第 3 章

Python网络数据采集

本章将介绍如何使用 Python 进行网络数据采集。首先介绍编写网络爬虫前需要掌握的技术概念，包括 HTML、HTTP、JavaScript 等构建现代网页的基础技术，以及网络数据采集需要遵循的道德法律规范。然后介绍三种使用 Python 编写网络爬虫的方法：使用爬虫基础库编写爬虫、使用爬虫框架编写爬虫及模拟人工访问浏览器编写爬虫。

学习目标

本章学习目标如下。

(1) 掌握 HTML 的基础语法，了解标签和属性这两个概念的意义。

(2) 掌握 HTTP 的基本概念，了解 HTTP 请求的种类和使用场景，以及常见的返回状态码的含义。

(3) 掌握 JavaScript 的特点，了解异步加载的原理。

(4) 掌握使用 Request 库和 BeautifulSoup 库编写爬虫的方法。

(5) 掌握 Scrapy 框架的原理及构建爬虫的方法。

(6) 掌握 Selenium 库的原理及爬取异步加载网页的方法。

3.1 网络爬虫基础

3.1.1 HTML

HTML(Hyper Text Markup Language，超文本标记语言)是一种用于创建网页的标记语言。网页浏览器可以读取 HTML 文档，并将其渲染成可视化网页。HTML 使用一系列标签描述一个网页中各个部分的位置和作用(如网页的标题、段落和列表等)，但它不像编程语言那样能够进行逻辑运算或数据处理，因此它是一种标记语言，而非编程语言。HTML 除了用于结构化信息，也可用于在一定程度上描述网页各个部分的外观和语义。HTML 还允许嵌入图像与对象，并且可以用于创建交互式表单。

HTML 的语言形式为包含于尖括号中的 HTML 标签(如< html >、< head >、< title >)，

浏览器使用这些标签来诠释网页内容，但不会将它们显示在页面上。HTML 往往与 JavaScript、CSS(Cascading Style Sheets，层叠样式表)结合使用，以设计令人赏心悦目的网页、网页应用程序及移动应用程序的用户界面。例如，HTML 可以嵌入如 JavaScript 的脚本语言，它们会影响 HTML 网页的行为，实现页面的交互和动态效果。网页也可以引用 CSS 来定义文本和其他元素的外观与布局。维护 HTML 和 CSS 标准的组织——万维网联盟(W3C)鼓励使用 CSS 替代一些用于描述外观的 HTML 元素。

HTML 包含标签、属性、基于字符的数据类型、字符实体、文档类型声明等关键部分，详细介绍如下。

(1) 标签。HTML 标签是 HTML 文档中最基本的构建单元，用于定义文档的结构和内容。HTML 标签通常成对出现，如< h1 >与 </h1 >。其中，第 1 个标签是开始标签，第 2 个标签是结束标签，两个标签之间为元素的内容。如果有些标签没有内容，那么只需要一个开始标签，这又称为“自闭合标签”，如用于嵌入图像的< img >标签。

(2) 属性。开始标签可以包含属性，这些属性有标识文本区段、将样式信息绑定到文本演示和提供引用资源来源等附加信息的作用。属性通常以名称＝"值"的形式出现，多个属性之间用空格隔开。例如，< img src＝"image. jpg" alt＝"Description of the image">是一个图像标签，有 src 和 alt 两个属性，分别代表图像的来源地址和替代文本。可以看到，图像的来源地址为"image. jpg"，替代文本为"Description of the image"，用于在图像无法显示时显示描述信息。

(3) 基于字符的数据类型。HTML 中的文本内容属于基于字符的数据类型，它指标签之间的文本内容，如段落、标题、链接文本等。这些文本数据用于向用户展示信息。

(4) 字符实体。有时候需要在 HTML 文档中使用一些特殊字符，如小于号(＜)或大于号(＞)等，但这些字符可能与 HTML 标签冲突。为了解决这个问题，可以使用字符实体。字符实体以 &. 开头，以；结尾，表示一个特定的字符。例如 &.lt；表示小于号(＜)，其中 lt 为实体名称。实体名称有对应的实体编号，例如 lt 对应的实体编号为＃60，用 &.＃60；也可以表示小于号。使用实体名称而不使用实体编号的好处是，实体名称易于记忆。但浏览器也许并不支持所有实体名称(对实体编号的支持度却很好)。

(5) 文档类型声明。文档类型声明的形式为 <!DOCTYPE >，它用尖括号包围，但不是一个 HTML 标签，它用于告知 Web 浏览器页面使用了哪种 HTML 版本。文档类型声明位于文档中最前面的位置，处于< html >标签之前。常见的文档类型声明为<!DOCTYPE html >，它告知浏览器页面使用的是 HTML5 标准，这会触发标准模式渲染，意味着浏览器会按照 HTML5 标准的要求来解析和显示页面内容。如果页面使用旧版本的 HTML 标准，则需要对应的声明。

一个 HTML 元素的一般形式为<标签 属性 1＝"值 1" 属性 2＝"值 2">内容</标签>。一个 HTML 元素的名称即为标签使用的名称。注意，结束标签的名称前面有一个斜杠“/”，空元素不需要也不允许使用结束标签。如果元素属性未标明，则使用其默认值。

HTML 文档由嵌套的 HTML 元素构成。在一个元素的开始标签与结束标签之间也可以封装另外的元素或文本，它们是被嵌套元素的子元素。HTML 浏览器或其他媒介可以从上下文识别出元素的闭合端及由 HTML 标准定义的结构规则，这些规则非常

复杂。下面将简单介绍 HTML 常见的元素与嵌套。

HTML 文档的头部元素为< head >…</head >。标题元素也会被嵌套在头部元素中,示例如下:

```
< head >
    < title > Title </title >
</head >
```

HTML 的标题元素由< h1 >~< h6 >六个标签构成,字体由大到小递减,示例如下:

```
< h1 >标题 1 </h1 >
< h2 >标题 2 </h2 >
< h3 >标题 3 </h3 >
< h4 >标题 4 </h4 >
< h5 >标题 5 </h5 >
< h6 >标题 6 </h6 >
```

HTML 的段落元素的示例如下:

```
< p >第一段</p >
< p >第二段</p >
```

HTML 的换行标签为< br >。< br >与< p >之间的差异在于,br 换行但不改变页面的语义结构,而 p 元素的内容成段,示例如下:

```
< p >
这是一个< br >使用 br < br >换行< br >的段落。
</p >
```

HTML 使用< a >标签来为文本创建链接,href 属性为链接的 URL 地址,示例如下:

```
< a href = "http://www.baidu.com">一个指向百度的链接</a >
```

HTML 使用<!-- -->来标记注释,示例如下:

```
<!-- 这是一行注释 -- >
```

大多数元素的属性值由单引号或双引号包围,有些属性值的内容包含特定字符,在 HTML 中可以去掉引号。注意,许多元素存在一些共通属性,常见的共通属性的介绍如下。

(1) id 属性为元素提供了在全文档内的唯一标识。它用于识别元素,以便样式表可以改变其表现属性,脚本也可以改变、显示、删除其内容或对其进行格式化。

(2) class 属性提供一种将类似元素分类的方式,常被用于语义化或格式化。例如,一个 HTML 文档可指定类 class="标记"来表明所有具有这一类值的元素都从属于文档的主文本。格式化后,这样的元素可能会聚集在一起并作为页面脚注,而不会出现在 HTML 代码中。类值也可进行多重声明。例如,class="标记 重要"将元素同时放入"标记"与"重要"两类中。

