pyswagger Documentation

Release 0.8.27

Mission Liao

February 14, 2017

Contents

pyswagger is a type-safe, dynamic, spec-complaint Swagger client.

- *type-safe*: Instead of manipulating json files as objects directly, we load them and produce our own object set. Every object in Swagger spec has a correspondence in pyswagger. During construction of those objects, rules of Swagger Spec would be checked.
- *dynamic*: Unlike swagger-codegen, pyswagger doesn't need preprocessing.
- *spec-complaint*: pyswagger support Swagger 1.2.

The main idea of pyswagger is to provide something easier to use than raw json, and develop things around that.

CHAPTER 1

Contents

1.1 Main Components

App, *Security*, *Client* are components you would touch first when adapting pyswagger.

1.1.1 App

App carries Swagger API definition, other components would rely on it but not json files. You need to access Operation object via **App.op** by providing *nickname* or *resource name* plus *nickname*.

```
app = App._create_('http://petstore.swagger.wordnik.com/api/api-docs')
assert app.op['getPetsByStatus'] == app.op['pet', 'getPetsByStatus']

# resource name is required when nicknames collid
app.op['user', 'getById']
app.op['pet', 'getById']
```

The Operation object is callable, and can be provided by a set of *Primitives*, then return a pair of *Request* and *Response*.

1.1.2 Security

Security is a placeholder of authorizations,

```
# must be initialized with App
auth = Security(app)

# insert autorization information
app.update_with('simple_basicAuth', ('user', 'password'))
app.update_with('simple_apiKey', 'token123')
app.update_with('simple_oath2', 'token123456')

# pass into a client
client = TornadoClient(auth)

# authorization would be applied automatically
client.request(...)
```

1.1.3 Client

Clients are wrapper layer to hide implementation details from different http-request libraries.

To implement a customized client, please refer to *customized client*

Below is a code to demostrate the relation between these components.

```
app = App._create_('http://petstore.swagger.wordnik.com/api/api-docs')
auth = Security(app)
client = Client(auth)

# get Request and Response from Swagger.op
req, resp = app.op['getPetById'](Id=1)

# call request
resp = client.request((req, resp))

# get data back
assert resp.data.id == 1
```

1.1.4 Reference

class pyswagger.core.**App**(*url=None*, *url_load_hook=None*, *sep='!##!'*, *prim=None*, *mime_codec=None*, *resolver=None*)

Major component of pyswagger

This object is tended to be used in read-only manner. Therefore, all accessible attributes are almost read-only properties.

classmethod _create_(*kls*, *url*, *strict=True*)

factory of App

Parameters

- **url** (*str*) – url of path of Swagger API definition
- **strict** (*bool*) – when in strict mode, exception would be raised if not valid.

Returns the created App object

Return type *App*

Raises

- **ValueError** – if url is wrong
- **NotImplementedError** – the swagger version is not supported.

_validate()

check if this Swagger API valid or not.

Parameters **strict** (*bool*) – when in strict mode, exception would be raised if not valid.

Returns validation errors

Return type list of tuple(where, type, msg).

classmethod create(*kls*, *url*, *strict=True*)

factory of App

Parameters

- **url** (*str*) – url of path of Swagger API definition

- **strict** (*bool*) – when in strict mode, exception would be raised if not valid.

Returns the created App object

Return type *App*

Raises

- **ValueError** – if url is wrong
- **NotImplementedError** – the swagger version is not supported.

dump()
dump into Swagger Document

Return type dict

Returns dict representation of Swagger

classmethod load(*kls*, *url*, *getter=None*, *parser=None*, *url_load_hook=None*, *sep='!##!'*, *prim=None*, *mime_codec=None*, *resolver=None*)
load json as a raw App

Parameters

- **url** (*str*) – url of path of Swagger API definition
- **getter** (*sub class/instance of Getter*) – customized Getter
- **parser** (*pyswagger.base.Context*) – the parser to parse the loaded json.
- **app_cache** (*dict*) – the cache shared by related App
- **url_load_hook** (*func*) – hook to patch the url to load json
- **sep** (*str*) – scope-separater used in this App
- **pyswager.primitives.Primitive** (*prim*) – factory for primitives in Swagger
- **pyswagger.primitives.MimeCodec** (mime_codec) – MIME codec
- **resolver** – pyswagger.resolve.Resolver: customized resolver used as default when none is provided when resolving

Returns the created App object

Return type *App*

Raises

- **ValueError** – if url is wrong
- **NotImplementedError** – the swagger version is not supported.

load_obj(*jref*, *getter=None*, *parser=None*)
load a object(those in spec._version_.objects) from a JSON reference.

m
backward compatible to access Swagger.definitions in Swagger 2.0, and Resource.Model in Swagger 1.2.

ex. a Model:user in Resource:Users, access it by .m['Users', 'user']. For Schema object in Swagger 2.0, just access it by it key in json.

