# 第7章

# **网页信息采集**

The largest repository of unstructured text is the Web, and if you know how to crawl it, then you have all the data you need readily available for your experiments. Hence, web crawling is something worth learning for people who are interested in NLTK. This chapter is all about gathering data from the Web.

非结构化文本的最大的存储库是Web，如果你知道如何抓取它，那么你有你需要的所有数据随时可用于你的实验。 因此，网络爬行对于对NLTK感兴趣的人来说是值得学习的。 本章主要从Web上收集数据。

In this chapter we will use an amazing Python library called Scrapy to write our web crawlers. We will provide you all the details to configure different settings that are required. We will write some of the most common spider strategies and many use cases. Scrapy also requires some understanding about XPath, crawling, scraping, and some concepts related to the Web in general. We will touch upon these topics and make sure you understand their practical aspects, before really getting in to their implementation. By the end of this chapter, you will have a better understand of web crawler.

在本章中，我们将使用一个名为Scrapy的令人惊叹的Python库来编写我们的网络爬虫。 我们将为您提供所有详细信息，以配置所需的不同设置。 我们将写一些最常见的蜘蛛策略和许多用例。 Scrapy还需要一些有关XPath，爬网，抓取和一些与Web相关的概念的了解。 我们将探讨这些主题，并确保您了解其实际方面，然后才真正进入实施。 到本章结束时，您将更好地了解网络爬虫。

* How we can write our own crawler using Scrapy
* Understanding about all the major Scrapy functionality
* 我们如何使用Scrapy编写我们自己的爬虫
* 了解所有主要的Scrapy功能

## 网页爬虫

One of the biggest web crawler is Google that crawls the entire World Wide Web (WWW). Google has to traverse every page that exists on the Web and scrape/crawl the entire content.

Google最大的网页抓取工具之一是抓取整个万维网（WWW）。 Google必须遍历网络上存在的每个网页，并抓取/抓取整个内容。

A web crawler is a computer program that systematically browses the web page by page and also scrapes/crawls the content of the pages. A web crawler can also parse the next set of URLs to be visited from the crawled content. So, if these processes run indefinitely over the entire Web, we can crawl through all the web pages. Web crawlers are interchangeably also called spiders, bots, and scrapers. They all mean the same.

网络爬行器是按页面系统地浏览网页并且还刮擦/爬取页面的内容的计算机程序。网络爬虫还可以从爬行的内容解析要访问的下一组URL。因此，如果这些进程在整个Web上无限期运行，我们可以抓取所有网页。网络爬虫可互换地称为蜘蛛，机器人和刮铲。他们都是一样的。

There are a few main points we need to think about before writing our first crawler. Now, every time a web crawler traverses a page, we must decide what kind of content we want to select and what content we want to ignore. For applications such as a search engine, we should ignore all the images, js files, css files, and other files and should concentrate only on HTML content that can be indexed and exposed to the search. In some information extraction engines, we select specific tags or parts of a web page. We also need to extract the URLs if we want to do the crawling recursively. This brings us to the topic of crawling strategy. Here, we need to decide whether we want to go recursively in depth first manner or breadth first manner. We want to follow all the URLs on the next page and then go in depth first manner till we get the URLs, or we should go to all the URLs in the next page and do this recursively.

在写我们的第一个爬虫之前，我们需要考虑几个要点。现在，每当网页抓取工具遍历网页时，我们必须决定要选择什么类型的内容以及我们要忽略的内容。对于诸如搜索引擎的应用程序，我们应该忽略所有的图像，js文件，css文件和其他文件，并且应该只集中在可以索引和暴露给搜索的HTML内容。在一些信息提取引擎中，我们选择特定标签或网页的部分。如果我们想要递归地进行抓取，我们还需要提取网址。这使我们的爬行策略的主题。在这里，我们需要决定是否要以深度第一方式或广度优先方式递归。我们想要跟踪下一页上的所有网址，然后以深度优先的方式进行，直到我们获取网址，或者我们应该转到下一页中的所有网址，并递归地执行。

We also need to make sure that we are not going in the self loop stage because essentially, we traverse a graph in most of the cases. We need to make sure we have a clear revisit strategy for a page. One of the most talked about crawled policies is focused crawling, where we know what kind of domains/topics we are looking for, and the ones that need to be crawled. Some of these issues will be discussed in more detail in the spider section.

