Cache Alchemy Documentation

Release 0.4.5

GuangTian Li

Aug 10, 2021

Contents:

CHAPTER 1

Cache Alchemy

The Python Cache Toolkit.

- Free software: MIT license
- Documentation: https://cache-alchemy.readthedocs.io/en/latest/

1.1 Installation

```
$ pipenv install cache-alchemy
```

Only **Python 3.6+** is supported.

1.2 Example

```
import dataclasses

from redis import Redis

from cache_alchemy import memory_cache, json_cache, pickle_cache
from cache_alchemy.config import DefaultConfig
```

(continues on next page)

(continued from previous page)

```
config = DefaultConfig()
config.cache_redis_client = Redis.from_url(config.CACHE_ALCHEMY_REDIS_URL)


@dataclasses.dataclass
class User:
    name: str


@pickle_cache()
def get(name: str) -> User:
    return User(name=name)


@memory_cache()
def add(i: complex, j: complex) -> complex:
    return i + j


@json_cache()
def add(i: int, j: int) -> int:
    return i + j
```

1.3 Features

- Distributed cache
- Cache clear and partial clear with specific function parameter
- Cache clear cascade by dependency
- Cache `Json Serializable` function return value with **json_cache**
- Cache Python Object function return value with **pickle_cache**
- Cache any function return value with **memory_cache**
- LRU Dict support

1.4 TODO

CHAPTER 2

Installation

2.1 Stable release

To install Cache Alchemy, run this command in your terminal:

```
$ pipenv install cache-alchemy
```

This is the preferred method to install cache-alchemy, 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 cache-alchemy can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/GuangTianLi/cache-alchemy
```

Or download the tarball:

```
$ curl  -OL https://github.com/GuangTianLi/cache-alchemy/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/cache-alchemy#egg=cache_alchemy
```

CHAPTER 3

Usage

Warning: The cache decorator must be used after config initialized.

Warning: The cache_redis_client must be assigned after config initialized if you want to use distributed cache and set decode_responses to False.

To use Cache Alchemy in a project.

```
from cache_alchemy import memory_cache, json_cache, method_json_cache, property_json_
↪cache
from cache_alchemy.config import DefaultConfig
from redis import Redis

config = DefaultConfig()
config.cache_redis_client = Redis.from_url(config.CACHE_ALCHEMY_REDIS_URL)

@memory_cache()
def add(i: complex, j: complex) -> complex:
    return i + j

@json_cache()
def add(i: int, j: int) -> int:
    return i + j

class Foo:
    x = 2

    @classmethod
    @method_json_cache()
    def add(cls, y: int) -> int:
        return cls.x + b
```

(continues on next page)

(continued from previous page)

```
    @method_json_cache()
    def pow(self, y: int) -> int:
        return pow(self.x, y)

    @property
    @property_json_cache()
    def name(self) -> int:
        return self.x

# Using decorated function to clear cache
add.cache_clear()
```

3.1 Json Cache

Note: Json related cache only support function which return the pure JSON serializable object. Otherwise there is a different between return value and cached value which will cause some unexpected behavior. If you want to cache python object e.g dataclass, see *Pickle Cache*.

3.2 Pickle Cache

Pickle cache use package - pickle to serializing and de-serializing a Python object structure which can handle and cache custom classes e.g: dataclass.

```
import dataclasses

from redis import Redis

from cache_alchemy import pickle_cache
from cache_alchemy.config import DefaultConfig


@dataclasses.dataclass
class User:
    name: str


config = DefaultConfig()
config.cache_redis_client = Redis.from_url(config.CACHE_ALCHEMY_REDIS_URL)


@pickle_cache()
def add(i: complex, j: complex) -> complex:
    return i + j


@pickle_cache()
def access_user(name: str) -> User:
    return User(name=name)
```

3.3 Configuration

You can define your custom config by inherit from *DefaultConfig* which defined a list of configuration available in Cache Alchemy and their default values.

Note: DefaultConfig is defined by *configalchemy* - https://configalchemy.readthedocs.io

3.4 General Memory Cache

Cache Alchemy use distributed backend as default backend to cache function return value.

By setting `CACHE_ALCHEMY_MEMORY_BACKEND` to `cache_alchemy.backends.memory.MemoryCache` can enable general memory cache backend.

