JavaScript 2

Медиавеб-технологии



**На этом уроке**

1. Подробно изучим элемент audio и его применение для воспроизведения медиафайлов и медиапотоков.
2. Рассмотрим интерфейс HTMLMediaElement, его свойства, методы и события.
3. Узнаем о дополнительных полях и отличиях элемента video.
4. Применим getUserMedia API с MediaStream API для захвата и воспроизведения медиапотока с микрофона пользователя.
5. Кратко перечислим, какие ещё медиа API доступны в современных браузерах.

**Оглавление**

[Медиа в интернете](#_1wxdxcktp9q7)

[Элемент audio](#_yz81p19q1507)

[Автовоспроизведение аудио](#_s9im3pouvqh)

[Элементы управления](#_j2orspmzw10j)

[Предварительная загрузка аудио](#_i5eksjj20g38)

[Добавление нескольких источников аудиопотока](#_r6rlrq7l9tm)

[Отображение контента, если audio не поддерживается](#_7hoorxtb5lqy)

[Интерфейс HTMLMediaElement](#_3kjak8r3evum)

[События HTMLMediaElement](#_z122ikqj8eg4)

[Свойства и методы HTMLMediaElement](#_4p576uapcikt)

[Медиабуферизация, поиск и временные интервалы](#_1rixl0aatsl)

[Создание собственного индикатора буферизации](#_kyeklq9d8vxi)

[Пара слов о Played](#_ts2mevqv7qv9)

[Интерфейс MediaStream](#_phnq66o5yfv5)

[События MediaStream](#_ksxaziqioxb0)

[Свойства и методы MediaStream](#_maz4o1em3cpu)

[Использование MediaStream в качестве источника Audio](#_550dy0ql85e0)

[Интерактивный доступ к микрофону](#_kapf6cdmzbre)

[Получение разрешения на доступ к микрофону](#_9hs8ueb0ywki)

[Доступ к необработанным данным микрофона](#_3cn7urs670kp)

[Сохранение данных с микрофона](#_uymrugb7rn9o)

[Использование Permission API для проверки наличия доступа](#_lv10h0ewar05)

[Заключение](#_nvo63fzg21vg)

[Глоссарий](#_6q7pw21hlcon)

[Практическое задание](#_694cl58gipqk)

[Дополнительные материалы](#_44sinio)

[Используемые источники](#_z337ya)

# Медиа в интернете

Последние годы возможности браузеров по отображению, созданию и управлению аудио, видео и другими медиаданными растут всё более быстрыми темпами. Сейчас нам доступно значительное количество различных медиа API, а также HTML-элементов, интерфейсов DOM и других функций, которые позволяют не только выполнять эти задачи, но и использовать мультимедиа в тандеме с другими технологиями. На этом уроке мы рассмотрим медиаэлементы и основные медиа API, которые чаще всего используются разработчиками для добавления и воспроизведения медиафайлов и потоков в современных веб-приложениях.

# Элемент audio

DOM-элемент audio предназначен для добавления аудиоконтента на HTML-страницу. Этот элемент обладает широкими возможностями и позволяет воспроизводить аудиопоток из различных источников, таких как медиафайлы и медиапотоки, например, MediaStream со встроенного в устройство микрофона.

В простейшем случае добавление элемента audio на страницу происходит в скрытом режиме:

| <audio src="file.mp3"></audio> |
| --- |

По умолчанию браузер не показывает никаких элементов управления для этого элемента и не начинает воспроизведение.

**Важно**! *Все медиаэлементы, которые мы рассмотрим на этом уроке, поддерживают политику одинакового источника (same-origin policy). Если источник аудиопотока находится на другом домене, владелец домена должен разрешить к нему доступ путём установки специального заголовка Access-Control-Allow-Origin.*

Атрибут **crossorigin** указывает, следует ли использовать CORS при загрузке мультимедиа-источника. Допустимые значения:

1. **anonymous**: запрос cross-origin, то есть с HTTP-заголовком Origin, выполняется, но параметры доступа не передаются — нет cookie, не используется стандарт X.509 или базовая HTTP-аутентификация.
2. **use-credentials**: запрос cross-origin, то есть с HTTP-заголовком Origin, выполняется с передачей параметров доступа — есть cookie, используется стандарт X.509 или базовая HTTP-аутентификация.

Если этот атрибут не задан, то CORS при загрузке мультимедиа не используется — без отправки HTTP-заголовка Origin. Если задан неправильно, то он обрабатывается так, как если бы использовалось значение **anonymous**. Для получения дополнительной информации смотрите [настройки атрибутов CORS](https://developer.mozilla.org/ru/docs/Web/HTML/Attributes/crossorigin).

## Автовоспроизведение аудио

Воспроизведение аудиопотока начнётся только в том случае, если установлен атрибут **autoplay**:

| <audio src="file.mp3" autoplay></audio> |
| --- |

**Важно!** *Мобильные браузеры не поддерживают автовоспроизведение.  
Воспроизведение в мобильных браузерах может начаться лишь при определённых условиях, например, как реакция на взаимодействие пользователя со страницей (нажатие кнопки). Для получения дополнительной информации смотрите* [*Autoplay guide*](https://developer.mozilla.org/en-US/docs/Web/Media/Autoplay_guide)*.*

| <audio src="file.mp3" autoplay></audio> <button>Начать воспроизведение</button>  <script>  const audio = document.querySelector('audio')  const handleAutoplay = () => {  // Проверим, что браузер не начал автовоспроизведение  if (audio.paused) {  audio.play()  }  }  handleAutoplay() // Это не будет работать  setTimeout(handleAutoplay, 1000) // Это тоже не будет работать  const button = document.querySelector('button')  button.addEventListener('click', handleAutoplay, false) // А это будет </script> |
| --- |

