 元素上单击鼠标右键，选择 Copy→Copy Xpath，如图 3-3 所示。

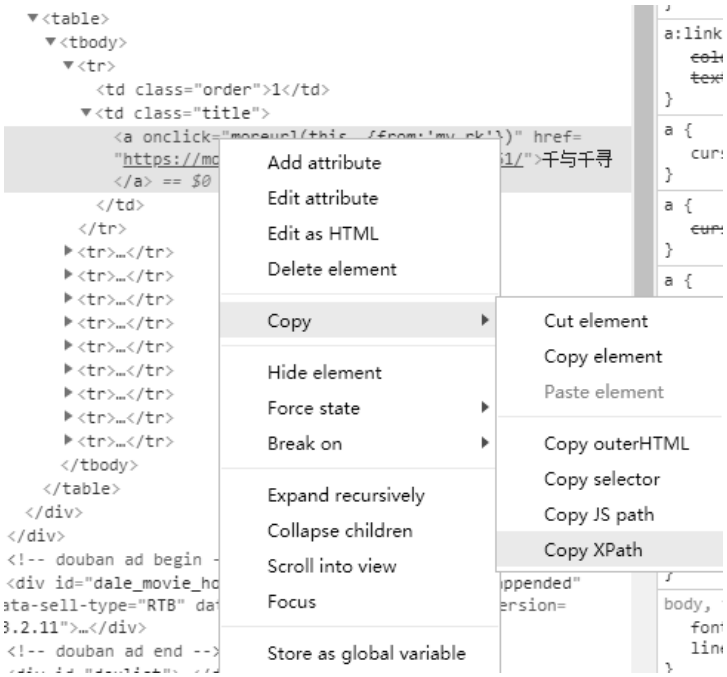

图 3-3

通过上面的操作可以获取类似下面的 XPath:

```
//*[@id="billboard"]/div[2]/table/tbody/tr[1]/td[2]/a
```

其实，我们认为谷歌开发者工具可以获取元素的 XPath 是其非常强大的地方。这也是为什么选择使用 lxml.html 库的原因，像获取这种单一的数据后，就可以直接在代码中使用了。

但是，有一点需要特别的注意，因为谷歌开发者工具过于强大，以至于在分析网页时会将不规范网页进行一些规范化。这是什么意思呢？我们先在网页上单击鼠标右键，选择“查看网页源代码”命令，如图 3-4 所示。

图 3-4

浏览器会重新打开一个窗口，这个窗口就是当前网页真实的 HTML 代码，如图 3-5 所示。

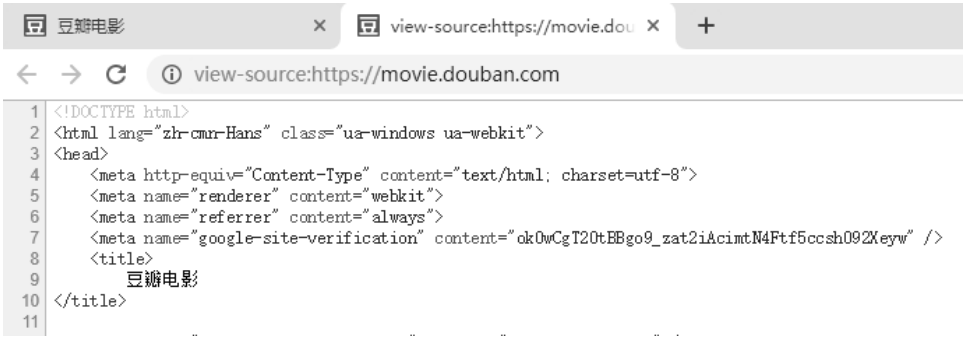

图 3-5

我们在这个页面上直接通过 Ctrl+F 快捷键来查看“一周口碑榜”。

```
<div id="billboard" class="s" data-dstat-areaid="75" data-dstat-mode="click,expose">
    <div class="billboard-hd">
        <h2>一周口碑榜<span><a href="/chart">更多榜单&raquo;</a></span></h2>
    </div>
    <div class="billboard-bd">
        <table>
                <tr>
                    <td class="order">1</td>
                    <td class="title"><a onclick="moreurl(this, {from:'mv_rk'})"
ef="https://movie.douban.com/subject/1291561/">千与千寻</a></td>
                </tr>
                <tr>
```

然后将这里的 HTML 代码与元素查看面板里的代码进行对比，如图 3-6 所示。

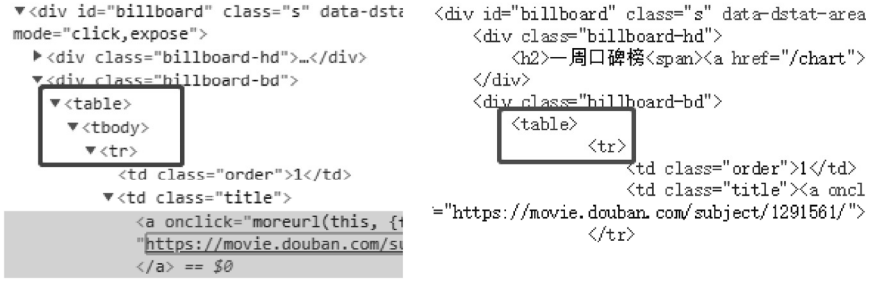

图 3-6

左边是元素查看面板里的 HTML 代码，右边是网页上实际的 HTML 代码。谷歌开发者工具复制的 Xpath 使用的是规范化后的 HTML 代码。然而在使用 Python 代码来进行网页抓取时，往往获取的是原始的 HTML（有可能就是不规范的 HTML）。在这种情况下，如果继续通过谷歌开发者工具的 Xpath 来搜索 HTML，通常会找不到指定的元素。

这经常出现在 Xpath 里有<table>标签的情况，谷歌开发者工具在规范化<table>标签时会加上<tbody>标签。因此对于前面的 Xpath：

```
//*[@id="billboard"]/div[2]/table/tbody/tr[1]/td[2]/a
```

