Aiohttp Web Framework - Documentation

Asynchronous HTTP Client/Server for asyncio and Python.

What is Aiohttp Framework ?

Aiohttp Framework is a popular asynchronous framework based on the asyncio library, which has been around since the first days of the library.

Like Flask, it provides a request object and a router to redirect queries to functions that handle them

The asyncio library's event loop is wrapped into an Application object, which handles most of the orchestration work. As a microservice developer, you can just focus on building your views as you would do with Flask.

Why to use Aiohttp Framework?

- Supports both Client and HTTP Server.
- Supports both Server WebSockets and Client WebSockets out-of-the-box without the Callback Hell.
- Web-server has Middlewares, Signals and pluggable routing.

How Aiohttp is Different?

Unicode support *aiohttp* does requoting of incoming request path.Unicode (non-ASCII) symbols are processed transparently on both *route adding* and *resolving* (internally everything is converted to percent-encoding form by yarl library).But in case of custom regular expressions for Variable Resources please take care that URL is *percent encoded*: if you pass Unicode patterns they don't match to *requoted* path.

- Web Handler Cancellation
- Warning

web-handler execution could be canceled on every await if client drops connection without reading entire response's BODY.
The behavior is very different from classic WSGI frameworks like Flask and Django.
Sometimes it is a desirable behavior: on processing GET request the code might fetch data from database or other web resource, the fetching is potentially slow.
Canceling this fetch is very good: the peer dropped connection already, there is no reason to waste time and resources (memory etc) by getting data from DB without any chance to send it back to peer.But sometimes the cancellation is bad:
on POST request very often is needed to save data to DB regardless to peer closing .
Cancellation prevention could be implemented in several ways:

- Applying **asyncio.shield()** to coroutine that saves data into DB.

Spawning a new task for DB saving.

- Using aiojobs or other third party library.

asyncio.shield() works pretty good. The only disadvantage is you need to split web handler into exactly two async functions: one for handler itself and other for protected code.

Installation

Library Installation

$ pip install aiohttp

You may want to install *optional* cchardet library as faster replacement for chardet:

$ pip install cchardet

For speeding up DNS resolving by client API you may install aiodns as well. This option is highly recommended:

$ pip install aiodns

Installing speedups altogether

The following will get you aiohttp along with chardet, aiodns and brotlipy in one bundle. No need to type separate commands anymore!

$ pip install aiohttp[speedups]

Example:

```
# filename: app.py

from aiohttp import web
import json

async def handle(request):
   response_obj = { 'status' : 'success' }
   return web.Response(text=json.dumps(response_obj))

app = web.Application()
app.router.add_get('/', handle)

web.run_app(app)
```

Testing our API

We can then run our new REST API by calling python app.py which should start our app on http://0.0.0.0 a.k.a http://localhost on port 8080 by default.

When you navigate to http://localhost:8080 you should see our {"status": "success"} being returned in the browser.

POST Requests and Query Parameters

Now that we've successfully defined a very basic, single endpoint API we can now start to build on top of this and start exposing different routes that use different verbs. Let's create a simple POST request endpoint that takes in name via a query parameter. We'll want the final URL of this endpoint to look like **so: http://localhost:8080/user?name=elliot**.

Let's define the handler function new_user(request) now.

```
async def new_user(request):
   try:
      # happy path where name is set
```

```
        user = request.query['name']

        # Process our new user

        print("Creating new user with name: " , user)

        response_obj = { 'status' : 'success' }

        # return a success json response with status code 200 i.e. 'OK'

        return web.Response(text=json.dumps(response_obj), status=200)

    except Exception as e:

        # Bad path where name is not set

        response_obj = { 'status' : 'failed', 'reason': str(e) }

        # return failed with a status code of 500 i.e. 'Server Error'

        return web.Response(text=json.dumps(response_obj), status=500)
```

Once we have successfully defined this new handler function we will have to register it in our routes like so:

```
app.router.add_post('/user', new_user)
```

Try run your application now and send a POST request to http://localhost:8080/user?name=test and you should see the following output in the console:

Dependencies

- Python 3.5.3+
- *async_timeout*
- *attrs*
- *chardet*
- *multidict*
- *yarl*
- *Optional* cchardet as faster replacement for chardet.

Pros:

- The major **advantage of asyncio** approach vs green-threads approach is that with **asyncio** we have cooperative multitasking and places in code where some task can yield control are clearly indicated by await , async with and async for . It makes it easier to reason about common concurrency problem of data races
- Asynchronous programming is well suited for tasks that include reading and writing files frequently or sending data back and forth from a server.
- Asynchronous programs perform I/O operations in a non-blocking fashion, meaning that they can perform other tasks while waiting for data to return from a client rather than waiting idly, wasting resources and time.

Cons:

- no out-of-the-box template engine but Jinja2 or Mako could be applied
- no WSGI support,
- supports routing for static files

AIOHTTP Alternatives & Comparisons

AIOHTTP vs GraphQL: What are the differences?

Developers describe **AIOHTTP** as "*Asynchronous HTTP Client/Server for asyncio and Python*". It is an Async http client/server framework. It supports both client and server Web-Sockets out-of-the-box and avoids Callback It provides Web-server with middlewares and pluggable routing..

On the other hand, **GraphQL** is detailed as "*A data query language and runtime*". GraphQL is a data query language and runtime designed and used at Facebook to request and deliver data to mobile and web apps since 2012.

AIOHTTP and GraphQL are primarily classified as **"Microframeworks (Backend)"** and **"Query Languages"** tools respectively.

Some of the features offered by AIOHTTP are:

- asyncio
- client
- server

On the other hand, GraphQL provides the following key features:

- Hierarchical
- Product-centric

AIOHTTP vs ExpressJS: What are the differences?

Developers describe **AIOHTTP** as "*Asynchronous HTTP Client/Server for asyncio and Python*". It is an Async http client/server framework. It supports both client and server Web-Sockets out-of-the-box and avoids Callback It provides Web-server with middlewares and pluggable routing.. On the other hand, **ExpressJS** is detailed as "*Sinatra inspired web development framework for node.js -- insanely fast, flexible, and simple*". Express is a minimal and flexible node.js web application framework, providing a robust set of features for building single and multi-page, and hybrid web applications.

AIOHTTP and ExpressJS belong to **"Microframeworks (Backend)"** category of the tech stack.

Some of the features offered by AIOHTTP are:

- asyncio
- client
- server

On the other hand, ExpressJS provides the following key features:

- Robust routing
- HTTP helpers (redirection, caching, etc)
- View system supporting 14+ template engines

ExpressJS is an open source tool with **45K** GitHub stars and **7.55K** GitHub forks.

Twitter, **Intuit**, and **OpenGov** are some of the popular companies that use ExpressJS, whereas

AIOHTTP is used by **Uploadcare**, **Hotjar**, and **Hivestack**. ExpressJS has a brosader approval, being mentioned in **1202** company stacks & **4118** developers stacks; compared to AIOHTTP, which is listed in **10** company stacks and **12** developer stacks

References:

1. https://docs.aiohttp.org/en/stable
2. https://medium.com/radix-ai-blog/performant-http-with-aiohttp-in-python-3-756580e54eff
3. https://stackshare.io/aiohttp