Атрибут **loop,** если он установлен, перезапустит воспроизведение звука сначала. В противном случае воспроизведение останавливается в конце файла:

| <audio src="file.mp3" autoplay loop></audio> |
| --- |

Атрибут **muted** используется для установки в ноль уровня громкости звука:

| <audio src="file.mp3" autoplay loop muted></audio> |
| --- |

Вы можете указать MIME-тип аудиофайла, используя атрибут **type**. Если не установлен, браузер попытается определить его автоматически:

| <audio  autoplay  loop  src="file.mp3"  type="audio/mpeg"  ></audio> |
| --- |

## Элементы управления

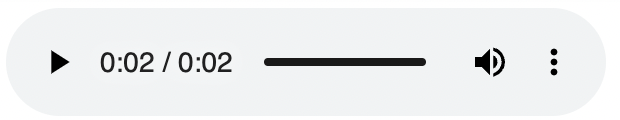
Атрибуты **autoplay, loop** и **muted** позволяют управлять воспроизведением аудио, но пользователь не сможет его остановить, изменить уровень громкости или перемещаться по дорожке.

Чтобы показать встроенные элементы управления, надо добавить атрибут **controls**:

| <audio src="file.mp3" autoplay controls></audio> |
| --- |

Элементы управления могут выглядеть по-разному в зависимости от типа браузера.

Так элементы управления выглядят в Chrome:



А, например, в «Сафари» эти же элементы выглядят иначе:



Chrome позволяет ограниченно стилизовать элементы управления через CSS, используя селекторы псевдоэлементов с вендорными префиксами:

| <style type="text/css">  audio::-webkit-media-controls-panel {  background-color: lightblue;  } </style> <audio src="file.mp3" autoplay controls></audio> |
| --- |

Однако изменить вид стандартных элементов управления кросс-браузерно не получится.  
В этом случае обычно создаётся свой интерфейс и не выводятся стандартные элементы.

Есть возможность настроить набор элементов управления, которые будут выведены посредством атрибута **controllist**.

| <audio  src="file.mp3"  controls  controlslist="nofullscreen nodownload noremoteplayback noplaybackrate" ></audio> |
| --- |

## Предварительная загрузка аудио

Если атрибут **autoplay** не установлен, при загрузке страницы браузеры будут загружать только метаданные аудиопотока, например, для определения его размера и длительности, но не загружать сам аудиопоток.

Мы можем принудительно загрузить аудиопоток, воспользовавшись значением **auto** атрибута **preload,** или вовсе отключить какую-либо загрузку, указав значение **none:**

| <audio src="file.mp3" preload="auto"></audio> |
| --- |

При установленном атрибуте autoplay наличие атрибута preload игнорируется.

## Добавление нескольких источников аудиопотока

Разные версии браузеров используют разные аудиокодеки. Возможно, вы хотите использовать более новый стандарт, который сокращает размер файла, но при этом всё равно поддерживать старые браузеры. Добиться этого можно, указав несколько разных типов источников посредством элементов **source**:

| <audio controls>  <source src="file.opus" type="audio/ogg; codecs=opus" />  <source src="file.ogg" type="audio/ogg; codecs=vorbis" />  <source src="file.mp3" type="audio/mpeg" /> </audio> |
| --- |

## Отображение контента, если audio не поддерживается

Элемент **audio** достаточно хорошо поддерживается современными браузерами, вплоть до IE9, поэтому в настоящее время нет необходимости выводить подсказку, но у нас есть эта опция. Вы просто добавляете элементу содержимое, которое будет выведено только в случае отсутствия его поддержки:

| <audio src="file.mp3">  Ваш браузер не поддерживает <code>audio</code> элемент! </audio> |
| --- |

Итак, ещё раз перечислим атрибуты элемента аудио, рассмотренные выше:

1. **autoplay:** если указан, аудио начнёт воспроизводиться автоматически, как только сможет это сделать, не дожидаясь завершения загрузки всего файла.
2. **controls**: если указан, браузер выведет элементы управления, позволяющие пользователю управлять воспроизведением аудио, в том числе громкостью, перемоткой и паузой.
3. **crossorigin**: указывает, следует ли использовать CORS при загрузке мультимедиа. Допустимые значения: **anonymous**, **use-credentials.** Если этот атрибут не задан, то CORS при загрузке мультимедиа не используется. А если задан неправильно, то он обрабатывается так, как если бы использовалось значение **anonymous**.
4. **loop:** если указан, проигрыватель будет автоматически возвращаться в начало при достижении конца аудио.
5. **muted:** указывает, будет ли звук изначально отключён. Значение по умолчанию — **false**.
6. **preload**: определяет способ загрузки браузером данных медиаресурса. Возможные значения: **none**, **metadata**, **auto**. Значение по умолчанию отличается для каждого браузера. Спецификация рекомендует установить значение **metadata**.

Примечание: атрибут **autoplay** имеет приоритет над **preload**. Если **autoplay** указан, браузер начнёт загрузку аудио для воспроизведения.

1. **src**: URL-адрес медиафайла для загрузки и воспроизведения. Этот атрибут считается необязательным, вместо него можно использовать элемент **source** внутри элемента **audio**.

# Интерфейс HTMLMediaElement

Интерфейс HTMLMediaElement добавляет к HTMLElement свойства и методы, необходимые для поддержки базовых мультимедийных возможностей, общих для аудио и видео. Элементы HTMLVideoElement и HTMLAudioElement наследуют этот интерфейс.

Этот интерфейс также наследует свойства от своих предков HTMLElement, Element, Node и EventTarget.



