sprockets.mixins.cors

Release 0.1

July 30, 2015

Contents

This library exposes a mix-in that adds some useful CORS hooks over a basic `tornado.web.RequestHandler`.

CHAPTER 1

Examples

```
#!/usr/bin/env python

import logging

from tornado import ioloop, web

from sprockets.mixins import cors


class SimpleRequestHandler(cors.CORSMixin, web.RequestHandler):
    """Very simple request handler that CORS enables the GET endpoint."""

    def initialize(self, creds=False, req_headers=None):
        super(SimpleRequestHandler, self).initialize()
        self.cors.allowed_methods.add('GET')
        self.cors.credentials_supported = creds
        if req_headers:
            self.cors.request_headers.update(hdr.lower() for hdr in req_headers)

    def prepare(self):
        # This is used to test that the mixin does not interfere
        # with request failures.  You really shouldn't call super()
        # after you explicitly finish() but anyway...
        if 'X-Fail' in self.request.headers:
            self.set_status(400)
            self.finish()

        super(SimpleRequestHandler, self).prepare()

    def get(self):
        self.set_status(204)
        self.finish()


if __name__ == '__main__':
    logging.basicConfig(
        level=logging.DEBUG,
        format='%(levelname)-8s %(name)s: %(message)s')
    app = web.Application([
        web.url('/public', SimpleRequestHandler),
        web.url('/private', SimpleRequestHandler, {'creds': True}),
    ], cors_origins=['http://www.example.com'], debug=True)
    app.listen(8000)
```

```
    iol = ioloop.IOLoop.instance()
    try:
        iol.start()
    except KeyboardInterrupt:
        logging.info('stopping IOLoop')
        iol.add_callback(iol.stop)
```

CHAPTER 2

License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Sprockets nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

2.1 API Documentation

Tornado RequestHandler mix-in for implementing a CORS enabled endpoint.

The CORS specification describes a method of securing javascript access to web resources across access domains. This module implements a mix-in to be used with `tornado.web.RequestHandler` that provides much of the functionality required by CORS.

class `sprockets.mixins.cors.`**CORSMixin**

Mix this in over a `tornado.web.RequestHandler` for CORS support.

cors

A *`CORSSettings`* instance that controls the behavior of the mix-in.

options()

Respond to an OPTIONS request.

This method relies on `self.SUPPORTED_METHODS` for the content of the Allow response header. The CORS specific headers are generated based on the *cors* attribute.

class sprockets.mixins.cors.**CORSSettings**

Configures the CORS behavior.

allowed_methods

The set of CORS accepted HTTP methods. This controls the *Access-Control-Allow-Methods* response header.

allowed_origins

The set of origins that are allowed for the endpoint. This controls the *Access-Control-Allow-Origin* response header. If the requested origin is in this set, then the origin is allowed; otherwise, a 403 Forbidden is returned.

credentials_supported

Should the mix-in generate the *Access-Control-Allow-Credentials* header in the response.

request_headers

A set of header names that are acceptable in cross-origin requests. Headers added to this set **MUST** be lower-cased before adding them to the set.

2.2 How to Contribute

Do you want to contribute fixes or improvements?

AWesome! *Thank you very much, and let's get started.*

2.2.1 Set up a development environment

The first thing that you need is a development environment so that you can run the test suite, update the documentation, and everything else that is involved in contributing. The easiest way to do that is to create a virtual environment for your endevours:

```
$ virtualenv -p python2.7 env
```

Don't worry about writing code against previous versions of Python unless you you don't have a choice. That is why we run our tests through tox. If you don't have a choice, then install virtualenv to create the environment instead. The next step is to install the development tools that this project uses. These are listed in *requires/development.txt*:

```
$ env/bin/pip install -qr requires/development.txt
```

At this point, you will have everything that you need to develop at your disposal. *setup.py* is the swiss-army knife in your development tool chest. It provides the following commands:

./setup.py nosetests Run the test suite using nose and generate a nice coverage report.

