## 介绍

Scrapy 使用 Python 实现的一个开源爬虫框架。秉承着 “Don’t Repeat Yourself” 的原则，Scrapy 提供了一套编写爬虫的基础框架和编写过程中常见问题的一些解决方案。Scrapy 主要拥有下面这些功能和特点：

* 内置数据提取器（Selector），支持XPath 和 Scrapy 自己的 CSS Selector 语法，并且支持正则表达式，方便从网页提取信息。
* 交互式的命令行工具，方便测试 Selector 和 debugging 爬虫。
* 支持将数据导出为 JSON，CSV，XML 格式。
* 内置了很多拓展和中间件用于处理：
  + cookies 和 session
  + HTTP 的压缩，认证，缓存
  + robots.txt
  + 爬虫深度限制
* 可推展性强，运行自己编写特定功能的插件

除了列出的这些，还有很多小功能，比如内置的文件、图片下载器等等。另外，Scrapy 基于 twisted 这个高性能的事件驱动网络引擎框架，也就是说，Scrapy 爬虫拥有很高的性能。

下面的内容我们来实现一个爬取实验楼所有课程信息的爬虫。

### 知识点

* scrapy 爬虫框架介绍
* scrapy 框架安装
* 数据提取器：CSS 和 XPATH
* scrapy shell
* 正则表达式数据提取
* start\_urls

## 安装

为了方便管理 Python 版本和依赖包，首先在 Code 下创建一个 python3.5 的虚拟环境（实验环境已经安装了 virtualenv）：

virtualenv -p python3.5 venv

激活环境：

. venv/bin/activate

后面的 scrapy 爬虫实验都基于这个虚拟环境。

安装 scrapy:

pip3 install scrapy

现在在命令行输入 scrapy，出现下面的内容说明已经安装成功了。

Scrapy 1.4.0 - no active project

Usage:

scrapy <command> [options] [args]

Available commands:

bench Run quick benchmark test

fetch Fetch a URL using the Scrapy downloader

genspider Generate new spider using pre-defined templates

runspider Run a self-contained spider (without creating a project)

settings Get settings values

shell Interactive scraping console

startproject Create new project

version Print Scrapy version

view Open URL in browser, as seen by Scrapy

[ more ] More commands available when run from project directory

Use "scrapy <command> -h" to see more info about a command

从中我们也可以看到 scrapy 提供的命令行命令和它的功能简介。

Scrapy 安装操作视频：

## 数据提取器

在开始编写爬虫前，我们先来学习一下 scrapy 的数据提取器（Selector），因为爬虫的本质就是为了获取数据，所以在编写爬虫的过程中需要编写很多数据提取的代码。

scrapy 内置两种数据提取语法： CSS 和 XPath 。下面通过例子来看看怎么使用，有这样一个 HTML 文件：

<html>

<head>

<base href='http://example.com/' />

<title>Example website</title>

</head>

<body>

<div id='images'>

<a href='image1.html'>Name: My image 1 <br /><img src='image1\_thumb.jpg' /></a>

<a href='image2.html'>Name: My image 2 <br /><img src='image2\_thumb.jpg' /></a>

<a href='image3.html'>Name: My image 3 <br /><img src='image3\_thumb.jpg' /></a>

<a href='image4.html'>Name: My image 4 <br /><img src='image4\_thumb.jpg' /></a>

<a href='image5.html'>Name: My image 5 <br /><img src='image5\_thumb.jpg' /></a>

</div>

</body></html>

这是 scrapy 官方提供的一个网页，方便我们练习 Selector，它的地址是

http://doc.scrapy.org/en/latest/\_static/selectors-sample1.html

### scrapy shell

scrapy shell 提供了一个交互式的 Python 环境方便我们测试和debug 爬虫，使用方法是

scrapy shell [url]

需要提供一个网页的 url，执行命令后，scrapy 会自动去下载这个 url 对应的网页，将结果封装为 scrapy 内部的一个 response 对象并注入到 python shell 中，在这个 response 对象上，可以直接使用 scrapy 内置的css 和 xpath 数据提取器。

运行下面的命令下载上面的网页并进入 shell：

scrapy shell http://doc.scrapy.org/en/latest/\_static/selectors-sample1.html

对于网页的源代码分析，推荐使用 Chrome 浏览器，在页面上的元素右键选择 检查 可以得到源代码，在源代码上可以拷贝出 xpath 路径。

### CSS Selector

顾名思义，css selector 就是 css 的语法来定位标签。例如要提取例子网页中 ID 为 images 的 div 下所有 a 标签的文本，使用 css 语法可以这样写：

>>> response.css('div#images a::text').extract()

['Name: My image 1 ', 'Name: My image 2 ', 'Name: My image 3 ', 'Name: My image 4 ', 'Name: My image 5 ']

div#images 表示 id 为 images 的 div，如果是类名为 images，这里就是 div.images。div a 表示该 div 下所有 a 标签，::text 表示提取文本，extract 函数执行提取操作，返回一个**列表**。如果只想要列表中第一个 a 标签下的文本，可以使用 extract\_first 函数：