我们还需要确保我们不会进入自循环阶段，因为实质上，我们在大多数情况下遍历图。我们需要确保我们有一个清晰的页面重新访问策略。最受关注的抓取政策之一是专注抓取，我们知道我们正在寻找什么样的域/主题，以及需要抓取的域。其中一些问题将在蜘蛛节中更详细地讨论。

|  |
| --- |
| Take a look at the video on Udacity at  看看Udacity的视频  https://www.youtube.com/ watch?v=CDXOcvUNBaA. |

## 编写第一个爬虫程序

Let's start with a very basic crawler that will crawl the entire content of a web page. To write the crawlers, we will use Scrapy. Scrapy is a one of the best crawling solutions using Python. We will explore all the different features of Scrapy in this chapter. First, we need to install Scrapy for this exercise.

让我们从一个非常基本的抓取工具开始，抓取网页的整个内容。 要编写爬虫，我们将使用Scrapy。 Scrapy是使用Python的最好的爬虫解决方案之一。 我们将在本章中探讨Scrapy的所有不同功能。 首先，我们需要为此练习安装Scrapy。

To do this, type in the following command:

为此，请键入以下命令：

$ pip install scrapy

This is the easiest way of installing Scrapy using a package manager. Let's now test whether we got everything right or not. (Ideally, Scrapy should now be part of sys.path):

这是使用包管理器安装Scrapy的最简单的方法。 让我们现在来测试我们是否拥有一切正确。 （理想情况下，Scrapy现在应该是sys.path的一部分）：

>>>import scrapy

|  |
| --- |
| If there is any error, then take a look at  如果有任何错误，那么请看看  http://doc.scrapy.org/ en/latest/intro/install.html. |

At this point, we have Scrapy working for you. Let's start with an example spider app with Scrapy:

在这一点上，我们有Scrapy为你工作。 让我们从一个带有Scrapy的示例蜘蛛应用程序开始：

$ scrapy startproject tutorial

Once you write the preceding command, the directory structure should look like the following:

一旦编写上述命令，目录结构应如下所示：

tutorial/ scrapy.cfg #the project configuration file tutorial/ #the project's python module, you'll later import your code from here. \_\_init\_\_.py items.py #the project's items file. pipelines.py #the project's pipelines file. settings.py # the project's settings file. spiders/ #a directory where you'll later put your spiders. \_\_init\_\_.py

The top folder will be given the name of the example tutorial in this case. Then, there is the project configuration file (scrapy.cfg) that will define the kind of setting file that should be used for the project. It also provides the deploy URLs for the project. Another important part of tutorial setting.py is where we can decide what kind of item pipeline and spider will be used. The item.py and pipline.py are the files that define the data and kind of preprocessing we need to do on the parsed item. The spider folder will contain the different spiders you wrote for the specific URLs.

在这种情况下，顶层文件夹将给出示例教程的名称。 然后，有一个项目配置文件（scrapy.cfg），它将定义项目应该使用的设置文件的种类。 它还提供了项目的部署URL。 教程setting.py的另一个重要部分是，我们可以决定使用什么样的项目管道和蜘蛛。 item.py和pipline.py是定义我们在解析项目上需要做的预处理的数据和种类的文件。 蜘蛛文件夹将包含您为特定网址编写的不同蜘蛛。

For our first test spider, we will dump the contents of a news in a local file. We need to create a file named NewsSpider.py, and put it in the path /tutorial/spiders. Let's write the first spider:

对于我们的第一个测试蜘蛛，我们将在本地文件中转储新闻的内容。 我们需要创建一个名为NewsSpider.py的文件，并将其放在路径/ tutorial / spiders中。 让我们写第一个蜘蛛：

>>>from scrapy.spider import BaseSpider >>>class NewsSpider(BaseSpider):

>>> name = "news"

>>> allowed\_domains = ["nytimes.com"]

>>> start\_URLss = [

>>> 'http://www.nytimes.com/'

>>> ] >>>def parse(self, response):

>>> filename = response.URLs.split("/")[-2]

>>> open(filename, 'wb').write(response.body)

Once we have this spider ready, we can start crawling using the following command:

一旦我们准备好了这个蜘蛛，我们可以使用以下命令开始抓取：

$ scrapy crawl news

After you enter the preceding command, you should see some logs like this:

输入上述命令后，您应该看到如下所示的日志：

[scrapy] INFO: Scrapy 0.24.5 started (bot: tutorial)

[scrapy] INFO: Optional features available: ssl, http11, boto

[scrapy] INFO: Overridden settings: {'NEWSPIDER\_MODULE': 'tutorial. spiders', 'SPIDER\_MODULES': ['tutorial.spiders'], 'BOT\_NAME': 'tutorial'}

[scrapy] INFO: Enabled extensions: LogStats, TelnetConsole, CloseSpider,

WebService, CoreStats, SpiderState

If you don't see logs like the ones shown in the preceding snippet, you have missed something. Check the location of the spider and other Scrapy-related settings, such as the name of the spider matching to the crawl command, and whether setting.py is configured for the same spider and item pipeline or not.

