Scrapy框架

参考书目：https://www.jianshu.com/p/b807653e97bb

1介绍和安装



**Scrapy运行流程大概如下：**

1. 首先，引擎从调度器中取出一个链接(URL)用于接下来的抓取
2. 引擎把URL封装成一个请求(Request)传给下载器，下载器把资源下载下来，并封装成应答包(Response)
3. 然后，爬虫解析Response
4. 若是解析出实体（Item）,则交给实体管道进行进一步的处理。
5. 若是解析出的是链接（URL）,则把URL交给Scheduler等待抓取

2HTML和XPATH

提取信息

**2.1css**

利用CSS进行提取：

>>> response.css('title')？/

//”ath='descendant-or-self::title' data='<title>Quotes to Scrape</title>'>]

只提取标题的文本：

>>> response.css('title::text').extract()

['Quotes to Scrape']

::text表示只提取文本，去掉的话，显示如下：

>>> response.css('title').extract()

['<title>Quotes to Scrape</title>']

因为返回对象是一个列表，只提取第一个的话，使用：

>>> response.css('title::text').extract\_first()

'Quotes to Scrape'

或者，使用序号：

>>> response.css('title::text')[0].extract()

'Quotes to Scrape'

前者更好，可以避免潜在的序号错误。

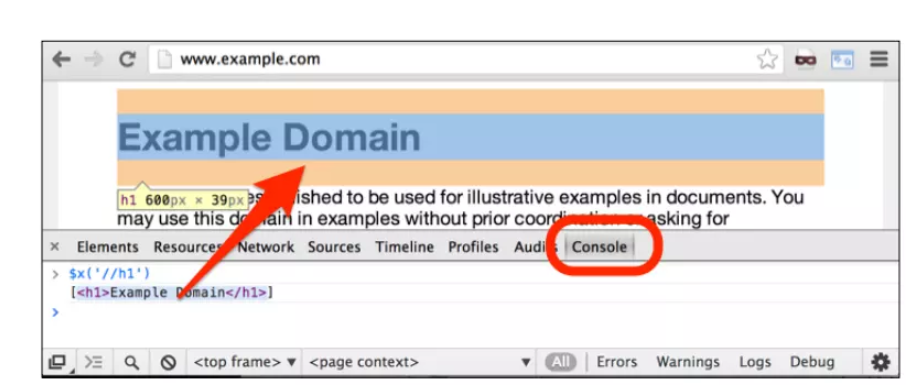
**2.2xpath**

>>> response.xpath('//title')

[<Selector xpath='//title' data='<title>Quotes to Scrape</title>'>]

>>> response.xpath('//title/text()').extract\_first() 'Quotes to Scrape'

在Chrome中使用XPath，在开发者工具中点击控制台标签，使用$x功能。例如，在网页[http://example.com/](https://link.jianshu.com?t=http:/example.com/" \t "_blank)的控制台，输入$x('//h1')，就可以移动到<h1>元素，如截图所示：



你在控制台中看到的是一个包含所选元素的JavaScript数组。如果你将光标移动到这个数组上，你可以看到被选择的元素被高亮显示。这个功能很有用。

下面展示一些XPath表达式的常见使用。先来看看在维基百科上是怎么使用的。维基百科的页面非常稳定，不会在短时间内改变排版。

* 取得id为firstHeading的div下的span的text：
* //h1[@id="firstHeading"]/span/text()
* 取得id为toc的div下的ul内的URL：
* //div[@id="toc"]/ul//a/@href
* 在任意class包含ltr和class包含skin-vector的元素之内，取得h1的text，这两个字符串可能在同一class内，或不在。
* //\*[contains(@class,"ltr") and contains(@class,"skin-vector")]//h1//text()

实际应用中，你会在XPath中频繁地使用class。在这几个例子中，你需要记住，因为CSS的板式原因，你会看到HTML的元素总会包含许多特定的class属性。这意味着，有的<div>的class是link，其他导航栏的<div>的class就是link active。后者是当前生效的链接，因此是可见或是用CSS特殊色高亮显示的。当抓取的时候，你通常是对含有某个属性的元素感兴趣的，就像之前的link和link active。XPath的contains( )函数就可以帮你选择包含某一class的所有元素。

