

# Модуль asyncio позволяет писать параллельный (асинхронный) КОД



ddos-guard.net

РЕКЛАМА

### Купите выделенный сервер с Anti-DDoS в ЦОД Tier III

4,9 ★ Рейтинг организации ⓘ

Узнать больше

[Стандартная библиотека Python3.](#) / Модуль asyncio позволяет писать параллельный (асинхронный) код

[Модуль asyncio](#) - это библиотека для написания параллельного (асинхронного) кода с использованием [синтаксиса async/await](#) и часто идеально подходит для высокоуровневого структурированного кода с **привязкой к сетевому вводу-выводу** и не связанного с блокирующими вызовами.

Асинхронное программирование сильно отличается от классического "последовательного" программирования.

Модуль asyncio предоставляет инструменты для создания асинхронных/параллельных приложений с использованием сопрограмм. В то время как модуль [threading](#) реализует параллелизм через потоки приложений, а модуль [multiprocessing](#) реализует параллелизм с использованием системных процессов.

Модуль asyncio использует **однопоточный однопроцессный подход**, в котором части приложения взаимодействуют для явного переключения задач в оптимальное время. Чаще всего это переключение контекста происходит, когда программа не блокирует ожидание чтения или записи данных (например http или socket протоколы). Этот модуль также включает поддержку кода планирования для запуска задачи в определенное время в будущем, чтобы одна сопрограмма могла ждать завершения другой для обработки системных сигналов, а также для распознавания других событий, которые могут быть причиной изменения приложением того, над чем оно работает.

*Примечание:* Не пытайтесь использовать [модуль asyncio](#) для кода, который использует вызовы, блокирующие поток программы, например файловый ввод/вывод. Из-за блокировок ваш код будет выполняться синхронно. Для таких задач есть альтернативные встроенные модули, такие как [threading](#) и [multiprocessing](#), у которых практически одинаковый API. Если вам все же необходим запуск блокирующих операций из асинхронного кода, то воспользуйтесь [модулем concurrent.futures](#), чтобы запустить такие операции в отдельном потоке или на другом ядре процессора.

## Модуль asyncio предоставляет высокоуровневый API.

Высокоуровневый API позволяет:

- одновременно запускать сопрограммы Python и полностью контролировать их выполнение;
- выполнять **сетевой ввод-вывод** и IPC;
- контролировать ход подпроцессов;
- распределять задачи по очередям;
- синхронизировать параллельный код;

Цикл событий - это ядро любого приложения, использующего [модуль asyncio](#). Циклы событий запускают асинхронные задачи и обратные вызовы, выполняют **сетевые операции** ввода-вывода и запускают подпроцессы.

## Простой пример программы высокоуровневого API:

```
import asyncio

async def main():
    print('Hello ...')
    await asyncio.sleep(1)
    print('... World!')

# Python 3.7+
as Вверх un(main())
# Hello ...
```

```
# задержка в 1 секунду
# ... World!
```

## Кроме того, существуют низкоуровневый API:

Разработчики приложений обычно должны использовать высокоуровневые функции `asyncio`, такие как [`asyncio.run\(\)`](#), так как редко нужно ссылаться на объект цикла или вызывать его методы. Низкоуровневый API в основном предназначен для авторов, библиотек и фреймворков, которым нужен более тонкий контроль над поведением цикла событий.

Низкоуровневый API обеспечивает:

- создание и управление циклами событий, которые предоставляют асинхронный API для работы в сети, запуска подпроцессов, обработки сигналов ОС и т. д;
- реализацию эффективных протоколов с использованием транспортов;
- обеспечения взаимодействия объектов `future` на основе обратного вызова и высокоуровневого кода, основанного на [синтаксисе `async/await`](#).

### Содержание раздела:

- [КРАТКИЙ ОБЗОР МАТЕРИАЛА.](#)
- [Сопрограммы и механизмы их запуска модулем `asyncio`](#)
- [Что такое `awaitable` объект модуля `asyncio`](#)
- [Функция `run\(\)` модуля `asyncio`](#)
- [Менеджер контекста `Runner\(\)` модуля `asyncio`](#)
- [Функция `create\_task\(\)` модуля `asyncio`](#)
- [Группы задач `TaskGroup\(\)` модуля `asyncio`](#)
- [Класс `Task\(\)` модуля `asyncio`](#)
- [Функция `sleep\(\)` модуля `asyncio`](#)
- [Функция `gather\(\)` модуля `asyncio`](#)
- [Функция `shield\(\)` модуля `asyncio`](#)
- [Асинхронный менеджер `timeout\(\)` модуля `asyncio`](#)
- [Асинхронный менеджер `timeout at\(\)` модуля `asyncio`](#)
- [Функция `wait for\(\)` модуля `asyncio`](#)
- [Функция `as\_completed\(\)` модуля `asyncio`](#)
- [Функция `wait\(\)` модуля `asyncio`](#)
- [Функция `to\_thread\(\)` модуля `asyncio`](#)
- [Функция `run\_coroutine\_threadsafe\(\)` модуля `asyncio`](#)
- [Функции `current task\(\)` и `all tasks\(\)` модуля `asyncio`](#)
- [Использование очереди `asyncio.Queue`](#)
- [Примитивы синхронизации задач в `asyncio`](#)
- [Запуск внешних программ из кода `asyncio`](#)
- [Работа с сетевыми соединениями модуля `asyncio`](#)
- [Объект `Future` модуля `asyncio` Python и связанные функции](#)
- [Создание и получение текущего цикла событий, модуль `asyncio`](#)
- [Создание, запуск и остановка цикла событий модуля `asyncio`](#)
- [Планирование обратных вызовов из цикла событий `asyncio`](#)
- [Создание `Future` и `Task` из цикла событий `asyncio`](#)
- [Создание пулов потоков и процессов из цикла событий `asyncio`](#)
- [Создание TCP, UDP и Unix соединений из цикла событий `asyncio`](#)
- [Создание сетевых серверов из цикла событий `asyncio`](#)
- [Создание subprocessa из цикла событий `asyncio`](#)
- [Работа с сокетами напрямую из цикла событий `asyncio`](#)
- [Передача файлов из цикла событий `asyncio`](#)
- [Наблюдение за дескрипторами файлов из цикла событий `asyncio`](#)
- [DNS запросы из цикла событий `asyncio`](#)
- [Unix в циклах событий `asyncio`](#)
- [Перемещение и многопоточность в цикле событий `asyncio`](#)

Вверх

- [Объекты Transport и Protocol в цикле событий asyncio](#)
- [Включение режима отладки в asyncio](#)
- [Обработка исключений в цикле событий модуля asyncio](#)
- [Исключения модуля asyncio Python](#)

ХОЧУ ПОМОЧЬ  
ПРОЕКТУ

