instahashtag

Felipe Faria

Apr 01, 2021

CONTENTS

1 wrapper **1**

1.1 tag 1

1.2 graph 3

1.3 maps 5

2 api **7**

3 http **10**

Python Module Index **12**

Index **13**

CHAPTER

ONE

WRAPPER

```
from instahashtag import Tag, Graph, Maps
```

Complete object-oriented wrappers over the API. Allows quering the server and accessing the return information directly via the Python objects.

Note: For all of the classes below if the `aio` flag is `False` then the object will automatically query the API without needing further action. However, if the `aio` flag is set to `True` one must `await` the `call()` function associated with the object to make the query to the server.

```
from instahashtag import Maps
import asyncio

async def main():
    tag = Maps(
        x1=-80.48712034709753,
        y1=25.750749758162012,
        x2=-79.82794065959753,
        y2=25.854604964203453,
        zoom=12,
        aio=True
    )
    await tag.call()
```

1.1 tag

```
from instahashtag import Tag
```

Python wrapper for the *Tag* API call.

class Tag

__init__(*self*, *hashtag: str*, *aio: bool = False*) → None
Initializes a new Tag object.

```
from instahashtag import Tag

tag = Tag("miami")
```

hashtag
Hashtag that was used to retrieve information from.

```
tag.hashtag # >>> miami
```

geo
List containing two float coordinates relating to the hashtag.

```
tag.geo # >>> [25.821117872941034, -80.20722606661316]
```

rank
Integer representation of the rank of the hashtag relative to others.

```
tag.rank # >>> 83
```

exists
Bool representation of whether or not the passed hashtag exists.

```
tag.exists # >>> True
```

results
List of `TagResult` containing information on related hashtags.

```
tag.results # >>> [Result(tag=..., rank=..., geo=[...,...], media_count=..
↪., relevance=..., abs_relevance=...), ...]
result = tag.results[0]
```

class TagResult(*tag: str*, *rank: int*, *geo: List[float]*, *media_count: int*, *relevance: int*, *absRelevance: float*)
Object that represents the individual items inside the `Tag.results` list.

tag
Related hashtag.

```
result.tag # >>> miamibeach
```

rank
Ranking of the tag.

```
result.rank # >>> 74
```

geo
List of floats that contains coordinates to the hashtag.

```
result.geo # >>> [25.819434533299013, -80.16981253812398]
```

media_count
Number of posts in the hashtag.

```
result.media_count # >>> 4329283
```

relevance
Relevance of the hashtag to the queried hashtag.

```
result.relevance # >>> 99
```

absRelevance
Absolute relevance to the queried hashtag.

```
result.absRelevance # >>> 0.0060874452062492975
```

__gt__(*self*, *other: Result*) → bool

Allows comparisson of `Result` objects by their `rank`.

Note: This function allows the user to utilize the Python built-in functions.

```
from instahashtag import Tag

tag = Tag("miami")

min(tag.results)
max(tag.results)
sort(tag.results)
```

async call(*self*) → None

Asynchronously queries the API.

See note on the top of the *wrapper* documentation.

1.2 graph

```
from instahashtag import Graph
```

Python wrapper for the *Graph* API call.

class Graph

__init__(*self*, *hashtag: str*, *aio: bool = False*) → None

Initializes a new Graph object.

```
from instahashtag import Graph

graph = Graph("miami")
```

hashtag

Hashtag that was used to retrieve information from.

```
graph.hashtag # >>> miami
```

exists

Boolean that dictates whether or not the passed hashtag exists.

```
graph.exists # >>> True
```

root_pos

List of floats indicating where the root position of the graph is.

```
graph.root_post # >>> [0.4495344797287565, 0.40752168227901403]
```

nodes

List of `GraphNode` containing information on related hashtags.

```
graph.nodes # >>> [Node(id=..., relevance=..., weight=..., x=..., y=...),␣
↪...]
node = graph.nodes[0]
```

class GraphNode(*id: str*, *relevance: float*, *weight: float*, *x: float*, *y: float*)
Object that represents the individual edges inside the `Graph.nodes` list.

id
Hashtag name.

```
node.name # >>> liv
```

relevance
Relevance of the node.

```
node.relevance # >>> 0.5417327185029007
```

weight
Weight of the node.

```
node.weight # >>> 0.4523809523809524
```

x
Position of the node in relation to the x-axis.

```
node.x # >>> 0.20431805750484297
```

y
Position of the node in relation to the y-axis.

```
node.y # >>> 0.6194782200730474
```

edges
List of `GraphEdge` containing information on the connection between hashtags.

```
graph.edges # >>> [Edge(a=..., b=..., id=...#..., weight=...), ...]
edge = graph.edges[0]
```

class GraphEdge(*a: str*, *b: str*, *id: str*, *weight: int*)
Object that represents the individual edges inside the `Graph.edges` list.