* 选择class属性是infobox的table的第一张图片的URL：
* //table[@class="infobox"]//img[1]/@src
* 选择class属性是reflist开头的div下面的所有URL链接：
* //div[starts-with(@class,"reflist")]//a/@href
* 选择div下面的所有URL链接，并且这个div的下一个相邻元素的子元素包含文字References：
* //\*[text()="References"]/../following-sibling::div//a
* 取得所有图片的URL：
* //img/@src

**提前应对网页发生改变**

一些基本原则可以帮助我们降低表达式失效的概率：

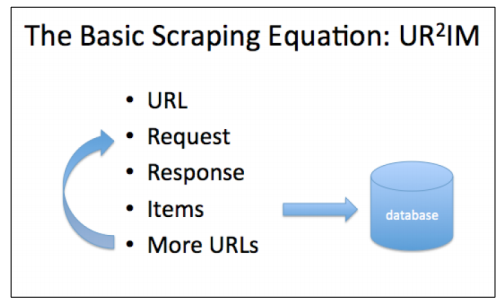
1避免使用数组序号

2用class抓取效果不一定好

3id通常是最可靠的

**3创建项目（在scrapy shell中进行）**

**UR2IM——基础抓取过程**  
每个网站都是不同的，对每个网站进行额外的研究不可避免，碰到特别生僻的问题，也许还要用Scrapy的邮件列表咨询。寻求解答，去哪里找、怎么找，前提是要熟悉整个过程和相关术语。Scrapy的基本过程，可以写成字母缩略语UR2IM，见下图。



url: 如果你想加载一个网页，你可以使用Scrapy终端，如下所示

* scrapy shell -s USER\_AGENT="Mozilla/5.0" <your url here e.g. http://www.gumtree.com/p/studios-bedsits-rent/...>

要进行调试，可以在Scrapy语句后面添加 –pdb

scrapy shell --pdb https:*//gumtree.com*

**请求和响应**

要打印reponse.body的前50个字母

>>> response.body[:50]

'<!DOCTYPE html>\n<html>\n<head>\n<meta charset="UTF-8"'

**抓取对象**

下一步是从响应文件中提取信息，输入到Item

只提取h1标签里的文字

>>> response.xpath('//h1/text()').extract()

[u'set unique family well']

**4一个scrapy项目**

1.1使用命令：

scrapy startproject  properties    
定义items：修改PropertiesItem这个类。修改文件properties/items.py

1. **from** scrapy.item **import** Item, Field
3. **class** PropertiesItem(Item):
4. # Primary fields
5. title = Field()
6. price = Field()
7. description = Field()
8. address = Field()
9. image\_URL = Field()
11. # Calculated fields
12. images = Field()
13. location = Field()
15. # Housekeeping fields
16. url = Field()
17. project = Field()
18. spider = Field()
19. server = Field()
20. date = Field()

编写爬虫：用scrapy genspider命令，

$ scrapy genspider basic web

使用模块中的模板“basic”创建了一个爬虫“basic”

properties.spiders.basic

刚才的命令是，生成一个名字是basic的默认文件，它的限制是在web上爬取URL。我们可以取消这个限制。这个爬虫使用的是basic这个模板。你可以用scrapy genspider –l查看所有的模板，然后用参数–t利用模板生成想要的爬虫

下面来开始编辑这个爬虫。start\_URL更改为在Scrapy命令行中使用过的URL。然后用爬虫事先准备的log()方法输出内容。

1. **import** scrapy
2. **class** BasicSpider(scrapy.Spider):
3. name = "basic"
4. allowed\_domains = ["web"]
5. start\_URL = (
6. 'http://web:9312/properties/property\_000000.html',
7. )
8. **def** parse(self, response):
9. self.log("title: %s" % response.xpath(
10. '//\*[@itemprop="name"][1]/text()').extract())
11. self.log("price: %s" % response.xpath(
12. '//\*[@itemprop="price"][1]/text()').re('[.0-9]+'))
13. self.log("description: %s" % response.xpath(
14. '//\*[@itemprop="description"][1]/text()').extract())
15. self.log("address: %s" % response.xpath(
16. '//\*[@itemtype="http://schema.org/'
17. 'Place"][1]/text()').extract())
18. self.log("image\_URL: %s" % response.xpath(
19. '//\*[@itemprop="image"][1]/@src').extract())

