



ПРОДВИНУТАЯ РАБОТА С ВЕБСОКЕТАМИ



АНТОН ВАРНАВСКИЙ



АНТОН ВАРНАВСКИЙ



anton.varnauski@gmail.com



[anton varnavskiy](#)

ПЛАН ЗАНЯТИЯ

1. [Установка соединения в деталях, разбор заголовков](#)
2. [Подписка на события](#)
3. [Отправка бинарных сообщений](#)
4. [Получение бинарных сообщений](#)
5. [Ping / Pong](#)
6. [Отладка вебсокетов](#)
7. [Защищенные вебсокет](#)
8. [COMET](#)
 - HTTP Polling
 - HTTP Long Polling



РАЗБОР ЗАГОЛОВКОВ

ПРИМЕР ЗАПРОСА

Пример запроса от браузера при создании объекта:

```
1 | new WebSocket('ws://server.example.com/chat');
```

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Origin: http://netology.ru
Sec-WebSocket-Key: vggrBKQ/UpD6Tea7luZx4g==
Sec-WebSocket-Version: 13
```

Сервер анализирует полученные заголовки и решает, разрешать или нет веб-сокеты соединение.

ОПИСАНИЯ ЗАГОЛОВКОВ

GET, Host — Стандартные строки HTTP-запроса.

Upgrade, Connection — Указывают, что браузер хочет перейти на websocket.

Origin — Протокол, домен и порт, откуда отправлен запрос.

Sec-WebSocket-Key — Случайный ключ, который генерируется браузером: 16 байт в кодировке Base64.

Sec-WebSocket-Version — Версия протокола. Текущая версия: 13.

Обратите внимание: все заголовки, кроме GET и Host, браузер генерирует сам, без возможности вмешательства JavaScript.

«ПОДДЕЛКА» WEBSOCKET

Создать и отправить подобный запрос с помощью `XMLHttpRequest` невозможно по одной простой причине: указанные выше заголовки запрещены к установке методом `setRequestHeader`.

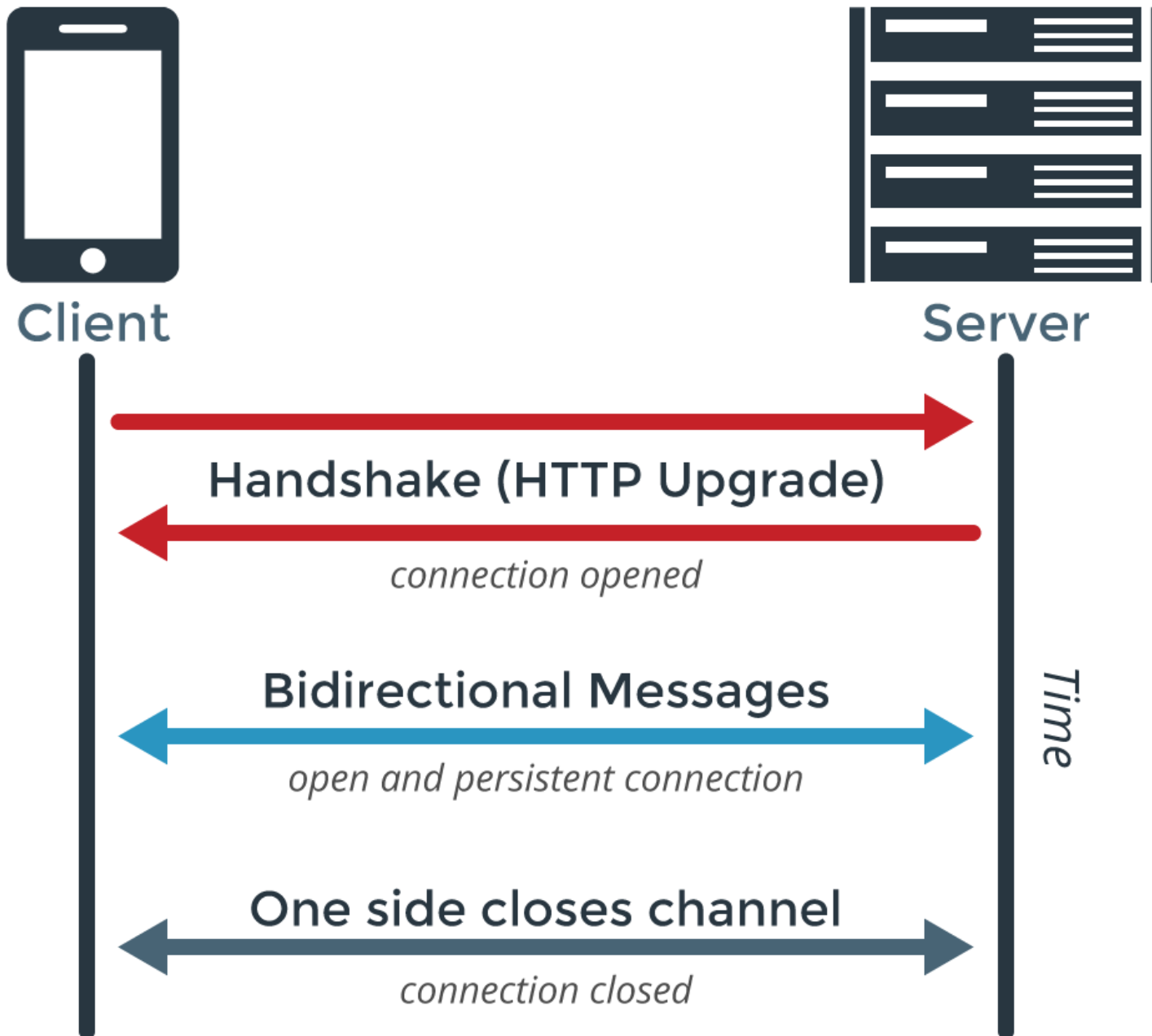
ОДОБРЕНИЕ ВЕБ-СОКЕТА

Если сервер понимает веб-сокет протокол и разрешает открыть соединение, то выдаст ответ следующего вида:

```
HTTP/1.1 101 Switching Protocols  
Upgrade: websocket  
Connection: Upgrade  
Sec-WebSocket-Accept: IMdrdZIIAYPcm0EBFrU81GIzdp8=
```

В полученном ответе значение заголовка `Sec-WebSocket-Accept` представляет собой перекодированный по специальному алгоритму ключ `Sec-WebSocket-Key`. Браузер использует его для проверки того, что ответ предназначенся именно ему.

Затем данные передаются по специальному протоколу, структура которого («фреймы») будет рассмотрена далее. И это уже совсем не протокол HTTP.





ПОДПИСКА НА СОБЫТІЯ

ПОДПИСЫВАЕМСЯ НА СОБЫТИЯ

Подписаться на события вебсокетов можно с помощью `addEventListener`.

Например, таким образом можно подписаться на открытие веб-сокета:

```
1 var connection = new WebSocket('ws://example.com/');
2 connection.addEventListener('open', () => {
3     console.log('Вебсокет-соединение открыто');
4 });
```

Так же, как на открытие, можно подписаться на сообщение или на закрытие.

ИНФОРМАЦИЯ ОТ СЕРВЕРА

В случае подписки на событие у нас появляется еще одно событие с информацией, которую прислал сервер:

```
1 connection.addEventListener('message', event => {  
2   console.log(event.data);  
3 });
```

ЗАКРЫТИЕ СОЕДИНЕНИЯ

Сервер также присылает информацию и при закрытии соединения. Подписаться на него можно следующим образом:

```
1 connection.addEventListener('close', event => {  
2   console.log(event.code);  
3 });
```


В событии закрытия приходит код закрытия, который сообщает о том, по какой причине соединения было закрыто.

КОДЫ ЗАКРЫТИЯ

- 1000 — нормальное закрытие
- 1003 — соединение завершено по причине получения данных, которые не удалось разобрать.

Например, сторона, которая понимает только текстовые данные, может закрыть соединение с кодом 1003, если приняла бинарное сообщение.

В зависимости от кода закрытия мы можем решить, что делать дальше — вывести пользователю сообщение об ошибке (код закрытия — ошибка) или прекратить работу с вебсокетом (нормальное закрытие).



ОТПРАВКА БИНАРНЫХ СООБЩЕНИЙ



ОТПРАВКА ФАЙЛОВ

Через вебсокет мы можем отправлять не только строки или закодированные в строку объекты, а также файлы. Файлы могут быть отправлены, как `Blob` или `ArrayBuffer`.

ОТПРАВЛЯЕМ СОДЕРЖИМОЕ `<canvas>`

Отправка canvas `ImageData` в виде `ArrayBuffer`:

```
1  const image = ctx.getImageData(0, 0, 400, 320);  
2  const binary = Uint8Array.from(image.data);  
3  connection.send(binary.buffer);
```

`ctx` — КОНТЕКСТ НАШЕГО ХОЛСТА.


ОТПРАВКА Blob

И пример с отправкой Blob:

```
1  const fileField = document
2    .querySelector('input[type="file"]');
3  const image = fileField.files[0];
4  connection.send(image);
```

Аналогично может быть отправлен любой файл:

```
1  const fileField = document
2    .querySelector('input[type="file"]');
3  const file = fileField.files[0];
4  connection.send(file);
```



ПОЛУЧЕНИЕ БИНАРНЫХ СООБЩЕНИЙ

ОПРЕДЕЛЯЕМ ФОРМАТ

Бинарные сообщения могут быть получены в `Blob` или `ArrayBufer` форматах.

Для указания формата, в котором мы хотим получать поступающие с сервера бинарные данные, необходимо определить свойство соединения `binaryType`, которое может принимать два значения:

- `blob`, используется по умолчанию.
- `arraybuffer`.

ПОЛУЧАЕМ СООБЩЕНИЕ

```
1 connection.binaryType = 'arraybuffer';  
2 connection.addEventListener('message', event => {  
3     console.log(event.data.byteLength);  
4 });
```

Если при получении бинарных данных свойство `binaryType` не задано, то они будут доступны в формате `Blob`.

Обратите внимание, что при отправке бинарного сообщения мы не задавали свойство соединения `binaryType`. Это нужно только при получении.



PING / PONG

ПРОВЕРКА СВЯЗИ

В протокол веб-сокет встроена проверка связи при помощи управляющих фреймов типа PING и PONG (операционные коды `0x9` и `0xA`).

Та сторона, которая хочет проверить соединение, отправляет фрейм PING с произвольным телом. Его получатель должен в разумное время ответить фреймом PONG с тем же телом.

Этот функционал встроен в браузерную реализацию, так что браузер сам ответит на PING сервера или пошлет PING. Мы из JavaScript не можем влиять на этот процесс.

Иначе говоря, сервер всегда знает, жив ли посетитель, или у него проблема с сетью. И браузер тоже.

ЧИСТОЕ ЗАКРЫТИЕ

При закрытии соединения сторона, желающая это сделать (обе стороны в WebSocket равноправны), отправляет закрывающий фрейм (операционный код `0x8`), в теле которого указывает причину закрытия.

В браузерной реализации эта причина будет содержаться в свойстве `reason` события `onclose`.

Наличие такого фрейма позволяет отличить «чистое закрытие» от обрыва связи.

В браузерной реализации событие `onclose` при чистом закрытии имеет `event.wasClean = true`.



ОТЛАДКА ВЕБСОКЕТОВ



ИНСТРУМЕНТАРИЙ

Для отладки вебсокетов можно использовать **Chrome Developer DevTools**, которые позволяют отслеживать отправляемые и принимаемые сообщения, смотреть и анализировать заголовки, видеть возникающие ошибки.

ПРОСМОТР ЗАГОЛОВКОВ И СООБЩЕНИЙ


Все это доступно во вкладке **Network**. Отфильтровать вебсокеты-соединения можно при помощи фильтра **WS**. После выбора открытого вебсокета-соединения будут доступны 4 вкладки. Наибольший интерес предоставляют первые две — «Заголовки» и «Фреймы». В первой можно увидеть отправляемые заголовки и заголовки полученного ответа. Во второй можно увидеть отправляемые и получаемые сообщения.

