
requests-cache Documentation

Release 0.3.0

Roman Haritonov

Apr 22, 2021

Contents

1	User guide	3
1.1	Installation	3
1.2	Usage	3
1.3	Persistence	5
1.4	Backward incompatible changes	5
2	API	7
2.1	Public api	7
2.2	Cache backends	9
2.3	Internal modules which can be used outside	11
3	Indices and tables	13
	Python Module Index	15
	Index	17

Requests-cache is a transparent persistent cache for [requests](#) (version $\geq 1.1.0$ see *Backward incompatible changes*) library.

Source code and issue tracking can be found at [GitHub](#).

Contents:

1.1 Installation

Install with `pip` or `easy_install`:

```
pip install --upgrade requests-cache
```

or download latest version from version control:

```
git clone git://github.com/reclosedev/requests-cache.git
cd requests-cache
python setup.py install
```

1.2 Usage

There is two ways of using `requests_cache`:

- Using `CachedSession` instead `requests.Session`
- Monkey patching `requests` to use `CachedSession` by default

Monkey-patching allows to add caching to existent program by adding just two lines:

Import `requests_cache` and call `install_cache()`

```
import requests
import requests_cache

requests_cache.install_cache()
```

And you can use `requests`, all responses will be cached transparently!

For example, following code will take only 1-2 seconds instead 10:

```
for i in range(10):
    requests.get('http://httpbin.org/delay/1')
```

Cache can be configured with some options, such as cache filename, backend (sqlite, mongodb, memory), expiration time, etc. E.g. cache stored in sqlite database (default format) named 'test_cache.sqlite' with expiration set to 300 seconds can be configured as:

```
requests_cache.install_cache('test_cache', backend='sqlite', expire_after=300)
```

See also:

Full list of options can be found in `requests_cache.install_cache()` reference

Transparent caching is achieved by monkey-patching requests library. It is possible to uninstall this patch with `requests_cache.uninstall_cache()`.

Also, you can use `requests_cache.disabled()` context manager for temporary disabling caching:

```
with requests_cache.disabled():
    print(requests.get('http://httpbin.org/ip').text)
```

If Response is taken from cache, `from_cache` attribute will be `True`:

```
>>> import requests
>>> import requests_cache
>>> requests_cache.install_cache()
>>> requests_cache.clear()
>>> r = requests.get('http://httpbin.org/get')
>>> r.from_cache
False
>>> r = requests.get('http://httpbin.org/get')
>>> r.from_cache
True
```

It can be used, for example, for request throttling with help of requests hook system:

```
import time
import requests
import requests_cache

def make_throttle_hook(timeout=1.0):
    """
    Returns a response hook function which sleeps for `timeout` seconds if
    response is not cached
    """
    def hook(response):
        if not getattr(response, 'from_cache', False):
            print('sleeping')
            time.sleep(timeout)
        return response
    return hook

if __name__ == '__main__':
    requests_cache.install_cache('wait_test')
    requests_cache.clear()

    s = requests_cache.CachedSession()
    s.hooks = {'response': make_throttle_hook(0.1)}
```

(continues on next page)

(continued from previous page)

```
s.get('http://httpbin.org/delay/get')
s.get('http://httpbin.org/delay/get')
```

See also:[example.py](#)

1.3 Persistence

`requests_cache` designed to support different backends for persistent storage. By default it uses `sqlite` database. Type of storage can be selected with `backend` argument of `install_cache()`.

List of available backends:

- `'sqlite'` - `sqlite` database (**default**)
- `'memory'` - not persistent, stores all data in Python dict in memory
- `'mongodb'` - (**experimental**) MongoDB database (`pymongo` required)

Note: `pymongo` doesn't work fine with `gevent` which powers `grequests`, but there is some workarounds, see question on [StackOverflow](#).

You can write your own and pass instance to `install_cache()` or `CachedSession` constructor. See [Cache backends](#) API documentation and sources.

1.4 Backward incompatible changes

There is backward incompatible changes introduced in version 0.3.0:

- `expire_after` is now seconds
- UTC time in cache
- Deleted `requests_cache.enabled()` context manager
- Storage backends are now using hash for keys
- Renamed methods in backends

For more information see [API reference](#) .

