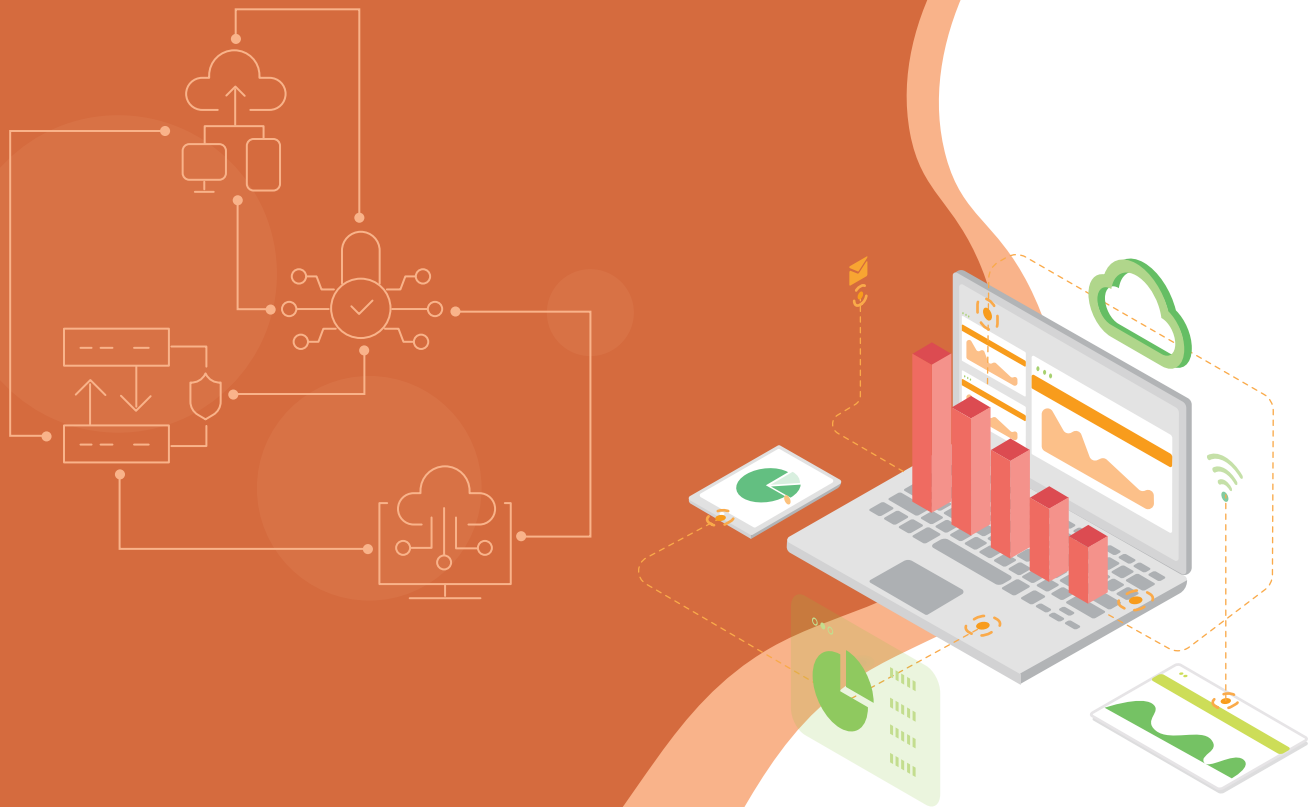
新时代计算机人才培养系列教材
“互联网+”新形态一体化教材

大数据采集与预处理技术

主编◎夏国清　洪　洲　陈　统



上海交通大学出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

新时代计算机人才培养系列教材

“互联网+”新形态一体化教材

大数据采集与预处理技术

主编◎夏国清　洪　洲　陈　统

扫一扫
学习资源库

上海交通大學出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

内容提要

本书按照“理论＋实战”的形式编写，将企业项目需求分解为单独的任务，全面系统地讲解了大数据采集与预处理的相关知识与技术。全书针对数据采集的不同来源，将知识内容分为五个项目，包括网络数据采集、分布式消息系统Kafka、实时数据库采集工具Canal和Maxwell、ETL日志采集技术栈以及ETL工具——Kettle。本书针对大数据采集与预处理的关键技术及其应用场景，从数据的采集、存储和分析等多个方面介绍了大数据的数据处理流程，通过任务实例为读者展示了如何有效地使用技术或工具。本书可作为大数据相关专业的教学用书，也可作为相关技术人员培训或工作的参考用书。

图书在版编目（CIP）数据

大数据采集与预处理技术 / 夏国清，洪洲，陈统主编 . —上海：上海交通大学出版社，2024.2

ISBN 978-7-313-30169-7

Ⅰ . ①大… Ⅱ . ①夏… ②洪… ③陈… Ⅲ . ①数据采集 ②数据处理 Ⅳ . ① TP274

中国国家版本馆CIP数据核字（2024）第035000号

大数据采集与预处理技术

DASHUJU CAIJI YU YUCHULI JISHU

主　　编：夏国清　洪　洲　陈　统
地　　址：上海市番禺路951号
出版发行：上海交通大学出版社
电　　话：021-6407 1208
邮政编码：200030
印　　制：北京荣玉印刷有限公司
经　　销：全国新华书店
开　　本：889 mm × 1194 mm　1/16
印　　张：16
字　　数：419千字
版　　次：2024年2月第1版
印　　次：2024年2月第1次印刷
书　　号：ISBN 978-7-313-30169-7
电子书号：ISBN 978-7-89424-521-2
定　　价：59.80元

前言

大数据是一种庞大的数据集，其规模、复杂性和产生速度使得传统数据处理工具难以应对。大数据涉及数据的收集、存储、处理、分析和可视化等多个方面，其应用范围涵盖了商业、科研、教育、医疗等诸多领域。大数据是应对当前信息时代挑战的重要学科，通过对大数据的学习，学生可以掌握数据处理和分析的技能，提升数据分析能力。同时，大数据学科对企业和组织来说，也有助于提升其决策的科学性和准确性，增强市场竞争力。大数据学科在教育改革、产业发展、科研创新等方面都具有重要的地位。

本书按照“理论＋实践”的方式，共设置了 5 个单独的项目，引导读者从易到难、循序渐进地学习大数据采集的相关知识及应用。教材针对不同的数据来源，介绍了网络爬虫、Kafka、Canal、ELK 和 Kettle 这 5 个关键技术的基本概念和应用场景，从数据的采集、存储和分析等方面全面介绍大数据采集的相关知识，贯穿了整个大数据的数据处理流程。下面是本教材各个项目的提要。

项目一讲述了网络爬虫的基本概念和工作原理，讲解了使用 Python 编写爬虫获取数据的常见方法以及应对反爬机制的策略。此外，项目一还讨论了编写爬虫时面临的道德和法律问题，引导读者树立遵守法律的意识。

项目二讲述了分布式消息系统 Kafka，讨论了 Kafka 的基本概念、应用场景、架构和核心组件，讲解了 Kafka 集群部署的基本操作，并利用 Kafka 构建一个可靠的实时数据流处理应用程序。

项目三主要使用 Canal 和 Maxwell 这两款开源的 MySQL 数据库增量数据订阅工具实现对关系型数据库的数据采集，讲解 Canal 的工作原理和架构，以及如何配置 Canal 的 TCP 模式和 Kafka 模式，并演示如何使用 Canal 和 MaxWell 来实现将数据库的数据变更实时发送到 Kafka。

项目四主要介绍了 ELK（Elasticsearch、Logstash 和 Kibana）这一组流行的日志管理和分析平台工具，讨论了每个组件的基本概念和功能，并演示了如何使用 ELK 收集、存储、分析和可视化日志数据，介绍了 ELK 的实时监控、事件分析和异常检测等高级功能。

项目五介绍了 Kettle 这一开源的 ETL（Extract-Transform-Load）工具，讲述了 Kettle 的基本概念和架构，并演示如何使用 Kettle 来实现数据抽取、数据转换和数据加载，讨论了 Kettle 的一些高级功能和应用场景，如数据仓库集成、数据质量管理和数据迁移。

本书的主要特色如下。

1. 融入二十大精神与思政元素

本书将二十大精神与思政元素融入项目之中，实现思想政治教育与知识体系教育的有机统一，让读者在学习大数据采集与预处理技术相关知识的同时，切身感受我国在大数据处理和应用方面的优势，养成良好的职业习惯。

2. 知识结构完善，重点突出

本书涵盖了大数据采集与预处理的核心知识，知识范围广，内容全面。本书在每个项目中利用思维导图梳理项目实施所需的知识点和技能点，便于读者快速理解该项目的学习内容，突出重点和难点。

3. 从易到难，逐步提升

针对大数据采集与预处理技术过程比较繁杂、不易于记忆的特点，本书内容编排上由浅入深，在每个项目中设置了任务实践，并在课后设置了巩固与提高，有助于学生循序渐进地学习相关技能。

4. 融入“1+X”职业技能标准

本书将数据采集职业技能等级证书（中级）中的职业技能等级标准与专业教学标准进行融合，在一定程度上实现了“书证融合”，以此提高读者的职业技能和职业素养。

5. 与实际岗位接轨

本书在编写时注重将“讲、学、练、做”融为一体。读者可以通过每个项目的学习了解相关技术的基础知识，通过任务实践提高分析和解决实际问题的能力，养成良好的项目开发习惯。通过学习本书，读者可以掌握网络爬虫、Kafka、Canal、ELK 和 Kettle 这 5 个关键技术的基本原理和应用方法，能够使用网络爬虫收集数据、使用 Kafka 构建大数据采集与预处理技术实时数据流处理应用、使用 Canal 实现数据同步和增量更新、使用 ELK 管理和分析日志数据、使用 Kettle 实现数据的抽取转换和加载。

本书还提供一些实用的案例，帮助读者更好地应用这些技术解决实际问题。同时本书配有丰富的数字化资源，包含了微课视频、电子课件（PPT）、案例源代码、试题库等，有需要者可致电 13810412048 或发邮件至 2393867076@qq.com 领取。

本书由广东职业技术学院夏国清、洪洲和广东轩辕网络科技有限公司陈统主编，熊勇、陈瑶、祺捷鹏、李桂凤、王士先担任副主编。全书由夏国清统稿。在此，一并向为本书编写做出贡献的老师表示衷心的感谢！

由于编者水平有限，书中可能存在疏漏和不妥之处，敬请读者批评指正。

夏国清

2023 年 9 月

目录

项目五 ETL 工具——Kettle / 186

项目一 网络数据采集

项目导航

知识目标

1. 了解爬虫的概念和基本原理。
2. 掌握在 Python 中实现爬虫请求的 urllib 库和 requests 库。
3. 掌握常见的数据解析的方法（正则表达式、BeautifulSoup、XPath、PyQuery）。
4. 掌握爬取动态渲染网页数据的知识。
5. 了解与网络爬虫有关的法律法规。

技能目标

1. 能叙述爬虫的原理，具有通过 urllib 或 requests 库实现请求并获得 HTML 源码的能力。
2. 能利用正则表达式、BeautifulSoup、XPath、PyQuery 进行网页数据的解析和提取。
3. 能开发基于 Selenium 的爬虫程序爬取动态渲染网页数据。
4. 能开发基于 Scrapy 的爬虫应用程序。

素养目标

1. 能够从多个角度分析和解决问题，培养解决实际问题的能力。
2. 树立法律意识，养成遵纪守法的习惯，增强道德意识和风险意识。

项目描述

互联网上有着千千万万的站点，公开的资源也极其丰富，很多时候都可以从互联网上直接搜寻到我们所需要的信息。然而通过人工采集的方式获取数据，其效率十分低下。此时可以通过网络爬虫程序去自动地进行数据分析和数据提取，这样可以大大提高数据采集的效率并减轻人工的负担。

在本项目中我们将通过四个任务来学习爬虫的相关内容。

任务一　认识网络爬虫

案例导入

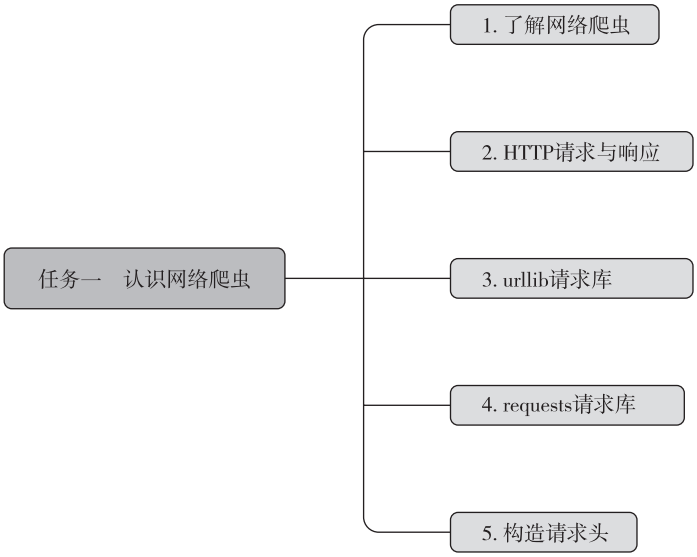
以打开数字中国建设峰会（以下简称数字中国）官网为例，在地址栏中输入相应网址，其网站首页页面如图 1-1-1 所示。那么如何能用程序来模拟用户的这个访问过程呢？

图 1-1-1　第六届数字中国建设峰会官网首页

思考：用户按下回车键后，浏览器显示了网站的首页。那么，从用户按下回车键到浏览器加载出页面这段时间发生了什么？对于其中的页面数据，我们又该如何获取呢？

任务导航

在本任务中，我们将完成一个爬虫程序，用来向数字中国官网发送请求以获得其首页的 HTML 源码，最后将源码保存到文件中。下面让我们根据知识框架一起开始学习吧！



一、了解网络爬虫

什么是网络爬虫？它有什么作用？它的功能是怎样实现的？可能初学者会有这样一系列的疑问，下面就来一一解答这些疑问。

（一）什么是网络爬虫

网络爬虫（又称网页蜘蛛或网络机器人）是一种按照一定的规则，自动地抓取互联网上的信息的程序或脚本。它针对既定的抓取目标，有选择地访问网页及相关的链接，获取所需要的数据资源。由于网络爬虫系统能为搜索引擎系统提供数据来源，所以很多大型的网络搜索引擎系统都被称为基于 Web 数据采集的搜索引擎系统，包括 Google、百度等著名的搜索引擎都是通过爬虫获取信息的，由此可见网络爬虫的重要性。

网络爬虫可以沿着网页中的通道（指向其他网页的超链接）获取多个网页中的数据。一般来说，互联网中的网页不是独立存在的，多个网页通过超链接互相连接，形成一个类似于蛛网的网络，网络爬虫可以沿着这个网络爬取网页上的数据，整个网页上的数据对爬虫来说“触手可及”。本项目将会介绍利用 Python 实现网络爬虫。

