



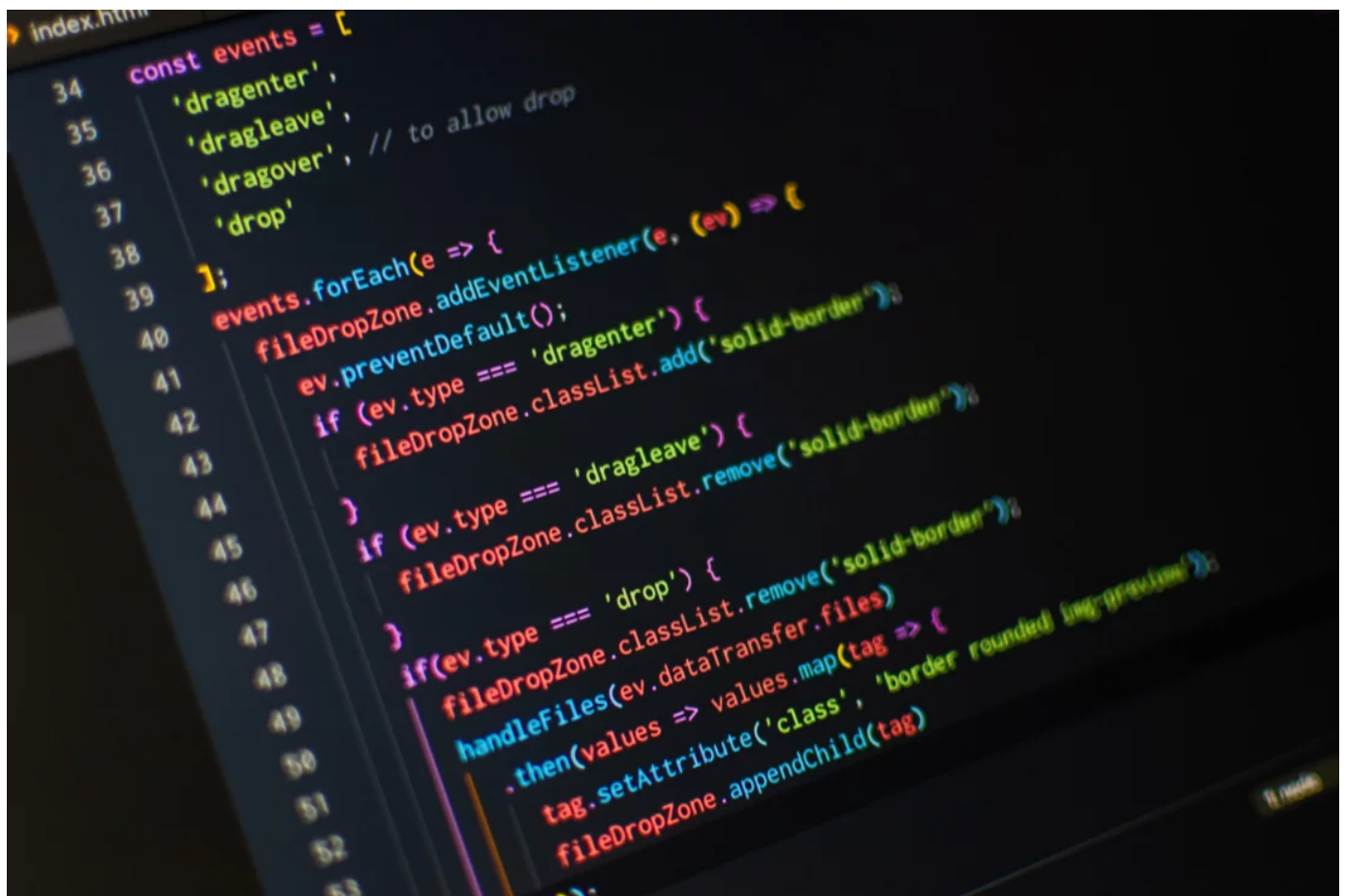
17 мая 2022 в 17:57

WebSocket: разбираем как работает

Веб-разработка*, Тестирование веб-сервисов*

Ожидает приглашения

Сегодня поговорим о том как работает WS, напишем простенький клиент на JS, обсудим как дебажить данный протокол ну и просто обсудим несколько интересных фактов. Погнали 🐼



