Асинхронное и многопоточное программирование

№ урока: 3 **Курс:** Python Advanced

Средства обучения: Python3.6, PyCharm

Обзор, цель и назначение урока

Изучить основы многопоточности. Получить опыт работы с модулем threading в Python. Рассмотреть способы синхронизации работы потоков. Разобраться с понятием GIL в Python и ограничений, которые накладываются на эталонную реализацию языка Python- CPython. Рассмотреть примеры работы с модулем concurrent.futures.
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:

- Создавать многопоточные программы.
- Понимать ограничение CPython накладываемые GIL при написании многопоточных программ.
- Использовать модули threading и concurrent.futures.
- 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 многопоточности.
2. GIL в Python.
3. Изучение модуля threading: Thread, Lock, RLock, Event, Semaphore, Timer.
4. Изучение библиотеки concurrent. futures.
5. Основные понятия асинхронности.
6. Сопрограммы/корутины и ключевые слова async/await.
7. Модуль asyncio и запуск цикла событий.
8. Запуск сопрограмм в цикле событий. Примеры и различные варианты.
9. Примеры сторонних библиотек и фреймворков: aiohttp, gevent и tornado.

Резюме

- Каждая программа запускается в отдельном процессе. Минимальной единицей программы является поток. Каждый процесс состоит из некоторого количества потоков. Каждая программа состоит как минимум из одного потока, который называется главным потоком. Многопоточность используется для запуска параллельных вычислений с целью ускорения вычислений или использования всей мощи многоядерной архитектуры.
- В эталонной реализации языка Python- CPython, которую мы используем в своей практике для создания приложений, существует специальный механизм, называемый GIL (Global Interpreter Lock). Данный механизм накладывает ограничение на многопоточные программы и позволяет только одному потоку выполняться в конкретный момент времени. То есть мы не имеем как таковой многопоточности в явном виде в CPython. Существую и другие реализации языка Python, такие как IronPython, Jython, и они не

CyberBionic systematics
CyberBionic Systematics ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

имеют механизма GIL. То есть, даже если мы пишем многопоточную программу, создаваемые потоки не будут выполняться параллельно из-за блокировки.

- С стандартной библиотеки языка Python имеется модуль threading, который предназначен для создания многопоточных программ и содержит в себе набор классов для данных задач. Модуль также предоставляет специальные механизмы синхронизации процессов и потоков, такие как блокировки, события, семафоры и условные переменные.
- Начиная с 3 версии в Python доступен модуль concurrent.futures, который предоставляет удобные интерфейсы для запуска многопоточных и многопроцессных вычислений. Существует два класса для таких задач: ThreadPoolExecutor и ProcessPoolExecutor. Соответственно ThreadPoolExecutore- это класс для создания пула потоков, а ProcessPoolExecutor- для пула процессов.
- В рамках библиотеки concurrent.futures имеется специальный класс Future (часто называемого «фьючером» или «футуром»), который является примером Promise (так называемых обещаний). Он позволяет получить результат отложенного вычисления. То есть задача, которая вернет результат в ближайшем будущем, например, после выполнения какого-то набора нагруженных операций или, как пример, обращения к стороннему серверу. Данный класс имеет набор методов, таких как result, done, running, позволяющих проверять текущее состояние Future и получать результат выполнения по завершению вычислений.
- Executor-ы напрямую связаны с классом Future. При добавлении задачи на выполнение, соответствующий класс executor-а возвращает объект Future, который связан с передаваемой задачей на исполнение. И через полученный Future, как уже говорилось ранее, можно получить результат выполнения данной задачи из пула.
- Множество обычных задач, которые мы решаем каждый день, используя любой язык программирования, могут быть решены с использованием асинхронного подхода. Где за переключение управления между выполнением задач, может отвечать сама программа, а не процессор. Благодаря механизму сопрограмм, мы можем создавать такие программы, которые сами будут уведомлять о том, что можно переключиться к другой задаче, а не блокировать на какое-то время все вычисления.
- Для запуска таких программ существует специальный пул задач, куда можно складывать сопрограммы, и они будут последовательно переключаться друг между другом. Такой пул задач называется циклом событий (Event Loop). 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.

Закрепление материала

- Что такое потоки и как они связаны с процессами?
- Какое минимальное количество потоков содержит любая программа?
- В чем преимущество многопоточных программ?
- Что такое GIL и в чем особенности данного механизма?
- Какой класс из модуля threading необходимо использовать для создания потока, от которого можно наследоваться или же просто передать функцию для исполнения?
- Какие примитивы синхронизации процессов и потоков вы знаете?
- Какой класс используется для создание пула потоков из модуля concurrent.futures?
- Какой класс используется для создание пула процессов из модуля concurrent.futures?
- Как добавить задачу на исполнение в пулл Executor-а?
- Что такое Future?
- Что такое асинхронность?
- Кто переключает управление между задачами в многопоточном программировании?

CyberBionic systematics

CyberBionic Systematics ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

- 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?

Дополнительное задание

Задание 1
Создайте сокет-сервер, который будет ожидать подключения по tcp-протоколу и запускать сокет-клиентов, которые подключаются на TCP сокет. Используйте примеры для создания TCP сокетов из предыдущих примеров. В качестве примера сервера можно использовать вычисление произведения двух чисел, где клиент отправляет два числа на tcp-сокет, а сервер отвечает результатом сложения и ожидает от клиента новую пару чисел. В случае если сокет послал сообщение с текстом **close**, то закрывать соединение с обеих сторон и завершать выполнение потока обработки клиента.

Задание 2
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.

Самостоятельная деятельность учащегося

Задание 1
Создайте функцию по вычислению факториала числа. Запустите несколько задач, используя ThreadPoolExecutor и замерьте скорость их выполнения, а затем замерьте скорость вычисления, используя тот же самый набор задач на ProcessPoolExecutor. В качестве примеров, используйте крайние значения, начиная от минимальных и заканчивая максимально возможными, чтобы увидеть прирост или потерю производительности.

Задание 2
Создайте три функции, одна из которых читает файл на диске с заданным именем и проверяет наличие строку “Wow! ”. В случае, если файла нет, то засыпает на 5 секунд, а затем снова продолжает поиск по файлу. В случае, если файл есть, то открывает его и ищет строку “Wow!”. При наличии данной строки закрывает файл и генерирует событие, а другая функция ожидает данное событие и в случае его возникновения выполняет удаление этого файла. В случае если строки «Wow!» не было найдено в файле, то засыпать на 5 секунд. Создайте файл руками и проверьте выполнение программы.

Задание 3
Разработайте сокет сервер на основе библиотеки asyncio. Сокет сервер должен выполнять сложение двух чисел, как из предыдущего примера по многопоточности.

CyberBionic systematics
CyberBionic Systematics ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Рекомендуемые ресурсы

Официальный сайт Python (3.6) - threading
https://docs.python.org/3.6/library/threading.html
Официальный сайт Python (3.6) – concurrent.features
https://docs.python.org/3/library/concurrent.futures.html
Официальный сайт Python (3.6) - asyncio
https://docs.python.org/3.6/library/asyncio.html
Официальный сайт AIOHHTTP
https://aiohttp.readthedocs.io/en/stable/

CyberBionic systematics

CyberBionic Systematics ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com