asyncio-run-in-process Documentation

Release 0.1.0-alpha.10

The Ethereum Foundation

Sep 10, 2020

Contents

Simple asyncio friendly replacement for multiprocessing to run a coroutine in an isolated process

CHAPTER 1

Contents

1.1 asyncio_run_in_process package

1.1.1 Submodules

1.1.2 asyncio_run_in_process.abc module

1.1.3 asyncio_run_in_process.exceptions module

1.1.4 asyncio_run_in_process.process module

1.1.5 asyncio_run_in_process.run_in_process module

1.1.6 asyncio_run_in_process.state module

1.1.7 asyncio_run_in_process.typing module

1.1.8 Module contents

1.2 Release Notes

1.2.1 v0.1.0-alpha.7

- Added new APIs to run the coroutine with trio in the subprocess
- Several bug fixes

1.2.2 v0.1.0-alpha.1

- Launched repository, claimed names for pip, RTD, github, etc

CHAPTER 2

Quickstart

We use `run_in_process` for something we want run in a process in a blocking manner.

```
from asyncio_run_in_process import run_in_process

async def fib(n):
    if n <= 1:
        return n
    else:
        return await fib(n - 1) + await fib(n - 2)

# runs in a separate process
result = await run_in_process(fib, 10)
print(f"The 10th fibbonacci number is {result}")
```

We use `open_in_process` for something we want to run in the background.

```
from asyncio_run_in_process import open_in_process:

async def fib(n):
    if n <= 1:
        return n
    else:
        return await fib(n - 1) + await fib(n - 2)

# runs in a separate process
async with open_in_process(fib, 10) as proc:
    # do some other things here while it runs in the background.
    ...
    # the context will block here until the process has finished.
# once the context exits the result is available on the process.
print(f"The 10th fibbonacci number is {proc.result}")
```

Both functions above will run the coroutine in an asyncio event loop, but should you want to run them with `trio`, the `run_in_process_with_trio` and `open_in_process_with_trio` functions can be used.

CHAPTER 3

Maximum number of running processes

By default we can only have up to `MAX_PROCESSES` running at any given moment, but that can be changed via the `ASYNCIO_RUN_IN_PROCESS_MAX_PROCS` environment variable.

CHAPTER 4

Gotchas

If a function passed to `open_in_process` uses asyncio's `loop.run_in_executor()` to run synchronous code, you must ensure the task/process terminates in case the `loop.run_in_executor()` call is cancelled, or else `open_in_process` will not ever return. This is necessary because asyncio does not cancel the thread/process it starts (in `loop.run_in_executor()`), and that prevents `open_in_process` from terminating. One way to ensure that is to have the code running in the executor react to an event that gets set when `loop.run_in_executor()` returns.

CHAPTER 5

Indices and tables

- genindex
- modindex