

第 3 章



Scrapy爬虫

本章学习目标

- 了解 Scrapy 爬虫的概念。
- 掌握 Scrapy 爬虫框架的安装。
- 了解 Scrapy 爬虫的原理与流程。
- 掌握 Scrapy 爬虫框架的实现方式。

本章先向读者介绍 Scrapy 爬虫的概念,再介绍 Scrapy 爬虫的原理与流程,最后介绍 Scrapy 爬虫框架的实现。

3.1 Scrapy 爬虫概述



视频讲解

1. Scrapy 的含义

Scrapy 是使用 Python 语言编写的开源网络爬虫框架,也是一个为了爬取网站数据,提取结构性数据而编写的应用框架。Scrapy 可以应用在数据挖掘、信息处理或存储历史数据等一系列程序中。

Scrapy 简单易用、灵活,并且是跨平台的,在 Linux 及 Windows 平台上都可以使用。Scrapy 框架目前可以支持 Python 2.7 及 Python 3+ 版本,本章主要介绍在 Windows 7 中 Python 3.7 版本下 Scrapy 框架的应用。



视频讲解

2. Scrapy 的安装

在 Windows 7 中安装 Scrapy 框架的命令为 `pip install Scrapy`。注意安装 Scrapy 通

常不会直接成功,因为 Scrapy 框架的安装还需要多个包的支持。

1) 下载 Scrapy 的 twisted 包

Scrapy 依赖 twisted 包,同样使用 whl 格式的包进行安装,进入网站 <http://www.lfd.uci.edu/~gohlke/pythonlibs/>,在网页中搜索 twisted,找到其对应的 whl 包并下载。

2) 下载 Scrapy 的 whl 包

在网上输入下载地址“<http://www.lfd.uci.edu/~gohlke/pythonlibs/>”,进入该网站找到所需要的 Scrapy 的 whl 包。

3) 下载 Scrapy 的 lxml 包

lxml 包是用来做 xpath 提取的,这个包非常容易安装,直接在命令行窗口中输入“pip install lxml”即可。

4) 下载 Scrapy 的 zope.interface 包

zope.interface 包是 Scrapy 爬虫的接口库,可直接在命令行窗口中输入“pip install zope.interface”。

5) 下载 Scrapy 的 pywin32 包

pywin32 是一个第三方模块库,主要作用是方便 Python 开发者快速调用 Windows API 的一个模块库,可直接在命令行中输入“pip install pywin32”。

6) 下载 Scrapy 的 pyOpenSSL 包

pyOpenSSL 是 Python 中的一套基于网络通信的协议,可直接在命令行窗口中输入“pip install pyOpenSSL”。

7) 使用命令安装 Scrapy

在安装 Scrapy 框架之前必须依次安装 twisted 包、whl 包、lxml 包、zope.interface 包、pywin32 包和 pyOpenSSL 包,并在上述包全部安装完成后运行 pip install scrapy 命令来安装 Scrapy 框架。为了确保 Scrapy 框架已经安装成功,可在 Python 中输入“import scrapy”命令,如果出现图 3-1 所示的界面,则表示安装成功。

```
C:\Users\sox>python
Python 3.7.0 <v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51> [MSC v.1914 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import scrapy
>>>
```

图 3-1 导入 Scrapy 库

接着再输入“scrapy.version_info”命令显示 Scrapy 库的版本,运行结果如图 3-2 所示。

```
>>> scrapy.version_info
(1, 5, 1)
```

图 3-2 显示 Scrapy 库的版本

从图 3-2 中可以看出,当前安装的版本是 1.5.1。在安装完成后,即可使用 Scrapy 框架爬取网页中的数据。

3.2 Scrapy 原理



视频讲解

3.2.1 Scrapy 框架的架构

1. Scrapy 架构的组成

Scrapy 框架由 Engine、Scheduler、Downloader、Spiders、Item Pipeline、Downloader Middlewares 及 Spider Middlewares 等几部分组成，具体结构如图 3-3 所示。

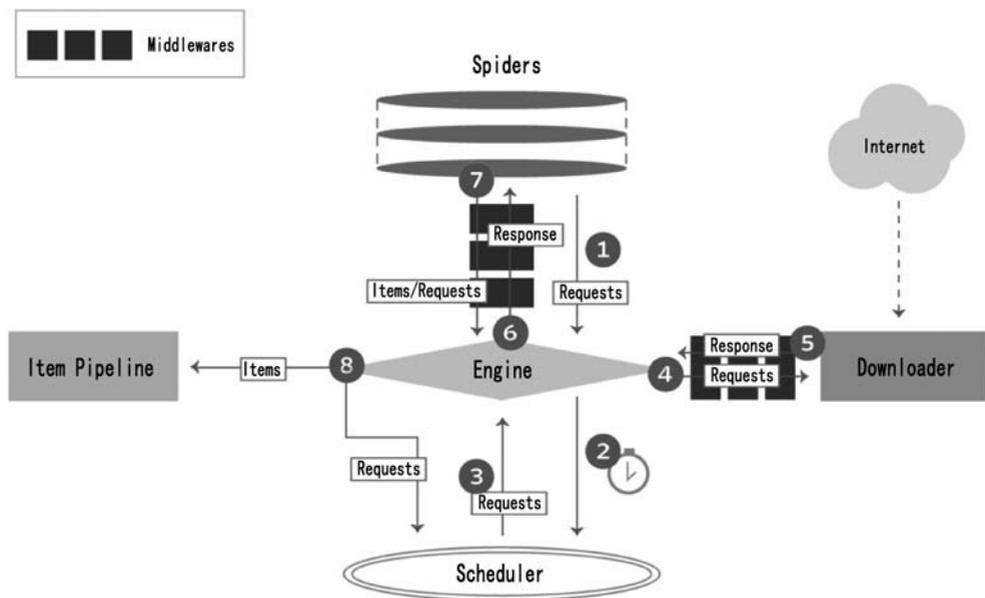


图 3-3 Scrapy 架构的组成

Scrapy 框架中具体组件的作用如下。

1) Engine

Engine 也称为 Scrapy 引擎，它是爬虫工作的核心，负责控制数据流在系统的所有组件中的流动，并在相应动作发生时触发事件。

2) Scheduler

Scheduler 也称为调度器，它从 Engine 接受 Request 并将它们入队，以便之后 Engine 请求它们时提供给 Engine。

3) Downloader

Downloader 也称为下载器，它从 Internet 获取页面数据并提供给 Engine，而后提供给 Spider。

4) Spiders

Spiders 也可称为 Spider，中文一般译作蜘蛛，它是 Scrapy 用户编写用于分析由下载器返回的 Response，并提取出 Item 和额外的 URL 的类，每个 Spider 都能处理一个域名

或一组域名。Spider 的整个抓取流程如下：

(1) 获取第一个 URL 的初始请求,当请求返回后调取一个回调函数。第一个请求是通过调用 `start_requests()` 方法实现的,该方法默认从 `start_urls` 的 URL 中生成请求,并执行解析来调用回调函数。

(2) 在回调函数中解析网页响应并返回项目对象和请求对象或两者的迭代。这些请求也包含一个回调,然后被 Scrapy 下载,由指定的回调处理。

(3) 在回调函数中解析网站的内容,使用 `xpath` 选择器并生成解析的数据项。

(4) 从 Spider 返回的项目通常会进驻到项目管道。

5) Item Pipeline

Item Pipeline 也称为数据管道,它的主要责任是处理由 Spider 从网页中抽取的数据,主要任务是清洗、验证和存储数据。页面被解析后将被发送到数据管道,并经过几个特定的步骤处理数据。每个数据管道的组件都是由一个简单的方法组成的 Python 类。