需要要修改为：

```
//*[@id="billboard"]/div[2]/table/tr[1]/td[2]/a
```

现在我们看看相关代码。

3.4.2 动手敲代码

代码 3-6

```
1.  import requests
2.  import lxml.html
3.
4.  # 获取豆瓣电影首页
5.  http_response = requests.get('https://movie.douban.com')
6.  # 设置中文编码
7.  http_response.encoding = 'utf-8'
8.
9.  html = lxml.html.fromstring(http_response.text)
10. # 通过XPath 来获取排名第一的电影
11. title = html.xpath('//*[@id="billboard"]/div[2]/table/tr[1]/td[2]/a')
12. print(title)
13. print(title[0].text_content())
```

在代码 3-6 中，先是导入需要的两个库：requests 和 lxml.html，然后使用 requests 来获取豆瓣电影首页的 HTML，并设置中文编码，以防止在出现中文乱码：

```
1.  import requests
2.  import lxml.html
3.
4.  # 获取豆瓣电影首页
5.  http_response = requests.get('https://movie.douban.com')
6.  # 设置中文编码
7.  http_response.encoding = 'utf-8'
```

在获取 HTML 文本后，利用 lxml.html.fromstring()方法把 HTML 文本转换成可以使用 XPath 分析的对象，并保存在变量 html 中：

```
9.  html = lxml.html.fromstring(http_response.text)
```

调用 html.xpath()方法，把前面从谷歌浏览器里获取的 XPath 作为 xpath()参数传递进来，这样就可以获取到电影标题的元素了：

```
11. title = html.xpath('//*[@id="billboard"]/div[2]/table/tr[1]/td[2]/a')
12. print(title)
13. print(title[0].text_content())
```

我们直接在 Jupyter 里运行上面的代码，会得到以下输出：

```
[<Element a at 0x2100a8189f8>]
千与千寻
```

这正好是当前排名第一的电影。

知识库

在代码 3-6 的最后，我们使用了 title[0].text_content()方法来获取电影的标题。这里要说明一下，lxml.html 的 xpath()方法返回的结果始终是一个数组，即使没有找到元素（空数组）。我们

使用 Python 的获取数组元素的方法，取得 lxml.html 的元素，然后再使用 text_content()方法就可以获取电影名称所在元素（<a>元素）之间的文本。

3.4.3 小结

上面这个比较简单的抓取实例将会是后续复杂网页抓取的基础，实例中我们将前面用到的几个技术结合起来，共同完成抓取的任务。下面总结一下网页抓取的简单步骤。

第一步：获取 HTML 文本。

这一步通常是利用 requests 库来进行的，使用它的 GET 或 POST 等方法来发送 HTTP 请求。有时会设置请求的头部，或者设置一些中文的编码，以防止乱码出现。

第二步：转换 HTML 对象。

在获取到 HTML 文本后，会使用 lxml.html.fromstring()方法把 HTML 代码转换成可以使用 XPath 来处理的对象。

第三步：使用谷歌开发者工具获取元素的 XPath。

这一步其实也可以提前进行，利用工具帮助我们方便地查找元素的 XPath。但要注意，开发者工具在规范化网页时，会给我们造成一些小麻烦，需要自己处理。

第四步：使用 lxml.html 的 xpath()方法获取元素。

利用第三步查找目标元素的 XPath。我们可以使用 lxml.html 的 xpath()方法来获取目标元素。最后使用 text._content()方法来提取需要的文本（数据）。

3.4.4 扩展

上面实例获取的是排名第一的电影标题，如果要获取榜单上全部的电影标题该怎么做呢？我们先使用工具查看包含所有电影标题的元素，如图 3-7 所示。

图 3-7

通过观察，我们会发现所有的电影标题都包含在<tr>元素里。而每一个<tr>元素都包含 2 个<td>元素，在第 2 个<td>元素里，有 1 个<a>标签，它才是最终电影标题所在的地方。接下来回顾一下代码 3-6 中获取的 XPath。

```
//*[@id="billboard"]/div[2]/table/tr[1]/td[2]/a
```

它获取的是第 1 个<tr>元素中的第 2 个<td>下面的<a>元素，现在应该比较明显了，如果要获取所有的电影标题，就应该是所有<tr>元素中第 2 个<td>下面的<a>元素，所以 XPath 应该是：

```
//*[@id="billboard"]/div[2]/table/tr/td[2]/a
```

接下来就可以更新代码 3-6 了。

代码 3-7

```
1.  import requests
2.  import lxml.html
3.
4.  # 获取豆瓣电影首页
5.  http_response = requests.get('https://movie.douban.com')
6.  # 设置中文编码
7.  http_response.encoding = 'utf-8'
8.
9.  html = lxml.html.fromstring(http_response.text)
10. # 通过 XPath 来获取所有电影标题
11. titles = html.XPath('//*[@id="billboard"]/div[2]/table/tr/td[2]/a')
12. for title in titles:
13. print(title.text_content())
```

3.5 获取电影数据（上）

本节我们将获取一些网页上比较常见的数据结构，如列表数据。在豆瓣电影的 Top 250 中就有类似的列表数据（https://movie.douban.com/top250），如图 3-8 所示。

图 3-8

在开始抓取前，应检查 movie.douban.com 下的 robots.txt 文件，https://movie.douban.com/robots.txt。

```
User-agent: *
Disallow: /subject_search
Disallow: /amazon_search
Disallow: /search
Disallow: /group/search
Disallow: /event/search
Disallow: /celebrities/search
Disallow: /location/drama/search
Disallow: /forum/
Disallow: /new_subject
Disallow: /service/iframe
Disallow: /j/
Disallow: /link2/
Disallow: /recommend/
Disallow: /doubanapp/card
Disallow: /update/topic/
Sitemap: https://www.douban.com/sitemap_index.xml
Sitemap: https://www.douban.com/sitemap_updated_index.xml
# Craw1-delay: 5

User-agent: Wandoujia Spider
Disallow: /
```

