Scrapy教程

在本教程中，我们假设您的系统上已经安装了Scrapy。如果不是这种情况，请参阅安装指南。

我们将刮掉quotes.toscrape.com，这是一个列出着名作家引用的网站。

本教程将指导您完成以下任务：

1. 创建一个新的Scrapy项目
2. 编写蜘蛛来抓取网站并提取数据
3. 使用命令行导出已删除的数据
4. 改变蜘蛛以递归方式跟随链接
5. 使用蜘蛛参数

Scrapy是用Python编写的。如果您不熟悉该语言，您可能需要先了解语言是什么，以充分利用Scrapy。

如果您已经熟悉其他语言，并希望快速学习Python，我们建议您阅读Dive Into Python 3。或者，您可以按照Python教程进行操作。

如果您不熟悉编程并希望从Python开始，那么您可能会发现有用的在线书籍“ 学习Python困难之路”。您还可以查看非程序员的Python资源列表。

创建项目

在开始抓取之前，您必须设置一个新的Scrapy项目。输入您要存储代码的目录并运行：

```
scrapy startproject tutorial
```

这将创建一个 `tutorial` 包含以下内容的目录：

```
tutorial/
    scrapy.cfg            # deploy configuration file

    tutorial/             # project's Python module, you'll import your code from here
        __init__.py

        items.py          # project items definition file

        middlewares.py    # project middlewares file

        pipelines.py      # project pipelines file

        settings.py       # project settings file

        spiders/          # a directory where you'll later put your spiders
            __init__.py
```

我们的第一个蜘蛛

蜘蛛是您定义的类，Scrapy用来从网站（或一组网站）中提取信息。它们必须子类化 `scrapy.Spider` 并定义要生成的初始请求，可选地如何跟踪页面中的链接，以及如何解析下载的页面内容以提取数据。

这是我们第一个蜘蛛的代码。将其保存在 项目目录 `quotes_spider.py` 下的 `tutorial/spiders` 文件中：

```
import scrapy


class QuotesSpider(scrapy.Spider):
    name = "quotes"

    def start_requests(self):
        urls = [
            'http://quotes.toscrape.com/page/1/',
            'http://quotes.toscrape.com/page/2/',
        ]
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)

    def parse(self, response):
        page = response.url.split("/")[-2]
        filename = 'quotes-%s.html' % page
        with open(filename, 'wb') as f:
            f.write(response.body)
        self.log('Saved file %s' % filename)
```

如您所见，我们的Spider子类 `scrapy.Spider` 并定义了一些属性和方法：

- `name` ：识别蜘蛛。它在项目中必须是唯一的，也就是说，您不能为不同的Spiders设置相同的名称。
- `start_requests()` ：必须返回Spider将开始爬行的可迭代请求（您可以返回请求列表或编写生成器函数）。后续请求将从这些初始请求中连续生成。
- `parse()` ：将调用一个方法来处理为每个请求下载的响应。响应参数是 `TextResponse` 保存页面内容的实例，并具有处理它的其他有用方法。

该 `parse()` 方法通常解析响应，将抽取的数据提取为dicts，并查找要遵循的新URL并 `Request` 从中创建新的request（ ）。

如何运行我们的蜘蛛

要让我们的蜘蛛工作，请转到项目的顶级目录并运行：

```
scrapy crawl quotes
```

此命令运行 `quotes` 我们刚添加的名称的spider ，它将发送一些 `quotes.toscrape.com` 域请求。您将获得类似于此的输出：

```
... (omitted for brevity)
2016-12-16 21:24:05 [scrapy.core.engine] INFO: Spider opened
2016-12-16 21:24:05 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pages/min),
scraped 0 items (at 0 items/min)
2016-12-16 21:24:05 [scrapy.extensions.telnet] DEBUG: Telnet console listening on
127.0.0.1:6023
2016-12-16 21:24:05 [scrapy.core.engine] DEBUG: Crawled (404) <GET
http://quotes.toscrape.com/robots.txt> (referer: None)
2016-12-16 21:24:05 [scrapy.core.engine] DEBUG: Crawled (200) <GET
http://quotes.toscrape.com/page/1/> (referer: None)
2016-12-16 21:24:05 [scrapy.core.engine] DEBUG: Crawled (200) <GET
http://quotes.toscrape.com/page/2/> (referer: None)
2016-12-16 21:24:05 [quotes] DEBUG: Saved file quotes-1.html
2016-12-16 21:24:05 [quotes] DEBUG: Saved file quotes-2.html
2016-12-16 21:24:05 [scrapy.core.engine] INFO: Closing spider (finished)
...
```

