

[Раздел](#)[Сетевые запросы](#)[Навигация по уроку](#)[Простой пример](#)[Открытие веб-сокета](#)[Передача данных](#)[Ограничение скорости](#)[Закрытие подключения](#)[Состояние соединения](#)[Пример чата](#)[Итого](#)[Комментарии](#)[Поделиться](#)[Редактировать на GitHub](#)[Курсы](#) [Форум](#) [Тесты знаний](#) [Скринкасты](#) ▼[Купить EPUB/PDF](#)[🏠](#) → [Сетевые запросы](#)

30 января 2023 г.

# WebSocket

Протокол `WebSocket` («веб-сокет»), описанный в спецификации [RFC 6455](#), обеспечивает возможность обмена данными между браузером и сервером через постоянное соединение. Данные передаются по нему в обоих направлениях в виде «пакетов», без разрыва соединения и дополнительных HTTP-запросов.

`WebSocket` особенно хорош для сервисов, которые нуждаются в постоянном обмене данными, например онлайн игры, торговые площадки, работающие в реальном времени, и т.д.

## Простой пример

Чтобы открыть веб-сокет-соединение, нам нужно создать объект `new WebSocket`, указав в `url`-адресе специальный протокол `ws`:

```
1 let socket = new WebSocket("ws://javascript.info");
```

Также существует протокол `wss://`, использующий шифрование. Это как HTTPS для веб-сокетов.

### Всегда предпочитайте `wss://`

Протокол `wss://` не только использует шифрование, но и обладает повышенной надёжностью.

Это потому, что данные `ws://` не зашифрованы, видны для любого посредника. Старые прокси-серверы не знают о `WebSocket`, они могут увидеть «странные» заголовки и закрыть соединение.

С другой стороны, `wss://` — это `WebSocket` поверх TLS (так же, как HTTPS — это HTTP поверх TLS), безопасный транспортный уровень шифрует данные от отправителя и расшифровывает на стороне получателя. Пакеты данных передаются в зашифрованном виде через прокси, которые не могут видеть, что внутри, и всегда пропускают их.

Как только объект `WebSocket` создан, мы должны слушать его события. Их всего 4:

- **open** — соединение установлено,
- **message** — получены данные,
- **error** — ошибка,
- **close** — соединение закрыто.

...А если мы хотим отправить что-нибудь, то вызов `socket.send(data)` сделает это.

Вот пример:



Раздел

Сетевые запросы

Навигация по уроку

Простой пример

Открытие веб-сокета

Передача данных

Ограничение скорости

Закрытие подключения

Состояние соединения

Пример чата

Итого

Комментарии

Поделиться

Редактировать на GitHub



```
1 let socket = new WebSocket("wss://javascript.info/article/websocket/demo/hello");
2
3 socket.onopen = function(e) {
4   alert("[open] Соединение установлено");
5   alert("Отправляем данные на сервер");
6   socket.send("Меня зовут Джон");
7 };
8
9 socket.onmessage = function(event) {
10  alert(`[message] Данные получены с сервера: ${event.data}`);
11 };
12
13 socket.onclose = function(event) {
14   if (event.wasClean) {
15     alert(`[close] Соединение закрыто чисто, код=${event.code} причина=${event.reason}`);
16   } else {
17     // например, сервер убил процесс или сеть недоступна
18     // обычно в этом случае event.code 1006
19     alert(`[close] Соединение прервано`);
20   }
21 };
22
23 socket.onerror = function(error) {
24   alert(`[error]`);
25 };
```



Для демонстрации есть небольшой пример сервера [server.js](#), написанного на Node.js, для запуска примера выше. Он отвечает «Привет с сервера, Джон», после ожидает 5 секунд и закрывает соединение.



Так вы увидите события `open` → `message` → `close`.