(3) style 属性可以将表现形式赋予一个特定元素。与使用 id 属性或 class 属性从样

式表中选择元素并赋予样式相比，使用 style 属性被认为是一个更好的做法，但有时这对一个简单、专用的样式来说显得太烦琐。

(4) title 属性用于给元素附加说明。在大多数浏览器中，这一属性显示为工具提示。

3.1.2　HTTP

HTTP(Hypertext Transfer Protocol，超文本传输协议)是用于客户端(用户)和服务器端(网站)传输超文本数据(如 HTML 文档)的协议，简单来说，就是客户端和服务器端进行数据传输的一种规则。通过使用网页浏览器、网络爬虫或者其他工具，客户端可以发起一个 HTTP 请求到服务器的指定端口(默认端口号为 80)，一般称这个客户端为用户代理(User Agent)。应答的服务器上存储着一些资源，如 HTML 文件和图像，一般称这个应答服务器为源服务器(Origin Server)。一旦收到请求，应答服务器会向客户端返回一个状态(如"HTTP/1.1 200 OK")及对应内容(如请求的文件、错误消息或者其他信息)。

在用户代理和源服务器中间可能存在多个“中间层”，如代理服务器、网关或者隧道(Tunnel)等。HTTP 中，并没有规定必须使用某一种特定的中间层协议(如互联网上非常流行的 TCP 传输协议)。事实上，HTTP 可以在任何互联网协议上或其他网络上实现。HTTP 假定其下层协议提供可靠的传输。因此，任何能够提供这种保证的协议都可以被其使用。

HTTP 使用统一资源标识符(Uniform Resource Identifier，URI)来连接、传输特定的资源文件。统一资源定位符(Uniform Resource Locator，URL)是一种特殊类型的 URI，它包含了用于查找某个资源的充足的信息，是互联网上用来定位某一处资源的地址。示例 URL 如下所示：

```
http://www.example.com/path/to/resource?param1=value1&param2=value2#section1
```

可以看出，一个完整的 URL 包括以下 6 部分。

(1) 协议(Protocol)。这指的是用于访问资源的通信协议，如 HTTP、HTTPS、FTP(File Transfer Protocol)等。在 URL 中，协议通常在冒号(:)之前，如 http:// 或 https://。HTTPS 是 HTTP 的安全版本，利用 TLS/SSL 协议进行加密，确保数据在传输过程中的安全性和完整性。HTTPS 通常用于保护网站用户的隐私和敏感信息，如登录凭证、支付信息等。FTP 是一种用于在网络上进行文件传输的标准协议，它是明文传输，安全性较低。

(2) 域名(Domain Name)。它是由一串用点分隔的字符组成的互联网上某一台计算机或计算机组的名称，用于在数据传输时标识计算机的电子方位。域名可以说是 IP 地址的代称，目的是便于记忆 IP 地址，它通常包括主机名和域名后缀。例如在 example.com 中，example 是主机名，com 是域名后缀。域名是一个经过注册的互联网标识符，由注册机构(如 ICANN)管理。通常，在浏览器中输入域名时，浏览器会通过域名系统(DNS)将其解析为对应的 IP 地址，然后连接到该 IP 地址的服务器上获取网页或其他资源。

(3) 端口(Port)。端口部分是可选的,如果未明确指定,就使用协议的默认端口。HTTP 协议的默认端口号是 80,而 HTTPS 的默认端口号是 443。如果使用了非默认端口,则会在域名后面加上冒号和端口号。

(4) 路径(Path):路径指定了服务器上资源的具体位置。它是由/分隔的一系列文件夹名或文件名。

(5) 查询参数(Query Parameters)。查询参数是可选的,用于向服务器传递额外的信息,通常以?开头,参数之间使用 & 分隔,如?key1=value1&key2=value2。

(6) 片段标识符(Fragment Identifier)。片段标识符指定了资源中的特定片段或位置,以#开头,如#section1。

HTTP 的请求方法有很多种,在编写爬虫时需要理解的主要有以下 7 种。

(1) GET:向服务器请求获取特定资源的内容。通常用于读取数据,不应该用于影响服务器状态的操作,其中一个原因是 GET 方法可能会被网络爬虫等随意访问。

(2) HEAD:类似于 GET 方法,但服务器不会返回资源的实际内容,只会返回资源的元信息(如大小、类型等)。该方法适用于仅需要获取资源信息而不需要内容的情况。

(3) POST:向服务器提交数据,请求服务器进行处理,如提交表单或上传文件。该方法通常会导致服务器创建新的资源或修改现有资源。

(4) PUT:将请求中的数据上传指定的资源位置,用于更新资源的内容。

(5) DELETE:请求服务器删除指定资源。

(6) TRACE:回显服务器收到的请求,主要用于测试或诊断。

(7) OPTIONS:使服务器传回该资源所支持的所有 HTTP 请求方法。用'*'代替资源名称,向 Web 服务器发送 OPTIONS 请求,可以测试服务器功能是否正常运作。

这些方法的名称是区分大小写的。当某个请求所针对的资源不支持对应的请求方法时,服务器应返回状态码 405 Method Not Allowed;当服务器不认识或者不支持对应的请求方法时,应返回状态码 501 Not Implemented。此外,还有以下 12 种常见的返回状态码。

(1) 200 OK:请求成功。服务器成功处理了请求。

(2) 201 Created:请求已经被实现,并且有一个新的资源已经根据请求的需要而建立。

(3) 204 No Content:服务器成功处理了请求,但没有返回任何内容。

(4) 400 Bad Request:客户端发送的请求无效,服务器无法理解。

(5) 401 Unauthorized:请求需要身份验证。客户端需要提供有效的身份验证信息。

(6) 403 Forbidden:服务器拒绝请求。客户端没有权限访问请求的资源。

(7) 404 Not Found:服务器无法找到请求的资源。

(8) 500 Internal Server Error:服务器遇到了一个未知的错误,无法完成请求。

(9) 502 Bad Gateway:服务器作为网关或代理,从上游服务器收到无效响应。

(10) 503 Service Unavailable:服务器当前无法处理请求,通常是由于临时的过载或维护。

(11) 504 Gateway Timeout:服务器作为网关或代理,未能及时从上游服务器接收请求。

(12) 418 I'm A Teapot：一些服务器使用这个响应来处理它们不想处理的请求，如爬虫程序发出的请求。

3.1.3　JavaScript

JavaScript 一般被定义为一种面向对象、动态类型的解释性语言，最初由 Netscape (网景)公司推出，目的是作为新一代浏览器的脚本语言，换句话说，JavaScript 不是为“网站服务器”提供的语言(如 PHP)，而是为“用户浏览器”提供的语言。从客户端-服务器端的角度来说，JavaScript 无疑是一种“客户端”语言。但是由于 JavaScript 受到业界的热烈欢迎，并且开发者社区非常活跃，因此目前的 JavaScript 已经开始朝更为综合的方向发展。随着 V8 引擎(可以提高 JavaScript 的解释执行效率)和 Node.js(专为服务器端定制的 JavaScript)等新潮流的出现，JavaScript 已经开始涉足“服务器端”。在 TIOBE 排名(关于程序设计语言受欢迎度的排名)上，JavaScript 稳居前 10 名。对于一个正式的网页来说，一种经典技术搭配是：HTML 决定网页的基本内容，CSS 描述网页的样式布局，JavaScript 控制用户与网页的交互。

注意，很多人会将 JavaScript 与 Java 联系起来，认为它是 Java 的某种派生语言，实际上 JavaScript 在设计原则上更多地受到了 Scheme(一种函数式编程语言)和 C 的影响。除了变量类型和命名规范等细节，JavaScript 与 Java 的关系并不大。Netscape 公司最初将其命名为 LiveScript，但当时正与 Sun 公司合作，并且 Java 获得了巨大成功，为了“蹭热度”，遂将名字改为 JavaScript。JavaScript 推出后，受到了业界的一致肯定，对 JavaScript 的支持也成为现代浏览器的基本要求。浏览器的脚本语言还包括用于 FLASH 动画的 ActionScript 等。

