Item Loaders

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 7)

Unknown directive type "module".

```
.. module:: scrapy.loader
   :synopsis: Item Loader class
```

Item Loaders provide a convenient mechanism for populating scraped :ref:`items <topics-items>`. Even though items can be populated directly, Item Loaders provide a much more convenient API for populating them from a scraping process, by automating some common tasks like parsing the raw extracted data before assigning it.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 10); *backlink*

Unknown interpreted text role "ref".

In other words, :ref:`items <topics-items>` provide the *container* of scraped data, while Item Loaders provide the mechanism for *populating* that container.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 15); *backlink*

Unknown interpreted text role "ref".

Item Loaders are designed to provide a flexible, efficient and easy mechanism for extending and overriding different field parsing rules, either by spider, or by source format (HTML, XML, etc) without becoming a nightmare to maintain.

Note

Item Loaders are an extension of the itemloaders library that make it easier to work with Scrapy by adding support for :ref:`responses <topics-request-response>`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 23); *backlink*

Unknown interpreted text role "ref".

Using Item Loaders to populate items

To use an Item Loader, you must first instantiate it. You can either instantiate it with an :ref:`item object <topics-items>` or without one, in which case an :ref:`item object <topics-items>` is automatically created in the Item Loader `__init__` method using the :ref:`item <topics-items>` class specified in the :attr:`ItemLoader.default_item_class` attribute.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 30); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 30); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 30); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 30); *backlink*

Unknown interpreted text role "attr".

Then, you start collecting values into the Item Loader, typically using :ref:`Selectors <topics-selectors>`. You can add more than one value to the same item field; the Item Loader will know how to "join" those values later using a proper processing function.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 36); *backlink*

Unknown interpreted text role "ref".

Note

Collected data is internally stored as lists, allowing to add several values to the same field. If an `item` argument is passed when creating a loader, each of the item's values will be stored as-is if it's already an iterable, or wrapped with a list if it's a single value.

Here is a typical Item Loader usage in a :ref:`Spider <topics-spiders>`, using the :ref:`Product item <topics-items-declaring>` declared in the :ref:`Items chapter <topics-items>`:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 47); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 47); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 47); *backlink*

Unknown interpreted text role "ref".

```
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()
```

By quickly looking at that code, we can see the `name` field is being extracted from two different XPath locations in the page:

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

In other words, data is being collected by extracting it from two XPath locations, using the :meth:`~ItemLoader.add_xpath` method. This is the data that will be assigned to the `name` field later.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 69); *backlink*

Unknown interpreted text role "meth".

Afterwards, similar calls are used for `price` and `stock` fields (the latter using a CSS selector with the :meth:`~ItemLoader.add_css` method), and finally the `last_update` field is populated directly with a literal value (`today`) using a different method: :meth:`~ItemLoader.add_value`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 73); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 73); *backlink*

Unknown interpreted text role "meth".

Finally, when all data is collected, the :meth:`ItemLoader.load_item` method is called which actually returns the item populated with the data previously extracted and collected with the :meth:`~ItemLoader.add_xpath`, :meth:`~ItemLoader.add_css`, and :meth:`~ItemLoader.add_value` calls.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 78); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 78); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 78); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 78); *backlink*

Unknown interpreted text role "meth".

Working with dataclass items

By default, :ref:`dataclass items <dataclass-items>` require all fields to be passed when created. This could be an issue when using dataclass items with item loaders: unless a pre-populated item is passed to the loader, fields will be populated incrementally using the loader's :meth:`~ItemLoader.add_xpath`, :meth:`~ItemLoader.add_css` and :meth:`~ItemLoader.add_value` methods.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 89); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 89); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 89); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 89); *backlink*

Unknown interpreted text role "meth".