（二）网络爬虫的应用

网络爬虫有非常广泛的应用。目前网络爬虫主要应用于对万维网数据的挖掘，典型的应用就是搜索引擎。除了搜索引擎之外，越来越多的网络爬虫也广泛应用于工作与生活中。

在大数据时代，数据的采集是一项重要的工作，如果单靠人力进行信息采集，不仅效率低下，过程繁琐，搜集数据的成本也较高。此时，如果使用网络爬虫对数据信息进行自动采集，则会大大提高数据采集的效率。网络爬虫的应用领域十分广泛，它可以应用于搜索引擎中对站点进行爬取收录，应

用于数据分析与挖掘中对数据进行采集，应用于金融分析中对金融数据进行采集。此外，还可以将网络爬虫应用于舆情监测与分析、目标客户数据收集等领域。

通过网络爬虫的学习，读者可以设计并实现一款小型的搜索引擎。当然，这个爬虫在性能或者算法上可能比不上主流的搜索引擎，但是其个性化的程度会非常高，并且也有利于读者更深层次地理解搜索引擎内部的工作原理。

（三）网络爬虫的基本流程

网络爬虫的基本流程如图 1-1-2 所示，各个流程的说明如下。

（1）首先选取一部分精心挑选的种子 URL。

（2）将这些种子 URL 放入待抓取 URL 队列。

（3）从待抓取 URL 队列中取出待抓取 URL，解析 DNS（domain name system，域名解析系统），得到主机的网络协议地址（internet protocol address），将 URL 对应的网页下载下来，存储进已下载的网页源码库中。然后，将这些 URL 放进已抓取 URL 队列。

（4）分析已抓取 URL 队列中的 URL，分析其中是否包含未抓取的 URL，并将未抓取的 URL 放入待抓取 URL 队列，从而进入下一个循环。

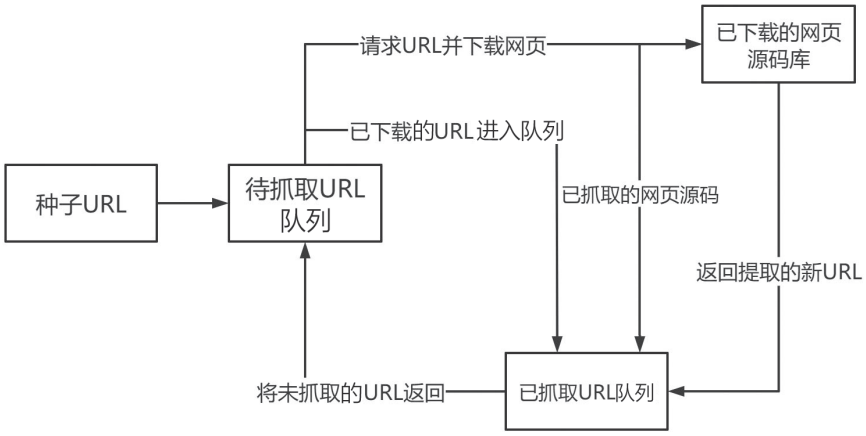


图 1-1-2　网络爬虫的基本流程

（四）什么是 HTTP 请求与响应

以打开数字中国网站为例，在输入网址、按下回车键这个过程中，到底发生了什么呢？

其实这个过程涉及 DNS 解析、TCP 连接的建立与关闭、HTTP 请求与响应等多个阶段。其中最重要的一个过程就是 HTTP 请求与响应，我们可以将浏览器看作客户端（client），网站的服务器就是服务端（server），它们通过发送 HTTP 请求来通信。HTTP 通信过程如图 1-1-3 所示。

图 1-1-3　HTTP 通信过程

（五）什么是 HTTP 消息

通常超文本传输协议（hyper text transfer protocol，HTTP）的消息简称 HTTP 消息，包括客户机向服务器的请求消息和服务器向客户机的响应消息。

这两种类型的消息由起始行、头域及代表头域结束的空行和可选的消息体组成。HTTP 的头域包括通用头、请求头、响应头和实体头 4 个部分。每个头域由域名、冒号（:）和域值 3 个部分组成。域名不区分大小写，域值前可以添加任何数量的空格符，头域可以被扩展为多行，在每行开始处，使用至少一个空格或制表符。

客户端首先会发送一个 HTTP 请求到服务器。HTTP 请求是由 4 个部分组成的，分别是请求行、请求头、空行、请求体（数据）。HTTP 请求消息的一般格式如图 1-1-4 所示。

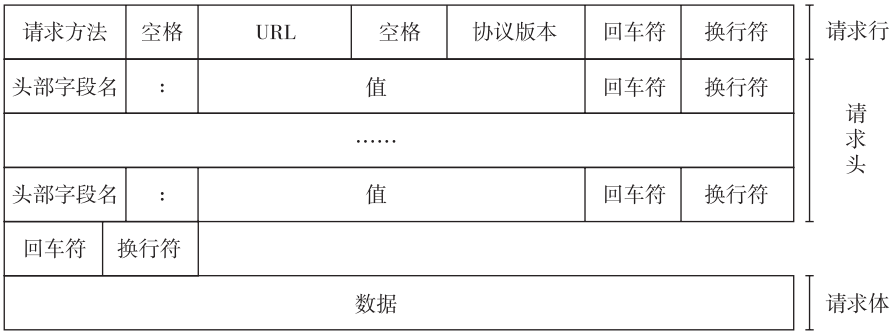


图 1-1-4　HTTP 请求消息的一般格式

服务器接收到 HTTP 请求之后会做出响应。HTTP 响应也是由 4 个部分组成的，分别是状态行、响应头、空行以及响应体。HTTP 响应消息的一般格式如图 1-1-5 所示。

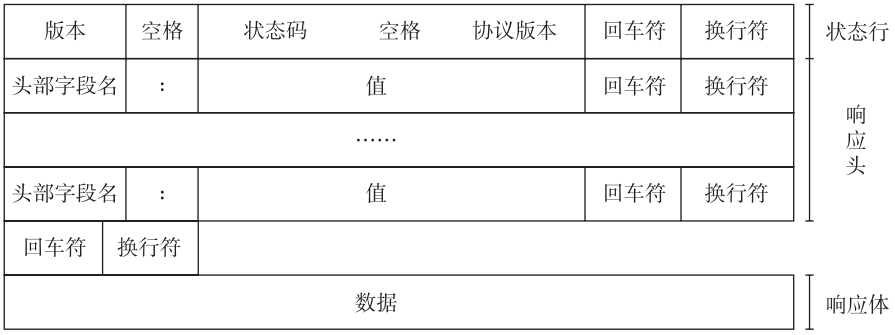


图 1-1-5　HTTP 响应消息的一般格式

在客户端和服务端的一个来回通信中还包含了一些其他的术语，相关的介绍如下。

（1）Request URL：表示请求的 URL。

（2）Request Method：表示请求的方法，常见的有 GET 和 POST。除此之外，HTTP 的请求方法还有 OPTION、HEAD、DELETE、PUT 等。

① GET：表示向指定的资源发出“显示”请求。

② POST：表示向指定资源提交数据，请求服务器进行处理（如提交表单或上传文件），数据包含在请求体中。这个请求可能会创建新的资源、修改现有资源，或二者皆有。

（3）Status Code：显示 HTTP 请求和状态码，表示 HTTP 请求的状态。此处显示 200，表示请求

已被服务器接收、理解和处理。状态码的第一个数字代表当前响应的类型。HTTP 协议中有以下几种响应类型。

① 消息：请求已被服务器接收，继续处理。

② 成功：请求已成功被服务器接收、理解并接受。

③ 重定向：需要后续操作才能完成这一请求。

④ 请求错误：请求含有语法错误或者无法被执行。

⑤ 服务器错误：服务器在处理某个正确请求时发生错误。

（4）HTTP 请求头包括以下内容。

① Cookie：为了辨别用户身份、进行 session 跟踪而储存在用户本地的数据。

② User-Agent：表示浏览器标识。

③ Accept-Language：表示浏览器所支持的语言类型。

④ Accept-Charset：告诉 Web 服务器浏览器可以接受哪些字符编码。

⑤ Accept ：表示浏览器支持的多用途互联网邮件扩展（multipurpose internet mail extensions，MIME）类型。

⑥ Accept-Encoding：表示浏览器有能力解码的编码类型。

⑦ Connection：表示客户端与服务器连接的类型。

在网络数据采集业务中，读取 URL、下载网页是爬虫必备的功能，需要和 HTTP 消息打交道。

二、 实现爬虫的请求

了解爬虫的相关知识后，接下来就是在程序中去实现它。有很多语言都可以实现爬虫程序，其中以 Python 语言最为方便，Python 提供了丰富的官方包和第三方开发的爬虫库，是我们做爬虫程序开发的利器。下面介绍 urllib 和 requests 两个请求库。

（一）用 urllib 实现 HTTP 请求

urllib 是 Python 内置的实现网络请求的模块，可以快速实现 HTTP 请求。在爬取网页数据时，用户只需关注请求的 URL 格式、请求的参数和请求头类型，而不需要关心其底层是怎样实现的。

先来实现一个完整的请求与响应模型。urllib 提供了一个基础函数 urlopen，通过向 URL 发出请求来获取数据。使用 urlopen 函数时，首先需要导入 urllib 中的 request 模块，这是最基本的 HTTP 请求模块，可以模拟请求的发送。

例 1-1-1： 实现一个完整的请求与响应模型，示例代码如下所示。

```
import urllib.request
res = urllib.request.urlopen('http://www.tup.tsinghua.edu.cn')
print(res.read().decode('utf-8'))
```

上述代码是一个请求网页的案例，运行代码，部分输出结果如下（篇幅限制，只截取一部分）。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
    <title></title>
</head>
<script type="text/javascript">
    //平台、设备和操作系统
    var system = {
        win: false,
        mac: false,
        xll: false
    };
</script>
<body>

</body>
</html>
```

实际上，如果我们打开某网站的首页（这里使用的网址为 www.tup.tsinghua.edu.cn），在页面上右击鼠标并选择“检查”快捷菜单项，浏览器会弹出浏览器的开发者面板（以谷歌浏览器为例，其他浏览器类似），在开发者面板的“元素”选项卡中就会显示网页的源代码，和刚才输出的内容一模一样。也就是说，例 1-1-1 仅仅用了几行代码，就能把网站首页的代码都下载下来。

其实可以将上面对网站的请求响应分为两步，第一步是请求，第二步是响应。

例 1-1-2： 将对网站的请求响应分成两步，示例代码如下所示。

```
import urllib.request
#请求
req=urllib.request.Request('http://www.tup.tsinghua.edu.cn')
#响应
res = urllib.request.urlopen(req)
html=res.read().decode('utf-8')
print(html)
```

当我们把请求响应步骤分为两步后，就需要构建一个 Request 对象来作为 urlopen 方法的参数。与此同时，我们可以在 Request 对象中传入更多的内容，如添加 Cookie 信息、设置要接收的类型。

例 1-1-1 和例 1-1-2 发送的都是 GET 请求，接下来讲解 POST 请求。使用 urllib 发送 POST 请求和发送 GET 请求类似，只是增加了请求数据。

例 1-1-3： 发送 POST 请求，示例代码如下所示。

```
import urllib.request
url = 'http://www.tup.tsinghua.edu.cn/member/dl.aspx'
fields = {
```

```
        'username':'qiye',
        'password':'qiye_pass'
    }
    # 将数据进行编码
    postdata = urllib.parse.urlencode(fields).encode('utf-8')
    # 构建 request 请求
    req = urllib.request.Request(url,postdata)
    # 发送请求
    res = urllib.request.urlopen(req)
    print(res.read().decode('utf-8'))
```

在进行注册、登录等操作时，表单信息会通过 POST 传递。我们需要分析页面结构，构建表单数据 fields，使用 urlencode() 进行编码处理。经过编码处理后会返回字符串，此时指定 UTF-8 为编码格式得到 postdata。POST 发送的数据必须是字节类型或文件对象。最后再通过 Request() 传递 postdata，使用 urlopen() 方法发送请求。有时候即便 POST 请求的数据是正确的，服务器也会拒绝访问。问题出在请求头信息中，服务器会检查请求头，根据请求头判断是否是来自浏览器的访问，这也是反爬的常用手段。

接下来对请求头 Headers 进行处理，将上面请求网页的案例改写一下，加上请求头信息，设置请求头中的 User-Agent 域和 Referer 域信息。

例 1-1-4： 在构造请求时设置请求头，示例代码如下所示。

```
    # 请求头 Headers 处理：设置请求头中的 User-Agent
    import urllib.request
    url = 'http://www.tup.tsinghua.edu.cn/member/dl.aspx'
    data = {
        'username' : 'qiye',
        'password' : 'qiye_pass'
    }
    headers = {
        'User-Agent':'Mozilla/5.0 (WindowsNT10.0; Win64; x64) AppleWebKit/537.36 (KHTML, likeGecko)
Chrome/60.0.3112.113 Safari/537.36'
    }
    res = urllib.request.urlopen('post',url+"/post",data=data,headers=headers)
    print(res.read().decode())
```

如果我们对网页的访问过于频繁，那么网站服务器可能会屏蔽对应 IP 地址，此时我们可以使用 IP 代理。首先，通过互联网上公开的 IP 代理网站找到一个可用的 IP 地址，然后通过 IP 地址构建 ProxyHandler 对象，将 HTTP 和代理 IP 地址以字典形式作为参数传入，设置代理服务器的信息，构建 opener 对象，将 ProxyHandler 对象传入，再使用 opener 中的 open() 方法发送 HTTP 请求。

例 1-1-5：使用代理实现请求，示例代码如下所示。

```
# 使用 IP 代理
import urllib.request
# 创建 Handler 对象，添加代理 IP
httpproxy_hander = urllib.request.ProxyHandler({"http": "183.221.241.103:9443"})
# 创建 opener 对象
opener = urllib.request.build_opener(httpproxy_hander)
# 创建 request 对象
request = urllib.request.Request('http://www.tup.tsinghua.edu.cn')
# 发送请求
res = opener.open(request)
print(res.read())
```

假设需要爬取几百个网站，在抓取时部分网站可能无法响应或者需要等待几十秒才能返回数据，那么程序运行将需要长时间的等待，这显然是不可行的。此时可以设置一个请求超时时间，一旦超过这个时间并且服务器没有返回响应内容，程序就会抛出一个超时异常，可以用 try 语句来捕获该异常。在例 1-1-5 的基础上添加超时设置，如例 1-1-6 所示。

例 1-1-6：在构造的请求中设置超时时间参数，示例代码如下所示。