为了在网页中使用 JavaScript，开发者一般会把 JavaScript 脚本程序写在 HTML 的 < script >标签中，构成< script >元素。在 HTML 语法里，如果需要引用外部脚本文件，可以在 src 属性中设置其地址。引用外部脚本文件的< script >元素示例如图 3-1 所示。

```
▼<script>
      Do(function() {
        var app_qr = $('.app-qr');
        app_qr.hover(function() {
          app_qr.addClass('open');
        }, function() {
          app_qr.removeClass('open');
        });
      });

    </script>
  </div>
▶<div id="anony-sns" class="section">…</div>
▶<div id="anony-time" class="section">…</div>
▶<div id="anony-video" class="section">…</div>
▶<div id="anony-movie" class="section">…</div>
▶<div id="anony-group" class="section">…</div>
▶<div id="anony-book" class="section">…</div>
▶<div id="anony-music" class="section">…</div>
▶<div id="anony-market" class="section">…</div>
▶<div id="anony-events" class="section">…</div>
▼<div class="wrapper">
    <div id="dale_anonymous_home_page_bottom" class="extra"></div>
  ▶<div id="ft">…</div>
  </div>
  <script type="text/javascript" src="https://img3.doubanio.com/f/shire/72ced6d…/js/
  jquery.min.js" async="true"></script> == $0
```

图 3-1　豆瓣首页网页源码中的< script >元素

JavaScript 在语法结构上类似于 C++等面向对象的语言，循环语句、条件语句等与 Python 中的写法有较大的差异，但其弱类型特点会更符合 Python 开发者的使用习惯。

使用 JavaScript 可以实现网页的异步加载，即 AJAX(Asynchronous JavaScript and XML，异步 JavaScript 和 XML)方法。AJAX 不是一种新的编程语言，而是一种新方法，它的优点是在不重新加载整个页面的情况下，可以与服务器交换数据并更新部分网页内容。AJAX 不需要任何浏览器插件，但需要用户允许浏览器执行 JavaScript。

以知乎的首页信息流为例，如图 3-2 所示，其与用户的主要交互方式就是用户通过下拉页面(可通过滚动鼠标滚轮、鼠标拖动滚动条等)查看更多动态，而在一部分动态(对于知乎而言包括用户的点赞和回答等)展示完毕后，就会显示一段加载动画并呈现后续的内容。在这个过程中，页面动画其实只是“障眼法”，背后过程是 JavaScript 脚本请求服务器发送相关数据，并最终将收到的数据加载到页面之中。页面没有进行全部刷新，而是刷新了一部分，通过这种异步加载的方式完成了对新内容的获取和呈现，这个过程就是典型的 AJAX 应用。

图 3-2　知乎首页的动态刷新

由于爬虫没有像浏览器那样执行 JavaScript 脚本的能力，因此不会为网页运行 JavaScript，最终爬取的结果会和浏览器显示的结果有差异，很多时候不能直接获取想要的关键信息。为了解决这个问题，基于 Python 编写的爬虫程序可以做出两种改进。一种改进是通过分析 AJAX 内容(需要开发者手动观察和实验)，观察加载内容时的请求目标、请求内容和请求的参数等信息，最终编写程序来模拟这样的 JavaScript 请求，进而获取信息，这个过程也可以叫作“逆向工程”。另一种改进是直接模拟出浏览器环境，使程序通过浏览器模拟工具运行 JavaScript，最终通过浏览器渲染后的页面获得信息。

3.1.4　Robots 协议

Robots 协议用于管理爬虫行为，指示网络爬虫可以抓取哪些页面。

对于个人编写的实验性质的爬虫而言，一般不会存在法律和道德问题。但随着与互联网知识产权相关的法律法规的逐渐完善，在使用自己的爬虫时，需要遵守网站的规定。2013 年，百度公司以违反 Robots 协议为由，对 360 公司提起了诉讼，指控其通过不正当竞争手段抓取、复制百度网站的内容，并索赔 1 亿元。法院对此案做出了裁决，表示尊重 Robots 协议及对用户原创内容(UGC)数据的保护，360 公司最终被判赔偿百度公司 70 万元。2014 年 8 月，微博宣布停止脉脉对微博开放平台的所有接口的使用权。其理由是脉脉通过恶意抓取行为获取并使用了未经微博用户授权的档案数据，违反了微博开放平台的开发者协议。随着《中华人民共和国网络安全法》的出台，针对企业利用爬虫技

术获取网络上特定信息的行为做出了一些规定。对于个人开发者而言，需要注意：

（1）不应该访问和抓取某些充满不良信息的网站，包括一些充斥暴力、色情或反动信息的网站。

（2）注意版权意识。如果想爬取的信息是其他作者的原创内容，未经作者或版权所有者的授权不应该将这些信息用作其他用途，尤其是商业方面的行为。

（3）保持对网站的善意。如果没有经过网站运营者的同意，使爬虫程序对目标网站的性能产生了一定影响，恶意造成了服务器资源的大量浪费，那么这是不道德的，有时也会触犯相关法律法规。爬虫技术的爱好者并不是一个试图攻击网站的黑客。在编写分布式大规模爬虫时，更需要注意这点。

（4）遵循 Robots 协议和网站服务协议。虽然 Robots 协议并没有强制性约束爬虫程序的能力，但在实际的爬虫编写过程中，仍应该尽可能遵循 Robots 协议的内容。

Robots 协议没有标准的语法，但网站一般都遵循业界共有的习惯。文件的第一行内容是“user-agent：”，表明哪些爬虫（程序）需要遵守下面的规则，然后是一组“allow：”和“disallow：”，决定是否允许该 user-agent 访问网站的这部分内容。星号（*）为通配符。如果一条规则后跟着一条矛盾的规则，那么以后一条规则为准。Robots 协议可能还会规定 Crawl-delay，即爬虫抓取延迟，如果在 Robots 协议中看到“Crawl-delay：5”，那么说明网站希望程序能够在两次下载请求中给出 5s 的下载间隔。

3.2　Python 爬虫基础库编写爬虫

3.2.1　Requests 库采集网页

Python 的 Requests 库是一个强大而简洁的 HTTP 库，它基于 Python 内置的 urllib 库编写。与内置的 urllib 库相比，Requests 库封装了很多方法，使发送 HTTP 请求变得非常简单。Requests 库可以用于与 Web 服务进行通信、爬取网页内容、处理 API 等。使用以下命令安装 Requests 库。

```
pip install requests
```

Requests 库支持各种类型的 HTTP 请求，包括 GET、POST、PUT、DELETE 等，可以根据需要使用相应的方法来发送不同类型的请求，并根据响应做出相应的处理。使用 Requests 库向网页发送一个 GET 请求，并打印返回内容的文本形式，代码如下。

```
import requests
# 发送一个简单的 GET 请求
response = requests.get('https://api.github.com')
# 输出响应内容
print(response.text)
```

返回的输出如下所示。通过访问 https://api.github.com 可以印证，网页内容与 Python 输出的响应内容一致。

```
{"current_user_url":"https://api.github.com/user","current_user_authorizations_html_
url":"https://github.com/settings/connections/applications{/client_id}","authorizations_
url":"https://api.github.com/authorizations","code_search_url":"https://api.github.
com/search/code?q={query}{&page,per_page,sort,order}","commit_search_url":"https://
api.github.com/search/commits?q={query}{&page,per_page,sort,order}","emails_url":
"https://api.github.com/user/emails","emojis_url":"https://api.github.com/emojis",
"events_url":"https://api.github.com/events","feeds_url":"https://api.github.com/
feeds","followers_url":"https://api.github.com/user/followers","following_url":
"https://api.github.com/user/following{/target}","gists_url":"https://api.github.com/
gists{/gist_id}","hub_url":"https://api.github.com/hub","issue_search_url":"https://
api.github.com/search/issues?q={query}{&page,per_page,sort,order}","issues_url":
"https://api.github.com/issues","keys_url":"https://api.github.com/user/keys","label_
search_url":"https://api.github.com/search/labels?q={query}&repository_id={repository_id}
{&page,per_page}","notifications_url":"https://api.github.com/notifications",
"organization_url":"https://api.github.com/orgs/{org}","organization_repositories_
url":"https://api.github.com/orgs/{org}/repos{?type,page,per_page,sort}","organization_
teams_url":"https://api.github.com/orgs/{org}/teams","public_gists_url":"https://api.
github.com/gists/public","rate_limit_url":"https://api.github.com/rate_limit",
"repository_url":"https://api.github.com/repos/{owner}/{repo}","repository_search_
url":"https://api.github.com/search/repositories?q={query}{&page,per_page,sort,
order}","current_user_repositories_url":"https://api.github.com/user/repos{?type,page,
per_page,sort}","starred_url":"https://api.github.com/user/starred{/owner}{/repo}",
"starred_gists_url":"https://api.github.com/gists/starred","topic_search_url":
"https://api.github.com/search/topics?q={query}{&page,per_page}","user_url":"https://
api.github.com/users/{user}","user_organizations_url":"https://api.github.com/user/
orgs","user_repositories_url":"https://api.github.com/users/{user}/repos{?type,page,per_
page,sort}","user_search_url":"https://api.github.com/search/users?q={query}{&page,per_
page,sort,order}"}
```