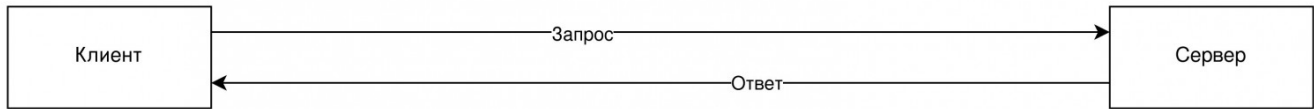
Первая ступень развития: HTTP

Стоит начать с того что такое HTTP?

HTTP - Протокол для передачи гипертекстовых данных (Hyper Text Transfer Protocol), который используется повсеместно. HTTP используется в клиент-серверной архитектуре, там всю работу можно показать с помощью одной диаграммы:

HTTP

Данная диаграмма иллюстрирует работу HTTP в клиент-серверной архитектуре



Особенности HTTP

- HTTP не поддерживает соединение, после того, как отдает ответ на запрос.
- HTTP обязует клиентов заранее оговаривать действие, которое клиент хочет сделать в заголовке (HTTP Headers) - GET, POST, PUT, DELETE
- Мы отправляем заголовок что хотим сделать каждый раз, как обраемся к серверу

Существует большое количество сайтов, с помощью которых вы можете посмотреть как работает HTTP, давайте возьмем забавный сайт с REST API по мультивселенной Рик и Морти — <https://rickandmortyapi.com/documentation>.

Вы можете отправить запрос с помощью Postman или обычного cURL, я буду использовать второй 😊 Давайте возьмем информацию о персонаже (Рике) и посмотрим что нам пришлёт сервер, для этого используем данную ссылку:
<https://rickandmortyapi.com/api/character/1>

```
# Отправляем запрос с помощью cURL и парсим пришедший ответ с помощью JQ
curl https://rickandmortyapi.com/api/character/1 | jq
```

После данной команды мы получим следующий ответ. Как мы видим мы просто отправили запрос на получение информации с сервера (GET), сервер отдал нам информацию и после этого мы разрываем соединение. После того как мы получим ответ мы ничего не знаем о сервере 👁👁



```
> curl https://rickandmortyapi.com/api/character/1 | jq
% Total    % Received % Xferd  Average Speed   Time    Time     Current
                                 Dload  Upload   Total   Spent    Left   Speed
100    2719    100    2719    0     0    10789      0 --:--:-- --:--:-- --:--:-- 10747

{
  "id": 1,
  "name": "Rick Sanchez",
  "status": "Alive",
  "species": "Human",
  "type": "",
  "gender": "Male",
  "origin": {
    "name": "Earth (C-137)",
    "url": "https://rickandmortyapi.com/api/location/1"
  },
  "location": {
    "name": "Citadel of Ricks",
    "url": "https://rickandmortyapi.com/api/location/3"
  },
  "image": "https://rickandmortyapi.com/api/character/avatar/1.jpeg",
  "episode": [
    "https://rickandmortyapi.com/api/episode/1",
    "https://rickandmortyapi.com/api/episode/2",
    "https://rickandmortyapi.com/api/episode/3",
    "https://rickandmortyapi.com/api/episode/4",
    "https://rickandmortyapi.com/api/episode/5",
    "https://rickandmortyapi.com/api/episode/6",
    "https://rickandmortyapi.com/api/episode/7",
    "https://rickandmortyapi.com/api/episode/8",
    "https://rickandmortyapi.com/api/episode/9",
    "https://rickandmortyapi.com/api/episode/10",
    "https://rickandmortyapi.com/api/episode/11",
    "https://rickandmortyapi.com/api/episode/12",
    "https://rickandmortyapi.com/api/episode/13",
    "https://rickandmortyapi.com/api/episode/14",
    "https://rickandmortyapi.com/api/episode/15",
    "https://rickandmortyapi.com/api/episode/16",
    "https://rickandmortyapi.com/api/episode/17",
    "https://rickandmortyapi.com/api/episode/18",
    "https://rickandmortyapi.com/api/episode/19",
    "https://rickandmortyapi.com/api/episode/20",
    "https://rickandmortyapi.com/api/episode/21"
  ]
}
```