现在，检查当前目录中的文件。您应该注意到已经创建了两个新文件：*quotes-1.html*和*quotes-2.html*，以及各个URL的内容，正如我们的 `parse` 方法所指示的那样。

注意

如果您想知道为什么我们还没有解析HTML，请继续，我们将很快介绍。

引擎盖下发生了什么？

Scrapy会调度Spider方法 `scrapy.Request` 返回的对象 `start_requests` 。在收到每个响应后，它实例化 `Response` 对象并调用与请求相关的回调方法（在本例中为 `parse` 方法），将响应作为参数传递。

start_requests方法的快捷方式

您可以只使用URL列表定义类属性，而不是实现从URL `start_requests()` 生成 `scrapy.Request` 对象的方法 `start_urls` 。然后，默认实现将使用此列表 `start_requests()` 来为您的spider创建初始请求：

```
import scrapy


class QuotesSpider(scrapy.Spider):
    name = "quotes"
    start_urls = [
        'http://quotes.toscrape.com/page/1/',
        'http://quotes.toscrape.com/page/2/',
    ]

    def parse(self, response):
        page = response.url.split("/")[-2]
        filename = 'quotes-%s.html' % page
        with open(filename, 'wb') as f:
            f.write(response.body)
```

`parse()`即使我们没有明确告诉Scrapy这样做，也会调用该方法来处理这些URL的每个请求。发生这种情况的原因`parse()`是Scrapy的默认回调方法，在没有明确分配回调的情况下调用请求。

提取数据

学习如何使用Scrapy提取数据的最佳方法是使用shell Scrapy shell尝试选择器。跑：

```
scrapy shell 'http://quotes.toscrape.com/page/1/'
```

注意

当从命令行运行Scrapy shell时，请记住始终将URL括在引号中，否则包含参数（即`&`字符）的url 将不起作用。

在Windows上，请使用双引号：

```
scrapy shell "http://quotes.toscrape.com/page/1/"
```

你会看到类似的东西：

```
[ ... Scrapy log here ... ]
2016-09-19 12:09:27 [scrapy.core.engine] DEBUG: Crawled (200) <GET
http://quotes.toscrape.com/page/1/> (referer: None)
[s] Available Scrapy objects:
[s]   scrapy     scrapy module (contains scrapy.Request, scrapy.Selector, etc)
[s]   crawler    <scrapy.crawler.Crawler object at 0x7fa91d888c90>
[s]   item       {}
[s]   request    <GET http://quotes.toscrape.com/page/1/>
[s]   response   <200 http://quotes.toscrape.com/page/1/>
[s]   settings   <scrapy.settings.Settings object at 0x7fa91d888c10>
[s]   spider     <DefaultSpider 'default' at 0x7fa91c8af990>
[s] Useful shortcuts:
[s]   shelp()           Shell help (print this help)
[s]   fetch(req_or_url) Fetch request (or URL) and update local objects
[s]   view(response)    View response in a browser
>>>
```

使用shell，您可以尝试使用CSS和响应对象选择元素：

```
>>> response.css('title')
[<Selector xpath='descendant-or-self::title' data='<title>Quotes to Scrape</title>'>]
```

运行的结果`response.css('title')`是一个类似于列表的对象`SelectorList`，它表示一个`Selector`包含XML / HTML元素的对象列表， 并允许您运行进一步的查询以细化选择或提取数据。

