Scrapbook

Release 1

odoku

Mar 22, 2018

Contents

Scrapbook is simple scraping library.

```
from scrapbook import Element, Content
import requests


class Twitter(Content):
    username = Element(
        xpath='//*[@id="page-container"]/div[2]/div/div'
              '/div[1]/div/div/div/div[1]/h2/a/span/b/text()',
    )
    screen_name = Element(
        xpath='//*[@id="page-container"]/div[2]/div/div/'
              'div[1]/div/div/div/div[1]/h1/a/text()',
    )


response = requests.get('https://twitter.com/odoku')
data = Twitter().parse(response.text)

print(data)
```

CHAPTER 1

Requirements

- Python 2.7 or Python 3.3+

CHAPTER 2

Installation

```
pip install scrapbook
```

CHAPTER 3

Page

3.1 Element

Get the element specified by xpath from HTML.

```
from scrapbook import Element
import requests

response = requests.get('https://twitter.com/odoku')
screen_name = Element(
    xpath='//*[@id="page-container"]/div[2]/div/div'
          '/div[1]/div/div/div/div[1]/h2/a/span/b/text()',
)
name = screen_name.parse(response.text)

print(name)
```

3.1.1 Arguments

```
Element(
    xpath: Optional[str] = None,
    filter: Union[Callable, str, list[Union[Callable, str]] = scrapbook.filters.
↪through,
    parser: Union[Callable, str] = scrapbook.parsers.First(),
)
```

xpath

Specify the xpath of the element you want to retrieve.

```
el = Element(xpath='/html/body/p/text()')
texts = el.parse()
```

filter

Any processing can be performed on the acquired value.

```
def clean(values):
    return [v.strip() for v values]

el = Element(xpath='/html/body/p/text()', filter=clean)
```

More than one filter can be specified.

```
def to_int(value):
    return [int(v.strip()) for v values]

Element(xpath='/html/body/p/text()', filter=[to_int, sum])
```

parser

You can specify a function to parse the element specified by xpath.

```
def parse_link(selector):
    # selector is parsel.SelectorList
    return {
        'url': selector.xpath('./@href').extract_first(),
        'text': selector.xpath('./text()').extract_first(),
    }

Element(xpath='/html/body/a', parser=parse_link)
```

3.1.2 Methods

parse

```
parse(html: Union[str, parsel.Selector, parsel.SelectorList])
```

Parse html.

```
html = '<html><body><p>Hello!</p></body></html>'
el = Element(xpath='/html/body/p/text()')
text = el.parse()  # Hello!
```

3.2 Content

You can handle multiple Elements at once.

```
from scrapbook import Element, Content
import requests


class Twitter(Content):
    username = Element(
        xpath='//*[@id="page-container"]/div[2]/div/div'
              '/div[1]/div/div/div/div[1]/h2/a/span/b/text()',
    )
    screen_name = Element(
        xpath='//*[@id="page-container"]/div[2]/div/div/'
              'div[1]/div/div/div/div[1]/h1/a',
    )


response = requests.get('https://twitter.com/odoku')
data = Twitter().parse(response.text)

print(data)
```

3.2.1 Include filter/parser functions

You can define the filter / parser specified in the Element in the Content.

```
class Page(Content):
    username = Element(
        xpath='//*[@id="username"]',
        parse='parse_username',
        filter='filter_username',
    )

    def parse_username(self, selector):
        return selector.xpath('./text()').extract_first()

    def filter_username(self, value):
        return value.replace('username: ', '').strip()
```

3.2.2 Nest

Content can be nested.

```
class Profile(Content):
    username = Element(xpath='./path/to/username/text()')
    screen_name = Element(xpath='./path/to/screen_name/text()')

class Page(Content):
    profile = Profile(xpath='//*[@id="profile"]')
```

