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` :

```
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:

```
let socket = new WebSocket("wss://javascript.info/article/websocket/demo/hello");

socket.onopen = function(e) {
  alert("[open] Соединение установлено");
  alert("Отправляем данные на сервер");
  socket.send("Меня зовут Джон");
};

socket.onmessage = function(event) {
  alert(`[message] Данные получены с сервера: ${event.data}`);
};

socket.onclose = function(event) {
  if (event.wasClean) {
    alert(`[close] Соединение закрыто чисто, код=${event.code} причина=${event.reason}`);
  } else {
    // например, сервер убил процесс или сеть недоступна
    // обычно в этом случае event.code 1006
    alert('[close] Соединение прервано');
  }
};

socket.onerror = function(error) {
  alert(`[error]`);
};
```

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`.

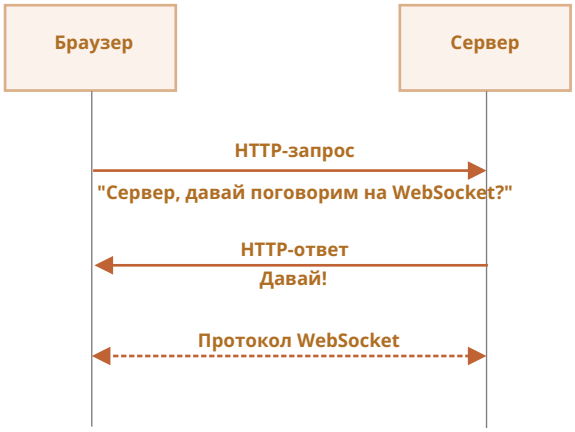
В общем-то, всё, мы уже можем общаться по протоколу WebSocket. Просто, не так ли?

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.



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

```
GET /chat
Host: javascript.info
Origin: https://javascript.info
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Key: Iv8io/9s+lYFgZWcXczP8Q==
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:

```
101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: hsBlbuDTkk24srzEOTBUlZAlC2g=
```

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:

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

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

Например, запрос:

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

Ответ:

```
101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: hsBlbuDTkk24srzEOTBUlZAlC2g=
Sec-WebSocket-Extensions: deflate-frame
Sec-WebSocket-Protocol: soap
```

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

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"`:

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

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.

```
// каждые 100мс проверить сокет и отправить больше данных,
// только если все текущие отосланы
setInterval(() => {
  if (socket.bufferedAmount == 0) {
    socket.send(moreData());
  }
}, 100);
```

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:

```
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:

```
// закрывающая сторона:
socket.close(1000, "работа закончена");

// другая сторона:
socket.onclose = event => {
  // event.code === 1000
  // event.reason === "работа закончена"
  // event.wasClean === true (закрыто чисто)
};
```

`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.

```
// в случае, если соединение сброшено
socket.onclose = event => {
```

```
  // event.code === 1006
  // event.reason === ""
  // event.wasClean === false (нет закрывающего кадра)
};
```

Состояние соединения

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:

```
<!-- форма сообщений -->
<form name="publish">
  <input type="text" name="message">
  <input type="submit" value="Отправить">
</form>

<!-- div с сообщениями -->
<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:

```
let socket = new WebSocket("wss://javascript.info/article/websocket/chat/ws");

// отправка сообщения из формы
document.forms.publish.onsubmit = function() {
  let outgoingMessage = this.message.value;

  socket.send(outgoingMessage);
  return false;
};

// получение сообщения - отобразить данные в div#messages
socket.onmessage = function(event) {
  let message = event.data;

  let messageElem = document.createElement('div');
  messageElem.textContent = message;
  document.getElementById('messages').prepend(messageElem);
}
```

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

```
const ws = new require('ws');
const wss = new ws.Server({noServer: true});

const clients = new Set();

http.createServer((req, res) => {
  // в реальном проекте здесь может также быть код для обработки отличных от websocket-запр
  // здесь мы работаем с каждым запросом как с веб-сокетом
  wss.handleUpgrade(req, req.socket, Buffer.alloc(0), onSocketConnect);
});

function onSocketConnect(ws) {
  clients.add(ws);

  ws.on('message', function(message) {
    message = message.slice(0, 50); // максимальный размер сообщения 50

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

  ws.on('close', function() {
    clients.delete(ws);
  });
}
```

Вот рабочий пример:

```
22
21
20
19
18
17
```

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.

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.

Предыдущий урок Следующий урок

Поделиться

Карта учебника

Комментарии

- Если вам кажется, что в статье что-то не так - вместо комментария напишите на GitHub.
- Для одной строки кода используйте тег `<code>`, для нескольких строк кода — тег `<pre>`, если больше 10 строк — ссылку на песочницу (plnkr, JSBin, codepen...)
- Если что-то непонятно в статье — пишите, что именно и с какого места.

44 Комментариев Войти ▾

G Присоединиться к обсуждению...

ВОЙТИ С ПОМОЩЬЮ ИЛИ ЧЕРЕЗ DISQUS (?)

