12 用 ItemLoader 解决网页数据多样性的问题

更新时间：2019-06-14 14:35:06

> 学习这件事不在乎有没有人教你，最重要的是在于你自己有没有觉悟和恒心。
>
> —— 法布尔

像网易这类新闻门户已历了多年的迭代，虽然同属新闻但不同栏目的网页数据结构都不尽相同，本节将介绍应对这类在同一网站上出现用不同结构的页面显示同一类数据的情况下巧妙地运用`ItemLoader`提取数据的方法。

项目加载器提供了一种方便的机制来填充抓取的项目。即使可以使用自己的类似字典的API填充项目，项目加载器提供了一个更方便的 API，通过自动化一些常见的任务，如解析原始提取的数据，然后分配它从剪贴过程中填充他们。

换句话说，`Items` 是承载了数据本身的容器，而 `ItemLoader` 是负责数据的收集、处理、填充。

项目加载器旨在提供一种灵活，高效和容易的机制，通过爬虫或源格式（HTML，XML等）扩展和覆盖不同的字段解析规则，而不会成为维护的噩梦。

采用 `Processor` 与 `ItemLoader` 加载 Item 简化加载逻辑

为什么会有"维护的噩梦"？先来回顾一下网易爬虫的 `parse_item` 的代码：

```
def parse_item(self, response):
    item = NewsItem()
    selector = Selector(response)
    item['title'] = selector.css('#epContentLeft>h1::text').get() # 获取标题

    item['pub_date'] = selector.css('#epContentLeft .post_time_source::text').get()
    if item['pub_date'] is not None: # 判断发布时间不为空则处理时间格式
        item['pub_date'] = item['pub_date'].split()[0]

    item['desc'] = selector.css('#epContentLeft .post_desc::text').get()
    if item['desc'] is not None: # 判断描述不为空则对文本进行去除空格操作
        item['desc'] = item['desc'].strip()

    item['body'] = selector.css('#endText::text').get()
    if item['body'] is not None:  # 判断正文不为空则对文本进行去除空格操作
        item['body'] = item['body'].strip()

    item['link'] = response.url
    return item
```

上述代码分别对 `desc`、`pub_date`、`body` 的内容进行的去空格的操作，然后又对 `pub_date` 进行字符串的分离操作，其实这些操作都是非常之常见，因为网页上抓取下来的数据格式总是千奇百怪，更复杂的情况可能还要对提取出的字段值进行类型转换甚至计算操作。一但字段增多或页面的元素因为更新发生改变这将真的会演变成一场噩梦，代码将会变得极为难读更难以维护。

使用项目加载器(`ItemLoader`)就可以彻底改变这一困局。它可以将数据值的处理规则封装到的 Item 中而在分析函数(`parse_item`)内则只保留提取规则。

要使用项目加载器(`ItemLoader`)，你必须首先实例化它。我们可以将自定义的Item类的实例化对象作为`item`参数传入 `ItemLoader` 的构造函数。

然后开始使用选择器获取数据到项装载程序。你可以向同一项目字段添加多个值； 项目加载器将知道如何使用适当的处理函数“加入”这些值。

接下来我们就用 `ItemLoader` 的方式来改写网易蜘蛛的`parse_item`方法，在代码中引入 ItemLoader (`from scrapy.loader import ItemLoader`):

```
def parse_item(self, response):
    loader = ItemLoader(item=NewsItem(), response=response)
    loader.add_css('title', '#epContentLeft>h1::text')
    loader.add_css('pub_date', '#epContentLeft .post_time_source::text')
    loader.add_css('desc', '#epContentLeft .post_desc::text')
    loader.add_css('body', '#endText::text')
    loader.add_value('link',response.url)
    yield loader.load_item()
```

如果打开调试器进入断点观察 `loader.load_item()` 的结果会得到以下的数据:

```
news_item = [
 'title':['前11个月国企利润同比增长15.6% 偿债能力有所提升'],
 'pub_date': ['2018-12-29'],
 'desc':['内容太长，此处略去'],
 'body':['内容太长，此处略去'],
 'link':['https://news.163.com/18/1229/13/E46QMLQH000189FH.html']
]
```