```
#IP 代理添加超时参数
import urllib.request
# 创建 Handler 对象，添加代理 IP
httpproxy_hander = urllib.request.ProxyHandler({"http": "183.221.241.103:9443"})
# 创建 opener 对象
opener = urllib.request.build_opener(httpproxy_hander)
# 创建 request 对象
request = urllib.request.Request('http://www.tup.tsinghua.edu.cn')
# 发送请求
res = opener.open(request,timeout=2)
print(res.read())
```

在此基础上，发送 HTTP 请求时可能会抛出异常，需要添加异常处理语句捕获异常。网络请求常见的异常主要有 URLError、HttpError 两种。

例 1-1-7：以 URLError 为例，添加异常处理，示例代码如下所示。

```
#IP 代理添加超时参数
import urllib.request
# 创建 Handler 对象，添加代理 IP
httpproxy_hander = urllib.request.ProxyHandler({"http": "183.221.241.103:9443"})
# 创建 opener 对象
opener = urllib.request.build_opener(httpproxy_hander)
```

```
# 创建 request 对象，设置一个不存在的 url
req = urllib.request.Request('http://www.abcdssg.com')
# 发送请求，添加 try 捕获异常
try:
    res = opener.open(req)
except urllib.error.URLError as err:
    print(err)
print(res.read())
```

运行代码后，输出的结果如下。

```
<urlopen error [WinError 10060] 由于连接方在一段时间后没有正确答复或连接的主机没有反应，连接尝试失败。>
```

此次请求失败的原因为没有找到指定的服务器。

（二）用 requests 实现 HTTP 请求

requests 是用 Python 语言编写的基于 urllib 的第三方 HTTP 请求库，采用的是 Apache2 Licensed 开源协议。requests 比 urllib 更加方便，可以节约大量的工作。使用 requests 实现网络爬虫的步骤：使用 requests 库发起网络请求获取 HTML（hype text markup languge，超文本标记语言）文件；利用正则表达式等字符串解析手段或者 Beautiful Soup 库（第三方库）对获取到的 HTML 文件进行解析，实现信息提取。

1. 安装 requests 库

requests 需要先安装再使用。requests 的安装方式一般有两种。

（1）使用 pip 安装，安装的命令如下所示。

```
pip install requests
```

（2）在 Github 网站上下载 requests 的源代码，将源代码压缩包进行解压缩，然后进入解压缩的文件夹，运行 setup.py 文件即可完成安装。

2. 使用 requests 构造请求

（1）构造基本的 GET 请求。

例 1-1-8： 实现一个完整的 GET 请求与响应模型，示例代码如下所示。

```
import requests
r = requests.get('http://www.szzg.gov.cn/')
print(r.content)
```

通过例 1-1-8 可以看到，requests 发起请求的实现方式比 urllib 简洁得多。

（2）构造 POST 请求。requests 实现 POST 请求同样非常简单，且更加具有 Python 风格。

例 1-1-9： 实现一个完整的 POST 请求与响应模型，示例代码如下所示。

```
import requests
postdata={'key':'value'}
r = requests.post('http://httpbin.org/post',data=postdata)
print(r.content)
```

（3）在 GET 请求中传递参数。

在日常生活中，我们会经常看到类似“ http://zzk.cnXX XX.com/s/blogpost?Keywords=blog:qiyeboy &pageindex=1”的 URL 链接，这种链接在网址后面会紧跟着一些特殊的字符串，而这些字符串其实是请求携带的参数。那么携带参数的 GET 请求该如何发送呢？一般来说，通过直接完整的 URL 就可以发起请求，不过 requests 还提供了通过传递参数发送 GET 请求的方式。

例 1-1-10： 在 GET 请求中传递参数，示例代码如下所示。

```
import requests
payload = {'Keywords': 'blog:qiyeboy','pageindex':1}
r = requests.get('http://zzk.cnblogs.com/s/blogpost?', params=payload)
print(r.url)
```

通过打印结果可以看到最终的 URL 变成了如下所示的 URL 地址。

```
http://zzk.cnblogs.com/s/blogpost?Keywords=blog:qiyeboy&pageindex=1
```

例 1-1-11： 实现一个完整的 HTTP 请求模型，示例代码如下所示。

```
import requests
url='http://zzk.cnblogs.com/'
data=requests.get(url)
print(data.status_code)
print(data.content)
```

借助 requests 库通过几行代码就完成了一个简单的对某网站的 HTTP 请求。例 1-1-11 中的第一个 print 语句输出了这个请求的状态码，结果返回的是 200，表示访问正常。例 1-1-11 中的最后一个 print 语句是输出响应的二进制内容。执行代码，得到的响应内容（只截取了部分）如图 1-1-6 所示。

```
200
b'<!DOCTYPE html>\n<html lang="zh-cn">\n<head>\n    <meta charset="utf-8" />\n    <
meta name="viewport" content="width=device-width, initial-scale=1" />\n    <meta na
me="referrer" content="always" />\n    <meta http-equiv="X-UA-Compatible" content
="IE=edge" />\n    <title>\xe5\x8d\x9a\xe5\xae\xa2\xe5\x9b\xad - \xe5\xbc\x80\xe5\x
8f\x91\xe8\x80\x85\xe7\x9a\x84\xe7\xbd\x91\xe4\xb8\x8a\xe5\xae\xb6\xe5\x9b\xad</tit
le>\n        <meta name="keywords" content="\xe5\xbc\x80\xe5\x8f\x91\xe8\x80\x85, \x
e7\xa8\x8b\xe5\xba\x8f\xe5\x91\x98, \xe5\x8d\x9a\xe5\xae\xa2\xe5\x9b\xad, \xe7\xa8\x8
b\xe5\xba\x8f\xe7\x8c\xbf, \xe7\xa8\x8b\xe5\xba\x8f\xe5\xaa\x9b, \xe6\x9e\x81\xe5\xae
\xa2, \xe7\xa0\x81\xe5\x86\x9c, \xe7\xbc\x96\xe7\xa8\x8b, \xe4\xbb\xa3\xe7\xa0\x81, \xe
8\xbd\xaf\xe4\xbb\xb6\xe5\xbc\x80\xe5\x8f\x91, \xe5\xbc\x80\xe6\xba\x90, IT\xe7\xbd\x
91\xe7\xab\x99, \xe6\x8a\x80\xe6\x9c\xaf\xe7\xa4\xbe\xe5\x8c\xba, Developer, Programme
r, Coder, Geek, Coding, Code" />\n        <meta name="description" content="\xe5\x8d\x9
```

图 1-1-6　响应内容

可以看到“print(data.content)”的输出是以一个字母 b 开头的，表示输出的是二进制数据，而网页中的中文字符在二进制编码下就会显示成乱码。如果要让中文字符正常显示，一般用“data.text”命令，有时还需要设置编码。

（三）构造请求头

很多网站能接收用户的正常访问请求，但却拒绝由爬虫程序发起的请求，这是因为服务器后端对请求头做了限制。如果一个爬虫频繁地发送请求，就会占用站点服务器的资源，影响用户的正常访问。

现在不少网站都采用各种反爬机制来识别爬虫，常用的方式包括：通过甄别访问请求的发起者的类型从而做出不同的响应；通过监测同一 IP 地址的访问频率从而过滤非法请求；通过验证码过滤请求。反爬的保护策略比较多，也在不断发展之中。在学习爬虫的过程中，我们应该了解与网络爬虫有关的法律法规，树立遵守法律的意识，以正确合法的方式使用爬虫。

既然许多站点采用了反爬技术，那么爬虫是否就无法工作了呢？答案是否定的，我们还可以通过一些手段绕开限制。这里只对绕开甄别访问请求的反爬机制做一下介绍。

用 urllib 或 requests 编程实现向服务器发送请求，服务器接收后会进行一些判断，查看此次请求是用户的访问还是程序发起的请求，那么服务器是通过什么信息判断呢？

用浏览器（Edge 或 Chrome 都可以）打开数字中国的首页，进入到开发者面板（按“Ctrl+Shift+I”组合键），单击“网络”选项卡，刷新一下网页，可以看到用户正常访问时的请求头信息，如图 1-1-7 所示。

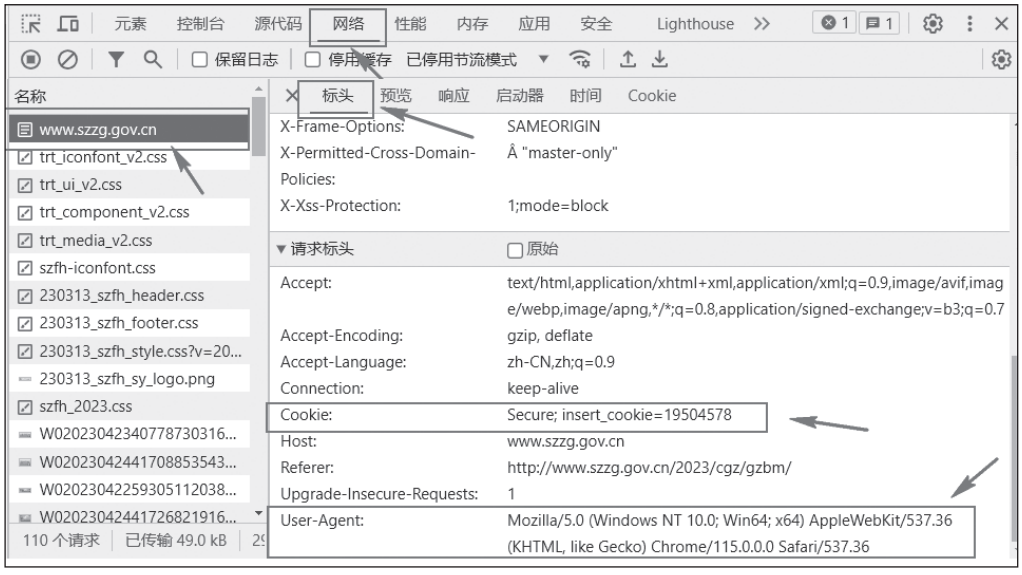


图 1-1-7　用户正常访问时的请求头信息

在“名称”列单击第 1 个文件“www.szzg.gov.cn”（它是一个 document 类型，是最初请求的文件），打开该文件的服务器响应信息列表，查看“标头”选项卡下的内容，它展示了请求头（headers）信息，图 1-1-7 中加了方框的两处分别是“Cookie”和“User-Agent”。这两处信息很重要，可以让服务器判别正在访问的用户的身份。Cookie 是保存在客户机上的用户信息，在一次访问的会话过程中通过 Cookie 可以免于重复验证用户身份，而 User-Agent 的作用主要是告诉服务器用户的浏览器类型，让服务器根据不同类型的浏览器返回不同的响应结果。

前面在使用 urllib 或 requests 发送请求时没有设置这些内容，所以服务器能很方便地判别出这些请求不是正常用户的访问请求，如果站点采用了反爬机制，就不会给出正常的响应，我们就得不到需要的数据了。

那么如何解决这个问题呢？换句话说，如何让我们的程序发出的请求可以模仿用户的正常访问呢？答案是复制浏览器发送的请求的 Cookie 和 User-Agent 信息，将这些信息构造成一个字典类型的数据，添加到由程序构造的请求中。此外，还可以使用 fake-useragent 库获取随机的 User-Agent 信息。

下面以猫眼电影栏目榜单为例，对比添加 Cookie 和 User-Agent 请求头前后发起请求返回数据的变化。

例 1-1-12： 直接请求猫眼电影榜单栏目，示例代码如下所示。

```
import requests
myurl = "https://www.maoyan.com/board"
res = requests.get(myurl)
res.encoding = "utf-8"
print(res.text)
```

运行该代码可以发现，从输出的网页源码中找不到该页面上的影片数据。这说明不添加 Cookie 和 User-Agent 直接请求是得不到影片数据的。

接下来为该段代码添加请求头。用 Chrome 浏览器打开猫眼电影榜单的网页，然后切换到开发者面板，找到“名称”列中的文件“ board”，单击该文件，打开响应信息清单，分别复制 Cookie 和 User-Agent 两项的值。

例 1-1-13： 在请求中构造请求头信息，示例代码如下所示。

```
import requests
myurl = "https://www.maoyan.com/board"
# 构造目标网站的请求头，请求头是一个字典类型的数据
myheaders = {
    'User-Agent' : 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/113.0.0.0 Safari/537.36',                    # 注意，这里有个英文逗号
    'Cookie' : 'uuid_n_v=v1; uuid=.....-f4c-c81-25||8'
# 注：Cookie 内容太长，本书省略了中间部分
}
res = requests.get(url=myurl, headers=myheaders)
# 在 get() 函数中通过 headers 关键字参数应用请求头
res.encoding = "utf-8"
print(res.text)
```

运行例 1-1-13 中的代码后，可以从输出中找到电影的数据。通过构造请求头数据就可以将程序发出的请求伪装成用户正常访问的请求，从而突破服务器对爬虫的限制。

当然，用构造请求头突破网站反爬限制的方法只对部分网站有用。在后面的项目中还会介绍基于 Selenium 的爬虫，它适合更多的网站。

三、任务实践

学习了前面的知识后，下面分别使用 urllib 库和 requests 库来实现任务导航中的要求，即获得数字中国首页的 HTML 源码并将源码保存到文件中。

（1）使用 urllib 库实现请求，示例代码如下所示。

```
import urllib.request

headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.0.0 Safari/537.36",
    "Cookie": "Secure; insert_cookie=77731460"
}
url = 'http://www.szzg.gov.cn/'
request = urllib.request.Request(url=url, headers=headers)      # 构造请求对象
response = urllib.request.urlopen(request)                      # 发送请求获得响应
html = response.read().decode('utf-8')                          # 解码获得源码
print(html)
with open('szzg_index.txt', 'w', encoding='utf-8') as file:     # 写入文本文件
    file.write(html)
```

（2）用 requests 库实现请求，示例代码如下所示。