要从上面的标题中提取文本，您可以执行以下操作：

```
>>> response.css('title::text').extract()
['Quotes to Scrape']
```

这里有两点需要注意：一个是我们已经添加`::text`到CSS查询中，意味着我们只想在元素内直接选择文本元素`<title>`。如果我们不指定`::text`，我们将获得完整的title元素，包括其标签：

```
>>> response.css('title').extract()
['<title>Quotes to Scrape</title>']
```

另一件事是调用的结果`.extract()`是一个列表，因为我们正在处理一个实例`SelectorList`。当你知道你只想要第一个结果时，就像在这种情况下，你可以这样做：

```
>>> response.css('title::text').extract_first()
'Quotes to Scrape'
```

作为替代方案，你可以写：

```
>>> response.css('title::text')[0].extract()
'Quotes to Scrape'
```

但是，当它找不到与选择匹配的任何元素时，使用`.extract_first()`避免`IndexError`和返回`None`。

这里有一个教训：对于大多数抓取代码，您希望它能够在页面上找不到任何内容时对错误具有弹性，因此即使某些部分无法被删除，您也至少可以获得**一些**数据。

除了`extract()`和`extract_first()`方法之外，您还可以使用该`re()`方法使用*正则表达式*进行提取：

```
>>> response.css('title::text').re(r'Quotes.*')
['Quotes to Scrape']
>>> response.css('title::text').re(r'Q\w+')
['Quotes']
>>> response.css('title::text').re(r'(\w+) to (\w+)')
['Quotes', 'Scrape']
```

为了找到合适的CSS选择器，您可能会发现使用Web浏览器中的shell打开响应页面非常有用`view(response)`。您可以使用浏览器开发人员工具或扩展程序（如Firebug）（请参阅有关使用Firebug进行抓取和使用Firefox进行抓取的部分）。

Selector Gadget也是一个很好的工具，可以快速找到视觉选择元素的CSS选择器，它可以在许多浏览器中使用。

XPath：简要介绍

除了CSS，Scrapy选择器还支持使用XPath表达式：

```
>>> response.xpath('//title')
[<Selector xpath='//title' data='<title>Quotes to Scrape</title>'>]
>>> response.xpath('//title/text()').extract_first()
'Quotes to Scrape'
```

XPath表达式非常强大，是Scrapy Selectors的基础。实际上，CSS选择器在引擎盖下转换为XPath。如果仔细阅读shell中选择器对象的文本表示，则可以看到。

虽然可能不像CSS选择器那样流行，但XPath表达式提供了更多功能，因为除了导航结构之外，它还可以查看内容。使用XPath，您可以选择以下内容：*选择包含文本“下一页”的链接*。这使得XPath非常适合抓取任务，我们鼓励你学习XPath，即使你已经知道如何构造CSS选择器，它也会使抓取更容易。

我们不会在这里介绍XPath的大部分内容，但您可以在此处阅读有关在Scrapy选择器中使用XPath的更多信息。要了解有关XPath的更多信息，我们建议本教程通过示例学习XPath，本教程将学习“如何在XPath中思考”。

提取引号和作者

现在你已经了解了一些关于选择和提取的知识，让我们通过编写代码从网页中提取引号来完成我们的蜘蛛。

http://quotes.toscrape.com中的每个引用都由HTML元素表示，如下所示：

```
<div class="quote">
    <span class="text">“The world as we have created it is a process of our
    thinking. It cannot be changed without changing our thinking.”</span>
    <span>
        by <small class="author">Albert Einstein</small>
        <a href="/author/Albert-Einstein">(about)</a>
    </span>
    <div class="tags">
        Tags:
        <a class="tag" href="/tag/change/page/1/">change</a>
        <a class="tag" href="/tag/deep-thoughts/page/1/">deep-thoughts</a>
        <a class="tag" href="/tag/thinking/page/1/">thinking</a>
        <a class="tag" href="/tag/world/page/1/">world</a>
    </div>
</div>
```

