aiotraversal

Release 0.8.2

January 24, 2016

Contents

This is small web framework, around aiohttp_traversal.

CHAPTER 1

Hello World

app.py:

```
import asyncio

from aiohttp.web import Response

from aiohttp_traversal.ext.resources import Root
from aiohttp_traversal.ext.views import View, RESTView

from aiotraversal import Application
from aiotraversal.cmd import run


class HelloView(View):
    @asyncio.coroutine
    def __call__(self):
        return Response(text="Hello World!\n")


class HelloJSON(RESTView):
    methods = {'get'}

    @asyncio.coroutine
    def get(self):
        return dict(text="Hello World!")


def main():
    loop = asyncio.get_event_loop()

    app = Application()  # create main application instance

    with app.configure(loop=loop) as config:  # start configure process
        config.include('aiotraversal.cmd')  # include module for command-line parsing
        config.include('aiotraversal.serve')  # include module for start serving
        config.bind_view(Root, HelloView)  # add view for '/'
        config.bind_view(Root, HelloJSON, 'json')  # add view for '/json'

    run(app, loop=loop)  # start application


if __name__ == '__main__':
    main()
```

```
$ python app.py serve
```

```
$ curl http://localhost:8080
Hello World!
$ curl http://localhost:8080/json
{"text": "Hello World!"}
```

CHAPTER 2

Content

2.1 Command-line interface

Module `aiotraversal.cmd` makes if easy to write command-line interfaces for you application.

2.1.1 Parser

After `config.include('aiotraversal.cmd')`, you can use these objects:

- `config['cmd']['parser']`: *ArgumentParser* instance;
- `config['cmd']['subparsers']`: subparsers for create commands, the main tool for extend console application;

After configure process finished:

- `app['cmd']['args']`: `Namespace` instance from `config['cmd']['parser'].parse_args()`;
- `app['cmd']['run_func']`: function for *Function run*;

Add commands

Something like this:

```
import asyncio

from aiotraversal import Application
from aiotraversal.cmd import run

def cmd_func(app, loop):
    print('cmd_func is called!')


def main():
    loop = asyncio.get_event_loop()
    app = Application()

    with app.configure(loop=loop) as config:
        config.include('aiotraversal.cmd')  # include

        subparsers = config['cmd']['subparsers']
        parser_test = subparsers.add_parser('test_command', help="Test")  # create subparser
```

```
        parser_test.set_defaults(func=cmd_func)  # add function for start
        # ... extend subpaser with `parser_test.add_argument`

    run(app, loop)  # in this place cmd_func is called
```

Now, if run your application with argument `test_command` (e.g. `my_cmd test_command`), "cmd_func is called!" printed.

Default key `func` of subparser, is magic for bind functions to commands. It is called from the *Function run* with two arguments: `app` and `loop`.

ArgumentParser

`ArgumentParser` is modified for grouping subcommand arguments.

StackOverflow with this solution.

2.1.2 Function `run`

After configure process, you must run `aiotraversal.cmd.run`. It run `app['cmd']['run_func']`, finish application and close loop.

2.2 Serve helper

For add command serving, include `aiotraversal.serve`:

```
import asyncio

from aiotraversal import Application
from aiotraversal.cmd import run

def main():
    loop = asyncio.get_event_loop()
    app = Application()  # create main application instance

    with app.configure(loop=loop) as config:
        config.include('aiotraversal.cmd')
        config.include('aiotraversal.serve')  # include module for serving
        # some other includes

    run(app, loop=loop)  # start application
```

```
$ python app.py serve --help
usage: app.py serve [-h] [--listen HOST:PORT] [--static DIR]

optional arguments:
  -h, --help          show this help message and exit
  --listen HOST:PORT  host and port for listen (default 'localhost:8080')
  --static DIR        Serve static files
```

2.2.1 Objects

- `config['cmd']['parser_serve']`: subparser for `serve` command;

2.2.2 Arguments

`serve` command have some arguments.

`--listen`

Adderss for listen. Default `localhost:8080`.

Host or port may be not specified. E.g.:

- `--listen 0.0.0.0` equal `--listen 0.0.0.0:8080`
- `--listen :8082` equal `--listen localhost:8082`

`--static`

Serve static directory. Specified directory is can be found in `GET /static/`.

Warning: Do not use in production! Access to files is synchronous!

2.2.3 Settings

If *aiotraversal.settings* is included, you can use settings for setup default values.

For example:

```
[serve]
listen = "10.0.0.15:8080"
static = "/srv/static"
```

2.3 Settings

Just include *Settings*.

Settings uses zini.

All keys in section named "app", set to `config['settings']`. Other sections set to `config['settings'][section_name]`.

For example:

```
[app]
author = "ZZZ"

[serve]
listen = ":6543"
```

```
assert app['settings']['author'] == 'ZZZ'
assert app['settings']['serve']['listen'] == ':6543'
```

2.3.1 Objects

- `config['settings_ini']`: instance of *Zini <https://github.com/zzzsochi/zini>*;
- `config['settings']['file']`: path to file with settings;

2.3.2 Arguments

aiotraversal.cmd module is **not** required.

`--settings`

Path to ini-file with setings. This is setup `config['settings']['file']`.