aioeventlet Documentation

Release 0.5.1

Victor Stinner

September 12, 2016

Contents

aioeventlet implements the asyncio API (PEP 3156) on top of eventlet. It makes possible to write asyncio code in a project currently written for eventlet.

aioeventlet allows to use greenthreads in asyncio coroutines, and to use asyncio coroutines, tasks and futures in greenthreads: see *yield_future()* and *wrap_greenthread()* functions.

The main visible difference between aioeventlet and asyncio is the behaviour of `run_forever()`: `run_forever()` blocks with asyncio, whereas it runs in a greenthread with aioeventlet. It means that aioeventlet event loop can run in an greenthread while the Python main thread runs other greenthreads in parallel.

- aioeventlet documentation
- aioeventlet project in the Python Cheeseshop (PyPI)
- aioeventlet project at Bitbucket
- Copyright/license: Open source, Apache 2.0. Enjoy!

CHAPTER 1

Table Of Contents

1.1 Usage

1.1.1 Use aioeventlet with asyncio

aioeventlet can be used with asyncio, coroutines written with `yield from ...`. To use aioeventlet with asyncio, set the event loop policy before using an event loop. Example:

```
import aioeventlet
import asyncio

asyncio.set_event_loop_policy(aioeventlet.EventLoopPolicy())
# ....
```

Setting the event loop policy should be enough to examples of the asyncio documentation with the aioeventlet event loop.

Hello World:

```
import aioeventlet
import asyncio

def hello_world():
    print("Hello World")
    loop.stop()

asyncio.set_event_loop_policy(aioeventlet.EventLoopPolicy())
loop = asyncio.get_event_loop()
loop.call_soon(hello_world)
loop.run_forever()
loop.close()
```

See also:

The asyncio documentation.

1.1.2 Use aioeventlet with trollius

Warning: The trollius project is now deprecated. It's now recommended to use aioeventlet with asyncio.

aioeventlet can be used with trollius, coroutines written with `yield From(...)`. Using aioeventlet with trollius is a good start to port project written for eventlet to trollius.

To use aioeventlet with trollius, set the event loop policy before using an event loop, example:

```
import aioeventlet
import trollius

trollius.set_event_loop_policy(aioeventlet.EventLoopPolicy())
# ....
```

Hello World:

```
import aioeventlet
import trollius as asyncio

def hello_world():
    print("Hello World")
    loop.stop()

asyncio.set_event_loop_policy(aioeventlet.EventLoopPolicy())
loop = asyncio.get_event_loop()
loop.call_soon(hello_world)
loop.run_forever()
loop.close()
```

See also:

Trollius documentation.

1.1.3 Threads

Running an event loop in a thread different than the main thread is currently experimental.

An eventlet Event object is not thread-safe, it must only be used in the same thread. Use threading.Event to signal events between threads, and threading.Queue to pass data between threads.

Use `threading = eventlet.patcher.original('threading')` to get the original threading instead of `import threading`.

It is not possible to run two aioeventlet event loops in the same thread.

1.1.4 Debug mode

To enable the debug mode globally when using trollius, set the environment variable `TROLLIUSDEBUG` to `1`. To see debug traces, set the log level of the trollius logger to `logging.DEBUG`. The simplest configuration is:

```
import logging
# ...
logging.basicConfig(level=logging.DEBUG)
```

If you use asyncio, use the `PYTHONASYNCIODEBUG` environment variable instead of the `TROLLIUSDEBUG` variable.

You can also call `loop.set_debug(True)` to enable the debug mode of the event loop, but it enables less debug checks.

See also:

Read the Develop with asyncio section of the asyncio documentation.

1.2 API

aioeventlet specific functions:

Warning: aioeventlet API is not considered as stable yet.

1.2.1 yield_future

yield_future(*future*, *loop=None*)

Wait for a future, a task, or a coroutine object from a greenthread.

Return the result or raise the exception of the future.

The function must not be called from the greenthread running the aioeventlet event loop.

Changed in version 0.4: Rename the function from `wrap_future()` to *`yield_future()`*.

Changed in version 0.3: Coroutine objects are also accepted. Added the *loop* parameter. An exception is raised if it is called from the greenthread of the aioeventlet event loop.