One approach to overcome this is to define items using the :func:`~dataclasses.field` function, with a `default` argument:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 95); *backlink*

Unknown interpreted text role "func".

```
from dataclasses import dataclass, field
from typing import Optional

@dataclass
class InventoryItem:
    name: Optional[str] = field(default=None)
    price: Optional[float] = field(default=None)
    stock: Optional[int] = field(default=None)
```

Input and Output processors

An Item Loader contains one input processor and one output processor for each (item) field. The input processor processes the extracted data as soon as it's received (through the :meth:`~ItemLoader.add_xpath`, :meth:`~ItemLoader.add_css` or :meth:`~ItemLoader.add_value` methods) and the result of the input processor is collected and kept inside the ItemLoader. After collecting all data, the :meth:`ItemLoader.load_item` method is called to populate and get the populated :ref:`item object <topics-items>`. That's when the output processor is called with the data previously collected (and processed using the input processor). The result of the output processor is the final value that gets assigned to the item.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 113); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 113); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 113); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 113); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 113); *backlink*

Unknown interpreted text role "ref".

Let's see an example to illustrate how the input and output processors are called for a particular field (the same applies for any other field):

```
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)
```

So what happens is:

1. Data from `xpath1` is extracted, and passed through the *input processor* of the `name` field. The result of the input processor is collected and kept in the Item Loader (but not yet assigned to the item).
2. Data from `xpath2` is extracted, and passed through the same *input processor* used in (1). The result of the input processor is appended to the data collected in (1) (if any).
3. This case is similar to the previous ones, except that the data is extracted from the `css` CSS selector, and passed through the same *input processor* used in (1) and (2). The result of the input processor is appended to the data collected in (1) and (2) (if any).
4. This case is also similar to the previous ones, except that the value to be collected is assigned directly, instead of being extracted from a XPath expression or a CSS selector. However, the value is still passed through the input processors. In this case, since the value is not iterable it is converted to an iterable of a single element before passing it to the input processor, because input processor always receive iterables.
5. The data collected in steps (1), (2), (3) and (4) is passed through the *output processor* of the `name` field. The result of the output processor is the value assigned to the `name` field in the item.

It's worth noticing that processors are just callable objects, which are called with the data to be parsed, and return a parsed value. So you can use any function as input or output processor. The only requirement is that they must accept one (and only one) positional argument, which will be an iterable.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 167)

Unknown directive type "versionchanged".

```
.. versionchanged:: 2.0
   Processors no longer need to be methods.
```

Note

Both input and output processors must receive an iterable as their first argument. The output of those functions can be anything. The result of input processors will be appended to an internal list (in the Loader) containing the collected values (for that field). The result of the output processors is the value that will be finally assigned to the item.

The other thing you need to keep in mind is that the values returned by input processors are collected internally (in lists) and then passed to output processors to populate the fields.

Last, but not least, itemloaders comes with some :ref:`commonly used processors <itemloaders:built-in-processors>` built-in for convenience.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 180); *backlink*

Unknown interpreted text role "ref".

Declaring Item Loaders

Item Loaders are declared using a class definition syntax. Here is an example:

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

class ProductLoader(ItemLoader):

    default_output_processor = TakeFirst()

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

    price_in = MapCompose(str.strip)

    # ...
```

As you can see, input processors are declared using the `_in` suffix while output processors are declared using the `_out` suffix. And you can also declare a default input/output processors using the :attr:`ItemLoader.default_input_processor` and :attr:`ItemLoader.default_output_processor` attributes.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 203); *backlink*

Unknown interpreted text role "attr".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 203); *backlink*

Unknown interpreted text role "attr".

Declaring Input and Output Processors

As seen in the previous section, input and output processors can be declared in the Item Loader definition, and it's very common to declare input processors this way. However, there is one more place where you can specify the input and output processors to use: in the :ref:`Item Field <topics-items-fields>` metadata. Here is an example:

**System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-

master\docs\topics\[scrapy-master][docs][topics]loaders.rst, line 214); *backlink*

Unknown interpreted text role "ref".