如果您没有看到如上面代码段中显示的日志，那么您错过了某些内容。检查蜘蛛和其他Scrapy相关设置的位置，例如与crawl命令匹配的蜘蛛名称，以及是否为相同的蜘蛛和项目管道配置了setting.py。

Now, if you are successful, there should be a file in your local folder with the name www.nytimes.com that has the entire web content of the www.nytimes.com page.

现在，如果您成功，应在您的本地文件夹中有一个名称为www.nytimes.com的文件，其中包含www.nytimes.com页面的整个Web内容。

Let's see some of the terms that we used in the spider code in more detail: • name: This is the name of the spider that works as an identifier for Scrapy to look for the spider class. So, the crawl command argument and this name should always match. Also make sure that it's unique and case sensitive. • start\_urls: This is a list of URLs where the spider will begin to crawl. The crawler with start from a seed URL and using the parse() method, it will parse and look for the next URL to crawl. Instead of just a single seed URL, we can provide a list of URLs that can start the crawl.

让我们更详细地看看我们在蜘蛛代码中使用的一些术语：•name：这是作为Scrapy的标识符，用于寻找spider类的蜘蛛的名称。因此，crawl命令参数和此名称应始终匹配。还要确保它是唯一的和区分大小写。 •start\_urls：这是蜘蛛将开始抓取的网址列表。抓取工具从种子URL开始，并使用parse（）方法，它将解析并查找下一个要抓取的网址。我们可以提供可以开始抓取的网址列表，而不只是一个种子网址。

• parse(): This method is called to parse the data from start URLs. The logic of what kind of element is to be selected for specific attributes of item. This could be as simple as dumping the entire content of HTML to as complex as many parse methods callable from parse, and different selectors for individual item attributes.

•parse（）：调用此方法以解析来自起始URL的数据。对于项目的特定属性，选择什么样的元素的逻辑。这可以简单到将HTML的整个内容转换为与可从解析中调用的多种解析方法一样复杂，以及针对单个项目属性使用不同的选择器。

So, the code does nothing but starts with the given URLs (in this case, www.nytimes. com) and crawls the entire content of the page. Typically, a crawler is more complex and will do much more than this; now, let's take a step back and understand what happened behind the scenes. For this, take a look at the following figure:

因此，代码不会从给定的URL（在这种情况下，www.nytimes.com）开始，并抓取页面的整个内容。通常，爬虫比较复杂，会做的更多;现在，让我们退后一步，了解幕后发生了什么。为此，看看下图：

（图：图中翻译）

## Scrapy库中的数据流

The data flow in Scrapy is controlled by the execution engine and goes like this:

Scrapy中的数据流由执行引擎控制，如下所示：

1. The process starts with locating the chosen spider and opening the first URL from the list of start\_urls.

该过程开始于找到所选的蜘蛛并从start\_urls列表打开第一个URL。

2. The first URL is then scheduled as a request in a scheduler. This is more of an internal to Scrapy.

然后，第一个URL被调度为调度程序中的请求。 这更多是Scrapy的内部。

3. The Scrapy engine then looks for the next set of URLs to crawl.

Scrapy引擎然后查找要爬网的下一组URL。

4. The scheduler then sends the next URLs to the engine and the engine then forwards it to the downloader using the downloaded middleware. These middlewares are where we place different proxies and user-agent settings.

调度器然后将下一个URL发送到引擎，然后引擎使用下载的中间件将其转发到下载器。 这些中间件是我们放置不同代理和用户代理设置的地方。

5. The downloader downloads the response from the page and passes it to the spider, where the parse method selects specific elements from the response.

下载器从页面下载响应，并将其传递给蜘蛛，其中解析方法从响应中选择特定元素。

6. Then, the spider sends the processed item to the engine.

然后，蜘蛛将处理的物品发送到发动机。

7. The engine sends the processed response to the item pipeline, where we can add some post processing.

引擎将处理后的响应发送到项目管道，在那里我们可以添加一些后处理。

8. The same process continues for each URL until there are no remaining requests.

对每个URL继续相同的过程，直到没有剩余的请求。

### The Scrapy shell

The best way to understand Scrapy is to use it through a shell and to get your hands dirty with some of the initial commands and tools provided by Scrapy. It allows you to experiment and develop your XPath expressions that you can put into your spider code.

理解Scrapy的最好方法是通过shell使用它，并使用Scrapy提供的一些初始命令和工具使你的手变脏。 它允许你实验和开发你的XPath表达式，你可以放入你的蜘蛛代码。

To experiment with the Scrapy shell, I would recommend you to install one of the developer tools (Chrome) and Firebug (Mozilla Firefox) as a plugin. This tool will help us dig down to the very specific part that we want from the web page.