对于上述代码我们首先来解读一下 `ItemLoader` 的 `add_XXXX` 方法系列，它们一共有三个，第一个参数是声明将数据填充到 `NewsItem` 的哪个属性中，第二个参数则分别如下:

- `add_css` - 声明css选择器
- `add_xpath` - 声明xpath选择器
- `add_value` - 向字段直接填充一个值

`ItemLoader` 通过上述的这些 `add_XXX` 方法的定义建立了与 `NewsItem` 对象的属性映射结构，当调用`load_item` 函数时 `ItemLoader` 才会一次性地将数据填充到 `NewsItem` 实例中。

> 注：发现三种方式填充的数据，均为`List`类型

使用 `ItemLoader` 的最大好处是将数据的结构与提取逻辑有效地分离开来：

- `Item` 负责处理"属性值"的问题
- `ItemLoader` 则负责处理提取的逻辑

上述的代码就展示了 `ItemLoader` 处理提取逻辑的方式，那么 `Item` 又应该如何处理"值”的问题呢？这就得益于 `Field`对象在构造时传入的 `in_processor` 和 `out_processor` 了，我们又可以将之称为输入和输出处理器。

对 In/Out Processor 进行深度的使用分析

输入和输出处理器可以在Item Loader定义中声明，也可以在 `Item` 中声明，我们先来看看如何在 `Item` 中声明，以下是对 `NewsItem` 的改写：

```
# -*- coding: utf-8 -*-
from scrapy.item import Item, Field
from scrapy.loader.processors import TakeFirst, MapCompose, Compose, Identity, Join
from w3lib.html import remove_tags

class NewsItem(Item):
    # 在NewsItem对数据进行逻辑处理
    title = Field(output_processor=TakeFirst())
    desc = Field(input_processor=MapCompose(str.strip,
                                            stop_on_none=True),
                 output_processor=TakeFirst())
    link = Field(output_processor=TakeFirst())
    pub_date = Field(input_processor=MapCompose(lambda v: v.split()[0],
                                                stop_on_none=True),
                     output_processor=TakeFirst())
    body = Field(input_processor=MapCompose(remove_tags, str.strip,
                                            stop_on_none=True),
                 output_processor=TakeFirst())
```

这里最大的区别就是已经不再需要对 `None` 值进行处理，对于 `None` 值， `ItemLoader` 会先作出一次处理，对于为空的值都会直接跳过不作处理以避免因为没有处理空而导致的异常。

开始使用输入/输出处理器时会感觉有点困惑：

```
- 到底输入/输出处理器是什么？
- 应该用什么输入/输出处理器？
```

首先，我们得从 `ItemLoader` 的工作原理来了解他们，输入/输出处理器实际上就是一个函数，当 `ItemLoader` 从响应内容中按我们声明的提取方法取出内容时会以一个 `list` 对象向 `Item` 赋值(相当于我们前文中使用`Selector.getAll()`)，例如:

```
loader.add_css('title', '#epContentLeft>h1::text')
```

在执行 `loader.load_item()` 后，数据项的 `title` 属性的值就会是以下这样的内容：

```
['在h1中的标题文字']
```

当不声明 `Field` 的任何处理器时，`ItemLoader` 会使用一个 `Identity` 处理器向 `NewsItem['title']` 赋值，`Identity` 是一个什么都不做的处理器，它只是用于将来自于 `ItemLoader` 的值如实地传递给`NewsItem['title']` 属性。

输入/输出处理器就是对值的处理函数，区别只在于它们被调用的时机不同。

> 简言之：当 ItemLoader 调用 `add_xxx` 函数时输入处理器就会被调用，当 `load_item` 方法被调用时所有的输出处理器就会被统一调用。

以下就是 Scrapy 所提供的输入/输出处理器：

Identity：最简单的处理器，不进行任何处理，直接返回原来的数据。无参数。

TakeFirst：返回第一个非空（non-null/ non-empty）值，常用于单值字段的输出处理器，无参数。

Compose：

- 用给定的多个函数的组合，来构造的处理器。**list对象（注意不是指list中的元素）**，依次被传递到第一个函数，然后输出，再传递到第二个函数，一个接着一个，直到最后一个函数返回整个处理器的输出。
- 默认情况下，当遇到None值（list中有None值）的时候停止处理。可以通过传递参数stop_on_none = False改变这种行为。
- 每个函数可以选择接收一个loader_context参数。

