**Scrapy Item Loaders机制详解**

2015年11月25日 20:19:33

阅读数：8705

**1. Items**

爬虫的主要任务就是从非结构化的数据中获得结构化的数据。   
Item 对象是种简单的容器，保存了爬取到得数据。 其提供了 类似于词典(dictionary-like) 的API以及用于声明可用字段的简单语法。

**声明Item**   
Item使用简单的class定义语法以及 Field 对象来声明。例如:

import scrapy

class Product(scrapy.Item):

name = scrapy.Field()

price = scrapy.Field()

stock = scrapy.Field()

last\_updated = scrapy.Field(serializer=str)

**Item字段：**   
Field 对象指明了每个字段的元数据(metadata)。例如上面例子中 last\_updated 中指明了该字段的序列化函数。

可以为每个字段指明任何类型的元数据。Field 对象对接受的值没有任何限制。Field 对象中保存的每个键可以由多个组件使用，并且只有这些组件知道这个键的存在。设置 Field 对象的主要目的就是在一个地方定义好所有的元数据。

需要注意的是，用来声明item的 Field 对象并没有被赋值为class的属性。 不过您可以通过 Item.fields 属性进行访问。

**2. 用Item Loader来填充Item**

Item Loaders提供了一种便捷的方式填充抓取到的 Items 。 虽然Items可以使用自带的类字典形式API填充，但是Items Loaders提供了更便捷的API， 可以分析原始数据并对Item进行赋值。

从另一方面来说， Items 提供保存抓取数据的 *容器* ， 而 Item Loaders提供的是 *填充* 容器的机制。

Item Loaders提供的是一种灵活，高效的机制，可以更方便的被spider或source format (HTML, XML, etc)扩展，并override更易于维护的、不同的内容分析规则。

要使用Item Loader, 你必须先将它实例化. 可以使用类似字典的对象来进行实例化, 或者不使用对象也可以, 当不用对象进行实例化的时候,Item会自动使用 ItemLoader.default\_item\_class 属性中指定的Item 类在Item Loader constructor中实例化.

然后,你开始收集数值到Item Loader时,通常使用 Selectors. 你可以在同一个item field 里面添加多个数值;Item Loader将知道如何用合适的处理函数来“添加”这些数值.

下面是在 Spider 中典型的Item Loader的用法, 使用 Items chapter 中声明的 Product item:

from scrapy.contrib.loader import ItemLoader

from myproject.items import Product

def parse(self, response):

l = ItemLoader(item=Product(), response=response)

l.add\_xpath('name', '//div[@class="product\_name"]')

l.add\_xpath('name', '//div[@class="product\_title"]')

l.add\_xpath('price', '//p[@id="price"]')

l.add\_css('stock', 'p#stock]')

l.add\_value('last\_updated', 'today') # you can also use literal values

return l.load\_item()

我们可以看到发现 name 字段被从页面中两个不同的XPath位置提取:

1. //div[@class="product\_name"]
2. //div[@class="product\_title"]

换言之,数据通过用 add\_xpath() 的方法,把从两个不同的XPath位置提取的数据收集起来. 这是将在以后分配给 name 字段中的数据｡

之后,类似的请求被用于 price 和 stock 字段 (后者使用 CSS selector 和 add\_css() 方法), 最后使用不同的方法 add\_value() 对 last\_update 填充文本值( today ).

最终, 当所有数据被收集起来之后, 调用 ItemLoader.load\_item() 方法, 实际上填充并且返回了之前通过调用 add\_xpath(), add\_css(), and add\_value() 所提取和收集到的数据的Item.

**3. 输入处理器与输出处理器**

* Item Loader在每个字段中都包含了一个输入处理器和一个输出处理器｡
* 输入处理器收到数据时立刻提取数据 (通过 add\_xpath(), add\_css() 或者 add\_value() 方法) 之后输入处理器的结果被收集起来并且保存在ItemLoader内(但尚未分配给该Item).
* 收集到所有的数据后, 调用 ItemLoader.load\_item() 方法来填充,并得到填充后的 Item 对象。在这一步中**先调用输出处理器来处理之前收集到的数据，然后再存入Item中**。输出处理器的结果是被分配到Item的最终值｡

