tornado-smack Documentation

Release 0.1

ybrs

December 18, 2016

Contents

CHAPTER 1

Syntactic sugar for tornado

Turns your application from this:

```
class MainHandler(tornado.web.RequestHandler):
    def get(self, name):
        self.write("Hello, world %s " % name)

application = tornado.web.Application([
    (r"^/foobar/(\w+)/?$", MainHandler),
])

if __name__ == "__main__":
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

to this:

```
from tornado_smack import App

app = App()

@app.route("/foobar/<name>")
def foobar(name):
    return "hello world %s" % name

if __name__ == "__main__":
    app.run(debug=True)
```

CHAPTER 2

Using templates

you can use ./templates folder - default path - and return a template easily like this:

```
from tornado_smack import render_template
@app.route("/foobar/<name>")
def foobar(name):
    return render_template('foobar.html', name=name)
```

CHAPTER 3

Using Async. Handlers

also for your async pleasure, you can do this:

```
@app.route('/async', methods=['GET', 'HEAD'])
@coroutine
def homepage(self):
    http_client = AsyncHTTPClient()
    response = yield http_client.fetch("https://google.com/")
    self.write(response.body)
```

CHAPTER 4

Using Smack with Tornado together

you can always use them together like this:

```
from tornado_smack import App
import tornado.web

app = App()

@app.route("/foobar/<id>")
def foobar2(id):
    return "hello world %s" % id

class MainHandler(tornado.web.RequestHandler):
    def get(self, name):
        self.write("Hello, world %s " % name)


application = tornado.web.Application([
    (r"^/foo/(\w+)/?$", MainHandler),
] + app.get_routes())


if __name__ == "__main__":
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

CHAPTER 5

Installation

You can install it with pip:

```
pip install tornado_smack
```

CHAPTER 6

Api Documentation

class tornado_smack.app.**App**(*debug=False*, *template_path=None*, *template_engine='tornado'*)

Example usage:

```
from tornado_smack import App

app = App(debug=True)

@app.route("/hello")
def foo():
    return "hello"
```

Parameters

- **debug** – enables werkzeug debugger
- **template_path** – we normally look for template in ./templates folder of your app.py you can explicitly set for some other template path

get_routes()

returns our compiled routes and classes as a list to be used in tornado

is_werkzeug_route(*route*)

does it look like a werkzeug route or direct reg exp. of tornado.

route(*rule*, *methods=None*, *werkzeug_route=None*, *tornado_route=None*, *handler_bases=None*, *nowrap=None*)

our super handy dandy routing function, usually you create an application, and decorate your functions so they become RequestHandlers:

```
app = App()

@app.route("/hello")
def hello():
    return "foo"
```

Parameters

- **rule** – this can be either a werkzeug route or a reg.expression as in tornado. we try to understand the type of it automatically - wheter werkzeug or reg.exp. - this by checking with a regexp. If it is a werkzeug route, we simply get the compiled reg. exp from werkzeug and pass it to tornado handlers.

- **methods** – methods can be a combination of ['GET', 'POST', 'HEAD', 'PUT'...] any http verb that tornado accepts. Behind the scenes we create a class and attach these methods.

  for example something like:

```
class HelloHandler(tornado.web.RequestHandler):
    def get(self):
```

- **werkzeug_route** – we explicitly tell that this is a werkzeug route, in case auto detection fails.
- **tornado_route** – we explicitly tell that this is a tornado reg. exp. route
- **handler_bases** – for debug we create DebuggableHandler, and for normal operations we create tornado.web.RequestHandler but in case you want to use your own classes for request handling, you can pass it with handler_bases parameter. So behind the scenes this:

```
@route("/foo", handler_bases=(MyHandler,))
def foo():
    pass
```

  becomes this:

```
class HelloHandler(MyHandler):
    def get(self):
        ...
```

  if you set a base class for your FooHandler, in debug mode we'll add DebuggableHandler in between handler.__class__.__mro__ (<class 'tornado_smack.app.FooHandler'>, <class 'tornado_smack.app.DebuggableHandler'>, <class '__main__.MyBaseHandler'>, <class 'tornado.web.RequestHandler'>, <type 'object'>)

- **nowrap** – if you add use self - or handler - as your first parameter:

```
@route('/foo')
def foo(self):
    self.write("hello")
```

  if becomes something like this:

```
class HelloHandler(tornado.web.RequestHandler):
    def get(self):
        self.write("hello")
```

  if you omit self as your first parameter:

```
@route('/foo')
def foo():
    return "hello"
```

  we implicitly wrap foo so it becomes something like this:

```
class HelloHandler(tornado.web.RequestHandler):
    def get(self, *args, **kwargs):
        def wrapper(*args, **kwargs):
            return foo(*args, **kwargs)
        wrapper(*args, **kwargs)
```

in case you want to use some other name for your first parameter, or for some other reason you can explicitly say don't wrap.

in case you are using tornado.coroutine or some other tornado decorator, we don't wrap your function - because simply it won't work. so this:

```
@route('/foo')
@coroutine
def foo():
    ...
```

will give you an error.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

A

G

I

R