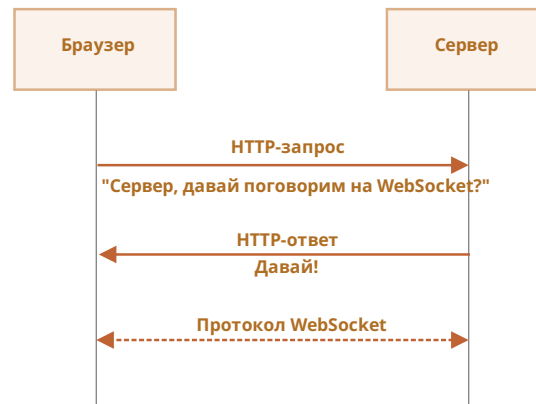
В общем-то, всё, мы уже можем общаться по протоколу WebSocket. Просто, не так ли?

Теперь давайте поговорим более подробно.

## Открытие веб-сокета

Когда `new WebSocket(url)` создан, он тут же сам начинает устанавливать соединение.

Браузер, при помощи специальных заголовков, спрашивает сервер: «Ты поддерживаешь Websocket?» и если сервер отвечает «да», они начинают работать по протоколу WebSocket, который уже не является HTTP.

[Раздел](#)[Сетевые запросы](#)[Навигация по уроку](#)[Простой пример](#)[Открытие веб-сокета](#)[Передача данных](#)[Ограничение скорости](#)[Заккрытие подключения](#)[Состояние соединения](#)[Пример чата](#)[Итого](#)[Комментарии](#)[Поделиться](#)[Редактировать на GitHub](#)

Вот пример заголовков для запроса, который делает `new WebSocket("wss://javascript.info/chat")`.

```
1 GET /chat
2 Host: javascript.info
3 Origin: https://javascript.info
4 Connection: Upgrade
5 Upgrade: websocket
6 Sec-WebSocket-Key: Iv8io/9s+lYFgZWcXczP8Q==
7 Sec-WebSocket-Version: 13
```

- `Origin` – источник текущей страницы (например `https://javascript.info`). Объект `WebSocket` по своей природе не завязан на текущий источник. Нет никаких специальных заголовков или других ограничений. Старые сервера всё равно не могут работать с `WebSocket`, поэтому проблем с совместимостью нет. Но заголовок `Origin` важен, так как он позволяет серверу решать, использовать ли `WebSocket` с этим сайтом.
- `Connection: Upgrade` – сигнализирует, что клиент хотел бы изменить протокол.
- `Upgrade: websocket` – запрошен протокол «websocket».
- `Sec-WebSocket-Key` – случайный ключ, созданный браузером для обеспечения безопасности.
- `Sec-WebSocket-Version` – версия протокола `WebSocket`, текущая версия 13.

#### Запрос `WebSocket` нельзя эмулировать

Мы не можем использовать `XMLHttpRequest` или `fetch` для создания такого HTTP-запроса, потому что JavaScript не позволяет устанавливать такие заголовки.

Если сервер согласен переключиться на `WebSocket`, то он должен отправить в ответ код 101:

```
1 101 Switching Protocols
2 Upgrade: websocket
3 Connection: Upgrade
4 Sec-WebSocket-Accept: hsB1buDTkk24srzE0TBu1ZA1C2g=
```

[Раздел](#)[Сетевые запросы](#)[Навигация по уроку](#)[Простой пример](#)[Открытие веб-сокета](#)[Передача данных](#)[Ограничение скорости](#)[Закрытие подключения](#)[Состояние соединения](#)[Пример чата](#)[Итого](#)[Комментарии](#)[Поделиться](#)[Редактировать на GitHub](#)

Здесь `Sec-WebSocket-Accept` – это `Sec-WebSocket-Key`, перекодированный с помощью специального алгоритма. Браузер использует его, чтобы убедиться, что ответ соответствует запросу.

После этого данные передаются по протоколу `WebSocket`, и вскоре мы увидим его структуру («фреймы»). И это вовсе не `HTTP`.