parse()：是spider的一个方法。被调用时，每个初始URL完成下载后生成的 Response 对象将会作为唯一的参数传递给该函数。该方法负责解析返回的数据(response data)，提取数据(生成item)以及生成需要进一步处理的URL的 Request 对象。

这个方法负责解析返回的数据、匹配抓取的数据(解析为 item )并跟踪更多URL。

来看另一个命令，scrapy parse。它可以让我们选择最合适的爬虫来解析URL。用—spider命令可以设定爬虫

$ scrapy parse --spider=basic http:*//web:9312/properties/property\_000001.html*

**填充一个项目**

首先，引入类PropertiesItem。它位于properties目录中的item.py文件，因此在模块properties.items中。它的导入命令是：

from properties.items import PropertiesItem

然后我们要实例化，并进行返回。这很简单。在parse()方法中，我们加入声明item = PropertiesItem()，它产生了一个新项目，然后为它分配表达式：

item['title'] = response.xpath('//\*[@itemprop="name"][1]/text()').extract()

最后，我们用return item返回项目。更新后的properties/spiders/basic.py文件如下：

1. **import** scrapy
2. **from** properties.items **import** PropertiesItem
3. **class** BasicSpider(scrapy.Spider):
4. name = "basic"
5. allowed\_domains = ["web"]
6. start\_URL = (
7. 'http://web:9312/properties/property\_000000.html',
8. )
9. **def** parse(self, response):
10. item = PropertiesItem()
11. item['title'] = response.xpath(
12. '//\*[@itemprop="name"][1]/text()').extract()
13. item['price'] = response.xpath(
14. '//\*[@itemprop="price"][1]/text()').re('[.0-9]+')
15. item['description'] = response.xpath(
16. '//\*[@itemprop="description"][1]/text()').extract()
17. item['address'] = response.xpath(
18. '//\*[@itemtype="http://schema.org/'
19. 'Place"][1]/text()').extract()
20. item['image\_URL'] = response.xpath(
21. '//\*[@itemprop="image"][1]/@src').extract()
22. **return** item
23. 。

**保存到文件**

1. $ scrapy crawl basic -o items.json
2. $ cat items.json
3. [{"price": ["334.39"], "address": ["Angel, London"], "description":
4. ["website court ... offered"], "image\_URL": ["../images/i01.jpg"],
5. "title": ["set unique family well"]}]
6. $ scrapy crawl basic -o items.jl
7. $ cat items.jl
8. {"price": ["334.39"], "address": ["Angel, London"], "description":
9. ["website court ... offered"], "image\_URL": ["../images/i01.jpg"],
10. "title": ["set unique family well"]}
11. $ scrapy crawl basic -o items.csv
12. $ cat items.csv
13. description,title,url,price,spider,image\_URL...
14. "...offered",set unique family well,,334.39,,../images/i01.jpg
15. $ scrapy crawl basic -o items.xml
16. $ cat items.xml
17. <?xml version="1.0" encoding="utf-8"?>
18. <items><item><price><value>334.39</value></price>...</item></items>

不用我们写任何代码，我们就可以用这些格式进行存储。Scrapy可以自动识别输出文件的后缀名，并进行输出。这段代码中涵盖了一些常用的格式。CSV和XML文件很流行，因为可以被Excel直接打开。JSON文件很流行是因为它的开放性和与JavaScript的密切关系。JSON和JSON Line格式的区别是.json文件是在一个大数组中存储JSON对象。这意味着如果你有一个1GB的文件，你可能必须现在内存中存储，然后才能传给解析器。相对的，.jl文件每行都有一个JSON对象，所以读取效率更高。

