aiohttp$_{utils}$

Release 3.2.1

Feb 14, 2023

Contents

Release v3.2.1. (*Changelog*)

aiohttp_utils provides handy utilities for building aiohttp.web applications.

- Method-based handlers ("resources")
- Routing utilities
- Content negotiation with JSON rendering by default

Everything is optional. You can use as much (or as little) of this toolkit as you need.

```
from aiohttp import web
from aiohttp_utils import Response, routing, negotiation

app = web.Application(router=routing.ResourceRouter())

# Method-based handlers
class HelloResource:

    async def get(self, request):
        name = request.GET.get('name', 'World')
        return Response({
            'message': 'Hello ' + name
        })


app.router.add_resource_object('/', HelloResource())

# Content negotiation
negotiation.setup(
    app, renderers={
        'application/json': negotiation.render_json
    }
)
```

CHAPTER 1

Install

```
$ pip install aiohttp_utils
```

Ready to get started? Go on to one of the the usage guides below or check out some examples.

CHAPTER 2

Guides

Below are usage guides for each of the modules.

2.1 `negotiation` - Content negotiation

Content negotiation is the process of selecting an appropriate representation (e.g. JSON, HTML, etc.) to return to a client based on the client's and/or server's preferences.

If no custom renderers are supplied, this plugin will render responses to JSON.

```
import asyncio

from aiohttp import web
from aiohttp_utils import negotiation, Response

async def handler(request):
    return Response({'message': "Let's negotiate"})

app = web.Application()
app.router.add_route('GET', '/', handler)

# No configuration: renders to JSON by default
negotiation.setup(app)
```

We can consume the app with httpie.

```
$ pip install httpie
$ http :5000/
HTTP/1.1 200 OK
CONNECTION: keep-alive
CONTENT-LENGTH: 30
CONTENT-TYPE: application/json
DATE: Thu, 22 Oct 2015 06:03:39 GMT
```

(continues on next page)

(continued from previous page)

```
SERVER: Python/3.5 aiohttp/0.17.4

{
    "message": "Let's negotiate"
}
```

Note: Handlers **MUST** return an *aiohttp_utils.negotiation.Response* (which can be imported from the top-level `aiohttp_utils` module) for data to be properly negotiated. *aiohttp_utils.negotiation.Response* is the same as `aiohttp.web.Response`, except that its first argument is `data`, the data to be negotiated.

2.1.1 Customizing negotiation

Renderers are just callables that receive a `request` and the data to render.

Renderers can return either the rendered representation of the data or a `Response`.

Example:

```
# A simple text renderer
def render_text(request, data):
    return data.encode(request.charset)

# OR, if you need to parametrize your renderer, you can use a class

class TextRenderer:
    def __init__(self, charset):
        self.charset = charset

    def __call__(self, request, data):
        return data.encode(self.charset)

render_text_utf8 = TextRenderer('utf-8')
render_text_utf16 = TextRenderer('utf-16')
```

You can then pass your renderers to *setup* with a corresponding media type.

```
from collections import OrderedDict
from aiohttp_utils import negotiation

negotiation.setup(app, renderers=OrderedDict([
    ('text/plain', render_text),
    ('application/json', negotiation.render_json),
]))
```

Note: We use an `OrderedDict` of renderers because priority is given to the first specified renderer when the client passes an unsupported media type.

By default, rendering the value returned by a handler according to content negotiation will only occur if this value is considered True. If you want to enforce the rendering whatever the boolean interpretation of the returned value you can set the `force_rendering` flag:

```
from collections import OrderedDict
from aiohttp_utils import negotiation

negotiation.setup(app, force_rendering=True,
                  renderers=OrderedDict([
    ('application/json', negotiation.render_json),
]))
```

aiohttp_utils.negotiation.**setup**(*app: aiohttp.web_app.Application*, ***, *negotiator: callable = <function select_renderer>*, *renderers: collections.OrderedDict = {'application/json': <JSONRenderer()>}*, *force_negotiation: bool = True*, *force_rendering: bool = False*)

Set up the negotiation middleware. Reads configuration from `app['AIOHTTP_UTILS']`.