Example of greenthread waiting for a trollius task. The `progress()` callback is called regulary to see that the event loop in not blocked:

```
import aioeventlet
import eventlet
import trollius as asyncio
from trollius import From, Return

def progress():
    print("computation in progress...")
    loop.call_later(0.5, progress)

@asyncio.coroutine
def coro_slow_sum(x, y):
    yield From(asyncio.sleep(1.0))
    raise Return(x + y)

def green_sum():
    loop.call_soon(progress)

    value = aioeventlet.yield_future(coro_slow_sum(1, 2))
    print("1 + 2 = %s" % value)

    loop.stop()

asyncio.set_event_loop_policy(aioeventlet.EventLoopPolicy())
eventlet.spawn(green_sum)
loop = asyncio.get_event_loop()
loop.run_forever()
loop.close()
```

Output:

```
computation in progress...
computation in progress...
computation in progress...
1 + 2 = 3
```

1.2.2 wrap_greenthread

wrap_greenthread(*gt*)

Wrap an eventlet GreenThread, or a greenlet, into a Future object.

The Future object waits for the completion of a greenthread. The result or the exception of the greenthread will be stored in the Future object.

The greenthread must be wrapped before its execution starts. If the greenthread is running or already finished, an exception is raised.

For greenlets, the `run` attribute must be set.

Changed in version 0.3: An exception is now raised if the greenthread is running or already finished. In debug mode, the exception is not more logged to sys.stderr for greenthreads.

Example of trollius coroutine waiting for a greenthread. The `progress()` callback is called regulary to see that the event loop in not blocked:

```
import aioeventlet
import eventlet
import trollius as asyncio
from trollius import From, Return

def progress():
    print("computation in progress...")
    loop.call_later(0.5, progress)

def slow_sum(x, y):
    eventlet.sleep(1.0)
    return x + y

@asyncio.coroutine
def coro_sum():
    loop.call_soon(progress)

    gt = eventlet.spawn(slow_sum, 1, 2)
    fut = aioeventlet.wrap_greenthread(gt, loop=loop)

    result = yield From(fut)
    print("1 + 2 = %s" % result)

asyncio.set_event_loop_policy(aioeventlet.EventLoopPolicy())
loop = asyncio.get_event_loop()
loop.run_until_complete(coro_sum())
loop.close()
```

Output:

```
computation in progress...
computation in progress...
computation in progress...
1 + 2 = 3
```

1.3 Installation

1.3.1 Install aioeventlet with pip

Type:

```
pip install aioeventlet
```

1.3.2 Install aioeventlet on Windows with pip

Procedure for Python 2.7:

- If pip is not installed yet, install pip: download `get-pip.py` and type:

```
\Python27\python.exe get-pip.py
```

- Install aioeventlet with pip:

```
\Python27\python.exe -m pip install aioeventlet
```

- pip also installs dependencies: `eventlet` and `trollius`

1.3.3 Manual installation of aioeventlet

Requirements:

- eventlet 0.14 or newer
- asyncio or trollius:
  - Python 3.4 and newer: asyncio is now part of the stdlib (only eventlet is needed)
  - Python 3.3: need Tulip 0.4.1 or newer (`pip install asyncio`), but Tulip 3.4.1 or newer is recommended
  - Python 2.7: need Trollius 0.3 or newer (`pip install trollius`), but Trollius 1.0 or newer is recommended

Type:

```
python setup.py install
```

1.4 Run tests

1.4.1 Run tests with tox

The tox project can be used to build a virtual environment with all runtime and test dependencies and run tests against different Python versions (2.7, 3.3, 3.4, 3.5).

To test all Python versions, just type:

```
tox
```

To run tests with Python 2.7, type:

```
tox -e py27
```

To run tests against other Python versions:

- `py27`: Python 2.7
- `py27_patch`: Python 2.7 with eventlet monkey patching
- `py27_old`: Python 2.7 with the oldest supported versions of eventlet and trollius
- `py33`: Python 3.3
- `py3_patch`: Python 3 with eventlet monkey patching
- `py3_old`: Python 3 with the oldest supported versions of eventlet and tulip
- `py34`: Python 3.4
- `py35`: Python 3.5

1.4.2 Run tests manually

To run unit tests, the `mock` module is need on Python older than 3.3.

