fodio Documentation

Release 1.0.0

Nathan Zilora

Jul 17, 2018

Table of Contents

CHAPTER 1

fodio API

1.1 FodioObj

class fodio.**FodioObj**
Bases: object

This class does literally nothing. It's inherited by Item and ItemAttr, so ItemMeta knows what to deal with.

1.2 ItemAttr

class fodio.**ItemAttr**(*css_selector: str*, *accept_multiples: bool = False*, *raise_not_found: bool = True*)
Bases: *fodio.FodioObj*

This is the parent class of all other Attr classes.

Variables

- **selector** – The value passed in as css_selector
- **accept_multiples** – The value passed in as accept_multiples
- **raise_not_found** – The value passed in as raise_not_found

1.3 TextAttr

class fodio.**TextAttr**(*css_selector: str*, *accept_multiples: bool = False*, *raise_not_found: bool = True*)
Bases: *fodio.ItemAttr*

Finds the the text from a css selector.

load(*document: str*) → Union[List[str], NoneType, str]
Get the text from the matched contents from _find.

Parameters **document** (*str*) – The HTML relative to the item.

Returns Either a list of strings (if accept_multiples), None (if not raise_not_found) or a string.

Return type Union[List[str], None, str]

1.4 LinkAttr

class fodio.**LinkAttr**(*css_selector: str*, *accept_multiples: bool = False*, *raise_not_found: bool = True*)

Bases: *fodio.ItemAttr*

Finds the text in an a tag, along side it's href attribute based on the css selector

Variables ***LINK*** – A named tuple representing the "link". contains .text and .url

LINK

alias of Link

load(*document: str*) → Union[List[<function NamedTuple at 0x7f155395cc80>], NoneType, <function NamedTuple at 0x7f155395cc80>]

Get the link from the matched contents from _find.

Parameters **document** (*str*) – The HTML relative to the item.

Returns Either a list of LinkAttr.LINK (if accept_multiples), None (if not raise_not_found) or a LinkAttr.LINK.

1.5 CustomAttr

class fodio.**CustomAttr**(*attrs: Iterable[str]*, *css_selector: str*, *accept_multiples: bool = False*, *raise_not_found: bool = True*)

Bases: *fodio.ItemAttr*

This is a ItemAttr in which you can obtain any of a node's attributes. If raise_not_found is False, if it can't find an attribute on the node, the value will be None instead,

Variables **value** – An Iterable containing the values passed into attrs

load(*document: str*) → Union[List[Union[dict, NoneType]], NoneType, dict]

Get the node's attribues based on the document.

Parameters **document** (*str*) – The HTML relative to the css_selector.

Returns Either nothing if not taise_not_found, or a dict / list of dicts with the attr names -> attr values.

Return type Union[List[Union[dict, None]], None, dict]

1.6 ItemMeta

class fodio.**ItemMeta**

Bases: type

Add all class variables that inherit FodioObj to a _ATTRS class variables

1.7 Item

class fodio.**Item**
Bases: *fodio.FodioObj*

An object to represent data on a page. To use, create class variables with Attr objects pointed at the data you desire. They will all share the first css selector passed in by the page class.

It's also important to note that you MUST INCLUDE A META CLASS. For example,

```
>>> class SomeSite(Item):
...     ...
...     class Meta:
...         selector = ".hello-there"
...         root_url = "https://some.site"
```

Variables **_ATTRS** – A list containing the names for the ItemAttrs.

classmethod **from_html**(*document: str*) → Union[Dict[str, Any], List[Dict[str, Any]]]
Load a HTML document for parsing, and a shared the selected segment with ItemAttrs.

Parameters **document** (*str*) – The whole HTML page.

Returns A dict with the keys as class var names to the ItemAttrs, and values as the parsed data. This will be a list if multiple entries for the Meta selector are found.

Return type Union[Dict[str, Any], List[Dict[str, Any]]]

classmethod **load**(*document: str*) → Dict[str, Any]
Shorthand for from_html. Mainly used to make item objects compatible as ItemAttrs.

See from_html for more information.

classmethod **search**(*url_path: str*) → Dict[str, Any]
Fetch for the URL and parse the document based on the items.

Parameters **url_path** (*str*) – The URL path in the Meta.root_url site to parse.

Returns A dict with the keys being the class variables and values as the loaded items.

