freesia Documentation

Release 0.0.1

ArianX

Apr 30, 2019

Contents:

CHAPTER 1

API

1.1 app.py

This module implements the async app of the web framework.

class freesia.app.**Freesia**

The main class of this framework.

add_route(*rule: str*, *methods: Iterable[str] = None*, *target: Callable = None*, *options: MutableMapping[KT, VT] = None*, *view_func: Callable = None*) → None

Internal method of *route()*.

Parameters

- **rule** – url rule
- **methods** – the method that the target function should handles.
- **target** – target function
- **options** – optional prams
- **view_func** – the class based view. See *freesia.view.View*.

Returns None

cast(*res: Any*) → freesia.utils.Response

Cast the res made by the user's handler to the normal response.

Parameters **res** – route returned value

Returns the instance of freesia.response.Response

dispatch_request(*request: aiohttp.web_request.BaseRequest*) → freesia.utils.Response

Dispatch request.

Parameters **request** – the instance of aiohttp.web.BaseRequest

Returns

groups = None
collected groups

handler(*request: aiohttp.web_request.BaseRequest*) → freesia.utils.Response
hands out a incoming request

Parameters **request** – the instance of `aiohttp.web.BaseRequest`

Returns result

register_group(*group: Any*) → None
Register `freesia.groups.Group` to the app.

Parameters **group** – The instance of `freesia.groups.Group`.

Returns None

route(*rule: str*, ***options*) → Callable
Register the new route to the framework.

Parameters

- **rule** – url rule
- **options** – optional params

Returns a decorator to collect the target function

route_cls
alias of *freesia.route.Route*

rules = None
collected routes

run(*host='localhost'*, *port=8080*)
start a async serve

serve(*host: str*, *port: int*)
Start to serve. Should be placed in a event loop.

Parameters

- **host** – host
- **port** – port

Returns None

set_filter(*name: str, url_filter: Tuple[str, Union[None, Callable], Union[None, Callable]]*)
Add url filter. For more information see *route_cls*

Parameters

- **name** – name of the url filter
- **url_filter** – A tuple that includ regex, in_filter and out_filter

Returns None

traverse_middleware(*request: aiohttp.web_request.BaseRequest*, *user_handler: Callable*) → Any
Call all registered middleware.

url_map_cls
alias of *freesia.route.Router*

use(*middleware: Iterable[T_co]*) → None
Register the middleware for this framework. See example:

```
async def middleware(request, handler):
    print("enter middleware")
    return await handler()

app = Freesia()
app.use([middleware])
```

Parameters **middleware** – A tuple of the middleware.

Returns None

1.2 route.py

This module implements the route class of the framework.

class freesia.route.**AbstractRoute**(*rule: str, methods: Iterable[str], target: Callable[[...], Any], options: MutableMapping[KT, VT]*)

AbstractRoute can only be used if you want to replace the default *Route*. If you really want to do, you should inherit this class and implement the methods `__init__()`, `set_filter()` it requires. Then replace the default `app.Freesia.route_cls` with you own defined class before instantiating `app.Freesia`. See example:

```
class CustomRoute(AbstractRoute):
    def __init__(self, rule, methods, target, options):
        pass

    def set_filter(self, name, url_filter):
        pass

Freesia.route_cls = CustomRoute
```

Parameters

- **rule** – The url rule of the route.
- **methods** – The method list that this route can accept.
- **target** – The handler function that handles the request.
- **options** – Optional control parameters.

class freesia.route.**AbstractRouter**

AbstractRouter can only be used if you want to replace the default *Router*. If you really want to do, you should inherit this class and implement the methods `add_route()`, `get()` it requires. Then replace the default `app.Freesia.url_map_cls` with you own defined class before instantiating `app.Freesia`. See example:

```
class CustomRouter(AbstractRouter):
    def add_route(self, route):
        pass

    def get(self, rule, method):
        pass

Freesia.url_map_cls = CustomRouter
```

class freesia.route.**Route**(*rule*, *methods*, *target*, *options*)
Default route class.

Parameters

- **rule** – The url rule of the route.
- **methods** – The method list that this route can accept.
- **target** – The handler function that handles the request.
- **options** – Optional control parameters.

classmethod iter_token(*rule: str*) → Tuple[str, str]
Traverse the rule and generate the prefix and param info.

