Event loop

Мы уже познакомились с этой сущностью, которая управляет задачами в асинхронной программе.

Event Loop следит за асинхронными задачами и принимает решение, какой из них передать управление в определенный момент. Более подробно мы изучим его в следующей главе, а пока давайте посмотрим, что он умеет и как с ним работать.

Возможности Event Loop на примерах конкретных задач

Пример 1. Мы проходимся по списку id-шек и запросим асинхронно информацию о пользователе. Методом **run_until_complete()** будем запускать короутину в нашей программе.

```
import asyncio
import csv
from faker import Faker # библиотека нужна для генерации рандомных значений

fake = Faker()


async def request_users_data(uid: int) -> dict:
    await asyncio.sleep(0.5)
    return {"id": uid, "name": fake.name(), "email": fake.email()} # с помощью faker генерируем осмысленные
значения


ids = [i for i in range(1, 11)]
loop = asyncio.get_event_loop() # получаем loop, тк ранее он не существовал - создаем новый

with open("out.csv", "w") as fh: # открываем фаил для записи
    fieldnames = ["id", "name", "email"] # названия колонок в csv файле
    writer = csv.DictWriter(fh, fieldnames=fieldnames) # объект для записи данных в csv формате
    writer.writeheader() # записываем название колонок
    for uid in ids:
        writer.writerow(loop.run_until_complete(request_users_data(uid))) # запускаем короутину в loop и
дождаемся ее результата
```

Содержимое файла out.csv будет выглядеть примерно так:

```
id,name,email
1,Danielle Obrien,yhenry@example.org
2,Katelyn Mcdowell,tami64@example.org
3,Holly Guerra,ashley47@example.com
4,Robert Morton,wudwayne@example.net
5,Thomas Knight,wesley95@example.com
6,Taylor Hill,robertoguzman@example.com
7,Taylor Leblanc,marc51@example.org
8,Taylor Calderon,wileytammy@example.com
9,Benjamin Andersen,yhill@example.net
10,Maria Lawrence,keith46@example.net
```

Пример 2. Теперь нужно написать демона, фоновую программу, которая раз в минуту шлет информацию о состоянии системы в сторонний сервис.

```
import asyncio


async def send_statistic(): # наш "поход" в сторонний сервис
    print(f"{time.time()}: stat")


async def worker():
    while True:
        await asyncio.sleep(60)
        await send_statistic()


loop = asyncio.get_event_loop() # получаем loop
loop.create_task(worker()) # добавляем Task в event_loop

loop.run_forever() # навсегда передаем управлени программы на обработку task-ов добавленных в event loop
```

```
1640161216.4160142: stat
1640161226.424393: stat
1640161236.4276211: stat
```

В обоих случаях можно обойтись без явного использования loop в нашей программе, если запустить ее через **asyncio.run()**:

```
import asyncio


async def send_statistic():
    print(f"{time.time()}: stat")


async def main():
    while True:
        await asyncio.sleep(60)
        await send_statistic()

asyncio.run(main()) # запуск короутины используя упрощенный синтаксис
```