

接下来介绍一个简单的项目，完成一遍 Scrapy 抓取流程。通过这个过程，我们可以对 Scrapy 的基本用法和原理有大体了解。

## 本节目标

本节要完成的任务如下。

- 创建一个 Scrapy 项目。
- 创建一个 Spider 来抓取站点和处理数据。
- 通过命令行将抓取的内容导出。
- 将抓取的内容保存到 MongoDB 数据库。

本节抓取的目标站点为 <http://quotes.toscrape.com/>。

## 准备工作

我们需要安装好 Scrapy 框架、MongoDB 和 PyMongo 库。如果尚未安装，请参照之前几节的安装说明。

## 创建项目

创建一个 Scrapy 项目，项目文件可以直接用 scrapy 命令生成，命令如下所示：

```
scrapy startproject tutorial
```

这个命令可以在任意文件夹运行。如果提示权限问题，可以加 sudo 运行该命令。这个命令将会创建一个名为 tutorial 的文件夹，文件夹结构如下所示：

```
scrapy.cfg      # Scrapy 部署时的配置文件
tutorial        # 项目的模块，引入的时候需要从这里引入
  __init__.py   # 模块的初始化文件
  items.py      # Items 的定义，定义爬取的数据结构
  middlewares.py # Middlewares 的定义，定义爬取时的中间件
  pipelines.py  # Pipelines 的定义，定义数据管道
  settings.py   # 配置文件
  spiders       # 放置 Spiders 的文件夹
  __init__.py
```

## 创建 Spider

Spider 是自己定义的类，Scrapy 用它从网页里抓取内容，并解析抓取的结果。不过这个类必须继承 Scrapy 提供的 Spider 类 scrapy.Spider，还要定义 Spider 的名称和起始请求，以及怎样处理爬取后的结果的方法。

你也可以使用命令行创建一个 Spider。比如要生成 Quotes 这个 Spider，可以执行如下命令：

```
cd tutorial
scrapy genspider quotes
```

进入刚才创建的 tutorial 文件夹，然后执行 genspider 命令。第一个参数是 Spider 的名称，第二个参数是网站域名。执行完毕之后，spiders 文件夹中多了一个 quotes.py，它就是刚刚创建的 Spider，内容如下所示：

```
import scrapy

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

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

这里有三个属性——name、allowed\_domains 和 start\_urls，还有一个方法 parse。

- name: 它是每个项目唯一的名字，用来区分不同的 Spider。
- allowed\_domains: 它是允许爬取的域名，如果初始或后续的请求链接不是这个域名下的，则请求链接会被过滤掉。
- start\_urls: 它包含了 Spider 在启动时爬取的 url 列表，初始请求是由它来定义的。
- parse: 它是 Spider 的一个方法。默认情况下，被调用时 start\_urls 里面的链接构成的请求完成下载执行后，返回的响应就会作为唯一的参数传递给这个函数。该方法负责解析返回的响应、提取数据或者进一步生成要处理的请求。

## 创建 Item

Item 是保存爬取数据的容器，它的使用方法和字典类似。不过，相比字典，Item 多了额外的保护机制，可以避免拼写错误或者定义字段错误。

创建 Item 需要继承 scrapy.Item 类，并且定义类型为 scrapy.Field 的字段。观察目标网站，我们可以获取到的内容有 text、author、tags。

定义 Item，此时将 items.py 修改如下：

```
import scrapy

class QuoteItem(scrapy.Item):

    text = scrapy.Field()
    author = scrapy.Field()
    tags = scrapy.Field()
```

这里定义了三个字段，将类的名称修改为 QuoteItem，接下来爬取时我们会使用到这个 Item。

## 解析 Response

前面我们看到，parse 方法的参数 response 是 start\_urls 里面的链接爬取后的结果。所以在 parse 方法中，我们可以直接对 response 变量包含的内容进行解析，比如浏览请求结果的网页源代码，或者进一步分析源代码内容，或者找出结果中的链接而得到下一个请求。

我们可以看到网页中既有我们想要的结果，又有下一页的链接，这两部分内容我们都要进行处理。

首先看看网页结构，如图所示。每一页都有多个 class 为 quote 的区块，每个区块内都包含 text、author、tags。那么我们先找出所有的 quote，然后提取每一个 quote 中的内容。