让我们打开scrapy shell并播放一下以了解如何提取我们想要的数据：

```
$ scrapy shell 'http://quotes.toscrape.com'
```

我们得到了引用HTML元素的选择器列表：

```
>>> response.css("div.quote")
```

上面的查询返回的每个选择器允许我们对其子元素运行进一步的查询。让我们将第一个选择器分配给一个变量，这样我们就可以直接在特定的引号上运行CSS选择器：

```
>>> quote = response.css("div.quote")[0]
```

现在，让我们来提取 `title` ， `author` 而 `tags` 从报价使用 `quote` 我们刚刚创建的对象：

```
>>> title = quote.css("span.text::text").extract_first()
>>> title
'“The world as we have created it is a process of our thinking. It cannot be changed without
changing our thinking.”'
>>> author = quote.css("small.author::text").extract_first()
>>> author
'Albert Einstein'
```

鉴于标签是一个字符串列表，我们可以使用该 `.extract()` 方法获取所有这些：

```
>>> tags = quote.css("div.tags a.tag::text").extract()
>>> tags
['change', 'deep-thoughts', 'thinking', 'world']
```

在弄清楚如何提取每个位之后，我们现在可以迭代所有引号元素并将它们放在一起放入Python字典：

```
>>> for quote in response.css("div.quote"):
...     text = quote.css("span.text::text").extract_first()
...     author = quote.css("small.author::text").extract_first()
...     tags = quote.css("div.tags a.tag::text").extract()
...     print(dict(text=text, author=author, tags=tags))
{'tags': ['change', 'deep-thoughts', 'thinking', 'world'], 'author': 'Albert Einstein', 'text':
'“The world as we have created it is a process of our thinking. It cannot be changed without
changing our thinking.”'}
{'tags': ['abilities', 'choices'], 'author': 'J.K. Rowling', 'text': '“It is our choices,
Harry, that show what we truly are, far more than our abilities.”'}
    ... a few more of these, omitted for brevity
>>>
```

在我们的蜘蛛中提取数据

让我们回到我们的蜘蛛。到目前为止，它并没有特别提取任何数据，只是将整个HTML页面保存到本地文件中。让我们将上面的提取逻辑集成到我们的蜘蛛中。

Scrapy蜘蛛通常会生成许多包含从页面提取的数据的字典。为此，我们 `yield` 在回调中使用Python关键字，如下所示：

```
import scrapy


class QuotesSpider(scrapy.Spider):
    name = "quotes"
    start_urls = [
        'http://quotes.toscrape.com/page/1/',
        'http://quotes.toscrape.com/page/2/',
    ]

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').extract_first(),
                'author': quote.css('small.author::text').extract_first(),
                'tags': quote.css('div.tags a.tag::text').extract(),
            }
```

如果你运行这个蜘蛛，它将输出提取的数据与日志：

```
2016-09-19 18:57:19 [scrapy.core.scraper] DEBUG: Scraped from <200
http://quotes.toscrape.com/page/1/>
{'tags': ['life', 'love'], 'author': 'André Gide', 'text': '“It is better to be hated for what
you are than to be loved for what you are not.”'}
2016-09-19 18:57:19 [scrapy.core.scraper] DEBUG: Scraped from <200
http://quotes.toscrape.com/page/1/>
{'tags': ['edison', 'failure', 'inspirational', 'paraphrased'], 'author': 'Thomas A. Edison',
'text': "“I have not failed. I've just found 10,000 ways that won't work.”"}
```

存储已删除的数据

存储已删除数据的最简单方法是使用Feed导出，使用以下命令：

```
scrapy crawl quotes -o quotes.json
```

这将生成一个 `quotes.json` 包含所有已删除项目的文件，以JSON序列化。

由于历史原因，Scrapy会附加到给定文件而不是覆盖其内容。如果在第二次之前没有删除文件的情况下运行此命令两次，则最终会出现损坏的JSON文件。

您还可以使用其他格式，例如JSON Lines：

```
scrapy crawl quotes -o quotes.jl
```

该JSON行格式是有用的，因为它的流状，你可以很容易地新记录追加到它。当你运行两次时，它没有相同的JSON问题。此外，由于每条记录都是一个单独的行，您可以处理大文件而无需将所有内容都放在内存中，有些工具如JQ可以帮助您在命令行中执行此操作。