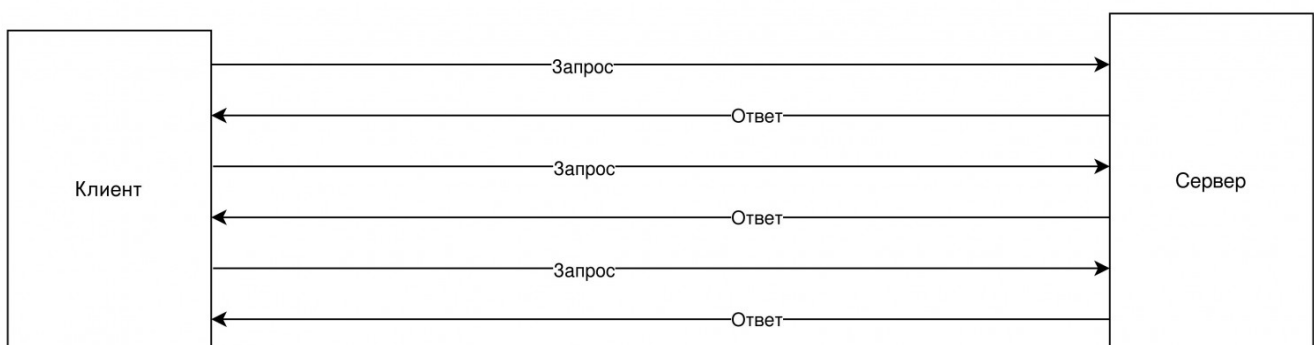
Нам пришли данные в формате JSON как ответ от сервера

Вторая ступень: AJAX

AJAX - асинхронные запросы с помощью JavaScript (Asynchronous JavaScript and XML). AJAX преследует все те же цели, что и HTTP, только делает это уже асинхронно. Если ранее нужно было для каждого запроса прописывать свой URL и перезагружать страницу, то теперь можно просто использовать AJAX и он сам будет отправлять нужные URL серверу и получать данные.

AJAX

Демонстрация работы AJAX-запросов



Особенности AJAX

- Все ещё обычный запрос, который не поддерживает соединение, после того, как отдает ответ на запрос.
- Все ещё заранее оговариваем действие, которое клиент хочет сделать в заголовке (HTTP Headers) - GET, POST, PUT, DELETE
- Мы отправляем заголовок что хотим сделать каждый раз, как обраемся к серверу
- Теперь мы делаем это асинхронно благодаря JavaScript

Самым простым примером AJAX является следующая реализация:

```
// Реализация на NodeJS ^17.5
// Будет работать в браузерах ~если вы не на Internet Explorer 8~

/**
 * Метод для того чтобы показать вывод AJAX-запроса
 * @param {string} url - ссылка, которую будем подтягивать
 * @returns {void}
 */
function showAJAXResponse(url) {

    // Выполняем запрос на сервер
    fetch(url)
        .then(res => res.json()) // Формируем JSON
        .then(data => console.log(data)); // Выводим
}

// Отправляем AJAX запрос 😊
showAJAXResponse('https://rickandmortyapi.com/api/character/1');
```

Вот что мы получим в итоге:



Мы можем выполнить множество таких запросов (серьезно, хоть 1000, если сервер позволит), как мы можем увидеть мы ничего не перезагружаем (нам даже перезагружать нечего 🤪), мы делаем все на бэк-энде Node.js)

Как мы можем увидеть, мы все ещё не держим связь с сервером. Мы отправили запрос, получили ответ и все 🤪. Что дальше происходит с сервером нам неизвестно.

