DD2480 - Lab 4 Report

Eloi Dieme,
Hugo Malmberg,
Olivia Aronsson,
Lovisa Strange,
Yuta Ojima

March 2024

1 Project

Name: Scrapy

URL: `https://github.com/scrapy/scrapy`

Scrapy is a tool for extracting information from websites.

2 Onboarding experience

We decided to continue on the same project as before. We did spend some time looking at other projects, which were also written mostly in Python and seemed like large enough projects. However, we decided against some of them, since they did not have a lot of open issues. The Scrapy project did have quite a few open issues, and there seemed to be some issues that were well documented and that no one had started to work on. Therefore, we felt like continuing on the same project would be a good idea. We also considered the time we would have to spend on setting up the environment for a new project, and since we had already spent the time getting the project to run, as well as getting the tests to work, it felt more effective to continue on the same project.

In particular, we found a few issues (including the issue that we ended up choosing), that were very clearly defined and described what needed to be done. When having decided on an issue, we also commented on it that we were working on it, and asked if it was still relevant for the project. We asked since it was quite an old issue that did not have any recent discussions. When doing this, we received a quick reply.

$\frac{hours}{section}$	Eloi Dieme	Hugo Malmberg	Olivia Aronsson	Lovisa Strange	Yuta Ojima
1	4	4	4	4	4
2	1	1	0.5	1	1
3	4	2	4	4	2
4	0.5	1	-	0.5	0.5
5	2	1	3	1.5	2
6	4.5	3	3	6	1
7	4	3	0.5	4	4
8	1	1	1	1	3.5
Total	21	16	16	22	18

Table 1: Time Spent per members

3 Effort spent

Here, the number of hours spent on the project for each team member is documented. The work is split into 8 different categories, as can be seen below.

1. plenary discussions/meetings
2. discussions within parts of the group
3. reading documentation
4. configuration and setup
5. analyzing code/output
6. writing documentation
7. writing code
8. running code

The time spent per member can be found in Table 1. And here are brief contributions by each member

- **Eloi Dieme** :
Added item_processor method to fetch and call processor functions
- **Hugo Malmberg** : Added_process_item method to FeedExporter class and report writing.
- **Olivia Aronsson** : Added documentation for standard signature processor function
- **Lovisa Strange** : Modified item_scraped method to process items before exporting, added default settings for item processors, report writing
- **Yuta Ojima** : Add some util functions in FeedExporter class for manipulating FEEDS fields and also add test cases for each functions.

4 Overview of issue(s) and work done

Title: Add item_processors feature to Scrapy FeedExporter extension #5905

URL: `https://github.com/scrapy/scrapy/issues/5905`

Summary in one or two sentences: Create a new feature for Scrapy FeedExporter extension that allows the addition of methods to modify the content of items right before they get exported into the Feeds. This will enable exporting a single item as multiple entries independently on each Feed and making modifications on the item depending on the specific feed.

Scope (functionality and code affected): The FeedExporter class within the `scrapy/extensions/exporters.py` that is used to specify settings for exporting scraped data to various formats (XML, JSON, CSV etc...).

5 Requirements for the new feature or requirements affected by functionality being refactored

We identified the following requirements, using the documentation in the issue as well as looking at the structure of the affected code:

1. The system must allow users to configure one or more processor methods per feed in the FEEDS setting. These configurations should specify the path of the processing function and any necessary arguments.
2. The system must dynamically import and apply configured processor methods to items before they are exported.
3. Processor methods should be able to modify item fields, add fields, filter items, or split an item into multiple records.
4. The system must support different processor configurations for each feed, allowing items to be processed uniquely depending on the target feed.
5. The item processing should introduce minimal overhead to the feed export process
6. The design should be modular, allowing for easy addition of new types of processor methods without significant modification to the core FeedExporter code.
7. This feature should be backwards compatible, not affecting existing FeedExporter configurations without processor methods.

6 Code changes

6.1 Patch

To see the changes, run:

```
git diff 2d46b4acf 37db15e55
```

7 Test results

See `test_logs_fe_after.txt` on the GitHub repo for the logs of test on the `feedexport.py` file. The failed and skipped tests are related to cloud feed storage and connection in particular, which is not configured on our machines. For the tests on the whole package, see `test_logs_all_after.txt`. And for the logs before modifications, see the `.._before.txt` versions of those files.

8 UML class diagram and its description

In Figure 1 is a UML diagram consisting of the classes affected by the issue we have worked on, as well as some classes related to this class.

8.1 Key changes/classes affected

The FeedExporter class is the main class affected by the changes. Indeed, a new attribute was added to store the item processors, and the initialization logic was slightly altered as well to fetch those functions from the feed settings. On top of that, methods such as `_import_processor` and `_process_item` were added to handle the base logic of the feature.

This class is linked to a few other classes. Firstly, feed storage is handled by various sub classes each related to one storage solution (two of which can bee seen in the UML diagram). They all inherit from an interface named `IFeedStorage`.

Related to the feed exporter class, there is also a way to filter the items allowed to be exported to a feed. This is done by the `ItemFilter`-class. Another class that is related to this class is the `FeedSlot`-class, which handles all the parameters of a particular slot, as well as starts and ends the exporting process.

9 Overall experience

9.0.1 Experiences related to the project

This project has been both challenging and rewarding for us. Challenging on the one hand because we needed to really understand how Scrapy source code is structured in order to make modifications that do not break the software or go against the good practices used to build it in the first place. There is some pressure in the sense that the code we write should actually work for end-users in real world applications and production environments. Thus making us think twice before implementing most features. Understanding the existing documentation, as well as the conventions used in the Scrapy project, was one

of the most time consuming parts of the project. On the other hand, it was rewarding because we learned about software development on a larger code base than projects we have worked on in the past. By reading a lot of different issues when searching for a suitable one, we also learned about how contributing to open source projects works in the real world, as well as how the community around these projects looks like. We should be more confident now when working on production-grade software.