从 robots.txt 文件中可知，豆瓣电影并没有禁止抓取/Top 250 页面，所以我们就可以进行数据抓取了。

笔记栏

我们还注意到豆瓣电影的 robots.txt 里有一个 Crawl-delay:5 的语句，它的意思是规定抓取频率为 5s/次。因此为了遵守网站的协议，在连续抓取数据时需要设置 5s 的等待时间。

3.5.1 过程分析

我们计划抓取 Top 250 页面上电影的标题及电影的评分，并且将这两个数据进行格式化输出，如图 3-9 所示。

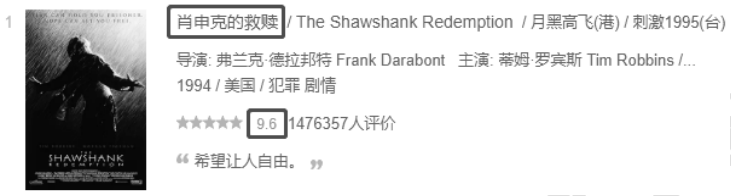

图 3-9

首先在豆瓣电影的 Top 250 页面上打开谷歌开发者工具，通过元素定位工具找到其中一部电影，选中该电影时，电影对应的 HTML 元素在右边会被高亮显示。我们将右边的 HTML 进

行一些折叠，就可以了解位于<li>元素里的整个电影信息，如图 3-10 所示。

图 3-10

图 3-10 中的右边有很多<li>元素，每个列表都对应一部电影，可以把鼠标指针移动到任意的<li>元素上，左边对应的电影也会被高亮显示。

在图 3-10 中，我们定位到了第 1 部电影《肖申克的救赎》的<li>元素，可直接通过单击鼠标右键来获取这个元素的 XPath。我们获取的这个<li>元素：

```
//*[@id="content"]/div/div[1]/ol/li[1]
```

在获取的 XPath 路径最后面有一个[1]，这表示选择的是第一个<li>元素。但我们的目的是获取所有的电影，因此只要去掉[1]即可：

```
//*[@id="content"]/div/div[1]/ol/li
```

新得到的 XPath 路径就可以帮助我们获取当前网页上所有电影的<li>元素。

使用 lxml.html 的 xpath()方法将返回的一个列表，列表里的元素是每个电影数据所在的<li>标签，所以我们会使用 Python 的 for 循环语句来依次获取电影列表里的所有<li>元素。

这里我们也可以使用第 3.4.4 节所述的方法完成任务，但该方法最终会导致我们获取两个列表，一个是所有电影的标题列表，另一个是所有电影评分的列表。

```
list1 = ['电影1标题','电影2标题','电影3标题', ...]
list2 = ['电影1评分','电影2评分','电影3评分', ...]
```

对于第 1 个电影，我们可以使用 list1[0]和 list2[0]来获取其数据，其他电影以此类推。这种方式有几个缺点：电影的数据是分离的，要获取单个电影的数据，需要在两个列表里抽取数据；只能根据列表的下标来对应同一个电影数据。如果出现电影数据丢失现象，则很难把电影的数据进行关联，比如：

```
list1 = ['电影1标题','电影2标题','电影3标题', ...]
list2 = ['电影1评分','电影2评分','电影4评分', ...]
```

如果第 3 个电影丢失了评分，再通过列表下标的方式获取电影的数据就会导致错误的结果。

所以我们换一种方法，先使用谷歌开发者工具，观察<li>元素下面的 HTML 结构，就会发现电影标题是在一个元素中，而这个元素并不是直接在<li>下面的，两个元素之间还嵌套了多个元素。如果把电影标题所在的<li>里面的所有 HTML 元素完全展开的话，还会发现第 1 个标题所在元素是在<li>下面的所有元素里的第 1 个，如图 3-11 所示。

图 3-11

所以我们使用 XPath 的语法//span[1]来获取第 1 个标题所在的元素。在代码中使用的是'.//span[1]'，表示要从当前的<li>元素下面查找所有的元素。

在找到电影标题所在的元素后，使用 text_content()方法来获取元素里的文本，也就是我们要找的电影标题。

接下来获取对应电影的评分，继续使用谷歌浏览器工具定位到评分的元素，如图 3-12 所示。

图 3-12

电影评分也在元素里，可以使用获取电影标题的方法来获取该元素。但在图 3-12 中，我们发现评分的元素有两个属性：class="rating_num"和 property="v:average"。通过这两个属性可以使用 XPath 的[@]语法，利用属性来定位元素。

笔记栏

在使用 XPath 对元素进行定位时，我们没有必要限制于某种方法，只要使用的 XPath 表达式能够唯一地定位所需的元素即可。在定位的过程中，我们可以输出一些信息来协助调试 XPath 的元素定位。

3.5.2 动手敲代码

代码 3-8

```
1.  import requests
2.  import lxml.html
3.
4.  # 获取豆瓣电影 Top 250 的网页，
5.  # 并转换了可使用 XPath 分析的对象
6.  http_response = requests.get('https://movie.douban.com/top250')
7.  http_response.encoding = 'utf-8'
8.  html = lxml.html.fromstring(http_response.text)
9.
10. # 获取所有电影列表的<li>元素
11. movies = html.xpath('//*[@id="content"]/div/div[1]/ol/li')
12.
13. for movie in movies:
14.     # 获取第一个电影标题所在的元素
15.     first_title_element = movie.xpath('.//span[1]')
16.     # 获取电影标题
17.     title = first_title_element[0].text_content()
18.     # 获取电影评分所在的元素
19.     rating_element = movie.xpath('.//span[@class="rating_num"][@property="v:average"]')
20.     # 获取评分
21.     rating = rating_element[0].text_content()
22.     # 格式化输出电影标题及评分
23.     print('<<{}>> - {}'.format(title, rating))
```

