Django-Redis Documentation

4.7.0

rapospectre

2017 01 17

Contents

Andrey Antukh, niwi@niwi.be 4.7.0

: RaPoSpectre

CHAPTER 1

1.

django-redis BSD , Django Redis cache/session .

1.1 1.1 django-redis ?

:

-
- redis-py URL
-
-
-
- /
-
- cache session
-
- redis /
- ( )
- unix
- Python 2.7, 3.4, 3.5 3.6

1.2 1.2 django-redis

- : 4.7.0
- : 3.8.4

1.3 1.3

3.6, 3.7 . . . , . .

3.7.0, 3.7.1. . . bug , bug , .

1.4 1.4

1.4.1 1.4.1 Django

- django-redis 3.8.x django 1.4, 1.5, 1.6, 1.7 (1.8)
- django-redis 4.4.x django 1.6, 1.7, 1.8, 1.9 1.10

1.4.2 1.4.2 Redis Server

- django-redis 3.x.y redis-server 2.6.x
- django-redis 4.x.y redis-server 2.8.x

1.4.3 1.4.3

django-redis redis-py >= 2.10.0.

CHAPTER 2

2.

2.1 2.1

django-redis pip :

```
pip install django-redis
```

2.2 2.2 cache backend

django-redis , django cache setting :

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

“”, 3.8.0 django-redis redis-py native url notation .

URL

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

URL scheme :

- redis://: TCP
- rediss://: SSL TCP
- unix://: Unix

:

- db , : redis://localhost?db=0
- redis:// scheme, , : redis://localhost/0

url , OPTIONS :

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

, uri , uri , .

2.3 2.3 session backend

Django cache backend session backend, django-redis session backend

```
SESSION_ENGINE = "django.contrib.sessions.backends.cache"
SESSION_CACHE_ALIAS = "default"
```

2.4 2.4 django-redis

django-redis Redis ([redis][]) , : fakerdis (https://github.com/jamesls/fakeredis) mockredis (https://github.com/locationlabs/mockredis). redis server .

fakeredis :

```
import fakeredis
CACHES = {
    "default": {
        "OPTIONS": {
            "REDIS_CLIENT_CLASS": "fakeredis.FakeStrictRedis",
        }
    }
}
```

, TestCase :

```
def tearDown(self):
    from django_redis import get_redis_connection
    get_redis_connection("default").flushall()
```

CHAPTER 3

3.

3.1 3.1 Pickle

django-redis pickle .

pickle. , PICKLE_VERSION :

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

3.2 3.2

SOCKET_TIMEOUT SOCKET_CONNECT_TIMEOUT :

```
CACHES = {
    "default": {
        # ...
        "OPTIONS": {
            "SOCKET_CONNECT_TIMEOUT": 5,  # in seconds
            "SOCKET_TIMEOUT": 5,  # in seconds
        }
    }
}
```

SOCKET_CONNECT_TIMEOUT : socket

SOCKET_TIMEOUT :

3.3 3.3

django-redis , . :

```
CACHES = {
    "default": {
        # ...
        "OPTIONS": {
            "COMPRESSOR": "django_redis.compressors.zlib.ZlibCompressor",
        }
    }
}
```

lzma :

```
import lzma

CACHES = {
    "default": {
        # ...
        "OPTIONS": {
            "COMPRESSOR": "django_redis.compressors.lzma.LzmaCompressor",
        }
    }
}
```

3.4 3.4 memcached

, redis , . memcached backend , django-redis .

memcached ( ), IGNORE_EXCEPTIONS :

```
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_REDIS_IGNORE_EXCEPTIONS = True
```

3.5 3.5

IGNORE_EXCEPTIONS DJANGO_REDIS_IGNORE_EXCEPTIONS , DJANGO_REDIS_LOG_IGNORED_EXCEPTIONS :

```
DJANGO_REDIS_LOG_IGNORED_EXCEPTIONS = True
```

logger , DJANGO_REDIS_LOGGER logger . logger DJANGO_REDIS_LOG_IGNORED_EXCEPTIONS=True **name** :

```
DJANGO_REDIS_LOGGER = 'some.specified.logger'
```

3.6 3.6

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

django-redis . django backend :

- timeout=0
- timeout=None

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

3.7 3.7 (value) ttl (time to live)

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

It returns:

redis , key ttl, django-redis ttl :

:

- 0 key ().
- None key .
- ttl key .

keys :

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

3.8 3.8 expire & persist

ttl , persist expire :

persist :

```
>>> cache.set("foo", "bar", timeout=22)
>>> cache.ttl("foo")
22
>>> cache.persist("foo")
>>> cache.ttl("foo")
None
```

expire :

```
>>> cache.set("foo", "bar", timeout=22)
>>> cache.expire("foo", timeout=5)
>>> cache.ttl("foo")
5
```

3.9 3.9 locks

django-redis redis . , .

python :

```
with cache.lock("somekey"):
    do_some_thing()
```

3.10 3.10 & (keys)

django-redis .

