unshortenit Documentation

Release

Jeff Kehler

Apr 01, 2018

Contents

1 Quickstart **3**
1.1 Installation . . . 3
1.2 Usage . . . 3
1.3 Advanced Usage . . . 3
1.4 Exceptions . . . 4

2 Extending UnshortenIt **5**
2.1 Adding Additional Domains to Existing modules . . . 5

Unshortenit is a Python module that allows you to easily unshorten short urls and extract urls hidden behind ad-based shorteners.

This module currently supports the following:

- Any 301 redirected link.
- Meta-Refresh redirected links.
- Adf.ly
- Adfoc.us
- Sh.st

CHAPTER 1

Quickstart

1.1 Installation

This package is available via the PIP repository. To install this package simply run this commmand:

```
$ pip install unshortenit
```

1.2 Usage

Using this library is very simple:

```
>>> from unshortenit import UnshortenIt
>>> unshortener = UnshortenIt()
>>> uri = unshortener.unshorten('https://href.li/?https://example.com')
>>> print(uri)
https://example.com
```

1.3 Advanced Usage

You can override the default timeout value and default headers globally for all modules via the UnshortenIt initializer:

```
>>> from unshortenit import UnshortenIt
>>> unshortener = UnshortenIt(default_timeout=30, default_headers=dict())
```

By default the library will not make any HTTP request if the url provided does not match any of the modules url patterns. To override this you may pass `force=True` to the `unshorten` method.

If you wish to force a specific module to be used for a link you may do so by using the `module='module_name'` parameter on the `unshorten` method. This is useful for ad-based shorteners that allow people to use custom domains or if you know a link is a meta refresh type.

In some cases you may come across shorteners that are nested one after another. By default the library will not check if the returned link is also another shortener link. If you wish to force it to always check the returned link as well you may pass `unshorten_nested=True` to the `unshorten` method.

1.4 Exceptions

Exceptions will not be handled by the module and will be passed up to the caller. You will need to wrap the unshorten call in your own try, except blocks.

This module includes 2 custom Exceptions `unshortenit.exceptions.NotFound` and `unshortenit.exceptions.UnshortenFailed`. The first will be triggered if the url returns a 404 status code. The later will occur if some sort of issue happened during the unshortening process. All other exceptions will be the default python requests library exceptions.

CHAPTER 2

Extending UnshortenIt

This library supports extensions by creating your own unshorter modules and registering them. This is useful for any custom sites you may need to handle.

To create your own module you have to subclass `unshortenit.module.UnshortenModule` and implement the `unshorte`

```
>>> from unshortenit.module import UnshortenModule
>>> class CustomModule(UnshortenModule):
...     name = 'my-custom-module'
...     domains = [
...         'example.com'
...     ]
...
...     def __init__(self, headers=None, timeout=30):
...         super().__init__(headers, timeout)
...
...     def unshorten(self, uri):
...         """ Implement custom unshorten logic here and return result """
...         return uri
...
>>> from unshortenit import UnshortenIt()
>>> unshortener = UnshortenIt()
>>> unshortener.register_module(CustomModule)
```

2.1 Adding Additional Domains to Existing modules

You may extend one of the existing modules by adding additional domains to the domain list:

```
>>> from unshortenit import UnshortenIt
>>> unshortener = UnshortenIt()
>>> unshortener.modules['meta-refresh'].add_domain('example.com')
```

Now whenever you pass a url from the example.com domain it will be parsed by the meta refresh module.