要试验Scrapy shell，我建议您安装一个开发工具（Chrome）和Firebug（Mozilla Firefox）作为插件。 这个工具将帮助我们挖掘我们想从网页的特定部分。

Now, let's start with a very interesting use case where we want to capture the trending topics from Google news (https://news.google.com/).

现在，让我们从一个非常有趣的用例开始，我们要从Google新闻（https://news.google.com/）捕获热门主题。

The steps to follow here are:

这里的步骤是：

1. Open https://news.google.com/ in your favorite browser.

2. Go to the trending topic section on Google news. Then, right-click on and select Inspect Element for the first topic, as shown in the following screenshot:

（图）

3. The moment you open this, there will be a side window that will pop up and you will get a view.

4. Search and select the div tag. For this example, we are interested in <div class="topic">.

5. Once this is done, you will come to know that we have actually parsed the specific part of the web page, as shown in the following screenshot:

1.在您最喜欢的浏览器中打开https://news.google.com/。

2.转到Google新闻的热门话题部分。 然后，右键单击并选择检查元素的第一个主题，如以下屏幕截图所示：

（图）

你打开这一刻的那一刻，会有一个侧面的窗口会弹出，你会得到一个视图。

4.搜索并选择div标签。 对于这个例子，我们感兴趣的是<div class =“topic”>。

一旦这完成，你会知道我们实际上解析了网页的特定部分，如下面的截图所示：

（图）

Now, what we actually did manually in the preceding steps can be done in an automated way. Scrapy uses an XML path language called XPath. XPath can be used to achieve this kind of functionality. So, let's see how we can implement the same example using Scrapy.

现在，我们实际上在之前的步骤中手动执行的操作可以以自动方式完成。 Scrapy使用称为XPath的XML路径语言。 XPath可以用来实现这种功能。 因此，让我们看看如何使用Scrapy实现相同的示例。

To use Scrapy, put the following command in you cmd:

要使用Scrapy，请在cmd中放入以下命令：

$scrapy shell https://news.google.com/

The moment you hit enter, the response of the Google news page is loaded in the Scrapy shell. Now, let's move to the most important aspect of Scrapy where we want to understand how to look for a specific HTML element of the page. Let's start and run the example of getting topics from Google news that are shown in the preceding image:

当您按Enter键时，Google新闻页面的响应会加载到Scrapy shell中。 现在，让我们转到Scrapy的最重要的方面，我们想要了解如何查找页面的特定HTML元素。 让我们开始并运行从Google新闻获取主题的示例，如上图所示：

In [1]: sel.xpath('//div[@class="topic"]').extract()

The output to this will be as follows:

其输出如下：

Out[1]:

[<Selector xpath='//div[@class="topic"]' data=u'<div class="topic"><a href="/news/sectio'>,

<Selector xpath='//div[@class="topic"]' data=u'<div class="topic"><a href="/news/sectio'>,

<Selector xpath='//div[@class="topic"]' data=u'<div class="topic"><a href="/news/sectio'>]

Now, we need to understand some of the functions that Scrapy and XPath provide to experiment with the shell and then, we need to update our spider to do more sophisticated stuff. Scrapy selectors are built with the help of the lxml library, which means that they're very similar in terms of speed and parsing accuracy.

现在，我们需要了解Scrapy和XPath提供的一些功能来实验shell，然后，我们需要更新我们的蜘蛛来做更复杂的工作。 Scrapy选择器是在lxml库的帮助下构建的，这意味着它们在速度和解析精度方面非常相似。

Let's have a look at some of the most frequently used methods provided for selectors:

让我们来看看为选择器提供的一些最常用的方法：

• xpath(): This returns a list of selectors, where each of the selectors represents the nodes selected by the XPath expression given as an argument.

•xpath（）：返回选择器列表，其中每个选择器表示由作为参数给定的XPath表达式选择的节点。

• css(): This returns a list of selectors. Here, each of the selectors represent the nodes selected by the CSS expression given as an argument.

•css（）：返回选择器列表。这里，每个选择器表示由作为参数给出的CSS表达式选择的节点。

• extract():This returns content as a string with the selected data.

•extract（）：返回带有所选数据的字符串的内容。

• re(): This returns a list of unicode strings extracted by applying the regular expression given as an argument.

