Асинхронное программирование

№ урока: 5 **Курс:** Python Advanced

Средства обучения: PyCharm

Обзор, цель и назначение урока

Изучить основы асинхронности, задачи для её применения. Разобраться с понятием сопрограммы/корутины и ключевыми словами async/await. Понимать назначение цикла событий (Event Loop). Рассмотреть примеры работы с модулем asyncio.

Изучив материал данного занятия, учащийся сможет:

- Понимать общую схему работы асинхронности и ее особенностей.
- Создавать асинхронные программы, используя async/await/yield from.
- Использовать Event Loop для запуска собственных сопрограмм.
- Использовать модуль asyncio для создания и запуска сопрограмм.

Содержание урока

1. Основные понятия асинхронности.
2. Сопрограммы/корутины и ключевые слова async/await.
3. Модуль asyncio и запуск цикла событий.
4. Запуск сопрограмм в цикле событий. Примеры и различные варианты.
5. Примеры сторонних библиотек и фреймворков: aiohttp, gevent и tornado.

Резюме

В каждой программе строки кода выполняются поочередно. Например, если у вас есть строка кода, которая запрашивает что-либо с сервера, то это означает, что ваша программа не делает ничего во время ожидания ответа. В некоторых случаях это допустимо, но во многих — нет. Одним из решений этой проблемы являются потоки (*threads*).
Потоки дают возможность вашей программе выполнять ряд задач одновременно. У потоков есть недостатки: многопоточные программы являются более сложными и более подвержены ошибкам, проблема состояние гонки (*race condition*), взаимная (*deadlock*) и активная (*livelock*) блокировка, исчерпание ресурсов (*resource starvation*).

Хотя асинхронное программирование решает проблемы потоков, оно было разработано для другой цели — для переключения контекста процессора. Когда у вас есть несколько потоков, **каждое ядро процессора может запускать только один поток за раз**. Для того, чтобы все потоки/процессы могли совместно использовать ресурсы, процессор очень часто переключает контекст. Чтобы упростить работу, процессор с произвольной периодичностью сохраняет всю контекстную информацию потока и переключается на другой поток.

Асинхронное программирование — это потоковая обработка программного обеспечения / пользовательского пространства, где приложение, а не процессор, управляет потоками и переключением контекста. В асинхронном программировании контекст переключается только в заданных точках переключения, а не с периодичностью, определенной CPU.

Представьте секретаря, который не тратит время впустую. У него есть пять заданий, которые он выполняет одновременно: отвечает на телефонные звонки, принимает посетителей, пытается забронировать билеты на самолет, контролирует графики встреч и заполняет документы. Теперь представьте, что такие задачи, как контроль графиков встреч, прием телефонных звонков и посетителей, повторяются не часто и распределены во времени. Таким образом, большую часть времени секретарь разговаривает по телефону с авиакомпанией, заполняя при этом документы. Когда поступит телефонный

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

звонок, он поставит разговор с авиакомпанией на паузу, ответит на звонок, а затем вернется к разговору с авиакомпанией. В любое время, когда новая задача потребует внимания секретаря, заполнение документов будет отложено, поскольку оно не критично. Секретарь, выполняющий несколько задач одновременно, переключает контекст в нужное ему время. **Он асинхронный.**

Для запуска асинхронных программ существует специальный пул задач, куда можно складывать сопрограммы, и они будут последовательно переключаться друг между другом. Такой пул задач называется циклом событий(EventLoop). Сопрограммы как раз-таки предназначены для запуска в цикле событий.
Начиная с версии 3.5 в Python добавили новые ключевые слова async/await, которые полностью заменяют yield/from. Однако для совместимости, был добавлен декоратор asyncio.coroutine, который можно использовать как раз для поддержки yield/from.
async/await
Библиотека Asyncio довольно мощная, поэтому Python сообщество решило сделать ее стандартной библиотекой. В синтаксис также добавили ключевое слово async. Ключевые слова предназначены для более четкого обозначения асинхронного кода. Поэтому теперь методы не путаются с генераторами (yield/from). Ключевое слово async идет до def, чтобы показать, что метод является асинхронным. Ключевое слово await показывает, что вы ожидаете завершения сопрограммы. Пример ключевыми словами async/await:

```
import asyncio

async def async_worker(number, divider):
    """
    вместо yield/yield from используем синтаксис async/await.
    """
    print('Worker {} started'.format(number))
    await asyncio.sleep(2)
    print(number / divider)
    return number / divider

async def gather_worker():
    # выполнение нескольких задач и получение результата от каждой из них
    # в результате выполнения мы получим список результатов в том же порядке,
    # в котором передавали сопрограммы.
    result = await asyncio.gather(
        async_worker(50, 10),
        async_worker(60, 10),
        async_worker(70, 10),
        async_worker(80, 10),
        async_worker(90, 10),
    )
    print(result)

event_loop = asyncio.get_event_loop()
task_list = [
    # async_worker(30, 10),
    # asyncio.ensure_future(async_worker(30, 10)),
    event_loop.create_task(gather_worker())
]
tasks = asyncio.wait(task_list)
event_loop.run_until_complete(tasks)
event_loop.close()
```

Библиотека Asyncio — не единственный способ писать асинхронные программы: можно использовать функции обратного вызова (tornado) или зеленые потоки (gevent).

CyberBionic systematics
CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Закрепление материала

- Что такое асинхронность?
- Кто переключает управление между задачами в многопоточном программировании?
- Кто переключает управление между задачами в сопрограммах?
- Как влияет GIL на работу асинхронных программ?
- Что такое сопрограмма и что связывает ее с генераторами?
- Что такое цикл событий, для чего он нужен?
- Зачем нужно ключевое слово async?
- Зачем нужно ключевое слово await?
- Начиная с какой версии языка Python он стал поддерживать в синтаксисе слова async и await?
- Какую сопрограмму можно использовать, чтобы уснуть на N секунд и переключиться на другую задачи?
- Для чего нужна библиотека aiohttp?

Самостоятельная деятельность учащегося

Задание 1
Создайте сопрограмму, которая получает контент с указанных ссылок и логирует ход выполнения в специальный файл используя стандартную библиотеку urllib, а затем проделайте то же самое с библиотекой aiohttp. Шаги, которые должны быть залогированы: начало запроса к адресу X, ответ для адреса X получен со статусом 200. Проверьте ход выполнения программы на >3 ресурсах и посмотрите последовательность записи логов в обоих вариантах и сравните результаты. Для двух видов задач используйте разные файла для логирования, чтобы сравнить полученный результат.

Задание 2
Разработайте сокет сервер на основе библиотеки asyncio.

Рекомендуемые ресурсы

Официальный сайт Python - asyncio
https://docs.python.org/3.11/library/asyncio.html
Официальный сайт AIOHHTP
https://aiohttp.readthedocs.io/en/stable/

CyberBionic systematics
CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com