This part of the documentation covers all the interfaces of *requests-cache*

2.1 Public api

2.1.1 requests_cache.core

Core functions for configuring cache and monkey patching requests

```
class requests_cache.core.CachedSession(cache_name='cache', backend='sqlite', expire_after=None, allowable_codes=(200, ), allowable_methods=('GET', ), **backend_options)
```

Requests Sessions with caching support.

Parameters

- **cache_name** – for *sqlite* backend: cache file will start with this prefix, e.g *cache.sqlite* for *mongodb*: it's used as database name
- **backend** – cache backend name e.g *'sqlite'*, *'mongodb'*, *'memory'*. (see *Persistence*). Or instance of backend implementation.
- **expire_after** (*float*) – number of seconds after cache will be expired or *None* (default) to ignore expiration
- **allowable_codes** (*tuple*) – limit caching only for response with this codes (default: 200)
- **allowable_methods** (*tuple*) – cache only requests of this methods (default: *'GET'*)
- **backend_options** – options for chosen backend. See corresponding *sqlite* and *mongo* backends API documentation

```
send(request, **kwargs)
```

Send a given PreparedRequest.

Return type requests.Response

request (*method*, *url*, *params=None*, *data=None*, *headers=None*, *cookies=None*, *files=None*, *auth=None*, *timeout=None*, *allow_redirects=True*, *proxies=None*, *hooks=None*, *stream=None*, *verify=None*, *cert=None*)

Constructs a Request, prepares it and sends it. Returns Response object.

Parameters

- **method** – method for the new Request object.
- **url** – URL for the new Request object.
- **params** – (optional) Dictionary or bytes to be sent in the query string for the Request.
- **data** – (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the Request.
- **json** – (optional) json to send in the body of the Request.
- **headers** – (optional) Dictionary of HTTP Headers to send with the Request.
- **cookies** – (optional) Dict or CookieJar object to send with the Request.
- **files** – (optional) Dictionary of 'filename': file-like-objects for multi-part encoding upload.
- **auth** – (optional) Auth tuple or callable to enable Basic/Digest/Custom HTTP Auth.
- **timeout** (*float or tuple*) – (optional) How long to wait for the server to send data before giving up, as a float, or a (connect timeout, read timeout) tuple.
- **allow_redirects** (*bool*) – (optional) Set to True by default.
- **proxies** – (optional) Dictionary mapping protocol or protocol and hostname to the URL of the proxy.
- **stream** – (optional) whether to immediately download the response content. Defaults to False.
- **verify** – (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use. Defaults to True. When set to False, requests will accept any TLS certificate presented by the server, and will ignore hostname mismatches and/or expired certificates, which will make your application vulnerable to man-in-the-middle (MitM) attacks. Setting verify to False may be useful during local development or testing.
- **cert** – (optional) if String, path to ssl client cert file (.pem). If Tuple, ('cert', 'key') pair.

Return type requests.Response

cache_disabled()

Context manager for temporary disabling cache

```
>>> s = CachedSession()
>>> with s.cache_disabled():
...     s.get('http://httpbin.org/ip')
```

`requests_cache.core.install_cache` (*cache_name='cache'*, *backend='sqlite'*, *expire_after=None*, *allowable_codes=(200,)*, *allowable_methods=('GET',)*, *session_factory=<class 'requests_cache.core.CachedSession'>*, ***backend_options*)

Installs cache for all Requests requests by monkey-patching Session

Parameters are the same as in `CachedSession`. Additional parameters:

Parameters `session_factory` – Session factory. It should inherit `CachedSession` (default)

```
requests_cache.core.configure(cache_name='cache', backend='sqlite', expire_after=None,
                              allowable_codes=(200, ), allowable_methods=('GET', ), session_factory=<class 'requests_cache.core.CachedSession'>,
                              **backend_options)
```

Installs cache for all Requests requests by monkey-patching `Session`

Parameters are the same as in `CachedSession`. Additional parameters:

Parameters `session_factory` – Session factory. It should inherit `CachedSession` (default)

```
requests_cache.core.uninstall_cache()
Restores requests.Session and disables cache
```

```
requests_cache.core.disabled()
Context manager for temporary disabling globally installed cache
```

Warning: not thread-safe

```
>>> with requests_cache.disabled():
...     request.get('http://httpbin.org/ip')
...     request.get('http://httpbin.org/get')
```

```
requests_cache.core.get_cache()
Returns internal cache object from globally installed CachedSession
```

```
requests_cache.core.clear()
Clears globally installed cache
```

2.2 Cache backends

2.2.1 requests_cache.backends.base

Contains `BaseCache` class which can be used as in-memory cache backend or extended to support persistence.

class `requests_cache.backends.base.BaseCache(*args, **kwargs)`

Base class for cache implementations, can be used as in-memory cache.