Parameters

- **app** (`Application`) – Application to set up.
- **negotiator** – Function that selects a renderer given a request, a dict of renderers, and a `force` parameter (whether to return a renderer even if the client passes an unsupported media type).
- **renderers** (`OrderedDict`) – Mapping of mediatypes to callable renderers.
- **force_negotiation** (`bool`) – Whether to return a renderer even if the client passes an unsupported media type).
- **force_rendering** (`bool`) – Whether to enforce rendering the result even if it considered False.

aiohttp_utils.negotiation.**negotiation_middleware**(*renderers={'application/json': <JSONRenderer()>}*, *negotiator=<function select_renderer>*, *force_negotiation=True*, *force_rendering=False*)

Middleware which selects a renderer for a given request then renders a handler's data to a `aiohttp.web.Response`.

class aiohttp_utils.negotiation.**Response**(*data=None*, **args*, ***kwargs*)

Same as `aiohttp.web.Response`, except that the constructor takes a `data` argument, which is the data to be negotiated by the *negotiation_middleware*.

aiohttp_utils.negotiation.**select_renderer**(*request: aiohttp.web_request.Request*, *renderers: collections.OrderedDict*, *force=True*)

Given a request, a list of renderers, and the `force` configuration option, return a two-tuple of: (media type, render callable). Uses mimeparse to find the best media type match from the ACCEPT header.

class aiohttp_utils.negotiation.**JSONRenderer**

Callable object which renders to JSON.

json_module = <module 'json' from '/home/docs/.pyenv/versions/3.7.9/lib/python3.7/json

aiohttp_utils.negotiation.**render_json = <JSONRenderer()>**

Render data to JSON. Singleton *JSONRenderer*. This can be passed to the `RENDERERS` configuration option, e.g. `('application/json', render_json)`.

2.2 routing - Routing utilities

Routing utilities.

class aiohttp_utils.routing.**ResourceRouter**

Router with an `add_resource()` method for registering method-based handlers, a.k.a "resources". Includes all the methods `aiohttp.web.UrlDispatcher` with the addition of `add_resource`.

Example:

```
from aiohttp import web
from aiohttp_utils.routing import ResourceRouter

app = web.Application(router=ResourceRouter())

class IndexResource:

    async def get(self, request):
        return web.Response(body=b'Got it', content_type='text/plain')

    async def post(self, request):
        return web.Response(body=b'Posted it', content_type='text/plain')


app.router.add_resource_object('/', IndexResource())

# Normal function-based handlers still work
async def handler(request):
    return web.Response()

app.router.add_route('GET', '/simple', handler)
```

By default, handler names will be registered with the name `<ClassName>:<method>`.

```
app.router['IndexResource:post'].url() == '/'
```

You can override the default names by passing a `names` dict to `add_resource`.

```
app.router.add_resource_object('/', IndexResource(), names={'get': 'index_get'})
app.router['index_get'].url() == '/'
```

add_resource_object(*path: str*, *resource*, *methods: tuple = ()*, *names: collections.abc.Mapping = None*)

Add routes by an resource instance's methods.

Parameters

- **path** (`str`) – route path. Should be started with slash (`'/'`).
- **resource** – A "resource" instance. May be an instance of a plain object.
- **methods** (`tuple`) – Methods (strings) to register.
- **names** (`Mapping`) – Dictionary of `name` overrides.

aiohttp_utils.routing.**add_route_context**(*app: aiohttp.web_app.Application*, *module=None*, *url_prefix: str = None*, *name_prefix: str = None*)

Context manager which yields a function for adding multiple routes from a given module.

Example:

```
# myapp/articles/views.py
async def list_articles(request):
    return web.Response(b'article list...')

async def create_article(request):
    return web.Response(b'created article...')
```

```
# myapp/app.py
from myapp.articles import views

with add_route_context(app, url_prefix='/api/', name_prefix='articles') as route:
    route('GET', '/articles/', views.list_articles)
    route('POST', '/articles/', views.create_article)

app.router['articles.list_articles'].url()  # /api/articles/
```

If you prefer, you can also pass module and handler names as strings.

```
with add_route_context(app, module='myapp.articles.views',
                       url_prefix='/api/', name_prefix='articles') as route:
    route('GET', '/articles/', 'list_articles')
    route('POST', '/articles/', 'create_article')
```

Parameters

- **app** (Application) – Application to add routes to.
- **module** – Import path to module (str) or module object which contains the handlers.
- **url_prefix** (str) – Prefix to prepend to all route paths.
- **name_prefix** (str) – Prefix to prepend to all route names.

aiohttp_utils.routing.**add_resource_context**(*app: aiohttp.web_app.Application*, *module=None*, *url_prefix: str = None*, *name_prefix: str = None*, *make_resource=<function <lambda>>*)

Context manager which yields a function for adding multiple resources from a given module to an app using *ResourceRouter*.