Item Pipeline 通常的执行过程如下：

(1) 清洗 HTML 数据。

(2) 验证解析到的数据。

(3) 检查数据是否重复。

(4) 将解析到的数据存储到数据库中。

6) Downloader Middlewares

Downloader Middlewares 也称为下载器中间件,它是介于 Scrapy Engine 和 Downloader 之间的中间件,主要用于从 Scrapy Engine 发送到 Downloader 的请求和响应。

7) Spider Middlewares

Spider Middlewares 也称为爬虫中间件,它是介于 Scrapy Engine 和 Spider 之间的框架,主要工作是处理 Spider 的响应输入和请求输出。

在整个框架组成中,Spiders 是最核心的组件,Scrapy 爬虫开发基本上是围绕 Spiders 展开的。

此外,在 Scrapy 框架中还有 3 种数据流对象,分别是 Request、Response 和 Item。

- Request: Scrapy 中的 HTTP 请求对象。
- Response: Scrapy 中的 HTTP 响应对象。
- Item: 一种简单的容器,用于保存爬取得到的数据。

2. Scrapy 框架的工作过程

当 Spider 要爬取某 URL 地址的页面时,首先用该 URL 构造一个 Request 对象,提交给 Engine(图 3-3 中的①),随后 Request 对象进入 Scheduler,按照某种调度算法排队,在之后的某个时候从队列中出来,由 Engine 提交给 Downloader(图 3-3 中的②、③、④)。Downloader 根据 Request 对象中的 URL 地址发送一次 HTTP 请求到目标网站服务器,并接受服务器返回的 HTTP 响应,构建一个 Response 对象(图 3-3 中的⑤),并由 Engine 将 Response 提交给 Spider(图 3-3 中的⑥),Spider 提取 Response 中的数据,构造出 Item

对象或者根据新的连接构造出 Request 对象。分别由 Engine 提交给 Item Pipeline 或者 Scheduler(图 3-3 中的⑦、⑧),这个过程反复进行,直到爬完所有的数据。

3.2.2 Request 对象和 Response 对象

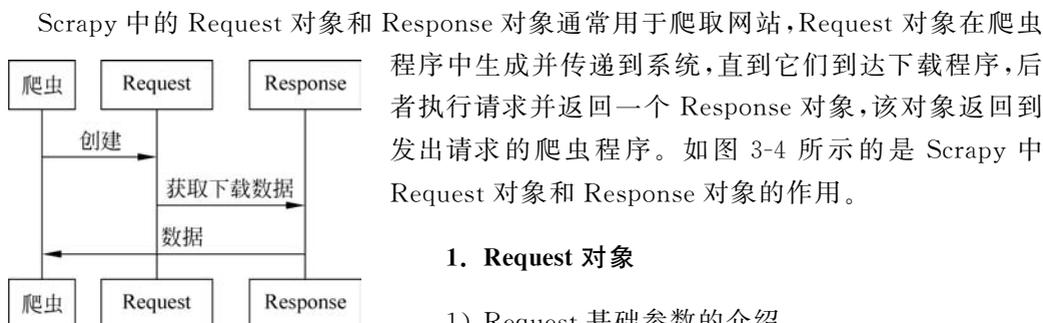


图 3-4 Scrapy 中 Request 对象和 Response 对象的作用

Scrapy 中的 Request 对象和 Response 对象通常用于爬取网站,Request 对象在爬虫程序中生成并传递到系统,直到它们到达下载程序,后者执行请求并返回一个 Response 对象,该对象返回到发出请求的爬虫程序。如图 3-4 所示的是 Scrapy 中 Request 对象和 Response 对象的作用。

1. Request 对象

1) Request 基础参数的介绍

Request 对象用于描述一个 HTTP 请求,由 Spider 产生,Request 构造函数的参数列表如下。

```
Request(url[, callback, method = 'GET', headers, body, cookies, meta, encoding = 'utf-8',
priority=0, dont_filter = False, errback])
```

参数的含义如下。

- url: 请求页面的 URL 地址。
- callback: 请求回来的 Response 处理函数,也称为回调函数。如果请求没有指定回调函数,将使用 spider 的 parse()方法。
- method: HTTP 请求的方法,默认为'GET'。
- headers: 请求的头部字典,dict 类型。dict 值可以是字符串(对于单值标头)或列表(对于多值标头)。如果 None 作为值传递,则不会发送 HTTP 头。
- body: 请求的正文,str 或 unicode 类型。如果 unicode 传递了 a,那么它被编码为 str 使用传递的编码(默认为 UTF-8)。如果 body 没有给出,则存储一个空字符串。不管这个参数的类型是什么,存储的最终值都将是一个 str(不会是 unicode 或 None)。
- cookies: 设置页面的 cookies,dict 类型。当某些网站返回 cookie(在响应中)时,这些 cookie 会存储在该域的 cookie 中,并在将来的请求中再次发送。
- meta: 用于在页面之间传递数据,dict 类型。Request 对象接收一个 meta 参数,一个字典对象,同时 Response 对象有一个 meta 属性可以获取到相应 request 传过来的 meta。
- encoding: 请求的编码,url 和 body 参数的默认编码为 UTF-8。
- priority: 请求的优先级(默认为 0)。调度器使用优先级来定义用于处理请求的顺序,具有较高优先级的请求将较早执行,允许取负值以指示相对低的优先级。
- dont_filter: 表示此请求不应由调度程序过滤,默认为 False。

- `errback`: 如果在处理请求时引发任何异常, 将调用此函数, 包括失败的 404 HTTP 错误等页面。

2) Request 对象方法的介绍

- `copy()`: 复制对象。
- `replace()`: 替换对象。

3) 参数的应用

(1) 将附加数据传递给回调函数。请求的回调函数是当该请求的响应被下载时将被调用的函数。回调函数将使用下载的 Request 对象作为其第一个参数来调用, 代码如下。

```
def parse_page1(self, response):
    return scrapy.Request("http://www.example.com/some_page.html",
                           callback = self.parse_page2)

def parse_page2(self, response):
    self.logger.info("Visited %s", response.url)
```

(2) 使用 `errback` 在请求处理中捕获异常。请求的 `errback` 是在处理异常时被调用的函数, 它接收一个 Twisted Failure 实例作为第一个参数, 并可用于跟踪连接超时、DNS 错误等, 代码如下。

```
class ErrbackSpider(scrapy.Spider):
    name = "errback_example"
    start_urls = [
        "http://www.httpbin.org/",
        "http://www.httpbin.org/status/404",
        "http://www.httpbin.org/status/500",
        "http://www.httpbin.org:12345/",
        "http://www.httphttpbinbin.org/",
    ]
    def errback_httpbin(self, failure):
        self.logger.error(repr(failure))
```

(3) 使用 `FormRequest` 通过 HTTP POST 发送数据。如果想在爬虫中模拟 HTML 表单 POST 并发送几个键值字段, 可以返回一个 `FormRequest` 对象, 代码如下。

```
return [FormRequest(url = "http://www.example.com/post/action", formdata = {'name': 'John
tom', 'age': '37'}, callback = self.after_post)]
```