## Расширения и подпротоколы

Могут быть дополнительные заголовки `Sec-WebSocket-Extensions` и `Sec-WebSocket-Protocol`, описывающие расширения и подпротоколы.

Например:

- `Sec-WebSocket-Extensions: deflate-frame` означает, что браузер поддерживает сжатие данных. Расширение – это что-то, связанное с передачей данных, расширяющее сам протокол `WebSocket`. Заголовок `Sec-WebSocket-Extensions` отправляется браузером автоматически со списком всевозможных расширений, которые он поддерживает.
- `Sec-WebSocket-Protocol: soap, wamp` означает, что мы будем передавать не только произвольные данные, но и данные в протоколах [SOAP](#) или `WAMP` (The WebSocket Application Messaging Protocol) – «протокол обмена сообщениями `WebSocket` приложений»). То есть этот заголовок описывает не передачу, а формат данных, который мы собираемся использовать. Официальные подпротоколы `WebSocket` регистрируются в [каталоге IANA](#).

Этот необязательный заголовок ставим мы сами, передавая массив подпротоколов вторым параметром `new WebSocket`, вот так:

```
1 let socket = new WebSocket("wss://javascript.info/chat", ["soap", "wamp"]);
```

Сервер должен ответить перечнем протоколов и расширений, которые он может использовать.

Например, запрос:

```
1 GET /chat
2 Host: javascript.info
3 Upgrade: websocket
4 Connection: Upgrade
5 Origin: https://javascript.info
6 Sec-WebSocket-Key: Iv8io/9s+lYFgZWcXczP8Q==
7 Sec-WebSocket-Version: 13
8 Sec-WebSocket-Extensions: deflate-frame
9 Sec-WebSocket-Protocol: soap, wamp
```

Ответ:

```
1 101 Switching Protocols
2 Upgrade: websocket
3 Connection: Upgrade
4 Sec-WebSocket-Accept: hsB1buDTkk24srzE0TBu1ZA1C2g=
5 Sec-WebSocket-Extensions: deflate-frame
6 Sec-WebSocket-Protocol: soap
```

Здесь сервер отвечает, что поддерживает расширение – `deflate-frame` и может использовать только протокол `SOAP` из всего списка запрошенных подпротоколов.

[Раздел](#)[Сетевые запросы](#)[Навигация по уроку](#)[Простой пример](#)[Открытие веб-сокета](#)[Передача данных](#)[Ограничение скорости](#)[Закрытие подключения](#)[Состояние соединения](#)[Пример чата](#)[Итого](#)[Комментарии](#)[Поделиться](#)[Редактировать на GitHub](#)

## Передача данных

Поток данных в WebSocket состоит из «фреймов», фрагментов данных, которые могут быть отправлены любой стороной, и которые могут быть следующих видов:

- «текстовые фреймы» – содержат текстовые данные, которые стороны отправляют друг другу.
- «бинарные фреймы» – содержат бинарные данные, которые стороны отправляют друг другу.
- «пинг-понг фреймы» используется для проверки соединения; отправляется с сервера, браузер реагирует на них автоматически.
- также есть «фрейм закрытия соединения» и некоторые другие служебные фреймы.

В браузере мы напрямую работаем только с текстовыми и бинарными фреймами.

**Метод `WebSocket.send()` может отправлять и текстовые, и бинарные данные.**

Вызов `socket.send(body)` принимает `body` в виде строки или любом бинарном формате включая `Blob`, `ArrayBuffer` и другие. Дополнительных настроек не требуется, просто отправляем в любом формате.

**При получении данных, текст всегда поступает в виде строки. А для бинарных данных мы можем выбрать один из двух форматов: `Blob` или `ArrayBuffer`.**

Это задаётся свойством `socket.binaryType`, по умолчанию оно равно `"blob"`, так что бинарные данные поступают в виде `Blob`-объектов.