9.0.2 Self-assessment: Way of working

As a team, this project made us think collectively about many decisions (do we keep the same project, what issues do we choose, how to divide the tasks for a particular issue and how to interact with other contributors on the project). We still continued to work as before, using GitHub issues and Discord to ensure effective communication and simple handling of tasks. We have become comfortable using Discord as our main communication method, and we have found a structure for our meetings that works well for us. Additionally, we also have more experience using GitHub in a systematic way than during the start of the course which has made working on the project together easier. Therefore regarding the Essence standard, we still believe to be in the 'In-Place' state for this project.

To reach the next state of "Working Well" the team would need to focus on implementing practices to evaluate the ways of working more. As it is, the tools and practices are working to support the way the team naturally work on a basic level, but more reflection could be done to evaluate how the team actually work and if there are more efficient tools. For example, it has previously been discussed in meetings that it is hard when one ticket block another, and that sometimes it is hard to know what ticket blocks which team members work. Perhaps the addition of a tool that visualises the workflow and priority of tickets would support the team more than the current ticket tool.

It would also be beneficial for the team to discuss their individual use of practices and tools in a team forum, to share knowledge and help each other tune their use of the practices and tools.

Additionally, since this is the first time many of the group members are experiencing working together in this type of group and with the tools chosen it is hard to achieve that the team naturally applies the practices without thinking about them. Given more time that would certainly happen but as we are still in the learning phase this goal in the checklist cannot be seen as fulfilled, thus is the team currently at the "In Place" state rather than the "Working Well".

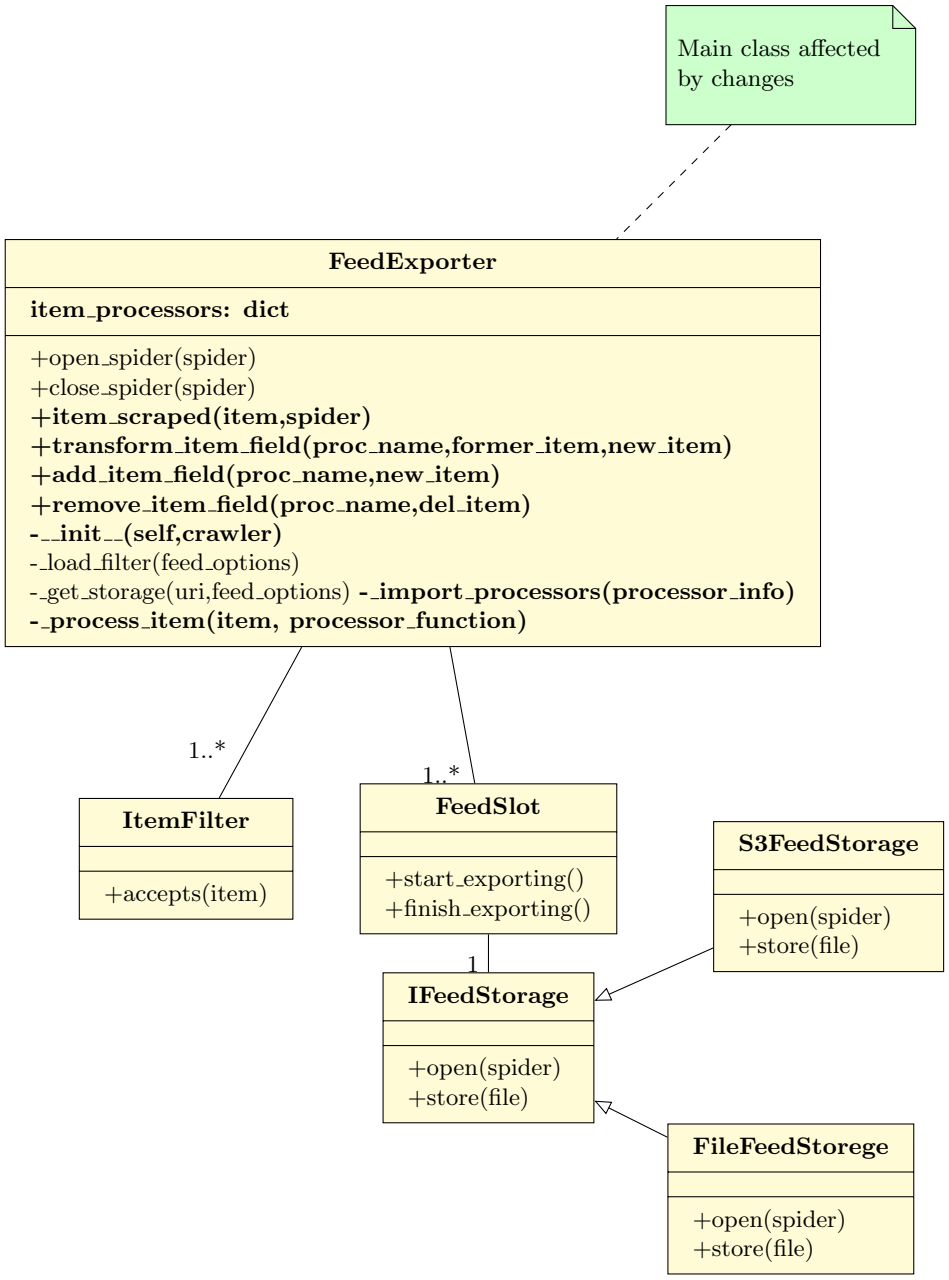

Figure 1: UML diagram over some of the classes affected by the changes made, as well as the changed or added functions and fields marked with **bold**. Not all classes or connections are represented in the diagram.