MapCompose：

- 与Compose处理器类似，区别在于各个函数结果在内部传递的方式（会涉及到list对象解包的步骤）：
  - 输入值是被迭代的处理的，List对象中的每一个元素被单独传入，第一个函数进行处理，然后处理的结果被连接起来形成一个新的迭代器，并被传入第二个函数，以此类推，直到最后一个函数。最后一个函数的输出被连接起来形成处理器的输出。
  - 每个函数能返回一个值或者一个值列表，也能返回None（会被下一个函数所忽略）
  - 这个处理器提供了很方便的方式来组合多个处理单值的函数。因此它常用于输入处理器，因为传递过来的是一个List对象。
- 与 Compose 处理器类似，它也能接受Loader context。

Join：

- 返回用分隔符连接后的值。分隔符默认为空格。不接受 Loader contexts。
- 当使用默认分隔符的时候，这个处理器等同于如下这个

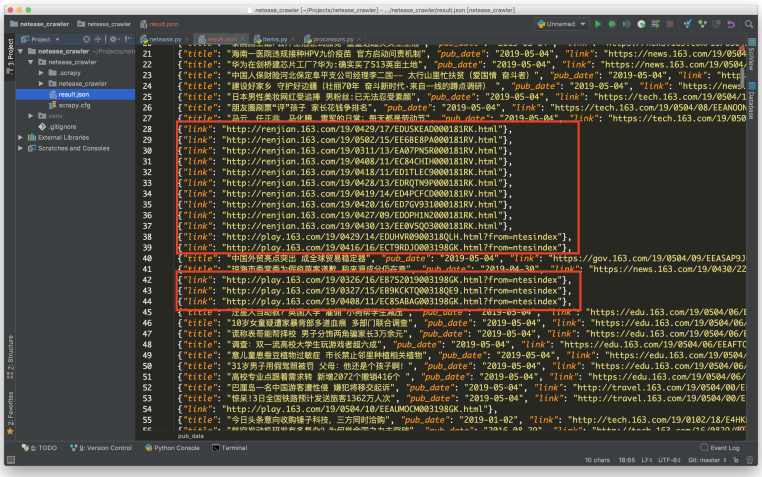
所以，我们字段属性存的是单值那一般上会直接使用`TakeFirst`作为输出处理器(`out_processor=TakeFirst()`)，从处理结果列表中将第一值拿出来；如果要对数据值进行附加处理(例如：去空格或者转类型)时就可以在`in_processor`加入处理器进行赋值前的预处理。

用以上的逻辑重新来理解`NewsItem`的定义：

```
title = Field(output_processor=TakeFirst()) # 提取列表中第一个元素作为title的值
# 对列表中的所有值进行去空格处理，一但遇到空值就不再进行处理(输出处理器将不会工作)
desc=Field(input_processor=MapCompose(str.strip,stop_on_none=True),
           output_processor=TakeFirst())
# 与title处理相同
link = Field(output_processor=TakeFirst())
pub_date = Field(input_processor=MapCompose(lambda v: v.split()[0],stop_on_none=True)                    output_processor
=TakeFirst())
# 删除HTML标记，然后去空格，并且遇到空值就停止处理
body = Field(input_processor=MapCompose(remove_tags, str.strip,stop_on_none=True),
             output_processor=TakeFirst())
```

从不同的页面提取同样的数据

将使用 `ItemLoader` 形式改进后的代码重新运行一次，然后打开的 `result.json` 文件查看爬取的结果，会发现以下的情况：



结果文件里有很多的数据只有 `link` 字段，而其它的字段全部为空！其实即使我们不用 `ItemLoader` 改写这个爬虫这种情况就早在项目开始出现了。只要将这些异常数据的URL直接在浏览器打开你就会发现问题的原因。下面我们就打开 http://play.163.com/19/0326/16/EB752019003198GK.html 观察原网页到底是个什么情况：