>>> response.css('div#images a::text').extract\_first()'Name: My image 1 '

extract\_first() 方法支持对没有匹配到的元素提供一个默认值：

>>> response.css('div#images p::text').extract\_first(default='默认值')'默认值'

div#images 下面并没有 p 标签，所以会返回提供的默认值。如果不提供 default 值的话会返回 None。

如果要提取所有 a 标签的 href 链接，可以这样写：

>>> response.css('div#images a::attr(href)').extract()

['image1.html', 'image2.html', 'image3.html', 'image4.html', 'image5.html']

不只是 href，任何标签的任意属性都可以用 attr() 提取。基于上面的知识，就能轻松写出提取所有图片的链接地址：

>>> response.css('div#images a img::attr(src)').extract()

['image1\_thumb.jpg', 'image2\_thumb.jpg', 'image3\_thumb.jpg', 'image4\_thumb.jpg', 'image5\_thumb.jpg']

如果 div 中有多个 class 的情况，用 css 提取器可以写为 div[class="class1 class2"]

CSS Selector 操作视频：

### XPath

XPath 是一门路径提取语言，常用于从 html/xml 文件中提取信息。它的基规则如下：

| **表达式** | **描述** |
| --- | --- |
| nodename | 选取此节点的所有子节点。 |
| / | 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。 |

用 scrapy 的 xpath 语法提取 div#images a 的所有文本，可以这样写：

>>> response.xpath('//div[@id="images"]/a/text()').extract()

['Name: My image 1 ', 'Name: My image 2 ', 'Name: My image 3 ', 'Name: My image 4 ', 'Name: My image 5 ']

// 表示当前文本，div[@id="images"] 表示 id 为 images 的 div，通过属性来选择一个标签，要以 tag[@attr="value"] 的方式表示。

用 XPath 语法提取所有图片链接：

>>> response.xpath('//div[@id="images"]/a/img/@src').extract()

['image1\_thumb.jpg', 'image2\_thumb.jpg', 'image3\_thumb.jpg', 'image4\_thumb.jpg', 'image5\_thumb.jpg']

比较俩种语法，CSS 语法比较简单，对新手比较友好，CSS 语法在底层其实是被转换成 XPath 的。XPath 虽然有一点学习成本，但是功能更强大，性能也要好一点。

### re 和 re\_first 方法

除了 extract() 和 extract\_first()方法， 还有 re() 和 re\_first() 方法可以用于 css() 或者 xpath() 方法返回的对象。

使用 extract() 直接提取的内容可能并不符合格式要求，比如上面获取的第一个 a 标签的 text 是这样的：Name: My image 1，现在要求不要开头的 Name: 和结尾的空格，这时候就可以使用 re() 替代 extract 方法，使用正则表达式对提取的内容做进一步的处理：

>>> response.css('div#images a::text').re('Name: (.+) ')

['My image 1', 'My image 2', 'My image 3', 'My image 4', 'My image 5']

re() 方法中定义的正则表达式会作用到每个提取到的文本中，只保留正则表达式中的子模式匹配到的内容，也就是 () 内的匹配内容。

re\_first() 方法支持只作用于第一个文本：

>>> response.css('div#images a::text').re\_first('Name: (.+) ')'My image 1'

## 实战

下面我们使用 scrapy 写一个爬虫，爬取实验楼所有课程名称、简介、类型和学习人数信息，并保存为 JSON 文本。

在 Code 下新建 shiyanlou\_courses\_spider.py 文件，写入 scrapy 爬虫的基本机构：

# -\*- coding:utf-8 -\*-import scrapy

class ShiyanlouCoursesSpider(scrapy.Spider):

""" 所有 scrapy 爬虫需要写一个 Spider 类，这个类要继承 scrapy.Spider 类。在这个类中定义要请求的网站和链接、如何从返回的网页提取数据等等。

"""

# 爬虫标识符号，在 scrapy 项目中可能会有多个爬虫，name 用于标识每个爬虫，不能相同

name = 'shiyanlou-courses'

def start\_requests(self):

""" 需要返回一个可迭代的对象，迭代的元素是 `scrapy.Request` 对象，可迭代对象可以是一个列表或者迭代器，这样 scrapy 就知道有哪些网页需要爬取了。`scrapy.Request` 接受一个 url 参数和一个 callback 参数，url 指明要爬取的网页，callback 是一个回调函数用于处理返回的网页，通常是一个提取数据的 parse 函数。

"""

def parse(self, response):

""" 这个方法作为 `scrapy.Request` 的 callback，在里面编写提取数据的代码。scrapy 中的下载器会下载 `start\_reqeusts` 中定义的每个 `Request` 并且结果封装为一个 response 对象传入这个方法。

"""

pass

分析实验楼的课程页面可以看出，一共有 22 个课程页面，URL 模版是这样的：

https://www.shiyanlou.com/courses/?category=all&course\_type=all&fee=all&tag=all&page={}

