aiobravado

Jun 19, 2018

Contents

Aiobravado is a python client library for Swagger 2.0 services, using an asynchronous programming model.

More information on Swagger can be found on the Swagger website

It aims to be a complete replacement to swagger codegen.

It is a fork of the bravado library, and is maintained by the same people that maintain bravado.

Features include:

- Dynamically generated client - no code generation needed!
- Strict validations to verify that your Swagger Schema is v2.0 compatible.
- HTTP request and response validation against your Swagger Schema.
- Swagger models as Python types (no need to deal with JSON).
- REPL friendly navigation of your Swagger schema with docstrings for Resources, Operations and Models.
- Ingestion of your Swagger schema via http or a local file path.

Contents:

CHAPTER 1

Quickstart

1.1 Usage

Install the latest stable version from PyPi:

```
$ pip install --upgrade aiobravado
```

1.2 Your first Hello World! (or Hello Pet)

Here is a simple example to try from a REPL (like IPython):

```
from aiobravado.client import SwaggerClient

client = await SwaggerClient.from_url("http://petstore.swagger.io/v2/swagger.json")
pet = await client.pet.getPetById(petId=42).result()
```

If you were lucky, and pet Id with 42 was present, you will get back a result. It will be a dynamically created instance of `aiobravado.model.Pet` with attributes `category`, etc. You can even try `pet.category.id` or `pet.tags[0]`.

Sample Response:

```
Pet(category=Category(id=0L, name=u''), status=u'', name=u'', tags=[Tag(id=0L, name=u'
↪')], photoUrls=[u''], id=2)
```

If you got a `404`, try some other petId.

1.3 Lets try a POST call

Here we will demonstrate how `aiobravado` hides all the `JSON` handling from the user, and makes the code more Pythonic.

```
Pet = client.get_model('Pet')
Category = client.get_model('Category')
pet = Pet(id=42, name="tommy", category=Category(id=24))
await client.pet.addPet(body=pet).result()
```

1.4 This is too fancy for me! I want a simple dict response!

`aiobravado` has taken care of that as well. Configure the client to not use models.

```
from aiobravado.client import SwaggerClient
from aiobravado.fido_client import FidoClient

client = await SwaggerClient.from_url(
    'http://petstore.swagger.io/v2/swagger.json',
    config={'use_models': False}
)

result = await client.pet.getPetById(petId=42).result(timeout=4)
```

`result` will look something like:

```
{
    'category': {
        'id': 0L,
        'name': u''
    },
    'id': 2,
    'name': u'',
    'photoUrls': [u''],
    'status': u'',
    'tags': [
        {'id': 0L, 'name': u''}
    ]
}
```

CHAPTER 2

Configuration

2.1 Client Configuration

You can configure certain behaviours when creating a `SwaggerClient`.

aiobravado and bravado-core use the same config dict. The full documentation for bravado-core config keys is available too.

```
from aiobravado.client import SwaggerClient, SwaggerFormat

my_super_duper_format = SwaggerFormat(...)

config = {
    # === aiobravado config ===

    # Determines what is returned by the service call.
    'also_return_response': False,

    # === bravado-core config ====

    #  validate incoming responses
    'validate_responses': True,

    # validate outgoing requests
    'validate_requests': True,

    # validate the swagger spec
    'validate_swagger_spec': True,

    # Use models (Python classes) instead of dicts for #/definitions/{models}
    'use_models': True,

    # List of user-defined formats
    'formats': [my_super_duper_format],
```

(continues on next page)

(continued from previous page)

```
}

client = SwaggerClient.from_url(..., config=config)
```

Config key	Type	Default	Description
also_return_response	boolean	False	Determines what is returned by the service call. Specifically, the return value of `HttpFuture.result()`. When `False`, the swagger result is returned. When `True`, the tuple `(swagger result, http response)` is returned. See *Getting access to the HTTP response*.

2.2 Per-request Configuration

Configuration can also be applied on a per-request basis by passing in `_request_options` to the service call.

```
client = SwaggerClient.from_url(...)
request_options = { ... }
client.pet.getPetById(petId=42, _request_options=request_options).result()
```

Config key	Type	Default	Description
connect_timeout	float	N/A	TCP connect timeout in seconds. This is passed along to the http_client when making a service call.
headers	dict	N/A	Dict of http headers to to send with the outgoing request.
response_callbacks	list of callables	[]	List of callables that are invoked after the incoming response has been validated and unmarshalled but before being returned to the calling client. This is useful for client decorators that would like to hook into the post-receive event. The callables are executed in the order they appear in the list. Two parameters are passed to each callable: - `incoming_response` of type `bravado_core.response.IncomingResponse` - `operation` of type `bravado_core.operation.Operation`
timeout	float	N/A	TCP idle timeout in seconds. This is passed along to the http_client when making a service call.
use_msgpack	boolean	False	If a msgpack serialization is desired for the response. This

CHAPTER 3

Advanced Usage

3.1 Validations

`aiobravado` validates the schema against the Swagger 2.0 Spec. Validations are also done on the requests and the responses.