需要注意的是，输入和输出处理器都是可调用对象，调用时传入需要被分析的数据， 处理后返回分析得到的值。因此你可以使用任意函数作为输入、输出处理器。 唯一需注意的是它们必须接收一个（并且只是一个）迭代器性质的positional参数。

**4. 声明Items Loaders**

Item Loaders 的声明类似于Items，以class的语法来声明：

from scrapy.contrib.loader import ItemLoader

from scrapy.contrib.loader.processor import TakeFirst, MapCompose, Join

class ProductLoader(ItemLoader):

default\_output\_processor = TakeFirst()

name\_in = MapCompose(unicode.title)

name\_out = Join()

price\_in = MapCompose(unicode.strip)

# ...

input processors 以\_in为后缀来声明，output processors 以\_out 为后缀来声明。也可以用ItemLoader.default\_input\_processor 和ItemLoader.default\_output\_processor 属性来声明默认的 input/output processors。

**5. 声明Input and Output Processors**

前面讲到，input and output processors可以在定义Item Loaders的时候声明，这是非常普遍的使用方法。但是，你也可以在定义Item的时候声明输入输出处理器。下面是例子：

import scrapy

from scrapy.contrib.loader.processor import Join, MapCompose, TakeFirst

from w3lib.html import remove\_tags

def filter\_price(value):

if value.isdigit():

return value

class ProductItem(scrapy.Item):

name = scrapy.Field(

input\_processor=MapCompose(remove\_tags),

output\_processor=Join(),

)

price = scrapy.Field(

input\_processor=MapCompose(remove\_tags, filter\_price),

output\_processor=TakeFirst(),

)

使用Item：

>>> from scrapy.contrib.loader import ItemLoader

>>> il = ItemLoader(item=Product())

>>> il.add\_value('name', [u'Welcome to my', u'<strong>website</strong>'])

>>> il.add\_value('price', [u'&euro;', u'<span>1000</span>'])

>>> il.load\_item()

{'name': u'Welcome to my website', 'price': u'1000'}

关于集中声明 input and output processors方式的优先级排序如下：

1. 在Item Loader 中声明的 field-specific 属性: field\_in and field\_out (most precedence)
2. Item中的字段元数据(input\_processor and output\_processor key)
3. Item Loader 默认处理器: ItemLoader.default\_input\_processor() and ItemLoader.default\_output\_processor() (least precedence)

**6. Item Loader Context**

Item Loader Context 是一个被Item Loader中的输入输出处理器共享的任意的键值对字典。它能在Item Loader声明、实例化、使用的时候传入。它用于调整输入输出处理器的行为。

举例来讲，函数parse\_length用于接收text值并且获取其长度：

def parse\_length(text, loader\_context):

unit = loader\_context.get('unit', 'm')

# ... length parsing code goes here ...

return parsed\_length

通过接收一个loader\_context参数，这个函数告诉Item Loader它能够接收Item Loader context。于是当函数被调用的时候Item Loader传递当前的active context给它。

有多种方式改变Item Loader context的值：

1. 修改当前 active Item Loader context：

loader = ItemLoader(product)   
loader.context[‘unit’] = ‘cm’

1. 在Item Loader实例化的时候：

loader = ItemLoader(product, unit=’cm’)

1. 对于那些支持带Item Loader context实例化的输入输出处理器（例如MapCompose），在Item Loader声明的时候修改它context：

class ProductLoader(ItemLoader):   
length\_out = MapCompose(parse\_length, unit=’cm’)

**7. ItemLoader object**

参见[官方文档](http://doc.scrapy.org/en/latest/topics/loaders.html" \l "itemloader-objects" \t "_blank)

**8. 重用和扩展Item Loaders**

当你的项目逐渐变大，使用了越来越多的spider的时候，维护变成了一个基本的问题。尤其是当你需要处理每个spider的许多不同的解析规则的时候，会出现很多的异常，迫使你开始考虑重用的问题。

Item Loader支持传统的Python继承机制来处理spider之间的差异。

例如，有些网站把它们的产Product名用三个短线封装起来（如：---Plasma TV---），而你想要去掉这些东西。

你可以通过reusing and extending默认Product Item Loader的方式去掉短线：

from scrapy.loader.processors import MapCompose

from myproject.ItemLoaders import ProductLoader

def strip\_dashes(x):

return x.strip('-')

class SiteSpecificLoader(ProductLoader):

name\_in = MapCompose(strip\_dashes, ProductLoader.name\_in)