Type pyswagger.utils.ScopeDict

mime_codec
mime codec used by this app

Type pyswagger.primitives.MimeCodec

op
list of Operations, organized by utils.ScopeDict

In Swagger 2.0, Operation(s) can be organized with Tags and Operation.operationId. ex. if there is an operation with tag:['user', 'security'] and operationId:get_one, here is the combination of keys to access it: - .op['user', 'get_one'] - .op['security', 'get_one'] - .op['get_one']

Type pyswagger.utils.ScopeDict of pyswagger.spec.v2_0.objects.Operation

prepare(*strict=True*)
preparation for loaded json

Parameters **strict** (*bool*) – when in strict mode, exception would be raised if not valid.

prepare_obj(*obj*, *jref*)
basic preparation of an object(those in sepc._version_.objects), and cache the 'prepared' object.

prim_factory
primitive factory used by this app

Type pyswagger.primitives.Primitive

raw
raw objects for original version of loaded resources. When loaded json is the latest version we supported, this property is the same as App.root

Type ex. when loading Swagger 1.2, the type is pyswagger.spec.v1_2.objects.ResourceList

resolve(*jref*, *parser=None*)
JSON reference resolver

Parameters

- **jref** (*str*) – a JSON Reference, refer to http://tools.ietf.org/html/draft-pbryan-zyp-json-ref-03 for details.
- **parser** (*pyswagger.base.Context*) – the parser corresponding to target object.

Returns the referenced object, wrapped by weakref.ProxyType

Return type weakref.ProxyType

Raises **ValueError** – if path is not valid

root
schema representation of Swagger API, its structure may be different from different version of Swagger.

There is 'Schema' object in swagger 2.0, that's why I change this property name from 'schema' to 'root'.

Type pyswagger.spec.v2_0.objects.Swagger

s(*p*, *b=('/', '#/paths')*)
shortcut of App.resolve. We provide a default base for '#/paths'. ex. to access '#/paths/~1user/get', just call App.s('user/get')

Parameters

- **p** (*str*) – path relative to base
- **b** (*tuple*) – a tuple (expected_prefix, base) to represent a 'base'

schemes
supported schemes, refer to Swagger.schemes in Swagger 2.0 for details

Type list of str, ex. ['http', 'https']

url

validate(*strict=True*)
check if this Swagger API valid or not.

Parameters **strict** (*bool*) – when in strict mode, exception would be raised if not valid.

Returns validation errors

Return type list of tuple(where, type, msg).

version
original version of loaded json

Type str

class pyswagger.core.**Security**(*app*)
security handler

__init__(*app*)
constructor

Parameters **app** (App) – App

__call__(*req*)
apply security info for a request.

Parameters **req** (Request) – the request to be authorized.

Returns the updated request

Return type *Request*

update_with(*name*, *security_info*)
insert/clear authorizations

Parameters

- **name** (*str*) – name of the security info to be updated
- **security_info** (**(username, password)** for *basicAuth*, **token** in str for *oauth2*, *apiKey*.) – the real security data, token, ...etc.

Raises **ValueError** – unsupported types of authorizations

class pyswagger.contrib.client.requests.**Client**(*auth=None*, *send_opt=None*)
Client implementation based on requests

class pyswagger.contrib.client.tornado.**TornadoClient**(*auth=None*)
Client implementation based on tornado.http.AsyncHTTPClient.

Request

class pyswagger.io.**Request**(*op*, *params*)
Request layer

base_path
base path of this request

Type str

data
data carried by this request, only valid after 'prepare'

Type byte

files
files of this Request

Type dict of (name, primitives.File)

header
header of this request, only valid after 'prepare'

Type dict

method
HTTP verb of this request

Type str

path
path of this request, only valid after 'prepare'

Type str

prepare(*scheme='http'*, *handle_files=True*, *encoding='utf-8'*)
make this request ready for Clients

Parameters

- **scheme** (*str*) – scheme used in this request
- **handle_files** (*bool*) – False to skip multipart/form-data encoding
- **encoding** (*str*) – encoding for body content.

Return type *Request*

query
query part of this request

Type dict

scheme
preferred scheme used in this request

schemes
required schemes for current Operation.