**清洗——项目加载器和杂务字段**

**创建协议**

整个项目的完整代码

1. **from** scrapy.loader.processors **import** MapCompose, Join
2. **from** scrapy.loader **import** ItemLoader
3. **from** properties.items **import** PropertiesItem
4. **import** datetime
5. **import** urlparse
6. **import** socket
7. **import** scrapy
9. **class** BasicSpider(scrapy.Spider):
10. name = "basic"
11. allowed\_domains = ["web"]
12. # Start on a property page
13. start\_URL = (
14. 'http://web:9312/properties/property\_000000.html',
15. )
16. **def** parse(self, response):
17. """ This function parses a property page.
18. @url http://web:9312/properties/property\_000000.html
19. @returns items 1
20. @scrapes title price description address image\_URL
21. @scrapes url project spider server date
22. """
23. # Create the loader using the response
24. l = ItemLoader(item=PropertiesItem(), response=response)
25. # Load fields using XPath expressions
26. l.add\_xpath('title', '//\*[@itemprop="name"][1]/text()',
27. MapCompose(unicode.strip, unicode.title))
28. l.add\_xpath('price', './/\*[@itemprop="price"][1]/text()',
29. MapCompose(**lambda** i: i.replace(',', ''),
30. float),
31. re='[,.0-9]+')
32. l.add\_xpath('description', '//\*[@itemprop="description"]'
33. '[1]/text()',
34. MapCompose(unicode.strip), Join())
35. l.add\_xpath('address',
36. '//\*[@itemtype="http://schema.org/Place"]'
37. '[1]/text()',
38. MapCompose(unicode.strip))
39. l.add\_xpath('image\_URL', '//\*[@itemprop="image"]'
40. '[1]/@src', MapCompose(
41. **lambda** i: urlparse.urljoin(response.url, i)))
42. # Housekeeping fields
43. l.add\_value('url', response.url)
44. l.add\_value('project', self.settings.get('BOT\_NAME'))
45. l.add\_value('spider', self.name)
46. l.add\_value('server', socket.gethostname())
47. l.add\_value('date', datetime.datetime.now())
48. **return** l.load\_item()

提取更多的URL

到目前为止，在爬虫的start\_URL中我们还是只加入了一条URL。因为这是一个元组，我们可以向里面加入多个URL，例如：

1. start\_URL = (
2. 'http://web:9312/properties/property\_000000.html',
3. 'http://web:9312/properties/property\_000001.html',
4. 'http://web:9312/properties/property\_000002.html',
5. )
6. 不够好。我们可以用一个文件当做URL源文件：
7. start\_URL = [i.strip() **for** i **in**
8. open('todo.URL.txt').readlines()]

**使用爬虫进行二维抓取**

**parse()方法进行水平和垂直的抓取**

1. **def** parse(self, response):
2. # Get the next index URL and yield Requests
3. next\_selector = response.xpath('//\*[contains(@class,'
4. '"next")]//@href')
5. **for** url **in** next\_selector.extract():
6. **yield** Request(urlparse.urljoin(response.url, url))
8. # Get item URL and yield Requests
9. item\_selector = response.xpath('//\*[@itemprop="url"]/@href')
10. **for** url **in** item\_selector.extract():
11. **yield** Request(urlparse.urljoin(response.url, url),
12. callback=self.parse\_item)

**用CrawlSpider二维抓取**

包含变量rules的parse()方法，以完成之前我们手写的内容。现在将start\_URL设定为索引首页，并将parse\_item()方法替换。这次不再使用parse()方法，而是将rules变成两个rules，一个负责水平抓取，一个负责垂直抓取：

1. rules = (
2. Rule(LinkExtractor(restrict\_xpaths='//\*[contains(@class,"next")]')),
3. Rule(LinkExtractor(restrict\_xpaths='//\*[@itemprop="url"]'),
4. callback='parse\_item')
5. )

**即代码改成如下**

1. class EasySpider(CrawlSpider):
2. name = 'easy'
3. allowed\_domains = ['web']
4. start\_URL = ['http://www.web/']
5. rules = (
6. Rule(LinkExtractor(restrict\_xpaths='//\*[contains(@class,"next")]')),
7. Rule(LinkExtractor(restrict\_xpaths='//\*[@itemprop="url"]'),
8. callback='parse\_item')
9. )
10. def parse\_item(self, response):

...