在小项目（如本教程中的项目）中，这应该足够了。但是，如果要使用已删除的项目执行更复杂的操作，可以编写项目管道。项目管道的占位符文件已在创建项目时为您设置`tutorial/pipelines.py`。如果您只想存储已删除的项目，则不需要实现任何项目管道。

以下链接

让我们说，你不需要从http://quotes.toscrape.com的前两页中抓取内容，而是需要来自网站所有页面的引用。

既然您知道如何从页面中提取数据，那么让我们看看如何从它们中获取链接。

首先是提取我们想要关注的页面的链接。检查我们的页面，我们可以看到下一页的链接带有以下标记：

```
<ul class="pager">
    <li class="next">
        <a href="/page/2/">Next <span aria-hidden="true">&rarr;</span></a>
    </li>
</ul>
```

我们可以尝试在shell中提取它：

```
>>> response.css('li.next a').extract_first()
'<a href="/page/2/">Next <span aria-hidden="true">→</span></a>'
```

这将获取锚元素，但我们需要该属性`href`。为此，Scrapy支持CSS扩展，让您选择属性内容，如下所示：

```
>>> response.css('li.next a::attr(href)').extract_first()
'/page/2/'
```

现在让我们看看我们的蜘蛛被修改为递归地跟随到下一页的链接，从中提取数据：

```
import scrapy


class QuotesSpider(scrapy.Spider):
    name = "quotes"
    start_urls = [
        'http://quotes.toscrape.com/page/1/',
    ]

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').extract_first(),
                'author': quote.css('small.author::text').extract_first(),
                'tags': quote.css('div.tags a.tag::text').extract(),
            }

        next_page = response.css('li.next a::attr(href)').extract_first()
        if next_page is not None:
            next_page = response.urljoin(next_page)
            yield scrapy.Request(next_page, callback=self.parse)
```

现在，在提取数据之后，该`parse()`方法查找到下一页的链接，使用该`urljoin()`方法构建完整的绝对URL（因为链接可以是相对的）并向下一页生成新请求，将自身注册为回调来处理下一页的数据提取，并保持爬网遍历所有页面。

你在这里看到的是Scrapy的跟踪链接机制：当你在回调方法中产生一个Request时，Scrapy会安排发送该请求并注册一个回调方法，以便在该请求完成时执行。

使用此功能，您可以根据您定义的规则构建遵循链接的复杂爬网程序，并根据其访问的页面提取不同类型的数据。

在我们的示例中，它创建了一种循环，跟随到下一页的所有链接，直到它找不到一个 - 用于爬行博客，论坛和其他具有分页的站点。

创建请求的快捷方式

作为创建Request对象的快捷方式，您可以使用`response.follow`：

```
import scrapy


class QuotesSpider(scrapy.Spider):
    name = "quotes"
    start_urls = [
        'http://quotes.toscrape.com/page/1/',
    ]

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').extract_first(),
                'author': quote.css('span small::text').extract_first(),
                'tags': quote.css('div.tags a.tag::text').extract(),
            }

        next_page = response.css('li.next a::attr(href)').extract_first()
        if next_page is not None:
            yield response.follow(next_page, callback=self.parse)
```

与scrapy.Request不同，它 `response.follow` 直接支持相对URL - 无需调用urljoin。注意， `response.follow` 只返回一个Request实例; 你仍然需要提出这个请求。

您也可以传递选择器 `response.follow` 而不是字符串; 此选择器应提取必要的属性：

```
for href in response.css('li.next a::attr(href)'):
    yield response.follow(href, callback=self.parse)
```

对于 `<a>` 元素，有一个快捷方式： `response.follow` 自动使用其href属性。所以代码可以进一步缩短：

```
for a in response.css('li.next a'):
    yield response.follow(a, callback=self.parse)
```