* 1另一种情形时继承Item Loader也很有用：有多种格式的源数据（如XML, HTML），在XML版本里面你想要去除CDATA:

from scrapy.loader.processors import MapCompose

from myproject.ItemLoaders import ProductLoader

from myproject.utils.xml import remove\_cdata

class XmlProductLoader(ProductLoader):

name\_in = MapCompose(remove\_cdata, ProductLoader.name\_in)

这便是扩展**输入处理器**的方法。

对于**输出处理器，更常用的方式是在Item字段元数据里声明。因为通常它们依赖于具体的字段而不是网站**。

还有很多其他方式开扩展、继承和覆盖Item Loader，不同的层次结构适于不同的项目。Scrapy只是提供了这些机制，不强制要求具体的组织方式。

**9. 内置的处理器**

尽管你可以使用可调用的函数作为输入输出处理器，Scrapy提供了一些常用的处理器。有些处理器，如MapCompose（通常用于输入处理器），能把多个函数执行的结果按顺序组合起来产生最终的输出。

下面是一些内置的处理器：

**9.1 Identity**

class scrapy.loader.processors.**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。

**9.2 TakeFirst**

class scrapy.loader.processors.**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。

示例如下：

>>> from scrapy.loader.processors import TakeFirst

>>> proc = TakeFirst()

>>> proc(['', 'one', 'two', 'three'])

'one'

**9.3 Join**

class scrapy.loader.processors.**Join(separator=u’ ‘)**

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，这个处理器等同于这个函数：

u' '.join

* 1
* 2

使用示例：

>>> from scrapy.loader.processors import Join

>>> proc = Join()

>>> proc(['one', 'two', 'three'])

u'one two three'

>>> proc = Join('<br>')

>>> proc(['one', 'two', 'three'])

u'one<br>two<br>three'

**9.4 Compose**

class scrapy.loader.processors.**Compose(\*functions, \*\*default\_loader\_context)**

用给定的多个函数的组合而构造的处理器。每个输入值被传递到第一个函数，然后其输出再传递到第二个函数，诸如此类，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。

使用示例：

>>> from scrapy.loader.processors import Compose

>>> proc = Compose(lambda v: v[0], str.upper)

>>> proc(['hello', 'world'])

'HELLO'

* 5

每个函数可以选择接收一个loader\_context参数。

**9.5 MapCompose**

class scrapy.loader.processors.**MapCompose(\*functions, \*\*default\_loader\_context)**

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：

* 输入值是被迭代的处理的，每一个元素被单独传入第一个函数进行处理。处理的结果被l连接起来(concatenate)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)
* **这个处理器提供了方便的方式来组合多个处理单值的函数**。因此它常用与输入处理器，因为用extract()函数提取出来的值是一个unicode strings列表。

下面的例子能说明这个处理器的工作方式：

>>> def filter\_world(x):

... return None if x == 'world' else x

...

>>> from scrapy.loader.processors import MapCompose

>>> proc = MapCompose(filter\_world, unicode.upper)

>>> proc([u'hello', u'world', u'this', u'is', u'scrapy'])

[u'HELLO, u'THIS', u'IS', u'SCRAPY']

与Compose处理器类似，它也能接受Loader context。

**9.6 SelectJmes**

class scrapy.loader.processors.**SelectJmes(json\_path)**

查询指定的JSON path并返回输出。需要jmespath(<https://github.com/jmespath/jmespath.py>)支持。每次接受一个输入。

示例：

>>> from scrapy.loader.processors import SelectJmes, Compose, MapCompose

>>> proc = SelectJmes("foo") #for direct use on lists and dictionaries

>>> proc({'foo': 'bar'})

'bar'

>>> proc({'foo': {'bar': 'baz'}})

{'bar': 'baz'}

与Json一起使用：

>>> import json

>>> proc\_single\_json\_str = Compose(json.loads, SelectJmes("foo"))

>>> proc\_single\_json\_str('{"foo": "bar"}')

u'bar'

>>> proc\_json\_list = Compose(json.loads, MapCompose(SelectJmes('foo')))

>>> proc\_json\_list('[{"foo":"bar"}, {"baz":"tar"}]')

[u'bar']