第二个举例

1.1 使用命令：创建工程

scrapy startproject tutorial

1.2在tutorial/spiders文件夹新建文件quotes\_spider.py，它的代码如下

1. **import** scrapy
3. **class** QuotesSpider(scrapy.Spider):
4. name = "quotes"
6. **def** start\_requests(self):
7. urls = [
8. 'http://quotes.toscrape.com/page/1/',
9. 'http://quotes.toscrape.com/page/2/',
10. ]
11. **for** url **in** urls:
12. **yield** scrapy.Request(url=url, callback=self.parse)
14. **def** parse(self, response):
15. page = response.url.split("/")[-2]
16. filename = 'quotes-%s.html' % page
17. with open(filename, 'wb') as f:
18. f.write(response.body)
19. self.log('Saved file %s' % filename)

start\_requests方法返回 scrapy.Request对象。每收到一个，就实例化一个Response对象，并调用和request绑定的调回方法（即parse），将response作为参数

1.3切换到根目录，运行爬虫

scrapy crawl quotes

根目录下会产生两个文件，quotes-1.html和quotes-2.html。

1.4**用爬虫提取信息**

使用Python的yield：

1. **import** scrapy
3. **class** QuotesSpider(scrapy.Spider):
4. name = "quotes"
5. start\_urls = [
6. 'http://quotes.toscrape.com/page/1/',
7. 'http://quotes.toscrape.com/page/2/',
8. ]
10. **def** parse(self, response):
11. **for** quote **in** response.css('div.quote'):
12. **yield** {
13. 'text': quote.css('span.text::text').extract\_first(),
14. 'author': quote.css('small.author::text').extract\_first(),
15. 'tags': quote.css('div.tags a.tag::text').extract(),
16. }

1.5保存数据

scrapy crawl quotes -o quotes.json

1.6提取下一页：

首先看下一页的链接：

1. <ul **class**="pager">
2. <li **class**="next">
3. <a href="/page/2/">Next <span aria-hidden="true">→</span></a>
4. </li>
5. </ul>

提取出来，只要href

1. >>> response.css('li.next a').extract\_first()
2. '<a href="/page/2/">Next <span aria-hidden="true">→</span></a>'
3. >>> response.css('li.next a::attr(href)').extract\_first()
4. '/page/2/'

利用urljoin生成完整的url，生成下一页的请求，就可以循环抓取了：

Css版本

1. **import** scrapy
3. **class** QuotesSpider(scrapy.Spider):
4. name = "quotes"
5. start\_urls = [
6. 'http://quotes.toscrape.com/page/1/',
7. ]
9. **def** parse(self, response):
10. **for** quote **in** response.css('div.quote'):
11. **yield** {
12. 'text': quote.css('span.text::text').extract\_first(),
13. 'author': quote.css('small.author::text').extract\_first(),
14. 'tags': quote.css('div.tags a.tag::text').extract(),
15. }
17. next\_page = response.css('li.next a::attr(href)').extract\_first()
18. **if** next\_page **is** **not** None:
19. next\_page = response.urljoin(next\_page)
20. **yield** scrapy.Request(next\_page, callback=self.parse)

xpath版本

1. **import** scrapy
3. **class** ToScrapeSpiderXPath(scrapy.Spider):
4. name = 'toscrape-xpath'
5. start\_urls = [
6. 'http://quotes.toscrape.com/',
7. ]
9. **def** parse(self, response):
10. **for** quote **in** response.xpath('//div[@class="quote"]'):
11. **yield** {
12. 'text': quote.xpath('./span[@class="text"]/text()').extract\_first(),
13. 'author': quote.xpath('.//small[@class="author"]/text()').extract\_first(),
14. 'tags': quote.xpath('.//div[@class="tags"]/a[@class="tag"]/text()').extract()
15. }
17. next\_page\_url = response.xpath('//li[@class="next"]/a/@href').extract\_first()
18. **if** next\_page\_url **is** **not** None:
19. **yield** scrapy.Request(response.urljoin(next\_page\_url))