•re（）：返回通过应用作为参数给出的正则表达式提取的Unicode字符串列表。

I am giving you a cheat sheet of these top 10 selector patterns that can cover most of your work for you. For a more complex selector, if you search the Web, there should be an easy solution that you can use. Let's start with extracting the title of the web page that is very generic for all web pages:

我给你一个这些十大选择器模式的备忘单，可以覆盖你的大部分工作。对于更复杂的选择器，如果您搜索Web，应该有一个容易的解决方案，您可以使用。让我们从提取所有网页非常通用的网页的标题开始：

In [2] :sel.xpath('//title/text()')

Out[2]: [<Selector xpath='//title/text()' data=u' Google News'>]

Now, once you have selected any element, you also want to extract for more processing. Let's extract the selected content. This is a generic method that works with any selector:

现在，一旦你选择了任何元素，你也想提取更多的处理。 让我们提取所选内容。 这是一个适用于任何选择器的通用方法：

In [3]: sel.xpath('//title/text()').extract()

Out[3]: [u' Google News']

The other very generic requirement is to look for all the elements in the given page. Let's achieve this with this selector:

另一个非常通用的要求是查找给定页面中的所有元素。 让我们用这个选择器来实现：

In [4]: sel.xpath('//ul/li')

Out [4] : list of elements (divs and all)

We can extract all the titles in the page with this selector:

我们可以使用此选择器提取页面中的所有标题：

In [5]: sel.xpath('//ul/li/a/text()').extract()

Out [5]: [ u'India',

u'World',

u'Business',

u'Technology',

u'Entertainment',

u'More Top Stories']

With this selector, you can extract all the hyperlinks in the web page:

使用此选择器，您可以提取网页中的所有超链接：

In [6]:sel.xpath('//ul/li/a/@href').extract()

Out [6] : List of urls

Let's select all the <td> and div elements:

In [7]:sel.xpath('td'')

In [8]:divs=sel.xpath("//div")

This will select all the divs elements and then, you can loop it:

In [9]: for d in divs: printd.extract()

This will print the entire content of each div in the entire page. So, in case you are not able to get the exact div name, you can also look at the regex-based search.

这将打印整个页面中每个div的整个内容。 因此，如果您无法获取确切的div名称，也可以查看基于正则表达式的搜索。

Now, let's select all div elements that contain the attribute class="topic":

现在，让我们选择包含属性class =“topic”的所有div元素：

In [10]:sel.xpath('/div[@class="topic"]').extract()

In [11]: sel.xpath("//h1").extract() # this includes the h1 tag

This will select all the <p> elements in the page and get the class of those elements:

In [12 ] for node in sel.xpath("//p"): print node.xpath("@class").extract()

Out[12] print all the <p>

In [13]: sel.xpath("//li[contains(@class, 'topic')]") Out[13]:

[<Selector xpath="//li[contains(@class, 'topic')]" data=u'<li class="navitem nv-FRONTPAGE selecte'>,

<Selector xpath="//li[contains(@class, 'topic')]" data=u'<li class="navitem nv-FRONTPAGE selecte'>]

Let's write some selector nuggets to get the data from a css file. If we just want to extract the title from the css file, typically, everything works the same, except you need to modify the syntax:

让我们编写一些选择器块以从css文件获取数据。 如果我们只想从css文件中提取标题，通常情况下，一切都是一样的，除非你需要修改语法：

In [14] :sel.css('title::text').extract()

Out[14]: [u'Google News']

Use the following command to list the names of all the images used in the page:

In[15]: sel.xpath('//a[contains(@href, "image")]/img/@src').extract()

Out [15] : Will list all the images if the web developer has put the images in /img/src

Let's see a regex-based selector:

In [16 ]sel.xpath('//title').re('(\w+)')

Out[16]: [u'title', u'Google', u'News', u'title']

In some cases, removing the namespaces can help us get the right pattern. A selector has an inbuilt remove\_namespaces() function to make sure that the entire document is scanned and all the namespaces are removed. Make sure before using it whether we want some of these namespaces to be part of the pattern or not. The following is example of remove\_namespaces() function:

在某些情况下，删除命名空间可以帮助我们获得正确的模式。 选择器具有内置的remove\_namespaces（）函数，以确保扫描整个文档，并删除所有命名空间。 在使用它之前，确保我们是否需要其中一些命名空间是模式的一部分。 以下是remove\_namespaces（）函数的示例：

In [17] sel.remove\_namespaces() sel.xpath("//link")

Now that we have more understanding about the selectors, let's modify the same old news spider that we built previously:

现在我们对选择器有了更多的了解，让我们修改我们之前构建的同一个旧的新闻蜘蛛：

>>>from scrapy.spider import BaseSpider >>>class NewsSpider(BaseSpider):

>>> name = "news"

>>> allowed\_domains = ["nytimes.com"]

>>> start\_URLss = [

>>> 'http://www.nytimes.com/'

>>> ]

>>>def parse(self, response):

>>> sel = Selector(response)

>>> sites = sel.xpath('//ul/li') >>> for site in sites:

>>> title = site.xpath('a/text()').extract()

>>> link = site.xpath('a/@href').extract()

>>> desc = site.xpath('text()').extract()

>>> print title, link, desc

Here, we mainly modified the parse method, which is one of the core of our spider. This spider can now crawl through the entire page, but we do a more structured parsing of the title, description, and URLs.