在代码 3-8 中，与前面的实例一样，使用 requests 将目标网页下载。对于具体的字符集，可以使用谷歌开发者工具在页面的<head>元素查看网页的字符集，在代码里设置对应的编码，如图 3-13 所示。

图 3-13

在经过 lxml.html.fromstring()方法将 HTML 代码转换后，可使用 xpath()方法获取所有的电影元素，并存放在 movies 列表里。通过 for 循环语句来读取每个电影的数据，每一次循环都获取一个电影所在的元素<li>，并保存在 movie 变量里。根据上面的分析，我们继续对 movie 这个变量使用 xpath('.//span[1]')方法，即可获取当前电影的标题。

```
    # 获取第一个电影标题所在的元素
    first_title_element = movie.xpath('.//span[1]')
    # 获取电影标题
    title = first_title_element[0].text_content()
```

使用同样的方法，继续获取电影的评分：

```
    # 获取电影评分所在的元素
    rating_element = movie.xpath('.//span[@class="rating_num"][@property="v:average"]')
    # 获取评分
    rating = rating_element[0].text_content()
```

最后将获取的电影标题和评分进行格式化输出：

```
    # 格式化输出电影标题及评分
    print('<<{}>> - {}'.format(title, rating))
```

可以在 Jupyter 中执行代码，得到的结果如图 3-14 所示。

```
<<肖申克的救赎>> - 9.6
<<霸王别姬>> - 9.6
<<这个杀手不太冷>> - 9.4
<<阿甘正传>> - 9.4
<<美丽人生>> - 9.5
<<泰坦尼克号>> - 9.4
<<千与千寻>> - 9.3
<<辛德勒的名单>> - 9.5
<<盗梦空间>> - 9.3
<<忠犬八公的故事>> - 9.3
<<机器人总动员>> - 9.3
<<三傻大闹宝莱坞>> - 9.2
<<放牛班的春天>> - 9.3
<<海上钢琴师>> - 9.2
<<楚门的世界>> - 9.2
<<大话西游之大圣娶亲>> - 9.2
<<星际穿越>> - 9.2
<<龙猫>> - 9.2
<<教父>> - 9.3
<<熔炉>> - 9.3
<<无间道>> - 9.2
<<疯狂动物城>> - 9.2
<<当幸福来敲门>> - 9.0
<<怦然心动>> - 9.0
<<触不可及>> - 9.2
```

图 3-14

3.5.3 小结

在本实例中，还是使用谷歌开发者工具获取元素的 XPath。但在获取电影数据时，我们采

用了一个新的方法，先找出每个电影数据所在的 HTML 元素（<li>）。然后在每个电影数据的 HTML 元素基础上，继续使用 XPath 的语法，找到基于该元素的下一级元素。这些下一级元素包含了同一个电影的信息，通过 Python 的 for 循环语句把同一个电影的数据输出，或者保存。

3.6 获取电影数据（下）

在 https://movie.douban.com/top250 网页上的电影数据只有 25 条。这只是 Top 250 电影的一部分。在第 2 章的后面，我们已经分析了豆瓣电影网导航条的翻页逻辑，如图 3-15 所示。

图 3-15

推测出了如下的 URL 变化规则：

第 1 页 URL：https://movie.douban.com/top250?start=25&filter=

第 2 页 URL：https://movie.douban.com/top250?start=25&filter=

第 3 页 URL：https://movie.douban.com/top250?start=50&filter=

第 4 页 URL：https://movie.douban.com/top250?start=75&filter=

第 5 页 URL：https://movie.douban.com/top250?start=100&filter=

……

根据 URL 的变化规律，可以推导出简单的公式：(p–1)×25，p 是页码。如果要自动抓取所有页面的电影数据，我们需要自动生成这些 URL，并将这些 URL 传递给 requests 的 get()方法，Python 就可以自动生成这些 URL。

代码 3-9

```
1.  url_tpl = 'https://movie.douban.com/top250?start={}&filter='
2.  for page in range(10):
3.      start = page * 25
4.      url = url_tpl.format(start)
5.      print(url)
```

代码 3-9 中的 rang(10)生成的刚好是 0～9 的整数，因此我们直接用它乘以 25 就能得到每一页 URL 里的 start 参数。把电影的 URL 定义在 url_tpl 变量里，并通过字符串的 format()方法来格式化 URL，最后可全部生成我们需要的电影数据的 URL：

```
https://movie.douban.com/top250?start=0&filter=
https://movie.douban.com/top250?start=25&filter=
https://movie.douban.com/top250?start=50&filter=
https://movie.douban.com/top250?start=75&filter=
https://movie.douban.com/top250?start=100&filter=
https://movie.douban.com/top250?start=125&filter=
https://movie.douban.com/top250?start=150&filter=
https://movie.douban.com/top250?start=175&filter=
https://movie.douban.com/top250?start=200&filter=
https://movie.douban.com/top250?start=225&filter=
```

3.6.1 过程分析

在 3.5 节中抓取电影数据时，我们只获取了电影的标题及评分，而且是在抓取的过程中对需要的数据进行定位并分析，也就是说数据是在线获取的。如果我们需要增加一些电影数据，如电影年代、标签等，就需要重新修改代码，再一次抓取数据。庆幸的是，我们抓取的数据不多，抓取过程的时间消耗也不会太大。但如果我们需要获取大量的数据，访问大量的网页，那么这种重复抓取的时间消耗是不可忽略的。

所以在接下来的实战中，我们会把电影的数据全部抓取下来，保存在一个文件里。如果后期需要再获取新的数据，或者需要重新分析时，就不需要再抓取豆瓣电影网页了。这样可避免时间或资源的浪费。

要保存所有的数据，我们可以修改代码 3-8 中的 for 循环：

```
1.  for movie in movies:
2.      # 强制转换为 Python 字符串
3.      movie_text = str(movie.text_content())
4.      # 去掉换行符，保证数据文本都输出在一行
5.      clean_movie_text = movie_text.replace("\n", "")
6.      print(clean_movie_text)
7.      print("----------------电影分隔----------------")
```