第三个例子：<https://www.jianshu.com/p/f36460267ac2>

第四个例子: https://www.jianshu.com/p/078ad2067419

接下来用现在最流行的方式，移动应用，让你的数据面向公众。

**5选择移动应用框架**

**创建数据库和集合**

**用Scrapy导入数据**

**创建移动应用**

**创建数据库接入服务**

设定用户界面

**将数据映射到用户界面**

**映射数据字段和用户组件**

**测试、分享、生成app**

**6一个具有登录功能的爬虫**

模拟一个POST请求登录，我们用Scrapy中的类FormRequest来做。

**from** scrapy.http **import** FormRequest

我们然后将start\_URL替换为start\_requests()方法。这么做是因为在本例中，比起URL，我们要做一些自定义的工作。更具体地，用下面的函数，我们创建并返回一个FormRequest:

1. # Start with a login request
2. **def** start\_requests(self):
3. **return** [
4. FormRequest(
5. "http://web:9312/dynamic/login",
6. formdata={"user": "user", "pass": "pass"}
7. )]

有些网站还需要提供别的随机信息，比如nonce。你会发现这个页面的表单有一个叫做nonce的隐藏字段。当你提交表单时，你必须既要提供正确的用户名密码，还要提交正确的浏览器发给你的nonce值

1. **def** start\_requests(self):
2. **return** [
3. Request(
4. "http://web:9312/dynamic/nonce",
5. callback=self.parse\_welcome)
6. ]
7. # Post welcome page's first form with the given user/pass
8. **def** parse\_welcome(self, response):
9. **return** FormRequest.from\_response(
10. response,
11. formdata={"user": "user", "pass": "pass"}
12. )

它最大的功能是，一字不差地包含了表单中所有的隐藏字段。我们只需使用formdata参数，填入user和pass字段，并返回FormRequest

**7使用JSON APIs和AJAX页面的爬虫**

有时，你会发现网页的HTML找不到数据。

Python提供了一个强大的JSON解析库。当我们import json时，我们可以使用json.loads（response.body）解析JSON，并转换成等价的Python对象，语句、列表和字典。

1. start\_URL = (
2. 'http://web:9312/properties/api.json',
3. )

这里，Scrapy会打开这个URL并使用Response作为参数调用parse()方法。我们可以import json，使用下面的代码解析JSON：

1. **def** parse(self, response):
2. base\_url = "http://web:9312/properties/"
3. js = json.loads(response.body)
4. **for** item **in** js:
5. id = item["id"]
6. url = base\_url + "property\_%06d.html" % id
7. **yield** Request(url, callback=self.parse\_item)

8**在响应间传递参数**

把JSON APIs中的信息存储到Item中，

我们要做的就是在parse()方法产生的Request中进行设置。然后，我们可以从parse\_item()的的Response中取回。Request有一个名为meta的字典，在Response中可以直接访问。对于我们的例子，给字典设一个title值以存储从JSON对象的返回值：

2. title = item["title"]
3. **yield** Request(url, meta={"title": title},callback=self.parse\_item)

在parse\_item()中，我们可以使用这个值，而不用XPath表达式：

1. l.add\_value('title', response.meta['title'],
2. MapCompose(unicode.strip, unicode.title))

你会注意到，我们从调用add\_xpath()切换到add\_value()，因为对于这个字段不需要使用XPath。我们现在运行爬虫，就可以在PropertyItems中看到api.json中的标题了。

**9一个加速30倍的项目爬虫**

使用selector对象

1. **def** parse(self, response):
2. # Get the next index URL and yield Requests
3. next\_sel = response.xpath('//\*[contains(@class,"next")]//@href')
4. **for** url **in** next\_sel.extract():
5. **yield** Request(urlparse.urljoin(response.url, url))
6. # Iterate through products and create PropertiesItems
7. selectors = response.xpath(
8. '//\*[@itemtype="http://schema.org/Product"]')
9. **for** selector **in** selectors:
10. **yield** self.parse\_item(selector, response)