```

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

def parse(self, response):
    quotes = response.css('.quote')
    for quote in quotes:
        item = QuoteItem()
        item['text'] = quote.css('.text::text').extract_first()
        item['author'] = quote.css('.author::text').extract_first()
        item['tags'] = quote.css('.tags .tag::text').extract()
        yield item

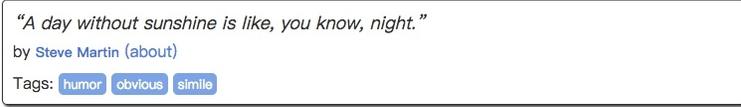
```

如此一来，首页的所有内容被解析出来，并被赋值成了一个 `QuoteItem`。

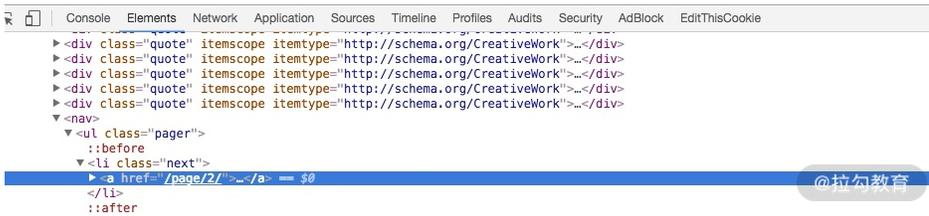
## 后续 Request

上面的操作实现了从初始页面抓取内容。那么，下一页的内容该如何抓取？这就需要我们从当前页面中找到信息来生成下一个请求，然后在下一个请求的页面里找到信息再构造下一个请求。这样循环往复迭代，从而实现整站的爬取。

将刚才的页面拉到最底部，如图所示。



Next →



有一个 `Next` 按钮，查看一下源代码，可以发现它的链接是 `/page/2/`，实际上全链接就是：<http://quotes.toscrape.com/page/2/>，通过这个链接我们就可以构造下一个请求。

构造请求时需要用到 `scrapy.Request`。这里我们传递两个参数——`url` 和 `callback`，这两个参数的说明如下。

- **url**: 它是请求链接。
- **callback**: 它是回调函数。当指定了该回调函数的请求完成之后，获取到响应，引擎会将该响应作为参数传递给这个回调函数。回调函数进行解析或生成下一个请求，回调函数如上文的 `parse()` 所示。

由于 `parse` 就是解析 `text`、`author`、`tags` 的方法，而下一页的结构和刚才已经解析的页面结构是一样的，所以我们可以再次使用 `parse` 方法来做页面解析。

接下来我们要做的就是利用选择器得到下一页链接并生成请求，在 `parse` 方法后追加如下的代码：

```

next = response.css('.pager .next a::attr(href)').extract_first()
url = response.urljoin(next)
yield scrapy.Request(url=url, callback=self.parse)

```

第一句代码首先通过 `CSS` 选择器获取下一个页面的链接，即要获取 `a` 超链接中的 `href` 属性。这里用到了 `::attr(href)` 操作。然后再调用 `extract_first` 方法获取内容。

第二句代码调用了 `urljoin` 方法，`urljoin()` 方法可以将相对 `URL` 构造成一个绝对的 `URL`。例如，获取到的下一页地址是 `/page/2`，`urljoin` 方法处理后得到的结果就是：<http://quotes.toscrape.com/page/2/>。

第三句代码通过 `url` 和 `callback` 变量构造了一个新的请求，回调函数 `callback` 依然使用 `parse` 方法。这个请求完成后，响应会重新经过 `parse` 方法处理，得到第二页的解析结果，然后生成第二页的下一页，也就是第三页的请求。这样爬虫就进入了一个循环，直到最后一页。

通过几行代码，我们就轻松实现了一个抓取循环，将每个页面的结果抓取下来了。现在，改写之后的整个 `Spider` 类如下所示：