在 for 循环里，对每一个电影数据直接使用 text_content()就可获取电影所在元素<li>里的所有文本。由于 text_content()返回的并不是 Python 的字符串，所以应使用 str()方法把 text_content()返回的内容进行强制转换。在该代码第 5 行中，使用了字符串的 replace()方法把换行符全部替换，保证数据文本都在同一行，以方便将数据输出到文本文件里。代码输出如图 3-16 所示。

```
                                                    1
肖申克的救赎                                          / The Shawshank Redempt
ion                                          / 月黑高飞(港)  /  刺激1995(台)
[可播放]
导演: 弗兰克·德拉邦特 Frank Darabont    主演: 蒂姆·罗宾斯 Tim Robbins /...
1994 / 美国 / 犯罪 剧情
9.6                                                                  147
6357人评价
希望让人自由。
----------------电影分隔----------------
                                                    2
霸王别姬                                        / 再见，我的妾  /  Farewell My Co
ncubine                                                          [可播放]
导演: 陈凯歌 Kaige Chen    主演: 张国荣 Leslie Cheung / 张丰毅 Fengyi Zha...
1993 / 中国 / 剧情 爱情 同性
9.6                                                                  109
4275人评价
风华绝代。
----------------电影分隔----------------
```

图 3-16

上面输出的文本属于比较原始的，它们将被保存在一个文件里。在第 4 章中将会继续对保存的原始文本进行分析。

3.6.2 动手敲代码

现在我们已经可以自动生成 URL 了，如何提取数据也确定了，那么下面可以进行一次正式

数据抓取了。

代码 3-10

```
1.  import time
2.  import codecs
3.  import requests
4.  import lxml.html
5.
6.  with codecs.open("movies.txt", "w", "utf-8") as f:
7.
8.      url_tpl = "https://movie.douban.com/top250?start={}&filter="
9.
10.     for page in range(10):
11.         # 打印一些信息显示抓取进度
12.         print("获取第{}页".format(page+1))
13.         start = page * 25
14.         url = url_tpl.format(start)
15.
16.         # 替换抓取的网址为生成的URL
17.         http_response = requests.get(url)
18.         http_response.encoding = "utf-8"
19.         html = lxml.html.fromstring(http_response.text)
20.         movies = html.xpath('//*[@id="content"]/div/div[1]/ol/li')
21.
22.         for movie in movies:
23.             # 强制转换为Python字符串
24.             movie_text = str(movie.text_content())
25.             # 去掉换行符，保证数据文本都输出在一行
26.             clean_movie_text = movie_text.replace("\n", "")
27.             # 直接将文本打印到打开的文件里
28.             print(clean_movie_text, file=f)
29.
30.         # 根据豆瓣电影的robots.txt，暂停5s
31.         time.sleep(5)
```

代码 3-10 是结合了几个部分的代码，主要包含 3 个结构：首先是打开了一个名为 movies.txt 的文件，该文件会保存在和 ipynb 文件相同的目录；接着是一个 for 循环来生成电影页面的 URL；最后也是一个 for 循环，用来访问每个页面所有的电影列表。

第 6 行代码，我们使用 with codecs.open()方法用 utf-8 编码打开了 movies.txt 文件用于存放电影数据文本。使用 utf-8 编码是为了兼容在网页上获取文本时遇到不常用的编码，可避免在写入文件时发生乱码错误。

```
6.  with codecs.open("movies.txt", "w", "utf-8") as f:
```

第 10～14 行代码使用的是代码 3-9 中的逻辑，在循环开始时打印一些信息，以方便查看进度。for 循环在生成 URL 之后，把 URL 传递给 requests.get()。

```
10.     for page in range(10):
```

```
11.         # 打印一些信息显示抓取进度
12.         print("获取第{}页".format(page+1))
13.         start = page * 25
14.         url = url_tpl.format(start)
```

requests.get()获取自动算出的 URL 对应网页的 HTML 代码文本，然后转换成可用 xpath()分析的对象，将每一页的电影都保存在 movies 列表里，movies 列表在每一次 for 循环都会被替换更新。

```
16.         # 替换抓取的网址为生成的URL
17.         http_response = requests.get(url)
18.         http_response.encoding = "utf-8"
19.         html = lxml.html.fromstring(http_response.text)
20.         movies = html.xpath('//*[@id="content"]/div/div[1]/ol/li')
```

最里面的一层 for 循环把当面网页的所有电影数据，通过前面分析的方式提取出来。

```
22.         for movie in movies:
23.             # 强制转换为Python字符串
24.             movie_text = str(movie.text_content())
25.             # 去掉换行符，保证数据文本都输出在一行
26.             clean_movie_text = movie_text.replace("\n", "")
27.             # 直接将文本打印到打开的文件里
28.             print(clean_movie_text, file=f)
```

第 28 行代码通过 print()语句的 file 参数把打印的目标设置为打开的文件，将每个电影数据文本写入 movies.txt 文件。

通过任意文本编辑器打开 movies.txt 文件就可以查看抓取下来的电影数据，如图 3-17 所示。文本文件的长度应该是 250 行。

```
1
肖申克的救赎                                                / The Sh
刺激1995(台)
弗兰克·德拉邦特 Frank Darabont   主演：蒂姆·罗宾斯 Tim Ro
剧情
9.6
希望让人自由。
2
霸王别姬                                                / 再见，我的妾
[可播放]
Zha...                                                  1993 / 中国 /
同性
9.6
风华绝代。
3
这个杀手不太冷                                                / Léon
[可播放]
...                                              1994 / 法国 / 剧情 动作
犯罪
9.4
怪蜀黍和小萝莉不得不说的故事。
4
阿甘正传                                                / Forrest Gu
[可播放]
...                                              1994 / 美国 / 剧情
爱情
9.4
一部美国近现代史。
```