第一部分中用于产生下一条索引请求的代码没有变动。不同的地方是第二部分，我们重复使用选择器调用parse\_item()方法，而不是用yield创建请求。这和原先使用的源代码很像：

1. **def** parse\_item(self, selector, response):
2. # Create the loader using the selector
3. l = ItemLoader(item=PropertiesItem(), selector=selector)
4. # Load fields using XPath expressions
5. l.add\_xpath('title', './/\*[@itemprop="name"][1]/text()',
6. MapCompose(unicode.strip, unicode.title))
7. l.add\_xpath('price', './/\*[@itemprop="price"][1]/text()',
8. MapCompose(**lambda** i: i.replace(',', ''), float),
9. re='[,.0-9]+')
10. l.add\_xpath('description',
11. './/\*[@itemprop="description"][1]/text()',
12. MapCompose(unicode.strip), Join())
13. l.add\_xpath('address',
14. './/\*[@itemtype="http://schema.org/Place"]'
15. '[1]/\*/text()',
16. MapCompose(unicode.strip))
17. make\_url = **lambda** i: urlparse.urljoin(response.url, i)
18. l.add\_xpath('image\_URL', './/\*[@itemprop="image"][1]/@src',
19. MapCompose(make\_url))
20. # Housekeeping fields
21. l.add\_xpath('url', './/\*[@itemprop="url"][1]/@href',
22. MapCompose(make\_url))
23. l.add\_value('project', self.settings.get('BOT\_NAME'))
24. l.add\_value('spider', self.name)
25. l.add\_value('server', socket.gethostname())
26. l.add\_value('date', datetime.datetime.now())
27. **return** l.load\_item()

1.ItemLoader现在使用selector作为源，不使用Response。这么做可以让ItemLoader更便捷，可以让我们从特定的区域而不是整个页面抓取信息。

2.通过在前面添加“.”使XPath表达式变为相关XPath。

3.在response.url给我们列表页的URL之前，我们必须自己编辑Item的URL。然后，它才能返回我们抓取网页的URL。我们必须用.//\*[@itemprop="url"][1]/@href提取URL，然后将它用MapCompose转化为URL绝对路径。

**10可以抓取Excel文件的爬虫**

因为从文件中读取的URL是我们事先不了解的，所以使用一个start\_requests()方法。对于每一行，我们都会创建Request。我们还要从request,meta的csv存储字段名和XPath，以便在我们的parse()函数中使用。然后，我们使用Item和ItemLoader填充Item的字段。

1. **import** csv
2. **import** scrapy
3. **from** scrapy.http **import** Request
4. **from** scrapy.loader **import** ItemLoader
5. **from** scrapy.item **import** Item, Field
6. **class** FromcsvSpider(scrapy.Spider):
7. name = "fromcsv"
8. **def** start\_requests(self):
9. with open("todo.csv", "rU") as f:
10. reader = csv.DictReader(f)
11. **for** line **in** reader:
12. request = Request(line.pop('url'))
13. request.meta['fields'] = line
14. **yield** request
15. **def** parse(self, response):
16. item = Item()
17. l = ItemLoader(item=item, response=response)
18. **for** name, xpath **in** response.meta['fields'].iteritems():
19. **if** xpath:
20. item.fields[name] = Field()
21. l.add\_xpath(name, xpath)
22. **return** l.load\_item()

meta是一个字典，主要是用解析函数之间传递值，常见的情况是：在parse中给item某些字段提取了值，但是另外一些值需要在parse2中提取，这时候需要将parse中的item传到parse2方法中处理，显然无法直接给parse2设置而外参数。 Request对象接受一个meta参数，一个字典对象，同时Response对象有一个meta属性可以取到相应request传过来的meta

**11Scrapinghub**

我们往往需要周期性的进行抓取。我们可以用Amazon、RackSpace等服务商的云主机，但这需要一些设置、配置和维护。这时候就需要Scrapinghub

**注册、登录、创建项目**