`Blob` – это высокоуровневый бинарный объект, он напрямую интегрируется с `<a>`, `<img>` и другими тегами, так что это вполне удобное значение по умолчанию. Но для обработки данных, если требуется доступ к отдельным байтам, мы можем изменить его на `"arraybuffer"`:

```
1 socket.binaryType = "arraybuffer";
2 socket.onmessage = (event) => {
3   // event.data является строкой (если текст) или arraybuffer (если двоичные данные)
4   };
```

## Ограничение скорости

Представим, что наше приложение генерирует много данных для отправки. Но у пользователя медленное соединение, возможно, он в интернете с мобильного телефона и не из города.

Мы можем вызывать `socket.send(data)` снова и снова. Но данные будут буферизованы (сохранены) в памяти и отправлены лишь с той скоростью, которую позволяет сеть.

Свойство `socket.bufferedAmount` хранит количество байт буферизованных данных на текущий момент, ожидающих отправки по сети.

Мы можем изучить его, чтобы увидеть, действительно ли сокет доступен для передачи.

```
1 // каждые 100мс проверить сокет и отправить больше данных,
2 // только если все текущие отсланы
3 setInterval(() => {
4   if (socket.bufferedAmount == 0) {
5     socket.send(moreData());
6   }
7 }, 100);
```

[Раздел](#)[Сетевые запросы](#)[Навигация по уроку](#)[Простой пример](#)[Открытие веб-сокета](#)[Передача данных](#)[Ограничение скорости](#)[Закрытие подключения](#)[Состояние соединения](#)[Пример чата](#)[Итого](#)[Комментарии](#)[Поделиться](#)[Редактировать на GitHub](#)

## Закрытие подключения

Обычно, когда сторона хочет закрыть соединение (браузер и сервер имеют равные права), они отправляют «фрейм закрытия соединения» с кодом закрытия и указывают причину в виде текста.

Метод для этого:

```
1 socket.close([code], [reason]);
```

- `code` – специальный WebSocket-код закрытия (не обязателен).
- `reason` – строка с описанием причины закрытия (не обязательна).

Затем противоположная сторона в обработчике события `close` получит и код `code` и причину `reason`, например:

```
1 // закрывающая сторона:
2 socket.close(1000, "работа закончена");
3
4 // другая сторона:
5 socket.onclose = event => {
6   // event.code === 1000
7   // event.reason === "работа закончена"
8   // event.wasClean === true (закрыто чисто)
9 };
```

`code` – это не любое число, а специальный код закрытия WebSocket.

Наиболее распространённые значения:

- `1000` – по умолчанию, нормальное закрытие,
- `1006` – невозможно установить такой код вручную, указывает, что соединение было потеряно (нет фрейма закрытия).

Есть и другие коды:

- `1001` – сторона отключилась, например сервер выключен или пользователь покинул страницу,
- `1009` – сообщение слишком большое для обработки,
- `1011` – непредвиденная ошибка на сервере,
- ...и так далее.

Полный список находится в [RFC6455, §7.4.1](#).

Коды WebSocket чем-то похожи на коды HTTP, но они разные. В частности, любые коды меньше `1000` зарезервированы. Если мы попытаемся установить такой код, то получим ошибку.

```
1 // в случае, если соединение сброшено
2 socket.onclose = event => {
3   // event.code === 1006
4   // event.reason === ""
5   // event.wasClean === false (нет закрывающего кадра)
6 };
```

## Состояние соединения

Раздел

Сетевые запросы

Навигация по уроку

Простой пример

Открытие веб-сокета

Передача данных

Ограничение скорости

Закрытие подключения

Состояние соединения

Пример чата

Итого

Комментарии

Поделиться

Редактировать на GitHub



Чтобы получить состояние соединения, существует дополнительное свойство `socket.readyState` со значениями:

- **0** – «CONNECTING»: соединение ещё не установлено,
- **1** – «OPEN»: обмен данными,
- **2** – «CLOSING»: соединение закрывается,
- **3** – «CLOSED»: соединение закрыто.

