物品装载机

项目加载程序提供了一种方便的机制来填充已删除的项目。尽管可以使用他们自己的类字典API来填充项目，但是项目加载器通过自动执行一些常见任务（例如在分配原始提取数据之前解析原始提取数据），提供了一种更方便的API，用于从抓取过程中填充它们。

换句话说，Items提供了抓取数据的*容器*，而Item Loaders提供了*填充*该容器的机制。

项目加载器旨在提供灵活，高效和简单的机制，用于通过蜘蛛或源格式（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。

使用项目加载器填充项目

要使用Item Loader，必须先实例化它。您可以使用类似dict的对象（例如Item或dict）实例化它，也可以不使用一个，在这种情况下，Item使用`ItemLoader.default_item_class`属性中指定的Item类在Item Loader构造函数中自动实例化。

然后，您开始将值集合到Item Loader中，通常使用 Selectors。您可以向同一项目字段添加多个值; Item Loader将知道如何使用适当的处理函数“加入”这些值。

下面是Spider中典型的Item Loader用法，使用Items章节中声明的Product项：

```
from scrapy.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()
```

通过快速查看该代码，我们可以看到该`name`字段是从页面中的两个不同XPath位置提取的：

1. `//div[@class="product_name"]`
2. `//div[@class="product_title"]`

换句话说，通过使用该`add_xpath()`方法从两个XPath位置提取数据来收集数据。这是稍后将分配给该`name`字段的数据。

之后，类似的调用用于 `price` 和 `stock` 字段（后者使用CSS选择器和 `add_css()` 方法），最后使用不同的方法 `last_update` 直接用字面值（ `today` ）填充字段： `add_value()` 。

最后，收集到的所有数据时，该 `ItemLoader.load_item()` 方法被调用，实际上返回填充了先前提取并与收集到的数据的项目 `add_xpath()` ， `add_css()` 和 `add_value()` 调用。

输入输出处理器

Item Loader包含一个输入处理器和一个输出处理器，用于每个（item）字段。输入处理器只要它接收处理所提取的数据（通过 `add_xpath()` ， `add_css()` 或 `add_value()` 方法）和输入处理器的结果被收集和保持在ItemLoader内部。收集所有数据后， `ItemLoader.load_item()` 调用该方法来填充并获取填充的 `Item` 对象。这是在使用先前收集的数据（并使用输入处理器处理）调用输出处理器时。输出处理器的结果是分配给项目的最终值。

让我们看一个例子来说明如何为特定字段调用输入和输出处理器（同样适用于任何其他字段）：

```
l = ItemLoader(Product(), some_selector)
l.add_xpath('name', xpath1) # (1)
l.add_xpath('name', xpath2) # (2)
l.add_css('name', css) # (3)
l.add_value('name', 'test') # (4)
return l.load_item() # (5)
```

那么会发生什么：

1. 从数据 `xpath1` 提取出来，并通过所传递的*输入处理器*的的 `name` 字段。输入处理器的结果被收集并保存在Item Loader中（但尚未分配给该项目）。
2. `xpath2` 提取数据，并通过（1）中使用的相同*输入处理器*。输入处理器的结果附加到（1）中收集的数据（如果有的话）。
3. 这种情况类似于前面的情况，除了从 `css` CSS选择器中提取数据，并通过（1）和（2）中使用的相同*输入处理器*。输入处理器的结果附加到（1）和（2）中收集的数据（如果有的话）。
4. 这种情况也类似于以前的情况，除了直接分配要收集的值，而不是从XPath表达式或CSS选择器中提取。但是，该值仍然通过输入处理器传递。在这种情况下，由于该值不可迭代，因此在将其传递给输入处理器之前将其转换为单个元素的可迭代，因为输入处理器始终接收可迭代。
5. 在步骤（1），（2），（3）和（4）中收集的数据通过该字段的*输出处理器* `name` 。输出处理器的结果是分配给 `name` 项目中字段的值。

值得注意的是，处理器只是可调用对象，可以使用要解析的数据调用它们，并返回解析后的值。因此您可以使用任何功能作为输入或输出处理器。唯一的要求是它们必须接受一个（且只有一个）位置参数，它将是一个迭代器。

注意

输入和输出处理器都必须接收迭代器作为它们的第一个参数。这些功能的输出可以是任何东西。输入处理器的结果将附加到包含收集值（对于该字段）的内部列表（在Loader中）。输出处理器的结果是最终分配给项目的值。