The screenshot shows the Chrome DevTools Network tab with the 'WS' filter selected. The 'Headers' sub-tab is active, displaying a table of data for a selected WebSocket frame. The table has columns for 'Name', 'Data', 'Length', and 'Time'. The first row shows 'Привет' with a length of 6 and a time of 11:22:36.097. The second row shows 'Привет' with a length of 6 and a time of 11:22:36.235. The 'Frames' sub-tab is also visible, showing a list of frames with checkboxes for selection. The status bar at the bottom indicates '2 / 69 requests | 0 B / 19.4 K..'. A message at the bottom right says 'Select frame to browse its content.'

Name	Data	Length	Time
<input type="checkbox"/> 1495869737692	Привет	6	11:22:36.097
<input type="checkbox"/> ?encoding=text	Привет	6	11:22:36.235

2 / 69 requests | 0 B / 19.4 K..

Select frame to browse its content.



ЗАЩИЩЕННЫЕ ВЕБСОКЕТЫ

ПРОТОКОЛЫ СОЕДИНЕНИЯ

Соединение можно открывать как `ws://` или как `wss://`. Протокол WS представляет собой WebSocket над HTTP, а протокол WSS, как можно догадаться, над HTTPS.



ПРЕИМУЩЕСТВО WSS

Первое преимущество WSS — бóльшая безопасность. Так же, как в случае с HTTPS.

Второе важное преимущество — бóльшая вероятность соединения.

Вероятность соединения увеличивается по той причине, что HTTPS шифрует трафик от клиента к серверу, а HTTP — нет.



ПРОКСИ И WS

Если между клиентом и сервером есть прокси, то в случае с HTTP все WebSocket-заголовки и данные передаются через него. Прокси имеет к ним доступ, ведь они никак не шифруются, и может расценить происходящее как нарушение протокола HTTP, обрезать заголовки или оборвать передачу.



ПРОКСИ И WSS

А в случае с WSS весь трафик сразу шифруется и через прокси проходит уже в закодированном виде. Поэтому заголовки гарантированно пройдут, и общая вероятность соединения через WSS выше, чем через WS.



COMET



***COMET** – общий термин, описывающий различные техники получения данных по инициативе сервера.*

*Можно сказать, что **AJAX** – это «отправил запрос – получил результат», а **COMET** – это «непрерывный канал, по которому приходят данные».*

ПРИМЕРЫ СОМЕТ-ПРИЛОЖЕНИЙ

- Чат — человек сидит и смотрит, что пишут другие. При этом новые сообщения приходят «сами по себе», он не должен нажимать на кнопку для обновления окна чата.
- Аукцион — человек смотрит на экран и видит, как обновляется текущая ставка за товар.
- Интерфейс редактирования — когда один редактор начинает изменять документ, другие видят информацию об этом. Возможно и совместное редактирование, когда редакторы видят изменения друг друга.

На текущий момент технология СОМЕТ удобно реализуется во всех браузерах.



HTTP POLLING (ЧАСТЫЕ ОПРОСЫ)

Первое, простейшее на первый взгляд, решение, с помощью которого можно непрерывно получать данные с сервера — каждый определенный промежуток времени, например, каждые 5 секунд, отправлять запрос на сервер. При получении запроса с клиента сервер поймет, что клиент онлайн и отправит ему нужные данные.



ПЛЮСЫ И МИНУСЫ

Основной плюс решения — простота реализации на клиенте.

Основные минусы данного подхода:

- Лишний трафик — с каждым запросом приходится отправлять одни и те же заголовки и получать одни и те же заголовки в ответе;