Validation example:

```
pet = Pet(id="I should be integer :(", name="tommy")
await client.pet.addPet(body=pet).result()
```

will result in an error like so:

```
TypeError: id's value: 'I should be integer :(' should be in types (<type 'long'>,
↪<type 'int'>)
```

Note: If you'd like to disable validation of outgoing requests, you can set `validate_requests` to `False` in the `config` passed to `SwaggerClient.from_url(...)`.

The same holds true for incoming responses with the `validate_responses` config option.

3.2 Adding Request Headers

`aiobravado` allows you to pass request headers along with any request.

```
Pet = client.get_model('Pet')
Category = client.get_model('Category')
pet = Pet(id=42, name="tommy", category=Category(id=24))
await swagger_client.pet.addPet(
    body=pet,
```

(continues on next page)

(continued from previous page)

```
    _request_options={"headers": {"foo": "bar"}},
).result()
```

3.3 Docstrings

`aiobravado` provides docstrings to operations and models to quickly get the parameter and response types. Due to an implementation limitation, an operation's docstring looks like a class docstring instead of a function docstring. However, the most useful information about parameters and return type is present in the `Docstring` section.

Note: The `help` built-in does not work as expected for docstrings. Use the `?` method instead.

```
>> petstore.pet.getPetById?

Type:       CallableOperation
String Form:<aiobravado.client.CallableOperation object at 0x241b5d0>
File:       /some/dir/aiobravado/bravado/client.py
Definition: c.pet.getPetById(self, **op_kwargs)
Docstring:
[GET] Find pet by ID

Returns a single pet

:param petId: ID of pet to return
:type petId: integer
:returns: 200: successful operation
:rtype: object
:returns: 400: Invalid ID supplied
:returns: 404: Pet not found
Constructor Docstring::type operation: :class:`bravado_core.operation.Operation`
Call def:   c.pet.getPetById(self, **op_kwargs)
Call docstring:
Invoke the actual HTTP request and return a future that encapsulates
the HTTP response.

:rtype: :class:`aiobravado.http_future.HTTPFuture`
```

Docstrings for models can be retrieved as expected:

```
>> pet_model = petstore.get_model('Pet')
>> pet_model?

Type:       type
String Form:<class 'bravado_core.model.Pet'>
File:       /some/dir/bravado_core/model.py
Docstring:
Attributes:

category: Category
id: integer
name: string
photoUrls: list of string
status: string - pet status in the store
```

(continues on next page)

(continued from previous page)

```
tags: list of Tag
Constructor information:
 Definition:pet_type(self, **kwargs)
```

3.4 Default Values

`aiobravado` uses the default values from the spec if the value is not provided in the request.

In the Pet Store example, operation `findPetsByStatus` has a `default` of `available`. That means, `aiobravado` will plug that value in if no value is provided for the parameter.

```
client.pet.findPetByStatus()
```

3.5 Loading swagger.json by file path

`aiobravado` also accepts `swagger.json` from a file path. Like so:

```
client = await SwaggerClient.from_url('file:///some/path/swagger.json')
```

Alternatively, you can also use the `load_file` helper method.

```
from aiobravado.swagger_model import load_file

client = await SwaggerClient.from_spec(await load_file('/path/to/swagger.json'))
```

3.6 Getting access to the HTTP response

The default behavior for a service call is to return the swagger result like so:

```
pet = await petstore.pet.getPetById(petId=42).result()
print pet.name
```

However, there are times when it is necessary to have access to the actual HTTP response so that the HTTP headers or HTTP status code can be used. This is easily done via configuration to return a `(swagger result, http response)` tuple from the service call.

```
petstore = await Swagger.from_url(..., config={'also_return_response': True})
pet, http_response = await petstore.pet.getPetById(petId=42).result()
assert isinstance(http_response, bravado_core.response.IncomingResponse)
print(http_response.headers)
print(http_response.status_code)
print(pet.name)
```

CHAPTER 4

Changelog

4.1 0.9.3 (2018-06-19)

- aiobravado doesn't work with the latest bravado-asyncio version, let's add a restriction to setup.py - PR #6.

4.2 0.9.2 (2018-02-08)

- Don't ship our own HTTP client, use bravado-asyncio - PR #2.

4.3 0.9.1 (2018-01-11)

- bravado version: 9.2.2
- Fix unmarshalling of non-JSON/msgpack responses - PR #1.

4.4 0.9.0 (2017-11-17)

- Initial release, based on the source code of bravado 9.2.0.

CHAPTER 5

Indices and tables

- genindex
- modindex