图 3-17

3.6.3 考虑加强代码的健壮性

在使用 requests 获取 URL 对应的资源里，有时网站服务器可能会响应一些不正确的数据。

这可能导致代码中的变量 http_response 没有获取正确的 HTML，最终导致后面的代码执行出错。有两种常见的方式可以增加 requests 获取 URL 资源里的健壮性。

（1）利用 status_code

前面介绍过 requests 的 HTTP 响应可以使用 status_code 来获取 HTTP 的响应码。正常情况下，发出一个 HTTP 请求，网站服务器的响应码为 200 时，可以判断这次请求是成功的，如果是非 200 的响应码，则请求有可能失败。这种情况我们就需要来处理一下，或者在代码里打印一些信息，提示发生了异常的情况。利用 Python 的 if…else…语句可以帮助我们，现在把代码 3-10 改进一下。

```
16.         # 替换抓取的网址为生成的 URL
17.         http_response = requests.get(url)
18.         http_response.encoding = "utf-8"
19.         if http_response.status_code == 200:
20.             html = lxml.html.fromstring(http_response.text)
21.             movies = html.xpath('//*[@id="content"]/div/div[1]/ol/li')
22.
23.             for movie in movies:
24.                 # 强制转换为 Python 字符串
25.                 movie_text = str(movie.text_content())
26.                 # 去掉换行符，保证数据文本都输出在一行
27.                 clean_movie_text = movie_text.replace("\n", "")
28.                 # 直接将文本打印到打开的文件里
29.                 print(clean_movie_text, file=f)
30.         else:
31.             print('抓取{}失败'.format(url))
```

完整的代码可以参考第 3 章的 Jupyter Notebook 文件。我们在代码的第 19 行增加一个判断语句，如果 HTTP 响应的 status_code 是 200，就执行正常的抓取操作，如果 status_code 不是 200，就提示当前的 URL 抓取失败了。也可以根据自己的需要更改第 31 行的代码，以打印更多的信息，或者添加自己的代码来进行处理。

（2）捕获异常

上面使用 status_code 来处理错误的方式比较有局限性，它的前提是网站服务器要响应 HTTP 请求。有时抓取 URL 时，网站根本没有响应，或者说是一个错误的 URL。这时，为了保证代码的健壮性，可以利用 Python 的异常捕获机制来完成。

```
16.         try:
17.             # 替换抓取的网址为生成的 URL
18.             http_response = requests.get(url)
19.             http_response.encoding = "utf-8"
20.             html = lxml.html.fromstring(http_response.text)
21.             movies = html.xpath('//*[@id="content"]/div/div[1]/ol/li')
22.
23.             for movie in movies:
24.                 # 强制转换为 python 字符串
25.                 movie_text = str(movie.text_content())
26.                 # 去掉换行符，保证数据文本都输出在一行
```

```
27.                 clean_movie_text = movie_text.replace("\n", "")
28.                 # 直接将文本打印到打开的文件里
29.                 print(clean_movie_text, file=f)
30.
31.             # 根据豆瓣电影的 robots.txt，暂停 5s
32.             time.sleep(5)
33.         except requests.exceptions.RequestException as err:
34.             print(err)
35.             print(print('抓取{}失败'.format(url)))
36.             # 继续抓取下一个 URL
37.             continue
```

上面的代码片段在代码 3-8 中包裹了一层 try…except…，在第 18 行代码中，如果 HTTP 请求出现了异常，第 33 行代码中就会捕获到异常，并执行第 34～37 行的代码。这部分其实就是出现异常的处理逻辑，最后的 continue 语句可以保存在抓取当前循环的 URL 出现异常时继续抓取下个循环的 URL。

3.6.4 小结

本章实例使用的是在抓取大量的多页数据时比较典型的方法：首先分析 URL 的翻页逻辑，通过代码来自动生成 URL；然后通过谷歌开发者工具分析需要抓取的数据；最后通过 lxml.html 库来利用 XPath 语法获取并保存需要的数据。

3.7 另类的网页抓取

前面抓取的网页都有一个特点，就是 URL 对应的网页是固定的，也就是通过浏览器打开指定 URL，网页在加载完成之后，内容是固定不变的。但有些网页有一些比较奇怪的行为，如豆瓣电影排行榜 https://movie.douban.com/chart 的分类排行榜的页面，如图 3-18 所示。

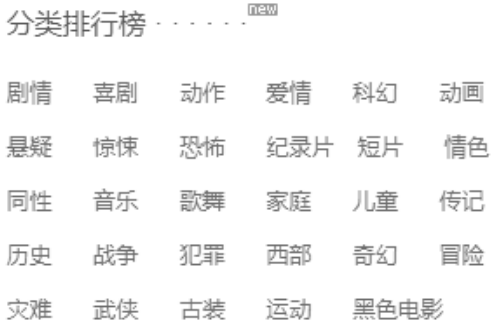

图 3-18

我们任选一个分类：喜剧，进入对应的页面，如图 3-19 所示。

在图 3-19 所示的页面中，如果向下滚动鼠标就会发现，滚动条快接近页面底部时，又有新的内容被加载。继续滚动，内容就会继续加载。

图 3-19

3.7.1 过程分析

上面的网页看起来与我们之前抓取的网页类似，也属于列表的数据。按照前面的方法先获取所有电影所在元素的 XPath：

```
//*[@id="content"]/div/div[1]/div[6]/div
```

然后使用下面的代码来获取电影元素。

代码 3-11

```
1.  import requests
2.  import lxml.html
3.
4.  http_response = requests.get('https://movie.douban.com/typerank?type_name=%E5%96%9C%E5%89%A7&type=
    24&interval_id=100:90&action=')
5.  http_response.encoding = 'utf-8'
6.  html = lxml.html.fromstring(http_response.text)
7.  movies = html.xpath('//*[@id="content"]/div/div[1]/div[6]/div')
8.  print(movies)
```