```
import requests

headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.0.0 Safari/537.36",
    "Cookie": "Secure; insert_cookie=77731460"
}
url = 'http://www.szzg.gov.cn/'
response = requests.get(url=url, headers=headers)               # GET 请求获得响应
response.encoding = 'utf-8'                                     # 设置响应的编码为 utf-8
html = response.text                                            # 解码获得源码
print(html)
with open('szzg_index2.txt', 'w', encoding='utf-8') as file:    # 写入文本文件
    file.write(html)
```

分别运行两段代码，可以看到它们的输出结果是一样的，在代码的 Python 文件所在目录下分别生成了 szzg_index1.txt 和 szzg_index2.txt 两个文件，文件内容如图 1-1-8 所示。

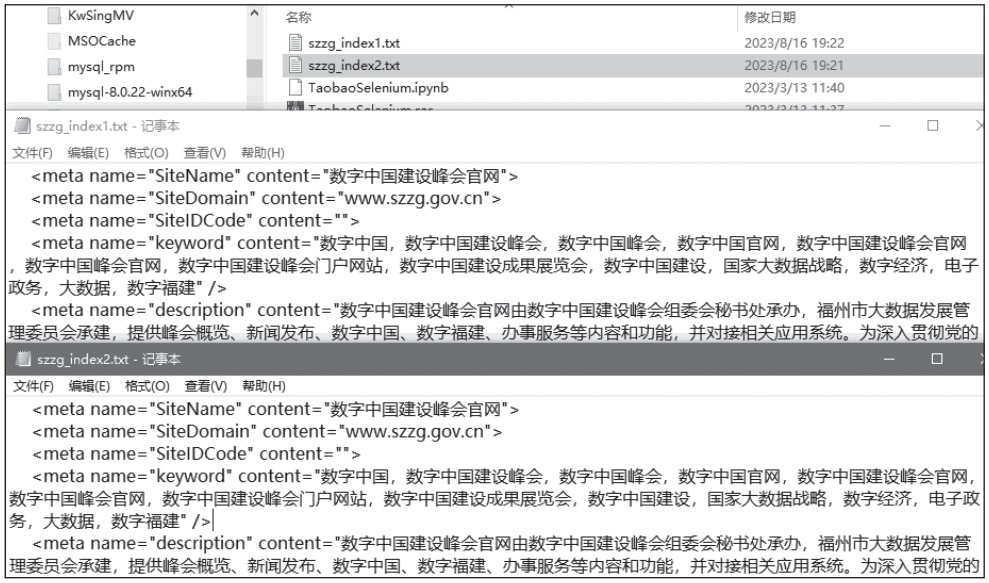


图 1-1-8　szzg_index1.txt 和 szzg_index2.txt 的文件内容

获取网页源码

巩/固/与/提/高

请使用 requests 库向 http://www.gdpt.edu.cn/ 发送请求，获得首页的源码并打印出来。

在线测试 1

任务二　解析数据

案例导入

完成任务一的学习后，我们可以通过爬虫程序向服务器发出请求，从而获得服务器的响应对象，其中包含了整个网页的 HTML 源码。但是获取的源码数据比较庞大、杂乱，并且其中大部分的数据并不是人们所关心的。针对这种情况，需要对爬取的数据进行过滤筛选，去掉没用的数据，留下有价值的数据。下面以名人引言网站（http://quotes.toscrape.com/）为例，学习从网页源码中解析出有价值的引言数据。名人引言网站首页如图 1-2-1 所示。

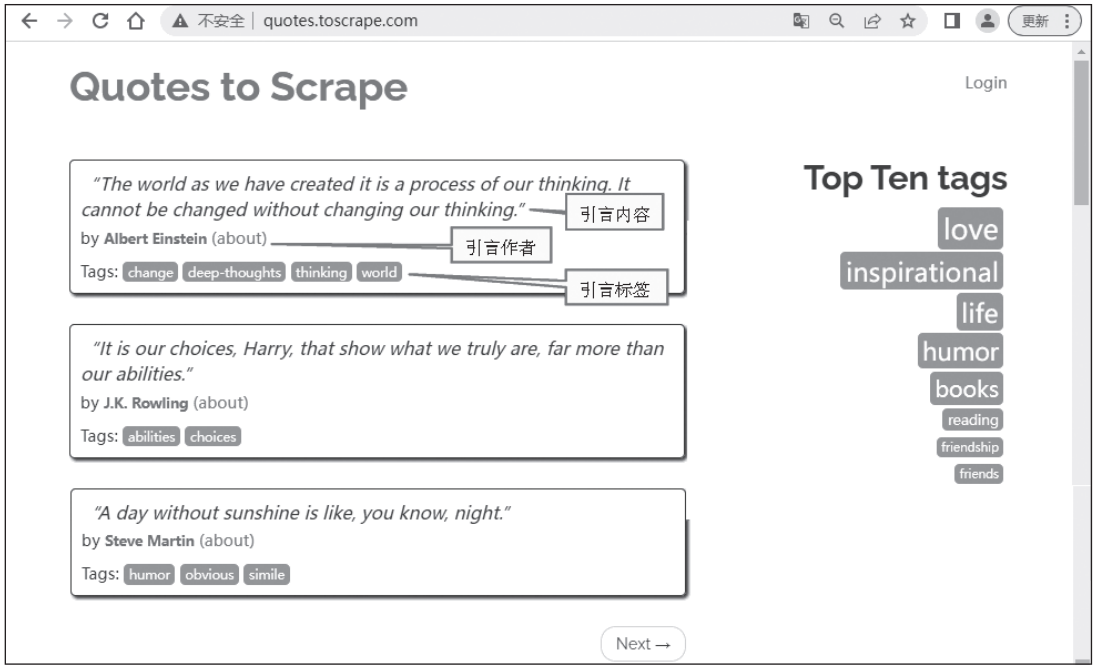


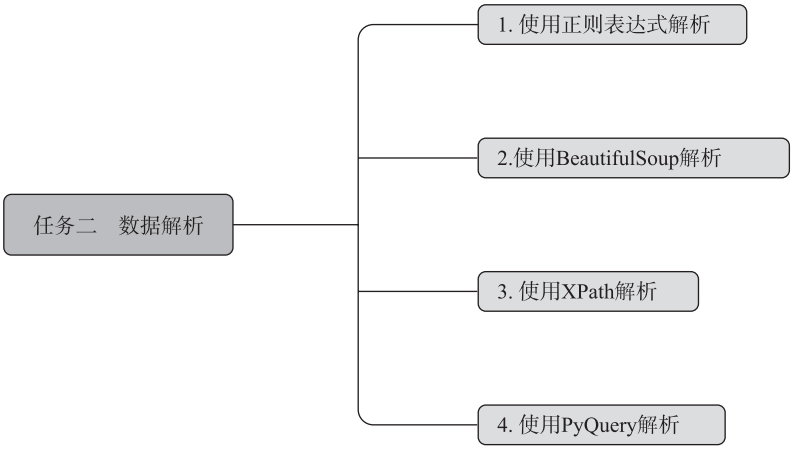
图 1-2-1　名人引言网站首页

思考： 你知道几种数据解析方式？

任务导航

在本任务中，我们将学习解析 HTML 网页中的数据。数据解析是 Python 网络数据爬虫开发中非常关键的步骤。HTML 网页的数据解析方式有很多种，本任务将从使用正则表达式解析、使用 BeautifulSoup 解析、使用 XPath 解析和使用 PyQuery 解析 4 个方面进行介绍。

图 1-2-1 所示的是一个展示名人引言的站点，它共有 10 个页面，每个页面展示 10 条引言。每条引言数据由引言内容、引言作者、引言标签 3 项信息构成。在网站页面底部有一个 Next 翻页按钮可以跳转到下一页。学会用数据解析任意一种方式将这 10 页共 100 条名人引言的数据爬取下来，并且保存到 CSV 文件 quotes.csv 中。下面让我们根据知识框架一起开始学习吧！



一、使用正则表达式解析

简单地说，正则表达式是一种可以用于模式匹配和替换的强大工具。在几乎所有的基于 Unix/Linux 系统的软件工具中都能找到正则表达式的痕迹，如 Perl 或 PHP 脚本语言。此外，JavaScript 这种脚本语言也提供了对正则表达式的支持。正则表达式已经成为一个通用的工具，被各类技术人员广泛使用。

（一）语法与使用

正则表达式是一个很强大的字符串处理工具，几乎任何关于字符串的操作都可以使用正则表达式来完成。对于数据采集工作者来说，需要每天和字符串打交道，正则表达式更是不可或缺的技能。正则表达式在不同开发语言中的使用方式可能不一样，但只要学会了任意一门语言的正则表达式用法，就能在其他语言中快速掌握，它们在本质上都是一样的。

首先，Python 中的正则表达式用法大致分为元字符、模式、函数、re 内置对象、分组和环视。

其次，所有关于正则表达式的操作都使用 Python 标准库中的 re 模块，部分正则表达式的匹配规则如表 1-2-1 所示。

表 1-2-1　部分正则表达式的匹配规则

模式	描述	模式	描述
.	匹配任意字符（不包括换行符）	\A	匹配字符串开始位置，忽略多行模式
^	匹配开始位置，多行模式下匹配每一行的开始	\Z	匹配字符串结束位置，忽略多行模式
$	匹配结束位置，多行模式下匹配每一行的结束	\b	匹配位于单词开始或结束位置的空字符串
*	匹配前一个元字符 0 次式 n 次	\B	匹配不在单词开始或结束位置的空字符串
+	匹配前一个元字符 1 次式 n 次	\d	匹配一个数字，相当于 [0—9]
?	匹配前一个元字符 0 到 1 次	\D	匹配非数字，相当于 [^0—9]

续表

模式	描述	模式	描述
{m,n}	匹配前一个元字符 m 到 n 次	\s	匹配任意空白字符，包括空格、制表符、换页符等，相当于 [\t\n\r\f\v]

（二）正则表达式函数

正则表达式是一个特殊的字符序列，它能帮助你方便地检查一个字符串是否与某种模式匹配。Python 自 1.5 版本之后就增加了 re 模块，提供了 Perl 风格的正则表达式模式。re 模块使 Python 语言拥有全部的正则表达式功能。使用 re.compile() 函数可以将一个正则表达式字符串和可选的标志参数编译成一个模式对象。该对象拥有一系列用于正则表达式匹配和替换的方法。

re 模块也提供了与这些方法功能完全一致的函数。这些函数使用一个模式对象作为它们的第一个参数。下面主要介绍 Python 中常用的正则表达式函数。

1. re.match 函数

re.match 函数按照从字符串的起始位置匹配的规则，该函数的语法如下所示。

```
re.match(pattern, string, flags=0)
```

re.match 函数参数说明如表 1-2-2 所示。

表 1-2-2　re.match 参数说明

参数	描述
pattern	匹配的正则表达式
string	要匹配的字符串
flags	标志位，用于控制正则表达式的匹配方式，如是否区分大小写、是否是多行匹配等

若匹配成功，则 re.match 函数返回一个匹配的对象，否则返回 None。可以使用 group(n) 或 groups() 函数来获取匹配的结果，示例代码如下所示。

```
import re
content = 'Hello 1234567 World_This tel 11012345457 is a Regex Demo'
pattern1 = re.compile('Hello\s(\d{7})\sWorld_This\stel\s(\d+)\sis')    # 编译得到模式
result1 = re.match(pattern1, content)                                  # 在 re.match() 中应用模式
print('数字：', result1.group(1))
print('电话号码：', result1.group(2))
print(result1.group())
```

运行后的输出结果如下所示。

```
数字：1234567
```

```
电话号码：18512345457
Hello 1234567 World_This tel 18512345457 is
```

可以看到，我们用模式中的“(\d{7})”成功匹配到了“1234567”，并用 group(l) 获取了它。后面用 group(2) 获取了模式中“(\d+)”匹配到的内容，即“18512345457”。group() 会输出完整的匹配结果，而 group(n) 会获得模式中第 n 个被“()”包围的匹配结果。

2. re.search 函数

re.search 函数用于扫描整个字符串，并返回第一个成功匹配的结果，该函数的语法如下所示。

```
re.search(pattern, string, flags=0)
```

re.search 函数参数说明如表 1-2-3 所示。

表 1-2-3　re.search 参数说明

参数	描述
pattern	匹配的正则表达式
string	要匹配的字符串
flags	标志位，用于控制正则表达式的匹配方式，如是否区分大小写、是否多行匹配等

若匹配成功，则 re.search 返回第一个成功匹配的对象，否则返回 None，示例代码如下所示。

```
import re
html = '''
<div id="institution">
<ul>
<li class="xz"><a href="jwb.html"> 教务部 </a></li>
<li id="finance" class="xz"><a href="cwb.html"> 财务部 </a></li>
</ul>
</div>
'''
pattern2 = re.compile('<a.*?href="(.*?)">(.*?)</a>', re.S)
result2 = re.search(pattern2, html)
if result:
    print(f' 部门：{result2.group(2)}，网址为：{result2.group(1)}')
```

运行后的输出结果如下所示。

```
部门：教务部，网址为：jwb.html
```

3. re.findall() 函数

re.findall() 的参数和 re.search() 一样，但是 re.findall() 可以匹配所有符合正则表达式的字符串，每次匹配到的目标会被构成元组，所有的元组以列表对象的形式返回，示例代码如下所示。

```
import re
html = '''
<div id="institution">
<ul>
<li class="xz"><a href="jwb.html"> 教务部 </a></li>
<li id="finance" class="xz"><a href="cwb.html"> 财务部 </a></li>
<li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span></a></li>
<li class="jx dept"><a href="fzxy.html"> 服装学院 </a></li>
<li class="jx"><a class="jx" href="ggkb.html"> 公共课部 </a></li>
<li class="xz"><a href="hqb.html"> 后勤部 </a>
</ul>
</div>
'''
pattern3 = re.compile('<a.*?href="(.*?)">(.*?)</a>', re.S)
result3 = re.findall(pattern3, html)
if result3:
    print(result3)
```

运行后的输出结果如下所示。

```
[('jwb.html', ' 教务部 '), ('cwb.html', ' 财务部 '), ('fzxy.html', '<span class="bold"> 纺织学院 </span>'), ('fzxy.html', ' 服装学院 '), ('ggkb.html', ' 公共课部 '), ('hqb.html', ' 后勤部 ')]
```

二、使用 BeautifulSoup 解析

通过正则表达式已经可以解析网页的数据，但是正则表达式写起来比较麻烦，有一点错误就会导致解析失败，编码效率较低。网页的 HTML 源码是一种类似于 XML 格式的文档，如果能够把它当做 XML 格式的文档处理，就可以高效地访问节点。

有很多第三方库就专门提供了对 XML 和 HTML 格式文档的处理功能，在这些第三方库中有一种叫做 BeautifulSoup 的解析库。下面介绍 BeautifulSoup 解析库的安装及用法。

(一) 安装 BeautifulSoup 库

本书使用的版本是 BeautifulSoup 4。它在解析数据时还需要调用解析器，最常用的是 lxml 解析器。我们还需要安装 BeautifulSoup 库和 lxml 库。安装这两个第三方库都比较简单，在 Windows 中打开命令窗口执行如下命令。

```
pip install beautifulsoup4
pip install lxml
```

当看到执行结果中出现 Successfully installed... 字样时，表示对应的库安装成功，如图 1-2-2 所示。

图 1-2-2　安装 Beautifulsoup 4 库和 lxml 库

（二）BeautifulSoup 解析用法

接下来学习使用 BeautifulSoup 解析的基本用法。假设例 1-2-1 中有一个字符串 html（本任务后面的例题都用此字符串来举例），它是类 HTML 格式的。

例 1-2-1： 演示使用 BeautifulSoup 解析的基本用法，示例代码如下所示。