```
from cache_alchemy import memory_cache
from cache_alchemy.config import DefaultConfig

class CacheConfig(DefaultConfig):
    CACHE_ALCHEMY_MEMORY_BACKEND = "cache_alchemy.backends.memory.MemoryCache"

config = CacheConfig()

@memory_cache()
def add(i: complex, j: complex) -> complex:
    return i + j
```

3.5 Define a cache dependency

Use cache dependency to declare dependency between two function.

```
@json_cache()
def add(a, b):
    return a + b

dependency = FunctionCacheDependency(add)

@json_cache(dependency=[dependency])
def add_and_double(a, b):
    return add(a, b) * 2
```

When cache of add has been cleared, add_and_double will clear cascade.

CHAPTER 4

API reference

4.1 Cache Function

cache_alchemy.**cache**(*limit: Optional[int], expire: Optional[int], is_method: bool, strict: bool, backend: str, dependency: List[cache_alchemy.dependency.CacheDependency], cache_key_prefix: str = ”, **kwargs*) → Callable[function, Callable[..., ReturnType]]

The base function to creat a cache object like this:

```
@cache(
    limit=1000,
    expire=60,
    is_method=False,
    strict=True,
    backend="cache_alchemy.backends.memory.MemoryCache",
    dependency=[],
)
def f(x, y):
    pass

# To clear cache
f.cache_clear()
```

Parameters

- **expire** (*int*) – expire time with an integer value used as seconds.
- **is_method** (*bool*) – If *True*, the first argument will be ignored in generate cache key.
- **strict** (*bool*) – If *False*, make a cache key in a way that is flat as possible rather than as a nested and strict structure that would support partially cache clear. it means that f(x=1, y=2) will now be treated as a distinct call from f(y=2, x=1) which will be cached separately.

4.2 DefaultConfig Object

class cache_alchemy.config.**DefaultConfig**

Bases: `configalchemy.configalchemy.BaseConfig`

CACHE_ALCHEMY_CACHE_KEY_PREFIX = ''
cache key prefix to avoid key conflict

CACHE_ALCHEMY_DEFAULT_EXPIRE = 86400
default cache expire time (seconds) - setting to 0 means uncached

CACHE_ALCHEMY_DEFAULT_LIMIT = 1000
default cache limit per function - setting to -1 means unlimited - setting to 0 means uncached

CACHE_ALCHEMY_JSON_BACKEND = 'cache_alchemy.backends.json.DistributedJsonCache'
distributed json cache backend - default: distributed cache which need assign client to config

CACHE_ALCHEMY_MEMORY_BACKEND = 'cache_alchemy.backends.memory.DistributedMemoryCache'
memory cache backend - default: distributed cache which need assign client to config

CACHE_ALCHEMY_PICKLE_BACKEND = 'cache_alchemy.backends.pickle.DistributedPickleCache'
memory cache backend - default: distributed cache which need assign client to config

CACHE_ALCHEMY_REDIS_URL = 'redis://127.0.0.1:6379/0'
default redis url

cache_redis_client = None
Need to be assigned after init, if use distributed cache

4.3 FunctionCacheDependency Object

Examples:

```
@json_cache()
def add(a, b):
    return a + b

dependency = FunctionCacheDependency(add)

@json_cache(dependency=[dependency])
def add_and_double(a, b):
    return add(a, b) * 2
```

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/cache_alchemy/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

cache_alchemy could always use more documentation, whether as part of the official cache_alchemy 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/cache_alchemy/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 *cache_alchemy* for local development.

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

```
$ git clone git@github.com:your_name_here/cache_alchemy.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 cache_alchemy/
$ 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://travis-ci.org/GuangTianLi/cache_alchemy/pull_requests 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 0.4.* (2020)

- Refactory redis cache to json cache
- Support pickle Cache
- Add backend class in function hash
- Add cache key prefix to avoid key conflict

7.2 0.2.* (2019)

- Support Partially Clear Cache with Arguments
- Support Flush Backend Cache
- Cache Redis Client Must Decode Responses

7.3 0.1.* (2019)

- Support Method and Property Cache
- Support cache as a decorator with no arguments.
- Init Project.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index