如果要将普通函数用作处理器，请确保将其 `self` 作为第一个参数接收 ：

```
def lowercase_processor(self, values):
    for v in values:
        yield v.lower()

class MyItemLoader(ItemLoader):
    name_in = lowercase_processor
```

这是因为无论何时将函数指定为类变量，它都会成为一个方法，并在调用时将实例作为第一个参数传递。有关详细信息，请参阅stackoverflow上的此答案。

您需要记住的另一件事是输入处理器返回的值在内部收集（在列表中），然后传递给输出处理器以填充字段。

最后，但并非最不重要的是，为了方便起见，Scrapy 内置了一些常用的处理器。

声明项目加载器

通过使用类定义语法将Item Loaders声明为Items。这是一个例子：

```
from scrapy.loader import ItemLoader
from scrapy.loader.processors import TakeFirst, MapCompose, Join

class ProductLoader(ItemLoader):

    default_output_processor = TakeFirst()

    name_in = MapCompose(unicode.title)
    name_out = Join()

    price_in = MapCompose(unicode.strip)

    # ...
```

如您所见，输入处理器使用 `_in` 后缀声明，而输出处理器使用 `_out` 后缀声明。您还可以使用 `ItemLoader.default_input_processor` 和 `ItemLoader.default_output_processor` 属性声明默认的输入/输出处理器 。

声明输入和输出处理器

如上一节所示，输入和输出处理器可以在Item Loader定义中声明，以这种方式声明输入处理器是很常见的。但是，还有一个地方可以指定要使用的输入和输出处理器：在项目字段 元数据中。这是一个例子：

```
import scrapy
from scrapy.loader.processors import Join, MapCompose, TakeFirst
from w3lib.html import remove_tags

def filter_price(value):
    if value.isdigit():
        return value

class Product(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(),
    )
```

```
>>> from scrapy.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'}
```

输入和输出处理器的优先顺序如下：

1. Item Loader特定于字段的属性：`field_in`和`field_out`（最优先）
2. 字段元数据（`input_processor`和`output_processor`键）
3. Item Loader默认值：`ItemLoader.default_input_processor()`和`ItemLoader.default_output_processor()`（最少优先级）

另请参阅：重用和扩展项目加载器。

项目加载器上下文

Item Loader Context是任意键/值的dict，它在Item Loader中的所有输入和输出处理器之间共享。它可以在声明，实例化或使用Item Loader时传递。它们用于修改输入/输出处理器的行为。

例如，假设您有一个函数`parse_length`接收文本值并从中提取长度：

```
def parse_length(text, loader_context):
    unit = loader_context.get('unit', 'm')
    # ... length parsing code goes here ...
    return parsed_length
```

通过接受`loader_context`参数，函数显式地告诉Item Loader它能够接收Item Loader上下文，因此Item Loader在调用它时传递当前活动的上下文，并且处理器函数（`parse_length`在这种情况下）可以使用它们。

有几种方法可以修改Item Loader上下文值：

1. 通过修改当前活动的Item Loader上下文（`context`属性）：

```
loader = ItemLoader(product)
loader.context['unit'] = 'cm'
```

2. 在Item Loader实例化中（Item Loader构造函数的关键字参数存储在Item Loader上下文中）：

```
loader = ItemLoader(product, unit='cm')
```

3. 在Item Loader声明中，对于那些支持使用Item Loader上下文实例化它们的输入/输出处理器。`MapCompose`是其中之一：

```
class ProductLoader(ItemLoader):
    length_out = MapCompose(parse_length, unit='cm')
```

ItemLoader对象

class **scrapy.loader.ItemLoader** ([*item* , *selector* , *response* ,] ** *kwargs*)

返回一个新的Item Loader来填充给定的Item。如果没有给出项目，则使用该类自动实例化一个项目`default_item_class`。

当使用*选择器*或*响应*参数进行实例化时，`ItemLoader`该类提供了使用选择器从网页中提取数据的便利机制。

参数：

- **项**（`Item`对象）-项目实例来填充利用后续调用`add_xpath()`，`add_css()`或`add_value()`。
- **selector**（`Selector` object）- 使用`add_xpath()`（resp。`add_css()`）或`replace_xpath()`（resp。`replace_css()`）方法时从中提取数据的选择器。
- **response**（`Response` object）- `default_selector_class`除非给出了选择器参数，否则用于构造选择器的响应，在这种情况下，忽略该参数。