将网址更改为 https://www.douban.com,代码如下。

```
import requests

# 发送一个简单的 GET 请求
response = requests.get('https://www.douban.com')
# 输出响应内容
print(response.text)
```

此段代码没有输出,即 response.text 没有内容。通过浏览器访问 https://www.douban.com,发现可以正常访问首页内容,这说明不是服务器端出现了问题,而是发出的请求出现了问题。requests 的 get 方法返回的响应对象 Response(即代码中的 response 实例)拥有 status_code 属性,可以快速查看响应的状态码,代码如下。

```
print('response.status_code:', response.status_code)
```

得到的输出如下。

```
response.status_code: 418
```

正常情况下,状态码应为 200OK,代表成功访问。状态码 418 I'm A Teapot 代表网

站服务器识别出这是一个由爬虫程序而非浏览器发送的请求，所以拒绝返回 URL 对应的内容。在这种情况下，需要将请求伪装成是由浏览器发送的。

网站通常会使用多种方法来识别程序发送的请求，最常用的方法是检查请求头中 User-Agent 字段的值。每个浏览器和程序都有一个唯一的用户代理字符串，其中包含了关于发送请求的软件的信息。网站可以通过检查这个字符串来确定请求是由浏览器还是程序发送的。使用以下代码查看程序默认的请求头。

```
default_headers = requests.utils.default_headers()
# 打印默认请求头
for header, value in default_headers.items():
    print(f"{header}: {value}")
```

输出如下。

```
User-Agent: python-requests/2.26.0
Accept-Encoding: gzip, deflate, br
```

可以看到，默认的请求头里 User-Agent 字段的值为 python-requests/2.26.0，即使用的编程语言和发出请求的包的名称。

如果想骗过浏览器，则需要使用浏览器的请求头来发送请求。在 Requests 库中，只需要在发送请求时指定 headers 参数即可伪装成浏览器的请求。浏览器的请求头可以从网络上查找，也可以进入浏览器的开发者模式，找到任意请求并复制其请求头。更改请求头的代码如下。

```
import requests
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/100.0.4896.75 Safari/537.36'
}
# 发送一个简单的 GET 请求
response = requests.get('https://www.douban.com', headers=headers)
# 输出响应内容
print(response.text)
```

此时的输出如下所示，由于输出内容太多，因此对部分内容做了省略。可以看出，返回的是一个完整的 HTML 文档，即未经过浏览器解析的网页源文件。

```
<!DOCTYPE HTML>
<html lang="zh-cmn-Hans" class="ua-windows ua-webkit">
<head>
<meta charset="UTF-8">
<meta name="google-site-verification" content="ok0wCgT20tBBgo9_zat2iAcimtN4Ftf5ccsh092Xeyw" />
<meta name="description" content="提供图书、电影、音乐唱片的推荐、评论和价格比较，以及
城市独特的文化生活。">
<meta name="keywords" content="豆瓣，小组，电影，同城，豆品，广播，登录豆瓣">
<meta property="qc:admins" content="2554215131764752166375" />
<meta property="wb:webmaster" content="375d4a17a4fa24c2" />
<meta name="mobile-agent" content="format=html5; url=https://m.douban.com">
```

```
<title>豆瓣</title>
<link rel="stylesheet" href="https://img1.doubanio.com/f/vendors/0035bb2f83e2cba49ecf
634fed57f9ff1bbd0d09/css/ui/dialog.css">
<link rel="stylesheet" href="https://img1.doubanio.com/f/vendors/3a8b90f5419888f58be
10eaba23e024bb4caf9c3/css/core/_init_.css">
<link rel="stylesheet" href="https://img1.doubanio.com/f/sns/bb6b4ad0c8690c51076d61d
6c101c842cd97ba1d/css/sns/anonymous_home/index.css">
<!-- COLLECTED CSS -->

<script type="text/javascript" src="https://img1.doubanio.com/f/vendors/0511abe9863c2
ea7084efa7e24d1d86c5b3974f1/js/jquery-1.10.2.min.js"></script>
<script src="https://img1.doubanio.com/f/vendors/b0d3faaf7a432605add54908e39e17746824
d6cc/js/separation/_all.js"></script>
<script src="https://img1.doubanio.com/f/vendors/e057439e70105417dffc6fab571688d52efe
ab23/js/douban.js"></script>
<script src="https://img1.doubanio.com/f/vendors/084b39fa262eabe5828059b3e8072184589b
6b89/js/core/_init_.js"></script>
<script src=""></script>
<script src="https://img1.doubanio.com/f/vendors/f25ae221544f39046484a823776f3aa01769
ee10/js/ui/dialog.js"></script>
<script src="https://img1.doubanio.com/f/sns/c714e1dc3cceb07b6e7c095e01fe136cf79726b1/
js/sns/fp/base.js"></script>
<script src="https://img1.doubanio.com/f/sns/6a6ebb88ef379a31fe198305b7cd75aafa3314f4/
js/sns/fp/lazypic.js"></script>
<script src="https://img1.doubanio.com/f/sns/8360a10d497f46c162c6c527954f580eedc4d4e0/
js/sns/fp/inp_label.js"></script>

<script src="https://img1.doubanio.com/f/vendors/7b710436122e209e64be54f3302aaae246f2
1273/js/lib/head.js"></script>
</head>

<body class=''>
...
</body>
</html>
```

至此,使用程序请求并获取目标页面的任务就完成了。可以看到,此时只得到了一份 HTML 文档,可读性非常差。从原始的 HTML 文档中解析关注的信息则需要其他的工具。

3.2.2 BeautifulSoup 库解析网页

BeautifulSoup 是一个 Python 库,用于从 HTML 和 XML 文档中提取数据。它提供了一种简单的方式来浏览、搜索和修改解析树,使得在网络爬虫和数据挖掘等任务中提取网页信息变得非常方便。

由于 BeautifulSoup 库并不是 Python 内置的,因此需要使用 pip 来安装。这里安装最新的版本(BeautifulSoup 4,也称为 bs4)。安装命令如下。

```
pip install beautifulsoup4
```

还可以使用简称来安装，命令如下。

```
pip install bs4
```

如果在安装中遇到了问题，可以访问 https://www.crummy.com/software/BeautifulSoup/bs4/doc/获得帮助。

BeautifulSoup 库中的主要工具就是 BeautifulSoup 对象，这个对象的意义是一个 HTML 文档的全部内容。使用以下代码，将 3.2.1 节获取的 HTML 文档转换为 BeautifulSoup 对象，并且以方便阅读的形式将 HTML 文档的内容打印出来。

```
import requests
from bs4 import BeautifulSoup
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/100.0.4896.75 Safari/537.36'
}
ht = requests.get('https://www.douban.com', headers=headers)
bs = BeautifulSoup(ht.content)
print(bs.prettify())
```