## События HTMLMediaElement

В плане прослушивания событий медиаэлементы ничем не отличаются от остальных DOM-элементов. Мы также можем использовать **addEventListener()** или присваивать обработчики прямо **on +** ***имя события*** свойствам медиаэлемента. Стоит отметить, что эти события не всплывают, однако при необходимости есть возможность использовать делегирование событий в фазе погружения:

| document.addEventListener('play', (event) => {  console.log(event.target) // audio/video элемент }, true) // Необходимо указать фазу захвата третьим параметром |
| --- |

1. **abort**: срабатывает, когда ресурс не был полностью загружен, однако не как результат ошибки, например, в случае вызова метода **load** в процессе загрузки ресурса.
2. **canplay**: срабатывает, когда браузер может начать воспроизведение, но было загружено недостаточно данных для воспроизведения без прерываний на буферизацию ещё не загруженного содержимого, исходя из скорости соединения.
3. **canplaythrough:** срабатывает, когда браузер может начать воспроизведение, и было загружено достаточно данных для воспроизведения без прерываний на буферизацию ещё не загруженного содержимого, исходя из скорости соединения.
4. **durationchange**: срабатывает в момент обновления атрибута **duration**.
5. **emptied**:срабатывает, когда медиасодержимое удаляется. Например, когда медиаресурс полностью (или частично) загружен, а метод HTMLMediaElement.load() вызван для его перезагрузки.
6. **ended**: срабатывает в момент окончания воспроизведения (<audio> или <video>) при достижении конца файла или по причине недоступности данных.
7. **error**: срабатывает, когда медиаресурс не может быть загружен из-за ошибки.
8. **loadeddata**: срабатывает, когда первый блок (фрейм) загружен.
9. **loadedmetadata**: срабатывает в момент загрузки метаданных.
10. **loadstart**: срабатывает в момент начала загрузки медиаресурса.
11. **pause**: срабатывает, когда запрос на остановку обработан, а воспроизведение остановлено, в основном при вызове метода **pause**.
12. **play**: срабатывает в момент изменения свойства **paused** из значения **true** в **false**, как результат вызова метода **play**, или изменения значения свойства **autoplay.**
13. **playing**: срабатывает, когда воспроизведение готово начаться после остановки на паузу или из-за задержки при получении недостающих данных.
14. **progress**: срабатывает периодически в процессе загрузки браузером данных медиаресурса.
15. **ratechange**: срабатывает при изменении скорости воспроизведения.
16. **seeked**: срабатывает в момент завершения операции поиска, когда пользователь отпускает ползунок на шкале прогресса воспроизведения трека.
17. **seeking**: срабатывает в момент начала операции поиска.
18. **stalled**: срабатывает, когда браузер пытается получить данные медиаресурса, но данные не поступают.
19. **suspend**: срабатывает, когда загрузка данных медиаресурса приостановлена.
20. **timeupdate**: срабатывает при обновлении текущего времени воспроизведения, представленного в атрибуте **currentTime**.
21. **volumechange**: срабатывает при изменении громкости.
22. **waiting**: срабатывает, когда воспроизведение приостановлено из-за временной нехватки данных медиаресурса.

## Свойства и методы HTMLMediaElement

Рассмотрим, какие именно свойства добавляет интерфейс HTMLMediaElement. Часть свойств представляют DOM-атрибуты элемента audio, изученные нами ранее:

1. **autoplay** (Boolean) — отражает значение HTML-атрибута **autoplay**, указывающий, должно ли воспроизведение начинаться автоматически, как только будет доступно достаточно медиафайлов, чтобы сделать это без прерывания.
2. **buffered** (TimeRanges) только для чтения — указатель на объект временных интервалов медиаресурса, который браузер буферизировал в момент обращения к свойству.
3. **controls** (Boolean) — отражает значение HTML-атрибута **controls**, указывающий, должны ли отображаться элементы пользовательского интерфейса управления воспроизведением медиаресурса.
4. **controlsList** (DOMTokenList) только для чтения — отражает значение HTML-атрибута. Свойство **controlslist** позволяет выбирать элементы управления для отображения на элементе мультимедиа, для отображения собственного набора элементов управления. DOMTokenList принимает одно или несколько из трёх возможных значений: **nodownload**, **nofullscreen** (только video) и **noremoteplayback**.
5. **crossOrigin** (DOMString) — отражает значение HTML-атрибута **crossorigin,** указывающий настройку CORS для этого медиаэлемента.
6. **currentSrc** (DOMString) только для чтения — возвращает абсолютный URL-адрес выбранного браузером медиаресурса.
7. **currentTime** (Number) — время текущей позиции воспроизведения в секундах. Установка этого значения переместит точку воспроизведения в новую позицию.
8. **duration** (Number) только для чтения — общая продолжительность медиаресурса в секундах. Если данные о медиаресурсе недоступны, поле получит значение **NaN**. А если медиаресурс неопределённой длины (в случае MediaStream), будет содержать значение **+Infinity**.
9. **ended** (Boolean) только для чтения — возвращает индикатор того, что воспроизведение медиаресурса завершено.
10. **error** (MediaError) только для чтения — содержит указатель на объект ошибки или null, если ошибки не было.
11. **loop** (Boolean) — отражает значение HTML-атрибута **loop**, указывающий, что воспроизведение медиа должно быть зациклено.
12. **muted** (Boolean) — отражает значение HTML-атрибута **muted,** указывающий, выключен ли звук у медиаресурса — **true**, если звук выключен, и **false** — если включён.
13. **networkState** (Number) только для чтения — содержит целочисленную константу (enumeration), отражающую текущее состояние получения медиаресурса по сети, со следующими вариантами значений:

| **Имя константы** | **Значение** | **Описание** |
| --- | --- | --- |
| NETWORK\_EMPTY | 0 | Пока нет данных, при этом поле **readyState** будет содержать значение HTMLMediaElement.HAVE\_NOTHING |
| NETWORK\_IDLE | 1 | Браузер выбрал источник медиаресурса, но загрузка ещё не начиналась |
| NETWORK\_LOADING | 2 | Идёт загрузка данных медиаресурса |
| NETWORK\_NO\_SOURCE | 3 | Браузер не нашёл источников медиаресурса |

| <audio controls preload="auto">  <source src="file.ogg" type="audio/ogg" /> </audio>  <script>  const audio = document.querySelector('audio')   audio.addEventListener('playing', () => {  // if (audio.networkState === 2) {  if (audio.networkState === HTMLMediaElement.NETWORK\_LOADING) {  console.log('Все ещё загружается...')  }  }) </script> |
| --- |