```

import scrapy
from tutorial.items import QuoteItem

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

    def parse(self, response):
        quotes = response.css('.quote')
        for quote in quotes:
            item = QuoteItem()
            item['text'] = quote.css('.text::text').extract_first()
            item['author'] = quote.css('.author::text').extract_first()
            item['tags'] = quote.css('.tags .tag::text').extract()
            yield item

        next = response.css('.pager .next a::attr("href")').extract_first()
        url = response.urljoin(next)
        yield scrapy.Request(url=url, callback=self.parse)

```

## 运行

接下来，进入目录，运行如下命令：

```
scrapy crawl quotes
```

就可以看到 `Scrapy` 的运行结果了。

```

2020-02-19 13:37:20 [scrapy.utils.log] INFO: Scrapy 1.3.0 started (bot: tutorial)
2020-02-19 13:37:20 [scrapy.utils.log] INFO: Overridden settings: {'NEWSPIDER_MODULE': 'tutorial.spiders', 'SPIDER_MODULES': ['tutorial.spiders'], 'ROBOTSTXT_OBEY': True, 'BOT_NAME': 'tutorial'}
2020-02-19 13:37:20 [scrapy.middleware] INFO: Enabled extensions:
['scrapy.extensions.logstats.LogStats',
 'scrapy.extensions.telnet.TelnetConsole',
 'scrapy.extensions.corestats.CoreStats']
2020-02-19 13:37:20 [scrapy.middleware] INFO: Enabled downloader middlewares:
['scrapy.downloadermiddlewares.robotstxt.RobotsTxtMiddleware',
 'scrapy.downloadermiddlewares.httppath.HttpAuthMiddleware',
 'scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMiddleware',
 'scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMiddleware',
 'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware',
 'scrapy.downloadermiddlewares.retry.RetryMiddleware',
 'scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware',

```

```
'scrapy.downloadermiddlewares.httppcompression.HttpCompressionMiddleware',
'scrapy.downloadermiddlewares.redirect.RedirectMiddleware',
'scrapy.downloadermiddlewares.cookies.CookiesMiddleware',
'scrapy.downloadermiddlewares.stats.DownloaderStats']
2020-02-19 13:37:20 [scrapy.middleware] INFO: Enabled spider middlewares:
['scrapy.spidermiddlewares.httperror.HttpErrorMiddleware',
'scrapy.spidermiddlewares.offsite.OffsiteMiddleware',
'scrapy.spidermiddlewares.referrer.RefererMiddleware',
'scrapy.spidermiddlewares.urllength.UrlLengthMiddleware',
'scrapy.spidermiddlewares.depth.DepthMiddleware']
2020-02-19 13:37:20 [scrapy.middleware] INFO: Enabled item pipelines:
[]
2020-02-19 13:37:20 [scrapy.core.engine] INFO: Spider opened
2020-02-19 13:37:20 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
2020-02-19 13:37:20 [scrapy.extensions.telnet] DEBUG: Telnet console listening on 127.0.0.1:6023
2020-02-19 13:37:21 [scrapy.core.engine] DEBUG: Crawled (404) <GET http://quotes.toscrape.com/robots.txt> (referer: None)
2020-02-19 13:37:21 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://quotes.toscrape.com/> (referer: None)
2020-02-19 13:37:21 [scrapy.core.scraper] DEBUG: Scraped from <200 http://quotes.toscrape.com/>
{'author': u'Albert Einstein',
'tags': [u'change', u'deep-thoughts', u'thinking', u'world'],
'text': u'\u201cThe world as we have created it is a process of our thinking. It cannot be changed without changing our thinking.\u201d'}
2020-02-19 13:37:21 [scrapy.core.scraper] DEBUG: Scraped from <200 http://quotes.toscrape.com/>
{'author': u'J.K. Rowling',
'tags': [u'abilities', u'choices'],
'text': u'\u201cIt is our choices, Harry, that show what we truly are, far more than our abilities.\u201d'}
...
2020-02-19 13:37:27 [scrapy.core.engine] INFO: Closing spider (finished)
2020-02-19 13:37:27 [scrapy.statscollectors] INFO: Dumping Scrapy stats:
{'downloader/request_bytes': 2859,
'downloader/request_count': 11,
'downloader/request_method_count/GET': 11,
'downloader/response_bytes': 24871,
'downloader/response_count': 11,
'downloader/response_status_count/200': 10,
'downloader/response_status_count/404': 1,
'dupefilter/filtered': 1,
'finish_reason': 'finished',
'finish_time': datetime.datetime(2017, 2, 19, 5, 37, 27, 227438),
'item_scraped_count': 100,
'log_count/DEBUG': 113,
'log_count/INFO': 7,
'request_depth_max': 10,
'response_received_count': 11,
'scheduler/dequeued': 10,
'scheduler/dequeued/memory': 10,
'scheduler/enqueued': 10,
'scheduler/enqueued/memory': 10,
'start_time': datetime.datetime(2017, 2, 19, 5, 37, 20, 321557)}
2020-02-19 13:37:27 [scrapy.core.engine] INFO: Spider closed (finished)
```

这里只是部分运行结果，中间一些抓取结果已省略。

首先，Scrapy 输出了当前的版本号，以及正在启动的项目名称。接着输出了当前 `settings.py` 中一些重写后的配置。然后输出了当前所应用的 `Middlewares` 和 `Pipelines`。`Middlewares` 默认是启用的，可以在 `settings.py` 中修改。`Pipelines` 默认是空，同样也可以在 `settings.py` 中配置。后面会对它们进行讲解。

接下来就是输出各个页面的抓取结果了，可以看到爬虫一边解析，一边翻页，直至将所有内容抓取完毕，然后终止。

最后，Scrapy 输出了整个抓取过程的统计信息，如请求的字节数、请求次数、响应次数、完成原因等。

整个 Scrapy 程序成功运行。我们通过非常简单的代码就完成了对一个网站内容的爬取，这样相比之前一点点写程序简洁很多。

## 保存到文件

运行完 Scrapy 后，我们只在控制台看到了输出结果。如果想保存结果该怎么办呢？

要完成这个任务其实不需要任何额外的代码，Scrapy 提供的 `Feed Exports` 可以轻松将抓取结果输出。例如，我们想将上面的结果保存成 JSON 文件，可以执行如下命令：

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

命令运行后，项目内多了一个 `quotes.json` 文件，文件包含了刚才抓取的所有内容，内容是 JSON 格式。

另外我们还可以每一个 `Item` 输出一行 JSON，输出后缀为 `jl`，为 `jsonline` 的缩写，命令如下所示：

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

或