a
Hashtag name that represents a node in the graph.

```
edge.a # >>> liv
```

b
Hashtag name that represents another node in the graph.

```
edge.a # >>> thingstodomiami
```

id
Order of the connection between nodes.

```
edge.id # >>> liv#thingstodomiami
```

weight
Weight of the edge.

```
edge.weight # >>> 0.44775669978922017
```

Note: The edge object of the graph represents *which* node is connected to which other node.

async call(*self*) → None

Asynchronously queries the API.

See note on the top of the *wrapper* documentation.

1.3 maps

```
from instahashtag import Maps
```

Python wrapper for the *Maps* API call.

class Maps

__init__(*self*, *x1: float*, *y1: float*, *x2: float*, *y2: float*, *zoom: int*, *aio: bool = False*) → None

Initializes a new map object.

```
from instahashtag import Maps

# Coordinates designate to Miami, FL.
maps = Maps(
    x1=-80.48712034709753,
    y1=25.750749758162012,
    x2=-79.82794065959753,
    y2=25.854604964203453,
    zoom=12
)
```

x1

Top left x-coordinate corner of the map.

```
maps.x1 # >>> -80.48712034709753
```

y1

Top left y-coordinate corner of the map.

```
maps.y1 # >>> 25.750749758162012
```

x2

Bottom right x-coordinate corner of the map.

```
maps.x2 # >>> -79.82794065959753
```

y2

Bottom right y-coordinate corner of the map.

```
maps.y2 # >>> 25.854604964203453
```

zoom

Number from 2 to 16 that designates the zoom factor.

```
maps.zoom # >>> 12
```

count
Number of hashtags in the resulting query.

```
maps.count # >>> 91
```

tags
List of hashtags.

```
maps.tags # >>> [MapTag(tag=..., centroid=[..., ...], weight=...), ...]
tag = maps.tags[0]
```

class MapsTag(*centroid: List[float]*, *tag: str*, *weight: int*)
Object that represents the individiual items inside the `Maps.tags` list.

centroid
List of floats that contains the location of the tag in the map (lon, lat).

```
tag.centroid # >>> [25.801775593361942, -80.20252247848369]
```

tag
Hashtag name.

```
tag.tag # >>> igersmiami
```

weight
Weight of the hashtag on the map.

```
tag.weight # >>> 49
```

__gt__(*self*, *other: Result*) → bool
Allows comparisson of `MapsTags` objects by their `weight`.

Note: This function allows the user to utilize the Python built-in functions.

```
from instahashtag import Maps

# Coordinates designate to Miami, FL.
maps = Maps(
    x1=-80.48712034709753,
    y1=25.750749758162012,
    x2=-79.82794065959753,
    y2=25.854604964203453,
    zoom=12
)

min(maps.tags)
max(maps.tags)
sort(maps.tags)
```

async call(*self*) → None
Asynchronously queries the API.

See note on the top of the *wrapper* documentation.

CHAPTER

TWO

API

```
from instahashtag import api
```

Direct API call to DisplayPurposes with `json` returns. Support for synchronous and asynchronous function call.

Note: This module can either be used synchronous or asynchronous via the `aio` flag passed to each of the functions below. By default `aio` is set to `False`, but when setting it to `True` the return will be an `Awaitable` that can be `await` to query the API.

Example One may use the `aio` flag to dictate whether to use the function asynchronously or synchronously.

```
from instahashtag import api

def io():
    """Uses 'requests' to send requests."""

    tag = api.tag(hashtag="instagram")
    graph = api.tag(hashtag="instagram")
    maps = api.maps(
        x1=-80.48712034709753,
        y1=25.750749758162012,
        x2=-79.82794065959753,
        y2=25.854604964203453,
        zoom=12,
    )

async def aio():
    """Uses 'aiohttp' to send requests."""

    tag = await api.tag(hashtag="instagram", aio=True)
    graph = await api.tag(hashtag="instagram", aio=True)
    maps = await api.maps(
        x1=-80.48712034709753,
        y1=25.750749758162012,
        x2=-79.82794065959753,
        y2=25.854604964203453,
        zoom=12,
    )
```

`tag`(*hashtag: str*, *aio: bool = False*) → Union[dict, Awaitable]
Sends a request to the server.

