Grole Documentation

Release 0.3.0

witchard

Jun 13, 2020

Contents:

CHAPTER 1

Install

Either download grole.py directly from github and place in your project folder, or *pip3 install grole*.

CHAPTER 2

Tutorial

2.1 Getting started

By default, grole will run a simple static file server in the current directory. To use this simply execute *grole.py*, or run *python -m grole*.

To serve your own functions, you first need a `Grole` object. The constructor accepts a *env* variable which is passed to your handler functions such that you can share state between them. Logging is done by the python logging module, if you want logging then run *logging.basicConfig(level=logging.INFO)*.

Once you have setup handler functions for your web API, you can then launch the server with `Grole.run()`. This takes the host and port to serve on and does not return until interrupted.

2.2 Registering routes

Routes are registered to a `Grole` object by decorating a function with the `Grole.route()` decorator. The decorator function takes a regular expression as the path to match, an array of HTTP methods (GET, POST, etc), and whether you want this function in the API doc. Docstrings of functions in the API doc are available through *env['doc']* within the handler function.

The order in which routes are registered is the order in which they will be tested when searching for a handler for a specific request.

2.3 Handling requests

Decorated functions registered to a `Grole` object with `Grole.route()` will be called if their associated regular expression matches that of a request (as well as the HTTP method).

A registered handler is given the following objects:

- env: The *env* dictionary that the `Grole` object was constructed with

- req: A `Request` object containing the full details of the request. The `re.MatchObject` from the path match is also added in as *req.match*.

We now know enough to make a simple web API. An example of how to return the hex value when visiting */<inteter>* is shown below:

```
from grole import Grole

app = Grole()

@app.route('/(\d+)')
def tohex(env, req):
    return hex(int(req.match.group(1)))

app.run()
```

If you need to do something *async* within your handler, e.g. access a database using aioodbc then simply declare your handler as *async* and *await* as needed.

2.4 Responding

In-built python types returned by registered request handlers are automatically converted into 200 OK HTTP responses. The following mappings apply:

- bytes: Sent directly with content type text/plain
- string: Encoded as bytes and sent with content type text/html
- others: Encoded as json and sent with content type application/json

Finer grained control of the response data can be achieved using `ResponseBody` or one of it's children. These allow for overriding of the content type. The following are available:

- `ResponseBody`: bytes based response
- `ResponseString`: string based response
- `ResponseJSON`: json encoded response
- `ResponseFile`: read a file to send as response

Control of the headers in the response can be achieved by returning a `Response` object. This allows for sending responses other than 200 OK, for example.

2.5 Helpers

Various helper functions are provided to simplify common operations:

- `serve_static()`: Serve static files under a directory. Optionally provide simple directory indexes.
- `serve_doc()`: Serve API documentation (docstrings) of registered request handlers using a simple plain text format.

CHAPTER 3

Examples

```
from grole import Grole

app = Grole()

@app.route('/(.*)?')
def index(env, req):
    name = req.match.group(1) or 'World'
    return 'Hello, {}!'.format(name)

app.run()
```

Run this script and then point your browser at http://localhost:1234/.

Grole also has an inbuilt simple file server which will serve all the files in a directory. Just run *grole.py* or *python -m grole*. This supports the following command line arguments:

- *–address* - The address to listen on, empty string for any address
- *–port* - The port to listen on
- *–directory* - The directory to serve
- *–noindex* - Do not show file indexes
- *–verbose* - Use verbose logging (level=DEBUG)
- *–quiet* - Use quiet logging (level=ERROR)

Further examples can be found within the examples folder on github.

CHAPTER 4

Grole API

Grole is a python (3.5+) nano web framework based on asyncio. It's goals are to be simple, embedable (single file and standard library only) and easy to use.

class `grole.`**`Grole`**(*env={}*)

Bases: `object`

A Grole Webserver

`__init__`(*env={}*)

Initialise a server

env is passed to request handlers to provide shared state. Note, env by default contains doc which is populated from registered route docstrings.

`route`(*path_regex, methods=['GET'], doc=True*)

Decorator to register a handler

Parameters:

- path_regex: Request path regex to match against for running the handler
- methods: HTTP methods to use this handler for
- doc: Add to internal doc structure

`run`(*host='localhost'*, *port=1234*, *ssl_context=None*)

Launch the server. Will run forever accepting connections until interrupted.

