SCRAPY

Documentation
Version 0.12

Index

- **What is Scrapy?**

  Scrapy is a fast high-level screen scraping and web crawling framework, used to crawl websites and extract structured data from their pages. It can be used for a wide range of purposes, from data mining to monitoring and automated testing.

- **Running Scrapy**

  Scrapy can be run in many ways. To run scrapy, the user must first enter the directory of the project and run this from the command line.

  scrapy crawl zappos.com --set FEED_URI=filename.json --set FEED_FORMAT=json

  crawl is the keyword that instructs scrapy which activity to be performed and zappos.com is the name of the crawler that it needs to crawl it with. FEED_URI is generally used to specify the name of the file while FEED_FORMAT specifies the type in which the data should be stored.

  The above command line argument will generate a items.json file containing all scraped items, serialized in JSON.

  Instead of JSON, we can have the output in other formats as well. The other formats supported are csv, xml, pickle, marshal.

  For csv, the command line argument changes to

  scrapy crawl zappos.com --set FEED_URI=filename.csv --set FEED_FORMAT=csv

  For this project, csv format has been implemented.

- **Implementation of Scrapy specific to this project**

This project has been tailored specifically to extract the product reviews (womens shoes) from the Zappos website.

The extracted information is stored in a modularized format for the following categories

- product_name *: the name of the product*
- posted *: the date when the review was posted*
- reviewer *: the name of the reviewer*
- overall *: rating on 5 for overall product satisfaction*
- comfort *: rating on 5 for comfort of the product*
- style *: rating on 5 for style satisfaction*
- size *: comments on size support*
- arch *: comments on arch support*
- width : *comments on width support*
- Description *: the description written by the reviewer including the links in his desc.*
- product_id

- url *: the url of the product review page*

The spider which I have defined uses 2 sets to keep a tab on the links that need to be visited

- Visited
- Full links scan

On the first hit, the crawler scans the landing page of Zappos.com. It uses the category links on the left side of the page to crawl even deeper into Zappos. This was done so that, the flexibility of moving from one category to the other can be easily incorporated.

After having recognized the link for Women Shoes, the crawler digs deep into this link and collects all further subcategories of Women Shoes. Each of these links are stored in the memory and they are accessed linearly by the parser. Further when the crawler goes one more step deeper, it collects all the links of the products within each of these subcategories and stores it in visited set. Thus visited set contain list of all the products (from first page to the last page)of one subcategory which needs to be parsed to read the reviews.

Once the crawler reaches the review page (from the links in visited set), it then keeps a track of all the pages of the reviews and stores the url to these pages in Full links scan set. Once this is done, each of the urls from this set is accessed, parsed and the data which is extracted is stored in an array. Depending upon how we run the program, this data from the array can be stored in various formats.

An additional level of data can be scraped/accessed depending upon additional reviews of the zappos. Each of the zappos page has about 25 reviews and additional review are posted on different page. This scrapy implementation is designed to extract data from these pages as well.

Hence to summarize, the scrapy can go to a depth of 3 -4 levels of data to extract information about the review from the users.

- **Points to be noted relevant to the code**
    - The data which is extracted using xpath get stored is assigned as a list by scrapy. Since we are bundling one item object and each object represents a description, each of its attributes is stored as a list. Hence to avoid the Unicode tags, this data needs to be popped from the list and then stored in the excel sheet.
    - This code relies heavily on the CSS of the zappos page. Any changes to the zappos page in terms of CSS would need changes in the xpath expressions.
    - The code uses the data structure ‘set’ to eliminate duplicate urls.
    - The links are generally stored in set which resides in ram. Hence stopping the program/crawler midway cannot be then resumed to start the program from the very same point the next time.
    - If the data from one particular link cannot be scraped from scrapy, there is an inbuilt mechanism to queue the request, and that request will be automatically be accessed later (no fixed time limit)

- **Working of scrapy**

Scrapy initially initializes the connection and depending upon which parts to fetch from the, it will first check whether the page exists and then accordingly fetch the results. After fetching, it needs to be parsed to fetch only that required data.

Thought only data pertinent to women's shoes has been scraped in this project, if the data set could have contained a larger set (mens shoes, childrens shoes) etc, larger RAM size would be recommended since scrapy would store the results in its buffer temporarily.

