Nameko Cache Tools Documentation

Release 1.0.0

Santiago Suarez Ordonez

Jun 11, 2018

Contents

A few tools to cache interactions between your nameko services, increasing resiliency and performance at the expense of consistency, when it makes sense.

To use nameko-cachetools in a project:

```
from nameko.rpc import rpc
from nameko_cachetools import CachedRpcProxy


class Service(object):
    name = "demo"

    other_service = CachedRpcProxy('other_service')

    @rpc
    def do_something(self, request):
        # this rpc response will be cached, further queries will be
        # timed and cached values will be returned if not response is
        # received or an exception is raised at the destination service
        other_service.do_something('hi')
```

To use a more advanced cache from the cachetools module:

```
from nameko.rpc import rpc
from nameko_cachetools import CachedRpcProxy
from cachetools import TTLCache


class Service(object):
    name = "demo"

    # use a TTL cache that will only hold 1024 different rpc interactions
    # and expire them afer 30 seconds
    other_service = CachedRpcProxy('other_service', cache=TTLCache(1024, 30))

    @rpc
    def do_something(self, request):
        # this rpc response will be cached. For the next 30 seconds,
        # further queries will not reach the target service but still
        # return the cached response
        other_service.do_something('hi')
```

CHAPTER 1

Caching strategies:

1.1 CachedRpcProxy

If a cached version of this request exists, a response from the cache is sent instead of hanging forever or raising an exception.

If a cached version doesn't exist, it will behave like a normal rpc, and wait indefinitey for a reply. All successful replies are cached.

WARNING: Do NOT use this for setters, rpcs meant to modify state in the target service

Arguments:

cache the cache to use. This should resemble a dict but can be more sophisticated, like the caches provided by the cachetools package.

failover_timeout if a cached version of this query exists, how long in seconds should your original request wait until it deems the target service as unresponsive and moves on to use a cached response

1.2 CacheFirstRpcProxy

Stores responses from the original services and keeps them cached.

If further requests come in with the same arguments and found in the cache, a response from the cache is sent instead of hitting the destination service.

WARNING: Do NOT use this for setters, rpcs meant to modify state in the target service

Arguments:

cache the cache to use. This should resemble a dict but can be more sophisticated, like the caches provided by the cachetools package.

1.2.1 Contents:

Installation

At the command line either via easy_install or pip:

```
$ easy_install nameko-cachetools
$ pip install nameko-cachetools
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv nameko-cachetools
$ pip install nameko-cachetools
```

Usage

To use nameko-cachetools in a project:

```
from nameko.rpc import rpc
from nameko_cachetools import CachedRpcProxy


class Service(object):
    name = "demo"

    other_service = CachedRpcProxy('other_service')

    @rpc
    def do_something(self, request):
        # this rpc response will be cached, further queries will be
        # timed and cached values will be returned if not response is
        # received or an exception is raised at the destination service
        other_service.do_something('hi')
```

To use a more advanced cache from the cachetools module:

```
from nameko.rpc import rpc
from nameko_cachetools import CachedRpcProxy
from cachetools import TTLCache


class Service(object):
    name = "demo"

    # use a TTL cache that will only hold 1024 different rpc interactions
    # and expire them afer 30 seconds
    other_service = CachedRpcProxy('other_service', cache=TTLCache(1024, 30))

    @rpc
    def do_something(self, request):
        # this rpc response will be cached. For the next 30 seconds,
        # further queries will not reach the target service but still
        # return the cached response
        other_service.do_something('hi')
```

Caching strategies:

CachedRpcProxy

If a cached version of this request exists, a response from the cache is sent instead of hanging forever or raising an exception.

If a cached version doesn't exist, it will behave like a normal rpc, and wait indefinitey for a reply. All successful replies are cached.

WARNING: Do NOT use this for setters, rpcs meant to modify state in the target service

Arguments:

cache the cache to use. This should resemble a dict but can be more sophisticated, like the caches provided by the cachetools package.

failover_timeout if a cached version of this query exists, how long in seconds should your original request wait until it deems the target service as unresponsive and moves on to use a cached response

CacheFirstRpcProxy

Stores responses from the original services and keeps them cached.

If further requests come in with the same arguments and found in the cache, a response from the cache is sent instead of hitting the destination service.

WARNING: Do NOT use this for setters, rpcs meant to modify state in the target service

Arguments:

cache the cache to use. This should resemble a dict but can be more sophisticated, like the caches provided by the cachetools package.

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:

Types of Contributions

Report Bugs

Report bugs at https://github.com/santiycr/nameko-cachetools/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.

Fix Bugs

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

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

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

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/santiycr/nameko-cachetools/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 :)

Get Started!

Ready to contribute? Here's how to set up *nameko-cachetools* for local development.

1. Fork the *nameko-cachetools* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/nameko-cachetools.git
```

3. Create a branch for local development:

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

Now you can make your changes locally.

4. When you're done making changes, check that your changes pass style and unit tests, including testing other Python versions with tox:

```
$ tox
```

To get tox, just pip install it.

5. 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
```

6. Submit a pull request through the GitHub website.

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 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/santiycr/nameko-cachetools under pull requests for active pull requests or run the `tox` command and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test test/test_nameko_cachetools.py
```

Credits

Development Lead

- Santiago Suarez Ordonez <santiycr@gmail.com>

Contributors

None yet. Why not be the first?

History

0.1.0 (2018-06-10)

- First release on PyPI.

1.2.2 Feedback

If you have any suggestions or questions about **Nameko Cache Tools** feel free to email me at santiycr@gmail.com.

If you encounter any errors or problems with **Nameko Cache Tools**, please let me know! Open an Issue at the GitHub http://github.com/santiycr/nameko-cachetools main repository.