Example:

```
# myapp/articles/views.py
class ArticleList:
    async def get(self, request):
        return web.Response(b'article list...')

class ArticleDetail:
    async def get(self, request):
        return web.Response(b'article detail...')
```

```
# myapp/app.py
from myapp.articles import views

with add_resource_context(app, url_prefix='/api/') as route:
    route('/articles/', views.ArticleList())
    route('/articles/{pk}', views.ArticleDetail())
```

(continues on next page)

(continued from previous page)

```
app.router['ArticleList:get'].url()  # /api/articles/
app.router['ArticleDetail:get'].url(pk='42')  # /api/articles/42
```

If you prefer, you can also pass module and class names as strings.

```
with add_resource_context(app, module='myapp.articles.views',
                          url_prefix='/api/') as route:
    route('/articles/', 'ArticleList')
    route('/articles/{pk}', 'ArticleDetail')
```

Note: If passing class names, the resource classes will be instantiated with no arguments. You can change this behavior by overriding `make_resource`.

```
# myapp/authors/views.py
class AuthorList:
    def __init__(self, db):
        self.db = db

    async def get(self, request):
        # Fetch authors from self.db...
```

```
# myapp/app.py
from myapp.database import db

with add_resource_context(app, module='myapp.authors.views',
                          url_prefix='/api/',
                          make_resource=lambda cls: cls(db=db)) as route:
    route('/authors/', 'AuthorList')
```

Parameters

- **app** (`Application`) – Application to add routes to.
- **resource** – Import path to module (str) or module object which contains the resource classes.
- **url_prefix** (`str`) – Prefix to prepend to all route paths.
- **name_prefix** (`str`) – Prefix to prepend to all route names.
- **make_resource** – Function which receives a resource class and returns a resource instance.

CHAPTER 3

Project info

3.1 Changelog

3.1.1 3.2.1 (2023-02-14)

- Actually bump package version...

3.1.2 3.2.0 (2023-02-14)

- Add support for Python 3.10 and 3.11
- Switch ci to gh actions
- Rename package name as aiohttp-utils

3.1.3 3.1.1 (2019-07-11)

- [negotiation]: Fix handling of aiohttp.web.Response when force_rendering is set.
- Add examples to the MANIFEST.

3.1.4 3.1.0 (2019-07-09)

- Add support for aiohttp>=3.
- Drop support for Python 3.4.
- Test against Python 3.7.
- [negotiation]: Add a new `force_rendering` config option.

3.1.5 3.0.0 (2016-03-16)

- Test against Python 3.6.
- [runner] *Backwards-incompatible*: The `runner` module is deprecated. Install `aiohttp-devtools` and use the `adev runserver` command instead.
- [path_norm] *Backwards-incompatible*: The `path_norm` module is removed, as it is now available in `aiohttp` in `aiohttp.web_middlewares.normalize_path_middleware`.

3.1.6 2.0.1 (2016-04-03)

- [runner] Fix compatibility with aiohttp>=0.21.0 (#2). Thanks @charlesfleche for reporting.

3.1.7 2.0.0 (2016-03-13)

- Fix compatibility with aiohttp>=0.21.0.
- [routing] *Backwards-incompatible*: Renamed `ResourceRouter.add_resource` to `ResourceRouter.add_resource_object` to prevent clashing with aiohttp's URLDispatcher.

3.1.8 1.0.0 (2015-10-27)

- [negotiation,path_norm] *Backwards-incompatible*: Changed signatures of `negotiation.setup` and `path_norm.setup` to be more explicit. Both now take keyword-only arguments which are the same as the module's configuration keys, except lowercased, e.g. `setup(app, append_slash=True, merge_slashes=True)`.
- [runner] Make `run` importable from top-level `aiohttp_utils` module.
- [runner] Fix behavior when passing `reload=False` when `app.debug=True`
- Improved docs.

3.1.9 0.1.0 (2015-10-25)

- First PyPI release. Includes `negotiation`, `path_norm`, `routing`, and `runner` modules.

3.2 Versioning

aiohttp_utils follows a form of sentimental versioning. Given that this package contains indepedendent modules with varying degrees of stability, semver doesn't quite fit.

Major breaking changes will be clearly documented in the *changelog*.

3.3 License

```
Copyright 2015-2018 Steven Loria
Copyright 2019      David Douard

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

Python Module Index

a

Index