截取一部分输出如下所示。可以看出，通过使用 prettify()方法，输出的内容加入了缩进，能够使读者更好地分析 HTML 文档内的嵌套结构。

```
<!DOCTYPE HTML>
<html class="ua-windows ua-webkit" lang="zh-cmn-Hans">
 <head>
  <meta charset="utf-8"/>
  <meta content="ok0wCgT20tBBgo9_zat2iAcimtN4Ftf5ccsh092Xeyw" name="google-site-
verification"/>
  <meta content="提供图书、电影、音乐唱片的推荐、评论和价格比较，以及城市独特的文化生
活。" name="description"/>
  <meta content="豆瓣，小组，电影，同城，豆品，广播，登录豆瓣" name="keywords"/>
  <meta content="255421513176475216637 5" property="qc:admins"/>
  <meta content="375d4a17a4fa24c2" property="wb:webmaster"/>
  <meta content="format=html5; url=https://m.douban.com" name="mobile-agent"/>
  <title>
   豆瓣
  </title>
```

使用以下代码来查看 HTML 文档中的元素。

```
print('title:', bs.title)
print('title.name:', bs.title.name)
print('title.text:', bs.title.text)
print('title.parent.name:', bs.title.parent.name)
```

输出如下。

```
title: <title>豆瓣</title>
title.name: title
```

```
title.text: 豆瓣
title.parent.name: head
```

可以看到,通过 title 即可方便地访问 HTML 文档中的 title 元素,通过 text 可以访问对应元素中的文本内容。还可以通过 parent 元素轻松访问对应元素节点的父节点。

若元素在 HTML 文档中出现多次,则需要使用 find_all()方法。使用 find_all()方法获取的结果都是由 Tag 对象组成的列表,Tag 对象也是 BeautifulSoup 库中的主要对象之一,它在逻辑上与 XML 或 HTML 文档中的标签相同,可以使用 name 和 attrs 来访问 Tag 对象的名字和属性,还可以使用类似字典的方法获取属性,以名为 tag 的 Tag 对象为例,获取其 href 属性的方法为 tag['href']。

find_all()方法的定义如下。

```
find_all(name, attrs, recursive, text, **kwargs)
```

该方法搜索当前这个 Tag 对象(这时 BeautifulSoup 对象可以被视为一个 Tag 对象,是所有 Tag 对象的根)的所有子节点,并判断是否符合搜索条件。name 参数可以查找所有名字为 name 的 Tag 对象,示例如下。

```
bs.find_all('tagname')
```

find_all()方法在搜索时支持根据元素是否含有某一属性来搜索,代码如下。

```
bs.find_all(href = 'https://book.douban.com').text
```

其结果应该是“豆瓣读书”。也可以同时使用多个属性来搜索,通过 find_all()方法的 attrs 参数定义一个字典参数来搜索多个属性,代码如下。

```
bs.find_all(attrs = {"href": re.compile('time'),"class":"title"})
```

可以看到,这行代码将所有"href"属性中含有'time'并且"class"属性为"title"的内容都找了出来。这行代码中还出现了 re.compile()方法,也就是说使用了正则表达式,如果传入正则表达式作为参数,BeautifulSoup 库会通过正则表达式的 match()方法来匹配内容。输出如下。

```
<a class = "title" href = "https://m.douban.com/time/column/120?dt_time_source = douban -
web_anonymous" target = "_blank">一个作家的养成——写作成长营名师直播课精选</a>
<a class = "title" href = "https://m.douban.com/time/column/265?dt_time_source = douban -
web_anonymous" target = "_blank">邂逅风骨、风度与风流——魏晋名士的生活美学</a>
<a class = "title" href = "https://m.douban.com/time/column/213?dt_time_source = douban -
web_anonymous" target = "_blank">我们的女性 400 年——文学里的女性主义简史</a>
<a class = "title" href = "https://m.douban.com/time/column/136?dt_time_source = douban -
web_anonymous" target = "_blank">生存大作战——植物世界里的八大超能力</a>
<a class = "title" href = "https://m.douban.com/time/column/246?dt_time_source = douban -
web_anonymous" target = "_blank">在故宫发现中国之美——5 位故宫专家带你走近珍藏国宝</a>
```

```
<a class="title" href="https://m.douban.com/time/column/101?dt_time_source=douban-
web_anonymous" target="_blank">花鸟鱼虫的生活意见——博物君的自然笔记</a>
<a class="title" href="https://m.douban.com/time/column/91?dt_time_source=douban-
web_anonymous" target="_blank">一个故事的诞生——22堂创意思维写作课</a>
<a class="title" href="https://m.douban.com/time/column/267?dt_time_source=douban-
web_anonymous" target="_blank">不能承受的生命之轻——听复旦教授梁永安深度解读</a>
<a class="title" href="https://m.douban.com/time/column/104?dt_time_source=douban-
web_anonymous" target="_blank">喝咖啡时遇见巴赫——听懂“音乐之父”的经典名曲</a>
<a class="title" href="https://m.douban.com/time/column/199?dt_time_source=douban-
web_anonymous" target="_blank">复旦沈奕斐的社会学爱情思维课</a>
```

BeautifulSoup 库还支持根据 CSS 来搜索，这时要使用“class_=”的形式，因为 class 在 Python 中是一个保留关键词。示例代码如下。

```
bs1.find_all(class_='video-title')
```

recursive 参数默认设置为 True，BeautifulSoup 库会检索当前 Tag 对象的所有子孙节点，如果只想搜索 Tag 对象的直接子节点，可以设置 recursive 参数为 False。

通过 text 参数可以搜索文档中的字符串内容，示例代码如下。

```
bs1.find_all(text=re.compile('银翼杀手'))[0].parent['href']
```

输出结果是'https://movie.douban.com/subject/10512661/'，这是电影《银翼杀手 2049》的豆瓣电影主页。当使用 text 参数时，find_all()返回的结果是一个字符串(NavigableString，就是指一个 Tag 对象中的字符串)列表，所做的是使用 parent 访问列表第一个元素所在的 Tag 对象然后获取 Tag 对象的 href 属性。

find_all()函数会返回全部的搜索结果，所以如果文档的树结构很大，那么很可能并不需要全部结果，limit 参数可以限制返回结果的数量。当搜索数量达到 limit 参数设置的值就会停止搜索。BeautifulSoup 库中还有一个 find()方法实际上就是当 limit 参数为 1 时的 find_all()方法。

由于 find_all()函数如此常用，因此在 BeautifulSoup 库中，BeautifulSoup 对象和 Tag 对象可以被当作一个 find_all()函数来使用，也就是说下面两行代码是等效的。

```
bs.find_all("a")
bs("a")
```

下面两行代码也是等价的。

```
soup.title.find_all(text="abc")
soup.title(text="abc")
```

至此，BeautifulSoup 库解析网页的基本方法已经介绍完毕。只要利用这些解析函数，将豆瓣首页中关心的内容保存到本地，即可完成一个简单的爬虫。Python 保存内容的方法非常多，这里不作示例说明。

严格地说，一个只处理单个静态页面的程序不能称为爬虫，只能算是一种简化的网

页抓取脚本。实际的爬虫程序面对的任务是根据某种抓取逻辑，重复遍历多个页面，甚至多个网站。在处理当前页面时，爬虫就应该考虑确定下一个将要访问的页面，下一个页面的链接地址有可能在当前页面的某个元素中，也可能从特定的数据库中读取（这取决于爬虫的爬取策略），通过从“爬取当前页”到“进入下一页”的循环，实现整个爬取过程。这个过程可以利用 Python 来实现，但是对于初学者来说，这需要大量的时间，同时其代码可能存在各种难以发现的问题，降低爬取效率。因此，3.3 节将介绍如何利用 Python 编写的爬虫框架快速开发优质爬虫程序，免去对实现细节的考虑，提高开发效率。

3.3 Scrapy 框架构建爬虫

3.3.1 Scrapy 框架简介

3.2 节介绍了如何使用 Requests 和 BeautifulSoup 两个 Python 第三方库来抓取、解析页面，进而实现数据爬取。初学者在进行上述爬虫工具的实践时，会发现一些问题，例如一部分代码比较通用，但是将其用到新项目时又不像导入第三方库那样方便；爬取出现问题需要调试时，缺少合适的调试工具；存储爬取的数据，尤其是需要调整存储格式时，涉及的代码很烦琐；想添加如延迟策略、代理 IP、多线程爬取等高级功能时，可能需要重构代码，时间成本太高，也容易出现隐藏很深的错误等。采用一个标准的框架来解决上述问题非常重要。

