Scrapy 爬虫高级应用

简介

本节内容主要介绍使用 scrapy 进阶的知识和技巧，包括页面追随，图片下载， 组成 item 的数据在多个页面，模拟登录。

知识点

页面追随

图片下载

Item 包含多个页面数据

模拟登录

页面跟随

在前面实现课程爬虫和用户爬虫中，因为实验楼的课程和用户 url 都是通过 id 来构造的，所以可以轻松构造一批顺序的 urls 给爬虫。但是在很多网站中，url 并不是那么轻松构造的，更常用的方法是从一个或者多个链接（start\_urls）爬取页面后，再从页面中解析需要的链接继续爬取，不断循环。

下面是一个简单的例子，从实验楼课程编号为 63 的课程主页，从课程相关的进阶课程获取下一批要爬取的课程。用前面所学的知识就能够完成这个程序，在看下面的代码前可以思考下怎么实现。

结合前面所学的知识，你可能会写出类似这样的代码：

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

import scrapy

class CoursesFollowSpider(scrapy.Spider):

name = 'courses\_follow'

start\_urls = ['https://shiyanlou.com/courses/63']

def parse(self, response):

yield {

'name': response.xpath('//h4[@class="course-infobox-title"]/span/text()').extract\_first(),

'author': response.xpath('//div[@class="mooc-info"]/div[@class="name"]/strong/text()').extract\_first()

}

# 从返回的 response 解析出“进阶课程”里的课程链接，依次构造

# 请求，再将本函数指定为回调函数，类似递归

for url in response.xpath('//div[@class="sidebox-body course-content"]/a/@href').extract():

# 解析出的 url 是相对 url，可以手动将它构造为全 url

# 或者使用 response.urljoin() 函数

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

完成页面跟随的核心就是最后 for 循环的代码。使用response.follow 函数可以对 for 循环代码做进一步简化：

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

import scrapy

class CoursesFollowSpider(scrapy.Spider):

name = 'courses\_follow'

start\_urls = ['https://shiyanlou.com/courses/63']

def parse(self, response):

yield {

'name': response.xpath('//h4[@class="course-infobox-title"]/span/text()').extract\_first(),

'author': response.xpath('//div[@class="mooc-info"]/div[@class="name"]/strong/text()').extract\_first()

}

# 不需要 extract 了

for url in response.xpath('//div[@class="sidebox-body course-content"]/a/@href'):

# 不需要构造全 url 了

yield response.follow(url, callback=self.parse)

页面追随实现视频：

您还没有安装flash播放器，请点击这里安装

安装后重启浏览器即可播放视频

Play Video

图片下载

scrapy 内部内置了下载图片的 pipeline。下面以下载实验楼课程首页每个课程的封面图片为例展示怎么使用它。

首先需要在 items.py 中定义一个 item ，它包含俩个必要的字段：

class CourseImageItem(scrapy.Item):

# 要下载的图片 url 列表

image\_urls = scrapy.Field()

# 下载的图片会先放着这里

images = scrapy.Field()

运行 scrapy genspider courses\_image shiyanlou.com/courses 生成一个爬虫，爬虫的核心工作就是解析所有图片的链接到 CourseImageItem 的 image\_urls 中。

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

import scrapy

from shiyanlou.items import CourseImageItem

class CoursesImageSpider(scrapy.Spider):

name = 'courses\_image'

start\_urls = ['https://www.shiyanlou.com/courses/']

def parse(self, response):

item = CourseImageItem()

＃解析图片链接到 item

item['image\_urls'] = response.xpath('//div[@class="course-img"]/img/@src').extract()

yield item

代码完成后需要在 settings 中启动 scrapy 内置的图片下载 pipeline，因为 ITEM\_PIPELINES 里的 pipelines 会按顺序作用在每个 item 上，而我们不需要 ShiyanlouPipeline 作用在图片 item 上，所以要把它注释掉

