tornado_json Documentation

Release 1.3.2

Author

January 13, 2017

Contents

Tornado-JSON is a small extension of Tornado with the intent providing the tools necessary to get a JSON API up and running quickly. See demos/helloworld/ for a quick example and the accompanying walkthrough in the documentation.

Some of the key features the included modules provide:

- Input and output JSON Schema validation by decorating RequestHandlers with `schema.validate`
- Automated *route generation* with `routes.get_routes(package)`
- *Automated Public API documentation* using schemas and provided descriptions
- Standardized output using the JSend specification

Contents:

CHAPTER 1

Installation

Simply run:

```
pip install Tornado-JSON
```

Alternatively, clone the GitHub repository:

```
git clone https://github.com/hfaran/Tornado-JSON.git
```

CHAPTER 2

Using Tornado-JSON

2.1 A Simple Hello World JSON API

I'll be referencing the helloworld example in the `demos` for this.

We want to do a lot of the same things we'd usually do when creating a Tornado app with a few differences.

2.1.1 helloworld.py

First, we'll import the required packages:

```
import tornado.ioloop
from tornado_json.routes import get_routes
from tornado_json.application import Application
```

Next we'll import the package containing our web app. This is the package where all of your RequestHandlers live.

```
import helloworld
```

Next, we write a lot of the same Tornado "boilerplate" as you'd find in the Tornado helloworld example, except, you don't have to manually specify routes because `tornado_json` gathers those for you and names them based on your project structure and RequestHandler names. You're free to customize `routes` however you want, of course, after they've been initially automatically generated.

```
def main():
    # Pass the web app's package the get_routes and it will generate
    #   routes based on the submodule names and ending with lowercase
    #   request handler name (with 'handler' removed from the end of the
    #   name if it is the name).
    # [("/api/helloworld", helloworld.api.HelloWorldHandler)]
    routes = get_routes(helloworld)

    # Create the application by passing routes and any settings
    application = Application(routes=routes, settings={})

    # Start the application on port 8888
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

2.1.2 helloworld/api.py

Now comes the fun part where we develop the actual web app. We'll import `APIHandler` (this is the handler you should subclass for API routes), and the `schema.validate` decorator which will validate input and output schema for us.

```
from tornado_json.requesthandlers import APIHandler
from tornado_json import schema

class HelloWorldHandler(APIHandler):
    """Hello!"""
    @schema.validate(...)
    def get(...):
        ...
```

Next, we'll start writing our `get` method, but before writing the body, we'll define an output schema for it and pass it as an argument to the `schema.validate` decorator which will automatically validate the output against the passed schema. In addition to the schema, the docstring for each HTTP method will be used by Tornado-JSON to generate public API documentation for that route which will be automatically generated when you run the app (see the Documentation Generation section for details). Input and output schemas are as per the JSON Schema standard.

```
@schema.validate(output_schema={"type": "string"})
def get(self):
    """Shouts hello to the world!"""
    ...
```

Finally we'll write our `get` method body which will write "Hello world!" back. Notice that rather than using `self.write` as we usually would, we simply return the data we want to write back, which will then be validated against the output schema and be written back according to the JSend specification. The `schema.validate` decorator handles all of this so be sure to decorate any HTTP methods with it.

```
@schema.validate(output_schema={"type": "string"})
def get(self):
    """Shouts hello to the world!"""
    return "Hello world!"
```

2.1.3 Running our Hello World app

Now, we can finally run the app `python helloworld.py`. You should be able to send a GET request to `localhost:8888/api/helloworld` and get a JSONic "Hello world!" back. Additionally, you'll notice an `API_Documentation.md` pop up in the directory, which contains the API Documentation you can give to users about your new and fantastic API.

2.2 Further Examples

See helloworld for further RequestHandler examples with features including:

- Asynchronous methods in RequestHandlers (must use `tornado_json.gen.coroutine` rather than `tornado.gen.coroutine`)
- POSTing (or PUTing, PATCHing etc.) data; `self.body`
- How to generate routes with URL patterns for RequestHandler methods with arguments
- and possibly more!

CHAPTER 3

Request Handler Guidelines

3.1 Schemas and Public API Documentation

Use the `schema.validate` decorator on methods which will automatically validate the request body and output against the schemas provided. The schemas must be valid JSON schemas; readthedocs for an example. Additionally, `return` the data from the request handler, rather than writing it back (the decorator will take care of that).

The docstring of the method, as well as the schemas will be used to generate **public** API documentation.

```
class ExampleHandler(APIHandler):
    @schema.validate(input_schema=..., output_schema=...)
    def post(self):
        """I am the public API documentation of this route"""
        ...
        return data
```

3.2 Assertions

Use `exceptions.api_assert` to fail when some the client does not meet some API pre-condition/requirement, e.g., an invalid or incomplete request is made. When using an assertion is not suitable, `raise APIError( ... )`; don't use `self.fail` directly.