```
import scrapy
from itemloaders.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', ['Welcome to my', '<strong>website</strong>'])
>>> il.add_value('price', ['&euro;', '<span>1000</span>'])
>>> il.load_item()
{'name': 'Welcome to my website', 'price': '1000'}
```

The precedence order, for both input and output processors, is as follows:

1. Item Loader field-specific attributes: `field_in` and `field_out` (most precedence)
2. Field metadata (`input_processor` and `output_processor` key)
3. Item Loader defaults: :meth:`ItemLoader.default_input_processor` and :meth:`ItemLoader.default_output_processor` (least precedence)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst, line 250); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst, line 250); *backlink*

Unknown interpreted text role "meth".

See also: :ref:`topics-loaders-extending`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst, line 253); *backlink*

Unknown interpreted text role "ref".

Item Loader Context

The Item Loader Context is a dict of arbitrary key/values which is shared among all input and output processors in the Item Loader. It can be passed when declaring, instantiating or using Item Loader. They are used to modify the behaviour of the input/output processors.

For example, suppose you have a function `parse_length` which receives a text value and extracts a length from it:

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

By accepting a `loader_context` argument the function is explicitly telling the Item Loader that it's able to receive an Item Loader context, so the Item Loader passes the currently active context when calling it, and the processor function (`parse_length` in this case) can thus use them.

There are several ways to modify Item Loader context values:

1. By modifying the currently active Item Loader context (:attr:`~ItemLoader.context` attribute):

   System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 280); *backlink*

   Unknown interpreted text role "attr".

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

2. On Item Loader instantiation (the keyword arguments of Item Loader `__init__` method are stored in the Item Loader context):

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

3. On Item Loader declaration, for those input/output processors that support instantiating them with an Item Loader context. :class:`~processor.MapCompose` is one of them:

   System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 291); *backlink*

   Unknown interpreted text role "class".

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

ItemLoader objects

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 302)

Unknown directive type "autoclass".

```
.. autoclass:: scrapy.loader.ItemLoader
    :members:
    :inherited-members:
```

Nested Loaders

When parsing related values from a subsection of a document, it can be useful to create nested loaders. Imagine you're extracting details from a footer of a page that looks something like:

Example:

```
<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>
```

Without nested loaders, you need to specify the full xpath (or css) for each value that you wish to extract.

Example:

```
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()
```

Instead, you can create a nested loader with the footer selector and add values relative to the footer. The functionality is the same but you avoid repeating the footer selector.

Example:

```
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()
```

You can nest loaders arbitrarily and they work with either xpath or css selectors. As a general guideline, use nested loaders when they make your code simpler but do not go overboard with nesting or your parser can become difficult to read.

Reusing and extending Item Loaders

As your project grows bigger and acquires more and more spiders, maintenance becomes a fundamental problem, especially when you have to deal with many different parsing rules for each spider, having a lot of exceptions, but also wanting to reuse the common processors.

Item Loaders are designed to ease the maintenance burden of parsing rules, without losing flexibility and, at the same time, providing a convenient mechanism for extending and overriding them. For this reason Item Loaders support traditional Python class inheritance for dealing with differences of specific spiders (or groups of spiders).

Suppose, for example, that some particular site encloses their product names in three dashes (e.g. `---Plasma TV---`) and you don't want to end up scraping those dashes in the final product names.

Here's how you can remove those dashes by reusing and extending the default Product Item Loader (`ProductLoader`):

```
from itemloaders.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)
```

Another case where extending Item Loaders can be very helpful is when you have multiple source formats, for example XML and HTML. In the XML version you may want to remove `CDATA` occurrences. Here's an example of how to do it:

```
from itemloaders.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)
```

And that's how you typically extend input processors.

As for output processors, it is more common to declare them in the field metadata, as they usually depend only on the field and not on each specific site parsing rule (as input processors do). See also: :ref:`topics-loaders-processors-declaring`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]loaders.rst`, line 397); *backlink*

Unknown interpreted text role "ref".

There are many other possible ways to extend, inherit and override your Item Loaders, and different Item Loaders hierarchies may fit better for different projects. Scrapy only provides the mechanism; it doesn't impose any specific organization of your Loaders collection - that's up to you and your project's needs.