ITEM\_PIPELINES = {

'scrapy.pipelines.images.ImagesPipeline': 100,

# 'shiyanlou.pipelines.ShiyanlouPipeline': 300

}

还需要配置图片存储的目录：

IMAGES\_STORE = 'images'

运行程序：

# 安装需要的 PIL 包

pip3 install image

# 执行图片下载爬虫

scrapy crawl courses\_image

scrapy 会将图片下载到 images/full 下面，保存的文件名是对原文件进行的 hash。为什么会有一个 full 目录呢？full 目录代表下载的图片的原尺寸的，因为 scrapy 可以配置改变下载图片的尺寸，比如在 settings 中给你添加下面的配置生成小图片：

IMAGES\_THUMBS = {

'small': (50, 50)

}

图片下载实现视频：

您还没有安装flash播放器，请点击这里安装

安装后重启浏览器即可播放视频

Play Video

组成 item 的数据在多个页面

在前面几节实现的爬虫中，组成 item 的数据全部都是在一个页面中获取的。但是在实际的爬虫项目中，经常会遇到从不同的页面抓取数据组成一个 item。下面通过一个例子展示怎么处理这种情况。

有一个需求，爬取实验楼课程首页所有课程的名称、封面图片链接和课程作者。课程名称和封面图片链接在课程主页就能爬到，课程作者只有点击课程，进入课程详情页面才能看到，怎么办呢？

scrapy 的解决方案是多级 request 与 parse。简单的说就是先请求课程首页，在回调函数 parse 中解析出课程名称和课程图片链接，然后在 parse 函数再构造一个请求到课程详情页面，再在处理课程详情页的回调函数中解析出课程作者。

首先在 items.py 中创建 相应的 Item 类：

class MultipageCourseItem(scrapy.Item):

name = scrapy.Field()

image = scrapy.Field()

author = scrapy.Field()

运行 scrapy genspider multipage shiyanlou.com/courses 生成一个爬虫，并修改代码如下：

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

import scrapy

from shiyanlou.items import MultipageCourseItem

class MultipageSpider(scrapy.Spider):

name = 'multipage'

start\_urls = ['https://www.shiyanlou.com/courses/']

def parse(self, response):

for course in response.css('a.course-box'):

item = MultipageCourseItem()

# 解析课程名称

item['name'] = course.xpath('.//div[@class="course-name"]/text()').extract\_first()

# 解析课程图片

item['image'] = course.xpath('.//img/@src').extract\_first()

# 构造课程详情页面的链接，爬取到的链接是相对链接，调用 urljoin 方法构造全链接

course\_url = response.urljoin(course.xpath('@href').extract\_first())

# 构造到课程详情页的请求，指定回调函数

request = scrapy.Request(course\_url, callback=self.parse\_author)

# 将未完成的 item 通过 meta 传入 parse\_author

request.meta['item'] = item

yield request

def parse\_author(self, response):

# 获取未完成的 item

item = response.meta['item']

# 解析课程作者

item['author'] = response.xpath('//div[@class="mooc-info"]/div[@class="name"]/strong/text()').extract\_first()

# item 构造完成，生成

yield item

关闭所有的 pipeline，运行爬虫，保存结果到文件中：

scrapy crawl multipage -o data.json

这部分的知识点不容易理解，可以参考下先前同学的一个提问来理解 https://www.shiyanlou.com/questions/50921

组成 item 的数据在多个页面操作视频：

您还没有安装flash播放器，请点击这里安装

安装后重启浏览器即可播放视频

Play Video

模拟登录

有些网页中的内容需要登录后才能看到，例如实验楼用户主页的中的这个模块

，只有在你自己的用户主页才能看见。

如果想要爬取登录后才能看到的内容就需要 scrapy 模拟出登录的状态再去抓取页面，解析数据。这个实验就是要模拟登录自己的主页，然后在自己的主页爬取图片中箭头所指的数据。

