Введение

В прошлой главе мы поняли, чем плохо вызывать блокирующие и cpu bound операции в асинхронном коде.

Но иногда вызвать синхронную операцию — или выполнить тяжелую cpu bound операцию — бывает необходимо.

Примеры

IO-операции

Нужно подключиться к системе, у которой нет асинхронной библиотеки.

Сейчас это не так распространено, но совсем недавно на практике была задача вызывать хранимые функции в SAP по протоколу rfc. Для него не было асинхронного коннектора. В такой ситуации нужно было писать свой коннектор и реализовывать протокол взаимодействия. Задача интересная, но достаточно нетривиальная и долгая, поэтому придумали альтернативный путь.

PS Однажды все-таки написали свой коннектор: asynctnt для tarantool.

CPU-операции

Нужно пройтись по всем ключам большого json в несколько мегабайт, который получили от внешнего сервиса.

Решение

Нужно запускать синхронные io bound или cpu bound операции не в event loop, а в отдельном потоке/процессе. В asyncio есть механизм асинхронного запуска операции в потоке/процессе run_in_executor.

Синтаксис asyncio.loop.run_in_executor:

```
import asyncio
import concurrent.futures

def blocking_io():
    # File operations (such as logging) can block the
    # event loop: run them in a thread pool.
    with open('/dev/urandom', 'rb') as f:
        return f.read(100)

def cpu_bound():
    # CPU-bound operations will block the event loop:
    # in general it is preferable to run them in a
    # process pool.
    return sum(i * i for i in range(10 ** 7))

async def main():
    loop = asyncio.get_running_loop()

    ## Options:

    # 1. Run in the default loop's executor:
    result = await loop.run_in_executor(
        None, blocking_io)
    print('default thread pool', result)

    # 2. Run in a custom thread pool:
    with concurrent.futures.ThreadPoolExecutor() as pool:
        result = await loop.run_in_executor(
            pool, blocking_io)
        print('custom thread pool', result)

    # 3. Run in a custom process pool:
    with concurrent.futures.ProcessPoolExecutor() as pool:
        result = await loop.run_in_executor(
            pool, cpu_bound)
        print('custom process pool', result)

asyncio.run(main())
```

В функцию **loop.run_in_executor** нужно передать **executor** (если передать None, то операции будут запускаться в потоках) и **синхронную** функцию, которую нужно исполнить.

Если в **loop.run_in_executor** передать асинхронную функцию **async def foo(): ...** , то executor ее вызовет без синтаксиса await. Будет создан объект корутины, но функция не выполнится.

Можно написать свой собственный executor. Он должен наследоваться от базового класса concurrent.futures.Executor.

Давайте посмотрим на примерах, какие проблемы позволяет решить run_in_executor. В задании нам понадобится mercury из последнего модуля в этой главе.