Имя

♡ 12 **Поделиться** Лучшие **Новые** Старые

Dedal
2 месяца назад

Отличная статья с хорошей подачей материала. Вот только не могу понять один момент, с какой целью функция возвращает false в примере отправки сообщения из формы на сервер:

```
let socket = new WebSocket("wss://javascript.info/article/websocket/chat/ws");
// отправка сообщения из формы
document.forms.publish.onsubmit = function() {
  let outgoingMessage = this.message.value;
  socket.send(outgoingMessage);
  return false;
};
```

0 0 Ответить • Поделиться ›

cia → Dedal
2 месяца назад

Чтобы предотвратить отправку формы на сервер, это аналогично preventDefault()

1 0 Ответить • Поделиться ›

Dedal → cia
2 месяца назад

Спасибо за ответ, т.е. получается это нужно для того, чтобы на сервер отправились исключительно только данные из переменной outgoingMessage ?!

0 0 Ответить • Поделиться ›

cia → Dedal
2 месяца назад

Не только. При отправке формы происходит переход на указанный в action адрес с post-данными. То есть страница перезагрузится и после этого, вдобавок, браузер будет спрашивать отправить ли повторно форму, если мы попробуем обновить страницу.

1 0 Ответить • Поделиться ›

E **Евгений Сазонов**
7 месяцев назад

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

6 0 Ответить • Поделиться ›

Dave looking for dick 🔞
год назад

Wow... Looks good

13 5 Ответить • Поделиться ›

Drino955 → Dave looking for dick 🔞
9 месяцев назад

да, дик дик

0 0 Ответить • Поделиться ›

T **Talgat Abdukulov**
год назад

А через сколько соединение разрывается при простое?

0 0 Ответить • Поделиться ›

А
2 года назад

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

1 1 Ответить • Поделиться ›

А **Алимусим Агаев** → avel-front
4 месяца назад

вЕРНУЛСЯ ?

0 0 Ответить • Поделиться ›

А **avel-front** → Алимусим Агаев
4 месяца назад

Работаю давно) Но комменты иногда читаю

0 0 Ответить • Поделиться ›

П **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

0 0 Ответить • Поделиться ›

Алексей Обухов → Павел Рамков
год назад

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

0 0 Ответить • Поделиться ›

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 копки 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 • Поделиться ›

foxhable → Николай Мельников
2 года назад

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

0 0 Ответить • Поделиться ›

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 • Поделиться ›

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 отдельный 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).

3 0 Ответить • Поделиться ›

Hrach Haghverdyan
3 года назад

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 • Поделиться ›

Dmitriy
3 года назад

Спасибо за материал.

Ещё бы простенький пример серверной части на PHP.

3 1 Ответить • Поделиться ›

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('закрыть', function() {...})
```

надо "сообщение" и "закрыть" 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 и никакие изменения в коде файла server.js не работали.

10 0 Ответить • Поделиться ›

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}`);
}
```

2 0 Ответить • Поделиться ›

Андреслав Козлов → Adam Itch
3 года назад

Спасибо!

0 0 Ответить • Поделиться ›

Gorr1995
4 года назад

Чат не работает ((

1 0 Ответить • Поделиться ›

Vitaly Sokolov → Gorr1995
4 года назад

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

3 1 Ответить • Поделиться ›

Мария Насонкина → Gorr1995
4 года назад

пишет код 1006 в консоли))

0 0 Ответить • Поделиться ›

P **Роман Василаки**
4 года назад
Будет 4 часть учебника по серверной части (node)?

0 0 Ответить • Поделиться ›

aLex → Роман Василаки
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 • Поделиться ›

A **avel-front** → aLex
2 года назад
Сохранил, спасибо

0 0 Ответить • Поделиться ›

Schulzkafer → aLex
3 года назад
метанит - отличный учебник, рекомендую

3 0 Ответить • Поделиться ›

mimic89 → Роман Василаки
4 года назад
Очень сомнительно.
Здесь есть скринкаст по ноде, но он не самый "новый", насколько я понимаю:
https://learn.javascript.ru...

1 0 Ответить • Поделиться ›

P **Роман Василаки** → mimic89
4 года назад
Хоть что-то, пойду формировать общее представление о инструментах node. Спасибо.

0 0 Ответить • Поделиться ›

G **Gorr1995** → mimic89
4 года назад
Будем откровенны , ты прав. Но он весьма хорош

0 0 Ответить • Поделиться ›

C **Сергей**
4 года назад
Поправьте socket.bufferType на socket.binaryType
источник: https://html.spec.whatwg.or...

1 0 Ответить • Поделиться ›

Шамиль Джакеев → Сергей
3 года назад
Открыл ПР с соответствующими правками.

0 0 Ответить • Поделиться ›

Rail Batyrshin
4 года назад
Thanks

0 0 Ответить • Поделиться ›

J **John Doe**
4 года назад
Многое стало понятно, большое спасибо!

1 1 Ответить • Поделиться ›

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 • Поделиться ›

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 • Поделиться ›

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 • Поделиться ›

VD **Viktar Daniliuk** → Tron_KZ
4 года назад

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

0 1 Ответить • Поделиться ›

T **Tron_KZ** → Viktar 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".

4 6 Ответить • Поделиться ›

Д **Дмитрий Толмачев** → Tron_KZ
2 года назад

умный... хочу все знать...

0 0 Ответить • Поделиться ›

AD **A D** → Tron_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 :)

5 1 Ответить • Поделиться ›

A **Abbos Tajimov**
4 года назад

Спасибо за хорошую статью

8 1 Ответить • Поделиться ›