通常网站都会有一个 login 页面，实验楼的 login 页面网址是：https://www.shiyanlou.com/login。打开这个网页，查看源码，可以看到里面的登录表单的相关代码。

模拟登录抓取的过程类似上面介绍的多页面抓取，只不过是将第一个页面的抓取变为提交一个登录表单，登录成功后， scrapy 会带着返回的 cookie 进行下面的抓取，这样就能抓取到登录才能看到的内容。

这个实验不要创建项目，在 Code 下面创建一个 login\_spider.py 脚本就可以了。下面是程序的基本结构和流程：

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

import scrapy

class LoginSpiderSpider(scrapy.Spider):

name = 'login\_spider'

start\_urls = ['https://www.shiyanlou.com/login']

def parse(self, response):

""" 模拟登录的核心就在这里，scrapy 会下载 start\_urls 里的

登录页面，将 response 传到这里，然后调用 FormRequest

模拟构造一个 POST 登录请求。FormRequest 继承自 Request，

所以 Request 的参数对它适用。FormRequest 有一类方法 `from\_response` 用于快速构建 FormRequest 对象。from\_response 方法会从第一步返回的 response 中获取请求的 url，form 表单信息等等，我们只需要指定必要的表单数据和回调函数就可以了。

"""

return scrapy.FormRequest.from\_response(

# 第一个参数必须传入上一步返回的 response

response,

# 以字典结构传入表单数据

formdata={},

# 指定回调函数

callback=self.after\_login

)

def after\_login(self, response):

""" 登录之后的代码和普通的 scrapy 爬虫一样，构造 Request，指定 callback ...

"""

pass

def parse\_after\_login(self, response):

pass

基于这个代码结构很容易写出代码：

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

import scrapy

class LoginSpiderSpider(scrapy.Spider):

name = 'login\_spider'

start\_urls = ['https://www.shiyanlou.com/login']

def parse(self, response):

# 获取表单的 csrf\_token

csrf\_token = response.xpath('//div[@class="login-body"]//input[@id="csrf\_token"]/@value').extract\_first()

self.logger.info(csrf\_token)

return scrapy.FormRequest.from\_response(

response,

formdata={

'csrf\_token': csrf\_token,

# 这里要改为自己的邮箱和密码

'login': 'example@email.com',

'password': 'password',

},

callback=self.after\_login

)

def after\_login(self, response):

# 登录成功后构造一个访问自己主页的 scrapy.Request

# 记得把 url 里的 id 换成你自己的，这部分数据只能看到自己的

return [scrapy.Request(

url='https://www.shiyanlou.com/user/634/',

callback=self.parse\_after\_login

)]

def parse\_after\_login(self, response):

""" 解析实验次数和实验时间数据，他们都在 span.info-text 结构中。实验次数位于第 2 个，实验时间位于第 3 个。

"""

return {

'lab\_count': response.xpath('(//span[@class="info-text"])[2]/text()').re\_first('[^\d]\*(\d\*)[^\d\*]'),

'lab\_minutes': response.xpath('(//span[@class="info-text"])[3]/text()').re\_first('[^\d]\*(\d\*)[^\d\*]')

}

运行脚本：

scrapy runspider login\_spider.py -o user.json

运行结束后，只会在登录后才能看的数据就被抓取下来并保存在 user.json 中了。

模拟登录实验楼操作视频：

您还没有安装flash播放器，请点击这里安装

安装后重启浏览器即可播放视频

Play Video

总结

本节内容主要通过实验楼用户爬虫的代码实例介绍如何使用 scrapy 进阶的知识和技巧，包括页面追随，图片下载， 组成 item 的数据在多个页面，模拟登录等。

本节实验中涉及到的知识点：

页面追随

图片下载

Item 包含多个页面数据

模拟登录

来源: 实验楼

链接: https://www.shiyanlou.com/courses/996

本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播