rainfall Documentation

Release 0.8.3

Anton Kasyanov

January 08, 2014

Contents

1 Quickstart **3**
- 1.1 Installation 3
- 1.2 Hello world 3

2 Features **5**
- 2.1 Coroutines 5
- 2.2 Template rendering 5
- 2.3 Url params 6
- 2.4 GET and POST params 6
- 2.5 Logging 6
- 2.6 Testing 6
- 2.7 ETag 7

3 API Reference **9**
- 3.1 rainfall.web 9
- 3.2 rainfall.http 10
- 3.3 rainfall.unittest 11

4 Indices and tables **13**

Python Module Index **15**

This is a micro asyncio web framework, a bit similiar to cyclone or tornado.

Contents:

CHAPTER 1

Quickstart

To start off, rainfall is a micro web framework around asyncio (ex tulip), similiar to the cyclone or tornado. Since it is asyncio based, rainfall is fully asyncronous.

The performance tests have shown that rainfall is not slower then twisted+cyclone and sometimes even faster (benchmark results will be posted later or you can test it by yourself).

1.1 Installation

As simple as:

```
pip install rainfall
```

Note: sometimes pip for python 3 is called pip3, but you may have it with other name

1.2 Hello world

Let's create a simple hello world app in example.py file like this:

```
import asyncio
from rainfall.web import Application, HTTPHandler


class HelloHandler(HTTPHandler):
    @asyncio.coroutine
    def handle(self, request):
        return 'Hello!'


app = Application(
    {
        r'^/$': HelloHandler(),
    },
)

if __name__ == '__main__':
    app.run()
```

Now you can run it by:

```
python3 example.py
```

And go to http://127.0.0.1:8888 in browser, you should see “Hello!”

The structure is the following:

1. First, you create `rainfall.web.HTTPHandler` with required handle() method. It takes `rainfall.http.HTTPRequest` and should return a str or `rainfall.http.HTTPError`
2. Second, you create `rainfall.web.Application`. When a new application is created, you should pass a dict of url: `rainfall.web.HTTPHandler` pairs, where url is regexp telling rainfall when to use this particular handler. If you have experience with Django, this works like django’s url patterns.

1.2.1 Testing

To test the rainfall apps you can use `rainfall.unittest.RainfallTestCase`

For more, see *Features*

CHAPTER 2

Features

Rainfall comes with a list of features, more are in development. If you feel that rainfall is missing a feature, please let me know.

2.1 Coroutines

Rainfall's `rainfall.web.HTTPHandler.handle()` may be a regular function or *asyncio.coroutine* and use all the asynchronous features like *yield from*.

Example:

```
class SleepHandler(HTTPHandler):
    @asyncio.coroutine
    def handle(self, request):
        yield from asyncio.sleep(0.1)
        return 'Done'
```

2.2 Template rendering

Rainfall uses Jinja2 if you need to render a template.

Example:

```
class TemplateHandler(HTTPHandler):
    def handle(self, request):
        return self.render('base.html', text='Rendered')


settings = {
    'template_path': os.path.join(os.path.dirname(__file__), "templates"),
}

app = Application(
    {
        r'^/template$': TemplateHandler(),
    },
    settings=settings,
)

app.run()
```

2.3 Url params

You can easily handle urls with params inside

Example:

```
class ParamHandler(HTTPHandler):
    def handle(self, request, number):
        return number

app = Application(
    {
        r'^/param/(?P<number>\d+)$': ParamHandler(),

    },
)
app.run()
```

2.4 GET and POST params

Using `rainfall.http.HTTPRequest.GET` and `rainfall.http.HTTPRequest.POST` you can easily handle forms data.

2.5 Logging

Rainfall uses standart python *logging* module. To configure the file for logs, use *logfile_path* in Application settings.

2.6 Testing

To test the rainfall apps you can use `rainfall.unittest.RainfallTestCase`

example.py:

```
import asyncio
from rainfall.web import Application, HTTPHandler


class HelloHandler(HTTPHandler):
    def handle(self, request):
        return 'Hello!'


app = Application(
    {
        r'^/$': HelloHandler(),
    },
)

# this is important for tests
if __name__ == '__main__':
    app.run()
```

test_basic.py:

```
from rainfall.unittest import RainfallTestCase

from example import app

class HTTPTestCase(RainfallTestCase):
    app = app

    def test_basic(self):
        r = self.client.query('/')
        self.assertEqual(r.status, 200)
        self.assertEqual(r.body, 'Hello!')
```

2.7 ETag

`rainfall.web.HTTPHandler` allows to use ETag for cache validation.