项目，选择器，响应和其余关键字参数分配给Loader上下文（可通过`context`属性访问）。

`ItemLoader`实例有以下方法：

get_value (*值* , **处理器* , ** *kwargs*)

处理给`value`定`processors`和关键字参数给定的给定。

可用关键字参数：

参数： re（*str 或者编译的正则表达式*）- 一个正则表达式，用于从`extract_regex()`处理器之前应用的方法中提取给定值的数据

例子：

```
>>> from scrapy.loader.processors import TakeFirst
>>> loader.get_value(u'name: foo', TakeFirst(), unicode.upper, re='name: (.+)')
'FOO`
```

`add_value`（*field_name*，*value*，** processors*，*** kwargs*）

处理然后添加给`value`定字段的给定。

首先通过`get_value()`给出`processors`和传递该值`kwargs`，然后通过 字段输入处理器 并将其结果附加到为该字段收集的数据中。如果该字段已包含收集的数据，则添加新数据。

给定的`field_name`可以是`None`，在这种情况下，可以添加多个字段的值。并且处理后的值应该是一个字典，其中field_name映射到值。

例子：

```
loader.add_value('name', u'Color TV')
loader.add_value('colours', [u'white', u'blue'])
loader.add_value('length', u'100')
loader.add_value('name', u'name: foo', TakeFirst(), re='name: (.+)')
loader.add_value(None, {'name': u'foo', 'sex': u'male'})
```

`replace_value`（*field_name*，*value*，** processors*，*** kwargs*）

与`add_value()`但相似但是用新值替换收集的数据而不是添加它。

`get_xpath`（*xpath*，**处理器*，*** kwargs*）

类似于`ItemLoader.get_value()`但接收XPath而不是值，该值用于从与此关联的选择器中提取unicode字符串列表`ItemLoader`。

参数：
- **xpath**（*str*）- 从中提取数据的XPath
- **re**（*str 或编译的正则表达式*）- 用于从所选XPath区域提取数据的正则表达式

例子：

```
# HTML snippet: <p class="product-name">Color TV</p>
loader.get_xpath('//p[@class="product-name"]')
# HTML snippet: <p id="price">the price is $1200</p>
loader.get_xpath('//p[@id="price"]', TakeFirst(), re='the price is (.*)')
```

add_xpath(*field_name* , *xpath* , ** processors* , *** kwargs*)

类似于 `ItemLoader.add_value()` 但接收XPath而不是值，该值用于从与此关联的选择器中提取unicode字符串列表 `ItemLoader` 。

见 `get_xpath()` 的 `kwargs` 。

参数： **xpath** (*str*) - 从中提取数据的XPath

例子：

```
# HTML snippet: <p class="product-name">Color TV</p>
loader.add_xpath('name', '//p[@class="product-name"]')
# HTML snippet: <p id="price">the price is $1200</p>
loader.add_xpath('price', '//p[@id="price"]', re='the price is (.*)')
```

replace_xpath(*field_name* , *xpath* , ** processors* , *** kwargs*)

类似于 `add_xpath()` 但替换收集的数据而不是添加它。

get_css(*css* , **处理器* , *** kwargs*)

类似 `ItemLoader.get_value()` 但接收CSS选择器而不是值，用于从与此关联的选择器中提取unicode字符串列表 `ItemLoader` 。

参数：
- **css** (*str*) - 从中提取数据的CSS选择器
- **re** (*str 或编译的正则表达式*) - 用于从所选CSS区域提取数据的正则表达式

例子：

```
# HTML snippet: <p class="product-name">Color TV</p>
loader.get_css('p.product-name')
# HTML snippet: <p id="price">the price is $1200</p>
loader.get_css('p#price', TakeFirst(), re='the price is (.*)')
```

add_css(*field_name* , *css* , ** processors* , *** kwargs*)

类似 `ItemLoader.add_value()` 但接收CSS选择器而不是值，用于从与此关联的选择器中提取unicode字符串列表 `ItemLoader` 。

见 `get_css()` 的 `kwargs` 。

参数： **css** (*str*) - 从中提取数据的CSS选择器

例子：

```
# HTML snippet: <p class="product-name">Color TV</p>
loader.add_css('name', 'p.product-name')
# HTML snippet: <p id="price">the price is $1200</p>
loader.add_css('price', 'p#price', re='the price is (.*)')
```

replace_css (*field_name* , *css* , ** processors* , *** kwargs*)

