Aiohttp web socket client

документация

Пример:

```
import aiohttp

async with aiohttp.ClientSession() as session:
    async with session.ws_connect('http://example.org/ws') as ws:
        async for msg in ws:
            print'new message', msg)
```

Разберемся.

Как и в случае с обычными http-запросами, нужно создать объект **ClientSession.** Для ws справедливы http-параметры: заголовки, cookies, timeout подключения и тд.

Работает так, потому что инициализация ws-соединения начинается с http-запроса вида:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

На что сервер должен ответить:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

Поэтому при подключении можно передавать дополнительные заголовки или cookies, например, для авторизации.

После создания сессии нужно подключиться к серверу по протоколу ws. Для этого нужно вызвать функцию **ws_connect**. Аргументы, которые принимает функция, можно посмотреть в документации**.**

В результате будет получен объект ws: ClientWebSocketResponse.

Полезные методы объекта ClientWebSocketResponse:

- **ping/pong** (корутина) — служебные методы, которые поддерживают ws-соединение. Если по tcp-соединению ничего не передавать, оно может закрыться по таймауту. В функции **ws_connect** есть включенный по умолчанию параметр **autoping**.
- **send_str/send_bytes/send_json (корутина)** — функции, которые позволяют отправлять данные в web socket на сервер.
- **receive** — корутина, которая позволяет получать данные из web socket с сервера. Корутина неявно обрабатывает сообщения PING, PONG и CLOSE, при этом не возвращая их. Она обрабатывает «игру в пинг-понг» и выполняет закрытие, если пришло сообщение CLOSE. Корутина **receive_str** разрешает получать только сообщения типа TEXT, **receive_bytes** — сообщения типа BINARY, **receive_json** — сообщения типа TEXT, и преобразовывает их в словарь.

При этом ClientWebSocketResponse является асинхронным итератором, исходных код клиента. При использовании синтаксиса ниже происходит вызов функции **receive**:

```
async for msg in ws:
    print('new message', msg)
```

Выше я упоминал про некоторые сообщения и их типы.

Переменная **msg** является объектом класса **WSMessage.** Вот его исходный код. Сообщение состоит из 3 полей: type, data, extra. Все доступные типы можно посмотреть в коде. Библиотека aiohttp обрабатывает служебные сообщения сама, а пользователю достаточно получать сообщения с помощью высокоуровневого интерфейса **receive**.