To extend it you can provide dictionary-like objects for `keys_map` and `responses` or override public methods.

keys_map = `None`
key -> *key_in_responses* mapping

responses = `None`
key_in_cache -> *response* mapping

save_response (*key*, *response*)
Save response to cache

Parameters

- **key** – key for this response
- **response** – response to save

Note: Response is reduced before saving (with `reduce_response()`) to make it picklable

add_key_mapping (*new_key*, *key_to_response*)

Adds mapping of *new_key* to *key_to_response* to make it possible to associate many keys with single response

Parameters

- **new_key** – new key (e.g. url from redirect)
- **key_to_response** – key which can be found in `responses`

Returns

get_response_and_time (*key*, *default*=(None, None))

Retrieves response and timestamp for *key* if it's stored in cache, otherwise returns *default*

Parameters

- **key** – key of resource
- **default** – return this if *key* not found in cache

Returns tuple (response, datetime)

Note: Response is restored after unpickling with `restore_response()`

delete (*key*)

Delete *key* from cache. Also deletes all responses from response history

delete_url (*url*)

Delete response associated with *url* from cache. Also deletes all responses from response history. Works only for GET requests

clear ()

Clear cache

has_key (*key*)

Returns *True* if cache has *key*, *False* otherwise

has_url (*url*)

Returns *True* if cache has *url*, *False* otherwise. Works only for GET request urls

reduce_response (*response*)

Reduce response object to make it compatible with `pickle`

restore_response (*response*)

Restore response object after unpickling

2.2.2 requests_cache.backends.sqlite

sqlite3 cache backend

class requests_cache.backends.sqlite.DbCache (*location*='cache', *fast_save*=False, *extension*='.sqlite', ***options*)

sqlite cache backend.

Reading is fast, saving is a bit slower. It can store big amount of data with low memory usage.

Parameters

- **location** – database filename prefix (default: 'cache')
- **fast_save** – Speedup cache saving up to 50 times but with possibility of data loss. See [backends.DbDict](#) for more info
- **extension** – extension for filename (default: '.sqlite')

2.2.3 requests_cache.backends.mongo

mongo cache backend

```
class requests_cache.backends.mongo.MongoCache (db_name='requests-cache', **options)
    mongo cache backend.
```

Parameters

- **db_name** – database name (default: 'requests-cache')
 - **connection** – (optional) pymongo.Connection
-

2.3 Internal modules which can be used outside

2.3.1 requests_cache.backends.dbdict

Dictionary-like objects for saving large data sets to *sqlite* database

```
class requests_cache.backends.storage.dbdict.DbDict (filename, table_name='data',
                                                    fast_save=False)
```

DbDict - a dictionary-like object for saving large datasets to *sqlite* database

It's possible to create multiply DbDict instances, which will be stored as separate tables in one database:

```
d1 = DbDict('test', 'table1')
d2 = DbDict('test', 'table2')
d3 = DbDict('test', 'table3')
```

all data will be stored in `test.sqlite` database into correspondent tables: `table1`, `table2` and `table3`

Parameters

- **filename** – filename for database (without extension)
- **table_name** – table name
- **fast_save** – If it's True, then *sqlite* will be configured with “PRAGMA synchronous = 0;” to speedup cache saving, but be careful, it's dangerous. Tests showed that insertion order of records can be wrong with this option.

can_commit = None

Transactions can be committed if this property is set to *True*

commit (force=False)

Commits pending transaction if *can_commit* or *force* is *True*

Parameters **force** – force commit, ignore *can_commit*

bulk_commit ()

Context manager used to speedup insertion of big number of records

```
>>> d1 = DbDict('test')
>>> with d1.bulk_commit():
...     for i in range(1000):
...         d1[i] = i * 2
```

clear() → None. Remove all items from D.

```
class requests_cache.backends.storage.dbdict.DbPickleDict(filename, table_name='data',
                                                         fast_save=False)
```

Same as *DbDict*, but pickles values before saving

Parameters

- **filename** – filename for database (without extension)
- **table_name** – table name
- **fast_save** – If it's True, then sqlite will be configured with “PRAGMA synchronous = 0;” to speedup cache saving, but be careful, it's dangerous. Tests showed that insertion order of records can be wrong with this option.