## Пример чата

Давайте рассмотрим пример чата с использованием WebSocket API и модуля WebSocket сервера Node.js <https://github.com/websockets/ws>. Основное внимание мы, конечно, уделим клиентской части, но и серверная весьма проста.

HTML: нам нужна форма `<form>` для отправки данных и `<div>` для отображения сообщений:

```
1 <!-- форма сообщений -->
2 <form name="publish">
3   <input type="text" name="message">
4   <input type="submit" value="Отправить">
5 </form>
6
7 <!-- div с сообщениями -->
8 <div id="messages"></div>
```

От JavaScript мы хотим 3 вещи:

1. Открыть соединение.
2. При отправке формы пользователем – вызвать `socket.send(message)` для сообщения.
3. При получении входящего сообщения – добавить его в `div#messages`.

Вот код:

```
1 let socket = new WebSocket("wss://javascript.info/article/websocket/chat/ws");
2
3 // отправка сообщения из формы
4 document.forms.publish.onsubmit = function() {
5   let outgoingMessage = this.message.value;
6
7   socket.send(outgoingMessage);
8   return false;
9 };
10
11 // получение сообщения - отобразить данные в div#messages
12 socket.onmessage = function(event) {
13   let message = event.data;
14
15   let messageElem = document.createElement('div');
16   messageElem.textContent = message;
17   document.getElementById('messages').prepend(messageElem);
18 }
```

Серверный код выходит за рамки этой главы. Здесь мы будем использовать Node.js, но вы не обязаны это делать. Другие

Раздел

Сетевые запросы

Навигация по уроку

Простой пример

Открытие веб-сокета

Передача данных

Ограничение скорости

Закрытие подключения

Состояние соединения

Пример чата

Итого

Комментарии

Поделиться

Редактировать на GitHub



платформы также поддерживают средства для работы с WebSocket.


Серверный алгоритм действий будет таким:

1. Создать `clients = new Set()` – набор сокетов.
2. Для каждого принятого веб-сокета – добавить его в набор `clients.add(socket)` и поставить ему обработчик события `message` для приёма сообщений.
3. Когда сообщение получено: перебрать клиентов `clients` и отправить его всем.
4. Когда подключение закрыто: `clients.delete(socket)`.

```
1  const ws = new require('ws');
2  const wss = new ws.Server({noServer: true});
3
4  const clients = new Set();
5
6  http.createServer((req, res) => {
7    // в реальном проекте здесь может также быть код для обработки отличных от websocket
8    // здесь мы работаем с каждым запросом как с веб-сокетом
9    wss.handleUpgrade(req, req.socket, Buffer.alloc(0), onSocketConnect);
10 });
11
12 function onSocketConnect(ws) {
13   clients.add(ws);
14
15   ws.on('message', function(message) {
16     message = message.slice(0, 50); // максимальный размер сообщения 50
17
18     for(let client of clients) {
19       client.send(message);
20     }
21   });
22
23   ws.on('close', function() {
24     clients.delete(ws);
25   });
26 }
```

Вот рабочий пример:

14  
2  
2  
7



Вы также можете скачать его (верхняя правая кнопка в ифрейме) и запустить локально. Только не забудьте установить [Node.js](#) и выполнить команду `npm install ws` до запуска.

## Итого

WebSocket – это современный способ иметь постоянное соединение между браузером и сервером.

- Нет ограничений, связанных с кросс-доменными запросами.
- Имеют хорошую поддержку браузерами.
- Могут отправлять/получать как строки, так и бинарные данные.



Раздел

Сетевые запросы

Навигация по уроку

Простой пример

Открытие веб-сокета

Передача данных

Ограничение скорости

Закрытие подключения

Состояние соединения

Пример чата

Итого

Комментарии

Поделиться

Редактировать на GitHub



API прост.