3.2.3 Inheritance

Content supports inheritance.

```
class Common(Content):
    title = Element(xpath='/path/to/title/text()')

class ProjectPage(Common):
    name = Element(xpath='/path/to/name/text()')

class TeamPage(Common):
    name = Element(xpath='/path/to/name/text()')
```

3.2.4 Arguments

```
Content(
    xpath: Optional[str] = None,
    filter: Union[Callable, str, list[Union[Callable, str]] = scrapbook.filters.
↪through,
    many: bool = False,
)
```

xpath

Specify the xpath of the element you want to parse. For the included Element, the element of the specified xpath is passed.

```
class Page(Content):
    username = Element(xpath='./span[1]/text()')

page = Page(xpath='//*[@id="profile"]')
data = page.parse(html)
```

filter

You can do arbitrary processing on the acquired value. As with Element, multiple filters can be specified.

```
class Page(Content):
    username = Element(xpath='./span[1]/text()')

def rename(value):
    alias = {'username': 'account'}
    return {alias.get(k, k): v for k, v in value.items()}

page = Page(xpath='//*[@id="profile"]', filter=rename)
data = page.parse(html)
```

many

If there are multiple elements specified by xpath, you can get it as a list by specifying *many* = *True*.

```
class Comemnt(Content):
    text = Element(xpath='./text()')

class Article(Content):
```

```
    title = Element(xpath='//*[@id="title"]')
    content = Element(xpath='//*[@id="content"]')
    comments = Comment(xpath='//*[@id="content-list"]/li', many=True)

article = Article()
data = article.parse(html)
```

3.2.5 Methods

parse

```
parse(
    html: Union[str, parsel.Selector, parsel.SelectorList],
    object: Optional[Any],
)
```

Parse html.

```
class Page(Content):
    content = Element(xpath='/html/body/p/text()')

html = '<html><body><p>Hello!</p></body></html>'
page = Page()
data = page.parse(html)  # {'content': 'Hello!'}
```

Map the value to the object specified in the object argument.

```
instance = PageModel()
page = Page()
instance = page.parse(html, object=instance)
```

3.2.6 Class Methods

inline

```
inline(
    xpath: str = None,
    filter: Union[Callable, str, list[Union[Callable, str]] = scrapbook.filters.
↪through,
    **attrs: Dict[str, Any]
)
```

Returns an instance of dynamically generated Content class.

```
class Page(Content):
    content = Content.inline(
        text=Element(xpath='/html/body/p/text()', filter='twice'),
    )

    def twice(self, value):
        return value * 2

html = '<html><body><p>Hello!</p></body></html>'
```

```
page = Page()
data = page.parse(html)  # {'content': {'text': 'Hello!Hello!'}}
```

3.3 Filters

By using various filters for Element or Content, you can set the retrieved value to your preferred format.

```
el = Element(
    xpath='//html/body/ul/li',
    filter=[
        Map(
            clean_text,
            Normalize(),
            Fetch(r'(?P<key>.+): (?P<count>\d+)'),
        ),
        lambda values: {v['key']: v['count'] for v in values},
    ],
)
```

3.3.1 Map

Execute the filter specified by argument for each element of list or dict.

```
filter = Map(clean_text, Equals('yes'))
result = filter({
    'AAA': '    no    ',
    'BBB': '    yes    ',
    'CCC': '    <strong>   yes  <strong>    ',
})

assert {
    'AAA': False,
    'BBB': True,
    'CCC': True,
} == result
```

It is also possible to call functions defined in the Content class.

```
class Page(Content):
    links = Element(xpath='//a/@href', parser=All(), filter=Map('filter_link'))

    def filter_link(self, value):
        url = urlparse(value)
        return url.netloc

page = Page(xpath='')
result = page.parse('''
    <a href="http://google.com">Google</a>
    <a href="http://twitter.com">Twitter</a>
    <a href="http://facebook.com">Facebook</a>
''')

assert {
    'links': [
```

```
        'google.com',
        'twitter.com',
        'facebook.com',
    ]
} == result
```

3.3.2 Through

It returns the passed value as it is. This is the default filter for Element / Content.

```
assert 10 == through(10)
```

3.3.3 TakeFirst

Get the first element of list. However, if the acquired element is None or ‘’, the next element is acquired.

```
assert 10 == take_first([None, '', 10])
```

3.3.4 CleanText

Perform the following cleaning process on the character string.