- **Features of Scrapy**
  - Simple
    Scrapy was designed with simplicity in mind, by providing the features you need without getting in your way
  - Productive
    Just write the rules to extract the data from web pages and let Scrapy crawl the entire web site for you
  - Fast
    Scrapy is used in production crawlers to completely scrape more than 500 retailer sites daily, all in one server
  - Extensible
    Scrapy was designed with extensibility in mind and so it provides several mechanisms to plug new code without having to touch the framework core
  - Portable
    Scrapy runs on Linux, Windows, Mac and BSD
  - Open Source and 100% Python
    Scrapy is completely written in Python, which makes it very easy to hack
  - Well-tested
    Scrapy has an extensive test suite with very good code coverage

- **Versions of Scrapy**

  Current stable release: 0.12 (used in this project)

  Development version: 0.13

- **Scrapy Tutorial**

Scrapy Tutorial can be accessed from this link http://doc.scrapy.org/en/0.12/intro/tutorial.html

- **Zappos DOM structure**

Only the relevant parts specific to the project would be stated here

The Landing page:

Body

- ➔ Div id=”wrap”
  - ➔ Div id=”content” class=” priContent pageHomepage layoutHomepage”
    - ➔ Div class=”topLeft subCallout”
      - ➔ All the data is within the div with class attribute as catNav (multiple)
      - ➔ All the links for various categories are within this div

The Categories page:

Body

- ➔ Div id=”wrap”
  - ➔ Div id=”navCont”
    - ➔ Div id=”naviCenter”
      - ➔ Div id=”zc2Select”
        - ➔ All links are present within this div whose href attribute begins with “/women-boots/….”
  - ➔ Div id=”content” class=”searchpage”
    - ➔ Div id=”resultWrap”
      - ➔ Div class=”sort top”
        - ➔ Div class=”pagination”
          - ➔ All pages are placed as hyperlinks
          - ➔ Span class=”last” contains the link to the last page
        - ➔ Div class=”searchResults”
          - ➔ Each hyperlink leads to the particular product page

The Products page:

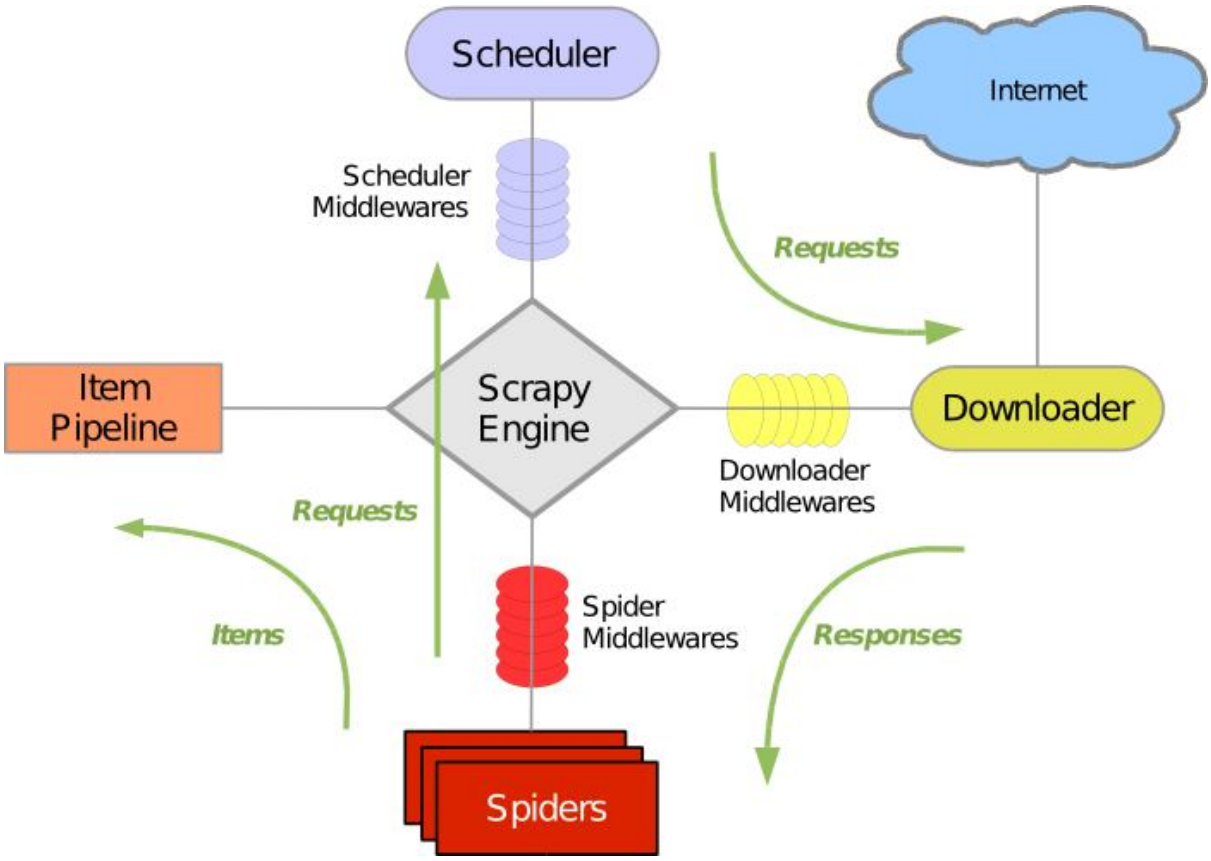
Body

- Div id=”wrap”
  - Div id=”content” class=”productPage”
    - div class="stripeOuter" id="contentWrapper"
      - div class="stripeInner"
        - span class="sku : Product ID
    - div id=”subWrapper”
      - div id=”other”
        - div class=”teethWhiteInner productReviews”
          - div class=”review”
            - div class="info"
              - Strong: Posted date
              - Strong: Reviewer name
              - Span strong : overall rating
              - Span strong: comfort
              - Span strong: style
            - Div class=”feel”
              - Span strong: shoe size
              - Span strong: shoe arch
              - Span strong: shoe width
            - Div class = “summary”
              - P: the description of review
          - Div class=”additional”
            - Hyperlink to additional reviews

Additional review page:

- Div id="wrap"
  - Div class="productReviews"
    - Div class="reviews"
      - Div class=" teethWhiteInner productReviews"
        - P class="top-pagination"
          - All the links to different pages full of reviews in <a>
        - div class="review"
          - div class="info"
            - Strong: Posted date
            - Strong: Reviewer name
            - Span strong : overall rating
            - Span strong: comfort
            - Span strong: style
          - Div class="feel"
            - Span strong: shoe size
            - Span strong: shoe arch
            - Span strong: shoe width
          - Div class = "summary"
            - P: the description of review
          - Div class="additional"
            - Hyperlink to additional reviews

- **Scrapy Architecture**



Components

- Scrapy Engine
  The engine is responsible for controlling the data flow between all components of the system, and triggering events when certain actions occur. See the Data Flow section below for more details.

- Scheduler
  The Scheduler receives requests from the engine and enqueues them for feeding them later (also to the engine) when the engine requests them.

- Downloader
  The Downloader is responsible for fetching web pages and feeding them to the engine which, in turn, feeds them to the spiders.

- Spiders
  Spiders are custom classes written by Scrapy users to parse responses and extract items (aka scraped items) from them or additional URLs (requests) to follow. Each spider is able to handle a specific domain (or group of domains). For more information see Spiders.

- Item Pipeline
  The Item Pipeline is responsible for processing the items once they have been extracted (or scraped) by the spiders. Typical tasks include cleansing, validation and persistence (like storing the item in a database). For more information see Item Pipeline.

- Downloader middlewares
  Downloader middlewares are specific hooks that sit between the Engine and the Downloader and process requests when they pass from the Engine to the Downloader, and responses that pass from Downloader to the Engine. They provide a convenient mechanism for extending Scrapy functionality by plugging custom code. For more information see Downloader Middleware.

- Spider middlewares
  Spider middlewares are specific hooks that sit between the Engine and the Spiders and are able to process spider input (responses) and output (items and requests). They provide a convenient mechanism for extending Scrapy functionality by plugging custom code. For more information see Spider Middleware.

- Scheduler middlewares
  Scheduler middlewares are specific hooks that sit between the Engine and the Scheduler and process requests when they pass from the Engine to the Scheduler and vice-versa. They provide a convenient mechanism for extending Scrapy functionality by plugging custom code.

Data flow

1. The data flow in Scrapy is controlled by the Engine, and goes like this:
2. The Engine opens a domain, locates the Spider that handles that domain, and asks the spider for the first URLs to crawl.
3. The Engine gets the first URLs to crawl from the Spider and schedules them in the Scheduler, as Requests.
4. The Engine asks the Scheduler for the next URLs to crawl.
5. The Scheduler returns the next URLs to crawl to the Engine and the Engine sends them to the Downloader, passing through the Downloader Middleware (request direction).
6. Once the page finishes downloading the Downloader generates a Response (with that page) and sends it to the Engine, passing through the Downloader Middleware (response direction).
7. The Engine receives the Response from the Downloader and sends it to the Spider for processing, passing through the Spider Middleware (input direction).