2. Response 对象

1) Response 基础参数的介绍

Response 对象用于描述一个 HTTP 响应, 由 Downloader 产生, Response 构造函数的参数列表如下。

```
Response(url[, status = 200, headers = None, body = b'', flags = None, request = None])
```

参数的含义如下。

- url: 响应页面的 URL 地址。
- status: 响应的 HTTP 状态,默认为 200。
- headers: 包含响应标题的类字典对象。可以使用 get() 返回具有指定名称的第一个标头值或使用 getlist() 返回具有指定名称的所有标头值来访问值。
- body: HTTP 响应正文。
- flags: 包含此响应的标志的列表。标志是用于标记响应的标签,例如 'cached'、'redirected' 等。
- request: 产生该 HTTP 响应的 request 对象。

2) Response 对象方法的介绍

- copy(): 返回一个新的响应。
- replace(): 返回具有相同成员的 Response 对象,但通过指定的任何关键字参数赋予新值的成员除外。

3) Response 响应子类的介绍

Response 对象是一个基类,根据响应内容有 3 个子类,分别是 TextResponse、HtmlResponse 和 XmlResponse。

(1) TextResponse 子类。TextResponse 支持新的构造函数,是对 Response 对象的补充,TextResponse 方法的参数如下。

```
TextResponse(url[, encoding[, ...]])
```

TextResponse 的主要作用是添加一个新的构造函数 encoding()。encoding(string) 是一个字符串,包含用于此响应的编码。如果创建一个 TextResponse 具有 Unicode 主体的对象,它将使用这个编码进行编码。如果 encoding() 是 None(默认值),则将在响应标头和正文中查找编码。

除此以外,TextResponse 还支持以下属性或对象。

- text: 文本形式的 HTTP 响应正文。
- Selector: 用于在 Response 中提取数据。在使用时先通过 xpath 或者 css 选择器选中页面中要提取的数据,再进行提取。
- xpath(query): 使用 xpath 选择器在 Response 中提取数据。
- css(query): 使用 css 选择器在 Response 中提取数据。
- urljoin(url): 用于构造绝对 URL。

(2) HtmlResponse 子类和 XmlResponse 子类。HtmlResponse 和 XmlResponse 两个类本身只是简单地继承了 TextResponse,因此它们是 TextResponse 的子类。用户通常爬取的网页,其内容大多是 HTML 文本,创建的就是 HtmlResponse 类。HtmlResponse 类有很多方法,但最常见的是 xpath(query)、css(query) 和 urljoin(url)。其中前两个方法用于提取数据,后一个方法用于构造绝对 URL。

3.2.3 Select 对象

1. Select 对象简介

分析 Response 对象的代码。

```
def selector(self):
    from scrapy.selector import Selector
    if self._cached_selector is None:
        self._cached_selector = Selector(self)
    return self._cached_selector
def xpath(self, query, **kwargs):
    return self.selector.xpath(query, **kwargs)
```

从上面的源代码可以看出,Scrapy 的数组组织结构是 Selector,它使用 xpath 选择器在 Response 中提取数据。

从页面中提取数据的核心技术是 HTTP 文本解析,在 Python 中常用的处理模块如下。

- BeautifulSoup: 一个非常流行的解析库,API 简单,但解析的速度慢。
- lxml: 一个使用 C 语言编写的 xml 解析库,解析速度快,API 相对比较复杂。

Scrapy 中的 Selector 对象是基于 lxml 库建立的,并且简化了 API 接口,使用方便。

2. Select 对象的用法

在使用 Selector 对象的时候要先使用 xpath 或者 css 选择器选中页面中要提取的数据,然后进行提取。

1) 创建对象

在 Python 中创建对象有以下两种方式。

(1) 将页面 HTML 文档字符串传递给 Selector 构造器的 text 参数。例如:

```
>>> from scrapy.selector import Selector
>>> text = """
<html>
<body>
  <h1>hello world</h1>
  <h1>hello scrapy</h1>
  <b>hello python</b>
  <ul>
    <li>c++</li>
    <li>java</li>
    <li>python</li>
  </ul>
</body>
</html>"""
```

```
"""
>>> selector = Selector(text = text)
>>> selector
<Selector xpath = None data = '<html>\n\t<body>\n\t\t<h1> hello world </h1>\n\t\t'>
```

(2) 使用一个 Response 对象构造 Selector 对象。例如：

```
>>> from scrapy.selector import Selector
>>> from scrapy.http import HtmlResponse
>>> text = """
<html>
  <body>
    <h1> hello world</h1>
    <h1> hello scrapy</h1>
    <b> hello python</b>
    <ul>
      <li>c++</li>
      <li>java</li>
      <li>python</li>
    </ul>
  </body>
</html>
"""
>>> response = HtmlResponse('url = http://www.example.com', body = text, encoding = "utf-8")
>>> selector = Selector(response = response)
>>> selector
<Selector xpath = None data = '<html>\n\t<body>\n\t\t<h1> hello world </h1>\n\t\t'>
```

在实际开发中，一般不需要手动创建 Selector 对象，在第一次访问一个 Response 对象的 Selector 属性时，Response 对象内部会以自身为参数自动创建 Selector 对象，并将 Selector 对象缓存，以便下次使用。

2) 选中数据

在 Scrapy 中使用选择器是基于 Selector 这个对象的，Selector 对象在 Scrapy 中是通过 xpath() 或 css() 方法来提取数据的。例如：

```
selector_list = selector.xpath('//h1') # 选取文档中所有的 h1
selector_list # 其中包含两个<h1>对应的 Selector 对象
<Selector xpath = './/h1'data = '<h1> Hello World </h1>'
<Selector xpath = './/h1'data = '<h1> Hello Scrapy </h1>'
```

xpath() 和 css() 方法返回一个 SelectorList 对象，包含每个被选中部分对应的 Selector 对象，SelectorList 支持列表接口，可以使用 for 语句迭代访问每一个 Selector 对象，例如：

```
for sel in Selector_list:
    print(sel.xpath('/text()'))
```

```
[<Selector xpath = './text()'data = 'Hello World']
[<Selector xpath = './text()'data = 'Hello Scrapy']
```

SelectorList 对象也有 xpath() 和 css() 方法,调用它们的方法为以接收到的参数分别调用其中一个 Selector 对象的 xpath、css,将所有搜集到的一个新的 SelectorList 对象返回给用户,例如:

```
selector_list.xpath('./text()')
<selector.xpath = './text()'data = 'Hello World'>
<selector.xpath = './text()'data = 'Hello Scrapy'>
selector.xpath('./path').css('li').xpath('./text()')
[<Selector xpath = './text()'data = 'C++>]
[<Selector xpath = './text()'data = 'java'>]
[<Selector xpath = './text()'data = 'python'>]
```

在具体实现中,Scrapy 使用 css 和 xpath 选择器来定位元素,它的基本方法如下。

- xpath(): 返回选择器列表,每个选择器代表使用 xpath 语法选择的节点。
- css(): 返回选择器列表,每个选择器代表使用 css 语法选择的节点。

(1) xpath。xpath 是 XML 路径语言,它是一种用来确定 XML 文档中某部分位置的语言。表 3-1 列举了常见的 xpath 路径表达式。

表 3-1 xpath 路径表达式

| 表 达 式 | 描 述 |
|----------|------------------------------|
| nodename | 选取次节点的所有子节点 |
| / | 从根节点选取 |
| // | 从匹配选择的当前节点选择文档中的节点,而不考虑它们的位置 |
| . | 选取当前节点 |
| .. | 选取当前节点的父节点 |
| @ | 选取属性 |

