

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](#) → [Сетевые запросы](#) 30-го ноября 2019

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:

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

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



```
12
13 socket.onclose = function(event) {
14     if (event.wasClean) {
15         alert(`[close] Соединение закрыто чисто, код=${event}`);
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] ${error.message}`);
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`.

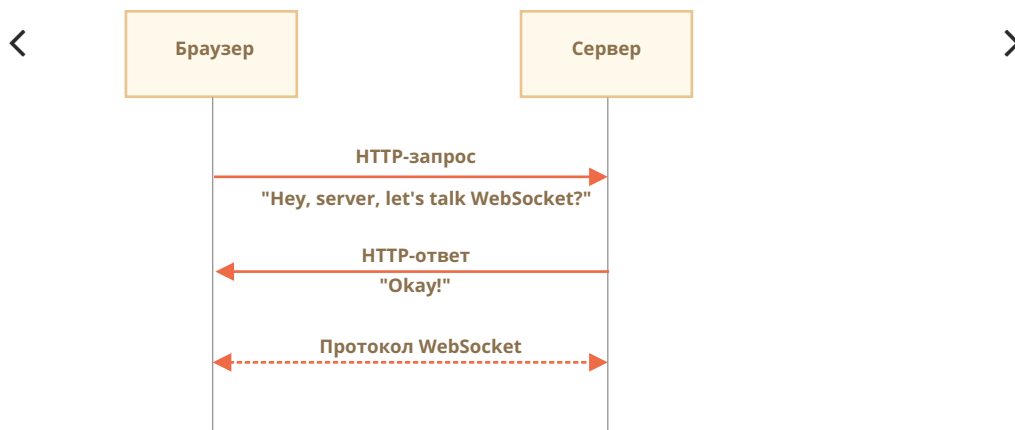
В общем-то, всё, мы уже можем общаться по протоколу WebSocket. Просто, не так ли?

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.



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+1YFgZWcXczP8Q==
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 все 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.

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](#)



- **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.

i 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=
```

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/cha
```

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+1YFgZWcXczP8Q==
7
```

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](#)



```
8 Sec-WebSocket-Version: 13
9 Sec-WebSocket-Extensions: deflate-frame
  Sec-WebSocket-Protocol: soap, wamp
```

Ответ:

```
1 101 Switching Protocols
2 Upgrade: websocket
3 Connection: Upgrade
4 Sec-WebSocket-Accept: hsBlbuDTkk24srzE0TBULzA1C2g=
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.

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-понт 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.bufferType`, по умолчанию оно равно `"blob"`, так что бинарные данные поступают в виде `Blob`-объектов.

`Blob` – это высокоуровневый бинарный объект, он напрямую интегрируется с `<a>`, `` 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.bufferType = "arraybuffer";
2 socket.onmessage = (event) => {
3   // event.data является строкой (если текст) или array
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.

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](#)



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 данн
2 // только если все текущие отсланы
3 setInterval(() => {
4   if (socket.bufferedAmount == 0) {
5     socket.send(moreData());
6   }
7 }, 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:

```
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 }
```

```
6 // event.wasClean === false (нет закрывающего кадра)
};
```

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](#)



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:

```
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/artic
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#messag
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(messageEl
18 }
```

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.

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](#)



- Для каждого принятого веб-сокета – добавить его в набор `clients.add(socket)` и поставить ему обработчик события `message` для приёма сообщений.
- Когда сообщение получено: перебрать клиентов `clients` и отправить его всем.
- 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
8    // здесь мы работаем с каждым запросом как с веб-сокетом
9    wss.handleUpgrade(req, req.socket, Buffer.alloc(0), () => {
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
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:

Send

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



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](#)



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...

© 2007–2020 Илья Кантор | [о проекте](#) | [связаться с нами](#) | [пользовательское соглашение](#) | [политика конфи](#)