- Задержки между отправкой запроса и ответом. В зависимости от скорости соединения скорость может быть довольно низкой;
- Более сложная реализация на сервере мультиклиентных решений.



HTTP LONG POLLING (ДЛИННЫЕ ОПРОСЫ)

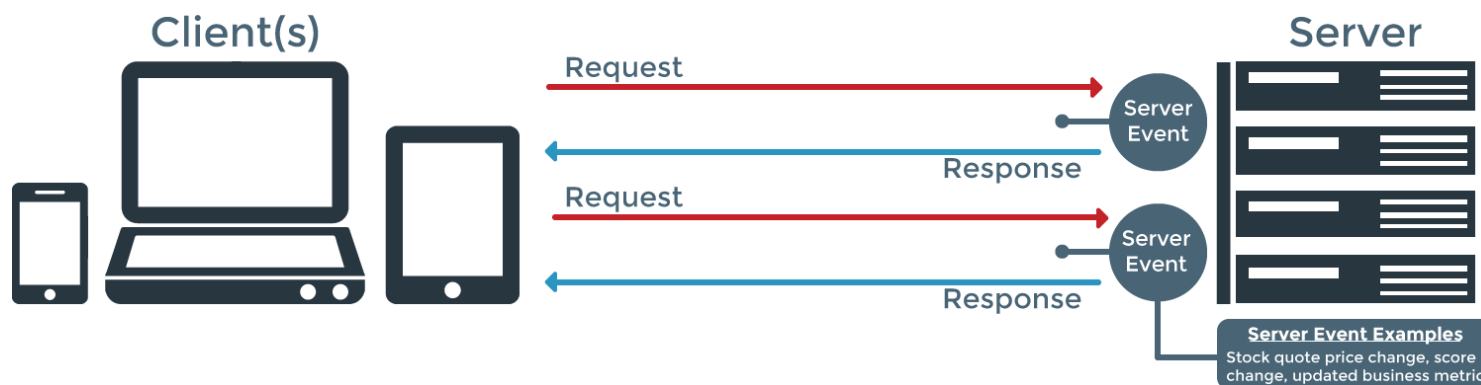
Есть еще другие похожие на вебсокеты техники постоянного общения клиентской части приложения с сервером. Одной из них является **HTTP Long Polling**.



***HTTP Long Polling** — это технология, которая позволяет получать информацию о новых событиях с помощью «длинных запросов». Сервер получает запрос, но отправляет ответ на него не сразу, а лишь тогда, когда произойдет какое-либо событие (например, поступит новое входящее сообщение), либо истечет заданное время ожидания.*

СХЕМА РАБОТЫ

Используя этот подход, Вы можете мгновенно отображать в своем приложении самые важные события.



НЕДОСТАТКИ ТЕХНОЛОГИИ

- Несколько сложнее организовать серверную часть приложения, так как она, помимо стандартной функциональности обработки запросов, должна еще организовать отказоустойчивость при длинных по времени открытых запросах.
- В случае, если клиентская часть приложения открывает запрос, но при этом теряет соединение (например, при использовании ненадежного WiFi-соединения), нужно организовать передпключение к серверу и запрос повторного запроса. Сервер, в свою очередь, должен понять, что предыдущий открытый запрос уже неактуален и ответ в нем не ожидается.
- Некоторые прокси накладывают ограничения для открытых долгое время запросов.
- Сложность поддержки одного соединения многими клиентами одновременно.



ИТОГИ

СОЕДИНЕНИЕ

- Создать XMLHttpRequest-запрос при помощи JavaScript нельзя.
- Сервер отвечает на понятный запрос перекодированным `Sec-WebSocket-Key`, который дает понять браузеру, что ответ лично для него.
- Можно подписаться на события открытия и закрытия соединения, а также на получение сообщения.
- По коду закрытия можно определить, по какой причине оно произошло.

СООБЩЕНИЯ

- Файлы можно отправлять как Blob или ArrayBuffer.
- Бинарные сообщения получаются в аналогичных форматах.
- Свойство `binareType` может принимать два значения – `blob` (по умолчанию) или `arraybuffer`.
- `binareType` нужно только при получении сообщений.

СВЯЗЬ

- Стороны обмениваются фреймами PING и PONG для проверки связи.
- Управление через JavaScript не доступно.
- Для закрытия соединения отправляется закрывающий фрейм с причиной закрытия в теле.
- Причина лежит в `reason` события `onclose`.
- При чистом закрытии `onclose` равен `event.wasClean = true`.


БЕЗОПАСНОСТЬ И COMET

- Протокол WSS безопаснее, чем WS, поскольку все данные шифруются сразу.
- COMET — это технология, при которой данные приходят по инициативе сервера.
- HTTP Polling — запросы отправляются на сервер через определенный промежуток времени. Это оборачивается лишним трафиком и задержкой между запросом и ответом.
- HTTP Long Polling — сервер отправляет ответ на полученный запрос только тогда, когда будет, что ответить. Например, придет новое сообщение.



Задавайте вопросы и напишите отзыв о лекции!

АНТОН ВАРНАВСКИЙ

 anton.varnauski@gmail.com

 [anton_varnavskiy](https://t.me/anton_varnavskiy)