demiurge Documentation

Release 0.2

Matias Bordese

September 20, 2014

Contents

PyQuery-based scraping micro-framework. Supports Python 2.x and 3.x.

Source code: https://github.com/matiasb/demiurge/

CHAPTER 1

Changes

0.2:

- Added docs.
- Added RelatedItem.
- Added field clean support.

0.1:

- Initial release.

CHAPTER 2

Installation

From PyPI:

```
pip install demiurge
```

CHAPTER 3

Usage

You can define items to be scraped using a declarative (Django-inspired) syntax:

```
import demiurge

class Torrent(demiurge.Item):
    url = demiurge.AttributeValueField(
        selector='td:eq(2) a:eq(1)', attr='href')
    name = demiurge.TextField(selector='td:eq(2) a:eq(2)')
    size = demiurge.TextField(selector='td:eq(3)')

    class Meta:
        selector = 'table.maintable:gt(0) tr:gt(0)'
        base_url = 'http://www.mininova.org'
```

At the moment, there are only two available fields: *TextField* and *AttributeValueField*. A *TextField* expects an optional *selector* argument, meanwhile *AttributeValueField* possible arguments are *selector* and *attr*.

selector specifies the PyQuery selector for the element, relative to the *Item* element (determined by the Meta *selector* attribute). If not specified, the current *Item* element is assumed.

On the other hand, *attr* parameter allows to retrieve an element attribute value instead of its text content.

In the example above, the *Item* selector is any row but not the first one, from the table with CSS class *maintable* (also ignoring the first ocurrence, sponsored results).

Each field selector is relative to the *Item* element (in this case, a table row). Then, *name* refers to the second anchor in the second cell.

New in version 0.2: Added clean_<fieldname> method support.

If you need an extra cleanup or processing for a field (for instance, to apply a regex), you can define a method *clean_<fieldname>* in your class. This method will receive the retrieved value for the field and should return the cleaned one. For example:

```
class Torrent(demiurge.Item):
    ...
    size = demiurge.TextField(selector='td:eq(3)')

    def clean_size(self, value):
        return value.replace('Mb', 'MB')
```

Once you defined your items, there are a couple of useful methods you can use, both expecting as argument a relative path to the *Item base_url* if it was defined in the *Item.Meta* class, or a full URL (if *base_url* was not specified):

```
>>> t = Torrent.one('/search/ubuntu/seeds')
>>> t.name
'Ubuntu 7.10 Desktop Live CD'
>>> t.size
u'695.81\xa0MB'
>>> t.url
'/get/1053846'
>>> t.html
u'<td>19\xa0Dec\xa007</td><td><a href="/cat/7">Software</a></td><td>...'

>>> results = Torrent.all('/search/ubuntu/seeds')
>>> len(results)
116
>>> for t in results[:3]:
...     print t.name, t.size
...
Ubuntu 7.10 Desktop Live CD 695.81 MB
Super Ubuntu 2008.09 - VMware image 871.95 MB
Portable Ubuntu 9.10 for Windows 559.78 MB
...
```

Any extra attributes defined in the *Item.Meta* class will be passed to PyQuery when doing the URL request (i.e. you could add, for example, *encoding* or *method*; if python-requests is available, there is a bunch of extra parameters you could use, such as: *auth*, *data*, *headers*, *verify*, *cert*, *config*, *hooks*, *proxies*).

Alternatively, there is an *all_from* method that will retrieve all items from a PyQuery object created from the given arguments (i.e. it will directly pass all specified parameters to PyQuery and scrap items from there).

3.1 Related items

New in version 0.2.

You can also define a RelatedItem. A RelatedItem is a different *Item* subclass related to the item it is defined in, being an element subitem (for example, a row could be a subitem of a table) or another item that could be found following a link in the main item (for example, the link to the details page of a search result entry):

```
class TorrentDetails(demiurge.Item):
    label = demiurge.TextField(selector='strong')
    value = demiurge.TextField()

    def clean_value(self, value):
        unlabel = value[value.find(':') + 1:]
        return unlabel.strip()

    class Meta:
        selector = 'div#specifications p'

class Torrent(demiurge.Item):
    url = demiurge.AttributeValueField(
        selector='td:eq(2) a:eq(1)', attr='href')
    name = demiurge.TextField(selector='td:eq(2) a:eq(2)')
    size = demiurge.TextField(selector='td:eq(3)')
    details = demiurge.RelatedItem(
        TorrentDetails, selector='td:eq(2) a:eq(2)', attr='href')

    class Meta:
```

```
        selector = 'table.maintable:gt(0) tr:gt(0)'
        base_url = 'http://www.mininova.org'
```

In the example above, for each torrent result you have a details attribute that will be evaluated when required, following the URL given by the 'href' attribute and getting the related TorrentDetails items from that other page:

```
>>> t = Torrent.one('/search/ubuntu/seeds')
>>> for detail in t.details:
...     print detail.label, detail.value
...
Category: Software > GNU/Linux
Total size: 695.81 megabyte
Added: 2467 days ago by Distribution
Share ratio: 17 seeds, 2 leechers
Last updated: 35 minutes ago
Downloads: 29,085
```

A RelatedItem returns a list of all the matching items. If you need to self-relate an Item with itself, you should use the 'self' parameter:

```
class SearchResults(demiurge.Item):
    ...
    next_page = demiurge.RelatedItem('self', selector='...', attr='...')
```

CHAPTER 4

Why *demiurge*?

Plato, as the speaker Timaeus, refers to the Demiurge frequently in the Socratic dialogue Timaeus, c. 360 BC. The main character refers to the Demiurge as the entity who "fashioned and shaped" the material world. Timaeus describes the Demiurge as unreservedly benevolent, and hence desirous of a world as good as possible. The world remains imperfect, however, because the Demiurge created the world out of a chaotic, indeterminate non-being.

http://en.wikipedia.org/wiki/Demiurge