在各种 Python 爬虫框架中，Scrapy 框架因为合理的设计、简便的用法和十分广泛的资料等优点脱颖而出，成为比较流行的爬虫框架选择。Scrapy 框架的官网是 https://scrapy.org/，读者可以随时访问并查看最新的消息，这里对其进行简单的介绍。

从构件上看，Scrapy 框架主要由以下组件组成。

（1）引擎（Scrapy）：用来处理整个系统的数据流处理，触发事务，它是框架的核心。

（2）调度器（Scheduler）：用来接受引擎发过来的请求，将请求放入队列中，并在引擎再次请求时返回。它决定下一个要抓取的网址，同时担负着网址去重这一重要工作。

（3）下载器（Downloader）：用于下载网页内容，并将网页内容返回给爬虫。下载器的基础是 twisted，这是一个 Python 网络引擎框架。

（4）爬虫（Spiders）：用于从特定的网页中提取自己需要的信息，即 Scrapy 中所谓的实体（Item）。也可以从中提取出链接，让 Scrapy 继续抓取下一个页面。

（5）数据管道（Data Pipeline）：负责处理爬虫从网页中抽取的实体，主要的功能是持久化信息、验证实体的有效性、清洗信息等。当页面被爬虫解析后，将被发送到管道，并经过特定的程序来处理数据。

（6）下载器中间件（Downloader Middlewares）：引擎和下载器之间的框架，主要工作是处理引擎与下载器之间的请求及响应。

（7）爬虫中间件（Spider Middlewares）：引擎和爬虫之间的框架，主要工作是处理爬虫的响应输入和请求输出。

它们之间的关系示意如图 3-3 所示。

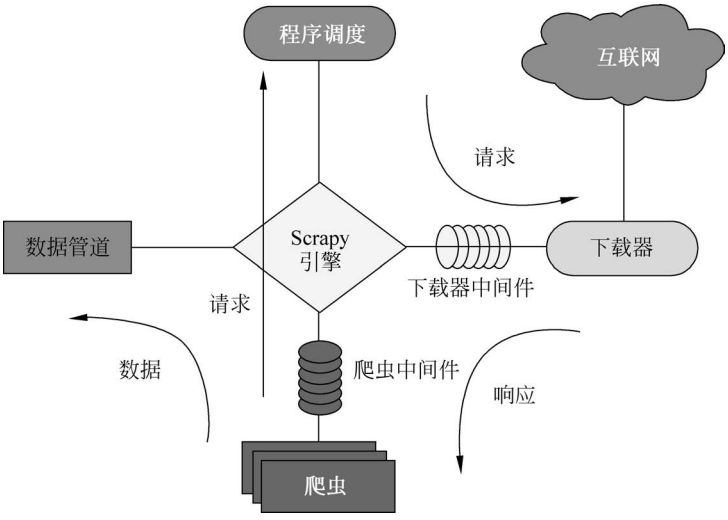

图 3-3　Scrapy 框架架构的组件之间的关系

具体来说，一个爬虫的工作流程如下。

(1) 引擎打开一个网站，找到处理该网站的爬虫，并向该爬虫请求第一个要爬取的 URL。

(2) 引擎从爬虫中获取到第一个要爬取的 URL 并在程序调度器中以请求调度。

(3) 引擎向程序调度器请求下一个要爬取的 URL。

(4) 程序调度器返回下一个要爬取的 URL 给引擎，引擎将 URL 通过下载器中间件转发给下载器。

(5) 一旦页面下载完毕，下载器会生成一个该页面的响应，并将其通过下载器中间件发送给引擎。

(6) 引擎从下载器中接收到响应并通过爬虫中间件发送给爬虫处理。

(7) 爬虫处理响应并返回爬取到的实体及发送新的响应给引擎。

(8) 引擎将爬取到的实体传递给数据管道，将爬虫返回的响应传递给调度器。

重复步骤(2)开始的过程直到调度器中没有更多的响应，最终引擎关闭网站。

3.3.2　Scrapy 框架安装

可以通过 pip 十分轻松地安装 Scrapy 框架。安装 Scrapy 框架首先需要使用以下命令安装 lxml 库：

```
pip install lxml
```

如果已经安装 lxml 库，就可以直接安装 Scrapy 框架，命令如下。

```
pip install scrapy
```

在终端中执行命令(后面的网址可以是其他域名，如 www.baidu.com)，命令如下。

```
scrapy shell www.douban.com
```

可以看到 Scrapy shell 的反馈，如图 3-4 所示。

```
[s] Available Scrapy objects:
[s]   scrapy     scrapy module (contains scrapy.Request, scrapy.Selector, etc)
[s]   crawler    <scrapy.crawler.Crawler object at 0x1053c0b70>
[s]   item       {}
[s]   request    <GET http://www.douban.com>
[s]   response   <403 http://www.douban.com>
[s]   settings   <scrapy.settings.Settings object at 0x10633b358>
[s]   spider     <DefaultSpider 'default' at 0x106682ef0>
[s] Useful shortcuts:
[s]   fetch(url[, redirect=True]) Fetch URL and update local objects (by default, redirect
s are followed)
[s]   fetch(req)                  Fetch a scrapy.Request and update local objects
[s]   shelp()           Shell help (print this help)
[s]   view(response)    View response in a browser
```

图 3-4　Scrapy shell 的反馈

使用 scrapy -v 命令可以查看目前安装的 Scrapy 框架的版本，如图 3-5 所示。

```
Scrapy 1.4.0 - no active project

Usage:
  scrapy <command> [options] [args]

Available commands:
  bench         Run quick benchmark test
  fetch         Fetch a URL using the Scrapy downloader
  genspider     Generate new spider using pre-defined templates
  runspider     Run a self-contained spider (without creating a project)
  settings      Get settings values
  shell         Interactive scraping console
  startproject  Create new project
  version       Print Scrapy version
  view          Open URL in browser, as seen by Scrapy

  [ more ]      More commands available when run from project directory

Use "scrapy <command> -h" to see more info about a command
```

图 3-5　查看 Scrapy 版本

看到这些信息就说明已经安装成功。

3.3.3　Scrapy 框架爬虫编写

为了创建一个 Scrapy 爬虫的项目，首先进入想要存放项目的目录下，这里在终端中使用命令创建一个新目录并进入：

```
mkdir newcrawler
cd newcrawler/
```

之后执行 Scrapy 框架的对应命令：

```
scrapy startproject newcrawler
```

```
newcrawler/
└── newcrawler
    ├── newcrawler
    │   ├── __init__.py
    │   ├── __pycache__
    │   ├── items.py
    │   ├── middlewares.py
    │   ├── pipelines.py
    │   ├── settings.py
    │   └── spiders
    │       ├── __init__.py
    │       └── __pycache__
    └── scrapy.cfg
```

图 3-6　newcrawler 目录的结构

可以发现目录下多出了一个名为 newcrawler 的目录，查看这个目录的结构，如图 3-6 所示，这是一个标准的 Scrapy 框架的爬虫项目结构。

其中 items.py 定义了爬虫的“实体”类，middlewares.py 是中间件文件，pipelines.py 是管道文件，spiders 文件夹下是具体的爬虫，scrapy.cfg 是爬虫的配置文件。

然后执行新建爬虫的命令：

```
scrapy genspider DoubanSpider douban.com
```

输出如下。

```
Created spider 'DoubanSpider' using template 'basic'
```

不难发现，genspider 命令创建了一个名为 DoubanSpider 的新爬虫脚本，这个爬虫对应的域为 douban.com。在输出中发现了一个名为 basic 的模板，这其实是 Scrapy 框架默认的基础爬虫模板。进入 DoubanSpider.py 中查看，如图 3-7 所示。

```
# -*- coding: utf-8 -*-
import scrapy


class DoubanspiderSpider(scrapy.Spider):
    name = 'DoubanSpider'
    allowed_domains = ['douban.com']
    start_urls = ['http://douban.com/']

    def parse(self, response):
        pass
```