```
scrapy crawl quotes -o quotes.jsonlines
```

输出格式还支持很多种，例如 `csv`、`xml`、`pickle`、`marshal` 等，还支持 `ftp`、`s3` 等远程输出，另外还可以通过自定义 `ItemExporter` 来实现其他的输出。

例如，下面命令对应的输出分别为 `csv`、`xml`、`pickle`、`marshal` 格式以及 `ftp` 远程输出：

```
scrapy crawl quotes -o quotes.csv
scrapy crawl quotes -o quotes.xml
scrapy crawl quotes -o quotes.pickle
scrapy crawl quotes -o quotes.marshal
scrapy crawl quotes -o ftp://user:pass@ftp.example.com/path/to/quotes.csv
```

其中，`ftp` 输出需要正确配置用户名、密码、地址、输出路径，否则会报错。

通过 Scrapy 提供的 `Feed Exports`，我们可以轻松地输出抓取结果到文件。对于一些小型项目来说，这应该足够了。不过如果想要更复杂的输出，如输出到数据库等，我们可以使用 `Item Pipeline` 来完成。

## 使用 Item Pipeline

如果想进行更复杂的操作，如将结果保存到 MongoDB 数据库，或者筛选某些有用的 `Item`，则我们可以定义 `Item Pipeline` 来实现。

`Item Pipeline` 为项目管道。当 `Item` 生成后，它会自动被送到 `Item Pipeline` 进行处理，我们常用 `Item Pipeline` 来做如下操作。

- 清洗 HTML 数据；
- 验证爬取数据，检查爬取字段；
- 查重并丢弃重复内容；
- 将爬取结果储存在数据库。

要实现 `Item Pipeline` 很简单，只需要定义一个类并实现 `process_item` 方法即可。启用 `Item Pipeline` 后，`Item Pipeline` 会自动调用这个方法。`process_item` 方法必须返回包含数据的字典或 `Item` 对象，或者抛出 `DroppedItem` 异常。

`process_item` 方法有两个参数。一个参数是 `item`，每次 `Spider` 生成的 `Item` 都会作为参数传递过来。另一个参数是 `spider`，就是 `Spider` 的实例。

接下来，我们实现一个 `Item Pipeline`，筛掉 `text` 长度大于 50 的 `Item`，并将结果保存到 MongoDB。

修改项目里的 `pipelines.py` 文件，之前用命令行自动生成的文件内容可以删掉，增加一个 `TextPipeline` 类，内容如下所示：