此外,在 xpath 中可以使用谓语句来查找某个特定的节点或者包含某个指定值的节点,谓语句被嵌在方括号中;可以对任意节点使用谓语句,并输出结果。表 3-2 给出了常见的谓语句表达式及输出结果。

表 3-2 常见的谓语句表达式及输出结果

| 表 达 式 | 输 出 结 果 |
|----------------------------------|---|
| / students/student[1] | 选取属于 students 元素的第一个 student 元素 |
| / students/student[last()] | 选取属于 students 元素的最后一个 student 元素 |
| / students/student[last()-1] | 选取属于 students 元素的倒数第二个 student 元素 |
| / students/student[position()<2] | 选取最前面的一个属于 students 元素的 student 元素 |
| // student[@id] | 选取所有拥有名为 id 属性的 student 元素 |
| // student[@id='00111'] | 选取所有 student 元素,且这些元素拥有值为 00111 的 id 属性 |

例如：

```
response.xpath('/html/body/div')      # 选取 body 下的所有 div
response.xpath('//a')                 # 选中文档中的所有 a
response.xpath('/html/body//div')     # 选中 body 下的所有节点中的 div, 无论它在什么位置
response.xpath('//a/text()')         # 选取所有 a 的文本
response.xpath('/html/div/*')        # 选取 div 的所有元素子节点
```

【例 3-1】 使用 Python 3 中的 lxml 库, 利用 xpath 对 HTML 进行解析。

① 安装 lxml 库, 命令如下：

```
pip install lxml
```

② 导入 lxml 库的 etree 模块, 命令如下：

```
from lxml import etree
```

接着声明一段 HTML 文本, 调用 HTML 类进行初始化, 成功构造一个 XPath 解析对象。该例使用了 lxml 的 etree 库, 然后利用 etree.HTML 初始化将文件打印出来, 如图 3-5 所示。

```
>>> from lxml import etree
>>> text='
... <div>
... <ul>
... <li>first</li>
... <li>second</li>
... <li>third</li>
... <li>fourth</li>
... </ul>
... </div>
...
>>> html=etree.HTML(text)
>>> result=etree.tostring(html)
>>> print(result)
b'<html><body><div>\n<ul>\n<li>first</li>\n<li>second</li>\n<li>third</li>\n<li>fourth</li>\n</ul>\n</div>\n</body></html>'
```

图 3-5 构造一个 XPath 解析对象

③ 查看 result 的类型, 如图 3-6 所示。

```
>>> print(type(result))
<class 'list'>
>>> print(type(result[0]))
<class 'lxml.etree.Element'>
>>>
```

图 3-6 查看 result 的类型

④ 使用 xpath 输出所有 li 节点, 使用 xpath 输出最后一个节点的内容、倒数第二个节点的内容以及第一个节点的内容, 如图 3-7 所示。

(2) css。css 即层叠样式表, 它的语法简单, 功能不如 xpath 强大。当调用 Selector 对象的 css 方法时, 内部 Python 库 cssSelect 将 css 转化为 xpath 再执行操作。表 3-3 列出了 css 的一些基本语法。

```

>>> result=html.xpath('//li')
>>> print(result)
[<Element li at 0x17bea94dac0>, <Element li at 0x17bea94db40>, <Element li at 0x17bea94dc00>, <Element li at 0x17bea94dc40>]
>>> result=html.xpath('//li[last()]')
>>> print(result[0].text)
fourth
>>> result=html.xpath('//li[last()-1]')
>>> print(result[0].text)
third
>>> result=html.xpath('//li[1]')
>>> print(result[0].text)
first
>>>

```

图 3-7 输出对应的节点内容

表 3-3 css 的基本语法

| 表 达 式 | 描 述 |
|-----------------------|-----------------------------|
| * | 选取所有元素 |
| E | 选取 E 元素 |
| E1,E2 | 选取 E1、E2 元素 |
| E1 E2 | 选取 E1 后代元素中的 E2 元素 |
| E1>E2 | 选取 E1 子元素中的 E2 元素 |
| E1+E2 | 选取兄弟中的 E2 元素 |
| . container | 选取 class 属性为 container 的元素 |
| # container | 选取 id 属性为 container 的元素 |
| [attr] | 选取包含 attr 属性的元素 |
| [attr=value] | 选取包含 attr 属性且值为 value 的元素 |
| E:nth-(last-)child(n) | 选取 E 元素,且该元素必须是其父元素的第 n 个元素 |
| E:empty | 选取没有子元素的 E 元素 |
| E::text | 选取 E 元素的文本节点(text node) |

例如:

```

response.css('div a::text').extract()           # 所有 div 下的所有 a 的文本
response.css('div a::attr(href)').extract()     # href 的值
response.css('div>a:nth-child(1)')             # 选中每个 div 的第一个 a 节点,会设定
                                                # 只在子节点中找,不会到孙节点中找
response.css('div:not(#container)')            # 选取所有 id 不是 container 的 div
response.css('div:first-child>a:last-child')   # 第一个 div 中的最后一个 a

```

3.2.4 Spider 开发流程

对于大多数用户来讲,Spider 是 Scrapy 框架中最核心的组件,Scrapy 爬虫在开发时通常是紧紧围绕 Spider 展开的。一般而言,实现一个 Spider 需要经过以下步骤。

- (1) 继承 scrapy.Spider。
- (2) 为 Spider 命名。
- (3) 设置爬虫的起始爬取点。
- (4) 实现页面的解析。

1. 继承 scrapy.Spider

Scrapy 框架提供了一个 Spider 基类，用户编写的 Spider 都需要继承它，代码如下。

```
import scrapy
class MeijuSpider(scrapy.Spider)
```

scrapy.Spider 这个基类实现了以下功能。

- (1) 提供了 Scrapy 引擎调用的接口。
- (2) 提供了用户使用的工具函数。
- (3) 提供了用户访问的属性。

2. 为 Spider 命名

在 Spider 中使用属性 name 为爬虫命名。该名称在项目中必须是独一无二的，不能和其他爬虫的名称相同。一般是以该网站(domain)来命名 Spider。例如用 Spider 爬取 mywebsite.com, 该 Spider 通常会被命名为 mywebsite。如果没有给爬虫命名，爬虫会在初始化时抛出 ValueError。

3. 设置起始爬取点

start_urls 定义 Spider 开始爬取数据的 URL 列表，随后生成的其他需要爬取的 URL 都是从这些 URL 对应的页面中提取数据生成出来的。例如：

```
start_urls = ['http://www.meijutt.com/new100.html']
```

实际上，对于起始爬取点的下载请求是由 Scrapy 引擎调用 Spider 对象的 start_requests() 提交的。这个方法必须返回该 Spider 的可迭代的初始 requests 对象供后续爬取。Scrapy 会在该 Spider 开始爬取数据时被调用，且只会被调用一次，因此可以很安全地将 start_requests() 作为一个生成器来执行。默认的执行会生成每一个在 start_urls 中的 URL 对应的 Request 对象。