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

, . redis server side cursors 2.8 , `iter_keys` `keys` , `iter_keys` , .

server side cursors

```
>>> from django.core.cache import cache
>>> cache.iter_keys("foo_*")
<generator object algo at 0x7ffa9c2713a8>
>>> next(cache.iter_keys("foo_*"))
"foo_1"
```

, `delete_pattern` , `keys` ,

delete_pattern

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

3.11 3.11 Redis

django-redis Redis , `SETNX` `INCR` .

set() `nx` `SETNX`

:

```
>>> from django.core.cache import cache
>>> cache.set("key", "value1", nx=True)
True
>>> cache.set("key", "value2", nx=True)
False
>>> cache.get("key")
"value1"
```

(value) (key) , `incr` `decr` Redis

3.12 3.12

Redis django cache . , django-redis `get_redis_connection(alias)` .

```
>>> from django_redis import get_redis_connection
>>> con = get_redis_connection("default")
>>> con
<redis.client.StrictRedis object at 0x2dc4510>
```

.

3.13 3.13

django-redis redis-py , . , backend .

redis-py ,

3.13.1 3.13.1

, `CACHES CONNECTION_POOL_KWARGS` :

```
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        ...
        "OPTIONS": {
            "CONNECTION_POOL_KWARGS": {"max_connections": 100}
        }
    }
}
```

:

```
from django.core.cache import get_cache
from django_redis import get_redis_connection

r = get_redis_connection("default")  # Use the name you have defined for Redis in settings.CACHES
connection_pool = r.connection_pool
print("Created connections so far: %d" % connection_pool._created_connections)
```

3.13.2 3.13.2

. django-redis `CONNECTION_POOL_CLASS`

myproj/mypool.py

```
from redis.connection import ConnectionPool

class MyOwnPool(ConnectionPool):
    # Just doing nothing, only for example purpose
    pass
```

setting.py

```
# Omitting all backend declaration boilerplate code.

"OPTIONS": {
    "CONNECTION_POOL_CLASS": "myproj.mypool.MyOwnPool",
}
```

3.13.3 3.13.3 connection factory

, django-redis connection factory .

django-redis Django setting `DJANGO_REDIS_CONNECTION_FACTORY` `django_redis.pool.ConnectionFactory` .

ConnectionFactory

```
# Note: Using Python 3 notation for code documentation ;)

class ConnectionFactory(object):
    def get_connection_pool(self, params:dict):
        # Given connection parameters in the `params` argument,
        # return new connection pool.
        # It should be overwritten if you want do something
        # before/after creating the connection pool, or return your
        # own connection pool.
        pass

    def get_connection(self, params:dict):
        # Given connection parameters in the `params` argument,
        # return a new connection.
        # It should be overwritten if you want to do something
        # before/after creating a new connection.
        # The default implementation uses `get_connection_pool`
        # to obtain a pool and create a new connection in the
        # newly obtained pool.
        pass

    def get_or_create_connection_pool(self, params:dict):
        # This is a high layer on top of `get_connection_pool` for
        # implementing a cache of created connection pools.
        # It should be overwritten if you want change the default
        # behavior.
        pass

    def make_connection_params(self, url:str) -> dict:
        # The responsibility of this method is to convert basic connection
        # parameters and other settings to fully connection pool ready
        # connection parameters.
        pass

    def connect(self, url:str):
        # This is really a public API and entry point for this
        # factory class. This encapsulates the main logic of creating
        # the previously mentioned `params` using `make_connection_params`
        # and creating a new connection using the `get_connection` method.
        pass
```

3.14 3.14

redis-py (django-redis Redis) Python Redis , , `hiredis`

hiredis C Redis , django-redis :

```
"OPTIONS": {
    "PARSER_CLASS": "redis.connection.HiredisParser",
}
```

3.15 3.15

django_redis .

3.15.1 3.15.1

, : .

, LOCATION :

```
"LOCATION": [
    "redis://127.0.0.1:6379/1",
    "redis://127.0.0.1:6378/1",
]
```

master , slave .

3.15.2 3.15.2

, , ,

```
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": [
            "redis://127.0.0.1:6379/1",
            "redis://127.0.0.1:6379/2",
        ],
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.ShardClient",
        }
    }
}
```

,

3.15.3 3.15.3

, Wikipedia

, , / (keys)

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

:

- CACHE_HERD_TIMEOUT: (: 60s)

3.16 3.16

. django-redis Python pickle .

json , JSONSerializer

```
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/1",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
            "SERIALIZER": "django_redis.serializers.json.JSONSerializer",
        }
    }
}
```

MsgPack http://msgpack.org/ (msgpack-python)

```
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/1",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
            "SERIALIZER": "django_redis.serializers.msgpack.MSGPackSerializer",
        }
    }
}
```

3.17 3.17 Redis

django-redis `redis.client.StrictClient` Redis , , fakeredis .

`REDIS_CLIENT_CLASS in the CACHES`, `REDIS_CLIENT_KWARGS` ().

```
CACHES = {
    "default": {
        "OPTIONS": {
            "REDIS_CLIENT_CLASS": "my.module.ClientClass",
            "REDIS_CLIENT_KWARGS": {"some_setting": True},
        }
    }
}
```

4.

```
Copyright (c) 2011-2015 Andrey Antukh <niwi@niwi.nz>
Copyright (c) 2011 Sean Bleier

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products
   derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```