WebApp2 RequestParser Documentation

Release 0.1.0

Eran Kampf

February 02, 2016

Contents

1 WebApp2 RequestParser **3**
- 1.1 Basic Argument Parsing . . . 3
- 1.2 Special Google AppEngine Arguments . . . 3

2 Installation **5**

3 Usage **7**

4 Contributing **9**
- 4.1 Types of Contributions . . . 9
- 4.2 Get Started! . . . 10
- 4.3 Pull Request Guidelines . . . 10
- 4.4 Tips . . . 11

5 Credits **13**
- 5.1 Development Lead . . . 13
- 5.2 Contributors . . . 13

6 History **15**

7 0.1.0 (2015-01-11) **17**

8 Indices and tables **19**

Contents:

CHAPTER 1

WebApp2 RequestParser

The *webapp2_requestparser* library is a Request parsing interface inspired by restful-flask's request parser.

Its interface is modelled after the argparse interface.

Its goal is to provide a uniform access to any variable on the webapp2.Request object and allowing handlers to provide a sort of "contract" where they specify the parameters they expect to be called with - making code easier to read and understand.

- Free software: BSD license
- Documentation: https://webapp2_requestparser.readthedocs.org.

<TBD - Documentation is still partial but mostly follows the same API the Flask library provide with a few additions>

1.1 Basic Argument Parsing

Here's a simple example of the request parser. It looks for two arguments in the webapp2.Request's *json* and *params* properties: one of type int, and the other of type str:

```
from webapp2_requestparser.parser import RequestParser

parser = RequestParser()
parser.add_argument('rate', type=int, help='Rate cannot be converted')
parser.add_argument('name', type=str)
args = parser.parse_args(self.request)
```

1.2 Special Google AppEngine Arguments

```
from webapp2_requestparser.parser import RequestParser
from webapp2_requestparser.arguments_ndb import EntityIDArgument

parser = RequestParser()
parser.add_argument('store_id', type=EntityIDArgument(Store), dest='store')
args = parser.parse_args(self.request)

# args.store is a Store instance
print(args.store)
```

CHAPTER 2

Installation

At the command line:

```
$ easy_install webapp2_requestparser
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv webapp2_requestparser
$ pip install webapp2_requestparser
```

CHAPTER 3

Usage

To use WebApp2 RequestParser in a project:

```
import webapp2_requestparser
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/ekampf/webapp2_requestparser/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

WebApp2 RequestParser could always use more documentation, whether as part of the official WebApp2 RequestParser docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/ekampf/webapp2_requestparser/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *webapp2_requestparser* for local development.

1. Fork the *webapp2_requestparser* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/webapp2_requestparser.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv webapp2_requestparser
$ cd webapp2_requestparser/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 webapp2_requestparser tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, and 3.4, and for PyPy. Check https://travis-ci.org/ekampf/webapp2_requestparser/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_webapp2_requestparser
```

CHAPTER 5

Credits

5.1 Development Lead

- Eran Kampf - http://www.developerzen.com

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

CHAPTER 7

0.1.0 (2015-01-11)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search