```

from scrapy.exceptions import DropItem

class TextPipeline(object):
    def __init__(self):
        self.limit = 50

    def process_item(self, item, spider):
        if item['text']:
            if len(item['text']) > self.limit:
                item['text'] = item['text'][0:self.limit].rstrip() + '...'
            return item
        else:
            return DropItem('Missing Text')

```

这段代码在构造方法里定义了限制长度为 50，实现了 `process_item` 方法，其参数是 `item` 和 `spider`。首先该方法判断 `item` 的 `text` 属性是否存在，如果不存在，则抛出 `DropItem` 异常；如果存在，再判断长度是否大于 50，如果大于，那就截断然后拼接省略号，再将 `item` 返回即可。

接下来，我们将处理后的 `item` 存入 MongoDB，定义另外一个 Pipeline。同样在 `pipelines.py` 中，我们实现另一个类 `MongoPipeline`，内容如下所示：

```

import pymongo

class MongoPipeline(object):
    def __init__(self, mongo_uri, mongo_db):
        self.mongo_uri = mongo_uri
        self.mongo_db = mongo_db

    @classmethod
    def from_crawler(cls, crawler):
        return cls(mongo_uri=crawler.settings.get('MONGO_URI'),
                  mongo_db=crawler.settings.get('MONGO_DB'))

    def open_spider(self, spider):
        self.client = pymongo.MongoClient(self.mongo_uri)
        self.db = self.client[self.mongo_db]

    def process_item(self, item, spider):
        name = item.__class__.__name__
        self.db[name].insert(dict(item))
        return item

    def close_spider(self, spider):
        self.client.close()

```

`MongoPipeline` 类实现了 API 定义的另外几个方法。

- `from_crawler`: 这是一个类方法，用 `@classmethod` 标识，是一种依赖注入的方式，方法的参数就是 `crawler`，通过 `crawler` 这个参数我们可以拿到全局配置的每个配置信息，在全局配置 `settings.py` 中我们可以定义 `MONGO_URI` 和 `MONGO_DB` 来指定 MongoDB 连接需要的地址和数据库名称，拿到配置信息之后返回类对象即可。所以这个方法的定义主要是用来获取 `settings.py` 中的配置的。
- `open_spider`: 当 `Spider` 被开启时，这个方法被调用。在这里主要进行了一些初始化操作。
- `close_spider`: 当 `Spider` 被关闭时，这个方法会调用，在这里将数据库连接关闭。

最主要的 `process_item` 方法则执行了数据插入操作。

定义好 `TextPipeline` 和 `MongoPipeline` 这两个类后，我们需要在 `settings.py` 中使用它们。MongoDB 的连接信息还需要定义。

我们在 `settings.py` 中加入如下内容：

```

ITEM_PIPELINES = {
    'tutorial.pipelines.TextPipeline': 300,
    'tutorial.pipelines.MongoPipeline': 400,
}
MONGO_URI='localhost'
MONGO_DB='tutorial'

```

赋值 `ITEM_PIPELINES` 字典，键名是 Pipeline 的类名称，键值是调用优先级，是一个数字，数字越小则对应的 Pipeline 越先被调用。

再重新执行爬取，命令如下所示：

```
scrapy crawl quotes
```

爬取结束后，MongoDB 中创建了一个 `tutorial` 的数据库、`QuoteItem` 的表，如图所示。

长的 `text` 已经被处理并追加了省略号，短的 `text` 保持不变，`author` 和 `tags` 也都相应保存。

## 代码

本节代码地址：<https://github.com/Python3WebSpider/ScrapyTutorial>。

## 结语

我们通过抓取 Quotes 网站完成了整个 Scrapy 的简单入门。但这只是冰山一角，还有很多内容等待我们去探索。