4. 实现页面的解析

parse() 方法是 Scrapy 默认的解析函数，用于回调处理下载的 response，并且当 response 没有指定回调函数时，该方法是 Scrapy 处理 response 的默认方法。

parse() 方法比较简单，只是对 response 调用 _parse_response() 方法，并设置 callback 为“parse_start_url, follow=True”（表明跟进链接）。如果设置了 callback，也就是 parse_start_url，会优先调用 callback 处理，然后调用 process_results() 方法来生成返回列表。例如：

```
def parse(self, response):
    return self._parse_response(response, self.parse_start_url, cb_kwargs = {}, follow = True)
```

Spider 中的常见属性和方法如表 3-4 所示。

表 3-4 Spider 中的常见属性和方法

| 常见属性和方法 | 含 义 |
|-----------------------|--|
| name | 定义 Spider 名称的字符串 |
| allowed_domains | 包含了 Spider 允许爬取的域名(domain)的列表,可选 |
| start_urls | 初始 URL 元组/列表。当没有指定特定的 URL 时,Spider 将从该列表中开始进行爬取 |
| custom_settings | 定义该 Spider 配置的字典,这个配置会在项目范围内运行这个 Spider 的时候生效 |
| crawler | 定义 Spider 实例绑定的 crawler 对象,这个属性是在初始化 Spider 类时由 from_crawler()方法设置的,crawler 对象概括了许多项目的组件 |
| settings | 运行 Spider 的配置,这是一个 settings 对象的实例 |
| logger | 用 Spider 名称创建的 Python 记录器,可以用来发送日志消息 |
| start_requests(self) | 该方法包含了 Spider 用于爬取(默认实现是使用 start_urls 的 URL)的第一个 Request |
| parse(self, response) | 当请求 URL 返回网页没有指定回调函数时,默认的 Request 对象回调函数,用来处理网页返回的 response,以及生成 Item 或者 Request 对象 |

3.3 Scrapy 的开发与实现

3.3.1 Scrapy 爬虫的开发流程

要开发 Scrapy 爬虫,一般有以下几步。

- (1) 新建项目。
- (2) 确定抓取网页目标。
- (3) 制作爬虫。
- (4) 设计管道存储爬取内容。

Scrapy 爬虫的实现步骤如图 3-8 所示。

新建项目(Project): 新建一个爬虫项目

明确目标(Items): 明确想要抓取的目标

制作爬虫(Spider): 制作爬虫开始爬取网页

存储内容(Pipeline): 设计管道存储爬取内容

图 3-8 Scrapy 爬虫的实现步骤



视频讲解

3.3.2 创建 Scrapy 项目并查看结构

1. Scrapy 常用命令介绍

(1) startproject。startproject 命令表示创建一个新工程,所有的爬虫程序都需要先创建新工程,语法为:

```
scrapy startproject < name >
```

其中 name 表示新工程的名称。

(2) genspider。genspider 命令表示在该工程下创建一个爬虫,语法为:

```
scrapy genspider < name > < domain >
```

其中 name 表示爬虫的名称, domain 表示要爬取的网站的域名。

(3) settings。settings 命令表示获取爬虫配置信息,语法为:

```
scrapy settings
```

(4) crawl。crawl 表示运行已经创建好的爬虫,语法为:

```
scrapy crawl < spider >
```

其中 spider 表示创建好的爬虫的名称。

(5) list。list 命令表示列出工程中的所有爬虫,语法为:

```
scrapy list
```

2. 创建 Scrapy 项目

【例 3-2】 创建一个最简单的 Spider 爬虫。

该例以爬取 Spider 爬虫专门的训练网站(<http://books.toscrape.com/>)为例,讲述 Scrapy 爬虫的创建和运行。该网站首页如图 3-9 所示。

1) 创建工程

在本地选中某一文件夹,同时按住 Shift 键右击,在弹出的对话框中选择“在此处打开命令窗口”,输入以下命令:

```
scrapy startproject movie
```

创建 Scrapy 工程,如图 3-10 所示。

2) 创建爬虫程序

在创建好了 Scrapy 工程以后,下一步就是创建爬虫程序。输入以下命令:

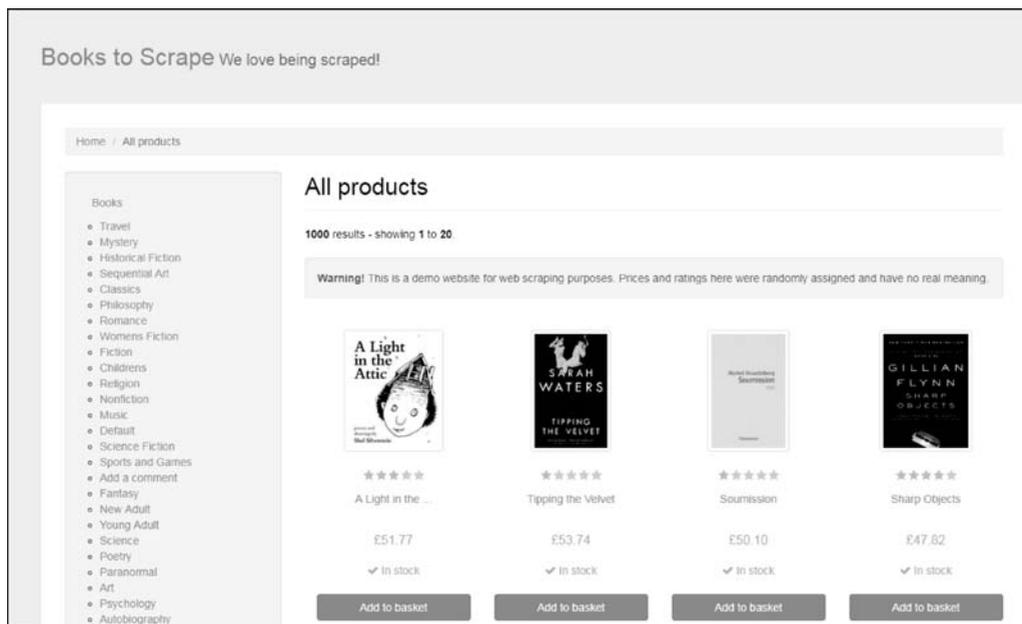


图 3-9 网站首页

```
C:\Users\xxx\Desktop\爬虫>scrapy startproject movie
New Scrapy project 'movie', using template directory 'd:\Users\xxx\AppData\Local\programs\python\python37\lib\site-packages\scrapy\templates\project', created in:
  C:\Users\xxx\Desktop\爬虫\movie
You can start your first spider with:
  cd movie
  scrapy genspider example example.com
```

图 3-10 创建 Scrapy 工程

```
scrapy genspider meiju meijutt.com
```

该命令创建 Spider 爬虫,并命名为 meiju,如图 3-11 所示。

```
C:\Users\xxx\Desktop\爬虫>cd movie
C:\Users\xxx\Desktop\爬虫\movie>scrapy genspider meiju meijutt.com
Created spider 'meiju' using template 'basic' in module:
  movie.spiders.meiju
```

图 3-11 创建爬虫 Spider

3. 查看并认识爬虫目录结构

使用 tree 命令查看目录结构,如图 3-12 所示。

目录结构含义如下。

- scrapy.cfg: 部署 Scrapy 爬虫的配置文件。
- movie/: 外层目录。
- items.py: Items 代码模板(继承类)。

```

scrapy.cfg
|
├── movie
│   ├── items.py
│   ├── middlewares.py
│   ├── pipelines.py
│   ├── settings.py
│   ├── settings.pyc
│   ├── __init__.py
│   └── __init__.pyc
├── spiders
│   ├── meiju.py
│   ├── __init__.py
│   └── __init__.pyc

