Item Pipeline

After an item has been scraped by a spider, it is sent to the Item Pipeline which processes it through several components that are executed sequentially.

Each item pipeline component (sometimes referred as just "Item Pipeline") is a Python class that implements a simple method. They receive an item and perform an action over it, also deciding if the item should continue through the pipeline or be dropped and no longer processed.

Typical uses of item pipelines are:

- cleansing HTML data
- validating scraped data (checking that the items contain certain fields)
- checking for duplicates (and dropping them)
- storing the scraped item in a database

Writing your own item pipeline

Each item pipeline component is a Python class that must implement the following method:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]item-pipeline.rst, line 28)

Unknown directive type "method".

```
.. method:: process_item(self, item, spider)

   This method is called for every item pipeline component.

   `item` is an :ref:`item object <item-types>`, see
   :ref:`supporting-item-types`.

   :meth:`process_item` must either: return an :ref:`item object <item-types>`,
   return a :class:`~twisted.internet.defer.Deferred` or raise a
   :exc:`~scrapy.exceptions.DropItem` exception.

   Dropped items are no longer processed by further pipeline components.

   :param item: the scraped item
   :type item: :ref:`item object <item-types>`

   :param spider: the spider which scraped the item
   :type spider: :class:`~scrapy.Spider` object
```

Additionally, they may also implement the following methods:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]item-pipeline.rst, line 49)

Unknown directive type "method".

```
.. method:: open_spider(self, spider)

   This method is called when the spider is opened.

   :param spider: the spider which was opened
   :type spider: :class:`~scrapy.Spider` object
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]item-pipeline.rst, line 56)

Unknown directive type "method".

```
.. method:: close_spider(self, spider)

   This method is called when the spider is closed.

   :param spider: the spider which was closed
   :type spider: :class:`~scrapy.Spider` object
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]item-pipeline.rst, line 63)

Unknown directive type "method".

```
.. method:: from_crawler(cls, crawler)

   If present, this classmethod is called to create a pipeline instance
   from a :class:`~scrapy.crawler.Crawler`. It must return a new instance
   of the pipeline. Crawler object provides access to all Scrapy core
   components like settings and signals; it is a way for pipeline to
   access them and hook its functionality into Scrapy.

   :param crawler: crawler that uses this pipeline
   :type crawler: :class:`~scrapy.crawler.Crawler` object
```

Item pipeline example

Price validation and dropping items with no prices

Let's take a look at the following hypothetical pipeline that adjusts the `price` attribute for those items that do not include VAT (`price_excludes_vat` attribute), and drops those items which don't contain a price:

```
from itemadapter import ItemAdapter
from scrapy.exceptions import DropItem
class PricePipeline:

    vat_factor = 1.15

    def process_item(self, item, spider):
        adapter = ItemAdapter(item)
        if adapter.get('price'):
            if adapter.get('price_excludes_vat'):
                adapter['price'] = adapter['price'] * self.vat_factor
            return item
        else:
            raise DropItem(f"Missing price in {item}")
```

Write items to a JSON file

The following pipeline stores all scraped items (from all spiders) into a single `items.jl` file, containing one item per line serialized in JSON format:

```
import json

from itemadapter import ItemAdapter

class JsonWriterPipeline:

    def open_spider(self, spider):
        self.file = open('items.jl', 'w')

    def close_spider(self, spider):
        self.file.close()

    def process_item(self, item, spider):
        line = json.dumps(ItemAdapter(item).asdict()) + "\n"
        self.file.write(line)
        return item
```

Note

The purpose of JsonWriterPipeline is just to introduce how to write item pipelines. If you really want to store all scraped items into a JSON file you should use the :ref:`Feed exports <topics-feed-exports>`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]item-pipeline.rst, line 126); *backlink*

Unknown interpreted text role "ref".

Write items to MongoDB

In this example we'll write items to MongoDB using pymongo. MongoDB address and database name are specified in Scrapy settings; MongoDB collection is named after item class.