但是，当在 Jupyter 里运行时，会发现上面的代码返回一个空的列表，并没有找到任何元素。一般这种情况会怀疑是 XPath 路径的问题，为了验证，我们直接查找 id="content"的元素，并修改第 7 行代码如下：

```
7.  movies = html.xpath('//*[@id="content"]')
```

运行代码后输出：[<Element div at 0x1d778052688>]，这一级的元素可以找到，从谷歌开发者工具来看，电影列表就位于 id="content"的 元素中。接下来，我们直接把这个 元素的内容打印出来。

代码 3-12

```
1.  import requests
2.  import lxml.html
3.
4.  http_response = requests.get('https://movie.douban.com/typerank?type_name=%E5%96%9C%E5%89%A7&type=24&
    interval_id=100:90&action=')
5.  http_response.encoding = 'utf-8'
6.  html = lxml.html.fromstring(http_response.text)
7.  movies = html.xpath('//*[@id="content"]')
8.  movie = movies[0]
9.  # 使用中 fromstring()相反的 tostring()方法
10. # 把 HTML 对象转换成文本
11. div_text = lxml.html.tostring(movie, encoding='utf-8')
12. # 由于使用 tostring()转换的文本是字节编码
13. # 所以使用 decode('utf-8')将其转换成 utf-8 编码再输出
14. print(div_text.decode('utf-8'))
```

打印出来的 HTML 代码是 id="content"这个 元素下面的所有 HTML 代码，在谷歌开发者工具的 HTML 代码面板里，电影列表位于 这个元素下面。在代码 3-12 的输出中，我们利用 Ctrl+F 快捷键查找 元素，把它跟谷歌开发者工具里的 HTML 代码进行对比，如图 3-20 所示。

图 3-20

从图 3-20 中可以发现，通过代码抓取的指定元素下面并没有电影的相关元素。这也就是为什么该网页没有常规的翻页导航，只能通过鼠标向下滚动来加载新内容的原因。

这种网页的行为在前面介绍 JavaScript 时提到过，通过单击按钮、滚动网页可以更改网页的内容。在第 2.5 节的代码中，我们把数据定义在 JavaScript 的变量里，通过这些数据来增加表格的内容。那么这里通过滚动，对网页进行内容增加时的数据又在哪里呢？

在第 2.6.4 节中，介绍了如何筛选 HTTP 请求的资源。其中有一个分类是 XHR，它通常用来加载 JavaScript 需要的数据。所以我们现在应该知道如何去解决问题了。

在图 3-19 所示的网页上，我们打开谷歌开发者工具，切换到网络面板，筛选 XHR 资源。

笔记栏

在谷歌开发者工具中可以使用  按钮来清除数据。

使用 XPath 对元素进行定位的时候，不必限制于某种方法，只要我们使用 XPath 表达式能够唯一定位需要的元素即可。在定位的过程中，可以输出一些信息来协助调试 XPath 的元素定位。

这时，我们刷新前面的网页，如图 3-21 所示。

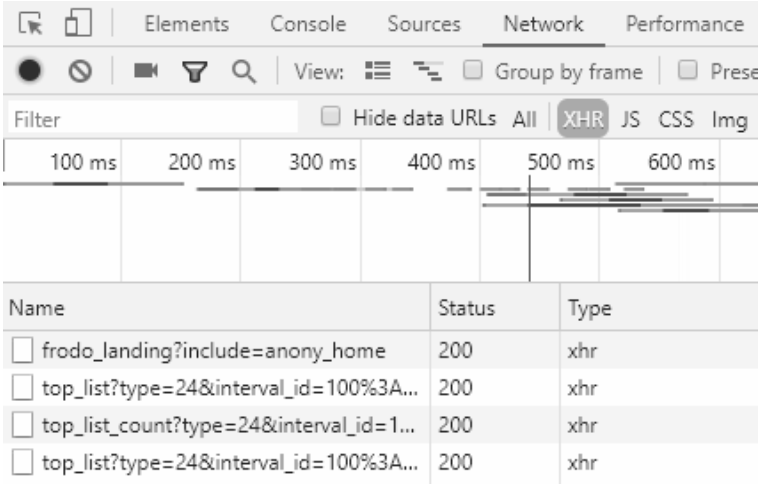

图 3-21

这时果然有一些资源被加载了，再继续滚动网页，会发现当左边的网页增加内容时，右边的资源列表也会新增一条内容，如图 3-22 所示。

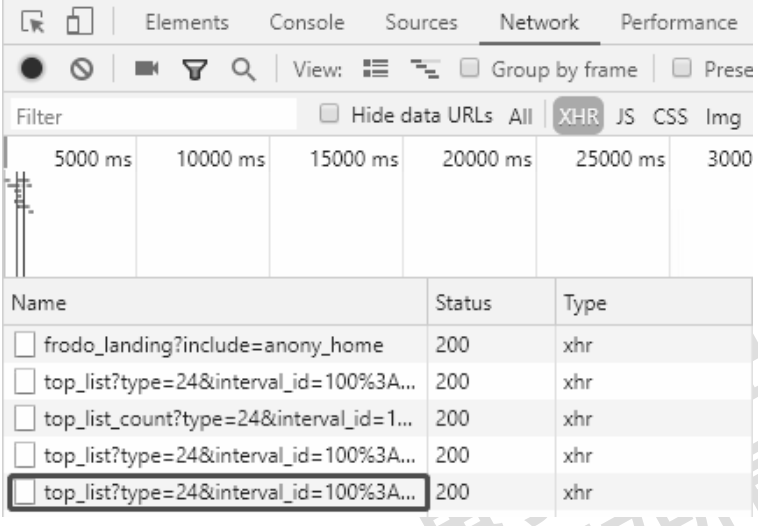

图 3-22

现在我们已经找到用代码抓取网页没有数据的原因了：因为这些数据是通过浏览器运行 JavaScript 代码之后加载数据重新生成的网页，而抓取网页的代码是没有运行 JavaScript 的功能的，所以只是抓取到了未运行 JavaScript 之前的 HTML 代码，当然也就不会有电影的数据了。

