asyncio developer notes Documentation

Carol Willing

Sep 20, 2018

Contents

This is an **unofficial** document for developing on Python's asyncio.

CHAPTER 1

History of asyncio

1.1 PEPs

- PEP 3156 Asynchronous IO Support Rebooted: the "asyncio" Module
  - source
  - Abstract: This is a proposal for asynchronous I/O in Python 3, starting at Python 3.3. Consider this the concrete proposal that is missing from PEP 3153. The proposal includes a pluggable event loop, transport and protocol abstractions similar to those in Twisted, and a higher-level scheduler based on `yield from` (PEP 380). The proposed package name is ''asyncio'.
- PEP 492 Coroutines with async and await syntax
  - source
  - Abstract: The growth of Internet and general connectivity has triggered the proportionate need for responsive and scalable code. This proposal aims to answer that need by making writing explicitly asynchronous, concurrent Python code easier and more Pythonic.

    It is proposed to make coroutines a proper standalone concept in Python, and introduce new supporting syntax. The ultimate goal is to help establish a common, easily approachable, mental model of asynchronous programming in Python and make it as close to synchronous programming as possible.

    This PEP assumes that the asynchronous tasks are scheduled and coordinated by an Event Loop similar to that of stdlib module asyncio.events.AbstractEventLoop. While the PEP is not tied to any specific Event Loop implementation, it is relevant only to the kind of coroutine that uses yield as a signal to the scheduler, indicating that the coroutine will be waiting until an event (such as IO) is completed.
- PEP 525 Asynchronous Generators
  - source
  - Abstract: PEP 492 introduced support for native coroutines and async/await syntax to Python 3.5. It is proposed here to extend Python's asynchronous capabilities by adding support for asynchronous generators.
- PEP 530 Asynchronous comprehensions

  - source
  - Abstract: PEP 492 and PEP 525 introduce support for native coroutines and asynchronous generators using async / await syntax. This PEP proposes to add asynchronous versions of list, set, dict comprehensions and generator expressions.

- PEP 567 Context Variables
  - source
  - Abstract: This PEP proposes a new contextvars module and a set of new CPython C APIs to support context variables. This concept is similar to thread-local storage (TLS), but, unlike TLS, it also allows correctly keeping track of values per asynchronous task, e.g. asyncio.Task.

  This proposal is a simplified version of PEP 550. The key difference is that this PEP is concerned only with solving the case for asynchronous tasks, not for generators. There are no proposed modifications to any built-in types or to the interpreter.

  This proposal is not strictly related to Python Context Managers. Although it does provide a mechanism that can be used by Context Managers to store their state.

1.2 Releases

- 3.4
- 3.5
- 3.6
- 3.7
- 3.8 (TBD)

CHAPTER 2

asyncio Concepts

- What is a coroutine?
- What is an event loop?
- When to use a Future or a Task?
- Callbacks
- Why would I set a policy?
- Why would I use a context?
- How do I use an async comprehension?
- Why choose asyncio instead of many threads?

CHAPTER 3

asyncio library files

3.1 Alphabetical

3.2 By functionality