Return type Dict[str, *Item*]

CHAPTER 2

fodio errors

2.1 FodioException

class fodio.errors.**FodioException**
Bases: Exception

2.2 InformationError

class fodio.errors.**InformationError**
Bases: *fodio.errors.FodioException*

What is fodio?

Fodio is a web scraping tool, used to easily traverse websites and collects data on. Some key concepts fodio was built on was simplicity and asynchronous-ity (No, I don't think that's a real word). Inspired by the Demiurge library, which is not active anymore, Fodio is here to save the day!

Why would I not use Scrapy?

Good Question! There are some key differences that are important too note.

1. Scrapy uses C extensions, which depending on the environment, can be a pain!
2. Scrapy is not asynchronous!
3. IMO, This is simpler.

Also, if you're on linux, you can use uvloop, which should help speed things up!

CHAPTER 3

Installation

To install with pypi, use simply `pip install fodio`

To install manually, either download and unpack, or clone this repository. Then, while in the root of the local repository, do `python setup.py install`

I'm not great at this documentaion thing, so let's just get right into the quick-start.

CHAPTER 4

Quick-Start

Too start scraping, inspect element will be your friend. You're going to be copying the CSS selectors in the elements you wish to harvest. In this case, we'll be walking though a github example that you can also find in the demo file, on the Fodio github page.

First, we need to make an item. Then "item" will represent a section of the page.

It's good too note, that this example won't have Items as ItemAttrs, but you can do that. I really suggest looking at the demos file, because that includes a bunch of great examples on what to do.

```
class GithubPage(Item):
    ...
    class Meta:
        ...
```

Notice the "Meta" class. This is used too hold your selector and root_url information. In this senerio, we want to get a github user's real name + description. First, let's narrow in on the profile card with the ".h-card" selector. Then, we'll supply the root_url "https://github.com"

```
class GithubPage(Item):
    ...
    class Meta:
        selector = ".h-card"
        root_url = "https://github.com"
```

Not that bad eh? Now that we have the profile card, we need to set some ItemAttributes to extract the information. We'll use a TextAttr object to extract the text from a node with the real name, and anther TextAttr for the description.

```
class GithubPage(Item):
    name = TextAttr(".p-name")
    desc = TextAttr(".p-note > div:nth-child(1)", raise_not_found=False)

    class Meta:
        selector = ".h-card"
        root_url = "https://github.com"
```

Note: Notice the `raise_not_found` kwarg. Some github profiles don't have a description, and we don't want it too raise an error if they don't. The same if True for the real name, however that element will exist if they have a real name or not, so we'll have to check manually.

Now time too run it! we'll create an asynchronous function called runner to do this. GithubPage has a classmethod called search which will add a path too the root_url, and fetch it. So let's do that.

```
accounts = ['Zwork101', 'bukson', 'torvalds', 'User1m', 'random-robbie']
async def runner():
    result = await GithubPage.search(account)
    print("{account_name}'s real name is {real_name}".format(
        account_name=account, real_name=result['name'] if result['name'] else "not␣
↪known"))
    print('{description}\n'.format(
        description=('"' + result['desc'] + '"') if result['desc'] else "No␣
↪Description"))
```

And we're done! After adding asyncio too run it, this is our final product:

```
import asyncio

from fodio import Item, TextAttr


class GithubPage(Item):
        name = TextAttr(".p-name")
        desc = TextAttr(".p-note > div:nth-child(1)", raise_not_found=False)

        class Meta:
            selector = ".h-card"
            root_url = "https://github.com"

accounts = ['Zwork101', 'bukson', 'torvalds', 'User1m', 'random-robbie']

async def runner():
    result = await GithubPage.search(account)
    print("{account_name}'s real name is {real_name}".format(
        account_name=account, real_name=result['name'] if result['name'] else "not␣
↪known"))
    print('{description}\n'.format(
        description=('"' + result['desc'] + '"') if result['desc'] else "No␣
↪Description"))

loop = asyncio.get_event_loop()
loop.run_until_complete(runner())
```

That wasn't that bad now was it? For more examples, see the demo folder. Otherwise, get scraping!

Wait, but why is it called "Fodio"

Well, apparently developers like to name their projects in latin, or some other language no one uses. Fodio (atleast this is what some sites said) can mean "delve" which is an english word for all you non-english majors / non-Magic: The Gathering players. Delve means to research and dig, 2 things this library does!

Index