- Removing HTML tags
- Decode HTML special characters
- Make 2 spaces or more of one contiguous space
- Remove Whitespace before and after

```
clean_text = CleanText()
assert 'aaa & bbb' == clean_text('<p>  aaa  &amp;  bbb  </p>')
```

You can specify how to handle empty values.

```
clean_text = CleanText(empty_value='empty')
assert 'empty' == clean_text('')
```

You can also replace the line feed code with a space.

```
clean_text = CleanText(remove_line_breaks=True)
assert 'a b' == clean_text('a\nb')
```

3.3.5 Equals

Returns True if the value matches the specified string.

```
equals = Equals('yes')
assert equals('yes')
```

3.3.6 Contains

Returns True if the specified character string is included in the character string.

```
contains = Contains('B')
assert contains('ABC')
```

3.3.7 Fetch

Extract values from strings using regular expressions.

```
fetch = Fetch(r'\d+')
assert '100' == fetch('Price: $100')
```

You can also get all matched values.

```
fetch = Fetch(r'\d+', all=True)
assert ['100', '20'] == fetch('Price: $100, Amount: 20')
```

It can also be returned as dict by specifying label.

```
fetch = Fetch(r'Price: $(?P<price>\d+), Amount: (?P<amount>\d+)')
assert {'price': '100', 'amount': '20'} == fetch('Price: $100, Amount: 20')
```

3.3.8 Replace

You can replace the string using regular expressions.

```
replace = Replace(r'A+', 'A')
assert 'ABC' == replace('AAAAABC')
```

3.3.9 Join

Returns a string formed by combining list with separator.

```
join = Join(',')
assert 'A,B,C' == join(['A', 'B', 'C'])
```

3.3.10 Split

Split a string into a list.

```
split = Split(',')
assert ['A', 'B', 'C'] == split('A,B,C')
```

3.3.11 Normalize

Returns the normalized string.

```
normalize = Normalize()
assert '12AB&%' == normalize('')
```

3.3.12 RenameKey

Rename the dict's key.

```
rename_key = RenameKey({'AAA': 'BBB'})
assert {'BBB': 10} == rename_key({'AAA': 10})
```

3.3.13 FilterDict

Returns dict with only the specified key.

```
filter_dict = FilterDict(['AAA', 'BBB'])
assert {'AAA': 10, 'BBB': 20} == filter_dict({'AAA': 10, 'BBB': 20, 'CCC': 30})
```

Other than the specified key can be returned.

```
filter_dict = FilterDict(['AAA', 'BBB'], ignore=True)
assert {'CCC': 30} == filter_dict({'AAA': 10, 'BBB': 20, 'CCC': 30})
```

3.3.14 Partial

You can execute it by specifying partial arguments to the function.

```
def add(a, b, c):
    return a + b + c

result = Partial(add, kwargs={'a': 10, 'c': 30}, arg_name='b')(20)
assert 60 == result
```

The Partial filter handles empty values safely, so it is convenient to use it as a wrapper for functions.

```
assert Partial(int)('') is None
assert Partial(int)('10') == 10
```

3.3.15 DateTime

Converts a Datetime String to a Datetime object.

```
parse_dt = DateTime()
assert datetime(2001, 2, 3, 4, 5, 6) == parse_dt('2001-02-03 04:05:06')
```

You can also handle timezone.

```
parse_dt = DateTime()
result = parse_dt('2001-02-03T04:05:06+09:00')
assert datetime(2001, 2, 3, 4, 5, 6, 0, tzoffset(None, 3600 * 9)) == result
```