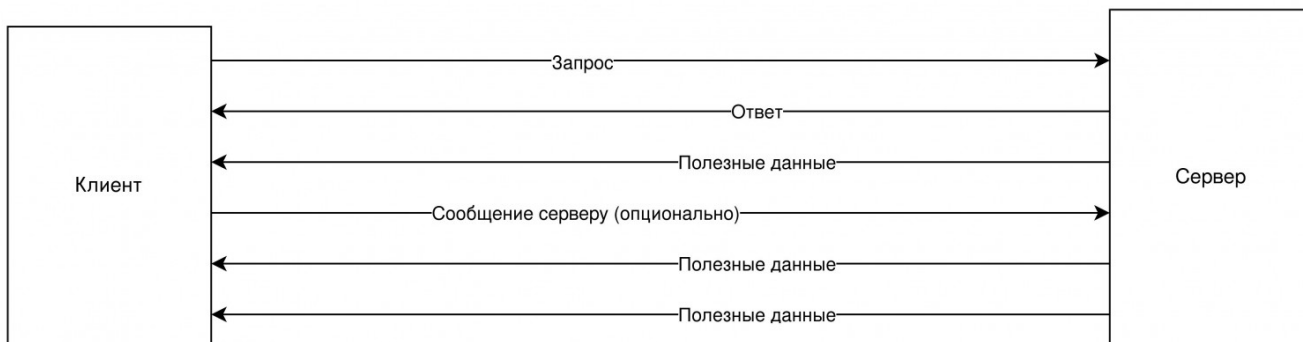
Третья ступень: WS или WebSocket

WebSocket - протокол для общения между клиентом и сервером, предоставляющий двухстороннее общение сверх протокола TCP.

Мы подключаем WS один раз, а затем сервер может отдавать нам ответы тогда, когда посчитает нужным:

WS

Демонстрация работы WebSocket-соединения



Как это работает?

Первое что мы делаем — отправляем обычный TCP-запрос на сервер, мы говорим, что хотим подключиться к серверу и ждём от него ответа. Такой процесс называется “рукопожатие” (*Handshake*), он используется повсеместно, например когда вы подключаетесь к роутеру ваш телефон отправляет запрос роутеру с ключами, роутер отвечает ОК и вы успешно подключаетесь.


Затем происходит обмен данными: допустим один из множества клиентов отправил HTTP-запрос серверу и нужно отдать ответ не только одному клиенту, а целой сети! Сервер в таком случае отдаст обычный ответ отправителю запроса, а всем другим пришлёт пакеты по WebSocket-соединению с полезными данными.

Разберем более подробно на примере. Вы — клиент, отправляете запрос серверу на подключение. Запрос и ответ будут выглядеть примерно так👁:

```
// Отправляем запрос серверу по ссылке example.com/connect-to-ws
// Вот что примерно мы пришлём:
GET /connect-to-ws HTTP/1.1
Host: example.com:8000
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGh1IHNBhXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
```

```
// А вот что нам на такой запрос ответит сервер при успешном рукопожатии:
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
```

```
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

Как мы видим сервер ответил не кодом 200 (успешное завершение запроса), а 101 — переключение протоколов. Это происходит потому, что мы отправили HTTP запрос, а хотим получить не только HTTP-ответ, а ещё и другие ответы по WS, сервер как бы предупреждает клиент, что будет присылать ответы множество раз  BACK

А как сервер узнает, что мы до сих пор подключены? 😊

Ответ на данный вопрос достаточно легкий — сервер и клиент играют в пинг-понг 😊 🍷 🤖

Сервер периодически присылает ответ по WS с просьбой о действии - послать запрос на сервер. Если клиент отвечает до истечения тайм-аута — он подключен, если нет, то происходит разрыв соединения до следующего рукопожатия 🙌

Полезно знать, что все ответы от сервера по WS игнорируются клиентом и сервером, до того как случится рукопожатие 🙌

Почему соединение называется двухсторонним (дуплексным), а ответы мы получаем только от сервера?

На самом деле мы не только получаем ответы от сервера, а ещё и можем в двухстороннем порядке отправлять через WS запросы! 🤖

Чтобы лучше понять, давайте рассмотрим лёгенький и полностью задокументированный код на JavaScript:

```
// Создаем WS-объект, с помощью него будем рулить потоками
// (отправка, принятие запросов)
const myWS = new WebSocket(url, protocols);

// До того как сервер и клиент совершат рукопожатие
// статус у WS-клиента будет CONNECTING
console.log(myWS.readyState); // CONNECTING

// После того как рукопожатие (Handshake) пройдет успешно
// readyState будет OPEN
console.log(myWS.readyState); // OPEN
```

После того как мы открыли соединение по WS мы сразу же можем отправить сообщение серверу:

```
/*
Не забываем, что мы можем отправить сообщение серверу
только если соединение открыто
Поместим все общение с сервером внутрь ивента onopen,
именно он срабатывает, когда соединение открыто
*/
myWS.onopen = function (event) {

    // Отправляем сообщение по WS
    myWS.send('Привет, сервер!');
}
```

Сервер получит данный запрос и возможно захочет ответить, но мы не сможем прочесть ответ! Почему? Да просто потому что у нас нет слушателя на событие получения сообщения от сервера 😊 Сделаем же его:

```
// Вешаем слушатель на принятие сообщений
myWS.onmessage = (event) => {

    // Делаем с данными все что захотим, но я их просто выведу 😊
    console.log(event.data);
}
```

Заккрытие соединения

Для закрытия соединения мы должны отправить запрос серверу, а он по истечению таймаута тоже должен отправить ответ на подтверждение закрытия. В JavaScript это делается одним методом 😊

```
// Закрываем соединение
myWS.close();