1. **paused** (Boolean) только для чтения — указывает, что воспроизведение медиаресурса остановлено.
2. **playbackRate** (Number) — указывает скорость, с которой воспроизводится медиаресурс.
3. **played** (TimeRanges) только для чтения — указатель на объект временных интервалов медиаресурса, который браузер воспроизвёл (если есть).
4. **preload** (DOMString) — отражает значение HTML-атрибута **preload**, определяющего способ загрузки браузером данных медиаресурса. Возможные значения: **none**, **metadata**, **auto**.
5. **readyState** (Number) только для чтения — содержит целочисленную константу (enumeration), отражающую состояние готовности медиаресурса со следующими вариантами значений:

| **Имя константы** | **Значение** | **Описание** |
| --- | --- | --- |
| HAVE\_NOTHING | 0 | Информация о медиаресурсе недоступна |
| HAVE\_METADATA | 1 | Браузер загрузил достаточно информации о медиаресурсе для инициализации метаданных. Вызов поиска (seeking) больше не вызовет ошибки |
| HAVE\_CURRENT\_DATA | 2 | Доступны данные для текущей позиции воспроизведения, но недостаточно, чтобы воспроизвести больше одного кадра |
| HAVE\_FUTURE\_DATA | 3 | Доступны данные для текущей позиции воспроизведения, а также, по крайней мере, на некоторое время в будущем — как минимум два кадра видео |
| HAVE\_ENOUGH\_DATA | 4 | Загружено достаточно данных для воспроизведения, и скорость загрузки стабильно высокая для воспроизведения медиапотока без прерываний |

| <audio controls preload="auto">  <source src="file.ogg" type="audio/ogg" /> </audio>  <script>  const audio = document.querySelector('audio')   audio.addEventListener('loadeddata', () => {  // if (audio.readyState > 2) {  if (audio.readyState > HTMLMediaElement.HAVE\_CURRENT\_DATA) {  audio.play()  }  }) </script> |
| --- |

1. **seekable** (TimeRanges) только для чтения — указатель на объект временных интервалов (если есть) медиаресурса, которые может выбрать пользователь.
2. **seeking** (Boolean) только для чтения — индикатор того, что медиаресурс находится в состоянии выбора новой позиции воспроизведения.
3. **src** (DOMString) — отражает значение HTML-атрибута **src**, определяющего URL-адрес медиаресурса для использования.
4. **textTracks** (TextTrackЛист) только для чтения — указатель на динамический список объектов TextTrack.
5. **volume** (Number) — значение громкости звука, от 0.0 (звук выключен) до 1.0 (максимальная громкость).

Помимо методов, наследуемых от базовых классов HTMLElement, Element, Node, и EventTarget, HTMLMediaElement поддерживает следующие методы:

1. **HTMLMediaElement.addTextTrack()**: добавляет объект типа TextTrack, например, как трек для субтитров.
2. **HTMLMediaElement.captureStream()**: возвращает MediaStream, захватывает media-поток медиасодержимого.
3. **HTMLMediaElement.canPlayType()**: принимает строку MIME-типа медиаресурса, может также включать параметры кодека. Вызов **canPlayType()** вернёт строку **probably,** если браузер способен воспроизвести этот формат, **maybe,** если недостаточно информации для определения возможности воспроизведения, или пустую строку, если этот тип не поддерживается.
4. **HTMLMediaElement.load()**: перезапускает процесс выбора оптимального источника медиаресурса из вариантов, указанных в элементах **source** или переданного в атрибуте **src** или единственном **source**-элементе.
5. **HTMLMediaElement.pause()**: останавливает воспроизведение медиаресурса.
6. **HTMLMediaElement.play()**: начинает воспроизведение медиаресурса.
7. **HTMLMediaElement.setMediaKeys()**: возвращает **Promise**. Устанавливает [MediaKeys](https://developer.mozilla.org/en-US/docs/Web/API/MediaKeys)-ключи для декодирования медиаресурса в процессе воспроизведения.
8. **HTMLMediaElement.setSinkId()**: устанавливает ID медиаустройства для воспроизведения медиа и возвращает **Promise**. Требует получения разрешения для доступа к соответствующему устройству.

## Медиабуферизация, поиск и временные интервалы

Иногда полезно знать, какая именно часть аудио или видео была загружена или воспроизводится без задержки — например, для отображения в буферном индикаторе загрузки медиапроигрывателя. В этом разделе мы рассмотрим, как создать буфер-индикатор или панель поиска, используя интерфейс TimeRanges и других методов медиа API.

Интерфейс TimeRanges содержит следующие свойства и методы:

1. **length** (Number) только для чтения— количество временных интервалов.
2. **start(index)**: начальное время указанного интервала.
3. **end(index)**: конечное время указанного интервала.

Атрибут HTMLMediaElement.**buffered** содержит временные интервалы частей медиаресурса, которые были загружены — объект TimeRanges. Обычно он непрерывный, но если пользователь перемещает точку воспроизведения в новую позицию, когда медиаэлемент занят буферизацией, тот может содержать пропуски.

Это будет одинаково работать как для <audio>, так и для <video>. Рассмотрим простой пример для audio:

| <audio src="file.mp3" controls></audio>  <script>  const audio = document.querySelector('audio')  const buffered = audio.buffered </script> |
| --- |

Указатель **buffered** хранит ссылку на объект с данными о частях буферизованных участков медиафайла: один или более — сколько успело буферизоваться.