类似于 `add_css()` 但替换收集的数据而不是添加它。

load_item ( )

使用目前为止收集的数据填充项目，并将其返回。收集的数据首先通过输出处理器，以获得分配给每个项目字段的最终值。

nested_xpath (*xpath*)

使用xpath选择器创建嵌套加载器。提供的选择器相对于与此关联的选择器应用 `ItemLoader` 。嵌套装载机股份 `Item` 与母公司 `ItemLoader` 如此呼吁 `add_xpath()` , `add_value()` , `replace_value()` 等会像预期的那样。

nested_css (*css*)

使用css选择器创建嵌套加载器。提供的选择器相对于与此关联的选择器应用 `ItemLoader` 。嵌套装载机股份 `Item` 与母公司 `ItemLoader` 如此呼吁 `add_xpath()` , `add_value()` , `replace_value()` 等会像预期的那样。

get_collected_values (*field_name*)

返回给定字段的收集值。

get_output_value (*field_name*)

对于给定字段，返回使用输出处理器解析的收集值。此方法根本不会填充或修改项目。

get_input_processor (*field_name*)

返回给定字段的输入处理器。

get_output_processor (*field_name*)

返回给定字段的输出处理器。

`ItemLoader` 实例具有以下属性：

item

`Item` 此Item Loader正在解析的对象。

context

此项加载器的当前活动上下文。

default_item_class

一个Item类（或工厂），用于在构造函数中未给出时实例化项。

default_input_processor

默认输入处理器，用于那些未指定一个的字段。

default_output_processor

默认输出处理器，用于那些未指定的字段。

default_selector_class

如果只在构造函数中给出响应，则用于构造 `selector` this 的类 `ItemLoader` 。如果在构造函数中给出了选择器，则忽略此属性。有时在子类中重写此属性。

selector

`Selector` 从中提取数据的对象。它是构造函数中给出的选择器，或者是使用构造函数在构造函数中给出的响应创建的 `default_selector_class` 。此属性是只读的。

嵌套加载器

解析文档子节中的相关值时，创建嵌套加载器会很有用。想象一下，您从页面的页脚中提取详细信息，如下所示：

例：

```
<footer>
    <a class="social" href="https://facebook.com/whatever">Like Us</a>
    <a class="social" href="https://twitter.com/whatever">Follow Us</a>
    <a class="email" href="mailto:whatever@example.com">Email Us</a>
</footer>
```

如果没有嵌套的加载器，则需要为要提取的每个值指定完整的xpath（或css）。

例：

```
loader = ItemLoader(item=Item())
# load stuff not in the footer
loader.add_xpath('social', '//footer/a[@class = "social"]/@href')
loader.add_xpath('email', '//footer/a[@class = "email"]/@href')
loader.load_item()
```

相反，您可以使用页脚选择器创建嵌套加载程序并添加相对于页脚的值。功能相同但您避免重复页脚选择器。

例：

```
loader = ItemLoader(item=Item())
# load stuff not in the footer
footer_loader = loader.nested_xpath('//footer')
footer_loader.add_xpath('social', 'a[@class = "social"]/@href')
footer_loader.add_xpath('email', 'a[@class = "email"]/@href')
# no need to call footer_loader.load_item()
loader.load_item()
```

您可以任意嵌套加载器，它们可以使用xpath或css选择器。作为一般准则，当它们使代码更简单时使用嵌套的加载器，但不要过度使用嵌套，否则您的解析器将变得难以阅读。

重用和扩展项加载器

随着您的项目变得越来越大并且获得越来越多的蜘蛛，维护成为一个基本问题，特别是当您必须为每个蜘蛛处理许多不同的解析规则，有很多例外，但也想要重用通用处理器时。

项目加载器旨在减轻解析规则的维护负担，同时不失去灵活性，同时提供扩展和覆盖它们的便捷机制。因此，Item Loaders支持传统的Python类继承，以处理特定蜘蛛（或蜘蛛组）的差异。

例如，假设某个特定站点用三个破折号（例如）包含其产品名称，并且您不希望最终在最终产品名称中删除这些破折号。`---Plasma TV---`

以下是通过重用和扩展默认Product Item Loader（`ProductLoader`）来删除这些破折号的方法：

```
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)
```

