
Django Async Redis Documentation

Release 0.1.0

Andrew Chen Wang

Oct 09, 2020

Contents:

1	Django Async Redis	1
1.1	Introduction	1
1.2	Requirements	1
1.3	User guide	1
1.4	Advanced usage	2
1.5	Notes	5
2	Installation	7
2.1	Stable release	7
2.2	From sources	7
3	Usage	9
4	Contributing	11
4.1	Types of Contributions	11
4.2	Get Started!	12
4.3	Pull Request Guidelines	13
4.4	Tips	13
5	Credits	15
5.1	Development Lead	15
5.2	Contributors	15
6	History	17
6.1	0.1.0 (2020-09-25)	17
7	Indices and tables	19

1.1 Introduction

django-async-redis is a full featured Redis cache and session backend for Django.

- Free software: Apache Software License 2.0
- Documentation: <https://django-async-redis.readthedocs.io>.

1.2 Requirements

- Python 3.6+
- Django 3.0+
- aioredis 1.0+
- Redis server 2.8+

1.3 User guide

1.3.1 Installation

Install with pip:

```
$ python -m pip install django-async-redis
```

1.3.2 Configure as cache backend

To start using `django-async-redis`, you should change your Django cache settings to something like:

```
CACHES = {
    "default": {
        "BACKEND": "django_async_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/1",
        "OPTIONS": {
            "CLIENT_CLASS": "django_async_redis.client.DefaultClient",
        }
    }
}
```

`django-async-redis` uses the `aioredis` native URL notation for connection strings, it allows better interoperability and has a connection string in more “standard” way. Some examples:

- `redis://[:password]@localhost:6379/0`
- `rediss://[:password]@localhost:6379/0`
- `unix://[:password]@/path/to/socket.sock?db=0`

Three URL schemes are supported:

- `redis://`: creates a normal TCP socket connection
- `rediss://`: creates a SSL wrapped TCP socket connection
- `unix://`: creates a Unix Domain Socket connection

There are several ways to specify a database number:

- A `db` querystring option, e.g. `redis://localhost?db=0`
- If using the `redis://` scheme, the `path` argument of the URL, e.g. `redis://localhost/0`

1.4 Advanced usage

1.4.1 Pickle version

For almost all values, `django-async-redis` uses `pickle` to serialize objects.

The latest available version of `pickle` is used by default. If you want set a concrete version, you can do it, using `PICKLE_VERSION` option:

```
CACHES = {
    "default": {
        # ...
        "OPTIONS": {
            "PICKLE_VERSION": -1 # Use the latest protocol version
        }
    }
}
```

1.4.2 Memcached exceptions behavior

In some situations, when Redis is only used for cache, you do not want exceptions when Redis is down. This is default behavior in the memcached backend and it can be emulated in django-async-redis.

For setup memcached like behaviour (ignore connection exceptions), you should set `IGNORE_EXCEPTIONS` settings on your cache configuration:

```
CACHES = {
    "default": {
        # ...
        "OPTIONS": {
            "IGNORE_EXCEPTIONS": True,
        }
    }
}
```

Also, you can apply the same settings to all configured caches, you can set the global flag in your settings:

```
DJANGO_ASYNC_REDIS_IGNORE_EXCEPTIONS = True
```

1.4.3 Log Ignored Exceptions

When ignoring exceptions with `IGNORE_EXCEPTIONS` or `DJANGO_ASYNC_REDIS_IGNORE_EXCEPTIONS`, you may optionally log exceptions using the global variable `DJANGO_ASYNC_REDIS_LOG_IGNORED_EXCEPTIONS` in your settings file:

```
DJANGO_ASYNC_REDIS_LOG_IGNORED_EXCEPTIONS = True
```

If you wish to specify the logger in which the exceptions are output, simply set the global variable `DJANGO_ASYNC_REDIS_LOGGER` to the string name and/or path of the desired logger. This will default to `__name__` if no logger is specified and `DJANGO_ASYNC_REDIS_LOG_IGNORED_EXCEPTIONS` is True:

```
DJANGO_ASYNC_REDIS_LOGGER = 'some.specified.logger'
```

1.4.4 Infinite timeout

django-async-redis comes with infinite timeouts support out of the box. And it behaves in the same way as the Django BaseCache backend specifies:

- `timeout=0` expires the value immediately.
- `timeout=None` infinite timeout

```
await cache.set_async("key", "value", timeout=None)
```

1.4.5 Get ttl (time-to-live) from key

With Redis, you can access to ttl of any stored key, for it, django-async-redis exposes `ttl_async` function.

It returns:

- 0 if key does not exists (or already expired).
- None for keys that exists but does not have any expiration.