./setup.py build_sphinx Generate the documentation using sphinx.

./setup.py flake8 Run flake8 over the code and report style violations.

If any of the preceding commands give you problems, then you will have to fix them **before** your pull request will be accepted.

2.2.2 Running Tests

The easiest (and quickest) way to run the test suite is to use the *nosetests* command. It will run the test suite against the currently installed python version and report not only the test result but the test coverage as well:

```
$ ./setup.py nosetests

running nosetests
running egg_info
writing dependency_links to sprockets.mixins.cors.egg-info/dependency_links.txt
writing top-level names to sprockets.mixins.cors.egg-info/top_level.txt
writing sprockets.mixins.cors.egg-info/PKG-INFO
reading manifest file 'sprockets.mixins.cors.egg-info/SOURCES.txt'
reading manifest template 'MANIFEST.in'
warning: no previously-included files matching '__pycache__'...
warning: no previously-included files matching '*.swp' found ...
writing manifest file 'sprockets.mixins.cors.egg-info/SOURCES.txt'
...

Name                        Stmts   Miss Branch BrMiss  Cover   Missing
-----------------------------------------------------------------------
...
-----------------------------------------------------------------------
TOTAL                          95      2     59      2    97%
-----------------------------------------------------------------------
Ran 44 tests in 0.054s

OK
```

That's the quick way to run tests. The slightly longer way is to run the detox utility. It will run the test suite against all of the supported python versions in parallel. This is essentially what Travis-CI will do when you issue a pull request anyway:

```
$ env/bin/detox
py27 recreate: /.../sprockets.mixins.cors/build/tox/py27
GLOB sdist-make: /.../sprockets.mixins.cors/setup.py
py34 recreate: /.../sprockets.mixins.cors/build/tox/py34
py27 installdeps: -rtest-requirements.txt, mock
py34 installdeps: -rtest-requirements.txt
py27 inst: /.../sprockets.mixins.cors/build/tox/dist/sprockets.mixins.cors-0.0.0.zip
py27 runtests: PYTHONHASHSEED='2156646470'
py27 runtests: commands[0] | /../sprockets.mixins.cors/build/tox/py27/bin/nosetests
py34 inst: /../sprockets.mixins.cors/.build/tox/dist/sprockets.mixins.cors-0.0.0.zip
py34 runtests: PYTHONHASHSEED='2156646470'
py34 runtests: commands[0] | /.../sprockets.mixins.cors/build/tox/py34/bin/nosetests
_____________________________________ summary _____________________________________
  py27: commands succeeded
  py34: commands succeeded
  congratulations :)
```

This is what you want to see. Now you can make your modifications and keep the tests passing.

2.2.3 Submitting a Pull Request

Once you have made your modifications, gotten all of the tests to pass, and added any necessary documentation, it is time to contribute back for posterity. You've probably already cloned this repository and created a new branch. If you haven't, then checkout what you have as a branch and roll back *master* to where you found it. Then push your

repository up to github and issue a pull request. Describe your changes in the request, if Travis isn't too annoyed someone will review it, and eventually merge it back.

2.3 Version History

2.3.1 0.1.1

- minor packaging changes

2.3.2 0.1.0

- initial implementation including:
  - *sprockets.mixins.cors.CORSMixin*
  - *sprockets.mixins.cors.CORSSettings*
  - Support for the following pre-flight request headers:
    - *Origin*
    - *Access-Control-Request-Method*
    - *Access-Control-Request-Headers*
  - Support for the following pre-flight response headers:
    - *Access-Control-Allow-Origin*
    - *Access-Control-Allow-Methods*
    - *Access-Control-Allow-Credentials*
    - *Access-Control-Allow-Headers*
  - Support for the following in-line response headers:
    - *Access-Control-Allow-Origin*
    - *Access-Control-Allow-Credentials*

S

A

C

O

R

S