Методы:

- `socket.send(data)` ,
- `socket.close([code], [reason])` .

События:

- `open` ,
- `message` ,
- `error` ,
- `close` .

WebSocket сам по себе не содержит такие функции, как переподключение при обрыве соединения, аутентификацию пользователей и другие механизмы высокого уровня. Для этого есть клиентские и серверные библиотеки, а также можно реализовать это вручную.

Иногда, чтобы добавить WebSocket к уже существующему проекту, WebSocket-сервер запускают параллельно с основным сервером. Они совместно используют одну базу данных. Запросы к WebSocket отправляются на `wss://ws.site.com` — поддомен, который ведёт к WebSocket-серверу, в то время как `https://site.com` ведёт на основной HTTP-сервер.

Конечно, возможны и другие пути интеграции.

Проводим [курсы по JavaScript и фреймворкам](#).

Комментарии

перед тем как писать...

Раздел

Сетевые запросы

Навигация по уроку

Простой пример

Открытие веб-сокета

Передача данных

Ограничение скорости

Закрытие подключения

Состояние соединения

Пример чата

Итого

Комментарии

Поделиться

Редактировать на GitHub

☰

📖

<

38 Комментариев

1

Войти ▾

G

Присоединиться к обсуждению...

войти с помощью

или через DISQUS ?

Имя

12

Поделиться

Лучшие

Новые

Старые

E


Евгений Сазонов

3 месяца назад

Отличная подача данных, для меня Learn Javascript это образец качественной информации без лишней воды и очень доступно. Спасибо большое!

10

Ответить •




Dave looking for dick 18

год назад

Wow... Looks good

134

Ответить •



Drino955

5 месяцев назад

да, дик дик

10

Ответить •

T

Talgat Abdukulov

год назад

А через сколько соединение разрывается при простое?

00

Ответить •

A

avel-front

2 года назад

В общих чертах запечатлил в памяти без серверных подробностей, через годик-два вернусь.

11

Ответить •

P

Павел Рамков

2 года назад

Здравствуйте. Хорошая статья, все понятно. Уточните а что значит постоянное соединение, это когда оно установлено и до момента закрытия? А как быть с соединением при перезагрузке, считается ли нормой разрыв и восстановление соединения после перезагрузки или как-то этот вопрос нужно решать. Подскажите как и нужно ли тут что-то

00

Ответить •

Стр. 10 из 15

07.10.2023, 11:45

Раздел

Сетевые запросы

Навигация по уроку

Простой пример

Открытие веб-сокета

Передача данных

Ограничение скорости

Закрытие подключения

Состояние соединения

Пример чата

Итого

Комментарии

Поделиться

Редактировать на GitHub

**Алексей Обухов**

→ Павел Гамков

год назад

на все вопросы ответ да. рекомендую запустить локально приложение чатик (вышеприведенное) и все вопросы отпадут

0

0

Ответить •

**Н****Николай Мельников**

2 года назад

данный пример запускается только если js написан в html. если js в отдельном файле, то все запускается, но при нажатии кнопки отправить страница улетает с ошибкой 404, а текст сообщения улетает в строку браузера. Никто не подскажет почему так происходит и как это исправить?

0

0

Ответить •

**foxhable**

→ Николай Мельников

2 года назад

Потому что клиент делает запрос на js файл, а сервер не умеет отправлять его.

0

0

Ответить •

**Н****Николай Мельников**

→ foxhable

2 года назад

Я ещё новичок в этом плане. А не подскажите, как поправить? Там серверная часть меняется?

0

0

Ответить •

**foxhable**

→ Николай Мельников

2 года назад

Да, серверная. Когда клиент заходит на условный сайт "www.site.com/" - он кидает запрос с url "/", сервер видит, что идет запрос на этот юрл и отправляет ответ, в большинстве случаев это index.html. Если в index.html будут подключены стили и скрипты через отдельные файлы, то от клиента отправятся ещё запросы на получение этих файлов. Если не "научить" сервер обрабатывать запрос файлов со стилями и скриптами, то он их не отправит и естественно у клиента может произойти ошибка. Более точнее это называется роутинг.