Если в процессе буферизации пользователь не перемещал ползунок по временной шкале трека — обычно существует только один временной интервал, иначе может появиться более одного интервала, как показано на рисунке ниже:



Может получиться, к примеру, два буферизированных интервала по времени — один с 0 по 5 секунду, а второй — с 15 по 19 секунду.

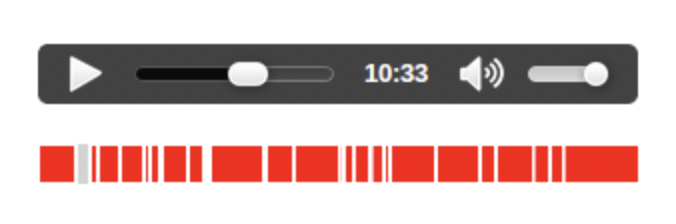
Для этого звукового файла TimeRanges будет иметь следующие доступные свойства:

| <audio src="file.mp3" controls></audio>  <script>  const audio = document.querySelector('audio')  const {buffered} = audio  console.log(buffered.length)) // Вернет 2  console.log(buffered.start(0)) // Вернет 0  console.log(buffered.end(0)) // Вернет 5  console.log(buffered.start(1)) // Вернет 15  console.log(buffered.end(1)) // Вернет 19 </script> |
| --- |

Напишем немного кода, чтобы визуализировать буферные временные интервалы:

| <div>  <audio src="file.mp3" controls></audio> </div> <div>  <canvas width="300" height="20"></canvas> </div>  <script>  const audio = document.querySelector('audio')  const {buffered, duration} = audio  const canvas = document.querySelector('canvas')  const context = canvas.getContext('2d')   context.fillStyle = 'lightgray'  context.fillRect(0, 0, canvas.width, canvas.height)  context.fillStyle = 'red'  context.strokeStyle = 'white'   const coef = canvas.width / duration // Длина одной секунды в пикселях   // отображение TimeRanges  audio.addEventListener('seeked', () => {  for (i = 0; i < buffered.length; i++) {  const startX = buffered.start(i) \* coef  const endX = buffered.end(i) \* coef  const width = endX - startX   context.fillRect(startX, 0, width, canvas.height)  context.rect(startX, 0, width, canvas.height)  context.stroke()  }  }) </script> |
| --- |

Для наглядности лучше использовать аудиофайл большого размера. Нажмите кнопку воспроизведения и передвиньте ползунок на панели прогресса. Вы должны получить результат, похожий на тот, что показан на изображении:



Каждый красный прямоугольник представляет один TimeRange.

## Создание собственного индикатора буферизации

Атрибут **seekable** хранит указатель на объект TimeRanges, представляющий части медиафайла, которые можно воспроизводить без задержки, независимо от того, была ли эта часть загружена или нет. Некоторые части медиа доступны для поиска, но не буферизированы, если на сервере разрешены запросы диапазона байтов. Запросы диапазона байтов позволяют доставить части медиафайла с сервера и поэтому могут быть готовы к воспроизведению почти сразу — таким образом, они доступны для поиска.

| <audio src="file.mp3" controls></audio>  <script>  const audio = document.querySelector('audio')  const {seekable} = audio </script> |
| --- |

Если мы хотим создать собственный проигрыватель медиа, то можем предоставить пользователю обратную связь о том, какая часть медиа готова к воспроизведению. На практике хороший способ сделать это — использовать атрибут **seekable**. Хотя, как мы видели выше, части носителя, доступные для поиска, не обязательно считаются смежными. Однако они часто бывают такими, и мы можем безопасно аппроксимировать эту информацию, чтобы дать пользователю указание, какие части медиа можно воспроизводить напрямую. Найдём эту точку в медиа, используя следующий код:

| const seekableEnd = seekable.end(seekable.length - 1) |
| --- |

**Важно!** *Значение* ***seekableEnd*** *фактически считается конечной точкой последнего временного интервала, доступного для поиска. На практике этого достаточно, поскольку браузер либо разрешает запросы диапазона, либо нет. Если это не так, то* ***audio.seekable*** *будет эквивалентен* ***audio.buffered****, что даст действительное указание на конец доступного для поиска носителя. А если используются* [*запросы диапазона*](https://developer.mozilla.org/ru/docs/Web/HTTP/Status/206)*, это значение, обычно почти мгновенно, становится продолжительностью медиа* ***audio.duration****.*

Однако в собственной реализации медиапроигрывателя желательно отображать, какая часть медиасодержимого была реально загружена. Это то, что обычно отображает встроенный в браузер плеер.

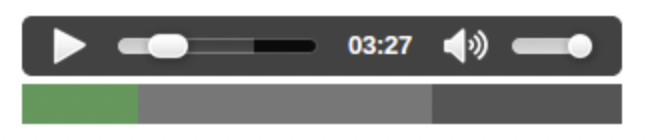
Итак, реализуем это:

| <style type="text/css"> .buffered {  height: 20px;  position: relative;  background: #555;  width: 300px; }  #buffered-amount {  display: block;  height: 100%;  background-color: #777;  width: 0; }  .progress {  margin-top: -20px;  height: 20px;  position: relative;  width: 300px; }  #progress-amount {  display: block;  height: 100%;  background-color: #595;  width: 0; } </style>  <audio src="file.mp3" controls></audio>  <div class="buffered">  <span id="buffered-amount"></span> </div> <div class="progress">  <span id="progress-amount"></span> </div>  <script>  const bufferedAmount = document.quesySelector('#buffered-amount')  const progressAmount = document.quesySelector('#progress-amount')  const audio = document.querySelector('audio')   audio.addEventListener('progress', () => {  const {buffered, currentTime, duration} = audio  if (duration > 0) {  for (var i = 0; i < buffered.length; i++) {  if (buffered.start(buffered.length - 1 - i) < currentTime) {  const endTime = buffered.end(buffered.length - 1 - i)  bufferedAmount.style.width = `${(endTime / duration) \* 100}%`  break  }  }  }  })   audio.addEventListener('timeupdate', () => {  const {currentTime, duration} = audio  if (duration > 0) {  progressAmount.style.width = `${(currentTime / duration) \* 100}%`  }  }) </script> |
| --- |

Обработчик события **progress** срабатывает при загрузке данных. Это событие, на которое надо реагировать, если мы хотим отобразить прогресс загрузки или буферизации.