这里，我们主要修改parse方法，这是我们蜘蛛的核心之一。 此蜘蛛可以抓取整个页面，但我们对标题，说明和网址进行了更加结构化的解析。

Now, let's write a more robust crawler using all the capabilities of Scrapy.

现在，让我们使用Scrapy的所有功能编写一个更强大的crawler。

### Items

Until now, we were just printing the crawled content on stdout or dumping it in a file. A better way to do this is to define items.py every time we write a crawler. The advantage of doing this is that we can consume these items in our parse method, and this can also give us output in any data format, such as XML, JSON, or CSV. So, if you go back to your old crawler, the items class will have a function like this:

到目前为止，我们只是在stdout上打印抓取的内容或将其转储到文件中。 一个更好的方法是每次我们编写一个crawler时定义items.py。 这样做的优点是，我们可以在我们的解析方法中使用这些项目，这也可以提供任何数据格式的输出，例如XML，JSON或CSV。 所以，如果你回到你的旧抓取器，items类将有这样的功能：

>>>fromscrapy.item import Item, Field >>>class NewsItem(scrapy.Item):

>>> # define the fields for your item here like:

>>> # name = scrapy.Field()

>>> pass

Now, let's make it like the following by adding different fields:

>>>from scrapy.item import Item, Field >>>class NewsItem(Item):

>>> title = Field()

>>> link = Field()

>>> desc = Field()

Here, we added field() to title, link, and desc. Once we have a field in place, our spider parse method can be modified to parse\_news\_item, where instead dumping the parsed fields to a file now it can be consumed by an item object.

在这里，我们添加了field（）到title，link和desc。一旦我们有一个字段，我们的蜘蛛解析方法可以修改为parse\_news\_item，而不是将解析的字段转储到一个文件，现在它可以被项目对象使用。

A Rule method is a way of specifying what kind of URL needs to be crawled after the current one. A Rule method provides SgmlLinkExtractor, which is a way of defining the URL pattern that needs to be extracted from the crawled page. A Rule method also provides a callback method, which is typically a pointer for a spider to look for the parsing method, which in this case is parse\_news\_item. In case we have a different way to parse, then we can have multiple rules and parse methods. A Rule method also has a Boolean parameter to follow, which specifies whether links should be followed by each response extracted with this rule. If the callback is None, follow defaults to True: otherwise, it default to False.

规则方法是一种指定在当前网址之后需要抓取哪种类型的网址的方法。规则方法提供了SgmlLinkExtractor，这是一种定义需要从已爬网页面提取的网址格式的方法。 Rule方法还提供了一个回调方法，它通常是一个用于寻找解析方法的蜘蛛的指针，在这种情况下是parse\_news\_item。如果我们有不同的方式来解析，那么我们可以有多个规则和解析方法。规则方法也有一个布尔参数，它指定是否应该使用此规则提取的每个响应后跟链接。如果回调为None，默认为True：否则默认为False。

One important point to note is that the Rule method does not use parse. This is because the name of the default callback method is parse() and if we use it, we are actually overriding it, and that can stop the functionality of the crawl spider. Now, let's jump on to the following code to understand the preceding methods and parameters:

需要注意的一个重要的点是规则方法不使用解析。这是因为默认回调方法的名称是parse（），如果我们使用它，我们实际上覆盖它，并且可以停止爬网蜘蛛的功能。现在，让我们跳到下面的代码来了解前面的方法和参数：

>>>from scrapy.contrib.spiders import CrawlSpider, Rule

>>>from scrapy.contrib.linkextractors.sgml import SgmlLinkExtractor

>>>from scrapy.selector import Selector

>>>from scrapy.item import NewsItem >>>class NewsSpider(CrawlSpider):

>>> name = 'news'

>>> allowed\_domains = ['news.google.com']

>>> start\_urls = ['https://news.google.com']

>>> rules = (

>>> # Extract links matching cnn.com

>>> Rule(SgmlLinkExtractor(allow=('cnn.com', ), deny=(http:// edition.cnn.com/', ))), >>> # Extract links matching 'news.google.com'

>>> Rule(SgmlLinkExtractor(allow=('news.google.com', )), callback='parse\_news\_item'), >>> ) >>> def parse\_news\_item(self, response):

>>> sel = Selector(response)

>>> item = NewsItem()

>>> item['title'] = sel.xpath('//title/text()').extract()

>>> item[topic] = sel.xpath('/div[@class="topic"]').extract()

>>> item['desc'] = sel.xpath('//td//text()').extract()

>>> return item

## The Sitemap spider

If the site provides sitemap.xml, then a better way to crawl the site is to use SiteMapSpider instead.