```
class ExampleHandler(APIHandler):
    @schema.validate()
    def post(self):
        ...
        api_assert(condition, status_code, log_message=log_message)
```

CHAPTER 4

Documentation Generation

4.1 Public API Usage Documentation

API Usage documentation is generated by the `tornado_json.api_doc_gen` module. The `api_doc_gen` method is run on startup so to generate documentation, simply run your app and the documentation will written to `API_Documentation.md`. in the current folder.

CHAPTER 5

Creating a REST API Using URL Annotations

You may have noticed that the automatic URL generation is meant to be quick and easy-to-use for simple cases (creating an API in 15 minutes kind of thing).

It is more powerful though, however, as you can customize it to get the URLs for RequestHandlers how you want without having to make additions to output from `routes.get_routes` yourself. This is done through the use of "URL annotations". `APIHandler` and `ViewHandler` have two "magic" attributes (`__urls__` and `__url_names__`) that allow you to define custom routes right in the handler body. See relevant documentation in the REST API example in the demos.

CHAPTER 6

Changelog

6.1 _

6.1.1 1.3.2

- Recovery release for PyPI (1.3.1 had an incomplete module included accidentally)

6.1.2 1.3.1

- Minor updates with versioning

6.1.3 1.3.0

- Added use_defaults support for schema.validate
- Added support for custom validators
- Bugfix: Fixed api_doc_gen duplicated entries
- Bugfix: Remove pyclbr and use inspect instead for module introspection

6.1.4 1.2.2

- *generate_docs* parameter added to *Application* for optional API documentation generation

6.1.5 1.2.1

- arg_pattern now contains hyphen
- Handle case where server would crash when generating docs for methods with

no docstring * Add support for tornado==3.x.x gen.coroutine * Add format_checker kwarg to schema.validate

6.1.6 1.2.0

- **Implement `tornado_json.gen.coroutine`**
  - As a fix for #59, a custom wrapper for the `tornado.gen.coroutine` wrapper has been implemented. This was necessary as we lose the original argspec through it because the wrapper simply has `(*args, **kwargs)` as its signature. Here, we annotate the original argspec as an attribute to the wrapper so it can be referenced later by Tornado-JSON when generating routes.

6.1.7 1.1.0

- Handle routes as `URLSpec` and >2-tuple in `api_doc_gen`
- Refactor `api_doc_gen`; now has public function `get_api_doc` for use

6.1.8 1.0.0

- Compatibility updates for `tornado>=4.0.0`

6.1.9 v0.41

- Fixed `JSendMixin` hanging if auto_finish was disabled

6.1.10 v0.40 - Replace `apid` with parameterized `schema.validate`

- The `apid` class-variable is no longer used
- Schemas are passed as arguments to `schema.validate`
- Method docstrings are used in public API documentation, in place of `apid[method]["doc"]`

6.1.11 v0.31 - On input schema of `None`, input is presumed to be `None`

- Rather than forcing an input schema of `None` with `GET` and `DELETE` methods, whether input is JSON-decoded or not, is dependent on whether the provided input schema is `None` or not. This means that `get` and `delete` methods can now have request bodies if desired.

6.1.12 v0.30 - URL Annotations

- Added `__urls__` and `__url_names__` attributes to allow flexible creation of custom URLs that make creating REST APIs etc. easy
- Added a REST API demo as an example for URL annotations
- Added URL annotations documentation
- Refactored and improved route generation in `routes`

6.1.13 v0.20 - Refactor of `utils` module

Functions that did not belong in `utils` were moved to more relevant modules. This change changes the interface for Tornado-JSON in quite a big way. The following changes were made (that are not backwards compatible).

- `api_assert` and `APIError` were moved to `tornado_json.exceptions`
- `io_schema` was renamed `validate` and moved to `tornado_json.schema`

6.1.14 v0.14 - Bugfixes thanks to 100% coverage

- Fixes related to error-writing in `io_schema` and `APIHandler.write_error`

6.1.15 v0.13 - Add asynchronous compatibility to io_schema

- Add asynchronous functionality to io_schema

6.1.16 v0.12 - Python3 support

- Python3.3, in addition to Python2.7, is now supported.

6.1.17 v0.11 - Duplicate route bugfix

- Fixed bug where duplicate routes would be created on existence of multiple HTTP methods.

6.1.18 v0.10 - Route generation with URL patterns

Route generation will now inspect method signatures in `APIHandler` and `ViewHandler` subclasses, and construct routes with URL patterns based on the signatures. URL patterns match `[a-zA-Z0-9_]+`.

Backwards Compatibility: `body` is no longer provided by `io_schema` as the sole argument to HTTP methods. Any existing code using `body` can now use `self.body` to get the same object.

6.1.19 v0.08 - Input and output example fields

- Add input_example and output_example fields
- status_code 400 on ValidationError
- Exclude delete from input validation

CHAPTER 7

tornado_json Package

7.1 api_doc_gen Module

tornado_json.api_doc_gen.**api_doc_gen**(*routes*)
Get and write API documentation for routes to file

tornado_json.api_doc_gen.**get_api_docs**(*routes*)
Generates GitHub Markdown formatted API documentation using provided schemas in RequestHandler methods and their docstrings.

Parameters **routes** (*[(url, RequestHandler), ...]*) – List of routes (this is ideally all possible routes of the app)

Return type str

Returns generated GFM-formatted documentation

7.2 application Module

