Minoshiro Documentation

Release 0.1.8

MaT1g3R, dashwav

Oct 06, 2018

Contents

Inspired by the original Reddit bot, Minoshiro is an async Python library that brings various web APIs for anime, manga, and light novel into one place.

CHAPTER 1

Requirements

Requires Python3.6+

CHAPTER 2

Features

- Simple and modern Pythonic API using `async/await` syntax
- Fetches search results from up to 9 different websites
- Cached search results for faster access
- Integrates with existing databases

CHAPTER 3

Contents

3.1 Install

To install the base version, simply install from PyPi:

```
pip install minoshiro
```

This library also comes with a PostgreSQL data controller, to use this you will need a PostgreSQL version 9.6 or above instance hosted.

To install with the built in PostgreSQL support, use:

```
pip install minoshiro[postgres]
```

or simply install asyncpg version 0.12.0 or later from PyPi

To achieve maximum speed with this library, uvloop is highly recommended.

3.2 Quickstart

This page gives a brief introduction to the library. It assumes you have the library installed, if you don't check the *Install* portion.

The code block below showcases the basic usage using the built in support for PostgreSQL and SQLite3.

```
from pathlib import Path

from minoshiro import Medium, Minoshiro, Site


async def postgres():
    postgres_config = {
        "host": "localhost",
```

(continues on next page)

(continued from previous page)

```
        "port": "5432",
        "user": "postgres",
        "database": "postgres"
    }

    robo = await Minoshiro.from_postgres(
        postgres_config, cache_pages=1
    )

    # get_data eagerly evaluates all the data and return them in a dict
    saekano = await robo.get_data(
        'Saenai Heroine no Sodatekata', Medium.ANIME
    )
    for site, data in saekano.items():
        print(site, data)

    # yield_data lazily evaluates the data and you can iterate over them
    async for site, data in robo.yield_data('New Game!', Medium.MANGA):
        print(site, data)


async def sqlite():
    # from_sqlite method accepts both a string or a Pathlib Path object.
    sqlite_path = 'path/to/sqlite/database'
    another_path = Path('another/sqlite/path')

    robo = await Minoshiro.from_sqlite(
        sqlite_path
    )

    # We only want the results from those 2 sites.
    sites = (Site.LNDB, Site.ANILIST)
    async for site, data in robo.get_data('overlord', Medium.LN, sites):
        print(site, data)

    another_robo = await Minoshiro.from_sqlite(
        mal_config, another_path
    )
    # Specify the seconds for HTTP request timeout.
    print(
        await another_robo.get_data(
            'Love Live Sunshine!', Medium.ANIME, timeout=10
        )
    )
```

3.3 Logging

Minoshiro logs errors and debug information via the logging Python module. The library comes with a basic logger that prints to STDERR. It is strongly recommended you log to a log file in addition to printing to STDERR.

Configuration of the logger can be as simple as:

```
from logging import FileHandler, Formatter
```

(continues on next page)

(continued from previous page)

```
from minoshiro import get_default_logger

logger = get_default_logger()
file_handler = FileHandler(filename='my_log_file.log')
file_handler.setFormatter(
    Formatter('%(asctime)s:%(levelname)s:%(name)s: %(message)s')
)
logger.addHandler(file_handler)
```

More advance setups are possible with the logging module. You can configure the logger to your liking as such:

```
from logging import getLogger

import minoshiro

my_logger = getLogger('minoshiro')

...
```

And finally, if you already have a logger set up in your application, you can simply use the existing logger instead of the one provided by the library.

3.4 Database

Minoshiro uses caching to make search results faster and more accurate.

3.4.1 Build in database support

Minoshiro comes with built in support for SQLite3 and PostgreSQL databases.

To use the built in SQLite3 support, simply use the `from_sqlite` method as such:

```
from minoshiro import Minoshiro

async def main():

    db_path = 'path/to/database'

    robo = await Minoshiro.from_sqlite(db_path)
```

To use the built in PostgreSQL support, you will need asyncpg to be installed. Check *Install* for more information. Then, use the `from_postgres` method as such:

```
from minoshiro import Minoshiro


async def main():
    db_config = {
        "host": "localhost",
        "port": "5432",
        "user": "postgres",
        "database": "postgres"
    }
```

(continues on next page)

(continued from previous page)

```
    robo = await Minoshiro.from_postgres(
        db_config, schema='my_schema'
    )
```

3.4.2 Extending DatabaseController

You may also write your custom implementation of the database controller if you wish. To get started, inherit from the `DataController` class as such:

```
from minoshiro import DataController


class MyDatabase(DataController):
    def __init__(self, logger):
        super().__init__(logger)
```

You will need to initialize the super class with a logger object.