```

图 3-12 目录结构

- middlewares.py: middlewares 代码模板(继承类)。
- pipelines.py: Pipelines 代码模板(继承类)。
- settings.py: Scrapy 爬虫的配置文件。
- settings.pyc: Scrapy 爬虫的配置文件,由 Python 生成,同 settings.py。
- __init__.py: 初始文件,无须修改。
- __init__.pyc: 初始化脚本,由 Python 生成,同 __init__.py。
- spiders/: Spiders 代码模板目录(继承类)。
- __init__.py: 初始化脚本。
- __init__.pyc: 初始化脚本,同 __init__.py。

创建好以后,可在文件夹中查看到如图 3-13~图 3-15 所示的目录。

| 名称 | 修改日期 | 类型 | 大小 |
|------------|-----------------|--------|------|
| movie | 2019/1/26 11:28 | 文件夹 | |
| scrapy.cfg | 2018/12/7 17:37 | CFG 文件 | 1 KB |

图 3-13 根目录结构

| 名称 | 修改日期 | 类型 | 大小 |
|-------------|-----------------|---------------------|------|
| __pycache__ | 2019/1/26 11:28 | 文件夹 | |
| spiders | 2018/12/7 17:38 | 文件夹 | |
| __init__ | 2018/9/4 22:20 | JetBrains PyChar... | 0 KB |
| items | 2018/12/7 17:39 | JetBrains PyChar... | 1 KB |
| middlewares | 2018/12/7 17:37 | JetBrains PyChar... | 4 KB |
| my_meiju | 2019/1/26 11:28 | 文本文档 | 0 KB |
| pipelines | 2018/12/7 17:41 | JetBrains PyChar... | 1 KB |
| settings | 2018/12/7 17:40 | JetBrains PyChar... | 4 KB |

图 3-14 movie 目录结构

| 名称 | 修改日期 | 类型 | 大小 |
|-------------|-----------------|---------------------|------|
| __pycache__ | 2018/12/7 17:41 | 文件夹 | |
| __init__ | 2018/9/4 22:20 | JetBrains PyChar... | 1 KB |
| meiju | 2018/12/7 17:40 | JetBrains PyChar... | 1 KB |

图 3-15 spiders 目录结构

3.3.3 编写代码并运行爬虫

1. 设置 Spider 爬虫

运行 Python, 在 meiju.py 中输入以下代码:

```
import scrapy
from movie.items import MovieItem
class MeijuSpider(scrapy.Spider):
    name = "meiju"
    allowed_domains = ["books.com"]
    start_urls = ['http://books.toscrape.com/catalogue/category/books/travel_2/index.html']
    def parse(self, response):
        movies = response.xpath('//ol[@class="row"]/li')
        for each_movie in movies:
            item = MovieItem()
            item['name'] = each_movie.xpath('article/h3/a/@title').extract()[0]
            yield item
```

在 Scrapy 中使用 xpath 提取网页中的路径, 如图 3-16 所示。

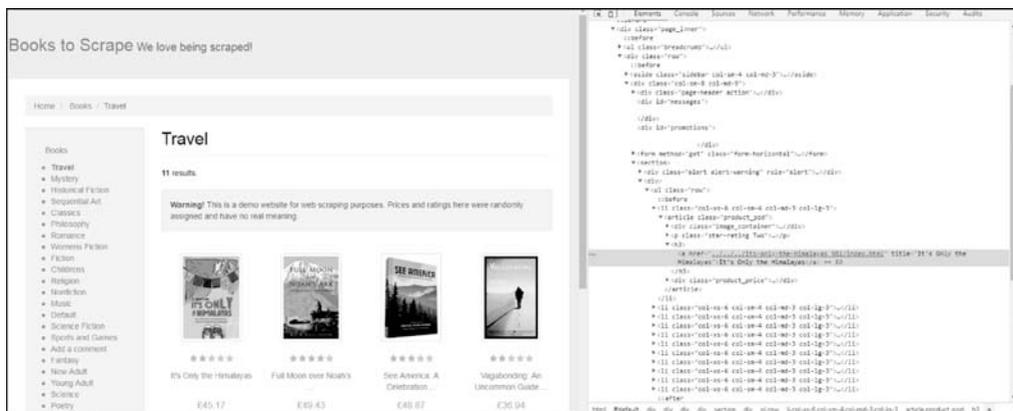


图 3-16 使用 xpath 提取网页中的路径

2. 设置 item 模板

在 items.py 中输入以下代码：

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

3. 设置配置文件

在 settings.py 中增加以下代码：

```
ITEM_PIPELINES = {'movie.pipelines.MoviePipeline':300}
```

4. 设置数据处理脚本

在 pipelines.py 中输入以下代码：

```
import json
class MoviePipeline(object):
    def process_item(self, item, spider):
        return item
```

5. 运行爬虫

在爬虫根目录中执行以下命令：

```
scrapy crawl meiju
```

其中,meiju 表示该爬虫 Spider 的名称,运行结果如图 3-17 所示。

从图 3-17 可以看出,该例要爬取的页面路径为 Home/Books/Travel,并通过 Spider 爬取图书的标题名称,例如 title="It's Only the Himalayas"等,如图 3-18 所示。

要爬取的网页 URL 如下：

```
start_urls = ['http://books.toscrape.com/catalogue/category/books/travel_2/index.html']
```

定位网页中的具体爬取位置的相关代码如下所示：

```
movies = response.xpath('//ol[@class="row"]/li')
```

```

2022-06-20 08:35:12 [scrapy.middleware] INFO: Enabled item pipelines:
['movie.pipelines.MoviePipeline']
2022-06-20 08:35:12 [scrapy.core.engine] INFO: Spider opened
2022-06-20 08:35:12 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
2022-06-20 08:35:12 [scrapy.extensions.telnet] DEBUG: Telnet console listening on 127.0.0.1:6023
2022-06-20 08:35:13 [scrapy.core.engine] DEBUG: Crawled (404) <GET http://books.toscrape.com/robots.txt> (referer: None)
2022-06-20 08:35:13 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://books.toscrape.com/catalogue/category/books/travel_2/index.html> (referer: None)
2022-06-20 08:35:13 [scrapy.core.scraper] DEBUG: Scraped from <200 http://books.toscrape.com/catalogue/category/books/travel_2/index.html>
<name>: "It's Only the Himalayas"
2022-06-20 08:35:13 [scrapy.core.scraper] DEBUG: Scraped from <200 http://books.toscrape.com/catalogue/category/books/travel_2/index.html>
<name>: "Full Moon over Noah's Ark: An Odyssey to Mount Ararat and Beyond"
2022-06-20 08:35:13 [scrapy.core.scraper] DEBUG: Scraped from <200 http://books.toscrape.com/catalogue/category/books/travel_2/index.html>
<name>: "See America: A Celebration of Our National Parks & Treasured Sites"
2022-06-20 08:35:13 [scrapy.core.scraper] DEBUG: Scraped from <200 http://books.toscrape.com/catalogue/category/books/travel_2/index.html>
<name>: "Vagabonding: An Uncommon Guide to the Art of Long-Term World Travel"
2022-06-20 08:35:13 [scrapy.core.scraper] DEBUG: Scraped from <200 http://books.toscrape.com/catalogue/category/books/travel_2/index.html>
<name>: "Under the Tuscan Sun"
2022-06-20 08:35:13 [scrapy.core.scraper] DEBUG: Scraped from <200 http://books.toscrape.com/catalogue/category/books/travel_2/index.html>
<name>: "A Summer in Europe"
2022-06-20 08:35:13 [scrapy.core.scraper] DEBUG: Scraped from <200 http://books.toscrape.com/catalogue/category/books/travel_2/index.html>
<name>: "The Great Railway Bazaar"
2022-06-20 08:35:13 [scrapy.core.scraper] DEBUG: Scraped from <200 http://books.toscrape.com/catalogue/category/books/travel_2/index.html>
<name>: "A Year in Provence (Provence #1)"
2022-06-20 08:35:13 [scrapy.core.scraper] DEBUG: Scraped from <200 http://books.toscrape.com/catalogue/category/books/travel_2/index.html>
<name>: "The Road to Little Bribling: Adventures of an American in Britain"
2022-06-20 08:35:13 [scrapy.core.scraper] DEBUG: Scraped from <200 http://books.toscrape.com/catalogue/category/books/travel_2/index.html>
<name>: "Neither Here nor There: Travels in Europe"
2022-06-20 08:35:13 [scrapy.core.scraper] DEBUG: Scraped from <200 http://books.toscrape.com/catalogue/category/books/travel_2/index.html>
<name>: "1,000 Places to See Before You Die"