图 3-7　DoubanSpider.py

可见它继承了 scrapy.Spider 类，其中还有一些类属性和方法。name 用来标识爬虫，它在项目中是唯一的，每一个爬虫有一个独特的 name。parse()是一个处理响应的方法，在 Scrapy 框架中，响应由每个请求生成。作为 parse()方法的参数，response 是 TextResponse 类的实例，它保存了页面的内容。start_urls 列表是一个代替 start_requests()方法的捷径。start_requests()方法的任务是从 URL 生成 scrapy.Request 对象，作为爬虫的初始请求。之后遇到的 Scrapy 框架爬虫基本都有着类似这样的结构。

进入 items.py 文件，会看到以下内容：

```
# -*- coding: utf-8 -*-

# Define here the models for your scraped items
#
# See documentation in:
# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy


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

为了定制 Scrapy 框架的爬虫，要根据自己的需求定义不同的实体，例如，创建一个针对页面中所有正文文字的爬虫，将 Items.py 的内容改写为：

```
class TextItem(scrapy.Item):
    # define the fields for your item here like:
    text = scrapy.Field()
```

之后编写 DoubanSpider.py 文件：

```
# - * - coding: utf-8 - * -
import scrapy
from scrapy.selector import Selector
from ..items import TextItem


class DoubanspiderSpider(scrapy.Spider):
    name = 'DoubanSpider'
    allowed_domains = ['douban.com']
    start_urls = ['https://www.douban.com/']

    def parse(self, response):
        item = TextItem()
        h1text = response.xpath('//a/text()').extract()
        print("Text is" + ''.join(h1text))
        item['text'] = h1text
        return item
```

注意,一个爬虫项目可以有多个不同的爬虫类,因为有时需要在一组网页中收集不同类别的信息(如一个电影介绍网页的演员表、剧情简介、海报图片等),可以为它们设定独立的实体类,再用不同的爬虫进行爬取。

这个爬虫会先进入 start_urls 列表中的页面(在这个例子中就是豆瓣网的首页),收集信息完毕后,确认没有未访问的 URL 就会停止。response.xpath('//a/text()').extract()这行语句将从 response(其中保存着网页信息)中使用 xpath 语句抽取出所有 a 标签的文字内容(text)。下一行语句会将它们逐一打印。

在运行这个简单的 Scrapy 框架的爬虫之前,进入 settings.py 文件中查看爬虫的默认设置,部分文件内容如下。

```
# Obey robots.txt rules
ROBOTSTXT_OBEY = True

# Configure maximum concurrent requests performed by Scrapy (default: 16)
#CONCURRENT_REQUESTS = 32

# Configure a delay for requests for the same Website (default: 0)
# See http://scrapy.readthedocs.org/en/latest/topics/settings.html#download-delay
# See also autothrottle settings and docs
#DOWNLOAD_DELAY = 3
```

具体细节如下。

如果启用 ROBOTSTXT_OBEY,Scrapy 框架就会遵循 Robots 协议的内容。

CONCURRENT_REQUESTS 设定了并发请求的最大值,在这里是被注释掉的,也就是说没有限制最大值。

DOWNLOAD_DELAY 设定了下载器在下载同一个网站的不同页面时需要等待的时间间隔。通过设置该选项,可以限制程序的爬取速度,减轻服务器压力。

另外一些 settings.py 中的重要设置包括:

(1) BOT_NAME:Scrapy 框架的爬虫项目的 bot 名称,使用 startproject 命令创建

项目时会自动赋值。

（2）ITEM_PIPELINES：保存项目中启用的管道及其对应顺序，使用一个字典结构。字典默认为空，值（value）一般设定在0～1000范围，数字越小代表优先级越高。

（3）LOG_ENABLED：是否启用日志记录，默认为True。

（4）LOG_LEVEL：设定开始记录日志的级别。

（5）USER_AGENT：默认的用户代理。

运行Scrapy框架的爬虫脚本后，往往会生成大量的程序调试信息，这对于观察程序的运行状态是很有用的。为了简洁，可以设置LOG_LEVEL。Python中的log级别一般有DEBUG、INFO、WARNING、ERROR、CRITICAL等，其“严重性”逐渐增长，包含的范围逐渐缩小。当把LOG_LEVEL设置为ERROR时，只显示ERROR和CRITICAL级别的日志。日志不仅可以在终端显示，也可以用Scrapy框架的命令行工具输出到文件中。

为了让爬虫看起来更像一个浏览器，需要更改默认的USER_AGENT参数。将USER_AGENT参数赋值的那一行代码取消注释并编辑，内容如下。

```
USER_AGENT = 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/36.0.1985.125 Safari/537.36'
```

完成设置后，可以开始运行这个爬虫，命令如下。

```
scrapy crawl spidername
```

其中，spidername是爬虫的名称，即爬虫类中的name属性。

运行程序并进行爬取后，可以看到类似图3-8所示的输出，说明爬虫成功进行了爬取。

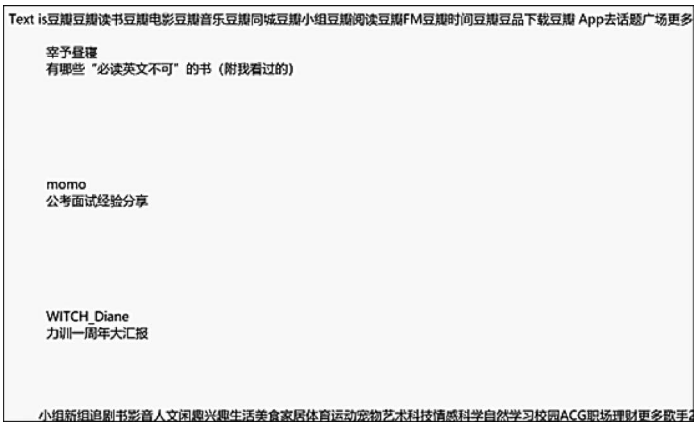

图3-8　程序运行后的输出

3.4　Selenium库模拟人工爬虫

3.4.1　Selenium库简介

Selenium库是一个用于浏览器应用程序测试的工具，它直接运行在浏览器中，通过

对网页的各种操作(如单击、滑动等)来模拟用户操作。框架底层利用 JavaScript 代码实现测试脚本,调用相应接口后,浏览器会自动按照脚本代码做出操作,可以从终端用户的角度测试应用程序。

Selenium 库本身只是个工具,而不是一个具体的浏览器,但是 Selenium 库支持包括 Chrome 和 Firefox 在内的主流浏览器。使用 Selenium 库编写爬虫,可以完全模拟人工的操作,被反爬虫策略限制的可能性大大降低。但是由于 Selenium 库需要操作浏览器,因此占用的资源较多,速度较慢,只适合构建中小型的爬虫程序。对于异步加载的页面,Selenium 库可以让爬虫构建者以普通人的思维获取新的内容,而不需要去了解甚至掌握异步加载的底层细节,因此 Selenium 库也适合不了解网页加载技术的人。

Selenium 库通常使用 XPath 来爬取网页中的特定元素。XPath(XML Path Language)是一种用来确定 XML 文档中某一确定位置的语言,由于 XML 语言与 HTML 语言的相似性,XPath 也可以定位 HTML 文档中元素的位置。可以在浏览器开发者工具中,右击对应元素的 HTML 代码来获得 XPath 路径。

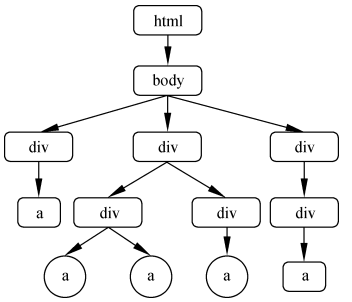

图 3-9 网页 HTML 文档的结构

XPath 路径以"/"开头,表示从根节点开始选取,然后加上节点名称以树状结构层层依次向下,直到选取到需要定位的元素。若只有节点名称,则会定位到当前对应该名称的所有子元素,如果加上[],则会定位到对应顺序的元素,序号从 1 开始。如果网页 HTML 文档结构如图 3-9 所示,则 XPath 为/html/body/div[2]/div/a 的元素是 3 个名字为 a 的圆形外框的节点。

3.4.2 Selenium 库与浏览器驱动安装

使用以下命令安装 Selenium 库。

