нетология
университет интернет-профессий

СПОСОБЫ ПОИСКА НУЖНОГО HTML-ЭЛЕМЕНТА

АРТЕМ ШАШКОВ / DOCDOC.RU

АРТЕМ ШАШКОВ

Front End Developer DocDoc.ru

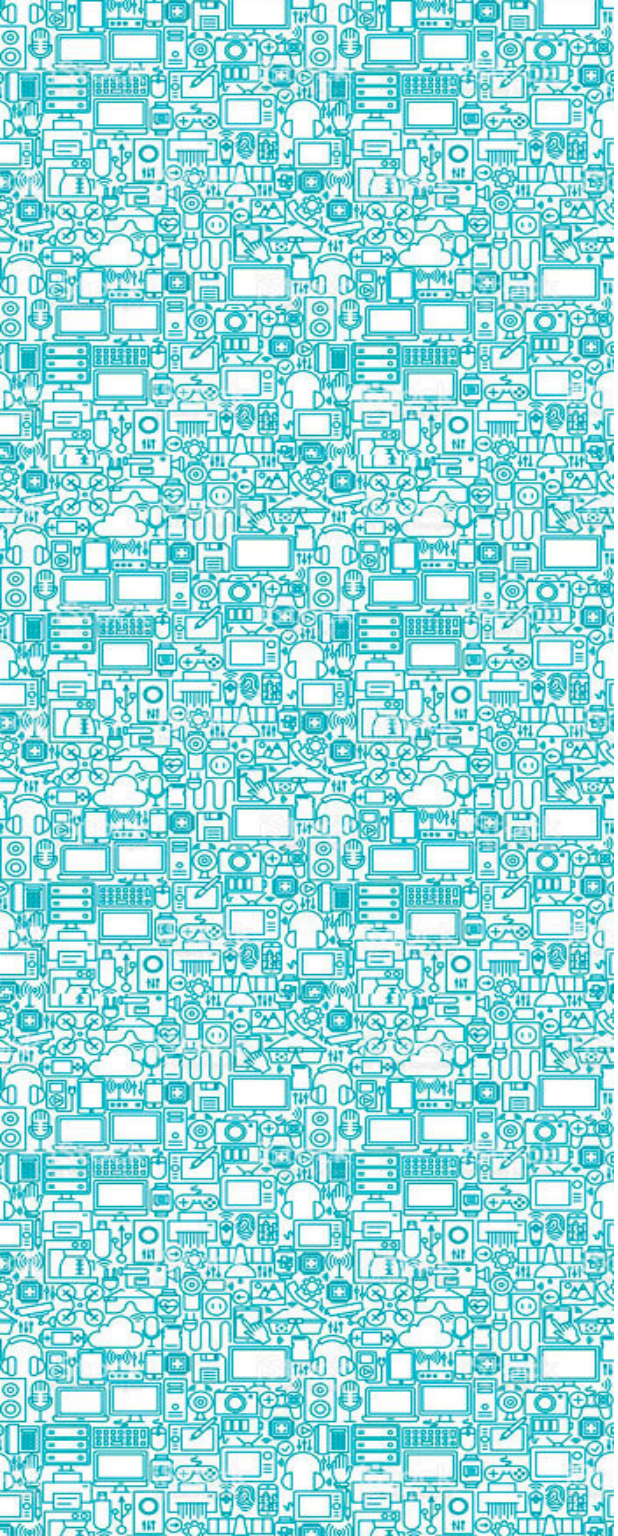
glomix@inbox.ru

Артем Шашков

ПЛАН ЗАНЯТИЯ

1. Введение в BOM
2. Поиск элементов DOM
   - `getElementsByTagName()`
   - `HTMLCollection`
   - `getElementsByClassName()`
   - Работаем с классами через `className` и `classList`
3. Аудио и видео
   - Тег `<audio>`
   - Тег `<video>`

ВВЕДЕНИЕ В BOM



«

BOM *— объектная модель браузера, то есть, все те объекты, с помощью которых можно взаимодействовать непосредственно с браузером.*

DOM? BOM?