The main point of this example is to show how to use :meth:`from_crawler` method and how to clean up the resources properly.:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]item-pipeline.rst, line 137); *backlink***
>
> Unknown interpreted text role "meth".

```
import pymongo
from itemadapter import ItemAdapter

class MongoPipeline:

    collection_name = 'scrapy_items'

    def __init__(self, mongo_uri, mongo_db):
        self.mongo_uri = mongo_uri
        self.mongo_db = mongo_db

    @classmethod
    def from_crawler(cls, crawler):
        return cls(
            mongo_uri=crawler.settings.get('MONGO_URI'),
            mongo_db=crawler.settings.get('MONGO_DATABASE', 'items')
        )

    def open_spider(self, spider):
        self.client = pymongo.MongoClient(self.mongo_uri)
        self.db = self.client[self.mongo_db]

    def close_spider(self, spider):
        self.client.close()

    def process_item(self, item, spider):
        self.db[self.collection_name].insert_one(ItemAdapter(item).asdict())
        return item
```

Take screenshot of item

This example demonstrates how to use :doc:`coroutine syntax <coroutines>` in the :meth:`process_item` method.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]item-pipeline.rst, line 178); *backlink***
>
> Unknown interpreted text role "doc".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]item-pipeline.rst, line 178); *backlink***
>
> Unknown interpreted text role "meth".

This item pipeline makes a request to a locally-running instance of Splash to render a screenshot of the item URL. After the request response is downloaded, the item pipeline saves the screenshot to a file and adds the filename to the item.

```
import hashlib
from urllib.parse import quote

import scrapy
from itemadapter import ItemAdapter
from scrapy.utils.defer import maybe_deferred_to_future


class ScreenshotPipeline:
    """Pipeline that uses Splash to render screenshot of
    every Scrapy item."""

    SPLASH_URL = "http://localhost:8050/render.png?url={}"

    async def process_item(self, item, spider):
        adapter = ItemAdapter(item)
        encoded_item_url = quote(adapter["url"])
        screenshot_url = self.SPLASH_URL.format(encoded_item_url)
        request = scrapy.Request(screenshot_url)
        response = await maybe_deferred_to_future(spider.crawler.engine.download(request, spider))
```

```
        if response.status != 200:
            # Error happened, return item.
            return item

        # Save screenshot to file, filename will be hash of url.
        url = adapter["url"]
        url_hash = hashlib.md5(url.encode("utf8")).hexdigest()
        filename = f"{url_hash}.png"
        with open(filename, "wb") as f:
            f.write(response.body)

        # Store filename in item.
        adapter["screenshot_filename"] = filename
        return item
```

Duplicates filter

A filter that looks for duplicate items, and drops those items that were already processed. Let's say that our items have a unique id, but our spider returns multiples items with the same id:

```
from itemadapter import ItemAdapter
from scrapy.exceptions import DropItem

class DuplicatesPipeline:

    def __init__(self):
        self.ids_seen = set()

    def process_item(self, item, spider):
        adapter = ItemAdapter(item)
        if adapter['id'] in self.ids_seen:
            raise DropItem(f"Duplicate item found: {item!r}")
        else:
            self.ids_seen.add(adapter['id'])
            return item
```

Activating an Item Pipeline component

To activate an Item Pipeline component you must add its class to the :setting:`ITEM_PIPELINES` setting, like in the following example:

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\scrapy-master\docs\topics\[scrapy-master][docs][topics]item-pipeline.rst`, line 254); *backlink*
>
> Unknown interpreted text role "setting".

```
ITEM_PIPELINES = {
    'myproject.pipelines.PricePipeline': 300,
    'myproject.pipelines.JsonWriterPipeline': 800,
}
```

The integer values you assign to classes in this setting determine the order in which they run: items go through from lower valued to higher valued classes. It's customary to define these numbers in the 0-1000 range.