这样就可以写出 start\_requests 方法：

def start\_requests(self):

# 课程列表页面 url 模版

url\_tmpl = 'https://www.shiyanlou.com/courses/?category=all&course\_type=all&fee=all&tag=all&page={}'

# 所有要爬取的页面

urls = (url\_tmpl.format(i) for i in range(1, 23))

# 返回一个生成器，生成 Request 对象，生成器是可迭代对象

for url in urls:

yield scrapy.Request(url=url, callback=self.parse)

scrapy 内部的下载器会下载每个 Request，然后将结果封装为 response 对象传入 parse 方法，这个对象和前面 scrapy shell 练习中的对象是一样的，也就是说你可以用 response.css() 或者 response.xpath() 来提取数据了。

通过分析实验楼课程页面的文档结构，以《Python 数据分析入门与进阶》课程为例，我们需要提取的数据主要包含在下面的 div 里面：

<div class="course-body">

<div class="course-name">Python 数据分析入门与进阶</div>

<div class="course-desc">在本训练营中，我们将学习怎么样使用 Python 进行数据分析。课程将从数据分析基础开始，一步步深入讲解。从 Python 的基础用法到数据分析的各种算法，并结合各种实例，讲解数据分析过程中的方方面面。</div>

<div class="course-footer">

<span class="course-per-num pull-left">

<i class="fa fa-users"></i>

133

</span>

<span class="course-bootcamp pull-right">训练营</span>

</div></div>

根据这个 div 可以用提取器写出 parse 方法：

def parse(self, response):

# 遍历每个课程的 div.course-body

for course in response.css('div.course-body'):

# 使用 css 语法对每个 course 提取数据

yield {

# 课程名称

'name': course.css('div.course-name::text').extract\_first(),

# 课程描述

'description': course.css('div.course-desc::text').extract\_first(),

# 课程类型，实验楼的课程有免费，会员，训练营三种，免费课程并没有字样显示，也就是说没有 span.pull-right 这个标签，没有这个标签就代表时免费课程，使用默认值 `免费｀就可以了。

'type': course.css('div.course-footer span.pull-right::text').extract\_first(default='Free'),

# 注意 // 前面的 .，没有点表示整个文档所有的 div.course-body，有 . 才表示当前迭代的这个 div.course-body

'students': course.xpath('.//span[contains(@class, "pull-left")]/text()[2]').re\_first('[^\d]\*(\d\*)[^\d]\*')

}

students 提取的课程学习人数，提取代码为什么要这样写，有几点需要注意一下。首先，我们要提取的数据在下面这个机构中的：

<span class="course-per-num pull-left">

<i class="fa fa-users"></i>

133

</span>

这个结构的文本包含了很多空白和换行，如果是以下面的 xpath 提取的话：

course.xpath('.//span[contains(@class, "pull-left")]/text()').extract()

结果是这样的，有俩个文本内容，第一个是<span>和 <i>之间的文本，第二个是 </i>和 </span>之间的文本：

['\n \t', '\n\t \n \t 151455\n \t\n\t\t\t\t']

text()[2]表示取第二个文本，re\_first('[^\d]\*(\d\*)[^\d]\*')表示只需要文本中的数字。

## 运行

按照上一步中的格式写好 spider 后，就能使用 scrapy 的 runspider 命令来运行爬虫了。

scrapy runspider shiyanlou\_courses\_spider.py -o data.json

注意这里输出得到的 data.json 文件中的中文显示成 unicode 编码的形式，所以看到感觉像是乱码，其实是正常的。

-o 参数表示打开一个文件，scrapy 默认会将结果序列化为 JSON 格式写入其中。爬虫运行完后，在当前目录打开 data.json 文件都能开到爬取到的数据了。

实验楼课程爬虫开发视频：

## start\_urls

scrapy.Spider 类已经有了一个默认的 start\_requests方法，我们的爬虫代码其实可以进一步简化，只提供需要爬取的 start\_urls，默认的 start\_requests 方法会根据 start\_urls 生成 Request 对象。所以，代码可以修改为：

import scrapy

class ShiyanlouCoursesSpider(scrapy.Spider):

name = 'shiyanlou-courses'

@property

def start\_urls(self):

""" start\_urls 需要返回一个可迭代对象，所以，你可以把它写成一个列表、元组或者生成器，这里用的是生成器

"""

url\_tmpl = 'https://www.shiyanlou.com/courses/?category=all&course\_type=all&fee=all&tag=all&page={}'

return (url\_tmpl.format(i) for i in range(1, 23))

def parse(self, response):

for course in response.css('div.course-body'):

yield {

'name': course.css('div.course-name::text').extract\_first(),

'description': course.css('div.course-desc::text').extract\_first(),

'type': course.css('div.course-footer span.pull-right::text').extract\_first(),

'students': course.xpath('.//span[contains(@class, "pull-left")]/text()[2]').re\_first('[^\d]\*(\d\*)[^\d]\*')

}