В браузере для взаимодействия с HTML-элементами страницы реализована модель *DOM* с главным объектом `document`.

Но есть другая модель — **BOM**. Примером определения этой модели могут служить различные всплывающие окна в браузере.

ДИАЛОГ С ПОЛЬЗОВАТЕЛЕМ

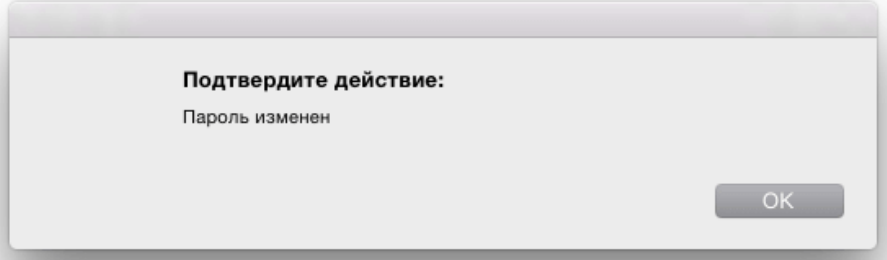
И в качестве первых объектов BOM мы познакомимся с функциями для создания всевозможных модальных окон. Это функции `alert`, `confirm` и `prompt`.

`alert`

`alert` используют для отображения всевозможных уведомлений для пользователя.

```
alert('Пароль изменен');
```

КАК ВЫГЛЯДИТ `alert`



`confirm`

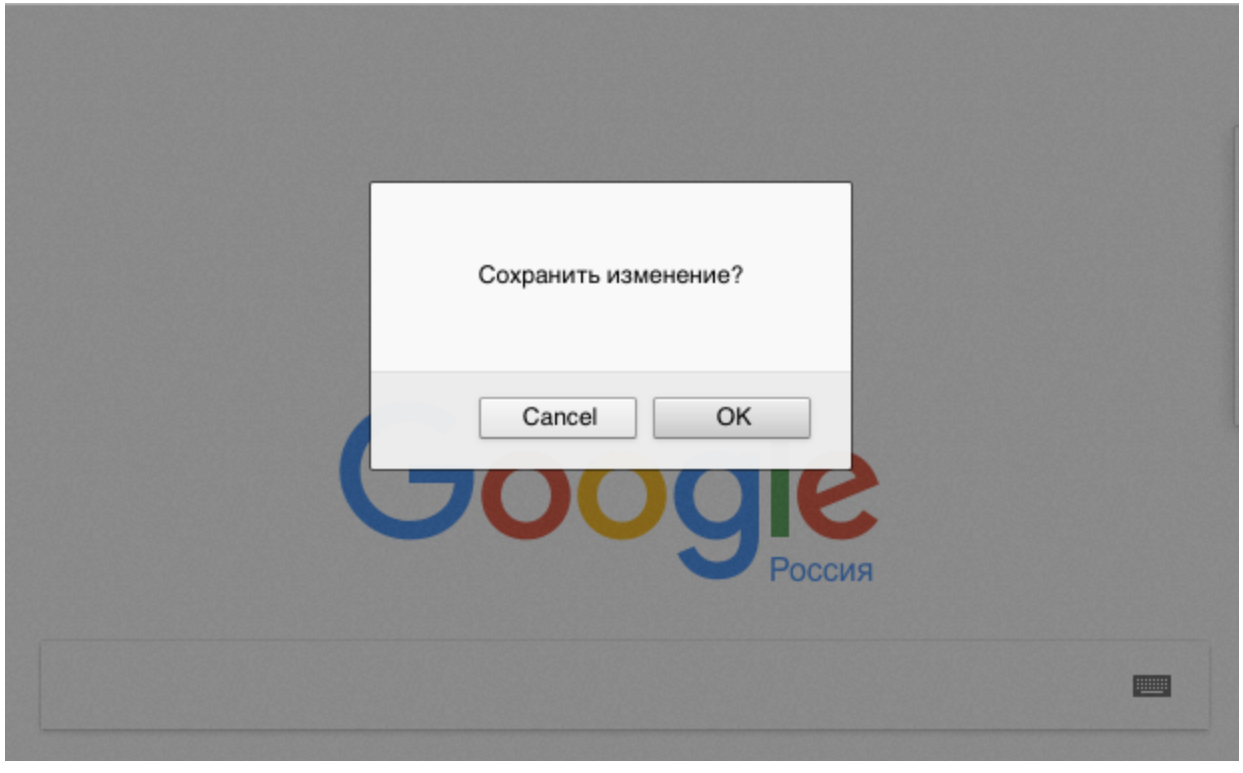
`confirm` служит для получения обратной связи от пользователя.

