pytest-asyncio

Release v0.20.1

Tin Tvrtković

May 01, 2023

CONTENTS

CHAPTER

ONE

CONCEPTS

1.1 asyncio event loops

pytest-asyncio runs each test item in its own asyncio event loop. The loop can be accessed via the `event_loop` fixture, which is automatically requested by all async tests.

```
async def test_provided_loop_is_running_loop(event_loop):
    assert event_loop is asyncio.get_running_loop()
```

You can think of *event_loop* as an autouse fixture for async tests.

1.2 Test discovery modes

Pytest-asyncio provides two modes for test discovery, *strict* and *auto*.

1.2.1 Strict mode

In strict mode pytest-asyncio will only run tests that have the *asyncio* marker and will only evaluate async fixtures decorated with `@pytest_asyncio.fixture`. Test functions and fixtures without these markers and decorators will not be handled by pytest-asyncio.

This mode is intended for projects that want so support multiple asynchronous programming libraries as it allows pytest-asyncio to coexist with other async testing plugins in the same codebase.

Pytest automatically enables installed plugins. As a result pytest plugins need to coexist peacefully in their default configuration. This is why strict mode is the default mode.

1.2.2 Auto mode

In *auto* mode pytest-asyncio automatically adds the *asyncio* marker to all asynchronous test functions. It will also take ownership of all async fixtures, regardless of whether they are decorated with `@pytest.fixture` or `@pytest_asyncio.fixture`.

This mode is intended for projects that use *asyncio* as their only asynchronous programming library. Auto mode makes for the simplest test and fixture configuration and is the recommended default.

If you intend to support multiple asynchronous programming libraries, e.g. *asyncio* and *trio*, strict mode will be the preferred option.

CHAPTER

TWO

REFERENCE

2.1 Configuration

The pytest-asyncio mode can be set by the `asyncio_mode` configuration option in the configuration file:

```
# pytest.ini
[pytest]
asyncio_mode = auto
```

The value can also be set via the `--asyncio-mode` command-line option:

```
$ pytest tests --asyncio-mode=strict
```

If the asyncio mode is set in both the pytest configuration file and the command-line option, the command-line option takes precedence. If no asyncio mode is specified, the mode defaults to *strict*.

2.2 Fixtures

2.2.1 event_loop

Creates a new asyncio event loop based on the current event loop policy. The new loop is available as the return value of this fixture or via asyncio.get_running_loop. The event loop is closed when the fixture scope ends. The fixture scope defaults to `function` scope.

```
def test_http_client(event_loop):
    url = "http://httpbin.org/get"
    resp = event_loop.run_until_complete(http_client(url))
    assert b"HTTP/1.1 200 OK" in resp
```

Note that, when using the `event_loop` fixture, you need to interact with the event loop using methods like `event_loop.run_until_complete`. If you want to *await* code inside your test function, you need to write a coroutine and use it as a test function. The *asyncio* marker is used to mark coroutines that should be treated as test functions.

The `event_loop` fixture can be overridden in any of the standard pytest locations, e.g. directly in the test file, or in `conftest.py`. This allows redefining the fixture scope, for example:

```
@pytest.fixture(scope="session")
def event_loop():
    policy = asyncio.get_event_loop_policy()
    loop = policy.new_event_loop()
```

(continues on next page)

(continued from previous page)

```
    yield loop
    loop.close()
```

If you need to change the type of the event loop, prefer setting a custom event loop policy over redefining the `event_loop` fixture.

If the `pytest.mark.asyncio` decorator is applied to a test function, the `event_loop` fixture will be requested automatically by the test function.

2.2.2 unused_tcp_port

Finds and yields a single unused TCP port on the localhost interface. Useful for binding temporary test servers.

2.2.3 unused_tcp_port_factory

A callable which returns a different unused TCP port each invocation. Useful when several unused TCP ports are required in a test.

```
def a_test(unused_tcp_port_factory):
    port1, port2 = unused_tcp_port_factory(), unused_tcp_port_factory()
    ...
```

2.2.4 unused_udp_port and unused_udp_port_factory

Works just like their TCP counterparts but returns unused UDP ports.

2.3 Markers

2.3.1 pytest.mark.asyncio

A coroutine or async generator with this marker will be treated as a test function by pytest. The marked function will be executed as an asyncio task in the event loop provided by the `event_loop` fixture.

In order to make your test code a little more concise, the pytest `pytestmark` feature can be used to mark entire modules or classes with this marker. Only test coroutines will be affected (by default, coroutines prefixed by `test_`), so, for example, fixtures are safe to define.

```
import asyncio

import pytest

# All test coroutines will be treated as marked.
pytestmark = pytest.mark.asyncio


async def test_example(event_loop):
    """No marker!"""
    await asyncio.sleep(0, loop=event_loop)
```

In *auto* mode, the `pytest.mark.asyncio` marker can be omitted, the marker is added automatically to *async* test functions.