如果网站提供sitemap.xml，那么更好的方式来抓取网站是使用SiteMapSpider。

Here, given sitemap.xml, the spider parses the URLs provided by the site itself. This is a more polite way of crawling and good practice:

在这里，给定sitemap.xml，蜘蛛会解析网站本身提供的网址。 这是一个更有礼貌的爬行方式和良好的做法：

>>>from scrapy.contrib.spiders import SitemapSpider >>>class MySpider(SitemapSpider):

>>> sitemap\_URLss = ['http://www.example.com/sitemap.xml']

>>> sitemap\_rules = [('/electronics/', 'parse\_electronics'), ('/ apparel/', 'parse\_apparel'),] >>> def 'parse\_electronics'(self, response):

>>> # you need to create an item for electronics,

>>> return >>> def 'parse\_apparel'(self, response):

>>> #you need to create an item for apparel

>>> return

In the preceding code, we wrote one parse method for each product category. It's a great use case if you want to build a price aggregator/comparator. You might want to parse different attributes for different products, for example, for electronics, you might want to scrape the tech specification, accessory, and price; while for apparels, you are more concerned about the size and color of the item. Try your hand at using one of the retailer sites and use shell to get the patterns to scrape the size, color, and price of different items. If you do this, you should be in a good shape to write your first industry standard spider.

在上面的代码中，我们为每个产品类别编写了一个解析方法。 这是一个伟大的用例，如果你想建立一个价格聚合/比较器。 您可能需要解析不同产品的不同属性，例如，对于电子产品，您可能想要刮除技术规格，附件和价格; 而对于服装，你更关心的项目的大小和颜色。 尝试你的手在使用零售商网站之一，并使用shell来获取模式，以刮除不同项目的大小，颜色和价格。 如果你这样做，你应该在一个良好的形状，写你的第一个工业标准的蜘蛛。

In some cases, you want to crawl a website that needs you to log in before you can enter some parts of the website. Now, Scrapy has a workaround that too. They implemented FormRequest, which is more of a POST call to the HTTP server and gets the response. Let's have a deeper look into the following spider code:

在某些情况下，您希望抓取需要您登录的网站，然后才能输入网站的某些部分。 现在，Scrapy也有一个解决方法。 他们实现了FormRequest，它更多的是对HTTP服务器的POST调用，并获得响应。 让我们更深入地了解下面的蜘蛛代码：

>>>class LoginSpider(BaseSpider):

>>> name = 'example.com'

>>> start\_URLss = ['http://www.example.com/users/login.php'] >>> def parse(self, response):

>>> return [FormRequest.from\_response(response,

formdata={'username': 'john', 'password': 'secret'}, callback=self.after\_ login)] >>> def after\_login(self, response):

>>> # check login succeed before going on

>>> if "authentication failed" in response.body:

>>> self.log("Login failed", level=log.ERROR)

>>> return

For a website that requires just the username and password without any captcha, the preceding code should work just by adding the specific login details. This is the part of the parse method since you need to log in the first page in the most of the cases. Once you log in, you can write your own after\_login callback method with items and other details.

对于只需要用户名和密码而没有任何验证码的网站，上述代码只需添加特定的登录详细信息即可。 这是解析方法的一部分，因为您需要在大多数情况下登录第一页。 一旦你登录，你可以编写自己的after\_login回调方法与项目和其他细节。

## The item pipeline

Let's talk about some more item postprocessing. Scrapy provides a way to define a pipeline for items as well, where you can define the kind of post processing an item has to go through. This is a very methodical and good program design.

让我们谈谈一些更多的项目后处理。 Scrapy提供了一种为项目定义管道的方法，您可以在其中定义项目必须经过的后处理类型。 这是一个非常有条理和良好的程序设计。

We need to build our own item pipeline if we want to post process scraped items, such as removing noise and case conversion, and in other cases, where we want to derive some values from the object, for example, to calculate the age from DOB or to calculate the discount price from the original price. In the end, we might want to dump the item separately into a file.