Parameters

- **`hashtag`** – Hashtag to retrieve info from.
- **`aio`** – If set to `True` will return a Future for use with `await` keyword. Defaults to `False`.

Returns Dictionary with given data (see Notes below).

Note: The return dictionary follows the following format:

```
{
    geo: [float, float],
    rank: int,
    results: [
        {
            absRelevance: float,
            geo: [float, float],
            media_count: int,
            rank: int,
            relevance: int,
            tag: str
        }
        ...
    ],
    tag: str,
    tagExists: bool
}
```

`graph` (*hashtag: str*, *aio: bool = False*) → Union[dict, Awaitable]

Generates graphing query to send to the server.

Parameters

- **`hashtag`** – Hashtag to retrieve info from.
- **`aio`** – If set to `True` will return a Future for use with `await` keyword. Defaults to `False`.

Returns Dictionary with given data (see Notes below).

Note: The return dictionary follows the following format:

```
{
    edges: [
        {
            a: str,
            b: str,
            id: str,
            weight: float
        }
        ...
    ],
    exists: bool,
    nodes: [
        id: str,
        relevance: float,
        weight: float,
        x: float,
        y: float
```

(continues on next page)

(continued from previous page)

```
    ],
    query: string,
    root_pos: [float, float]
}
```

maps (*x1: float*, *y1: float*, *x2: float*, *y2: float*, *zoom: float = 1*, *aio: bool = False*) → Union[dict, Awaitable]

Generates graphing query to send to the server.

Parameters

- **x1** – Top left x-coordinate corner of the map.
- **y1** – Top left y-coordinate corner of the map.
- **x2** – Bottom right x-coordinate corner of the map.
- **y2** – Bottom right y-coordinate corner of the map.
- **zoom** – Number from 2 to 16 that designates the zoom factor.
- **aio** – If set to `True` will return a Future for use with `await` keyword. Defaults to `False`.

Returns Dictionary with given data (see Notes below).

Note: The return dictionary follows the following format:

```
{
    count: int
    tags: [
        {
            "centroid": [
                0: float
                1: float
            ],
            tag: str,
            weight: int
        }
        ...
    ]
}
```

CHAPTER

THREE

HTTP

```
from instahashtag.http import Base, Requests, Aiohttp
```

Low-level HTTP calls to DisplayPurposes with `json` returns. Allows customization with "plug-and-play" support for any other Python HTTP request library. Comes out of the box with support for both the `requests` and `aiohttp` packages.

Note: The classes documented here are not intended to be used for querying the API- they are here to allow customization support for anyone looking to use their own HTTP request library to send requests to the API.

If you are looking for retrieving back the raw `json` objects without carying much about implementation, check out the documentations for the *api* module.

class endpoints

API endpoints.

```
class endpoints:
    tag = "https://apidisplaypurposes.com/tag/{}"
    graph = "https://apidisplaypurposes.com/graph/{}"
    maps = "https://apidisplaypurposes.com/local/?bbox={},{},{},{}&zoom={}"
```

class Base(*endpoint: str*, *headers: dict*)

Base class that properly processes and calls the API.

Allows one to "plug-and-play" with other request libraries with ease. If one wants to, they may inherit from this class and overwrite the abstract `call()` method.

Example While the module `httpx` is not supported out of the box, one may inherit from the `Base` class and utilize it.

```
from instahashtag.http import Base
import httpx

class httpx_io(Base):
    def call(self):
        req = httpx.get(self.endpoint, headers=self.headers)
        resp = req.text

        # Note that self.process is just a simple wrapper for 'json.loads'.
        # One has the option to also overwrite this to catch any extra␣
↪error(s),
        # or run processes on the data.
```

(continues on next page)

(continued from previous page)

```
        return self.process(resp)

class httpx_aio(Base):
    async def call(self):
        async with httpx.AsyncClient() as client:
            req = await client.get(self.endpoint, headers=self.headers)

        resp = req.text
        return self.process(resp)

def io():
    tag = httpx_io.tag(hashtag="instagram")
    graph = httpx_io.graph(hashtag="instagram")
    maps = httpx_io.maps(
        x1=-80.48712034709753,
        y1=25.750749758162012,
        x2=-79.82794065959753,
        y2=25.854604964203453,
        zoom=12,
    )

def aio():
    tag = await httpx_io.tag(hashtag="instagram")
    graph = await httpx_io.graph(hashtag="instagram")
    maps = await httpx_io.maps(
        x1=-80.48712034709753,
        y1=25.750749758162012,
        x2=-79.82794065959753,
        y2=25.854604964203453,
        zoom=12,
    )
```

abstract call() → NotImplemented
Abstract method that needs to be written to query the API endpoint.

static process(*resp: str*) → dict
Processes the reply returned back by *tag*, *graph*, and *maps*.