Parameters **rule** – url rule to be iter

Returns A tuple that includ the url filter name and param name.

match(*path: str*, *method: str*) → Union[None, List[Any]]
Check that this route matches the incoming parameters.

Parameters

- **path** – path to be matched
- **method** – the request method

Returns A List of the matching param or None.

param_check() → bool
Check if the number of parameters matches.

Returns bool

parse_pattern() → None
Parse the `rule` to get regex pattern then store in `regex_pattern`

Returns None

classmethod set_filter(*name: str, url_filter: Tuple[str, Union[None, Callable], Union[None, Callable]]*) → None
Set a custom filter to the route.

Parameters

- **name** – filter name
- **url_filter** – A tuple that includ regex, in_filter and out_filter

Returns None

class freesia.route.**Router**
Default router.

add_route(*route: freesia.route.Route*) → None
Add a route to the router.

Parameters **route** – the instance of the *Route*

Returns None

get(*path: str*, *method: str*) → Tuple[Callable, Tuple]
Match giving path. Throw a exception if not matches.

Parameters

- **path** – incoming path.

- **method** – the method of the request.

Returns A tuple include the handler function and the params.

get_from_static_url(*path: str*, *method: str*) → Tuple[Callable, Tuple]
Match the static url. Throw a exception if not matches.

Parameters

- **path** – incoming path
- **method** – the method of the request

Returns A tuple include the handler function and the params.

1.3 groups.py

This module implements the *Group* of the web framework.

class freesia.group.**Group**(*name: str*, *url_prefix: str*)
Use group to divide an app by the different logic. Its instance will be added in *freesia.app.Freesia.groups*.

Parameters

- **name** – Name of this group.
- **url_prefix** – Url prefix of this group. All rules registered to this group will be prefixed to the *url_prefix*.

1.4 view.py

This module implements the class based view of the web framework.

class freesia.view.**MethodMetaView**(*name*, *bases*, *d*)
A meta used by class based class to collect the implemented methods.

class freesia.view.**MethodView**(**args*, ***kwargs*)
Method based class view. See example:

```
class MyView(MethodView):
    def get(self, request, name):
        pass

app = Freesia()
app.add_route("/person/<name>", MyView.as_view())
```

class freesia.view.**View**(**args*, ***kwargs*)
The basic view class. You must create a new class to inherit it and implement the `View.dispatch_request()`. And the call `View.as_view()` with *freesia.app.Freesia.add_route()* to register the view. Like:

```
class MyView(View):
    self dispatch_request(self, request):
        pass

app = Freesia()
app.add_route("/my-view", view_func=MyView.as_view())
```

1.5 session.py

This module implements the cookie based async session.

class freesia.session.**Session**(*data: MutableMapping[KT, VT] = None*, *max_age: float = None*)
A dict like object to represent the session attribute.

class freesia.session.**SessionInterface**(*, *cookie_name: str = 'FREESIA_SESSION'*, *domain: str = None*, *max_age: float = None*, *path: str = '/'*, *secure: bool = False*, *httponly: bool = True*, *json_encoder: Callable = <function asy_json_dump>*, *json_decoder: Callable = <function asy_json_load>*)
Abstract session interface. Inherit this class and implement the `SessionInterface.load_session()` and `SessionInterface.save_session()`.

class freesia.session.**SimpleCookieSession**(*, *cookie_name: str = 'FREESIA_SESSION'*, *domain: str = None*, *max_age: float = None*, *path: str = '/'*, *secure: bool = False*, *httponly: bool = True*, *json_encoder: Callable = <function asy_json_dump>*, *json_decoder: Callable = <function asy_json_load>*)
Simple cookie session.

freesia.session.**get_session**(*request: aiohttp.web_request.BaseRequest*) → freesia.session.Session
Get session from request. It must be used after call *set_up_session()*.

freesia.session.**new_session**(*request: aiohttp.web_request.BaseRequest*) → freesia.session.Session
Build a new session then save in request. It must be used after call *set_up_session()*.

freesia.session.**set_up_session**(*app: freesia.app.Freesia*, *session_interface: Callable*)
Setup the session middleware to the app.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index