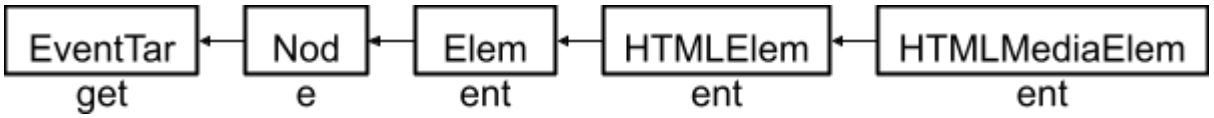
Работа с медиафайлами

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 данный 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 <on + *имя события*> свойствам медиаэлемента. Стоит отметить, что эти события не всплывают, однако при необходимости можно использовать делегирование событий в фазе погружения:

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

События HTMLMediaElement срабатывают в следующих случаях:

1. **abort —** когда ресурс не был полностью загружен, однако не как результат ошибки. Например, при вызове метода **load** в процессе загрузки ресурса.
2. **canplay —** 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 —** 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 —** в момент обновления атрибута **duration**.

5. **emptied** — когда медиасодержимое удаляется. Например, когда медиаресурс полностью (или частично) загружен, а метод HTMLMediaElement.load() вызван для его перезагрузки.
6. **ended** — в момент окончания воспроизведения (<audio> или <video>) при достижении конца файла или по причине недоступности данных.
7. **error** — когда медиаресурс не может быть загружен из-за ошибки.
8. **loadeddata** — когда первый блок (фрейм) загружен.
9. **loadedmetadata** — в момент загрузки метаданных.
10. **loadstart** — в момент начала загрузки медиаресурса.
11. **pause** — когда запрос на остановку обработан, а воспроизведение остановлено. Как правило, при вызове метода pause.
12. **play** — в момент изменения свойства paused из значения true в false, как результат вызова метода play или изменения значения свойства autoplay.
13. **playing** — 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** — при изменении скорости воспроизведения.
16. **seeked** — в момент завершения операции поиска, когда пользователь отпускает ползунок на шкале прогресса воспроизведения трека.
17. **seeking** — в момент начала операции поиска.
18. **stalled** — когда браузер пытается получить данные медиаресурса, но данные не поступают.
19. **suspend** — когда загрузка данных медиаресурса приостановлена.
20. **timeupdate** — при обновлении текущего времени воспроизведения, представленного в атрибуте currentTime.
21. **volumechange** — при изменении громкости.
22. **waiting** — 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, указывающего, 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, указывающего, должны ли отображаться элементы пользовательского интерфейса управления воспроизведением медиаресурса.

4. **controlsList (DOMTokenList)**, только для чтения — отражает значение HTML-атрибута. Свойство **controlslist** позволяет выбирать, какие элементы управления будут отображаться в интерфейсе медиапроигрывателя. DOMTokenList принимает одно или несколько из трёх возможных значений: **nodownload**, **nofullscreen** (только video) и **noremoteplayback**.

5. **crossOrigin** (DOMString) — отражает значение HTML-атрибута crossorigin, указываюего 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**, указывающего, что воспроизведение медиа должно быть зациклено.

12. **muted (Boolean) —** отражает значение HTML-атрибута **muted,** указывающего, выключен ли звук у медиаресурса **(true**, если выключен, **false** — включён**).**

13. **networkState (Number)**, только для чтения — содержит целочисленную константу (enumeration), которая отражает текущее состояние получения медиаресурса по сети и может принимать следующие значения:

Имя константы	**Значение**	**Описание**
NETWORK_EMPTY	0	Пока нет данных. При этом поле **readyState** содержит 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>
```

14. **paused** (Boolean), только для чтения — указывает, что воспроизведение медиаресурса остановлено.

15. **playbackRate** (Number) — указывает скорость, с которой воспроизводится медиаресурс.

16. **played** (TimeRanges) только для чтения — указатель на объект временных интервалов медиаресурса, который браузер воспроизвёл (если есть).

17. **preload (DOMString)** — отражает значение HTML-атрибута **preload**, определяющего способ загрузки браузером данных медиаресурса. Возможные значения: **none**, **metadata, auto.**

18. **readyState** (Number), только для чтения — содержит целочисленную константу (enumeration), отражающую состояние готовности медиаресурса. Может принимать следующие значения:

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 для как минимум двух кадров видео
HAVE_ENOUGH_DATA	4	Загружено достаточно данных для воспроизведения, и скорость загрузки стабильно высока 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>
```

19. **seekable** (TimeRanges), только для чтения — указатель на объект временных интервалов (если есть) медиаресурса, которые может выбрать пользователь.

20. **seeking** (Boolean), только для чтения — индикатор того, что медиаресурс находится в состоянии выбора новой позиции воспроизведения.

21. **src** (DOMString) — отражает значение HTML-атрибута src, определяющего URL-адрес медиаресурса для использования.

22. **textTracks** (TextTrackList), только для чтения — указатель на динамический список объектов TextTrack.
23. **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 данный 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 для декодирования медиаресурса в процессе воспроизведения.
8. **HTMLMediaElement.setSinkId()** — устанавливает ID медиаустройства для воспроизведения медиа и возвращает **Promise**. Требует получения разрешения на доступ 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 с помощью интерфейса 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 непрерывен, но если пользователь перемещал точку воспроизведения в новую позицию, пока медиаэлемент был занят буферизацией, могут появиться пропуски.

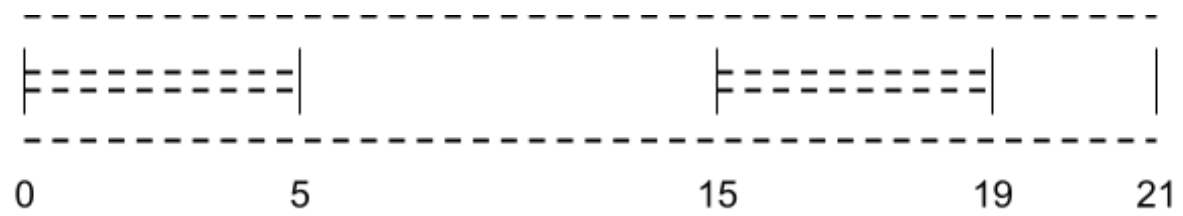
Это будет одинаково работать для <audio> и <video>. Рассмотрим простой пример для audio:

```
<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, в противном случае может появиться более одного интервала, как показано на рисунке ниже:



Может получиться, к примеру, два буферизированных интервала по времени: один с нулевой по пятую секунду, а второй — с пятнадцатой по девятнадцатую.

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 примерно следующего вида:

Каждый красный прямоугольник представляет один TimeRange.

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. Получить MediaStream можно либо посредством конструктора, либо вызовом MediaDevices.getUserMedia().

Конструктор MediaStream() создаёт и возвращает новый объект MediaStream. Можно создать поток на основе существующего, пустой поток или поток с указанным списком треков (массив объектов MediaStreamTrack).

События MediaStream: когда срабатывают

1. **addtrac —** при добавлении нового объекта MediaStreamTrack. Доступно как свойство **onaddtrack**.

2. **removetrack** — 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 могут получать доступ к видео и аудио со встроенных в устройство пользователя камер 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 так: создать простой элемент ввода файла, добавить фильтр для выбора только аудиофайлов, а также добавить атрибут **capture**, который укажет, что мы хотим получить файл 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. Android 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, надо будет 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 элементу потока в качестве источника
```

```
    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 доступа к медиаданным считается повсеместным, это наименее привлекательный вариант. В действительности мы хотим получить доступ к микрофону прямо на странице.

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».