В файле server.js есть строки:

```
if (...) {  
  ...  
} else if (req.url == '/') { // index.html  
  fs.createReadStream('./index.html').pipe(res);  
} else { // страница не найдена  
  res.writeHead(404);  
  res.end();  
}
```

Именно они отвечают за роутинг. Если не использовать сторонние библиотеки, то советую делать через конструкцию switch (в этом учебнике есть статья о нем).

2

0

Ответить •

**Hrach Haghverdyan**

2 года назад

Раздел

Сетевые запросы

Навигация по уроку

Простой пример

Открытие веб-сокета

Передача данных

Ограничение скорости

Закрытие подключения

Состояние соединения

Пример чата

Итого

Комментарии

Поделиться

Редактировать на GitHub



```
in index.html file edit to "let socket = new WebSocket('ws://localhost:8080/ws')"  
don't forgot in file folder open terminal and run 'node server.js', after that open in browser  
http://localhost:8080/  
everything are works !!!!  
for testing you can edit in server.js file  
for(let client of clients) {  
  client.send('from server: ${message}');  
}
```

1 0 Ответить •

D

Dmitriy

3 года назад

Спасибо за материал.

Ещё бы простенький пример серверной части на PHP.

3 1 Ответить •



Adam Itch

3 года назад

У меня чат на странице не заработал. Скачал архив, запустил локально тоже не работает. Потом поковырял код сервера и нашел причину:

```
ws.on('сообщение', function(message) { ... })  
ws.on('закреть', function() {...})
```

надо "сообщение" и "закреть" заменить на "message" и "close". Тогда все работает как надо.

И еще в index.html надо url присвоить ws://localhost:8080/ws иначе клиент коннектится на сервер [javascript.info](https://learn.javascript.info) и никакие изменения в коде файла server.js не работали.

10 0 Ответить •

S

sartor1

→ Adam Itch

год назад edited

Спасибо , без смены урл и аргументов в ф-ции "on" правда не работает.  
Не могу понять, почему сообщения уходят в "правильном", текстовом формате, а приходят в формате Blob. Т.е. у меня всё ещё работает не до конца  
UPD: Работает только если экранировать строку, которую отправляем на клиент, тут:

```
for(let client of clients) {  
  client.send(` ${message} `);  
}
```

1 0 Ответить •

A

Андреслав Козлов

→ Adam Itch

3 года назад

Спасибо!

0 0 Ответить •

G

Gorr1995

3 года назад

Раздел

Сетевые запросы

Навигация по уроку

Простой пример

Открытие веб-сокета

Передача данных

Ограничение скорости

Заккрытие подключения

Состояние соединения

Пример чата

Итого

Комментарии

Поделиться

Редактировать на GitHub



Чат не работает ((

1 0 Ответить •

**Vitaly Sokolov**

→ Gorr1995



3 года назад

Перезагрузите страницу и чат заработает. Наверное, там стоит какой-то таймаут для отключения socket connection.

3 1 Ответить •

**Мария Насонкина**

→ Gorr1995



3 года назад

пишет код 1006 в консоли))

0 0 Ответить •

Р

**Роман Василяки**

3 года назад

Будет 4 часть учебника по серверной части (node)?

0 0 Ответить •

**aLex**

→ Роман Василяки



3 года назад

Вот тут есть небольшой источник <https://metanit.com/web/nod...>, правда я сам за него еще не брался, но выглядит он как полноценный курс по node.js

2 0 Ответить •

А

**avel-front**

→ aLex



2 года назад

Сохранил, спасибо

0 0 Ответить •

**Schulzkafer**

→ aLex



2 года назад

метанит - отличный учебник, рекомендую

3 0 Ответить •

**mimic89**

→ Роман Василяки



3 года назад

Очень сомнительно.