我们需要构建自己的项目管道，如果我们想要处理刮擦的项目，如去除噪声和大小写转换，在其他情况下，我们想从对象中获取一些值，例如，从DOB计算年龄 或从原始价格计算折扣价格。 最后，我们可能需要将该项目单独转储到一个文件中。

The way to achieve this will be as follows:

实现这一点的方法如下：

1. We need to define an item pipeline in setting.py:

我们需要在setting.py中定义一个项目管道：

ITEM\_PIPELINES = {

'myproject.pipeline.CleanPipeline': 300,

'myproject.pipeline.AgePipeline': 500,

'myproject.pipeline.DuplicatesPipeline: 700,

'myproject.pipeline.JsonWriterPipeline': 800,

}

2. Let's write a class to clean the items:

让我们编写一个类来清理项目：

>>>from scrapy.exceptions import Item

>>>import datetime

>>>import datetime >>>class AgePipeline(object):

>>> def process\_item(self, item, spider):

>>> if item['DOB']:

>>> item['Age'] = (datetime.datetime. strptime(item['DOB'], '%d-%m-%y').date()-datetime.datetime. strptime('currentdate, '%d-%m-%y').date()).days/365

>>> return item

3. We need to derive the age from DOB. We used Python's date functions to achieve this:

我们需要从DOB推导出年龄。 我们使用Python的日期函数来实现这一点：

>>>from scrapy import signals

>>>from scrapy.exceptions import Item >>>class DuplicatesPipeline(object):

>>> def \_\_init\_\_(self):

>>> self.ids\_seen = set() >>> def process\_item(self, item, spider):

>>> if item['id'] in self.ids\_seen:

>>> raise DropItem("Duplicate item found: %s" % item) >>> else:

>>> self.ids\_seen.add(item['id'])

>>> return item

4. We also need to remove the duplicates. Python has the set() data structure that only contains unique values, we can create a pipline DuplicatesPipeline.py like below using Scrapy :

我们还需要删除重复的。 Python的set（）数据结构只包含唯一的值，我们可以使用Scrapy创建一个如下所示的pipline DuplicatesPipeline.py：

>>>from scrapy import signals

>>>from scrapy.exceptions import Item >>>class DuplicatesPipeline(object):

>>> def \_\_init\_\_(self):

>>> self.ids\_seen = set() >>> def process\_item(self, item, spider):

>>> if item['id'] in self.ids\_seen:

>>> raise DropItem("Duplicate item found: %s" % item) >>> else:

>>> self.ids\_seen.add(item['id'])

>>> return item

5. Let's finally write the item in the JSON file using JsonWriterPipeline.py pipeline:

让我们最后使用JsonWriterPipeline.py pipeline在JSON文件中编写项目：

>>>import json

>>>class JsonWriterPipeline(object):

>>> def \_\_init\_\_(self):

>>> self.file = open('items.txt', 'wb') >>> def process\_item(self, item, spider):

>>> line = json.dumps(dict(item)) + "\n"

>>> self.file.write(line)

>>> return item

## 外部参考资料

I encourage you to follow some simple spiders and try building some cool applications using these spiders. I would also like you to look at the following links for reference:

我鼓励你遵循一些简单的蜘蛛，并尝试使用这些蜘蛛构建一些很酷的应用程序。 我也希望你看看以下链接供参考：

* <http://doc.scrapy.org/en/latest/intro/tutorial.html>
* <http://doc.scrapy.org/en/latest/intro/overview.html>

## 本章小结

In this chapter, you learned about another great Python library and now, you don't need help from anybody for your data needs. You learned how you can write a very sophisticated crawling system, and now you know how to write a focused spider. In this chapter, we saw how to abstract the item logic from the main system and how to write some specific spider for the most common use cases. We know some of the most common settings that need to be taken care of in order to implement our own spider and we wrote some complex parse methods that can be reused. We understand selectors very well and know a hands-on way of figuring out what kind of elements we want for specific item attributes, and we also went through Firebug to get more of a practical understanding of selectors. Last but not least, very importantly, make sure that you follow the security guidelines of the websites you crawl.

在本章中，你学习了另一个伟大的Python库，现在，你不需要任何人的帮助为您的数据需要。你学会了如何编写一个非常复杂的爬虫系统，现在你知道如何写一个专注的蜘蛛。在本章中，我们看到了如何从主系统抽象项目逻辑，以及如何为最常见的用例编写一些特定的蜘蛛。我们知道一些最常见的设置，需要注意为了实现我们自己的蜘蛛，我们写了一些复杂的解析方法，可以重复使用。我们非常了解选择器，并且知道一个动手的方法来确定我们想要的特定项目属性，我们也通过Firebug获得更多的选择器的实际理解。最后但同样重要的是，请务必遵守您抓取的网站的安全指南。

In the next chapter, we will explore some essential Python libraries that can be used for natural language processing and machine learning.

在下一章中，我们将探讨一些基本的Python库，可以用于自然语言处理和机器学习。