```
html = '''
<div id="institution">
<ul>
<li class="xz"><a href="jwb.html"> 教务部 </a></li>
<li id="finance" class="xz"><a href="cwb.html"> 财务部 </a></li>
<li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span></a></li>
<li class="jx dept"><a href="fzgcxy.html"> 服装工程学院 </a></li>
<li class="jx"><a class="jx" href="ggkb.html"> 公共课部 </a></li>
<li class="xz"><a href="hqb.html"> 后勤部 </a>
</ul>
</div>
'''
from bs4 import BeautifulSoup          # 从 bs4 中引入 BeautifulSoup 模块
soup = BeautifulSoup(html,'lxml')      # 初始化对象
lis = soup.ul.li                       # 搜索 ul 下的 li（实际上是第一个 li）
print(type(lis))
print(lis)
print(lis.text)                        # 打印 li 节点的文本
```

运行该代码后，得到的输出结果如下所示。

```
<class 'bs4.element.Tag'>
<li class="xz"><a href="jwb.html"> 教务部 </a></li>
教务部
```

在例 1-2-1 中，可以看到初始化 soup 对象后，用 soup.ul.li 就可以得到第一个 li 节点，非常方便。

得到的节点对象类型为 bs4.element.Tag 对象，通过 text 属性可以得到该节点的非属性文本（实际上是其子节点 a 的文本）。

（三）节点选择器

在初始化得到了 soup 对象之后，可以通过“.节点名”的方式选择节点元素，这种方式就称为节点选择器。BeautifulSoup 还可以使用嵌套选择、子节点和子孙节点、父节点和祖先节点、兄弟节点等关联选择方式来选择节点元素。

1. 嵌套选择

在例 1-2-1 中，soup.ul.li 就是一种嵌套选择的用法，它实际上是先通过 soup.ul 选择了字符串中的 ul 节点，返回的对象是 bs4.element.Tag 类型，再通过 .soup.ul.li 选择 li 节点，得到的结果依然是 bs4.element.Tag 类型。

2. 子节点和子孙节点

有时需要得到某个节点的子节点或子孙节点可以分别通过 children 或 descendants 属性获取，如例 1-2-2 所示。

例 1-2-2： 输出“纺织学院”所在的 span 节点，示例代码如下所示。

```
lis = soup.ul.children
print(lis)
for i, li in enumerate(lis):
    print(i, li)
```

例 1-2-2 中选择了 ul 节点，再得到其子节点，也就是 6 个 li 节点。print(lis) 输出的是由所有子节点构成的一个生成器，是内存中的地址，无法直接看到内容。在 for 循环中用 enumerate() 处理 lis 是为了给每个子节点编号。运行该案例代码，截取的部分输出如下所示。

```
<list_iterator object at 0x0000020EE0114790>
0


1 <li class="xz"><a href="jwb.html"> 教务部 </a></li>
2


3 <li class="xz" id="finance"><a href="cwb.html"> 财务部 </a></li>
```

结果中每个子节点都是在一个从 0 开始的序号后面输出的。注意，输出后的结果中含有空行，这是因为每个 li 后有换行，所以在结果中也换行了。**特别注意：在 BeautifulSoup 的节点中会把节点内部的文本及换行都看做是子节点。**结果中编号为 0、编号为 2 的空行也是 ul 的子节点。

例 1-2-3： 输出 ul 节点的所有子孙节点，示例代码如下所示。

```
zs = soup.ul.descendants
for i, z in enumerate(zs):
    print(i, z)
```

运行代码，得到的输出结果如下所示。限于篇幅，这里只截取开头和结尾的一部分输出结果。

```
0

1 <li class="xz"><a href="jwb.html"> 教务部 </a></li>
2 <a href="jwb.html"> 教务部 </a>
3 教务部
4
……
22 <li class="xz"><a href="hqb.html"> 后勤部 </a>
</li>
23 <a href="hqb.html"> 后勤部 </a>
24 后勤部
25
```

通过 soup.ul.descendants 得到的是 ul 的所有直接子节点和所有子孙节点（里面的每一层子孙都会得到）。

3. 父节点和祖先节点

要选择某个节点的父节点，可以通过 parent 属性获取；要得到某个节点的祖先节点，可以通 parents 属性获取。它们的用法与 children 和 descendants 类似，这里就不举例了。

4. 兄弟节点

兄弟节点就是某个节点的同级节点。要取得某个节点的兄弟，可以使用 next_sibling、next_siblings、previous_sibling、previous_siblings 获取。next_sibling 表示下一个兄弟节点，next_siblings 表示后面的所有同级节点，previous_sibling 表示前面的一个同级节点，previous_siblings 表示前面的所有同级节点。下面通过例 1-2-4 来说明。

例 1-2-4： 演示 next_sibling 和 next_siblings 的用法，示例代码如下所示。

```
jwb_li = soup.ul.li
print(jwb_li)
enter_li = jwb_li.next_sibling          # 教务部所在 li 节点的下一个兄弟节点是换行符
print(f'enter_li:{enter_li}')
cwb_li = enter_li.next_sibling          # 第二个兄弟节点，即财务部所在的 li 节点
print(cwb_li)
print('----------'* 3)
others_li = cwb_li.next_siblings        # 财务部所在 li 节点后面的所有兄弟节点
for i,li in enumerate(others_li):       # 用 for 循环遍历
    print(f' 第 {i} 个兄弟节点：{li}')
```

运行代码，得到的输出结果如下所示。

```
<li class="xz"><a href="jwb.html"> 教务部 </a></li>
enter_li:
```

```
    <li class="xz" id="finance"><a href="cwb.html"> 财务部 </a></li>
    ------------------------------
    第 0 个兄弟节点：

    第 1 个兄弟节点：<li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span>
</a></li>
    第 2 个兄弟节点：

    第 3 个兄弟节点：<li class="jx dept"><a href="fzgcxy.html"> 服装工程学院 </a></li>
    第 4 个兄弟节点：

    第 5 个兄弟节点：<li class="jx"><a class="jx" href="ggkb.html"> 公共课部 </a></li>
    第 6 个兄弟节点：

    第 7 个兄弟节点：<li class="xz"><a href="hqb.html"> 后勤部 </a>
    </li>
```

cwb_li.next_siblings 的结果是财务部所在的 li 节点后面的所有同级节点组成的生成器，可以用 for 循环遍历。在结果中，编号为 0、2、4、6 的几个节点都是换行符，所以显示空行，编号为 1、3、5、7 的都是 li 节点。

（四）提取节点信息

选择好节点后，就可以提取节点的属性信息或文本信息。提取属性信息可以用 attrs[属性名]，提取文本信息可以用 string、text 属性，还可用 get_text() 方法。

通过 string 和 text 这两个属性获取的节点内的非属性文本基本上是一样的，它们可以获得指定节点或者其子孙节点的文本。当某个节点的标签没有封闭的时候，用 string 属性不能获得结果，但是 text 属性可以获取结果，且 text 获取的结果最后有一个换行符。用 get_text() 方法也可获取未封闭标签内的文本，但不包含最后的换行符，这个和 text 属性稍有区别。

例 1-2-5： 演示提取节点的非属性文本及节点属性的用法，示例代码如下所示。

```
    jwb_li = soup.ul.li                          # 选择第一个 li 节点，即教务部所在的 li 节点
    print(f'jwb_li: {jwb_li}')
    print(f'jwb_li.text = {jwb_li.text}')
    enter_li = jwb_li.next_sibling
    cwb_li = enter_li.next_sibling
    print(f'cwb_li.string = {cwb_li.string}')
    fzxy = soup.select('li:nth-child(3)')[0]    # 使用 CSS 选择器选择纺织学院所在的 li 节点
    print(fzxy)
    print(f'fzxy.string = {fzxy.string}')
    print(f' 教务部的超链接 href 属性：{jwb_li.a.attrs["href"]}')       # 获得 a 节点 href 属性的值
```

```
print(f'纺织学院的超链接为：{fzxy.a.attrs["href"]}')              # 获得 a 节点 href 属性的值
print(f'纺织学院所在 li 节点的 class 属性为：{fzxy.attrs["class"]}') # 获得 class 属性的值
print('---------- 演示 text 和 string 的区别 ----------')
hqb = soup.select('li:last-child')[0]
print(f'后勤部所在的 li：{hqb}')
print(f'hqb.string 的结果：{hqb.string}。')
print(f'hqb.text 的结果：{hqb.text}。')
print(f'hqb.get_text() 的结果：{hqb.get_text()}。')
```

运行代码，输出结果如下所示。

```
jwb_li：<li class="xz"><a href="jwb.html"> 教务部 </a></li>
jwb_li.text = 教务部
cwb_li.string = 财务部
<li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span></a></li>
fzxy.string = 纺织学院
教务部的超链接 href 属性：jwb.html
纺织学院的超链接为：fzxy.html
纺织学院所在 li 节点的 class 属性为：['jx', 'dept']
---------- 演示 text 和 string 的区别 ----------
后勤部所在的 li：<li class="xz"><a href="hqb.html"> 后勤部 </a>
</li>
hqb.string 的结果：None。
hqb.text 的结果：后勤部


hqb.get_text() 的结果：后勤部
```

在例 1-2-5 中的“ soup.select('li:nth-child(3)')[0]”一行用到了后面要介绍的 CSS 选择器，其中“li:nth-child(3)”表示选择第三个 li 节点，即纺织学院所在的 li 节点。

程序中第 10、第 11 行的 print() 语句是获取所在 li 内部的 a 节点的 href 属性，即超链接。

程序中第 12 行的 print() 语句是获取纺织学院所在的 li 节点的 class 属性，这个 li 的 class 有多个值，所以输出的是一个含有多个值的列表。

需要重点地关注一下最后 5 行代码的输出，这里使用“ select('li:last-child')”选择了最后一个 li 节点（即后勤部所在的 li 节点），该 li 节点的 li 标签没有结束的“ ”，在用 string 属性获取文本时没有结果，所以显示 None。用 text 属性获取文本会有结果，最后的“。”换行是因为这种情况下 text 属性获取的文本末尾有个换行符。用 get_text() 方法也获取到了文本，但是文本末尾没有换行符。

（五）方法选择器

BeautifulSoup 除了提供前面的节点选择器外，还提供了更灵活的方法选择器，主要有 find() 方法和 find_all() 方法。find() 方法查找的结果是符合要求的第一个节点（单个），而 find_all() 方法查找的结

果是符合要求的多个节点构成的列表。除了返回结果类型不同外，它们的用法基本类似。两个方法的用法格式如下。

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

其中各参数的含义如下。

（1）name：标签名或标签列表，可以是字符串或正则表达式。

（2）attrs：标签属性或属性字典，可以是字符串或字典。

（3）recursive：布尔值，表示是否对子孙节点进行递归搜索，默认为 True。

（4）text：可以是字符串或正则表达式，用于查找指定文本。

（5）kwargs：其他属性参数，如 class_。

下面通过例 1-2-6 来演示它们的用法。

例 1-2-6： 演示 find() 和 find_all() 方法，示例代码如下所示。

```
from bs4 import BeautifulSoup                    # 从 bs4 中引入 BeautifulSoup 模块
import re                                        # 引入正则表达式模块 re
soup = BeautifulSoup(html,'lxml')                # 初始化 BeautifulSoup 对象
# 多个查找条件，其中 text 为正则表达式
fzgcxy_a = soup.find(name='a', text=re.compile(' 服装 '))
print(f'fzgcxy_a = {fzgcxy_a}')
fzxy1 = soup.find(class_="jx")                   # 查找到的是符合条件的第一个 li，即纺织学院
print(f'fzxy1 结果为：{fzxy1}')
two_dept = soup.find_all(name="li", class_="jx dept") # 演示多个查找条件
print(f'two_dept 结果为：{two_dept}')
```

运行代码，输出结果如下所示。

```
fzgcxy_a=<a href="fzgcxy.html"> 服装工程学院 </a>
fzxy1 结果为：<li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span></a></li>
two_dept 结果为：[<li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span></a></li>, <li class="jx dept"><a href="fzgcxy.html"> 服装工程学院 </a></li>]
```

注意：

使用节点的 class 属性作查询条件时使用“class_”。对于节点的 id 属性和 class 属性可以采用简化形式输入，如 id="finance"、class_="jx dept"，但如果是使用其他属性作为方法的查询条件则需要用 attrs 参数的一般形式，即 attrs={"id":"finance"} 的形式。

在例 1-2-6 中，“text=re.compile(' 服装 ')”表示 text 参数传入了一个正则表达式，是搜索节点文字包含“服装”的节点的意思。

（六）CSS 选择器

前面介绍了节点选择器和方法选择器，除此之外还有更方便的选择器——CSS 选择器。在 HTML

文档中，很多节点标签都具有一些类似 id、class、name 等属性，在添加 CSS 时，经常会用节点上面的一些属性来定位节点标签。BeautifulSoup 的 Tag 对象的 CSS 选择器也是利用同样的原理来定位节点的。

使用 CSS 选择器时，需调用 Tag 对象的 select() 方法，在方法中传入节点相应的 CSS 选择器字符串。select() 方法返回的结果为列表类型，列表元素就是选中的节点。

例 1-2-7： 演示 select() 方法，示例代码如下所示。

```
from bs4 import BeautifulSoup                    # 从 bs4 中引入 BeautifulSoup 模块
soup = BeautifulSoup(html,'lxml')                # html 为例 1-2-1 中定义的字符串
cwb = soup.select('#finance')                    # 查找 id="finance" 的节点
print(f'cwb 的结果为：{cwb}')
print(f'cwb 中第一个元素为：{cwb[0]}，取出 cwb[0] 的文本为：{cwb[0].string}')
xz_lis = soup.select('li.xz')                    # 查找 class 属性为 xz 的 li 节点
print(f'xz_lis 的结果为：\n{xz_lis}')
print('--------- 如果返回的列表元素依然是 Tag 类型对象，还可以嵌套调用 select() -------')
for li in xz_lis:
    print(f' 行政部门 "{li.text.strip()}" 的链接地址为：{li.a.attrs["href"]}。')

print('------------ 下面演示伪类选择器 --------------')
jwb = soup.select('li.xz:nth-child(1)')      # 查找 class 属性为 xz 的第一个 li 节点
print(f'jwb 的结果为：{jwb}')
print(f'jwb[0] 的结果为：{jwb[0]}')
print(f'jwb[0].text 的结果为：{jwb[0].text}')
hqb = soup.select('li:last-child')[0].text
print(f' 最后一个 li 中的文本为：{hqb}')
```

运行代码，输出结果如下所示。