Обработчик события **timeupdate** срабатывает четыре раза в секунду во время воспроизведения медиа, и именно здесь мы обновляем нашу полосу прогресса.

Результат работы этого кода должен выглядеть примерно так, как на изображении, приведённом ниже, где светло-серая полоса представляет буферизованный прогресс, а зелёная полоса показывает ход воспроизведения:



## Пара слов о Played

Свойство **played** содержит указатель на объект TimeRanges временных интервалов медиаресурса, которые браузер воспроизвёл полностью. Если суммировать все интервалы **played**, то получим долю прослушанного аудио, что может быть полезно для сбора метрик, например:

| <audio src="file.mp3" controls></audio>  <script>  const audio = document.querySelector('audio')  const {played} = audio // объект TimeRanges  const handlePlayed = () => {  let totalPlayedSeconds = 0  for(i = 0; played.length; i++) {  totalPlayedSeconds += played.end(i) - played.start(i)  }  console.log(totalPlayedSeconds)  }   audio.addEventListener('pause', handlePlayed, false)  audio.addEventListener('ended', handlePlayed, false) </script> |
| --- |

# Интерфейс MediaStream

Интерфейс MediaStream представляет поток медиаданных и может использоваться в качестве источника медиасодержимого в HTMLMediaElement. Поток состоит из нескольких треков, таких как видео и аудиотреки. Каждый трек — экземпляр MediaStreamTrack. MediaStream можно получить либо посредством конструктора, либо вызовом MediaDevices.getUserMedia().

Конструктор MediaStream() создаёт и возвращает новый объект MediaStream. Можно создать пустой поток, поток основанный на существующем потоке, или поток содержащий указанный список треков (массив объектов MediaStreamTrack).

## События MediaStream

1. **addtrack**: срабатывает при добавлении нового объекта MediaStreamTrack. Доступно как свойство **onaddtrack**.
2. **removetrack**: срабатывает при удалении объекта MediaStreamTrack. Доступно как свойство **onremovetrack**.

## Свойства и методы MediaStream

Интерфейс MediaStream наследует свойства своего родителя EventTarget.

1. **active** (boolean) только для чтения — возвращает **true**, если MediaStream активен, иначе — **false**.
2. **readyState** (String) только для чтения — может принимать значение **live**, которое указывает, что поток подключён и делает всё возможное для предоставления данных в реальном времени. В этом случае получение данных можно включить или выключить посредством свойства **enabled**. Значение **ended** указывает, что поток завершён и больше не предоставляет новых данных.
3. **id** (DOMString) только для чтения — строка, содержащая 36 символов универсального уникального идентификатора (UUID) потока.

Помимо методов базового класса EventTarget, интерфейс MediaStream добавляет следующие методы:

1. MediaStream.addTrack(track): сохраняет копию объекта track (MediaStreamTrack). Если этот трек добавлялся в MediaStream ранее, он игнорируется.
2. MediaStream.clone(): возвращает точную копию объекта MediaStream. Исключение составляет значение свойства **id**, которое изменится на новое уникальное.
3. MediaStream.getAudioTracks(): возвращает список объектов MediaStreamTrack, хранимых в объекте MediaStream с атрибутом kind, имеющим значение audio. Порядок, в котором возвращаются элементы списка, не определён и может меняться между браузерами, а также между вызовами.
4. MediaStream.getTrackById(trackid): возвращает трек с trackid, соответствующем этому. Если в метод не передано аргументов или трека с этим trackid, возвращает null. А если несколько треков имеют одинаковый trackid, возвращает первый из них.
5. MediaStream.getTracks(): возвращает список всех объектов MediaStreamTrack, сохранённых в объекте MediaStream, вне зависимости от значения их свойства kind. Порядок, в котором возвращаются элементы списка, не определён и может меняться между браузерами, а также между вызовами.
6. MediaStream.getVideoTracks(): возвращает список объектов MediaStreamTrack, хранимых в объекте MediaStream с атрибутом kind, имеющим значение video. Порядок, в котором возвращаются элементы списка, не определён и может меняться между браузерами, а также между вызовами.
7. MediaStream.removeTrack(): удаляет объект MediaStreamTrack. Если этот трек не сохранён в объекте MediaStream, вызов игнорируется.

# Использование MediaStream в качестве источника Audio

Как было сказано выше, HTMLMediaElement, помимо воспроизведения медиафайлов, может принимать MediaStream в качестве источника медиа. В этой части урока мы рассмотрим пример воспроизведения аудиопотока со встроенного в устройство пользователя микрофона.

Многие браузеры сегодня имеют возможность получать доступ к видео и аудио со встроенной в устройство пользователя камеры и микрофона. Однако, в зависимости от браузера, это может быть полностью динамический и встроенный интерфейс или делегированный другому приложению на устройстве пользователя.

Самый простой способ — попросить пользователя предоставить предварительно записанный файл. Сделать это можно, создав простой элемент ввода файла и добавив фильтр, позволяющий выбирать только аудиофайлы, и атрибут **capture**, который указывает, что мы хотим получить его прямо с микрофона.

| <input type="file" accept="audio/\*" capture> |
| --- |

Этот метод работает на всех платформах. На десктопных компьютерах пользователю будет предложено загрузить файл из файловой системы, игнорируя атрибут захвата. В Safari на iOS откроется приложение микрофона, позволяющее записывать звук, а затем отправлять его обратно на веб-страницу. На Android он предоставит пользователю выбор, какое приложение использовать для записи звука, прежде чем отправлять его обратно на веб-страницу.