Здесь есть скринкаст по ноде, но он не самый "новый", насколько я понимаю:

<https://learn.javascript.ru...>

1 0 Ответить •

Р

**Роман Василяки**

→ mimic89



3 года назад

Хоть что-то, пойду формировать общее представление о инструментах node. Спасибо.

0 0 Ответить •

Раздел

Сетевые запросы

Навигация по уроку

Простой пример

Открытие веб-сокета

Передача данных

Ограничение скорости

Закрытие подключения

Состояние соединения

Пример чата

Итого

Комментарии

Поделиться

Редактировать на GitHub



G

Gorr1995

→ mimic89

3 года назад

Будем откровенны , ты прав. Но он весьма хорош

0 0 Ответить •

C

Сергей

3 года назад

Поправьте socket.bufferType на socket.binaryType  
источник: <https://html.spec.whatwg.or...>

1 0 Ответить •



Шамиль Джакеев

→ Сергей

2 года назад

Открыл ПР с соответствующими правками.

0 0 Ответить •



Rail Batyrshin

4 года назад

Thanks

0 0 Ответить •

J

John Doe

4 года назад

Многое стало понятно, большое спасибо!

1 1 Ответить •

T

Tron\_KZ

4 года назад

Добрый день! В примере сервера "server.js" используется метод `on()`. Откуда взялся этот метод, если модуль 'events' не был полкючен к серверу?

0 1 Ответить •

I

I'm woked

→ Tron\_KZ

4 года назад

Это медот из ws, причем тут 'events' не очень понятно) Каждый модуль может иметь методы с любыми названиями

0 1 Ответить •

T

Tron\_KZ

→ I'm woked

4 года назад

Мой вопрос почему-то удалили. Можете дать ссылку на этот метод из ws? В документации по ws про метод `on` ничего не нашел.

0 0 Ответить •

VD

Viktar Daniliuk

→ Tron\_KZ

4 года назад

<https://www.npmjs.com/packa...>

0 1 Ответить •

Раздел

Сетевые запросы

Навигация по уроку

Простой пример

Открытие веб-сокета

Передача данных

Ограничение скорости

Закрытие подключения

Состояние соединения

Пример чата

Итого

Комментарии

Поделиться

Редактировать на GitHub

☰

📖

<

>

T

Трон\_KZ

→ Viktor Daniliuk

4 года назад

Добрый день! В общем дико угораю с чуваков которые не разобравшись в вопросе или не зная ответа, пишут всякую х\*\*ню. Зачем позориться то?? В общем, сам спросил, сам себе и отвечаю, и возможно тем кто в будущем задастся таким же вопросом: Когда сервер подключает модуль "ws", запускается файл импортируется файл index.js у которого внутри есть строка `WebSocket.Server = require('./lib/websocket-server');`, то есть импортируется файл "websocket-server.js". У этого файла, в свою очередь, внутри есть строка `const EventEmitter = require('events');`. Это значит, что модуль "ws" подключает модуль "events".

46

Ответить •

D

Дмитрий Толмачев

→ Трон\_KZ

год назад

умный... хочу все знать...

00

Ответить •

AD

A D

→ Трон\_KZ

4 года назад

нам необязательно знать как реализован метод ``on(..)`` чтобы им корректно пользоваться. важно знать как его правильно вызвать и что он делает. инкапсуляция же :)

51

Ответить •

A

Abbos Tajimov

4 года назад

Спасибо за хорошую статью

81

Ответить •

Подписаться

О защите персональных данных

Не продавайте мои данные

© 2007—2023 Илья Кантор | [о проекте](#) | [связаться с нами](#) | [пользовательское соглашение](#) | [политика конфиденциальности](#)

Стр. 15 из 15

07.10.2023, 11:45