注意

`response.follow(response.css('li.next a'))` 无效是因为 `response.css` 返回一个类似于列表的对象，其中包含所有结果的选择器，而不是单个选择器。甲 `for` 象在上面的例子中循环，或 是好的。 `response.follow(response.css('li.next a')[0])`

更多示例和模式

这是另一个蜘蛛，它说明了回调和以下链接，这次是为了抓取作者信息：

```
import scrapy


class AuthorSpider(scrapy.Spider):
    name = 'author'

    start_urls = ['http://quotes.toscrape.com/']

    def parse(self, response):
        # follow links to author pages
        for href in response.css('.author + a::attr(href)'):
            yield response.follow(href, self.parse_author)

        # follow pagination links
        for href in response.css('li.next a::attr(href)'):
            yield response.follow(href, self.parse)

    def parse_author(self, response):
        def extract_with_css(query):
            return response.css(query).extract_first().strip()

        yield {
            'name': extract_with_css('h3.author-title::text'),
            'birthdate': extract_with_css('.author-born-date::text'),
            'bio': extract_with_css('.author-description::text'),
        }
```

这个蜘蛛将从主页面开始，它将跟随作者页面的所有链接，`parse_author`为每个页面调用回调，以及`parse`我们之前看到的与回调的分页链接。

这里我们将回调传递给`response.follow`位置参数以使代码更短; 它也适用于`scrapy.Request`。

该`parse_author`回调定义了一个辅助函数从CSS查询提取和清理数据，并产生了Python字典与作者的数据。

这个蜘蛛演示的另一个有趣的事情是，即使同一作者有很多引用，我们也不必担心多次访问同一作者页面。默认情况下，Scrapy会筛选出已访问过的URL的重复请求，从而避免因编程错误而导致服务器过多的问题。这可以通过设置进行配置`DUPEFILTER_CLASS`。

希望到现在为止，您已经很好地理解了如何使用Scrapy跟踪链接和回调的机制。

作为利用以下链接机制的另一个示例蜘蛛，请查看`CrawlSpider`类的通用蜘蛛，该蜘蛛实现了一个小规则引擎，您可以使用该规则引擎在其上编写爬虫。

此外，一个常见的模式是使用一个技巧来创建一个包含来自多个页面的数据的项目，以便将其他数据传递给回调。

使用蜘蛛参数

您可以`-a`在运行蜘蛛时使用该选项为您的蜘蛛提供命令行参数：

```
scrapy crawl quotes -o quotes-humor.json -a tag=humor
```

这些参数传递给Spider的`__init__`方法，默认情况下变为spider属性。

在此示例中，为参数提供的值`tag`将通过`self.tag`。您可以使用此选项使您的蜘蛛只获取具有特定标记的引号，并根据参数构建URL：

```
import scrapy


class QuotesSpider(scrapy.Spider):
    name = "quotes"

    def start_requests(self):
        url = 'http://quotes.toscrape.com/'
        tag = getattr(self, 'tag', None)
        if tag is not None:
            url = url + 'tag/' + tag
        yield scrapy.Request(url, self.parse)

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').extract_first(),
                'author': quote.css('small.author::text').extract_first(),
            }

        next_page = response.css('li.next a::attr(href)').extract_first()
        if next_page is not None:
            yield response.follow(next_page, self.parse)
```

如果您将`tag=humor`参数传递给此蜘蛛，您会注意到它只会访问`humor`标记中的URL ，例如`http://quotes.toscrape.com/tag/humor` 。

您可以了解更多关于此处理蜘蛛参数。

后续步骤

本教程仅介绍了Scrapy的基础知识，但这里还没有提到很多其他功能。检查还有什么？Scrapy中的一节 一览快速概述最重要的章节。

您可以继续阅读基本概念部分，以了解有关命令行工具，蜘蛛，选择器以及教程未涵盖的其他内容的更多信息，例如对已删除数据进行建模。如果您更喜欢使用示例项目，请查看“ 示例”部分。