Как только пользователь закончит запись и вернётся на веб-сайт, вам надо каким-то образом получить данные файла. Вы можете получить к ним доступ в обработчике события onchange элемента ввода, а затем прочитав свойство **files** объекта события.

| <input type="file" accept="audio/\*" capture id="recorder">  <audio id="player" controls></audio>  <script>  const recorder = document.getElementById('recorder')  const player = document.getElementById('player')   recorder.addEventListener('change', (event) => {  const [file] = event.target.files  const url = URL.createObjectURL(file)  // Добавление потока в качестве источника элементу  player.src = url  }) </script> |
| --- |

Получив доступ к файлу, вы можете делать с ним практически всё, что захотите, например:

1. Добавить его элементу audio для воспроизведения, как в примере выше.
2. Загрузить его на устройство пользователя.
3. Загрузить его на сервер, используя XMLHttpRequest / fetch / FormData.
4. Передать его через Web Audio API и применить к нему фильтры.

И хотя использование элемента ввода для получения доступа к медиаданным считается повсеместным, это наименее привлекательный вариант. В действительности мы хотим получить прямой доступ к микрофону прямо на странице.

## Интерактивный доступ к микрофону

Современные браузеры могут получить прямой доступ к микрофону. Это предоставляет нам возможность интеграции с веб-страницей, при которой пользователь никогда не покинет браузер.

### Получение разрешения на доступ к микрофону

Мы можем получить прямой доступ к микрофону посредством метода **getUserMedia**. Метод **getUserMedia** выведет пользователю запрос на получение доступа к встроенным в его устройство микрофонам и камерам.

Если пользователь разрешит доступ, API вернёт MediaStream, который будет содержать данные либо с камеры, либо с микрофона, и затем мы можем:

* прикрепить его к элементу **audio;**
* присоединить его к Web Audio AudioContext;
* сохранить его посредством MediaRecorder API.

Чтобы получить данные с микрофона, просто устанавливаем **audio: true** в объекте ограничений, который передаётся в API **getUserMedia**.

| <audio id="player" controls></audio>  <script>  const player = document.getElementById('player')   const handleSuccess = (stream) => {  if (window.URL) {  player.srcObject = stream  } else {  player.src = stream  }  }   navigator.mediaDevices.getUserMedia({audio: true, video: false})  .then(handleSuccess) </script> |
| --- |

Если нас интересует конкретный микрофон, надо сначала перечислить доступные микрофоны. Затем мы можем передать **deviceId** выбранного микрофона в вызов **getUserMedia**, например, для первого из доступных микрофонов:

| navigator.mediaDevices.enumerateDevices().then((devices) => {  const [firstMicrophone] = devices.filter(({kind}) => kind === 'audioinput')  navigator.mediaDevices.getUserMedia({  audio: {  deviceId: firstMicrophone.deviceId  }  }).then(handleSuccess) }) |
| --- |

В этом нет большой пользы, ведь всё, что делает **handleSuccess** — берёт аудиоданные и воспроизводит их.

### Доступ к необработанным данным микрофона

Чтобы получить доступ к необработанным данным микрофона, мы должны взять оригинальный поток, созданный **getUserMedia()**, и передать его в Web Audio API для обработки. Web Audio API — это API, принимающий источники ввода, такие как аудиопоток, и соединяющий эти источники с узлами, которые могут обрабатывать аудиоданные — регулировать, усилять, накладывать фильтры и прочее — и в конечном счёте вывести их на устройство вывода (динамики или наушники), чтобы пользователь мог их услышать.

Один из узлов, которые мы можем подключить, — **ScriptProcessorNode**. Этот узел будет генерировать событие **onaudioprocess** каждый раз при заполнении аудиобуфера, и нам надо его обработать. На этом этапе мы можем сохранить данные в собственном буфере для дальнейшего использования.

| const handleSuccess = (stream) => {  const context = new AudioContext()  const source = context.createMediaStreamSource(stream)  const processor = context.createScriptProcessor(1024, 1, 1)   source.connect(processor)  processor.connect(context.destination)   processor.onaudioprocess = (event) {  // Сделать что-нибудь с данными, например, конвертировать в WAV  console.log(event.inputBuffer)  } }  navigator.mediaDevices.getUserMedia({audio: true, video: false})  .then(handleSuccess) |
| --- |

Данные, которые хранятся в буфере, считаются необработанными «сырыми» данными микрофона. И у нас есть несколько вариантов того, что можно делать с этими данными:

1. Загрузить на сервер.
2. Хранить локально.
3. Преобразовать в специальный формат файла, например, WAV, и сохранить на сервер или локально.

### Сохранение данных с микрофона

Самый простой способ сохранить данные с микрофона — использовать MediaRecorder API.

| <a id="download">Скачать</a>  <button id="stop">Стоп</button>  <script>  const downloadLink = document.getElementById('download')  const stopButton = document.getElementById('stop')   const handleSuccess = (stream) => {  const options = {mimeType: 'audio/webm'}  const recordedChunks = []  const mediaRecorder = new MediaRecorder(stream, options)   mediaRecorder.addEventListener('dataavailable', (event) => {  if (event.data.size > 0) {  recordedChunks.push(event.data)  }  })   mediaRecorder.addEventListener('stop', () => {  downloadLink.href = URL.createObjectURL(new Blob(recordedChunks))  downloadLink.download = 'acetest.wav'  })   stopButton.addEventListener('click', () => {  mediaRecorder.stop()  })   mediaRecorder.start()  }   navigator.mediaDevices.getUserMedia({audio: true, video: false})  .then(handleSuccess) </script> |
| --- |

Код выше сохраняет данные в массив, который затем может быть конвертирован в Blob-объект и отправлен на веб-сервер или сохранён прямо в хранилище на устройстве пользователя.