// Ну и естественно слушаем событие onclose, чтобы выполнить какие-то действия
myWS.onclose = (event) => {
    // ...
};
```

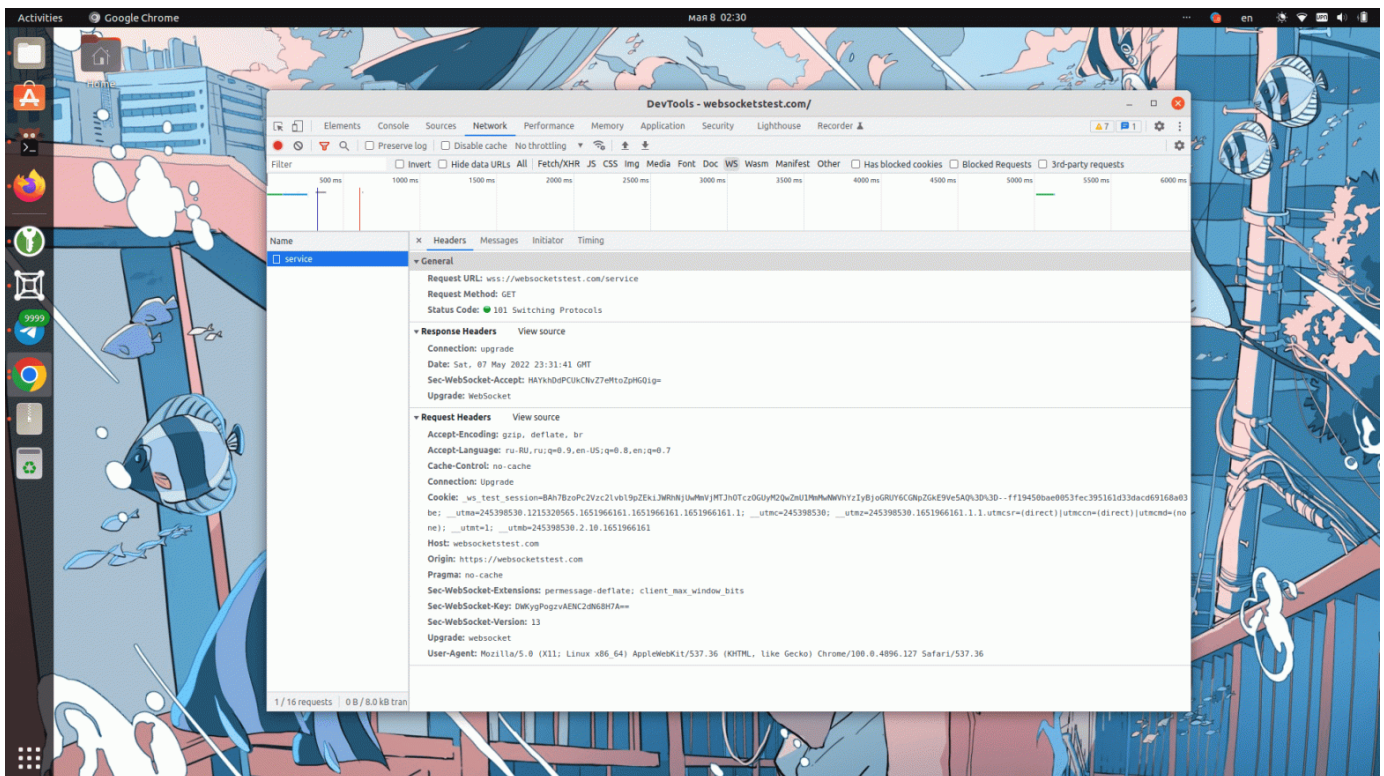

Особенности WS

- Поддерживает двухстороннее соединение в реальном времени
- Отправляет заголовок только один раз

Дебаггинг WS

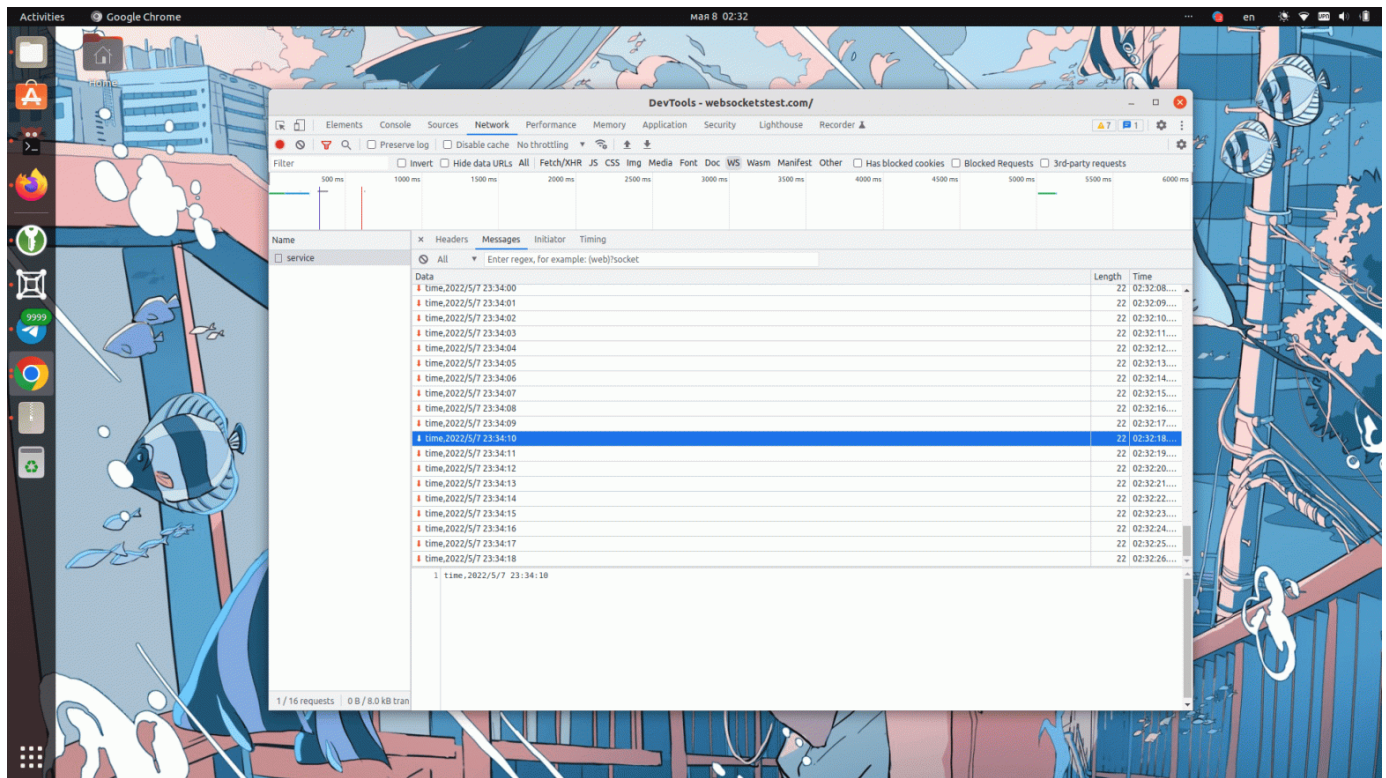
Отлаживать WS-соединение совсем несложно 😊 Рассмотрим пример отладки WS на Google Chrome 🌐, перейдем на данный сайт: <https://websockettest.com/>

Откроем DevTools, выберем вкладку Networks и перейдем в таб WS:



Как мы видим ответ от сервера действительно 101 Switching Protocols, однако как нам увидеть данные, которые приходят по WS, вкладки Response же нет 🤔

Вкладки Response нет, зато появилась новая - Messages. Открываем её и видим там примерно следующее:



Красной стрелкой вниз показаны пакеты, которые пришли нам (пусть вас не вводит в заблуждение красная стрелка, это не упавшие, а пришедшие пакеты), отправленные пакеты в свою очередь будут показаны зелёной стрелкой, которая стремится вверх↑

Вот и все 😊 Если вам было интересно читать статью и вы хотите больше такого контента, то можете перейти в телеграм-канал и подписаться, там много интересного материала ✨ Был рад поделиться информацией, увидимся ещё не раз 🙌

Теги: websocket, ws, web, devtools, http, ajax

Хабы: Веб-разработка, Тестирование веб-сервисов

