Aiohttp Client

официальная документация aiohttp

документация по aiohttp client

Пример простого GET-запроса:

```
import aiohttp
import asyncio

async def main():
    async with aiohttp.ClientSession() as session:
        async with session.get('http://httpbin.org/get') as resp:
            print(resp.status)
            print(await resp.text())

loop = asyncio.get_event_loop()
loop.run_until_complete(main())
```

Давайте разберемся, что тут происходит:

Шаг 1

Создается сессия ClientSession. Объект сессии можно сравнить с браузером, в нем хранятся общие параметры клиента: открытые соединения к серверу, cookies, общие заголовки, общие правила обработки timeout и так далее.

Необязательно создавать сессию на каждый запрос. По умолчанию сессия содержит несколько подключений к серверу (connection pool) и использует механизм keep-alives для того, чтобы не устанавливать каждый раз соединение с сервером, а переиспользовать уже установленные соединения. Переиспользование подключений может сильно ускорить ваше приложение.
С параметрами, которые принимает ClientSession, можно ознакомиться по ссылке.

Шаг 2

Выполняется GET-запрос.

> **async with session.get(...) as resp:**

session.get — принимает параметры запроса: url, данные запроса, заголовки и cookies, спецефичные для этого запроса и так далее. Со всеми параметрами можно ознакомиться по ссылке.

resp — переменная, из которой можно получить ответ после выполнения запроса HTTP. Все доступные поля и методы можно посмотреть по ссылке.

Стоит отметить, что какие-то результаты доступны сразу, например: статус, заголовки, cookies. Но тело ответа нужно вычитывать отдельно. Чтение ответа является тоже асинхронной операцией: методы read(), text(), json() являются корутинами, и их следует вызывать с ключевым словом await.

В некоторых ситуациях, таких, как чтение большого файла, нужно получать тело ответа частями. Но об этом поговорим немног позже.

Отличия **read(), text()** и **json()**

await resp.read() получает данные в байтах. Если вызвать print(await resp.read()), можно получить результат подобного вида:

```
>>> 'привет hello'.encode()
b'\xd0\xbf\xd1\x80\xd0\xb8\xd0\xb2\xd0\xb5\xd1\x82 hello'
>>>
```

await resp.text() вызывает await resp.read(), определяет кодировку и декодирует строку. В результате выполнения функции получим объект строки.
Для понимания работы функции стоит посмотреть ее исходный код.

await resp.json() делает то же самое, что text, только потом еще преобразовывает ответ в объект словаря.