The current implementation of this function is a simple `json.loads(resp)`, returning back a Python dictionary object. However, one may chose to overwrite this function to process the data in some more meaningful way.

classmethod tag(*hashtag: str*) → Any
Sends an API request to the `tag` endpoint.

classmethod graph(*hashtag: str*) → Any
Sends an API request to the `graph` endpoint.

classmethod maps(*x1: float*, *y1: float*, *x2: float*, *y2: float*, *zoom: int*) → Any
Sends an API request to the `maps` endpoint.

The classes below are the out-of-the-box implementations used by the higher layers.

class Requests(*endpoint: str*, *headers: dict*)
Class that utilizes the `request` module to make API request.

class Aiohttp(*endpoint: str*, *headers: dict*)
Class that utilizes the `aiohttp` module to make API request.

PYTHON MODULE INDEX

i

instahashtag.api, 7

INDEX

A

a (*Graph.GraphEdge attribute*), 4
absRelevance (*Tag.TagResult attribute*), 2
Aiohttp (*class in instahashtag.http*), 11

B

b (*Graph.GraphEdge attribute*), 4
Base (*class in instahashtag.http*), 10

C

call() (*Base method*), 11
centroid (*Maps.MapsTag attribute*), 6
count (*Maps attribute*), 6

E

edges (*Graph attribute*), 4
endpoints (*class in instahashtag.http*), 10
exists (*Graph attribute*), 3
exists (*Tag attribute*), 2

G

geo (*Tag attribute*), 2
geo (*Tag.TagResult attribute*), 2
Graph (*built-in class*), 3
graph() (*Base class method*), 11
graph() (*in module instahashtag.api*), 8
Graph.__init__() (*in module instahashtag.wrapper.Graph*), 3
Graph.call() (*in module instahashtag.wrapper.Graph*), 5
Graph.GraphEdge (*class in instahashtag.wrapper.graph*), 4
Graph.GraphNode (*class in instahashtag.wrapper.graph*), 4

H

hashtag (*Graph attribute*), 3
hashtag (*Tag attribute*), 1

I

id (*Graph.GraphEdge attribute*), 4
id (*Graph.GraphNode attribute*), 4
instahashtag.api
  module, 7

M

Maps (*built-in class*), 5
maps() (*Base class method*), 11
maps() (*in module instahashtag.api*), 9
Maps.__init__() (*in module instahashtag.wrapper.Maps*), 5
Maps.call() (*in module instahashtag.wrapper.Maps*), 6
Maps.MapsTag (*class in instahashtag.wrapper.maps*), 6
Maps.MapsTag.__gt__() (*in module instahashtag.wrapper.maps.MapsTag*), 6
media_count (*Tag.TagResult attribute*), 2
module
  instahashtag.api, 7

N

nodes (*Graph attribute*), 3

P

process() (*Base static method*), 11

R

rank (*Tag attribute*), 2
rank (*Tag.TagResult attribute*), 2
relevance (*Graph.GraphNode attribute*), 4
relevance (*Tag.TagResult attribute*), 2
Requests (*class in instahashtag.http*), 11
results (*Tag attribute*), 2
root_pos (*Graph attribute*), 3

T

Tag (*built-in class*), 1
tag (*Maps.MapsTag attribute*), 6
tag (*Tag.TagResult attribute*), 2
tag() (*Base class method*), 11
tag() (*in module instahashtag.api*), 7

Tag.__init__() (*in module instahashtag.wrapper.Tag*), 1
Tag.call() (*in module instahashtag.wrapper.Tag*), 3
Tag.TagResult (*class in instahashtag.wrapper.tag*), 2
Tag.TagResult.__gt__() (*in module instahashtag.wrapper.tag.TagResult*), 3
tags (*Maps attribute*), 6

W

weight (*Graph.GraphEdge attribute*), 4
weight (*Graph.GraphNode attribute*), 4
weight (*Maps.MapsTag attribute*), 6

X

x (*Graph.GraphNode attribute*), 4
x1 (*Maps attribute*), 5
x2 (*Maps attribute*), 5

Y

y (*Graph.GraphNode attribute*), 4
y1 (*Maps attribute*), 5
y2 (*Maps attribute*), 5

Z

zoom (*Maps attribute*), 5