```
cwb 的结果为：[<li class="xz" id="finance"><a href="cwb.html"> 财务部 </a></li>]
cwb 中第一个元素为：<li class="xz" id="finance"><a href="cwb.html"> 财务部 </a></li>，取出 cwb[0] 的文本为：财务部
xz_lis 的结果为：
[<li class="xz"><a href="jwb.html"> 教务部 </a></li>, <li class="xz" id="finance"><a href="cwb.html"> 财务部 </a></li>, <li class="xz"><a href="hqb.html"> 后勤部 </a>
</li>]
---------- 如果返回的列表元素依然是 Tag 类型对象，还可以嵌套调用 select() ----------
行政部门 " 教务部 " 的链接地址为：jwb.html。
行政部门 " 财务部 " 的链接地址为：cwb.html。
行政部门 " 后勤部 " 的链接地址为：hqb.html。
------------ 下面演示伪类选择器 --------------
jwb 的结果为：[<li class="xz"><a href="jwb.html"> 教务部 </a></li>]
jwb[0] 的结果为：<li class="xz"><a href="jwb.html"> 教务部 </a></li>
```

```
jwb[0].text 的结果为：教务部
最后一个 li 中的文本为：后勤部
```

例 1-2-7 中演示了在 select() 方法传入各种节点的 CSS 选择器字符串从而获得不同结果。可以发现，不管结果是一个节点还是多个节点，select() 返回的都是列表，因此如果需要用到其中的某个节点，就需要从返回的列表中用 [] 运算符取出对应的元素。

另外，列表中的元素类型依然是 Tag 类型的节点对象，这意味着如果需要在该节点的子孙中查找节点，还是可以用节点选择器、方法选择器或 CSS 选择器来获取节点的属性及文本。例 1-2-7 中的 for 循环就演示了对得到的 class 属性为 xz 的 3 个 li 节点的操作：获取文本、获取子节点 a 的 href 属性。

例 1-2-7 也演示了在 select() 方法中使用伪类选择器，伪类选择器是 CSS 选择器比较常用的用法。常见的 CSS 选择器如表 1-2-4 所示。

表 1-2-4　常见的 CSS 选择器

用法	说明
select("li")	表示通过节点标签名选择。如 select("a")、select("ul")
select("#finance")	表示通过节点的 id 属性查找。还可以结合节点名和 id 属性一起使用，如 select ("li#finance")
select(".xz")	表示通过节点的 class 属性查找。如果节点 class 有多个值，可以用多个“.”引导，如 select("li.jx.dept")
select("li.xz:nth-child(1)")	表示查找第一个符合“：”前面 CSS 选择器的节点，“:”与后面的“nth-child(1)”中间不能有空格
select("li.xz:nth-child(2)")	表示查找第二个符合“:”前面 CSS 选择器的节点
select("li:nth-child(2n") select("li:nth-child(2n+1)")	表示查找第偶数个符合前面 CSS 选择器的节点。如要查找第奇数个，只需改为 select("li:nth-child(2n+1)")
select("li:first-child") select("li:last-child")	分别表示符合“:”前面 CSS 选择器的第一个、最后一个节点
select("ul li.jx")	表示依据上次层级来定位节点，在不同层级之间用空格或者“>”隔开，select("ul> li.jx") 和 select("ul li.jx") 的效果是一样的

三、使用 XPath 解析

XPath 是基于文档的层次结构来确定查找路径的，它最初用来在 XML 文档中解析数据，由于 HTML 的文档结构类似于 XML，因此也可以用来解析 HTML。

使用 XPath 时，需要借助 lxml 库将 HTML 转换为 XML 文档树对象，之后就可以使用 XPath 语法查找此结构中的节点或元素。

（一）安装 lxml

Python 标准库中自带了 lxml 模块，但是性能不够好，而且缺乏一些人性化的 API（应用程序接

口）。相比之下，用 Python 实现的第三方库 lxml 增加了很多实用的功能，使用起来非常方便。lxml 大部分功能都存在于 lxml.etree 中。

（二）XPath 的基本使用

XPath 是一门在类 XML 文档中查找信息的语言。XPath 可用来在 HTML 或 XML 文档中对元素和属性进行遍历，下面介绍 XPath 解析的基本操作。

1. XPath 解析对象的初始化

在使用 XPath 对类 XML 或 HTML 文档进行数据解析之前，需要先进行初始化工作。有两种常见方式：一种是用字符串初始化；另一种是用文件初始化。

（1）用字符串初始化。该方式需要从 lxml 引入 etree 模块，然后用 etree.HTML() 方法完成初始化，该方法中的参数为字符串。

例 1-2-8：演示用字符串初始化 XPath 解析对象，示例代码如下所示。

```
from lxml import etree
html = '''
<div id="institution">
<ul>
<li class="xz"><a href="jwb.html"> 教务部 </a></li>
<li id="finance" class="xz"><a href="cwb.html"> 财务部 </a></li>
<li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span></a></li>
<li class="jx dept"><a href="fzgcxy.html"> 服装工程学院 </a></li>
<li class="jx"><a class="jx" href="ggkb.html"> 公共课部 </a></li>
<li class="xz"><a href="hqb.html"> 后勤部 </a>
</ul>
</div>
'''
xp = etree.HTML(html)                    # 调用 HTML 类初始化构造一个 XPath 对象
print(xp.xpath('//ul//a/text()'))        # 获取文档中所有 ul 节点下的所有 a 节点的文本
```

运行代码，输出结果如下所示。

```
['教务部', '财务部', '服装工程学院', '公共课部', '后勤部']
```

（2）用文件初始化。如果有一个 HTML 文件，可以直接通过调用 etree.parse() 方法用文件构造 XPath 解析对象，方法中的第一个参数为带路径的文件名，第二个参数为 etree.HTMLParser()。

例 1-2-9：演示用文件初始化 XPath 解析对象，示例代码如下所示。

```
from lxml import etree
parser = etree.HTMLParser(encoding='utf-8')     # 定义解析器使用 utf-8 编码，以免中文乱码
htmlElement = etree.parse('./institution.html',parser=parser)     # 初始化 XPath 解析对象
```

```
    # 获得解析对象的 bytes 字符串，指定 utf-8 编码
    text = etree.tostring(htmlElement,encoding='utf-8')
    print(text)
    print('-----------------------------')
    # 用 decode() 方法解码 bytes 字符串，解码时指定 utf-8 编码
    print(text.decode('utf-8'))
    print('-----------------------------')
    print(htmlElement.xpath('//li[1]/a/text()'))          # 查找第一个 li 节点下的 a 节点，获取文本
```

在运行例 1-2-9 前，请先准备 HTML 文件并将其放在本例程序文件的同目录下。可以使用例 1-2-8 中的字符串 html 作为 HTML 文件中的内容。用记事本程序打开准备好的 HTML 文件，将 html 字符串内容复制粘贴进去，然后另存为“institution.html”，如图 1-2-3 所示。

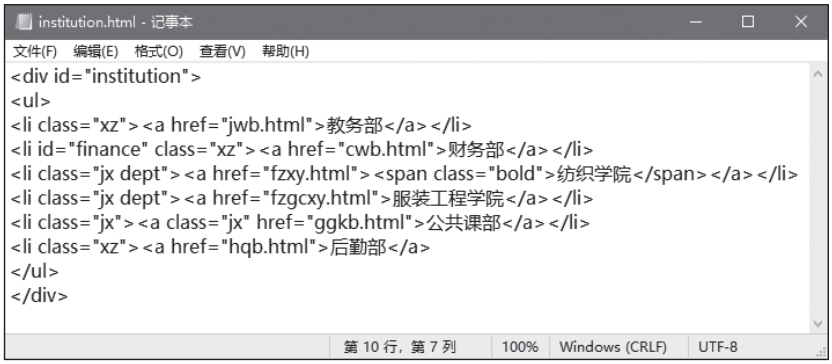


图 1-2-3　复制 html 字符串内容

例 1-2-9 演示了用 HTML 文件初始化 XPath 解析对象，然后用 etree.tostring() 将初始化后的字符串 html 输出，但是它是 bytes 类型，需要使用 decode() 将其解码成 str 类型。运行后的部分输出结果如下所示。

```
    b'<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN""http://www.w3.org.....
    <html><body><div id="institution">&#13;\n<ul>&#13;\n<li class="xz"><a href="jwb.html">\xe6\
x95\x99\xe5\x8a\xa1\xe9\x83\xa8</a></li>&#13;\n<li id="finance" class="xz"><a href="cwb.html">\xe8\
xb4\xa2\xe5\x8a\xa1\xe9\x83\xa8</a></li>&#13;\n.....
    </body></html>'
    ------------------------------
    <!DOCTYPE .....
    <html><body><div id="institution">&#13;
    <ul>&#13;
    <li class="xz"><a href="jwb.html"> 教务部 </a></li>&#13;
    <li id="finance" class="xz"><a href="cwb.html"> 财务部 </a></li>&#13;
    <li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span></a></li>&#13;
    <li class="jx dept"><a href="fzgcxy.html"> 服装工程学院 </a></li>&#13;
```

```
<li class="jx"><a class="jx" href="ggkb.html"> 公共课部 </a></li>&#13;
<li class="xz"><a href="hqb.html"> 后勤部 </a>&#13;
</li></ul>&#13;
</div></body></html>
--------------------------------
[' 教务部 ']
```

从运行结果来看，用 etree.tostring() 得到的是字节型的字符串，用 decode() 方法解码后就会得到 str 类型的字符串，还可以在 decode() 方法中传入编码类型以防止乱码。另外，初始化为 XPath 解析对象后，原来文档中缺少的标签 <html><body></body></html> 都被修正了，如果某个标签未闭合，也会被修正。

2. 节点的选取

XPath 使用路径表达式在类 XML 文档中选取节点。XPath 路径规则如表 1-2-5 所示。

表 1-2-5　XPath 路径规则

表达式	描述
nodename	选取此节点的所有子节点
/	从当前节点的直接子节点中选取，如在路径表达式最前面，则表示从文档的根节点开始
//	从当前节点的子孙节点中选取
.	选取当前节点
..	选取当前节点的父节点
@	选取属性

表 1-2-6 列出了一些路径表达式以及表达式的结果。

表 1-2-6　路径表达式及其结果

路径表达式	结果
bookstore	选取 bookstore 元素的所有子节点
/bookstore	选取根元素 bookstore。假如路径起始于正斜杠（/），则此路径始终代表某元素的绝对路径
bookstore/book	选取属于 bookstore 子元素的所有 book 元素
//book	选取所有 book 子元素，不管它们在文档中的位置
bookstore//book	选择属于 bookstore 元素后代的所有 book 元素，不管它们位于 bookstore 之下的什么位置
//@lang	选取名为 lang 的所有属性

例 1-2-8 的最后一行代码“xp.xpath('//ul//a/text()')”表示在文档的所有节点中搜索 ul，然后在 ul 节点的子孙节点中搜索 a 节点，最后获得这些 a 节点的文本内容。

例 1-2-9 的最后一行代码“htmlElement.xpath('//li[1]/a/text()')”表示搜索文档中第一个 li 节点，再

搜索其子节点中的 a 节点，并获取文本。

在节点选取中，除了使用表 1-2-1 中的路径规则外，往往还需要用节点关系来定位节点，如父节点、子节点。另外，还可以对同级节点按序选择。在选择节点时，也可以通过属性来匹配节点，下面就对这些方式进行介绍。

（1）父节点（parent）。

例 1-2-10： 演示选择父节点，示例代码如下所示。

```
html = '''
<book>
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2019</year>
    <price>29.99</price>
</book>'''
from lxml import etree
htmlElement = etree.HTML(html)
title = htmlElement.xpath('//title')[0]          # 搜索 title 节点，并从列表中取出第一项
book = title.xpath('..')                         # 搜索 title 节点的父节点
print(book)
print(type(book[0]))                             # 输出 book[0] 的类型
print(etree.tostring(book[0]).decode('utf-8'))   # 解码成字符串输出
```

运行代码，输出结果如下所示。

```
[<Element book at 0x20f3a051740>]
<class 'lxml.etree._Element'>
<book>
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2019</year>
    <price>29.99</price>
</book>
```

book 节点是 title、author、year 以及 price 元素的父节点。在例 1-2-10 中首先用“xpath('//title')[0]”搜寻 title 节点，由于 xpath() 返回的是列表，这里用“[0]”取第出一个元素，然后用“title.xpath('..')”得到其父节点 book。输出的第一行是一个列表，有一个列表元素，很明显该列表元素是一个对象在内存中的地址，这个对象类型为 lxml.etree._Element，也是所有节点对象的类型。最后将 book[0] 节点对象解码成 str 输出。

（2）子节点（children）。

一个节点可以有零个、一个或多个子节点。在例 1-2-10 的 HTML 结构中，title、author、year 以及 price 元素都是 book 元素的子节点。为了节省篇幅，下面在例 1-2-10 的基础上举例。

例 1-2-11：演示选择某节点的子节点，示例代码如下所示。

```
print(book[0].xpath('./title/text()'))
print(book[0].xpath('./year/text()'))
```

运行代码，输出结果如下所示。

```
['Harry Potter']
['2019']
```

（3）按序选择节点。

在使用路径规则时，可能会匹配到多个节点，同时又可能需要取出其中一个节点，如第一个、第二个、最后一个节点，此时就可以使用按序选择节点。

例 1-2-12：演示按序选择节点，示例代码如下所示。

```
from lxml import etree
html = '''
<div id="institution">
<ul>
<li class="xz"><a href="jwb.html"> 教务部 </a></li>
<li id="finance" class="xz"><a href="cwb.html"> 财务部 </a></li>
<li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span></a></li>
<li class="jx dept"><a href="fzgcxy.html"> 服装工程学院 </a></li>
<li class="jx"><a class="jx" href="ggkb.html"> 公共课部 </a></li>
<li class="xz"><a href="hqb.html"> 后勤部 </a>
</ul>
</div>
'''
xp = etree.HTML(html)                                  # 初始化 XPath 解析对象
result1 = xp.xpath('//li[1]/a/text()')                 # 第一个 li 节点下的 a 节点的文本
print(f'result1[0]={result1[0]}')
result2 = xp.xpath('//li[2]/a/text()')                 # 第二个 li 节点下的 a 节点的文本
print(f'result2[0]={result2[0]}')
result3 = xp.xpath('//li[last()]/a/text()')            # 最后一个 li 节点下的 a 节点的文本
print(f'result3[0]={result3[0]}')
result4 = xp.xpath('//li[position()<3]/a/text()')      # 前两个 li 节点下的 a 节点的文本
print(f'result4={result4}')                            # 注意没有用 [0]，因为有多个节点的文本
result5 = xp.xpath('//li[last()-1]/a/text()')          # 最后一个 li 前面的一个 li 下的 a 节点的文本
print(f'result5[0]={result5[0]}')
```

该段代码的运行结果如下所示。