Unnecessary information can be truncated.

```
parse_dt = DateTime(truncate_timezone=True)
result = parse_dt('2001-02-03T04:05:06+09:00')
assert datetime(2001, 2, 3, 4, 5, 6) == result
```

```
parse_dt = DateTime(truncate_time=True)
result = parse_dt('2001-02-03T04:05:06+09:00')
assert date(2001, 2, 3) == result
```

You can also specify the format.

```
parse_dt = DateTime(format='%d %m %Y')
result = parse_dt('01 02 2003')
assert datetime(2003, 2, 1) == result
```

3.3.16 Bool

Convert string to Bool type.

```
parse_bool_string = Bool()
assert parse_bool_string('true')
```

You can specify a string to treat as True.

```
parse_bool_string = Bool('OK', 'ok')
assert parse_bool_string('OK')
```

3.4 Parsers

3.4.1 First

Get only one element matching the specified xpath. This is the default parser of Element.

```
<html>
    <body>
        <p>
            AAA
            <br>
            BBB
            <br>
            CCCC
        </p>
    </body>
</html>
```

```
el = Element(xpath='//html/body/p/text()', parser=First)
text = el.parse(html)

assert 'AAA' == text
```

3.4.2 All

It converts all elements matching xpath into text and returns it as list.

```
<html>
    <body>
        <p>
            AAA
            <br>
            BBB
            <br>
            CCCC
        </p>
    </body>
</html>
```

```
el = Element(xpath='//html/body/p/text()', parser=All)
texts = el.parse(html)

assert ['AAA', 'BBB', 'CCC'] == texts
```

3.4.3 ParseTable

Parse basic table and return it as list.

```
<html>
    <body>
        <table>
            <tr>
                <th>Company</th>
                <th>Contact</th>
                <th>Country</th>
            </tr>
            <tr>
                <td>Alfreds Futterkiste</td>
                <td>Maria Anders</td>
                <td>Germany</td>
            </tr>
            <tr>
                <td>Centro comercial Moctezuma</td>
                <td>Francisco Chang</td>
                <td>Mexico</td>
            </tr>
        </table>
    </body>
</html>
```

```
el = Element(xpath='//html/body/table', parser=ParseTable())
data = el.parse(html)

assert [
    ['Alfreds Futterkiste', 'Maria Anders', 'Germany'],
    ['Centro comercial Moctezuma', 'Francisco Chang', 'Mexico'],
] == data
```

If there is a header in table, passing *has_header* = *True* will return dict with the value of header as key.

```
el = Element(xpath='//html/body/table', parser=ParseTable(has_header=True))
data = el.parse(html)

assert [
    {
        'Company': 'Alfreds Futterkiste',
        'Contact': 'Maria Anders',
        'Country': 'Germany',
    },
    {
        'Company': 'Centro comercial Moctezuma',
        'Contact': 'Francisco Chang',
        'Country': 'Mexico',
    },
] == data
```

3.4.4 ParseList

Parse elements such as <*ul*> and <*ol*> and return them as list.

```
<html>
    <body>
        <ol>
            <li>Coffee</li>
            <li>Tea</li>
            <li>Milk</li>
        </ol>
    </body>
</html>
```

```
el = Element(xpath='//html/body/ol', parser=ParseList())
data = el.parse(html)

assert ['Coffee', 'Tea', 'Milk'] == data
```

3.4.5 ParseDefinitionList

It parses <*dl*> and returns it as dict.

```
<html>
    <body>
        <dl>
            <dt>Coffee</dt>
            <dd>black hot drink</dd>
            <dt>Milk</dt>
            <dd>white cold drink</dd>
            <dd>white hot drink</dd>
        </dl>
    </body>
</html>
```

```
el = Element(xpath='//html/body/dl', parser=ParseDefinitionList())
data = el.parse(html)
```

```
assert {
    'Coffee': 'black hot drink',
    'Milk': [
        'white cold drink',
        'white hot drink',
    ]
} = data
```