Flask-aiohttp Documentation

Release 0.1.0

Geonu Choi

Jul 24, 2017

Contents

Flask-aiohttp adds asyncio & websocket[1] support to Flask using aiohttp.

For example

```
@app.route('/use-external-api')
@async
def use_external_api():
    response = yield from aiohttp.request(
        'GET', 'https://api.example.com/data/1')
    data = yield from response.read()

    return data
```

You can find more guides from following list:

[1] http://aiohttp.readthedocs.org/en/v0.15.1/web.html#websockets

CHAPTER 1

First of All

First of all, you can use this extension like another plain Flask extensions.

For example, You can initialize the extension like

```
from flask import Flask
from flask.ext.aiohttp import AioHTTP

app = Flask(__name__)

aio = AioHTTP(app)
```

or you can initialize it later

```
from flask import Flask
from flask.ext.aiohttp import AioHTTP

def create_app():
    app = Flask(__name__)
    aio.init_app(app)

aio = AioHTTP()
```

But, its application running method is different then plain Flask apps one. You have to run it on the asyncio's run loop.

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

You can also debug it using werkzeug debugger

```
if __name__ == '__main__':
    aio.run(app, debug=True)
```

You can use gunicorn using aiohttp

In myapp.py (or some module name you want to use)

```
from you_application_module import app as flask_app

app = flask_app.aiohttp_app
```

And run gunicorn by

```
gunicorn myapp:app -k aiohttp.worker.GunicornWebWorker -b localhost:8080
```

CHAPTER 2

Coroutine

You can use asyncio's coroutine[1] in flask's view function using this extension.

```
from flask.ext.aiohttp import async

@app.route('/late-response')
@async  # It marks view function as asyncio coroutine
def late_response():
    yield from asyncio.sleep(3)
    return "Sorry, I'm late!"
```

So, you can use aiohttp's request modules in flask.

```
from flask.ext.aiohttp import async

@app.route('/zuck')
@async
def zuck():
    response = yield from aiohttp.request(
        'GET', 'https://graph.facebook.com/zuck')
    data = yield from response.read()

    return data
```

And you can surely use flask's common feature.

```
import json
import urllib.parse

from flask import current_app, request
from flask.ext.aiohttp import async

@app.route('/fb/<name>')
@async
```

[1] https://docs.python.org/3/library/asyncio-task.html#coroutines

```python
def facebook_profile(name):
    if request.args.get('secure', False):
        url = 'https://graph.facebook.com/'
    else:
        url = 'http://graph.facebook.com/'
    url = url + urllib.parse.quote(name)
    response = yield from aiohttp.request('GET', url)
    data = yield from response.read()
    data = json.loads(data)

    def stream():
        if request.args.get('wrap', False):
            data = {
                'data': data
            }
        yield json.dumps(data)
    return current_app.response_class(stream())
```

Note: Since coroutine implemented by using streaming response, you have to be care about using request hook.

`before_request()`, `after_request()`, `teardown_request()` will be called twice.

Each asynchronous request's functions will be called in following sequence.

1. `before_request()`
2. **Flask-aiohttp's** streaming response containing coroutine
3. `after_request()`
4. `teardown_request()`

Streaming response starts here

5. `before_request()`
6. Your coroutine response
7. `after_request()`
8. `teardown_request()`

CHAPTER 3

WebSocket

Flask-aiohttp injects `WebSocketResponse` into your WSGI environ, and provides api to use it.

This is not elegant solution for using websocket. But, it would be best solution for now.

```
from flask.ext.aiohttp import websocket


@app.route('/echo')
@websocket
def echo():
    while True:
        msg = yield from aio.ws.receive_msg()

        if msg.tp == aiohttp.MsgType.text:
            aio.ws.send_str(msg.data)
        elif msg.tp == aiohttp.MsgType.close:
            print('websocket connection closed')
            break
        elif msg.tp == aiohttp.MsgType.error:
            print('ws connection closed with exception %s',
                  aio.ws.exception())
            break
```

You also can use most features of flask with websocket.

```
from flask.ext.aiohttp import websocket


@app.route('/hello/<name>')
@websocket
def hello(name):
    while True:
        msg = yield from aio.ws.receive_msg()

        if msg.tp == aiohttp.MsgType.text:
            aio.ws.send_str('Hello, {}'.format(name))
        elif msg.tp == aiohttp.MsgType.close:
            print('websocket connection closed')
            break
        elif msg.tp == aiohttp.MsgType.error:
            print('ws connection closed with exception %s',
                  aio.ws.exception())
            break
```

API

CHAPTER 4

flask_aiohttp

flask_aiohttp package

Submodules

flask_aiohttp.handler module

flask_aiohttp.helper module

flask_aiohttp.util module

Module contents

CHAPTER 5

Indices and tables

- genindex
- modindex
- search