```

图 3-17 运行爬虫结果

Home / Books / Travel

Travel

11 results

Warning! This is a demo website for web scraping purposes. Prices and ratings here were randomly assigned and have no real meaning.

| | | | |
|---|---|--|--|
| ★★★★★ It's Only the Himalayas £45.17 ✓ In stock Add to basket | ★★★★★ Full Moon over Noah's ... £49.43 ✓ In stock Add to basket | ★★★★★ See America: A Celebration ... £48.87 ✓ In stock Add to basket | ★★★★★ Vagabonding: An Uncommon Guide ... £36.94 ✓ In stock Add to basket |
| ★★★★★ Under the Tuscan Sun | ★★★★★ A Summer in Europe | ★★★★★ The Great Railway Bazaar | ★★★★★ A Year in Provence ... |

图 3-18 爬取每个标题名称

爬取图书标题的相关代码如下所示：

```
item['name'] = each_movie.xpath('article/h3/a/@title').extract()[0]
```

如果要让 Scrapy 爬虫爬取网页的结果保存到文本文件中，可打开 pipelinespy，写入以下代码：

```
import json
class MoviePipeline(object):
    def open_spider(self, spider):
        self.file = open('log.txt', 'w', encoding = 'utf-8')
    def close_spider(self, spider):
        self.file.close()
    def process_item(self, item, spider):
        self.file.write(str(item) + '\n')
```

保存该文件，再运行该爬虫程序，即可将数据写入 log.txt 中，如图 3-19 所示。

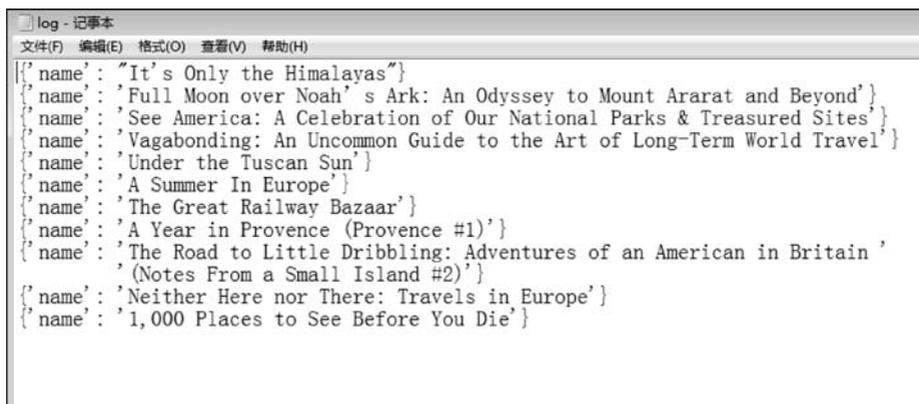


图 3-19 将数据写入 log.txt 中

3.4 本章小结

(1) Scrapy 是使用 Python 语言编写的开源网络爬虫框架，也是一个为了爬取网站数据，提取结构性数据而编写的应用框架。Scrapy 可以应用在包括数据挖掘、信息处理或存储历史数据等一系列的程序中。

(2) Scrapy 框架由 Scrapy Engine、Scheduler、Downloader、Spiders、Item Pipeline、Downloader middlewares 以及 Spider middlewares 等几部分组成。

(3) Scrapy 中的 Request 对象和 Response 对象通常用于爬取网站。Request 对象用来描述一个 HTTP 请求，它常由爬虫生成；而 Response 对象一般是由 Scrapy 自动构建，Response 对象有很多属性，可以用来提取网页中的数据。

(4) Spider 是 Scrapy 框架中最核心的组件，Scrapy 爬虫在开发时通常是紧紧围绕

Spider 展开的。

(5) 要开发 Scrapy 爬虫,一般步骤为新建项目、确定抓取网页目标、制作爬虫、设计管道存储爬取内容。

3.5 实训

1. 实训目的

通过本章实训了解 Scrapy 爬虫框架的特点,能使用 Scrapy 框架进行简单的网站数据爬取。

2. 实训内容

使用 Scrapy 框架编写爬虫访问网站。

(1) 访问专门的爬虫网站 <http://quotes.toscrape.com>,该网站有多个页面,例如 <http://quotes.toscrape.com/page/2/>、<http://quotes.toscrape.com/page/3/>等,首页页面如图 3-20 所示。

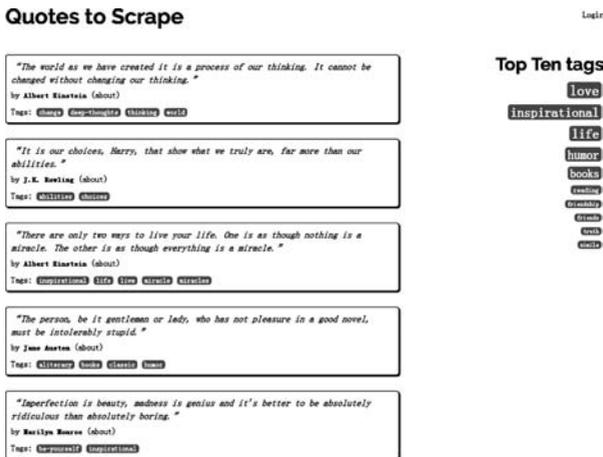


图 3-20 爬虫网站首页

(2) 爬取该页面中的每一个子区域对应的文本内容、作者和分类,如图 3-21 所示。

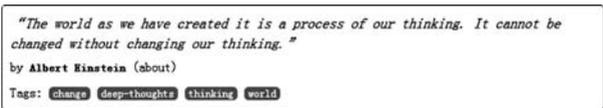


图 3-21 爬取网站相应内容

(3) 新建爬虫工程,命名为 quotesbot-master,并在此工程中创建 Spider 爬虫,命令为 toscrape-xpath。

(4) 打开 toscrape-xpath.py,输入以下代码。

```
import scrapy
class ToScrapeSpiderXPath(scrapy.Spider):
    name = 'toscrape-xpath'
    start_urls = [
        'http://quotes.toscrape.com/',
    ]
    def parse(self, response):
        for quote in response.xpath('//div[@class="quote"]'):
            yield {
                'text': quote.xpath('.//span[@class="text"]/text()').extract_first(),
                'author': quote.xpath('.//small[@class="author"]/text()').extract_first(),
                'tags': quote.xpath('.//div[@class="tags"]/a[@class="tag"]/text()')
                .extract()
            }
            next_page_url = response.xpath('//li[@class="next"]/a/@href').extract_first()
            if next_page_url is not None:
                yield scrapy.Request(response.urljoin(next_page_url))
```