2.4 Decorators

Asynchronous fixtures are defined just like ordinary pytest fixtures, except they should be decorated with `@pytest_asyncio.fixture`.

```
import pytest_asyncio


@pytest_asyncio.fixture
async def async_gen_fixture():
    await asyncio.sleep(0.1)
    yield "a value"


@pytest_asyncio.fixture(scope="module")
async def async_fixture():
    return await asyncio.sleep(0.1)
```

All scopes are supported, but if you use a non-function scope you will need to redefine the `event_loop` fixture to have the same or broader scope. Async fixtures need the event loop, and so must have the same or narrower scope than the `event_loop` fixture.

auto mode automatically converts async fixtures declared with the standard `@pytest.fixture` decorator to *asyncio-driven* versions.

2.5 Changelog

2.5.1 0.21.0 (2023-03-19)

- Drop compatibility with pytest 6.1. Pytest-asyncio now depends on pytest 7.0 or newer.
- pytest-asyncio cleans up any stale event loops when setting up and tearing down the event_loop fixture. This behavior has been deprecated and pytest-asyncio emits a DeprecationWarning when tearing down the event_loop fixture and current event loop has not been closed.

2.5.2 0.20.3 (2022-12-08)

- Prevent DeprecationWarning to bubble up on CPython 3.10.9 and 3.11.1. #460

2.5.3 0.20.2 (2022-11-11)

- Fixes an issue with async fixtures that are defined as methods on a test class not being rebound to the actual test instance. #197
- Replaced usage of deprecated `@pytest.mark.tryfirst` with `@pytest.hookimpl(tryfirst=True)` #438

2.5.4 0.20.1 (2022-10-21)

- Fixes an issue that warned about using an old version of pytest, even though the most recent version was installed. #430

2.5.5 0.20.0 (2022-10-21)

- BREAKING: Removed *legacy* mode. If you're upgrading from v0.19 and you haven't configured `asyncio_mode = legacy`, you can upgrade without taking any additional action. If you're upgrading from an earlier version or you have explicitly enabled *legacy* mode, you need to switch to *auto* or *strict* mode before upgrading to this version.
- Deprecate use of pytest v6.
- Fixed an issue which prevented fixture setup from being cached. #404

2.5.6 0.19.0 (2022-07-13)

- BREAKING: The default `asyncio_mode` is now *strict*. #293
- Removes *setup.py* since all relevant configuration is present *setup.cfg*. Users requiring an editable installation of pytest-asyncio need to use pip v21.1 or newer. #283
- Declare support for Python 3.11.

2.5.7 0.18.3 (2022-03-25)

- Adds pytest-trio to the test dependencies
- Fixes a bug that caused pytest-asyncio to try to set up async pytest_trio fixtures in strict mode. #298

2.5.8 0.18.2 (2022-03-03)

- Fix asyncio auto mode not marking static methods. #295
- Fix a compatibility issue with Hypothesis 6.39.0. #302

2.5.9 0.18.1 (2022-02-10)

- Fixes a regression that prevented async fixtures from working in synchronous tests. #286

2.5.10 0.18.0 (2022-02-07)

- Raise a warning if @pytest.mark.asyncio is applied to non-async function. #275
- Support parametrized `event_loop` fixture. #278

2.5.11 0.17.2 (2022-01-17)

- Require `typing-extensions` on Python<3.8 only. #269
- Fix a regression in tests collection introduced by 0.17.1, the plugin works fine with non-python tests again. #267

2.5.12 0.17.1 (2022-01-16)

- Fixes a bug that prevents async Hypothesis tests from working without explicit `asyncio` marker when `--asyncio-mode=auto` is set. #258
- Fixed a bug that closes the default event loop if the loop doesn't exist #257
- Added type annotations. #198
- Show asyncio mode in pytest report headers. #266
- Relax `asyncio_mode` type definition; it allows to support pytest 6.1+. #262

2.5.13 0.17.0 (2022-01-13)

- *pytest-asyncio* no longer alters existing event loop policies. #168, #188
- Drop support for Python 3.6
- Fixed an issue when pytest-asyncio was used in combination with *flaky* or inherited asynchronous Hypothesis tests. #178 #231
- Added flaky to test dependencies
- Added `unused_udp_port` and `unused_udp_port_factory` fixtures (similar to `unused_tcp_port` and `unused_tcp_port_factory` counterparts. #99
- Added the plugin modes: *strict*, *auto*, and *legacy*. See documentation for details. #125
- Correctly process `KeyboardInterrupt` during async fixture setup phase #219

2.5.14 0.16.0 (2021-10-16)