class tornado_json.application.**Application**(*routes*, *settings*, *db_conn=None*, *generate_docs=False*)
Bases: tornado.web.Application

Entry-point for the app

- Generate API documentation using provided routes
- Initialize the application

Parameters

- **routes** (*[(url, RequestHandler), ...]*) – List of routes for the app
- **settings** (*dict*) – Settings for the app
- **db_conn** – Database connection
- **generate_docs** (*bool*) – If set, will generate API documentation for provided routes. Documentation is written as API_Documentation.md in the cwd.

7.3 jsend Module

class tornado_json.jsend.**JSendMixin**
Bases: object

http://labs.omniti.com/labs/jsend

JSend is a specification that lays down some rules for how JSON responses from web servers should be formatted.

JSend focuses on application-level (as opposed to protocol- or transport-level) messaging which makes it ideal for use in REST-style applications and APIs.

error(*message*, *data=None*, *code=None*)
An error occurred in processing the request, i.e. an exception was thrown.

Parameters

- **data** (*A JSON-serializable object*) – A generic container for any other information about the error, i.e. the conditions that caused the error, stack traces, etc.
- **message** (*A JSON-serializable object*) – A meaningful, end-user-readable (or at the least log-worthy) message, explaining what went wrong
- **code** (*int*) – A numeric code corresponding to the error, if applicable

fail(*data*)
There was a problem with the data submitted, or some pre-condition of the API call wasn't satisfied.

Parameters **data** (*A JSON-serializable object*) – Provides the wrapper for the details of why the request failed. If the reasons for failure correspond to POST values, the response object's keys SHOULD correspond to those POST values.

success(*data*)
When an API call is successful, the JSend object is used as a simple envelope for the results, using the data key.

Parameters **data** (*A JSON-serializable object*) – Acts as the wrapper for any data returned by the API call. If the call returns no data, data should be set to null.

7.4 requesthandlers Module

class tornado_json.requesthandlers.**APIHandler**(*application*, *request*, ***kwargs*)
Bases: *tornado_json.requesthandlers.BaseHandler*, *tornado_json.jsend.JSendMixin*

RequestHandler for API calls

•Sets header as application/json

•Provides custom write_error that writes error back as JSON rather than as the standard HTML template

initialize()

•Set Content-type for JSON

write_error(*status_code*, ***kwargs*)
Override of RequestHandler.write_error

Calls error() or fail() from JSendMixin depending on which exception was raised with provided reason and status code.

Parameters **status_code** (*int*) – HTTP status code

class tornado_json.requesthandlers.**BaseHandler**(*application*, *request*, ***kwargs*)
Bases: tornado.web.RequestHandler

BaseHandler for all other RequestHandlers

db_conn
Returns database connection abstraction

If no database connection is available, raises an AttributeError

class tornado_json.requesthandlers.**ViewHandler**(*application*, *request*, ***kwargs*)
Bases: *tornado_json.requesthandlers.BaseHandler*

Handler for views

initialize()

•Set Content-type for HTML

7.5 routes Module

tornado_json.routes.**gen_submodule_names**(*package*)
Walk package and yield names of all submodules

Parameters **package** (*package*) – The package to get submodule names of

Returns Iterator that yields names of all submodules of package

Return type Iterator that yields str

tornado_json.routes.**get_module_routes**(*module_name*, *custom_routes=None*, *exclusions=None*, *arg_pattern='(?P<{}>[a-zA-Z0-9_\\-]+)'*)
Create and return routes for module_name

Routes are (url, RequestHandler) tuples

Returns list of routes for module_name with respect to exclusions and custom_routes. Returned routes are with URLs formatted such that they are forward-slash-separated by module/class level and end with the lowercase name of the RequestHandler (it will also remove 'handler' from the end of the name of the handler). For example, a requesthandler with the name helloworld.api.HelloWorldHandler would be assigned the url /api/helloworld. Additionally, if a method has extra arguments aside from self in its signature, routes with URL patterns will be generated to match r"(?P<{}>[a-zA-Z0-9_\-]+)".format(argname) for each argument. The aforementioned regex will match ONLY values with alphanumeric, hyphen and underscore characters. You can provide your own pattern by setting a arg_pattern param.

Return type [(url, RequestHandler), ...]

Parameters

- **module_name** (*str*) – Name of the module to get routes for
- **custom_routes** (*[(str, RequestHandler), ...]*) – List of routes that have custom URLs and therefore should be automagically generated
- **exclusions** (*[str, str, ...]*) – List of RequestHandler names that routes should not be generated for
- **arg_pattern** (*str*) – Default pattern for extra arguments of any method

tornado_json.routes.**get_routes**(*package*)

This will walk `package` and generates routes from any and all `APIHandler` and `ViewHandler` subclasses it finds. If you need to customize or remove any routes, you can do so to the list of returned routes that this generates.

Parameters **package** (*package*) – The package containing RequestHandlers to generate routes from

Returns List of routes for all submodules of `package`

Return type [(url, RequestHandler), ...]

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

t