

БИБЛИОТЕКА ASYNCIO

ВВЕДЕНИЕ

Задачи урока

- 01 Разобраться с библиотекой asyncio
- 02 Понять, как можно ускорить свой код, используя асинхронное программирование
- 03 Понять, как использовать event_loop

Библиотека asyncio

Библиотека Asynchronous I/O asyncio

это стандартная библиотека в языке программирования Python, которая предоставляет возможности для асинхронного программирования

Она была введена в стандартную библиотеку Python в версии 3.4 и предоставляет удобные инструменты для создания асинхронных приложений

Основная задача asyncio

обеспечить возможность эффективного выполнения нескольких задач асинхронно, внутри одного потока выполнения

Благодаря этому, асинхронное программирование становится более простым и эффективным, особенно, при обработке сетевых запросов или операций ввода-вывода, которые могут занимать большое количество времени

Основной строительный блок asyncio

Корутины (coroutines)

Корутины

это специальные функции, которые могут быть приостановлены и возобновлены в процессе их выполнения

! Важной особенностью корутин в `asyncio` является использование ключевого слова `async` перед их определением. Корутины позволяют программистам определить асинхронные операции и организовывать их выполнение в заданном порядке

! Другой важной частью `asyncio` являются **Event loop** (цикл событий). Event loop управляет исполнением асинхронных операций и управлением вызовов корутин. С его помощью можно запустить асинхронные операции, ожидать их результаты или отменять операции

В библиотеке `asyncio` также доступны различные инструменты для работы с сетью, файловой системой и другими асинхронными операциями

Например, `asyncio` предоставляет классы для создания сетевых серверов и клиентов, асинхронного чтения и записи файлов и даже возможность запуска внешних процессов асинхронно

Библиотека `asyncio` является мощным инструментом для разработки асинхронных приложений в Python. Она позволяет эффективно использовать ресурсы системы и создавать отзывчивые приложения, которые могут обрабатывать множество задач одновременно. Однако, чтобы использовать `asyncio`, необходимо понимать основные принципы асинхронного программирования и уметь правильно организовывать код с использованием корутин и `event loop`

Пример реализации без асинхронной реализации

```
...
import time
from turtle import pen

def parser(sitename):
    for page in range(0, 10):
        print(sitename, page)
        time.sleep(2)

if __name__ == '__main__':
    start_time = time.time()
    for site in ["site 1", "site 2", "site 3"]:
        parser(site)

    print("--- %s seconds ---" % (time.time() - start_time))
```

А теперь добавим библиотеку `asyncio` и асинхронный вызов методов

```
...
import time
from turtle import pen

import asyncio

async def parser(sitename):
    for page in range(0, 10):
        print(sitename, page)
        await asyncio.sleep(2)

if __name__ == '__main__':
    start_time = time.time()
    parsers = [
        asyncio.ensure_future(parser(site))
        for site in ["site 1", "site 2", "site 3"]
    ]
    print(parsers)
    event_loop = asyncio.get_event_loop()
    event_loop.run_until_complete(asyncio.gather(*parsers))
    event_loop.close()

    print("--- %s seconds ---" % (time.time() - start_time))
```

ВЫВОД

код, который работает асинхронно выполняет гораздо быстрее

Из асинхронных методов также можно получать данные

```
...
import time
from turtle import pen

import asyncio

async def get_all_pages(sitename):
    await asyncio.sleep(2)
    return range(0, 10)

async def get_page_date(site_name, page):
    await asyncio.sleep(1)
    return "asdasd"
```

```
async def parser(site_name):
    pages = await get_all_pages(site_name)
    for page in pages:
        data = await get_page_date(site_name, page)

    return site_name

if __name__ == '__main__':
    start_time = time.time()
    parsers = [
        asyncio.ensure_future(parser(site))
        for site in ["site 1", "site 2", "site 3"]
    ]
    event_loop = asyncio.get_event_loop()
    results = event_loop.run_until_complete(asyncio.gather(*parsers))
    event_loop.close()
    print(results)

    print("---- %s seconds ----" % (time.time() - start_time))
```

В event_loop мы выполняем метод parser который возвращает нам данные. При выполнении мы получим результаты для каждого сайта

ИТОГИ

- ✓ Если в программе есть код, который большую часть времени что-то ожидает - имеет смысл использовать асинхронность
- ✓ run_until_complete запускает асинхронное выполнение