- ttl value for any volatile key (any key that has expiration).

```
>>> from django.core.cache import cache
>>> await cache.set_async("foo", "value", timeout=25)
>>> await cache.ttl_async("foo")
25
>>> await cache.ttl_async("not-existent")
0
```

1.4.6 Expire & Persist

Additionally to the simple ttl query, you can send persist a concrete key or specify a new expiration timeout using the `persist_async` and `expire_async` methods:

```
>>> await cache.set_async("foo", "bar", timeout=22)
>>> await cache.ttl_async("foo")
22
>>> await cache.persist_async("foo")
>>> await cache.ttl_async("foo")
None
```

```
>>> await cache.set_async("foo", "bar", timeout=22)
>>> await cache.expire_async("foo", timeout=5)
>>> await cache.ttl_async("foo")
5
```

1.4.7 Scan & Delete keys in bulk

`django-async-redis` comes with some additional methods that help with searching or deleting keys using glob patterns.

```
>>> from django.core.cache import cache
>>> await cache.keys_async("foo_*")
["foo_1", "foo_2"]
```

A simple search like this will return all matched values. In databases with a large number of keys this isn't suitable method. Instead, you can use the `iter_keys_async` function that works like the `keys_async` function but uses Redis server side cursors. Calling `iter_keys_async` will return a generator that you can then iterate over efficiently.

```
>>> from django.core.cache import cache
>>> await cache.iter_keys_async("foo_*")
<async_generator object algo at 0x7ffa9c2713a8>
>>> (await cache.iter_keys_async("foo_*")).__anext__()
"foo_1"
```

For deleting keys, you should use `delete_pattern_async` which has the same glob pattern syntax as the `keys_async` function and returns the number of deleted keys.

```
>>> from django.core.cache import cache
>>> await cache.delete_pattern_async("foo_*")
```


1.4.8 Redis native commands

django-async-redis has limited support for some Redis atomic operations, such as the commands SETNX and INCR.

You can use the SETNX command through the backend `set_async()` method with the `nx` parameter:

```
>>> from django.core.cache import cache
>>> await cache.set_async("key", "value1", nx=True)
True
>>> await cache.set_async("key", "value2", nx=True)
False
>>> await cache.get_async("key")
"value1"
```

Also, the `incr_async` and `decr_async` methods use Redis atomic operations when the value that a key contains is suitable for it.

Note that setting `xx` to `True` overrides the `nx` flag according to aioredis.

1.4.9 Connection pools

Behind the scenes, django-async-redis uses the underlying aioredis connection pool implementation and exposes a simple way to configure it. Alternatively, you can directly customize a connection/connection pool creation for a backend.

The default aioredis behavior is to not close connections, recycling them when possible.

1.5 Notes

Since the majority of this code was ported from django-redis, there was one case that had needed a monkeypatch. In `django_async_redis.util`, we implement `CacheKey` which subclasses `str` which helps us know if a cache key was already created. Since aioredis, checks if the cache key is of type `str` (and others), I had to monkeypatch that check so that a `CacheKey` instance could also be accepted.

1.5.1 Credit

- Hey, I'm Andrew. I'm busy in college, but I wanted to help contribute to Django's async ecosystem.
- Lots of code and docs is taken from django-redis, including the tests. I just needed to port everything to asyncio and aioredis.
- I used cookiecutter-pypackage to generate this project.
- Thank you to Python Discord server's async topical chat for helping me understand when to use coroutines over sync functions and @Bast and @hmmmm in general because they're OG.

2.1 Stable release

To install Django Async Redis, run this command in your terminal:

```
$ pip install django-async-redis
```

This is the preferred method to install Django Async Redis, 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 Django Async Redis can be downloaded from the [Github repo](#).

You can either clone the public repository:

```
$ git clone git://github.com/Andrew-Chen-Wang/django-async-redis
```

Or download the [tarball](#):

```
$ curl -OJL https://github.com/Andrew-Chen-Wang/django-async-redis/tarball/master
```

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

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


CHAPTER 3

Usage

To use Django Async Redis in a project:

```
import django_async_redis
```


Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/Andrew-Chen-Wang/django_async_redis/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/Andrew-Chen-Wang/django_async_redis/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 :)

4.2 Get Started!

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

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

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

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

```
$ mkvirtualenv django_async_redis
$ cd django_async_redis/
$ python setup.py develop
```

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 flake8 and the tests, including testing other Python versions with tox:

```
$ black django_async_redis tests
$ python setup.py test or pytest
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

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

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 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 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/Andrew-Chen-Wang/django_async_redis/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ pytest tests.test_django_async_redis
```


5.1 Development Lead

- Andrew Chen Wang <acwangpython@gmail.com>

5.2 Contributors

None yet. Why not be the first?

6.1 0.1.0 (2020-09-25)

- First release on PyPI.

CHAPTER 7

Indices and tables

- `genindex`
- `modindex`
- `search`