Type list of str

url
url of this request, only valid after 'prepare'

Type str

Response

class pyswagger.io.**Response**(*op*)
Response layer

apply_with(*status=None*, *raw=None*, *header=None*)
update header, status code, raw datum, ...etc

Parameters

- **status** (*int*) – status code
- **raw** (*str*) – body content
- **header** (*dict*) – header section

Returns return self for chaining

Return type *Response*

data
responsed data

Type primitives.Model

header
header of Response

Type dict of list, ex. {'Content-Type': [xxx, xxx]}

raw
raw response

Type str

status
status code

Type int

1.2 Primitives

class pyswagger.primitives.**Byte**
for string type, byte format

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

apply_with(_, *v*, *ctx*)
constructor

Parameters **v** (*str*) – accept six.string_types, six.binary_type

to_json()
according to https://github.com/wordnik/swagger-spec/issues/50, we should exchange 'byte' type via base64 encoding.

class pyswagger.primitives.**Date**
for string type, date format

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

apply_with(_, *v*, *ctx*)
constructor

Parameters **v** (*timestamp in float, datetime.date object, or ISO-8601 in str*) – things used to constrcut date

class pyswagger.primitives.**Datetime**
for string type, datetime format

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

apply_with(_, *v*, *ctx*)
constructor

Parameters **v** (*timestamp in float, datetime.datetime object, or ISO-8601 in str*) – things used to constrcut date

class pyswagger.primitives.**Array**
for array type, or parameter when allowMultiple=True

__init__()
v: list or string_types

__str__()
array primitives should be for 'path', 'header', 'query'. Therefore, this kind of convertion is reasonable.

Returns the converted string

Return type str

apply_with(*obj*, *val*, *ctx*)

to_url()
special function for handling 'multi', refer to Swagger 2.0, Parameter Object, collectionFormat

class pyswagger.primitives.**Model**
for complex type: models

__init__()
constructor

__eq__(*other*)
equality operater, will skip checking when both value are None or no attribute.

Parameters **other** (*primitives.Model or dict*) – another model

__ne__(*other*)

apply_with(*obj*, *val*, *ctx*)
recursivly apply Schema object

Parameters

- **obj** (*obj.Model*) – model object to instruct how to create this model
- **val** (*dict*) – things used to construct this model

cleanup(*val*, *ctx*)

1.3 Extend pyswagger

This section describes things about extending pyswagger.

1.3.1 A new client implementation

To implement a customized client, you need to aware *BaseClient*

class pyswagger.core.**BaseClient**(*security=None*)
base implementation of SwaggerClient, below is an minimum example to extend this class

```
class MyClient(BaseClient):

    # declare supported schemes here
    __schemes__ = ['http', 'https']
```

```
        def request(self, req_and_resp, opt):
            # passing to parent for default patching behavior,
            # applying authorizations, ...etc.
            req, resp = super(MyClient, self).request(req_and_resp, opt)

            # perform request by req
            ...
            # apply result to resp
            resp.apply(header=header, raw=data_received, status=code)
            return resp
```

__init__(*security=None*)
constructor

Parameters **security** (Security) – the security holder

__weakref__
list of weak references to the object (if defined)

prepare_schemes(*req*)
make sure this client support schemes required by current request

Parameters **req** (pyswagger.io.Request) – current request object

request(*req_and_resp*, *opt*)
preprocess before performing a request, usually some patching. authorization also applied here.

Parameters **req_and_resp** (*(Request, Response)*) – tuple of Request and Response

Returns patched request and response

Return type Request, Response

CHAPTER 2

Getting Started

```
    from pyswagger import App, Security
    from pyswagger.contrib.client.requests import Client

    # load Swagger resource file into App object
    app = App._create_('http://petstore.swagger.wordnik.com/api/api-docs')

    # init Security for authorization
    auth = Security(app)
    auth.update_with('simple_basic_auth', ('user', 'password')) # basic auth
    auth.update_with('simple_api_key', '12312312312312312313q') # api key
    auth.update_with('simple_oauth2', '12334546556521123fsfss') # oauth2

    # init swagger client
    client = Client(auth)

    # a request to create a new pet
    pet_Tom=dict(id=1, name='Tom') # a dict is enough
    client.request(app.op['addPet'](body=pet_Tom))

    # a request to get the pet back
    pet = client.request(app.op['getPetById'])(petId=1).data
    assert pet.id == 1
    assert pet.name == 'Tom'

    # redirect all requests targeting 'petstore.swagger.wordnik.com'
    # to 'localhost:9001' for testing locally
    client.request(
        app.op['addPet'](body=pet_Tom),
        opt={'url_netloc': 'localhost:9001'}
        )

    # allowMultiple parameter
    client.request(app.op['getPetsByStatus'](status='sold')) # one value
    client.request(app.op['getPetsByStatus'](status=['available', 'sold'])) # multiple value, wrappe
```

Source code Report issues

p

Symbols

A

B

C

D

F

H

L

M

O

P

Q

R

S

T

U

V