Context loop Documentation

Release 0.1.1

Paweł Zadrożny

Mar 19, 2018

Contents:

Simple context manager utility for asyncio event loop. Context loop helps with async pieces of code to be scheduled and run within synchronous code.

Can be used with synchronous and asynchronous frameworks like Django, Flask or Tornado and Twisted.

CHAPTER 1

Context loop

Info Context loop.

Author Paweł Zadrożny @pawelzny <pawel.zny@gmail.com>

1.1 Features

- Work with sync and async frameworks
- Schedule tasks to existing loop or create new one
- No need to understand how async works
- No callbacks required
- Run async tasks whenever and wherever you want

1.2 Installation

```
pip install context-loop
```

Package: https://pypi.org/project/context-loop/

1.3 Documentation

Read full documentation at http://context-loop.readthedocs.io/en/stable/

1.4 Quick Example

```
>>> async def coro():
...     return await something_from_future()
...
>>> import cl.Loop
>>> with cl.Loop(coro(), coro(), coro()) as loop:
...    result = loop.run_until_complete()
...
>>> result
['success', 'success', 'success']
```

CHAPTER 2

Public API

class cl.loop.**Loop**(**futures*, *loop=None*, *return_exceptions=False*)

Asyncio Event loop context manager.

Context manager which get existing event loop or if none exist will create new one.

All coroutines are converted to task and scheduled to execute in near future. Scheduling is safe for long running tasks.

Example

Create coroutine using *@asyncio.coroutine* decorator or with async/await syntax

```
>>> async def wait_for_it(timeout):
...     await asyncio.sleep(timeout)
...     return 'success sleep for {} seconds'.format(timeout)
...
```

Use context manager to get result from one or more coroutines

```
>>> with Loop(wait_for_it(5), wait_for_it(3), return_exceptions=True) as loop:
...     result = loop.run_until_complete()
...
>>> result
['success sleep for 3 seconds', 'success sleep for 5 seconds']
```

When single coroutine has been scheduled to run, only single value will be returned.

```
>>> with Loop(wait_for_it(4)) as loop:
...     result = loop.run_until_complete()
...
>>> result
'success sleep for 4 seconds'
```

Parameters

- **futures** (*asyncio.Future, asyncio.coroutine*) – One or more coroutine or future.
- **loop** (*asyncio.AbstractEventLoop*) – Optional existing loop.
- **return_exceptions** (*Boolean*) – If True will return exceptions as result.
- **stop_when_done** (*Boolean*) – If True will close the loop on context exit.

futures = None
Gathered futures.

gather(**futures: typing.Union[asyncio.futures.Future, <function coroutine at 0x7f3b33e276a8>]*)
Gather list of futures/coros and return single Task ready to schedule.

Example

Prepare all futures to execution

```
>>> async def do_something():
...     return 'something'
...
>>> async def do_something_else():
...     return 'something_else'
...
```

Gather all tasks and then pass to context loop

```
>>> loop = Loop(return_exceptions=True)
>>> loop.gather(do_something(), do_something_else())
>>> with loop as l:
...     result = l.run_until_complete()
...
```

Parameters **futures** (*asyncio.Future, asyncio.coroutine*) – One or more coroutine or future.

Returns Futures grouped into single future

Return type asyncio.Task, asyncio.Future

run_until_complete()
Run loop until all futures are done.

Schedule futures for execution and wait until all are done. Return value from future, or list of values if multiple futures had been passed to constructor or gather method.

All results will be in the same order as order of futures passed to constructor.

Example

```
>>> async def slow():
...     await ultra_slow_task()
...     return 'ultra slow'
...
>>> async def fast():
...     await the_fastest_task_on_earth()
...
>>> with Loop(slow(), fast()) as loop:
...     result = loop.run_until_complete()
...
```

```
>>> result
['ultra slow', None]
```

Returns Value from future or list of values.

Return type None, list, Any

cancel()

Cancel pending futures.

If any of futures are already done its result will be lost. Result of loop execution will be None.

Example

```
>>> async def nuke_loop():
...     loop.cancel()
...
>>> loop = Loop()
>>> loop.gather(nuke_loop())
>>> with loop as lo:
...     result = lo.run_until_complete()
...
>>> result
None
```

CHAPTER 3

Credits

3.1 Development

- Paweł Zadrożny @pawelzny <pawel.zny@gmail.com>

3.2 Contributors

None yet. Why not be the first?

Read more how to contribute on *Contributing*.

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/pawelzny/context-loop/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” is open to whoever wants to implement it.

4.1.3 Implement Features

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

4.1.4 Write Documentation

authentication could always use more documentation, whether as part of the official authentication 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/pawelzny/context-loop/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 *context-loop* for local development.

1. Fork the *context-loop* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/context-loop.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 context-loop/
$ make install-dev
```

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:

```
$ make test-all
```

   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 and for PyPy3.5. Check https://circleci.com/gh/pawelzny/context-loop and make sure that the tests pass for all supported Python versions.

CHAPTER 5

LICENSE

MIT License

Copyright (c) 2018, Paweł Zadrożny @pawelzny <pawel.zny@gmail.com>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Python Module Index

C

Index