Run the following command:

```
python runtests.py -r
```

1.5 To do

- support monkey-patching enabled after loading the aioeventlet module
- register signals in eventlet hub, only needed for pyevent hub?
- port greenio examples to aioeventlet
- write unit tests for, and maybe also examples for:
  - TCP server
  - UDP socket
  - UNIX socket
  - pipes
  - signals
  - subprocesses
- experiment running an event loop in a thread different than the main thread

1.6 eventlet and Python 3

eventlet 0.17 or newer is recommanded for Python 3 when monkey-patching is enabled.

1.7 asyncio in OpenStack

Warning: The project of replacing eventlet with trollius using aioeventlet in OpenStack is abandonned. It might done "later" when Python 2 support will be removed from OpenStack which is not going to happen in a near future.

1.7.1 First part (in progress): add support for trollius coroutines

Prepare OpenStack (Oslo Messaging) to support trollius coroutines using `yield`: explicit asynchronous programming. Eventlet is still supported, used by default, and applications and libraries don't need to be modified at this point.

Already done:

- Write the trollius project: port asyncio to Python 2
- Stabilize trollius API
- Add trollius dependency to OpenStack
- Write the aioeventlet project to provide the asyncio API on top of eventlet

To do:

- Stabilize aioeventlet API
- Add aioeventlet dependency to OpenStack
- Write an aioeventlet executor for Oslo Messaging: rewrite greenio executor to replace greenio with aioeventlet

1.7.2 Second part (to do): rewrite code as trollius coroutines

Switch from implicit asynchronous programming (eventlet using greenthreads) to explicit asynchronous programming (trollius coroutines using `yield`). Need to modify OpenStack Common Libraries and applications. Modifications can be done step by step, the switch will take more than 6 months.

The first application candidate is Ceilometer. The Ceilometer project is young, developers are aware of eventlet issues and like Python 3, and Ceilometer don't rely so much on asynchronous programming: most time is spent into waiting the database anyway.

The goal is to port Ceilometer to explicit asynchronous programming during the cycle of OpenStack Kilo.

Some applications may continue to use implicit asynchronous programming. For example, nova is probably the most complex part beacuse it is and old project with a lot of legacy code, it has many drivers and the code base is large.

To do:

- Ceilometer: add trollius dependency and set the trollius event loop policy to aioeventlet
- Ceilometer: change Oslo Messaging executor from "eventlet" to "aioeventlet"
- Redesign the service class of Oslo Incubator to support aioeventlet and/or trollius. Currently, the class is designed for eventlet. The service class is instanciated before forking, which requires hacks on eventlet to update file descriptors.
- In Ceilometer and its OpenStack depedencencies: add new functions which are written with explicit asynchronous programming in mind (ex: trollius coroutines written with `yield`).
- Rewrite Ceilometer endpoints (RPC methods) as trollius coroutines.

Questions:

- What about WSGI? aiohttp is not compatible with trollius yet.
- The quantity of code which need to be ported to asynchronous programming is unknown right now.
- We should be prepared to see deadlocks. OpenStack was designed for eventlet which implicitly switch on blocking operations. Critical sections may not be protected with locks, or not the right kind of lock.
- For performances, blocking operations can be executed in threads. OpenStack code is probably not thread-safe, which means new kinds of race conditions. But the code executed in threads will be explicitly scheduled to be executed in a thread (with `loop.run_in_executor()`), so regressions can be easily identified.
- This part will take a lot of time. We may need to split it into subparts to have milestones, which is more attractive for developers.

1.7.3 Last part (to do): drop eventlet

Replace aioeventlet event loop with trollius event loop, drop aioeventlet and drop eventlet at the end.

This change will be done on applications one by one. This is no need to port all applications at once. The work will start on Ceilometer, as a follow up of the second part.

To do:

- Port remaining code to trollius
- Write a "trollius" executor for Oslo Messaging
- Ceilometer: Add a blocking call to `loop.run_forever()` in the `main()` function
- Ceilometer: Replace "aioeventlet" executor with "trollius" executor
- Ceilometer: Use the standard trollius event loop policy
- Ceilometer: drop the eventlet dependency

Questions:

- Putting a blocking call to `loop.run_forever()` may need to redesign Ceilometer, this part is unclear to me right now.