8. The Spider processes the Response and returns scraped Items and new Requests (to follow) to the Engine.
9. The Engine sends scraped Items (returned by the Spider) to the Item Pipeline and Requests (returned by spider) to the Scheduler
10. The process repeats (from step 2) until there are no more requests from the Scheduler, and the Engine closes the domain.

- Appendix
- **The Dynamics behind its functioning**

Scrapy is an application framework for crawling web sites and extracting structured data which can be used for a wide range of useful applications, like data mining, information processing or historical archival.

Even though Scrapy was originally designed for screen scraping (more precisely, web scraping), it can also be used to extract data using APIs (such as Amazon Associates Web Services) or as a general purpose web crawler

So you need to extract some information from a website, but the website doesn't provide any API or mechanism to access that info programmatically. Scrapy can help you extract that information.

Let's say we want to extract the URL, name, description and size of all torrent files added today in the Mininova site.

The list of all torrents added today can be found on this page:

http://www.mininova.org/today

Define the data you want to scrape

The first thing is to define the data we want to scrape. In Scrapy, this is done through Scrapy Items (Torrent files, in this case).

This would be our Item:

```
from scrapy.item import Item, Field

class Torrent(Item):

  url = Field()

  name = Field()

  description = Field()

  size = Field()
```

The next thing is to write a Spider which defines the start URL (http://www.mininova.org/today), the rules for following links and the rules for extracting the data from pages.

If we take a look at that page content we'll see that all torrent URLs are like http://www.mininova.org/tor/NUMBER where NUMBER is an integer. We'll use that to construct the regular expression for the links to follow: /tor/\d+.

We'll use XPath for selecting the data to extract from the web page HTML source. Let's take one of those torrent pages:

http://www.mininova.org/tor/2657665

And look at the page HTML source to construct the XPath to select the data we want which is: torrent name, description and size.

By looking at the page HTML source we can see that the file name is contained inside a <h1> tag:

```
<h1>Home[2009][Eng]XviD-ovd</h1>
```

An XPath expression to extract the name could be:

```
//h1/text()
```

Here is the spider code:

```
class MininovaSpider(CrawlSpider):

    name = 'mininova.org'
    allowed_domains = ['mininova.org']
    start_urls = ['http://www.mininova.org/today']
    rules = [Rule(SgmlLinkExtractor(allow=['/tor/\d+']), 'parse_torrent')]

    def parse_torrent(self, response):
        x = HtmlXPathSelector(response)

        torrent = TorrentItem()
        torrent['url'] = response.url
        torrent['name'] = x.select("//h1/text()").extract()
        torrent['description'] = x.select("//div[@id='description']").extract()
        torrent['size'] = x.select("//div[@id='info-left']/p[2]/text()[2]").extract()
        return torrent
```

For brevity's sake, we intentionally left out the import statements. The Torrent item is defined above.

Finally, we'll run the spider to crawl the site an output file scraped_data.json with the scraped data in JSON format:

```
scrapy crawl mininova.org --set FEED_URI=scraped_data.json --set FEED_FORMAT=json
```

This uses feed exports to generate the JSON file. You can easily change the export format (XML or CSV, for example) or the storage backend (FTP or Amazon S3, for example).

You can also write an item pipeline to store the items in a database very easily.

Review scraped data

If you check the scraped_data.json file after the process finishes, you'll see the scraped items there:

[{"url": "http://www.mininova.org/tor/2657665", "name": ["Home[2009][Eng]XviD-ovd"], "description": ["HOME - a documentary film by ..."], "size": ["699.69 megabyte"]},

 # ... other items ...

]

You'll notice that all field values (except for the url which was assigned directly) are actually lists. This is because the selectors return lists. You may want to store single values, or perform some additional parsing/cleansing to the values. That's what Item Loaders are for.

You've seen how to extract and store items from a website using Scrapy, but this is just the surface. Scrapy provides a lot of powerful features for making scraping easy and efficient, such as:

Built-in support for selecting and extracting data from HTML and XML sources

Built-in support for cleaning and sanitizing the scraped data using a collection of reusable filters (called Item Loaders) shared between all the spiders.

Built-in support for generating feed exports in multiple formats (JSON, CSV, XML) and storing them in multiple backends (FTP, S3, local filesystem)

A media pipeline for automatically downloading images (or any other media) associated with the scraped items

Support for extending Scrapy by plugging your own functionality using signals and a well-defined API (middlewares, extensions, and pipelines).

Wide range of built-in middlewares and extensions for:

- cookies and session handling
- HTTP compression
- HTTP authentication
- HTTP cache
- user-agent spoofing
- robots.txt
- crawl depth restriction
- and more