Next, you will need to implement ALL of the following methods. The methods MUST be defined with `async def`.

```
@abstractmethod
async def get_identifier(self, query: str,
                         medium: Medium) -> Optional[Dict[Site, str]]:
    """
    Get the identifier of a given search query.

    :param query: the search query.
    :type query: str

    :param medium: the medium type.
    :type medium: Medium

    :return:
        A dict of all identifiers for this search query for all sites,
        None if nothing is found.
    :rtype: Optional[Dict[Site, str]]
    """
    raise NotImplementedError

@abstractmethod
async def set_identifier(self, name: str, medium: Medium,
                         site: Site, identifier: str):
    """
    Set the identifier for a given name.

    :param name: the name.
    :type name: str

    :param medium: the medium type.
    :type medium: Medium

    :param site: the site.
    :type site: Site
```

(continues on next page)

(continued from previous page)

```
    :param identifier: the identifier.
    :type identifier: str
    """
    raise NotImplementedError

@abstractmethod
async def medium_data_by_id(self, id_: str, medium: Medium,
                            site: Site) -> Optional[dict]:
    """
    Get data by id.

    :param id_: the id.
    :type id_: str

    :param medium: the medium type.
    :type medium: Medium

    :param site: the site.
    :type site: Site

    :return: the data for that id if found.
    :rtype: Optional[dict]
    """
    raise NotImplementedError

@abstractmethod
async def set_medium_data(self, id_: str, medium: Medium,
                          site: Site, data: dict):
    """
    Set the data for a given id.

    :param id_: the id.
    :type id_: str

    :param medium: the medium type.
    :type medium: Medium

    :param site: the site.
    :type site: Site

    :param data: the data for the id.
    :type data: dict
    """
    raise NotImplementedError
```

3.5 API

The following section outlines the API of Minoshiro.

3.5.1 Default Logger

get_default_logger()
Return a basic default `logging.Logger`

Returns

A basic logger with a `logging.StreamHandler` attatched and with level `INFO`

3.5.2 Minoshiro

class Minoshiro(*db_controller*, *, *logger=None*, *loop=None*)

Represents the search instance.

It is suggested to use one of the class methods to create the instance if you wish to use one of the data controllers provided by the library.

Make sure you run the *pre_cache()* method if you initialized the class directly from the `__init__` method.

Parameters

- db_controller(*DataController*) - Any sub class of *DataController* will work here.
- logger(Optional[`logging.Logger`]) - The logger object. If it's not provided, will use the defualt logger provided by the library.
- loop(Optional[Event loop]) - An asyncio event loop. If not provided will use the default event loop.

classmethod from_postgres(*db_config = None*, *pool=None*, *, *schema='minoshiro'*, *cache_pages=0*, *logger=None*, *loop=None*)

This method is a *coroutine*

Get an instance of *Minoshiro* with *PostgresController* as the database controller.

Parameters

- db_config(`dict`) - A dict of database config for the connection. It should contain the keys in keyword arguments for the `asyncpg.connection.connect` function.

  An example config might look like this:

  ```
  db_config = {
      "host": 'localhost',
      "port": '5432',
      "user": 'postgres',
      "database": 'postgres'
  }
  ```

- pool(Pool) - an existing `asyncpg` connection pool.

  One of `db_config` or `pool` must not be None.

- schema(Optional[`str`]) - the name for the schema used. Defaults to `minoshiro`
- cache_pages(Optional[`int`]) - The number of pages of anime and manga from Anilist to cache before the instance is created. Each page contains 40 entries max.
- logger(Optional[`logging.Logger`]) - The logger object. If it's not provided, will use the defualt logger provided by the library.
- loop(Optional[Event loop]) - An asyncio event loop. If not provided will use the default event loop.

Returns

Instance of *Minoshiro* with *PostgresController* as the database controller.

classmethod from_sqlite(*path*, *, *cache_pages=0*, *logger=None*, *loop=None*)

This method is a *coroutine*

Get an instance of *Minoshiro* with *SqliteController* as the database controller.

Parameters

- path(Union[str, pathlib.Path]) - The path to the SQLite3 database, can either be a string or a Pathlib Path object.
- cache_pages(Optional[int]) - The number of pages of anime and manga from Anilist to cache before the instance is created. Each page contains 40 entries max.
- logger(Optional[logging.Logger]) - The logger object. If it's not provided, will use the defualt logger provided by the library.
- loop(Optional[Event loop]) - An asyncio event loop. If not provided will use the default event loop.

