ConfigAlchemy Documentation

Release 0.5.5

GuangTian Li

Jul 13, 2021

Contents:

CHAPTER 1

ConfigAlchemy

The Settings and Configuration on ideal practices for app development and package building.

- Free software: MIT license
- Documentation: https://configalchemy.readthedocs.io.

1.1 Installation

```
$ pipenv install configalchemy
```

Only **Python 3.6+** is supported.

1.2 Example

```
from configalchemy import BaseConfig

class DefaultConfig(BaseConfig):
    NAME = "test"

config = DefaultConfig()
```

(continues on next page)

(continued from previous page)

```
config.NAME
>>> 'test'
```

1.3 Features

- Base on The Twelve-Factor App Configuration.
- Configurable dynamic configurator
- Configuration-Oriented Development
  - Define default config value and its type which is used in your project
  - Use class to support inheritance to explicitly define configurable config
- Override config value from multiple source with **priority supported**
  - Callable function return value
  - File (default: json)
  - Environment Variables
- **Proper Typecast** before overriding
- Generic Config Type Support by custom typecast
- Lazy and Proxy Object Support.
- Extension
  - Full Apollo - A reliable configuration management system Features Support

1.4 TODO

- IOC - Injector, Singleton

CHAPTER 2

Installation

2.1 Stable release

To install ConfigAlchemy, run this command in your terminal:

```
$ pipenv install configalchemy
```

This is the preferred method to install configalchemy, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for configalchemy can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/GuangTianLi/configalchemy
```

Or download the tarball:

```
$ curl  -OL https://github.com/GuangTianLi/configalchemy/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

Or using pipenv install straightly:

```
$ pipenv install -e git+https://github.com/GuangTianLi/configalchemy#egg=configalchemy
```

CHAPTER 3

Usage

To use ConfigAlchemy in a project.

Note: the configuration key should be **uppercase**.

```
from configalchemy import BaseConfig

class DefaultConfig(BaseConfig):
    TEST = "test"

config = DefaultConfig()
```

3.1 How to Use the Config

Inherit from *BaseConfig* to dynamic configure your configuration.

Note: Default Priority Environment Variables > File > Function Return Value > Default Value

3.1.1 Define the Default Config Class

Inherit from *BaseConfig* to define your config value explicitly:

Note: the config key should be **uppercase**.

```
from configalchemy import BaseConfig

class DefaultConfig(BaseConfig):
    DEBUG = False
    TESTING = False
    DATABASE_URI = 'sqlite:///:memory:'

class ProductionConfig(DefaultConfig):
    DATABASE_URI = 'mysql://user@localhost/foo'

class DevelopmentConfig(DefaultConfig):
    DEBUG = True

class TestingConfig(DefaultConfig):
    TESTING = True

config = DevelopmentConfig()
>>> config.DEBUG
True
```

3.1.2 Enable Environment Variables

Define the **CONFIGALCHEMY_ENV_PREFIX** to enable access config from environment variables:

Note: The *BaseConfig* will try to load config value with f'{CONFIGALCHEMY_ENV_PREFIX}{key}'.

```
from configalchemy import BaseConfig
import os

os.environ['TEST_NAME'] = 'env'

class DefaultConfig(BaseConfig):
    CONFIGALCHEMY_ENV_PREFIX = 'TEST_'
    NAME = 'base'

config = DefaultConfig()

>>> config['NAME']
env
```

3.1.3 Enable Configure from File

Define **CONFIGALCHEMY_CONFIG_FILE** to enable access config from config file:

Note: Support JSON file

```
from configalchemy import BaseConfig

class DefaultConfig(BaseConfig):
    CONFIGALCHEMY_CONFIG_FILE = 'test.json' #: etc: {'NAME': 'json'}
```

(continues on next page)

(continued from previous page)

```
    NAME = 'base'

config = DefaultConfig()

>>> config['NAME']
json
```

3.1.4 Enable Configure with function return value

Define **CONFIGALCHEMY_ENABLE_FUNCTION** to configure from function return value (support coroutine):

```
from configalchemy import BaseConfig, ConfigType


class SyncDefaultConfig(BaseConfig):
    CONFIGALCHEMY_ENABLE_FUNCTION = True
    TYPE = "base"

    def configuration_function(self) -> ConfigType:
        return {"TYPE": "sync"}


class AsyncDefaultConfig(BaseConfig):
    CONFIGALCHEMY_ENABLE_FUNCTION = True
    TYPE = "base"

    async def async_function(self) -> ConfigType:
        return {"TYPE": "async"}


sync_config = SyncDefaultConfig()
async_config = AsyncDefaultConfig()

>>> sync_config['TYPE']
sync
>>> async_config['NAME']
async
```

3.2 Auto Validation and Dynamic typecast

When new value is assigned to config, the value will be validated and typecast if possible and the process bases on *default value* or *type annotations*.

```
from typing import Optional
from configalchemy import BaseConfig

class DefaultConfig(BaseConfig):
    id = 1
    name = "Tony"
    limit: Optional[int] = None

config = DefaultConfig()
```

(continues on next page)

(continued from previous page)

```
config.id = '10'
print(config.id) # 10
config.limit = 10
print(config.limit) # 10
```

3.2.1 Json Type

You can use Json data type - *configalchemy* will use json.loads to typecast.

```
import json

from configalchemy import BaseConfig
from configalchemy.types import Json

class DefaultConfig(BaseConfig):
    TEST_LIST: Json[list] = ["str"]
    TEST_DICT: Json[dict] = {"name": "default"}

config = DefaultConfig()
config.TEST_LIST = json.dumps(["test"])
config.TEST_LIST
>>> ["test"]
config.TEST_DICT = json.dumps({"name": "test"})
config.TEST_DICT
>>> {"name": "test"}
```