Optimization, can be done later:

- Oslo Messaging: watch directly the underlying file descriptor of sockets, instead of using a busy loop polling the notifier
- Ceilometer: use libraries supporting directly trollius to be able to run parallel tasks (ex: send multiple requests to a database)

1.7.4 openstack-dev mailing list

- [oslo] Progress of the port to Python 3 (Victor Stinner, Jan 6 2015)
- [oslo] Add a new aiogreen executor for Oslo Messaging (Victor Stinner, Nov 23 2014)
- [oslo] Asyncio and oslo.messaging (Mark McLoughlin, Jul 3 2014)
  - SQLAlchemy and asynchronous programming by Mike Bayer (author and maintainer of SQLAlchemy)
- [Solum][Oslo] Next Release of oslo.messaging? (Victor Stinner, Mar 18 2014)
- [solum] async / threading for python 2 and 3 (Victor Stinner, Feb 20 2014)

- Asynchrounous programming: replace eventlet with asyncio (Victor Stinner, Feb 4 2014)

1.7.5 History of asyncio in OpenStack

Threads and asyncio specs:

- Cross-project meeting: 2015-03-02
- Cross-project meeting: 2015-02-24T21:44:05

Maybe the good one, aioeventlet project:

- March 13, 2015: Joshua Harlow wrote the spec Replace eventlet + monkey-patching with ??
- February 17, 2015: Joshua Harlow wrote a different spec, Sew over eventlet + patching with threads
- February 5, 2015: new cross-project spec Replace eventlet with asyncio
- December 3, 2014: two patches posted to requirements: Add aioeventlet dependency and Drop greenio dependency.
- Novembre 23, 2014: two patches posted to Oslo Messaging: Add a new aioeventlet executor and Add an optional executor callback to dispatcher
- November 19, 2014: First release of the aioeventlet project

OpenStack Kilo Summit, November 3-7, 2014, at Paris:

- Python 3 in Oslo:
  - add a new greenio executor to Oslo Messaging
  - port eventlet to Python 3 (with monkey-patching): see the *status of the eventlet port to Python 3*
- What should we do about oslo.messaging?: add the new greenio executor
- Python 3.4 transition

New try with a greenio executor for Oslo Messaging:

- July 29, 2014: Doug Hellmann proposed the blueprint A 'greenio' executor for oslo.messaging, approved by Mark McLoughlin.
- July 24, 2014: Add greenio dependency merged into openstack/requirements
- July 22, 2014: Patch Add a new greenio executor proposed to Oslo Messaging
- July 21, 2014: Release of greenio 0.6 which is now compatible with Trollius
- July 21, 2014: Release of Trollius 1.0
- July 14, 2014: Patch Add a 'greenio' oslo.messaging executor (spec) merged into openstack/oslo-specs.
- July 7, 2014: Patch Fix AMQPListener for polling with timeout merged into Oslo Messaging
- July 2014: greenio executor, [openstack-dev] [oslo] Asyncio and oslo.messaging

First try with a trollius executor for Oslo Messaging:

- June 20, 2014: Patch Add an optional timeout parameter to Listener.poll merged into Oslo Messaging
- May 28, 2014: Meeting at OpenStack in action with Doug Hellman, Julien Danjou, Mehdi Abaakouk, Victor Stinner and Christophe to discuss the plan to port OpenStack to Python 3 and switch from eventlet to asyncio.
- April 23, 2014: Patch Allow trollius 0.2 merged into openstack/requirements
- March 21, 2014: Patch Replace ad-hoc coroutines with Trollius coroutines proposed to Heat. Heat coroutines are close to Trollius coroutines. Patch abandonned, need to be rewritten, maybe with aioeventlet.

- February 20, 2014: The full specification of the blueprint was written: Oslo/blueprints/asyncio
- February 8, 2014: Patch Add a new dependency: trollius merged into openstack/requirements
- February 27, 2014: Article Use the new asyncio module and Trollius in OpenStack published
- February 4, 2014: Patch Add a new asynchronous executor based on Trollius proposed to Oslo Messaging, but it was abandonned. Running a classic Trollius event loop in a dedicated thread doesn't fit well into eventlet event loop.

