AIOHTTP

Последствия стандартизации

После того как в Python появились волшебные слова `await` и `async`, была создана библиотека aiohttp. Это асинхронные HTTP-клиент и сервер, который построен поверх `asyncio` со встроенной поддержкой клиентских и серверных веб-сокетов.

Например, приложение, которое выводит заглавными буквами на главной странице случайную фразу, может выглядеть так:

Скопировать код

```
import aiohttp
from aiohttp import web

async def get_phrase():
    # Метод, который обращается к стороннему ресурсу и забирает данные
    async with aiohttp.ClientSession() as session:
        async with session.get('https://fish-text.ru/get', params={'type': 'title'}) as response:
            result = await response.json(content_type='text/html; charset=utf-8')
            return result.get('text')


async def index_handler(request):
    # Формируем ответ для клиента
    return web.Response(text=await get_phrase())


async def response_signal(request, response):
    # Увеличиваем все буквы в ответе
    response.text = response.text.upper()
    return response


async def make_app():
    # Объявляем приложение — ваш веб-сервер
    app = web.Application()

    # Добавим сигнал — метод, который нужно запустить при определённом событии
    # В этом случае он запустится после формирования ответа
    app.on_response_prepare.append(response_signal)
```

```
    # Добавим необходимый URL
    app.add_routes([web.get('/', index_handler)])
    return app

web.run_app(make_app())  # Запускаем приложение
```

Обратите внимание на то, как происходит запрос к другому сайту. В методе `get_phrase` находятся три неблокирующие операции:

- Установка контекста соединения, чтобы после выполнения всех действий в блоке освободились все задействованные сетевые ресурсы. Для каждой сессии можно задать таймауты, заголовки и прочие параметры, которые должны применяться к каждому соединению в её рамках.
- Формирование запроса, который отправится по сети.
- Загрузка содержимого. Здесь вы должны явно указать, что вам нужны данные из тела ответа, чтобы `aiohttp` выполнил загрузку.

Вы могли заметить, что структура приложения гораздо проще, чем у `Twisted`. Сравните реализации протокола: в `Twisted` вы должны описать три отдельных метода на каждое событие, и держать в памяти, когда вызывается каждый из них, и какие Deffered-цепочки в них задействованы. В `aiohttp` код пишется линейно и в нём нет коллбэков. Благодаря простоте кода и отличной документации `aiohttp` быстро обрёл популярность.

Особенности

- У `aiohttp` есть много функций для комфортной разработки асинхронного веб-приложения. Например, он поддерживает jinja-шаблоны и раздачу статических файлов, а также у него есть middlewares, пользовательские сессии и debugtoolbar.
- Его можно расширять почти до бесконечности с помощью множества библиотек, которые можно найти в GitHub или в одной из подборок.
- Сущности router и views построены как в популярных синхронных фреймворках. Вы можете писать обработчики как функции или классы, использовать Class Based Views, выносить роутер в отдельный файл, как в Django, или использовать декораторы, как во Flask.

- Вы можете декомпозировать ваш сервер на несколько приложений.
- Вам придётся явно указать ожидание закрытия соединений, которые задействованы на момент остановки сервера, прежде чем его выключить. Пока что в библиотеке нет встроенного элегантного решения.
- Несмотря на то что `aiohttp` создан для асинхронного взаимодействия по сети, разработчики настоятельно не рекомендуют задействовать его внутри синхронного фреймворка для опроса сторонних ресурсов.