Parameters:

- host: The host to listen on
- port: The port to listen on
- ssl_context: The SSL context passed to asyncio

class `grole.`**`Request`**

Bases: `object`

Represents a single HTTP request

The following members are populated with the request details:

- method: The request method
- location: The request location as it is sent
- path: The unescaped path part of the location
- query: The query string part of the location (if present)
- version: The request version, e.g. HTTP/1.1
- headers: Dictionary of headers from the request
- data: Raw data from the request body
- match: The re.MatchObject from the successful path matching

`body``()`
    Decodes body as string

`json``()`
    Decodes json object from the body

`class` `grole.`**`Response`**`(`*data=None*, *code=200*, *reason='OK'*, *headers={}*, *version='HTTP/1.1'*`)`
    Bases: `object`

Represents a single HTTP response

`__init__``(`*data=None*, *code=200*, *reason='OK'*, *headers={}*, *version='HTTP/1.1'*`)`
    Create a response

Parameters:

- data: Object to send e.g. ResponseBody / ResponseJSON.
- code: The response code, default 200
- reason: The response reason, default OK
- version: The response version, default HTTP/1.1
- headers: Dictionary of response headers, default is a Server header and those from the response body

Note, data is intelligently converted to an appropriate ResponseXYZ object depending on it's type.

`class` `grole.`**`ResponseBody`**`(`*data=b''*, *content_type='text/plain'*`)`
    Bases: `object`

Response body from a byte string

`__init__``(`*data=b''*, *content_type='text/plain'*`)`
    Initialise object, data is the data to send

Parameters:

- data: Byte data to send
- content_type: Value of Content-Type header, default text/plain

`class` `grole.`**`ResponseFile`**`(`*filename*, *content_type=None*`)`
    Bases: *`grole.ResponseBody`*

Respond with a file

Content type is guessed if not provided

__init__(*filename*, *content_type=None*)
Initialise object, data is the data to send

Parameters:

- filename: Name of file to read and send
- content_type: Value of Content-Type header, default is to guess from file extension

class grole.**ResponseJSON**(*data=”*, *content_type=’application/json’*)
Bases: `grole.ResponseString`

Response body encoded in json

__init__(*data=”*, *content_type=’application/json’*)
Initialise object, data is the data to send

Parameters:

- data: Object to encode as json for sending
- content_type: Value of Content-Type header, default application/json

class grole.**ResponseString**(*data=”*, *content_type=’text/html’*)
Bases: `grole.ResponseBody`

Response body from a string

__init__(*data=”*, *content_type=’text/html’*)
Initialise object, data is the data to send

Parameters:

- data: String data to send
- content_type: Value of Content-Type header, default text/plain

grole.**main**(*args=[’-b’, ’latex’, ’-D’, ’language=en’, ’-d’, ’_build/doctrees’, ’.’, ’_build/latex’]*)
Run Grole static file server

grole.**parse_args**(*args=[’-b’, ’latex’, ’-D’, ’language=en’, ’-d’, ’_build/doctrees’, ’.’, ’_build/latex’]*)
Parse command line arguments for Grole server running as static file server

grole.**serve_doc**(*app*, *url*)
Serve API documentation extracted from request handler docstrings

Parameters:

- app: Grole application object
- url: URL to serve at

grole.**serve_static**(*app*, *base_url*, *base_path*, *index=False*)
Serve a directory statically

Parameters:

- app: Grole application object
- base_url: Base URL to serve from, e.g. /static
- base_path: Base path to look for files in
- index: Provide simple directory indexes if True

Grole is a python (3.5+) nano web framework based on asyncio. It’s goals are to be simple, embedable (single file and standard library only) and easy to use. The authors intention is that it should support standing up quick and dirty web based APIs.

It's loosely based on bottle and flask, but unlike them does not require a WSGI capable server to handle more than one request at once. Sanic is similar, but it does not meet the embedable use-case.

A grole is a multi-spouted drinking vessel (https://en.wikipedia.org/wiki/Grole), which harks to this modules bottle/flask routes but with the ability to serve multiple drinkers at once!

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index