Returns

Instance of *Minoshiro* with *PostgresController* as the database controller.

pre_cache(*cache_pages*)
This method is a *coroutine*

Pre cache the database with anime and managa data.

This method is called by *from_postgres()* and *from_sqlite()*, so you do not need to call this method if you created ths class instance with those two methods.

Parameters

- cache_pages(int) - Number of Anilist pages to cache. There are 40 entries per page.

yield_data(*query*, *medium*, *sites*, ***, *timeout=3*)
This method is a *coroutine*

Yield the data for the search query from all sites.

Sites with no data found will be skipped.

Parameters

- query(str) - the search query
- medium(*Medium*) - the medium type
- sites(Optional[Iterable[*Site*]]) - an iterable of sites desired. If None is provided, will search all sites by default
- timeout(Optional[int]) - The timeout in seconds for each HTTP request. Defualt is 3.

Returns

An asynchronous generator that yields the site and data in a tuple for all sites requested.

get_data(*query*, *medium*, *sites*, ***, *timeout=3*)
This method is a *coroutine*

Get the data for the search query in a dict.

Sites with no data found will not be in the return value.

Parameters

- query(str) - the search query
- medium(*Medium*) - the medium type
- sites(Optional[Iterable[*Site*]]) - an iterable of sites desired. If None is provided, will search all sites by default

- timeout(Optional[int]) - The timeout in seconds for each HTTP request. Defualt is 3.

Returns

Data for all sites in a dict `{Site: data}`

Note

When retrieving data from the result of this method, use the `dict.get()` method instead of square brackets.

Example:

```
results = await search_instance.get_data(
    'Non Non Biyori', Medium.ANIME
)

# Good
anilist = results.get(Site.ANILIST)

# Bad, might raise KeyError
anilist = results[Site.ANILIST]
```

3.5.3 Enums

Minoshiro uses two enums to represent medium type and website.

class Site

MAL = 1

ANILIST = 2

ANIMEPLANET = 3

ANIDB = 4

KITSU = 5

MANGAUPDATES = 6

LNDB = 7

NOVELUPDATES = 8

VNDB = 9

class Medium

ANIME = 1

MANGA = 2

LN = 3

VN = 4

3.5.4 Database Controllers

class DataController(*logger*)
An ABC (abstract base class) that deals with database caching.

See *Extending DatabaseController* for details.

class PostgresController(*pool*, *logger*, *schema='minoshiro'*)
To be able to integrate with an existing database, all tables for minoshiro will be put under the `minoshiro` schema unless a different schema name is passed to the __init__ method.

Create the instance with the *`get_instance()`* method to make sure you have all the tables needed.

classmethod get_instance(*logger*, *connect_kwargs=None*, *pool=None*, *schema='minoshiro'*)
This method is a *coroutine*

Get a new instance of *`PostgresController`*

This method will create the appropriate tables needed.

Parameters

- logger(Optional[`logging.Logger`]) - The logger object. If it's not provided, will use the defualt logger provided by the library.
- connect_kwargs(`dict`) - A dict of database config for the connection. It should contain the keys in keyword arguments for the `asyncpg.connection.connect` function.

  An example config might look like this:

  ```
  db_config = {
      "host": 'localhost',
      "port": '5432',
      "user": 'postgres',
      "database": 'postgres'
  }
  ```

- pool(Pool) - an existing `asyncpg` connection pool.

  One of `db_config` or `pool` must not be None.

- schema(`str`) - the name for the schema used. Defaults to `minoshiro`

Returns

a new instance of *`PostgresController`*

class SqliteController(*path*, *logger*, *loop=None*)
A SQLite3 data controller.

Create the instance with the *`get_instance()`* method to make sure you have all the tables needed.

classmethod get_instance(*path*, *logger=None*, *loop=None*)
This method is a *coroutine*

Get a new instance of *`SqliteController`*

This method will create the appropriate tables needed.

Parameters

- path(Union[`str`, `pathlib.Path`]) - The path to the SQLite3 database, can either be a string or a Pathlib Path object.
- logger(Optional[`logging.Logger`]) - The logger object. If it's not provided, will use the defualt logger provided by the library.

- loop(Optional[Event loop]) - An asyncio event loop. If not provided will use the default event loop.

Returns

A new instance of *SqliteController*

CHAPTER 4

License

Minoshiro is released under the MIT License. See LICENSE file for details.

Index