很明显这个"网易爱玩"栏目与我们从一开始分析新闻栏目时的网页结构完全不同，所以我们的元素选择器根本没有办法提取到对应的内容。像网易这种经营多年门户网站不同的栏目由不同的项目组负责，甚至连开发人员也不同。可能是开发规范不一至、也可能是由于发展需要等多种我们无法估计的原因，导致同一个网站同样类别的数据会以不同的网页结构来呈现。

也就是说我们的爬虫从解决1对1的问题(一 个 `数据对象` **对应一套网页逻辑**)变成了解决1对多的问题，如果在没有 `ItemLoader` 的情况下，我前文所说的"维护地狱"之门将会为你打开。对每个网页的元素提取必须重做一次，对值的非空判断必须重做一次，对值的格式化必须重做一次，诸如此类。

用`ItemLoader`来处理就轻松地处理这种一对多式的爬取问题，对于新的网页元素结构需要重新分析和制作以下的映射表：

名称	字段	选择器
标题	title	`h1.article-h1`
发表日期	pub_date	无
摘要	desc	`.artical-summary`
正文	body	`#endText`
链接	link	当前请求中的URL

将以上的关系直接补充到 `ItemLoader` 中，代码如下所示：

```
def parse_item(self, response):
  loader = ItemLoader(item=NewsItem(), response=response)
  loader.add_css('title', '#epContentLeft>h1::text')
  loader.add_css('pub_date', '#epContentLeft .post_time_source::text')
  loader.add_css('desc', '#epContentLeft .post_desc::text')

  # 游戏栏目 play.163.com
  loader.add_css('title', 'h1.article-h1::text')
  loader.add_css('desc', '.artical-summary::text')

  loader.add_xpath('body', '//div[@id="endText"]')
  loader.add_value('link', response.url)
  return loader.load_item()
```

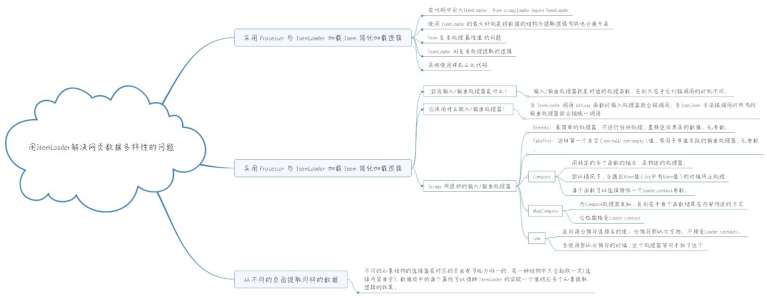
修改后重新运行爬虫可以观察到异常的数据就已经能被补回来了。这种处理的思路是：

> 不同的元素结构的选择器在对应的页面都可视为唯一的，在一种结构中只会起效一次(选择内容非空)，数据项中的每个属性可以借助 `ItemLoader` 的实现一个值对应多个元素提取逻辑的效果。

小结

对于网易各大栏目而言上述代码的重构是远远不够的，当然要将对方的网站彻底"扒光"的行为并不可取，毕竟爬虫能使用的资源是有限的，获取的数据也是有一定目标范围的。所以需要针对哪些栏目的结构增加提取逻辑可视需求而定，范围越是精确爬虫的爬网的总时长必然越短。

在示例的数据结构比较简单，当遇到数据提取逻辑更复杂、字段结构更多、值处理规则更多的情况就需要对`ItemLoader` 与 输入/输出处理器的运用更为灵活，例如你可以继承 `ItemLoader` 将所有提取逻辑全部封装到一个自定义的`ItemLoader` 中便于在其它的爬虫中重用。



11 编写网易爬虫NeteaseSpider 让它“爬”起来

13 去重处理— 高性能爬虫调优技术

精选留言 2

欢迎在这里发表留言，作者筛选后可公开显示

拙凡

感觉好多在罗列定义，缺少对比的例子。需要反复去其他网站学习这几篇教学文章里的内容。让人感觉课程内容太过定义化。希望老师能多给一些项目例子，并且多做对比，和类比的解释。

0 回复 2019-06-03

专杀小幕

上一章中的泛爬代码我爬了300多条数据，换成itemLoad后只爬了50条，，这是什么原因呢？还有用本章下边的代码后又跟本章上边的爬取内容对不上，并非补充。。原理意思懂了，但是实际操作很蒙圈

0 回复 2019-05-27