Example:

```
class EtagHandler(HTTPHandler):

    use_etag = True
    payload = "PowerOfYourHeart"

    def handle(self, request):
        return self.payload
```

Then we test it this way:

```
def test_etag_wo_ifnonematch(self):
    etag_awaiting = '"' + hashlib.sha1(EtagHandler.payload.encode('utf-8')).hexdigest() + '"'
    r = self.client.query(
        '/etag', method='GET'
    )
    self.assertEqual(r.status, 200)
    self.assertEqual(etag_awaiting, r.headers.get('ETag'))

def test_etag_with_ifnonematch(self):
    etag_awaiting = '"' + hashlib.sha1(EtagHandler.payload.encode('utf-8')).hexdigest() + '"'
    r = self.client.query(
        '/etag', method='GET',
        headers={
            "If-None-Match": etag_awaiting
        }
    )
    self.assertEqual(r.status, 304)
    self.assertEqual(r.body, '')
    self.assertEqual(etag_awaiting, r.headers.get('ETag'))
```

API Reference

This docs are generated from the docstirngs. Apart from that, feel free to study source code directly.

3.1 rainfall.web

class `rainfall.web.`**`Application`**(*handlers*, *settings=None*)

The core class that is used to create and start server

Parameters

- **handlers** – dict with url keys and HTTPHandler instance values
- **settings** – dict of app settings, defaults are settings = {

  'host': '127.0.0.1', 'port': 8888, 'logfile_path': None, 'template_path': None,

  }

Example:

```
app = Application({
    '/': HelloHandler(),
})
app.run()
```

`run`(*process_queue=None*, *greeting=True*)

Starts server on host and port given in settings, adds Ctrl-C signal handler.

Parameters

- **process_queue** – SimpleQueue, used by testing framework
- **greeting** – bool, wheather to print to strout or not

class `rainfall.web.`**`HTTPHandler`**

Used by HTTPServer to react for some url pattern.

All handling happens in handle method.

`handle`(*request*, ***kwargs*)

May be an asyncio.coroutine or a regular function

Parameters

- **request** – `rainfall.http.HTTPRequest`
- **kwargs** – arguments from url if any

Return type str (may be rendered with self.render()) or `rainfall.http.HTTPError`

`render` (*template_name*, ***kwargs*)

Uses jinja2 to render a template

Parameters

- **template_name** – what file to render
- **kwargs** – arguments to pass to jinja's render

Return type rendered string

class `rainfall.web.`**`HTTPServer`**

Http server itself, uses asyncio.Protocol. Not meant to be created manually, but by *rainfall.web.Application* class.

3.2 rainfall.http

exception `rainfall.http.`**`HTTPError`** (*code=500*, **args*, ***kwargs*)

Representes different http errors that you can return in handlers.

Parameters **code** – http error code

class `rainfall.http.`**`HTTPRequest`** (*raw*)

Rainfall implementation of the http request.

Parameters **raw** – raw text of full http request

`GET`

Return type dict, GET arguments

`POST`

Return type dict, POST arguments

`body`

Return type str, http body

`headers`

Return type dict, http headers

`method`

Return type str, http method

`path`

Return type str, http url

class `rainfall.http.`**`HTTPResponse`** (*body=''*, *code=200*)

Rainfall implementation of the http response.

Parameters

- **body** – response body
- **code** – response code
- **additional_headers** –

`compose` ()

Composes http response from code, headers and body

Return type str, composed http response

3.3 rainfall.unittest

class `rainfall.unittest.`**`RainfallTestCase`**(*methodName='runTest'*)

Use it for your rainfall test cases. In setUp rainfall server is starting in the separate Process.

You are required to specify an app variable for tests. E.g.

```
from example import my_first_app


class HTTPTestCase(RainfallTestCase):
    app = my_first_app

    def test_basic(self):
        r = self.client.query('/')
        self.assertEqual(r.status, 200)
        self.assertEqual(r.body, 'Hello!')
```

Inside you can use TestClient's instance via self.client

class `rainfall.unittest.`**`TestClient`**(*host*, *port*)

Helper to make request to the rainfall app. Created automatically by RainfallTestCase.

`query`(*url*, *method='GET'*, *params=None*, *headers={}*)

Run a query using url and method. Returns response object with status, reason, body

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

r

`rainfall.http`, 10
`rainfall.unittest`, 11
`rainfall.web`, 9