Во всплывающем окне отображается первый аргумент, с которым была вызвана функция `confirm`.

Возвращаемое значение равно `true`, если пользователь нажал на кнопку `Ok`, и `false` в противном случае.

```
const answer = confirm('Сохранить изменение?');
```

КАК ВЫГЛЯДИТ `confirm`



`prompt`

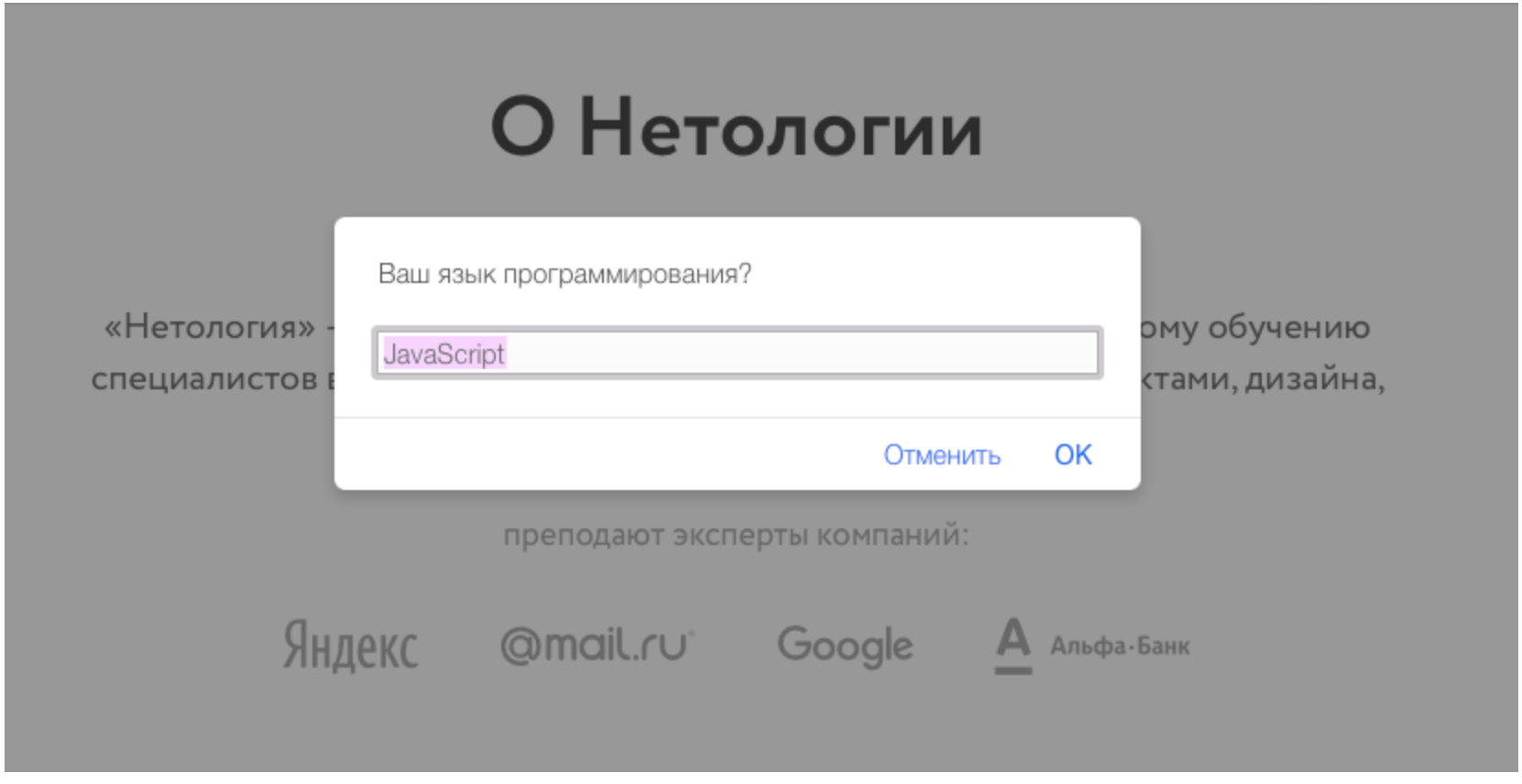
`prompt` используется для того, чтобы пользователь ввел какой-то текст или число.