First discussion around asyncio and OpenStack:

- December 19, 2013: Article Why should OpenStack move to Python 3 right now? published
- December 4, 2013: Blueprint Add a asyncio executor to oslo.messaging proposed by Flavio Percoco and accepted for OpenStack Icehouse by Mark McLoughlin

1.7.6 History of asynchronous programming in OpenStack

In the past, the Nova project used Tornado, then Twisted and it is now using eventlet which also became the defacto standard in OpenStack

1.8 Changelog

1.8.1 2016-02-22: Version 0.5.1

- Fix EventLoop.stop() for the new semantics of stop() introduced in Python 3.4.4 and Python 3.5.1. The method now writes into the self-pipe to wake-up the event loop. Otherwise, the stop() may not be taken in account immediatly, and even block forever.

1.8.2 2016-02-22: Version 0.5

- Unit tests now use the aiotest library.
- Fix for eventlet used with monkey-patching: inject the original threading module and the original threading.get_ident() function in asyncio.base_events.
- Drop support for Python 2.6 and Python 3.2. aioeventlet depends on trollius and pip which don't support these Python versions anymore.

1.8.3 2014-12-03: Version 0.4

- Add run_aiotest.py
- tox now also run the aiotest test suite
- Rename the project from `aiogreen` to `aioeventlet`
- Rename the `link_future()` function to *`yield_future()`*
- Running tests with eventlet monkey-patching now works with Python 3.

1.8.4 2014-11-23: version 0.3

- *wrap_greenthread()* now raises an exception if the greenthread is running or already finished. In debug mode, the exception is not more logged to sys.stderr for greenthreads.
- link_future() now accepts coroutine objects.
- link_future() now raises an exception if it is called from the greenthread of the aiogreen event loop.
- Fix eventlet detection of blocking tasks: cancel the alarm when the aiogreen event loop stops.
- Fix to run an event loop in a thread different than the main thread in debug mode: disable eventlet "debug_blocking", it is implemented with the SIGALRM signal, but signal handlers can only be set in the main thread.

1.8.5 2014-11-21: version 0.2

aiogreen event loop has been rewritten to reuse more asyncio/trollius code. It now only has a few specific code.

It is now possible use greenthreads in asyncio coroutines, and to use asyncio coroutines, tasks and futures in greenthreads: see link_future() and *wrap_greenthread()* functions.

All asyncio network are supported: TCP, UCP and UNIX clients and servers.

Support of pipes, signal handlers and subprocess is still experimental.

Changes:

- Add a Sphinx documentation published at http://aiogreen.readthedocs.org/
- Add the link_future() function: wait for a future from a greenthread.
- Add the *wrap_greenthread()* function: wrap a greenthread into a Future
- Support also eventlet 0.14, not only eventlet 0.15 or newer
- Support eventlet with monkey-patching
- Rewrite the code handling file descriptors to ensure that the listener is only called once per loop iteration, to respect asyncio specification.
- Simplify the loop iteration: remove custom code to reuse instead the asyncio/trollius code (_run_once)
- Reuse call_soon, call_soon_threadsafe, call_at, call_later from asyncio/trollius, remove custom code
- sock_connect() is now asynchronous
- Add a suite of automated unit tests
- Fix EventLoop.stop(): don't stop immediatly, but schedule stopping the event loop with call_soon()
- Add tox.ini to run tests with tox
- Setting debug mode of the event loop doesn't enable "debug_blocking" of eventlet on Windows anymore, the feature is not implemented on Windows in eventlet.
- add_reader() and add_writer() now cancels the previous handle and sets a new handle
- In debug mode, detect calls to call_soon() from greenthreads which are not threadsafe (would not wake up the event loop).
- Only set "debug_exceptions" of the eventlet hub when the debug mode of the event loop is enabled.

1.8.6 2014-11-19: version 0.1

- First public release

CHAPTER 2

Event loops

Projects used by aioeventlet:

- asyncio documentation
- trollius documentation
- tulip project
- eventlet documentation
- eventlet project
- greenlet documentation

See also:

- aiogevent: asyncio API implemented on top of gevent
- geventreactor: gevent-powered Twisted reactor
- greenio: Greenlets support for asyncio (PEP 3156)
- tulipcore: run gevent code on top of asyncio

W

Y