下面我们来看下这个 XHR 请求的资源是什么样子的。选中最后一条数据，并在右边的信息窗口中选择 Preview（预览）。展开其中一条数据，如图 3-23 所示。

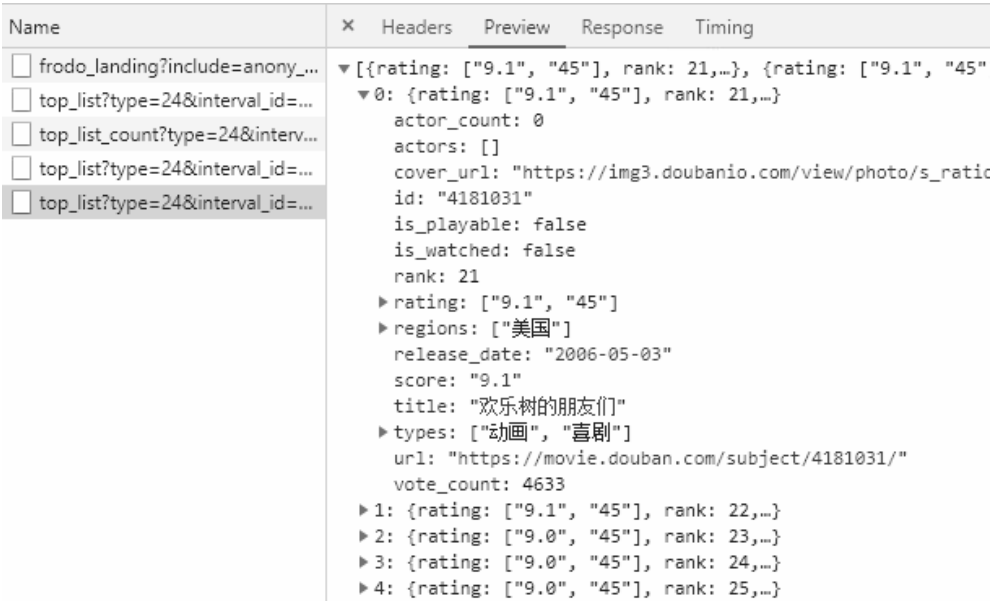

图 3-23

网页中的 XHR 请求，通常会返回一个 JSON 数组，而 JSON 数组与 Python 的字典可以通过 json 库来进行转换。对于这种类型的网页抓取的代码又是什么样子呢？

3.7.2 动手敲代码

对于这种类型的网页，我们一般不用 lxml.html 库。因为可以直接通过分析出来的 URL 获取数据，复制图 3-22 中选中条目的 URL：

```
https://movie.douban.com/j/chart/top_list?type=24&interval_id=100%3A90&action=&start=40&limit=20
```

对于这种 URL，我们可以参照以前介绍的方法来分析 URL，抓取多页的数据。这里我们只进行简单的演示，抓取单个 URL 的数据。

代码 3-13

```
1. import requests
2.
3. http_response = requests.get('https://movie.douban.com/j/chart/top_list?type=24&interval_id=100%3A90&
   action=&start=40&limit=20')
4. # 直接读取返回的 JSON 数据
5. movie_data = http_response.json()
6. print(type(movie_data))
7. for movie in movie_data:
```

```
8.      print(movie)
9.      print('----------------分隔----------------')
```

代码 3-13 的输出如图 3-24 所示。

图 3-24

这种方式获取数据的代码比分析网页获取数据的代码简单多了，请求的 URL 响应的是一个 JSON 数组，通过 http_response 的 json()方法可以直接获取。第 6 行代码我们直接打印出了变量 movie_data 的类型，可以确认 http_response.json()方法把 JSON 数组转换成了 Python 的列表对象。第 7、8 行代码，直接使用 for 循环把列表里的电影数据打印出来，每个电影数据都是一个 Python 字典。如果需要获取具体的数据，可以直接修改第 8 行代码。

3.7.3 小结

本章抓取的网页有一个特点，它通常是在浏览器里下载一些基本的网页，然后再通过浏览器的一些事件，如用户的鼠标滚动，发送一些 XHR 请求来获取服务器响应的数据。收到数据后利用 JavaScript 处理数据，将网页的内容更新。对于这种“动态”的网页，使用 XPath 一般不容易定位动态的元素，有些元素在用户触发某些事件时才能加载到浏览器里。通常 requests 库并不能触发这些事件，所以通过谷歌开发者工具分析了这种动态网页的数据来源，直接通过抓取 XHR 请求的 URL 来完成数据的抓取。

3.8 爬虫与网络机器人

本章的数据抓取与平时读者听到的爬虫是有区别的，但又有一定的联系。根据定义，爬虫（Web Crawler）又叫网络爬虫，是一种自动化、系统性浏览万维网（World Wide Web）的网络机器人。通常搜索引擎会使用网络爬虫来进行全网索引。而网络机器人则是通过编程语言开发的能够自动获取网页，然后进行分析的程序。网络机器人与我们的数据抓取程序比较接近。

本章的内容到这里就结束了，不会再继续深入讨论。因为专门的爬虫，还会涉及消息队列、URL 调度及各种搜索的算法等，这并不是本书的重点内容。在学习了数据抓取后，如果想进一步了解爬虫的话，可以寻找一些专门介绍爬虫的书籍来学习。

3.9 本章总结

本章通过几个实例来引入网页数据抓取的技术，把前面学习的 Python、HTML、谷歌开发者工具、XPath 等知识结合起来，打造出网页数据抓取的工具。

- 利用 requests 库来下载网页，lxml.html 来转换网页为可分析的对象。
- 通过 XPath 的语法，可以对 lxml.html 的对象元素进行定位，并获取数据。
- 通过分析 URL 的变化规则，对多页的数据进行抓取。
- 对于一些动态的网页，利用谷歌开发者工具分析获取数据的 URL，直接进行数据抓取。