asyncio

New in version 2.0.

Scrapy has partial support for `asyncio`. After you install the asyncio reactor, you may use `asyncio` and `asyncio`-powered libraries in any coroutine.

Installing the asyncio reactor

To enable `asyncio` support, set the `TWISTED_REACTOR` setting to `'twisted.internet.asyncioreactor.AsyncioSelectorReactor'`.

If you are using `CrawlerRunner`, you also need to install the `AsyncioSelectorReactor` reactor manually. You can do that using `install_reactor()`:

```
install_reactor('twisted.internet.asyncioreactor.AsyncioSelectorReactor')
```

Using custom asyncio loops

You can also use custom asyncio event loops with the asyncio reactor. Set the `ASYNCIO_EVENT_LOOP` setting to the import path of the desired event loop class to use it instead of the default asyncio event loop.

Windows-specific notes

The Windows implementation of `asyncio` can use two event loop implementations:

- `SelectorEventLoop`, default before Python 3.8, required when using Twisted.
- `ProactorEventLoop`, default since Python 3.8, cannot work with Twisted.

So on Python 3.8+ the event loop class needs to be changed.

Changed in version 2.6.0: The event loop class is changed automatically when you change the `TWISTED_REACTOR` setting or call `install_reactor()`.

To change the event loop class manually, call the following code before installing the reactor:

```
import asyncio
asyncio.set_event_loop_policy(asyncio.WindowsSelectorEventLoopPolicy())
```

You can put this in the same function that installs the reactor, if you do that yourself, or in some code that runs before the reactor is installed, e.g. `settings.py`.

Note

Other libraries you use may require `ProactorEventLoop`, e.g. because it supports subprocesses (this is the case with playwright), so you cannot use them together with Scrapy on Windows (but you should be able to use them on WSL or native Linux).

Awaiting on Deferreds

When the asyncio reactor isn't installed, you can await on Deferreds in the coroutines directly. When it is installed, this is not possible anymore, due to specifics of the Scrapy coroutine integration (the coroutines are wrapped into `asyncio.Future` objects, not into `Deferred` directly), and you need to wrap them into Futures. Scrapy provides two helpers for this:

`scrapy.utils.defer.deferred_to_future`(*d: twisted.internet.defer.Deferred*) → _asyncio.Future [source]

New in version 2.6.0.

Return an `asyncio.Future` object that wraps *d*.

When using the asyncio reactor, you cannot await on `Deferred` objects from Scrapy callables defined as coroutines, you can only await on `Future` objects. Wrapping `Deferred` objects into `Future` objects allows you to wait on them:

```
class MySpider(Spider):
    ...
    async def parse(self, response):
        d = treq.get('https://example.com/additional')
        additional_response = await deferred_to_future(d)
```

`scrapy.utils.defer.maybe_deferred_to_future`(*d: twisted.internet.defer.Deferred*) → Union[twisted.internet.defer.Deferred, _asyncio.Future] [source]

New in version 2.6.0.

Return *d* as an object that can be awaited from a Scrapy callable defined as a coroutine.

What you can await in Scrapy callables defined as coroutines depends on the value of `TWISTED_REACTOR`:

- When not using the asyncio reactor, you can only await on `Deferred` objects.
- When using the asyncio reactor, you can only await on `asyncio.Future` objects.

If you want to write code that uses `Deferred` objects but works with any reactor, use this function on all `Deferred` objects:

```
class MySpider(Spider):
    ...
    async def parse(self, response):
        d = treq.get('https://example.com/additional')
        extra_response = await maybe_deferred_to_future(d)
```

Tip

If you need to use these functions in code that aims to be compatible with lower versions of Scrapy that do not provide these functions, down to Scrapy 2.0 (earlier versions do not support `asyncio`), you can copy the implementation of these functions into your own code.