```
result1[0]= 教务部
result2[0]= 财务部
result3[0]= 后勤部
result4=['教务部','财务部']
result5[0]= 公共课部
```

result1 通过在路径 li 后使用“[1]”选择了第一个 li 节点，result2 选择了第二个 li 节点；result3 是在路径中的 li 后面使用“last()”选择了最后一个 li 节点；result4 使用“position()<3”按序选择了前两个 li 节点；result5 使用“last()-1”按序选择了倒数第二个 li 节点。

（4）属性匹配。

在选择节点的时候还可以通过节点的属性来匹配节点，这样在选取时就多了一些限制，可以帮助我们快速搜寻节点。使用属性匹配是在路径规则表达式的某个节点后面用方括号“[]”，在方括号里面用“@ 属性名 = 值”的形式限制属性。这只适合匹配单值属性，匹配多值属性时要用 contains()。

①属性为单值时的匹配。

下面在例 1-2-12 的基础上举例，假如要查找“财务部”所在的 li 节点，该 li 节点有一个 id="finance" 的属性和 class="xz" 的属性，这时可以用属性匹配帮助搜寻。

例 1-2-13： 演示属性匹配用法，示例代码如下所示。

```
print(xp.xpath('//li[@id="finance"]/a/text()'))
```

此行代码的输出为“['财务部']”，同样也可以用另一个 class 属性匹配，如例 1-2-14 所示。

例 1-2-14： 演示用 class 属性匹配用法，示例代码如下所示。

```
print(xp.xpath('//li[@class="xz"]/a/text()'))
```

此行代码的输出为“['教务部','财务部','后勤部']”，输出结果多了两个部门名称，这是因为原来的文档中满足 class="xz" 的有三个行政部门。如果想要区分，可以再添加一个属性来匹配，或者用按序选择，如例 1-2-15 所示。

例 1-2-15： 演示多个属性的匹配用法，示例代码如下所示。

```
print(xp.xpath('//li[@id="finance" and @class="xz"]/a/text()'))# 多个属性匹配，用 and 连接
print(xp.xpath('//li[@class="xz"][1]/a/text()'))              # 结合属性匹配和按序选择第一个
```

②属性为多值时的匹配。

当某个属性的值含有多个值时，如在例 1-2-12 的 HMTL 结构中，“服装工程学院”所在的 li 节点的 class 属性含多个值，分别为 jx 和 dept，这时就不能用“[@ 属性名 = 值]”这种方式匹配，需要用“[contains(@ 属性名，值)]”的方式匹配节点，如例 1-2-16 所示。

例 1-2-16： 演示属性的值有多个值的匹配用法，示例代码如下所示。

```
print(xp.xpath('//li[contains(@class,"jx") and contains(@class, "dept")]/a/text()'))
```

此行语句的输出结果为“['服装工程学院']”。

3. 节点文本的提取

在 XPath 中，如果要提取某个节点的非属性文本，可以在 XPath 规则表达式后面添加“/text()”。

例如，例 1-2-16 中的“ xp.xpath('//li[contains(@class,"jx") and contains(@class, "dept")]/a/text()')”是提取节点 a 的文本，输出结果是一个列表“ [' 服装工程学院 ']”。如果要得到该学院的字符串还要选取此列表的第一个元素，在后面添加“[0]”，示例代码如下所示。

```
print(xp.xpath('//li[contains(@class,"jx") and contains(@class, "dept")]/a/text()')[0])
```

此行语句的输出结果为“服装工程学院”。

4. 节点属性的提取

得到某个节点后，经常需要获取节点的某个属性的值，可以通过在路径表达式后面增加“ /@ 属性名”来获得属性值。

例 1-2-17：演示节点属性的提取，示例代码如下所示。

```
cwb = xp.xpath('//li[@id="finance"]/a')                    # 先查找财务部的 a 节点
# 再取 href 属性，用 [0] 得到第一个元素的内容
print(f' 财务部的链接地址为：{cwb[0].xpath("./@href")[0]}')
# 将查找财务部的 a 节点和取 href 属性合并
cwb_link = xp.xpath('//li[@id="finance"]/a/@href')[0]
print(f' 财务部的链接地址为 cwb_link={cwb_link}')
all_a_links = xp.xpath('//li/a/@href')                     # 所有 li 下 a 节点的 href 属性列表
print(all_a_links)
```

运行代码，得到的输出结果如下所示。

```
财务部的链接地址为：cwb.html
财务部的链接地址为 cwb_link=cwb.html
['jwb.html', 'cwb.html', 'fzxy.html', 'fzgcxy.html', 'ggkb.html', 'hqb.html']
```

四、使用 PyQuery 解析

PyQuery 解析库是一个类似于 jQuery 的 Python 库，借助它能够在 XML 文档中进行 jQuery 查询，PyQuery 使用 lxml 解析器可以在 XML 和 HTML 文档中进行快速操作，支持 CSS 选择器，使用起来非常方便。

（一）准备工作

准备工作包含 PyQuery 库的安装和 PyQuery 对象的初始化。

1. PyQuery 的安装

PyQuery 的安装采用 pip 安装方式，请在命令窗口中输入如下命令安装。

```
pip install pyquery
```

安装过程中需要联网下载文件。安装过程中的反馈信息如下所示。

```
C:\Users\jacqu>pip install pyquery
Collecting pyquery
    Using cached pyquery-2.0.0-py3-none-any.whl (22 kB)
Requirement already satisfied: lxml>=2.1 in c:\python39\lib\site-packages (from pyquery) (4.9.3)
Requirement already satisfied: cssselect>=1.2.0 in c:\python39\lib\site-packages (from pyquery)
(1.2.0)
Installing collected packages: pyquery
Successfully installed pyquery-2.0.0
```

出现“Successfully installed pyquery-2.0.0”的字样表示安装成功，可以验证一下是否可用。在命令窗口中进入 Python 交互环境，输入如下命令，如果没有报异常信息则表明 pyquery 安装成功。

```
python
from pyquery import PyQuery as pq
```

注意：

代码中前面 pyquery 小写，后面一个 PyQuery 中 P 和 Q 大写。如执行上面的代码没有任何提示，就说明 pyquery 可以使用了。

2. PyQuery 对象的初始化

PyQuery 的初始化有多种方式，在爬虫编程中比较常用的是传入 HTML 的文本（字符串），除此之外，它还支持传入网页的 URL、传入文件名等方式来初始化。

（1）传入字符串初始化。

例 1-2-18： 将字符串传入 PyQuery 构造函数完成初始化，示例代码如下所示。

```
html = '''
<div>
<ul>
<li class="xz"><a href="jwb.html"> 教务部 </a></li>
<li class="xz"><a href="cwb.html"> 财务部 </a></li>
<li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span></a></li>
<li class="jx dept"><a href="fzxy.html"> 服装工程学院 </a></li>
<li class="jx"><a href="ggkb.html"> 公共课部 </a></li>
<li class="xz"><a href="hqb.html"> 后勤部 </a>
</ul>
</div>
'''
from pyquery import PyQuery as pq       # 引入 PyQuery 模块并赋予别名 pq
doc = pq(html)                          # 传入字符串给 pq(), 并创建 PyQuery 对象
print(doc('li'))                        # CSS 选择器搜寻 li 节点
```

在使用字符串初始化的案例中，首先从 pyquery 模块中引入 PyQuery 这个类，并取别名为 pq。然后将字符串作为参数传递给 PyQuery 类的构造方法，这样就完成了初始化并保存到 doc 变量中。doc('li') 实际上类似于 CSS 选择器，在例 1-2-18 中传入的是节点标签名称 li，表示要查找所有 li 节点。运行代码，得到的输出结果如下所示。

```
<li class="xz"><a href="jwb.html"> 教务部 </a></li>
<li class="xz"><a href="cwb.html"> 财务部 </a></li>
<li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span></a></li>
<li class="jx dept"><a href="fzxy.html"> 服装工程学院 </a></li>
<li class="jx"><a href="ggkb.html"> 公共课部 </a></li>
<li class="xz"><a href="hqb.html"> 后勤部 </a>
</li>
```

（2）传入 URL 初始化。

例 1-2-19： 演示用 URL 完成 PyQuery 的初始化，示例代码如下所示。

```
from pyquery import PyQuery as pq
doc = pq(url='http://www.gdpt.edu.cn')
print(doc('title'))
```

运行代码，得到的输出结果如下所示。

```
<title> 广东职业技术学院 </title>&#13;
&#13;
```

（3）传入文件名初始化。如果已经将网页保存为文件，就可以利用 PyQuery 按照文件名初始化，此处可以将例 1-2-18 中的字符串粘贴到 txt 文件中，保存为“demo.html”使用。

例 1-2-20： 演示用 HTML 格式文件完成 PyQuery 的初始化，示例代码如下所示。

```
from pyquery import PyQuery as pq
doc = pq(filename='demo.html')
print(doc('li'))
```

执行结果与例 1-2-1 的结果一致。

（二）节点选择

使用 PyQuery 选择节点是非常方便的，它提供了 CSS 选择器，还提供了查找节点的方法选择器、伪类选择器。

1. CSS 选择器

CSS 选择器是 PyQuery 解析库的突出优势，它使用起来非常简洁灵活。我们熟练掌握之后，解析数据会更得心应手。

CSS 选择器使用 HTML 元素的标签名称或属性来查找元素。用于 CSS 选择器的属性主要是元素

的 id 属性、class 属性。熟悉这两个属性在 CSS 选择器中的写法非常重要，id 属性会被写作“#属性值”，class 属性会被写作“.属性值”。而某个元素如果有多个 class 属性值，在 CSS 选择器中会写作“.属性值 1 .属性值 2”（中间不能有空格，有空格就表示是子元素的 class 属性），可以参考表 1-2-4 中的介绍。

例 1-2-21： 演示 CSS 选择器的用法，示例代码如下所示。

```
html = '''
<div id="institution">
<ul>
<li class="xz"><a href="jwb.html"> 教务部 </a></li>
<li id="finance" class="xz"><a href="cwb.html"> 财务部 </a></li>
<li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span></a></li>
<li class="jx dept"><a href="fzxy.html"> 服装工程学院 </a></li>
<li class="jx"><a class="jx" href="ggkb.html"> 公共课部 </a></li>
<li class="xz"><a href="hqb.html"> 后勤部 </a>
</ul>
</div>
'''
from pyquery import PyQuery as pq
doc = pq(html)
print(doc('#institution ul li#finance'))
print(doc('#institution ul .jx'))
print(doc('.jx.dept'))
```

运行代码，得到的输出结果如下。

```
<li id="finance" class="xz"><a href="cwb.html"> 财务部 </a></li>

<li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span></a></li>
<li class="jx dept"><a href="fzxy.html"> 服装工程学院 </a></li>
<li class="jx"><a href="ggkb.html"> 公共课部 </a></li>

<li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span></a></li>
<li class="jx dept"><a href="fzxy.html"> 服装工程学院 </a></li>
```

代码最后 3 行是 3 条 print() 语句，输出结果有 3 组，用空行隔开。

第 1 组结果输出了包含“财务部”的 li 元素，对应倒数第 3 行代码，在其关键代码“doc('#institution ul li#finance')”中，先使用 id 属性查找，再在其子元素中查找 ul 元素，然后查找 id 属性为 finance 的元素，即财务部所在的 li 元素。

第 2 组输出结果对应代码中倒数第 2 行，它也是从“ ”元素开始，再在它的子元素中查找 ul，最后查找 class 属性为“jx”的元素，满足的结果有 3 个，即 3 个教学部门“纺织学

院”“服装工程学院”“公共课部”。

第 3 组输出结果对应最后一行代码。它的 CSS 选择器中的“ doc('.jx.dept')”表示查找 class 属性有两个值的元素，且 class 属性值分别为“ jx”和“ dept”。符合条件的有两个 li 元素，即包含“纺织学院”“服装工程学院”的 li 元素。注意，“.jx.dept”中间不能有空格。

2. 方法选择器

PyQuery 还提供了一些方法用来查找节点。这些查找方法有 find()、children()、parent() 和 siblings()。分别代表在子孙节点中查找、只在子节点中查找子节点、查找父节点和查找兄弟节点。

此处用之前的 html 字符串初始化 PyQuery 演示这 4 个方法的用法。

（1）用 find() 方法在所有子孙节点中查找，返回节点对象。

例 1-2-22：演示 find() 的用法，示例代码如下所示。

```
from pyquery import PyQuery as pq
doc = pq(html)
fin = doc.find('li#finance')
print(type(fin))
print(fin)
```

运行代码，得到的输出结果如下所示。

```
<class 'pyquery.pyquery.PyQuery'>
<li id="finance" class="xz"><a href="cwb.html"> 财务部 </a></li>
```

在 html 的子孙节点中查找 id 属性为“finance”的 li 节点时，只有“财务部”的 li 节点符合要求。

（2）用 children() 方法在子节点中查找，返回的是多个子节点构成的对象，可以使用 items() 方法得到一个可迭代的生成器。如果希望只在某个节点的直接子节点中查找，就可以使用 children() 查找。

例 1-2-23：演示 children() 的用法，示例代码如下所示。

```
from pyquery import PyQuery as pq
doc = pq(html)
lis = doc.find('.jx.dept')
for li in lis.items():
    print(li.children())
```

运行代码，得到的输出结果如下所示。

```
<a href="fzxy.html"><span class="bold"> 纺织学院 </span></a>
<a href="fzxy.html"> 服装工程学院 </a>
```

在例 1-2-23 中先查找到 class 属性值为“ jx dept”两个属性的节点，查询到两个节点，然后用遍历方式分别查找这两个 li 的子节点，即两个 a 节点。这里用到了 items() 方法，它会返回一个包含字典所有（键、值）元组的列表，后面会详细介绍。

（3）用 parent() 方法获取某个节点的父节点。

例 1-2-24： 演示 parent() 的用法，示例代码如下所示。

```
from pyquery import PyQuery as pq
doc = pq(html)
bold = doc.find('.bold')                    # 查找 class='bold' 的节点，即纺织学院所在的 span
li_fzxy = bold.parent().parent()            # 这里是查找父节点的父节点
print(li_fzxy)
```

运行代码，得到的输出结果如下所示。

```
<li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span></a></li>
```

此例中首先查找到 html 中 class 属性为“ bold”的节点（即 纺织学院 ），然后通过两次调用 parent() 方法来获得其父节点的父节点。

（4）siblings() 方法可以获得某个节点的兄弟节点。

例 1-2-25： 演示 siblings() 的用法，示例代码如下所示。

```
from pyquery import PyQuery as pq
doc = pq(html)
cwb = doc.find('li#finance')
siblings_li = cwb.siblings()
print(siblings_li)
for li in siblings_li.items():
    print(li.text())
```

运行代码，得到的输出结果如下所示。