其中,语句“text': quote.xpath('.//span[@class="text"]/text()').extract_first()”表示爬取页面子区域的文本内容;语句“author': quote.xpath('.//small[@class="author"]/text()').extract_first()”表示爬取页面子区域的作者;语句“tags': quote.xpath('.//div[@class="tags"]/a[@class="tag"]/text()').extract()”表示爬取页面子区域的分类。

语句“next_page_url = response.xpath('//li[@class="next"]/a/@href').extract_first()”表示依次爬取该网站的下一页。

(5) 在 pipelines.py 中输入以下代码。

```
class QuotesbotPipeline(object):
    def process_item(self, item, spider):
        return item
```

(6) 在 settings.py 中输入以下代码。

```
BOT_NAME = 'quotesbot'
SPIDER_MODULES = ['quotesbot.spiders']
NEWSPIDER_MODULE = 'quotesbot.spiders'
```

(7) 在 items.py 中输入以下代码。

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

(8) 运行该爬虫,在爬虫根目录中执行命令 scrapy crawl toscrape-xpath,可以得到爬取结果,如图 3-22 和图 3-23 所示。

```
管理员: C:\Windows\system32\cmd.exe
<'text': '"A wise girl kisses but doesn't love, listens but doesn't believe, an
d leaves before she is left.'"', 'author': 'Marilyn Monroe', 'tags': ['attribute
d-no-source']>
2020-12-28 11:50:29 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://quotes
.toscrape.com/page/5/>
<'text': '"Only in the darkness can you see the stars.'"', 'author': 'Martin Lu
ther King Jr.', 'tags': ['hope', 'inspirational']>
2020-12-28 11:50:29 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://quotes
.toscrape.com/page/5/>
<'text': '"It matters not what someone is born, but what they grow to be.'"', '
author': 'J.K. Rowling', 'tags': ['dumbledore']>
2020-12-28 11:50:29 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://quotes
.toscrape.com/page/5/>
<'text': '"Love does not begin and end the way we seem to think it does. Love i
s a battle, love is a war; love is a growing up.'"', 'author': 'James Baldwin',
'tags': ['love']>
2020-12-28 11:50:29 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://quotes
.toscrape.com/page/6/> (referer: http://quotes.toscrape.com/page/5/)
2020-12-28 11:50:30 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://quotes
.toscrape.com/page/6/>
<'text': '"There is nothing I would not do for those who are really my friends.
I have no notion of loving people by halves, it is not my nature.'"', 'author':
'Jane Austen', 'tags': ['friendship', 'love']>
2020-12-28 11:50:30 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://quotes
.toscrape.com/page/6/>
<'text': '"Be one thing every day that scares you.'"', 'author': 'Eleanor Roose
velt', 'tags': ['attributed', 'fear', 'inspiration']>
2020-12-28 11:50:30 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://quotes
.toscrape.com/page/6/>
<'text': '"I am good, but not an angel, I do sin, but I am not the devil, I am
just a small girl in a big world trying to find someone to love.'"', 'author':
'Marilyn Monroe', 'tags': ['attributed-no-source']>
2020-12-28 11:50:30 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://quotes
.toscrape.com/page/6/>
<'text': '"If I were not a physicist, I would probably be a musician. I often t
hink in music. I love my daydreams in music. I see my life in terms of music.'"
', 'author': 'Albert Einstein', 'tags': ['music']>
2020-12-28 11:50:30 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://quotes
.toscrape.com/page/6/>
<'text': '"If you only read the books that everyone else is reading, you can on
ly think what everyone else is thinking.'"', 'author': 'Haruki Murakami', 'tags
': ['books', 'thought']>
2020-12-28 11:50:30 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://quotes
```

图 3-22 运行该爬虫

```
管理员: C:\Windows\system32\cmd.exe
<'text': '"You have to write the book that wants to be written. And if the book
will be too difficult for grown-ups, then you write it for children.'"', 'autho
r': 'Madeleine L'Engle', 'tags': ['books', 'children', 'difficult', 'grown-ups',
'write', 'writers', 'writing']>
2020-12-28 11:50:31 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://quotes
.toscrape.com/page/10/>
<'text': '"Never tell the truth to people who are not worthy of it.'"', 'author
': 'Mark Twain', 'tags': ['truth']>
2020-12-28 11:50:31 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://quotes
.toscrape.com/page/10/>
<'text': '"A person's a person, no matter how small.'"', 'author': 'Dr. Seuss',
'tags': ['inspirational']>
2020-12-28 11:50:31 [scrapy.core.scrapers] DEBUG: Scraped from <200 http://quotes
.toscrape.com/page/10/>
<'text': '"... a mind needs books as a sword needs a whetstone, if it is to kee
p its edge.'"', 'author': 'George R.R. Martin', 'tags': ['books', 'mind']>
2020-12-28 11:50:31 [scrapy.core.engine] INFO: Closing spider (finished)
2020-12-28 11:50:31 [scrapy.statscollectors] INFO: Dumping Scrapy stats:
<downloader/request_bytes': 2881,
'downloader/request_count': 11,
'downloader/request_method_count/GET': 11,
'downloader/response_bytes': 23411,
'downloader/response_count': 11,
'downloader/response_status_count/200': 10,
'downloader/response_status_count/404': 1,
'elapsed_time_seconds': 3.997608,
'finish_reason': 'finished',
'finish_time': datetime.datetime(2020, 12, 28, 3, 50, 31, 445724),
'item_scraped_count': 100,
'log_count/DEBUG': 111,
'log_count/INFO': 10,
'request_depth_max': 9,
'response_received_count': 11,
'robotstxt/request_count': 1,
'robotstxt/response_count': 1,
'robotstxt/response_status_count/404': 1,
'scheduler/dequeued': 10,
'scheduler/dequeued/memory': 10,
'scheduler/enqueued': 10,
'scheduler/enqueued/memory': 10,
'start_time': datetime.datetime(2020, 12, 28, 3, 50, 27, 448116)>
2020-12-28 11:50:31 [scrapy.core.engine] INFO: Spider closed (finished)
```

图 3-23 爬取的页面内容

(9) 查看爬取的每一个具体内容,如图 3-24 所示。

```
{'text': '“Only in the darkness can you see the stars.”', 'author': 'Martin Luther King Jr.', 'tags': ['hope', 'inspirational']}
```

图 3-24 爬取的页面具体内容

习题

1. 什么是 Scrapy 框架?
2. 如何安装 Scrapy 框架?
3. Scrapy 框架有哪些特点?
4. Scrapy 框架的组成部分是什么?
5. Scrapy 框架的工作原理是什么?
6. 如何使用 Scrapy 框架来爬取页面?