2.3.2 requests_cache.backends.mongodict

Dictionary-like objects for saving large data sets to mongodb database

```
class requests_cache.backends.storage.mongodict.MongoDict(db_name, collection_name='mongo_dict_data',
                                                         connection=None)
```

MongoDict - a dictionary-like interface for mongo database

Parameters

- **db_name** – database name (be careful with production databases)
- **collection_name** – collection name (default: mongo_dict_data)
- **connection** – pymongo.Connection instance. If it's None (default) new connection with default options will be created

clear() → None. Remove all items from D.

```
class requests_cache.backends.storage.mongodict.MongoPickleDict(db_name, collection_name='mongo_dict_data',
                                                                connection=None)
```

Same as *MongoDict*, but pickles values before saving

Parameters

- **db_name** – database name (be careful with production databases)
- **collection_name** – collection name (default: mongo_dict_data)
- **connection** – pymongo.Connection instance. If it's None (default) new connection with default options will be created

CHAPTER 3

Indices and tables

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

r

`requests_cache.backends.base`, [9](#)
`requests_cache.backends.mongo`, [11](#)
`requests_cache.backends.sqlite`, [10](#)
`requests_cache.backends.storage.dbdict`,
 [11](#)
`requests_cache.backends.storage.mongodict`,
 [12](#)
`requests_cache.core`, [7](#)

A

`add_key_mapping()` (re-
quests_cache.backends.base.BaseCache
method), 10

B

BaseCache (class in *requests_cache.backends.base*), 9
`bulk_commit()` (re-
quests_cache.backends.storage.dbdict.DbDict
method), 11

C

`cache_disabled()` (re-
quests_cache.core.CachedSession *method*),
8

CachedSession (class in *requests_cache.core*), 7
`can_commit()` (*requests_cache.backends.storage.dbdict.DbDict*
attribute), 11

`clear()` (in module *requests_cache.core*), 9
`clear()` (*requests_cache.backends.base.BaseCache*
method), 10

`clear()` (*requests_cache.backends.storage.dbdict.DbDict*
method), 12

`clear()` (*requests_cache.backends.storage.mongodict.MongoDict*
method), 12

`commit()` (*requests_cache.backends.storage.dbdict.DbDict*
method), 11

`configure()` (in module *requests_cache.core*), 9

D

DbCache (class in *requests_cache.backends.sqlite*), 10
DbDict (class in re-
quests_cache.backends.storage.dbdict), 11

DbPickleDict (class in re-
quests_cache.backends.storage.dbdict), 12

`delete()` (*requests_cache.backends.base.BaseCache*
method), 10

`delete_url()` (*requests_cache.backends.base.BaseCache*
method), 10

`disabled()` (in module *requests_cache.core*), 9

G

`get_cache()` (in module *requests_cache.core*), 9
`get_response_and_time()` (re-
quests_cache.backends.base.BaseCache
method), 10

H

`has_key()` (*requests_cache.backends.base.BaseCache*
method), 10

`has_url()` (*requests_cache.backends.base.BaseCache*
method), 10

I

`install_cache()` (in module *requests_cache.core*),
8

K

`keys_map` (*requests_cache.backends.base.BaseCache*
attribute), 9

M

MongoCache (class in re-
quests_cache.backends.mongo), 11

MongoDict (class in re-
quests_cache.backends.storage.mongodict),
12

MongoPickleDict (class in re-
quests_cache.backends.storage.mongodict),
12

R

`reduce_response()` (re-
quests_cache.backends.base.BaseCache
method), 10

`request()` (*requests_cache.core.CachedSession*
method), 8

requests_cache.backends.base (module), 9

`requests_cache.backends.mongo` (*module*), 11
`requests_cache.backends.sqlite` (*module*), 10
`requests_cache.backends.storage.dbdict` (*module*), 11
`requests_cache.backends.storage.mongodict` (*module*), 12
`requests_cache.core` (*module*), 7
`responses` (*requests_cache.backends.base.BaseCache* attribute), 9
`restore_response()` (*requests_cache.backends.base.BaseCache* method), 10

S

`save_response()` (*requests_cache.backends.base.BaseCache* method), 9
`send()` (*requests_cache.core.CachedSession* method), 7

U

`uninstall_cache()` (*in module requests_cache.core*), 9