```
<li class="xz"><a href="jwb.html"> 教务部 </a></li>
<li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span></a></li>
<li class="jx dept"><a href="fzxy.html"> 服装工程学院 </a></li>
<li class="jx"><a href="ggkb.html"> 公共课部 </a></li>
<li class="xz"><a href="hqb.html"> 后勤部 </a>
</li>
教务部
纺织学院
服装工程学院
公共课部
后勤部
```

在此例中，输出结果分为两部分。前面一部分是“ print(siblings_li)”的输出。首先查找 id 属性为“ finance”的 li 节点，然后通过它调用 siblings() 方法获得其兄弟节点，即其他 5 个 li 节点，所以第一

组的输出就是其他 5 个 li 节点。后面一部分是通过 for 循环遍历这 5 个 li 节点并在循环体里面用 text() 函数获取每个 li 节点及其子孙节点的文本内容。

3. 伪类选择器

伪类选择器就是当我们在 CSS 选择器中写出了查找节点的 CSS 选择器字符串时，在字符串尾部用“:”分隔，然后跟上一些特殊的限定字符串，用来表示同类节点中的第一个、最后一个、第 n 个、第奇（偶）数个或包含指定文本的节点等。

例 1-2-26：演示伪类选择器的用法，示例代码如下所示。

```
html = '''
<div id="institution">
    <ul>
        <li class="xz"><a href="jwb.html"> 教务部 </a></li>
        <li id="finance" class="xz"><a href="cwb.html"> 财务部 </a></li>
        <li class="jx dept"><a href="fzxy.html"><span class="bold"> 纺织学院 </span></a></li>
        <li class="jx dept"><a href="fzxy.html"> 服装工程学院 </a></li>
        <li class="jx"><a class="jx" href="ggkb.html"> 公共课部 </a></li>
        <li class="xz"><a href="hqb.html"> 后勤部 </a>
    </ul>
</div>
'''
from pyquery import PyQuery as pq
doc = pq(html)
li1 =  doc('li:first-child')        # 注意 ":" 前后都不能有空格
print(' 第一个 li 中的文本：', li1.text())
li2 =  doc('li:last-child')
print(' 最后一个 li 中的文本：', li2.text())
#nth-child(2) 这种用法，数字是从 1 开始编号的，这里 "2" 表示第 2 项 li 节点
li3 =  doc('li:nth-child(2)')
print(' 第二个 li 中的文本：', li3.text())
#gt() 中数字是从 0 开始编号的，这里 "gt(1)" 表示第 2 个 li 之后的所有 li 节点
li4 =  doc('li:gt(1)')
print(' 从第三个 li 开始的 li 中的文本：', li4.text())
li5 =  doc('li:nth-child(2n)')
print(' 第偶数个 li 中的文本：', li5.text())
li6 =  doc('li:nth-child(2n+1)')
print(' 第奇数个 li 中的文本：', li6.text())
li7 =  doc('li:contains(" 学院 ")')
print(' 包含 ' 学院 ' 文字的 li 中的文本：', li7.text())
#lt() 中数字从 0 开始编号，这里 "lt(2)" 表示第 3 个 li 之前的两个 li 节点
```

```
li8 =  doc('li:lt(2)')
print('从第三个 li 之前的 li 中的文本：', li8.text())
```

运行代码，得到的输出结果如下所示。

```
第一个 li 中的文本：教务部
最后一个 li 中的文本：后勤部
第二个 li 中的文本：财务部
第二个 li 之后的所有 li 的文本：纺织学院 服装工程学院 公共课部 后勤部
第偶数个 li 中的文本：财务部 服装工程学院 后勤部
第奇数个 li 中的文本：教务部 纺织学院 公共课部
包含‘学院’文字的 li 中的文本：纺织学院 服装工程学院
第三个 li 之前的两个 li 的文本：教务部 财务部
```

（三）多节点的遍历

如果查找到的结果中包含多个节点，接下来往往需要依次从中选择一个节点进行操作。比如，要获得其中每个节点的子元素或者文本等，那么这时就需要对结果进行遍历。

对 PyQuery 的多个节点的查找结果进行遍历时需要用到 items() 方法，该方法会返回一个生成器（generator），可以用 for 循环遍历一次（只能遍历一次）。

（四）节点信息的提取

在进行数据解析时，查找到节点后，往往需要提取节点的一些信息，如节点的文本内容、节点的某个属性的值等。PyQuery 中也提供了获取节点信息的方法。

1. 获取节点的文本

在 PyQuery 中要获取节点的非属性文本可以使用 text() 方法，例 1-2-25 中的 for 循环中就用“print (li.text())”打印了每个 li 节点的文本。打印出来的结果“教务部、纺织学院、……”实际上是 li 的子节点或孙节点的文本内容，这也是 text() 方法的重要特点。

2. 获取节点的属性

有时候我们需要的数据恰好在某个节点的属性里面，此时就需要获取节点的属性，获取节点的某个属性需要用 attr() 来实现。

例 1-2-27： 演示 attr() 的用法，示例代码如下所示。

```
from pyquery import PyQuery as pq
html='<li class="jx dept"><a href="fzxy.html"><span class="bold">纺织学院 </span></a></li>'
doc = pq(html)
a = doc('a')
print(a.attr('href'))
```

运行代码，得到的输出结果如下所示。

```
fzxy.html
```

这里的“a.attr('href')”就是获取超链接 a 标签的 href 属性，即获取链接地址。

（五）节点的动态操作

在 PyQuery 中提供了对节点的动态操作方法，如为节点添加或移除 class 属性的方法 addClass() 和 removeClass()、从节点中移除子孙节点的方法 remove() 等。这些方法有时对数据解析可以起到很好的作用。

例 1-2-28：演示节点的动态操作，示例代码如下所示。

```
from pyquery import PyQuery as pq
html = '<li class="jx dept"><a href="fzxy.html"> 纺织学院 <span class="bold"><i>（品牌学院）</i></span></a></li>'
doc = pq(html)
a = doc('a')
a.addClass('myclass')
print(' 为 a 添加一个 class 属性 "myclass"：', a)
a.removeClass('myclass')
print(' 将 a 元素 class 属性 "myclass" 删除：', a)
print(' 删除 span 元素前 a 的文本：', a.text())
doc('span.bold').remove()
print(' 删除 span 元素后 a 的文本：',a.text())
```

运行代码，得到的输出结果如下所示。

```
为 a 添加一个 class 属性 "myclass"：<a href="fzxy.html"class="myclass"> 纺织学院 <span class="bold"><i>（品牌学院）</i></span></a>
将 a 元素 class 属性 "myclass" 删除：<a href="fzxy.html"class=""> 纺织学院 <span class="bold"><i>（品牌学院）</i></span></a>
删除 span 元素前 a 的文本：纺织学院（品牌学院）
删除 span 元素后 a 的文本：纺织学院
```

有时候，给某些节点的 class 属性添加某个值后就可以跟相同 class 属性的节点一起处理，这样可以提高效率。移除某些节点的 class 属性也有同样的作用，只是实现的思路是通过去除 class 属性保持多个节点的 class 属性值一致而已。从最后两行的输出结果可以明显看出 remove() 的作用，那就是在某些时候可以先将节点中的子孙节点去除，然后再获取节点的文本，这样就可以避免子孙节点内文本的干扰。

五、任务实践

（1）进入名人引言网站首页，打开开发者面板查看该网站的网页结构，如图 1-2-4 所示。

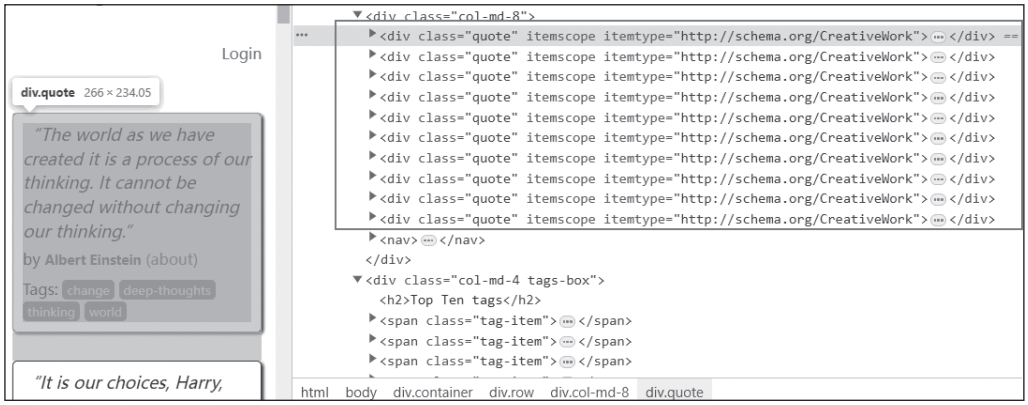


图 1-2-4　名人引言网站的网页结构

每页的 10 条引言对应源码中的 10 个 class 属性为“quote”的 div 元素。

（2）在图 1-2-5 的左边可以看到名人引言信息，单击选择检查工具“ ”（标①处），或按“Ctrl+Shift+C”组合键，然后单击第一条引言的文字内容（标②处），在右边“元素”选项卡窗口会定位到该条引言内容对应的源码，如图 1-2-5 中箭头所指的地方所示。

解析 Quotes 网站数据

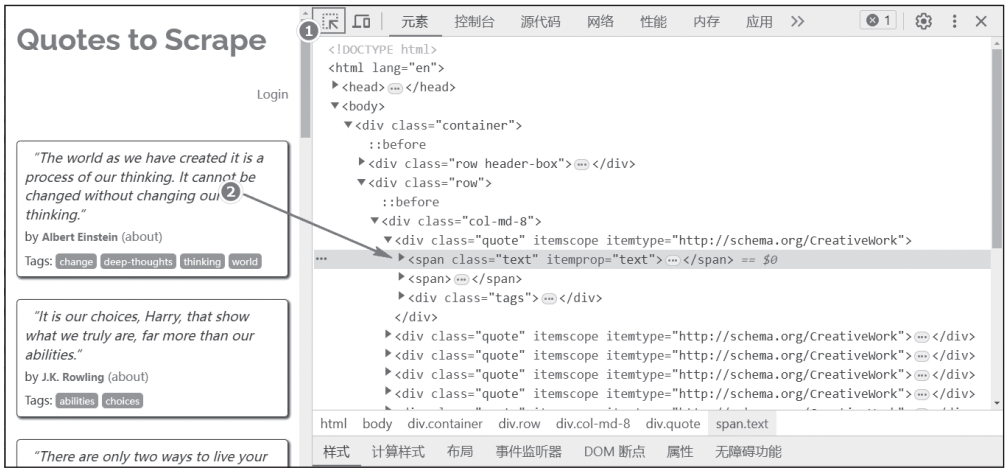


图 1-2-5　引言内容元素的源码

图 1-2-5 中箭头指向的是第一条引言在开发者面板的“元素”选项卡窗口中对应的 HTML，可以看到引言内容是在一个 span（class 属性为“text”）元素中。

（3）用“选中”工具指向引言作者，查看“元素”选项卡窗口中对应的 HTML，发现它是在一个 small 元素中，该元素有一个 class 属性为“author”，如图 1-2-6 所示。

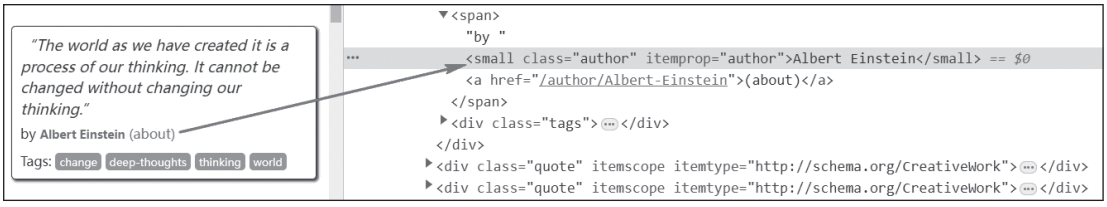


图 1-2-6　引言作者对应的源码

（4）查看引言分类 Tags 的源码，发现它是在一个 div 元素中，该 div 元素有一个 class 属性为“tags”，并且引言分类 Tags 是由多个 a 元素构成的，是 div 的子元素。这些 a 元素的文本即 Tags 显示的内容，这些 a 元素都有 class 为“tag”的属性，如图 1-2-7 所示。

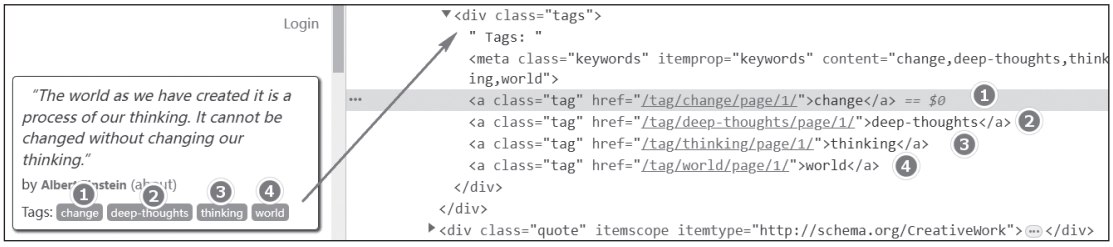


图 1-2-7　引言分类 Tags 对应的源码

（5）将网页垂直滚动条拉到页面底部，查看“Next”按钮的源码，如图 1-2-8 所示。可以看到它是在一个 class 属性为“next”的 li 元素中的超链接 a 元素，链接地址为“/page/2/”，这是首页上的 next 按钮，如果是第 2 页的 next 按钮，其地址是“/page/3/”。显然这些链接地址是不完整的相对地址，应该跟“http://quotes.toscrape.com”拼接起来才能正常工作。

图 1-2-8　Next 翻页按钮对应的源码

（6）基于以上对网页结构的分析，下面我们在前面介绍的 4 种数据解析方式中选择一种来完成代码的编写，这里以 BeautifulSoup 解析方式为例。

①首先定义一个函数 get_one_page(url, headers), 用来根据指定的 URL 请求网页的 HTML 源码，返回值为源码字符串，示例代码如下所示。

```
# 获取指定 URL 页面的源码函数，返回源码，输入参数为 URL 和请求头
import requests
def get_one_page(url,headers):
    try:
        response =  requests.get(url=url, headers=headers)
        if response.status_code == 200:
            return response.text
        else:
            return None
    except Exception as e:
        print(f' 出错了，错误信息为：{e}')
        return None
```

此方法使用 requests 的 get() 方法来获取网页的源代码，并添加了异常处理机制，如果产生异常会返回 None，并打印异常信息。

②定义解析引言数据的函数 parse_one_page(html)。选用 BeautifulSoup 来解析数据，示例代码如下所示。