uvloop Documentation

Release 0.4.15

Yury Selivanov

May 09, 2016

Contents

uvloop is a fast, drop-in replacement of the built-in asyncio event loop. *uvloop* is released under the MIT license.

uvloop and asyncio, combined with the power of async/await in Python 3.5, makes it easier than ever to write high-performance networking code in Python.

uvloop makes asyncio fast. In fact, it is at least 2x faster than nodejs, gevent, as well as any other Python asynchronous framework. The performance of uvloop-based asyncio is close to that of Go programs.

You can read more about uvloop in this blog post.

CHAPTER 1

Architecture

The asyncio module, introduced by PEP 3156, is a collection of network transports, protocols, and streams abstractions, with a pluggable event loop. The event loop is the heart of asyncio. It provides APIs for:

- scheduling calls,
- transmitting data over the network,
- performing DNS queries,
- handling OS signals,
- convenient abstractions to create servers and connections,
- working with subprocesses asynchronously.

uvloop implements the `asyncio.AbstractEventLoop` interface which means that it provides a drop-in replacement of the asyncio event loop.

uvloop is written in Cython and is built on top of libuv.

libuv is a high performance, multiplatform asynchronous I/O library used by nodejs. Because of how wide-spread and popular nodejs is, libuv is fast and stable.

uvloop implements all asyncio event loop APIs. High-level Python objects wrap low-level libuv structs and functions. Inheritance is used to keep the code DRY and ensure that any manual memory management is in sync with libuv primitives' lifespans.

CHAPTER 2

Contents

2.1 User Guide

This section of the documentation provides information about how to use uvloop.

2.1.1 Installation

uvloop is available from PyPI. It requires Python 3.5.

Use pip to install it.

```
$ pip install uvloop
```

2.1.2 Using uvloop

To make asyncio use the event loop provided by *uvloop*, you install the *uvloop* event loop policy:

```
import asyncio
import uvloop
asyncio.set_event_loop_policy(uvloop.EventLoopPolicy())
```

Alternatively, you can create an instance of the loop manually, using:

```
import asyncio
import uvloop
loop = uvloop.new_event_loop()
asyncio.set_event_loop(loop)
```

2.2 Developers Guide

The project is hosted on GitHub. and uses Travis for Continuous Integration.

A goal for the *uvloop* project is to provide a drop in replacement for the *asyncio* event loop. Any deviation from the behavior of the reference *asyncio* event loop is considered a bug.

If you have found a bug or have an idea for an enhancement that would improve the library, use the bug tracker.

2.2.1 Get the source

```
$ git clone --recursive git@github.com:MagicStack/uvloop.git
```

The `--recursive` argument is important. It will fetch the `libuv` source from the *libuv* Github repository.

2.2.2 Build

To build *uvloop*, you'll need `Cython` and Python 3.5.

Note: The best way to work on *uvloop* is to create a virtual env, so that you'll have Cython and Python commands pointing to the correct tools.

```
$ python3 -m venv myvenv
$ cd myvenv
$ source bin/activate
$ cd ..
```

Install Cython if not already present.

```
$ pip install Cython
```

Build *uvloop* by running the `make` rule from the top level directory.

```
$ cd uvloop
$ make
```

2.2.3 Test

The easiest method to run all of the unit tests is to run the `make test` rule from the top level directory. This runs the standard library `unittest` tool which discovers all the unit tests and runs them. It actually runs them twice, once with the *PYTHONASYNCIODEBUG* enabled and once without.

```
$ cd uvloop
$ make test
```

Individual Tests

Individual unit tests can be run using the standard library `unittest` or `pytest` package.

The easiest approach to ensure that `uvloop` can be found by Python is to install the package using `pip`:

```
$ cd uvloop
$ pip install -e .
```

You can then run the unit tests individually from the tests directory using `unittest`:

```
$ cd uvloop/tests
$ python -m unittest test_tcp
```

or using `pytest`:

```
$ cd uvloop/tests
$ py.test -k test_signals_sigint_uvcode
```

2.2.4 Documentation

To rebuild the project documentation, developers should run the `make docs` rule from the top level directory. It performs a number of steps to create a new set of sphinx html content.

This step requires Sphinx to be installed. Sphinx can be installed using pip:

```
$ pip install sphinx
```

Once Sphinx is available you can make the documentation using:

```
$ make docs
```

2.3 API

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.3.1 uvloop