- Add support for Python 3.10

2.5.15 0.15.1 (2021-04-22)

- Hotfix for errors while closing event loops while replacing them. #209 #210

2.5.16 0.15.0 (2021-04-19)

- Add support for Python 3.9
- Abandon support for Python 3.5. If you still require support for Python 3.5, please use pytest-asyncio v0.14 or earlier.
- Set `unused_tcp_port_factory` fixture scope to 'session'. #163
- Properly close event loops when replacing them. #208

2.5.17 0.14.0 (2020-06-24)

- Fix #162, and `event_loop` fixture behavior now is coherent on all scopes. #164

2.5.18 0.12.0 (2020-05-04)

- Run the event loop fixture as soon as possible. This helps with fixtures that have an implicit dependency on the event loop. #156

2.5.19 0.11.0 (2020-04-20)

- Test on 3.8, drop 3.3 and 3.4. Stick to 0.10 for these versions. #152
- Use the new Pytest 5.4.0 Function API. We therefore depend on pytest >= 5.4.0. #142
- Better `pytest.skip` support. #126

2.5.20 0.10.0 (2019-01-08)

- `pytest-asyncio` integrates with Hypothesis to support `@given` on async test functions using `asyncio`. #102
- Pytest 4.1 support. #105

2.5.21 0.9.0 (2018-07-28)

- Python 3.7 support.
- Remove `event_loop_process_pool` fixture and `pytest.mark.asyncio_process_pool` marker (see https://bugs.python.org/issue34075 for deprecation and removal details)

2.5.22 0.8.0 (2017-09-23)

- Improve integration with other packages (like aiohttp) with more careful event loop handling. #64

2.5.23 0.7.0 (2017-09-08)

- Python versions pre-3.6 can use the async_generator library for async fixtures. *#62 <https://github.com/pytest-dev/pytest-asyncio/pull/62>*

2.5.24 0.6.0 (2017-05-28)

- Support for Python versions pre-3.5 has been dropped.
- `pytestmark` now works on both module and class level.
- The `forbid_global_loop` parameter has been removed.
- Support for async and async gen fixtures has been added. #45
- The deprecation warning regarding `asyncio.async()` has been fixed. #51

2.5.25 0.5.0 (2016-09-07)

- Introduced a changelog. #31
- The `event_loop` fixture is again responsible for closing itself. This makes the fixture slightly harder to correctly override, but enables other fixtures to depend on it correctly. #30
- Deal with the event loop policy by wrapping a special pytest hook, `pytest_fixture_setup`. This allows setting the policy before fixtures dependent on the `event_loop` fixture run, thus allowing them to take advantage of the `forbid_global_loop` parameter. As a consequence of this, we now depend on pytest 3.0. #29

2.5.26 0.4.1 (2016-06-01)

- Fix a bug preventing the propagation of exceptions from the plugin. #25

2.5.27 0.4.0 (2016-05-30)

- Make `event_loop` fixtures simpler to override by closing them in the plugin, instead of directly in the fixture. #21
- Introduce the `forbid_global_loop` parameter. #21

2.5.28 0.3.0 (2015-12-19)

- Support for Python 3.5 `async`/`await` syntax. #17

2.5.29 0.2.0 (2015-08-01)

- `unused_tcp_port_factory` fixture. #10

2.5.30 0.1.1 (2015-04-23)

Initial release.

This section of the documentation provides descriptions of the individual parts provided by pytest-asyncio. The reference section also provides a chronological list of changes for each release.

CHAPTER

THREE

GETTING SUPPORT

3.1 Enterprise support

Tidelift works with maintainers of numerous open source projects to ensure enterprise-grade support for your software supply chain.

The Tidelift subscription includes security updates, verified license compliance, continuous software maintenance, and more. As a result, you get the guarantees provided by commercial software for the open source packages you use.

Consider signing up for the Tidelift subscription.

3.2 Direct maintainer support

If you require commercial support outside of the Tidelift subscription, reach out to Michael Seifert, one of the project's maintainers.

3.3 Community support

The GitHub page of pytest-asyncio offers free community support on a best-effort basis. Please use the issue tracker to report bugs and the discussions to ask questions.

pytest-asyncio is a pytest plugin. It facilitates testing of code that uses the asyncio library.

Specifically, pytest-asyncio provides support for coroutines as test functions. This allows users to *await* code inside their tests. For example, the following code is executed as a test item by pytest:

```
@pytest.mark.asyncio
async def test_some_asyncio_code():
    res = await library.do_something()
    assert b"expected result" == res
```

Note that test classes subclassing the standard unittest library are not supported. Users are advised to use unittest.IsolatedAsyncioTestCase or an async framework such as asynctest.

pytest-asyncio is available under the Apache License 2.0.