```
pip install selenium
```

然后,需要下载并安装一个对应浏览器版本的 webdriver 驱动文件。可以在"设置-帮助"路径下查到浏览器的版本信息。之后访问 https://chromedriver.storage.googleapis.com/index.html 下载对应的驱动文件,按提示完成安装。最后需要向存放浏览器驱动的目录添加环境变量,以便在本地使用。如果添加环境变量失败,则需要在编写程序时加入驱动地址参数,代码如下。

```
path = r'驱动所在绝对路径\chromedriver.exe'
browser = webdriver.Chrome(path)
```

3.4.3 Selenium 库爬虫编写

本节将介绍 Selenium 库的一些主要操作方法。

打开浏览器,并保存驱动名的代码如下。

```
driver = webdriver.Firefox()
```

利用驱动打开 URL 链接对应网页的代码如下。

```
driver.get(url)
```

利用驱动执行对应的 JavaScript 代码(变量 js 表示)的代码如下。

```
driver.execute_script(js)
```

利用驱动定位相关元素的 4 种主要方法如表 3-1 所示。

表 3-1　Selenium 定位元素的主要方法

定位元素方法	意　义
find_element_by_id()	通过元素 id 定位
find_element_by_name()	通过元素名定位
find_element_by_xpath()	通过 XPath 路径表达式定位
find_element_by_tag_name()	通过标签名定位

定位元素后,可以利用 click()、drag_and_drop()等函数实现点击、拖动等操作。

下面将爬取网易新闻中关键词为“中国芯片”的相关新闻。搜索发现网址为 https://www.163.com/search?keyword=中国芯片,观察得知网页通过向下滑动滚动条加载新闻。可以通过 Selenium 库模拟浏览器滑动滚动条的操作,通过 XPath 定位相关新闻链接。具体代码和分析如下:

```
# 下载 Selenium 库后,从 Selenium 库中引入 webdriver
from selenium import webdriver
import time


# 此处下载的是 Firefox 驱动,所以用 Firefox()方法打开浏览器
# 若下载的是 Chrome 驱动,则利用 Chrome()方法打开浏览器
driver = webdriver.Firefox()

# 将提取的新闻链接保存在 listhref 列表中
listhref = []
url = "https://www.163.com/search?keyword=中国芯片"

# 通过分析网页结构可知,网页的所有新闻都存放在"class" = "keyword_list"的节点下,右击复
# 制该节点 XPath 路径"/html/body/div[2]/div[2]/div[1]/div[2]",再对某一个新闻进行分析
# 得到新闻链接存放的节点 a 的 XPath 路径,此时不用添加标号,就可以查询到所有满足条件的
# 新闻链接
xpath_name = "/html/body/div[2]/div[2]/div[1]/div[2]/div/h3/a"

# 根据网页链接打开浏览器
driver.get(url=url)
```

```
# 这里设计了两个临时变量,分别保存现在滚动条距离页面顶层的高度和上一次滚动条的高度,
# 用来判断滚动条是否已经到达页面底部,无法继续下滑
nowTop = 0
tempTop = -1

# 不断向下滑动滚动条并且保存新闻链接
while True:
  # 保存网页链接存取的位置节点
  name = driver.find_elements_by_xpath(xpath_name)
  # 遍历各个节点
  for j in range(len(name)):
  # 判断当前下标有没有文本
    if name[j].text:
      # 如果有,则添加进列表,通过 get_attribute()方法获得'href'属性的值,即新闻链接
      listhref.append(name[j].get_attribute('href'))
    else:
      pass

  # 执行滑动操作
  driver.execute_script("window.scrollBy(0,1000)")
  # 睡眠
  time.sleep(5)

  # 获得滚动条距离顶部的距离
  nowTop = driver.execute_script("return document.documentElement.scrollTop || window.
pageYOffset || document.body.scrollTop;")

  # 如果滚动条距离顶部的距离不再变化,意味着已经到达页面底部,可以退出循环
  if nowTop == tempTop:
    break
  tempTop = nowTop

# 完成后关闭浏览器
driver.close()
# 检查新闻链接是否保存成功
print(listhref)
```

获得新闻链接的列表后,可以通过 Requests、BeautifulSoup 等库的方法或者 XPath 的方法获取新闻标题、时间和内容,此处留作练习。

思考与练习

选择题

1. 关于 HTML 元素属性的描述中,错误的是(　　)。

A. id 属性提供了元素在全文档内的唯一标识

B. class 属性用于将类似元素分类

C. style 属性不可以将外观形式赋予一个特定元素

D. title 属性在大多数浏览器中显示为工具提示

2. 在一个完整的 URL 中，用于向服务器传递额外信息的可选部分是(　　)。

A. 域名　　B. 端口

C. 查询参数　　D. 片段标识符

3. 当服务器无法理解客户端发送的请求时，会返回(　　)状态码。

A. 400 Bad Request　　B. 401 Unauthorized

C. 404 Not Found　　D. 500 Internal Server Error

4. HTTP 的(　　)请求方法用于向服务器请求获取特定资源的内容。

A. HEAD　　B. POST　　C. PUT　　D. GET

5. 如果想通过元素的标签名来定位元素，应该使用 Selenium 库的(　　)方法。

A. find_element_by_class_name　　B. find_element_by_tag_name

C. find_element_by_css_selector　　D. find_element_by_name

判断题

1. 在 HTML 中，id 属性提供了元素的全局唯一标识。(　　)

2. GET 请求通常用于产生影响服务器状态的操作。(　　)

3. 当服务器不支持客户端发送的请求方法时，应当返回状态码 501。(　　)

4. 404 Not Found 状态码表示服务器成功处理了请求，但没有返回任何内容。(　　)

5. 在 BeautifulSoup 库中，find_all()方法返回所有匹配的元素。(　　)

简答题

1. JavaScript 与 HTML 的关系是什么？

2. Scrapy 框架的组件包括哪些？

3. 什么是网页的异步加载？

4. Selenium 库如何模拟点击一个按钮？

5. 使用 Python Requests 库发送 HTTP GET 请求的基本代码是什么？

章节实训：股票报告爬虫编写

实训目标

本章实训的目标网址为 https://data.eastmoney.com/report/stock.jshtml。爬取目标是该页面的个股研报表格，爬取结果可以用 TXT、CSV 等格式存储，至少包含“股票代码”“股票简称”“报告名称”3 个字段的数据。

实训思路

该网站使用异步加载的方式加载表格，所以提供以下两种思路。

1. 进入浏览器开发者模式分析网页加载过程，可以发现异步加载时请求数据的接口为 https://reportapi.eastmoney.com/report/list?cb=datatable3684874&industryCode=*&pageSize=50&industry=*&rating=&ratingChange=&beginTime={}&endTime=2022-08-07&pageNo={}&fields=&qType=0&orgCode=&code=*&rcode=&p=3&pageNum=

3&pageNumber=3&=1659858042675。其中，花括号部分为需要设置的参数，请读者探索这些参数的含义，并使用这个接口模板构建需要爬取的 url_list。得到 url_list 后，可以使用 Python 的 Requests 库遍历访问，并将结果分析提取到本地。也可以使用 Scrapy 框架来搭建爬虫，此时遍历访问的细节不需要手动编写。

2. 使用 Selenium 库来模拟人工点击翻页，每次翻页后提取页面中的目标内容。

爬取过程中需要注意控制请求发出的频率。