Если пользователь ранее не предоставлял вашему сайту доступ к микрофону, то в момент вызова getUserMedia браузер спросит пользователя разрешение на использование микрофона.

**Важно**! *Будьте готовы к тому, что часть пользователей сайта откажут в запросе доступа к микрофону.*

### Использование Permission API для проверки наличия доступа

API getUserMedia не сообщает нам, есть ли у нас уже доступ к микрофону. В некоторых браузерах это можно решить через API разрешения. Объект **navigator.permission** позволяет запрашивать состояние доступа к определённым API без повторного запроса.

Чтобы узнать, есть ли у вас доступ к микрофону пользователя, надо передать {name: 'microphone'} в метод запроса, и он вернёт один из вариантов:

* **granted**: пользователь ранее уже предоставил доступ к микрофону;
* **prompt**: пользователь не предоставил доступ, и он будет запрошен при вызове getUserMedia;
* **denied**: система или пользователь явно заблокировали доступ, и мы не сможем его получить.

Таким образом, мы можем быстро проверить, надо ли нам изменить пользовательский интерфейс, чтобы он соответствовал действиям, которые должен предпринять пользователь.

| navigator.permissions.query({name:'microphone'}).then((result) => {  switch(result.state) {  case 'granted': {  return console.log('Пользователь уже разрешил доступ ранее')  }  case 'prompt': {  return console.log('Разрешение ещё не запрашивалось')  }  case: 'denied': {  return console.log('Пользователь запретил доступ')  }  } }) |
| --- |

# Заключение

Мы рассмотрели основные интерфейсы и элементы для добавления медиаконтента на страницу. Однако возможности современных браузеров этим не ограничиваются. Если вам интересно узнать о других API, рассмотрение которых выходит за рамки этого курса, обратите внимание на ссылки в разделе [«Дополнительные материалы»](#_44sinio).

# Глоссарий

**Cross-Origin Resource Sharing** (CORS) — механизм, использующий дополнительные HTTP-заголовки, чтобы дать возможность агенту пользователя получать разрешения на доступ к выбранным ресурсам с сервера на источнике (домене), отличном от того, что сайт использует в этот момент. Считается, что агент пользователя делает запрос с другого источника (cross-origin HTTP request), если источник текущего документа отличается от запрашиваемого ресурса доменом, протоколом или портом.

**Перечисление** (enumeration, или enum) состоит из набора именованных целочисленных констант. Объявление типа перечисления задаёт имя необязательного тега перечисления. Оно определяет набор именованных целочисленных идентификаторов, которые называются набором перечисления, константами перечислителя, перечислителями или элементами. Переменная с типом перечисления хранит одно из значений набора перечисления, определённого этим типом.

# Практическое задание

Разработайте видеопроигрыватель с кастомным интерфейсом, который включает:

1. Область просмотра видео — прямоугольная область, в которой проигрывается ролик.
2. Панель элементов управления воспроизведением с кнопками «Вперёд», «Назад», «Пауза/Воспроизведение».
3. Временная шкала воспроизведения с индикацией текущей позиции воспроизведения и продолжительности видео — время текущей позиции или общая продолжительность.
4. Индикатор загрузки и буферизации видео.

**Задания повышенной сложности**:

1. Клик на временной шкале (или перетаскивание ползунка) должно менять текущую позицию воспроизведения.
2. Кнопка просмотра в полноэкранном режиме.

# Дополнительные материалы

1. Статья [How to Style an Audio Element](https://blog.shahednasser.com/how-to-style-an-audio-element/).
2. Статья [Let’s Create a Custom Audio Player](https://css-tricks.com/lets-create-a-custom-audio-player/).
3. Статья [Same-origin policy](https://developer.mozilla.org/ru/docs/Web/Security/Same-origin_policy).
4. Статья [Cross-Origin Resource Sharing](https://developer.mozilla.org/ru/docs/Web/HTTP/CORS).
5. Документация [Access-Control-Allow-Origin](https://developer.mozilla.org/ru/docs/Web/HTTP/Headers/Access-Control-Allow-Origin).
6. Статья [Basic concepts behind Web Audio API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Basic_concepts_behind_Web_Audio_API).
7. Статья [Using Web Audio API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Using_Web_Audio_API).
8. Статья [Visualizations with Web Audio API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Visualizations_with_Web_Audio_API).
9. Статья [Web audio spatialization basics](https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Web_audio_spatialization_basics).
10. Документация по [SpeechRecognition API](https://developer.mozilla.org/en-US/docs/Web/API/SpeechRecognition).
11. Статья [What is WebRTC](https://blog.viibe.co/what-is-webrtc/).
12. Документация по [WebRTC](https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API).
13. Статья [WebRTC Guides](https://webrtc.org/getting-started/overview).

# Используемые источники

1. Документация по [audio](https://developer.mozilla.org/ru/docs/Web/HTML/Element/audio).
2. Статья [Recording Audio from the User](https://developers.google.com/web/fundamentals/media/recording-audio?hl=ja).
3. Статья [«Медиабуферизация, поиск и временные диапазоны»](https://developer.mozilla.org/ru/docs/Web/Guide/Audio_and_video_delivery/buffering_seeking_time_ranges).
4. Документация по [HTMLMediaElement](https://developer.mozilla.org/ru/docs/Web/API/HTMLMediaElement).
5. Статья [«Видео и аудиоконтент»](https://developer.mozilla.org/ru/docs/Learn/HTML/Multimedia_and_embedding/Video_and_audio_content).
6. Интерфейс [MediaStream](https://developer.mozilla.org/ru/docs/Web/API/MediaStream).