Данная статья не подлежит комментированию, поскольку её автор ещё не является полноправным участником сообщества. Вы сможете связаться с автором только после того, как он получит приглашение от кого-либо из участников сообщества. До этого момента его username будет скрыт псевдонимом.

О ПЕСОЧНИЦЕ

Это [«Песочница»](#) — раздел, в который попадают дебютные публикации пользователей, желающих стать полноправными участниками сообщества.

Если у вас есть [приглашение](#), отправьте его автору понравившейся публикации — тогда её смогут прочитать и обсудить все остальные пользователи Хабра.

Чтобы исключить предвзятость при оценке, все публикации анонимны, псевдонимы показываются случайным образом.

О МОДЕРАЦИИ

Не надо пропускать:

- рекламные и PR-публикации
- вопросы и просьбы (для них есть [Хабр Q&A](#));
- вакансии (используйте [Хабр Карьеру](#))
- статьи, ранее опубликованные на других сайтах;
- статьи без правильно расставленных знаков препинания, со смайликами, с обилием восклицательных знаков, неоправданным выделением слов и предложений и другим неуместным форматированием текста;
- жалобы на компании и предоставляемые услуги;
- низкокачественные переводы;
- куски программного кода без пояснений;
- односложные статьи;
- статьи, слабо относящиеся к IT-тематике или не относящиеся к ней вовсе.

Ваш аккаунт	Разделы	Информация	Услуги
Войти	Статьи	Устройство сайта	Корпоративный блог
Регистрация	Новости	Для авторов	Медийная реклама
	Хабы	Для компаний	Нативные проекты
	Компании	Документы	Образовательные
	Авторы	Соглашение	программы
	Песочница	Конфиденциальность	Стартапам



Настройка языка

Техническая поддержка