扩展项加载器非常有用的另一种情况是，当您有多种源格式时，例如XML和HTML。在XML版本中，您可能希望删除`CDATA`出现的内容。以下是如何执行此操作的示例：

```
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)
```

这就是您通常扩展输入处理器的方式。

对于输出处理器，更常见的是在字段元数据中声明它们，因为它们通常仅依赖于字段而不依赖于每个特定站点解析规则（如输入处理器那样）。另请参见： 声明输入和输出处理器。

还有许多其他可能的方法来扩展，继承和覆盖您的Item Loaders，并且不同的Item Loaders层次结构可能更适合不同的项目。Scrapy只提供机制; 它不会强加您的Loaders集合的任何特定组织 - 这取决于您和您的项目的需求。

可用的内置处理器

尽管您可以使用任何可调用函数作为输入和输出处理器，但Scrapy提供了一些常用的处理器，如下所述。其中一些`MapCompose`（如通常用作输入处理器）组成了按顺序执行的几个函数的输出，以产生最终的解析值。

以下是所有内置处理器的列表：

类 scrapy.loader.processors.Identity

最简单的处理器，它什么都不做。它返回原始值不变。它不接收任何构造函数参数，也不接受Loader上下文。

例：

```
>>> from scrapy.loader.processors import Identity
>>> proc = Identity()
>>> proc(['one', 'two', 'three'])
['one', 'two', 'three']
```

类 scrapy.loader.processors.TakeFirst

从接收的值返回第一个非null /非空值，因此它通常用作单值字段的输出处理器。它不接收任何构造函数参数，也不接受Loader上下文。

例：

```
>>> from scrapy.loader.processors import TakeFirst
>>> proc = TakeFirst()
>>> proc(['', 'one', 'two', 'three'])
'one'
```

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

返回与构造函数中给定的分隔符连接的值，默认为。它不接受Loader上下文。`u' '`

使用默认分隔符时，此处理器等效于以下功能：`u' '.join`

例子：

```
>>> 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'
```

class **scrapy.loader.processors.Compose** (** functions , ** default_loader_context*)

一种处理器，它由给定功能的组合构成。这意味着该处理器的每个输入值都被传递给第一个函数，并且该函数的结果被传递给第二个函数，依此类推，直到最后一个函数返回该处理器的输出值。

默认情况下，停止处理`None`值。可以通过传递关键字参数来更改此行为`stop_on_none=False`。

例：

```
>>> from scrapy.loader.processors import Compose
>>> proc = Compose(lambda v: v[0], str.upper)
>>> proc(['hello', 'world'])
'HELLO'
```

每个函数都可以选择接收`loader_context`参数。对于那些处理器，该处理器将通过该参数传递当前活动的Loader上下文。

在构造函数中传递的关键字参数用作传递给每个函数调用的默认Loader上下文值。但是，传递给函数的最终Loader上下文值将被当前活动的Loader上下文覆盖，该上下文可通过该`ItemLoader.context()`属性访问。

class **scrapy.loader.processors.MapCompose** (** functions , ** default_loader_context*)

一种处理器，它由给定功能的组合构成，类似于`Compose`处理器。与此处理器的不同之处在于内部结果在函数之间传递的方式，如下所示：

*迭代*该处理器的输入值，并将第一个函数应用于每个元素。这些函数调用的结果（每个元素一个）被连接起来构造一个新的iterable，然后用于应用第二个函数，依此类推，直到最后一个函数应用于所收集的值列表的每个值，远。最后一个函数的输出值连接在一起以产生该处理器的输出。

每个特定函数都可以返回值或值列表，这些值使用应用于其他输入值的同一函数返回的值列表进行展平。函数也可以返回， `None` 在这种情况下，忽略该函数的输出以进行链上的进一步处理。

此处理器提供了一种方便的方法来组合仅使用单个值（而不是迭代）的函数。由于这个原因， `MapCompose` 处理器通常用作输入处理器，因为数据通常使用选择器的 `extract()` 方法提取，选择器的 方法返回unicode字符串列表。

以下示例应阐明其工作原理：

```
>>> 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处理器一样，函数可以接收Loader上下文，构造函数关键字参数用作默认上下文值。请参阅 `Compose` 处理器了解更多信息

class scrapy.loader.processors.SelectJmes (*json_path*)

使用提供给构造函数的json路径查询值并返回输出。需要 jmespath（https://github.com/jmespath/jmespath.py）才能运行。该处理器一次只能输入一个输入。

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合作：

```
>>> 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']
```