jinja2-sanic Documentation

Release 0.1.0

Yun Xu

Aug 29, 2017

Contents

a jinja2 template renderer for Sanic. It supports:

- function based web handlers
- class-based views
- decoractors for convenient useage

CHAPTER 1

Quick start

Let's get started.

```
from sanic import Sanic
from sanic.views import HTTPMethodView
from sanic.exceptions import ServerError

app = Sanic("sanic_jinja2_render")

# Setup jinja2 environment
template = "<html><body><h1>{{Player}}</h1>{{Category}}</body></html>"
jinja2_sanic.setup(
    app,
    loader=jinja2.DictLoader(
        {
            "templates.jinja2": template
        }
    )
)

# Usage in function based web handlers
@app.route("/")
@jinja2_sanic.template("templates.jinja2")
async def func(request):
    return {
        "Player": "CR7",
        "Category": "Soccer",
    }

# Usage in class-based views
class SimpleView(HTTPMethodView):

    @jinja2_sanic.template("templates.jinja2")
    async def get(self, request):
        return {
            "Player": "CR7",
```

```
            "Category": "Soccer",
        }

# register class based view routes
app.add_route(SimpleView.as_view(), "/")

# Start Server
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000)
```

Contents:

1.1 Installation

1.1.1 Installing it globally

You can install jinja2-sanic globally with any Python package manager:

```
pip3 install jinja2-sanic
```

1.2 Example

A simple example.

```
from sanic import Sanic
from sanic.views import HTTPMethodView
from sanic.exceptions import ServerError

app = Sanic("sanic_jinja2_render")

# Setup jinja2 environment
template = "<html><body><h1>{{Player}}</h1>{{Category}}</body></html>"
jinja2_sanic.setup(
    app,
    loader=jinja2.DictLoader(
        {
            "templates.jinja2": template
        }
    )
)

# Usage in function based web handlers
@app.route("/")
@jinja2_sanic.template("templates.jinja2")
async def func(request):
    return {
        "Player": "CR7",
        "Category": "Soccer",
    }

# Usage in class-based views
class SimpleView(HTTPMethodView):
```

```
    @jinja2_sanic.template("templates.jinja2")
    async def get(self, request):
        return {
            "Player": "CR7",
            "Category": "Soccer",
        }

# register class based view routes
app.add_route(SimpleView.as_view(), "/")

# Start Server
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000)
```

If you have more complicated processing, you may wanna use *render_template()* function.

```
from sanic import Sanic
from sanic.views import HTTPMethodView
from sanic.exceptions import ServerError

app = Sanic("sanic_jinja2_render")

# Setup jinja2 environment
template = "<html><body><h1>{{Player}}</h1>{{Category}}</body></html>"
jinja2_sanic.setup(
    app,
    loader=jinja2.DictLoader(
        {
            "templates.jinja2": template
        }
    )
)

# Usage in function based web handlers
@app.route("/")
async def func(request):
    context = {
        "Player": "CR7",
        "Category": "Soccer",
    }
    resp = jinja2_sanic.render_template("templates.jinja2", request, context)

    # Custom Processing
    resp.headers['Access-Control-Allow-Origin'] = '*'

    return resp

# Usage in class-based views
class SimpleView(HTTPMethodView):

    async def get(self, request):
        context = {
            "Player": "CR7",
            "Category": "Soccer",
        }
        resp = jinja2_sanic.render_template("templates.jinja2", request, context)

        # Custom Processing
```

```
        resp.headers['Access-Control-Allow-Origin'] = '*'

        return resp

# register class based view routes
app.add_route(SimpleView.as_view(), "/")

# Start Server
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000)
```

1.3 Globals

By default there is one sanic specific *jinja2.Environment().globals* available in templates:

```
<body>
    <h1>Welcome to {{ app.name }}</h1>
</body>
```

Feel free to create issue or pull request if you wanna add anything useful as *globals*.

1.4 API References

jinja2_sanic.**get_env**(*app*, *, *app_key='sanic_jinja2_environment'*)
Get Jinja2 env by *app_key*.

Parameters

- **app** – a Sanic instance
- **app_key** – a optional key for application instance. If not provided, default value will be used.

jinja2_sanic.**render_string**(*template_name*, *request*, *context*, *, *app_key='sanic_jinja2_environment'*)
Render a string by filling Template template_name with context. Returns a string.

Parameters

- **template_name** – template name.
- **request** – a parameter from web-handler, sanic.request.Request instance.
- **context** – context for rendering.
- **app_key** – a optional key for application instance. If not provided, default value will be used.

jinja2_sanic.**render_template**(*template_name*, *request*, *context*, *, *app_key='sanic_jinja2_environment'*, *encoding='utf-8'*, *headers=None*, *status=200*)
Return sanic.response.Response which contains template template_name filled with context. Returned response has Content-Type header set to 'text/html'.

Parameters

- **template_name** – template name.

- **request** – a parameter from web-handler, sanic.request.Request instance.
- **context** – context for rendering.
- **encoding** – response encoding, 'utf-8' by default.
- **status** – HTTP status code for returned response, 200 (OK) by default.
- **app_key** – a optional key for application instance. If not provided, default value will be used.

jinja2_sanic.**setup**(*app*, **args*, *app_key='sanic_jinja2_environment'*, *context_processors=()*, *filters=None*, ***kwargs*)

Initialize jinja2.Environment object.

Parameters

- **app** – a Sanic instance
- **app_key** – an optional key for application instance. If not provided, default value will be used.
- **context_processors** – context processors that will be used in request middlewares.
- **and kwargs** (*args*) – will be passed to environment constructor.

jinja2_sanic.**template**(*template_name*, ***, *app_key='sanic_jinja2_environment'*, *encoding='utf-8'*, *headers=None*, *status=200*)

Decorate web-handler to convert returned dict context into sanic.response.Response filled with template_name template.

Parameters

- **template_name** – template name.
- **request** – a parameter from web-handler, sanic.request.Request instance.
- **context** – context for rendering.
- **encoding** – response encoding, 'utf-8' by default.
- **status** – HTTP status code for returned response, 200 (OK) by default.
- **app_key** – a optional key for application instance. If not provided, default value will be used.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

j

Index