3.3 Advanced Usage

3.3.1 Singleton

3.3.2 Nested Config for Modular Purpose

```
class NestedConfig(BaseConfig):
    NAME = "nested"
    ADDRESS = "default"

class DefaultConfig(BaseConfig):
    NESTED_CONFIG = NestedConfig()

config = DefaultConfig()
config.update(NESTED_CONFIG={"NAME": "updated"})
config["NESTED_CONFIG.ADDRESS"] = "address"
>>> config.NESTED_CONFIG.NAME
updated
>>> config.NESTED_CONFIG.ADDRESS
address
```

3.3.3 Lazy

Use *lazy* to turn any callable into a lazy evaluated callable. Results are memoized; the function is evaluated on first access.

```
from configalchemy.lazy import lazy, reset_lazy


def get_name():
    print("evaluating")
    return "World"


lazy_name = lazy(get_name)
>>> print(f"Hello {lazy_name}")
evaluating
Hello World
>>> print(f"Hello {lazy_name}")
Hello World
>>> reset_lazy(lazy_name)
>>> print(f"Hello {lazy_name}")
evaluating
Hello World
```

3.3.4 Proxy

Use *proxy* to turn any callable into a lazy evaluated callable. Results are not memoized; the function is evaluated on every access.

```
from configalchemy.lazy import proxy


def get_name():
    print("evaluating")
    return "World"


lazy_name = proxy(get_name)
>>> print(f"Hello {lazy_name}")
evaluating
Hello World
>>> print(f"Hello {lazy_name}")
evaluating
Hello World
```

3.3.5 Pool

Use *Pool* to turn any callable into a pool. Result will be return by a context manager. the function is evaluated on result first access.

```
from configalchemy.lazy import Pool


def connect():
```

(continues on next page)

(continued from previous page)

```
    print("connecting")
    return socket


connect_pool = Pool(connect)
>>> with connect_pool as connect:
...     connect.send("")
connecting
0
>>> with connect_pool as connect:
...     connect.send("")
0
```

3.3.6 Access config from Apollo

Apollo - A reliable configuration management system

You can inherit from *ApolloBaseConfig* to access config from Apollo.

```
from configalchemy.contrib.apollo import ApolloBaseConfig
class DefaultConfig(ApolloBaseConfig):

    #: apollo
    ENABLE_LONG_POLL = True
    APOLLO_SERVER_URL = ""
    APOLLO_APP_ID = ""
    APOLLO_CLUSTER = "default"
    APOLLO_NAMESPACE = "application"
```

CHAPTER 4

configalchemy

4.1 configalchemy package

4.1.1 BaseConfig module

class configalchemy.**BaseConfig**

Initialize the *BaseConfig* with the Priority:

```
configure from env > configure from local file > configure from function >␣
↪default configuration
```

Example of class-based configuration:

```
class DefaultConfig(BaseConfig):
    TEST = "test"

config = DefaultConfig()
```

CONFIGALCHEMY_ENABLE_FUNCTION = False

set to `True` if you want to override config from function return value.

CONFIGALCHEMY_ENV_PREFIX = ''

The prefix to construct the full environment variable key to access overrode config.

CONFIGALCHEMY_LOAD_FILE_SILENT = False

set to `True` if you want silent failure for missing files.

CONFIGALCHEMY_ROOT_PATH = ''

The the filename of the JSON file. This can either be an absolute filename or a filename relative to the *CONFIGALCHEMY_ROOT_PATH*.

CONFIGALCHEMY_SETITEM_PRIORITY = 99

The priority of config['TEST'] = value, config.TEST = value and config.update(TEST=value)

access_config_from_coroutine(*priority: int*) → bool
Async updates the values in the config from the configuration_function.

access_config_from_function(*priority: int*) → bool
Updates the values in the config from the configuration_function.

from_mapping(**mappings*, *priority: int*) → bool
Updates the config like `update()` ignoring items with non-upper keys.

4.1.2 ApolloBaseConfig module

class configalchemy.contrib.apollo.**ApolloBaseConfig**

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/GuangTianLi/configalchemy/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

configalchemy could always use more documentation, whether as part of the official configalchemy docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/GuangTianLi/configalchemy/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *configalchemy* for local development.

1. Fork the *configalchemy* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/configalchemy.git
```

3. Install your local copy into a virtualenv. Assuming you have Pipenv installed, this is how you set up your fork for local development:

```
$ cd configalchemy/
$ make init
$ pipenv shell
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass the tests.:

```
$ make lint
$ make test
```

- *tag* - https://gitmoji.carloscuesta.me/

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m ":tag: [#id] Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6+. Check https://github.com/GuangTianLi/configalchemy/actions and make sure that the tests pass for all supported Python versions.

CHAPTER 6

Credits

6.1 Development Lead

- GuangTian Li <guangtian_li@qq.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.5.* (2020-12)

- Support SingletonMetaClass
- Remove unpack feature in *BaseConfig* by removing Mapping Class
- Support nested config for large modular purpose
- Support local object
- Support generic pool object
- Support Dot Notation to update *BaseConfig* object

7.2 0.4.* (2020-06)

- Refatory configure function
- Support property configuration

7.3 0.3.* (2020-03)

- Add proxy and lazy module.
- Add find_caller to trace source of config value
- Refactory ConfigMeta data structure

7.4 0.2.* (2019-08)

- Change global variable to weak reference
- Remove Lock (Prepare to implement optimistic raw lock if necessary)
- Improve Priority Data Structure
- Improve Field Validation
- Use OOP to define call function
- Properer validation and typecast
- Improve Type Annotations
- Import JSON type

7.5 0.1.0 (2019-08-01)

- Init Project.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Index

A

B

C

F