Первый аргумент функции `prompt` — это вступительный текст в модальном окне, а второй — значение по умолчанию в поле ввода.

Возвращающее значение равно либо вводу пользователя, если он нажал `Ok`, либо `null` в случае отмены.

```
const value = prompt('Ваш язык программирования?', 'JavaScript');
```

КАК ВЫГЛЯДИТ `prompt`



ПРЕДОСТЕРЕЖЕНИЕ

Функции отображения модальных окон практически не используют в современных веб-приложениях. Во-первых, потому, что они блокируют собой работу всего приложения, а во-вторых, потому, что внешний вид этих модальных окон нельзя стилизовать.

ПОИСК ЭЛЕМЕНТОВ DOM

РАСШИРЯЕМ АРСЕНАЛ

На прошлом занятии мы познакомились с самым простым способом поиска объекта-узла по его атрибуту `id` — `document.getElementById()`.

На этом занятии продолжим углубляться в тему поиска и познакомимся с поиском по имени тегов и по классу.

`getElementsByTagName()`

Этот метод используется для получения ссылок на все объекты-узлы, название тега которых равно первому аргументу.

Например, вспомним пример с кнопками, но теперь используем `getElementsByTagName()`.

ПРИМЕР С КНОПКАМИ

```
<button>Кнопка 1</button>
<button>Кнопка 2</button>
<button>Кнопка 3</button>
<script>
  const btns = document.getElementsByTagName('button');

  function logElement() {
    console.log(this);
  }

  for (let i = 0; i < btns.length; i++) {
    btns[i].onclick = logElement;
  }
</script>
```

РАЗБЕРЕМ ПРИМЕР

Объект `btns` — это коллекция найденных объектов узлов. У этого объекта определены свойства `length` и свойства под индексами от `0` до `length-1`, поэтому мы можем перебрать элементы этого объекта через обычный цикл `for`.

РЕГИСТРОНЕЗАВИСИМЫЙ МЕТОД

Метод `getElementsByTagName()` является регистронезависимым.

Это значит, что `getElementsByTagName('button')` и `getElementsByTagName('BUTTON')` вернут одинаковые коллекции.

СОКРАЩАЕМ ОБЛАСТЬ ПОИСКА

Сейчас мы нашли все кнопки в документе, потому что вызвали его как метод объекта `document`. Но `getElementsByTagName` можно вызывать у любого объекта-узла, тогда поиск будет совершаться только по его потомкам.

ИЩЕМ В РАЗНЫХ *РОДИТЕЛЯХ*

```
<div id="div1">
  <button>Кнопка 1</button>
</div>
<div id="div2">
  <button>Кнопка 2</button>
  <button>Кнопка 3</button>
</div>
<script>
  const div1 = document.getElementById('div1');
  const div2 = document.getElementById('div2');
  const btns1 = div1.getElementsByTagName('button');
  const btns2 = div2.getElementsByTagName('button');
  console.log(btns1.length); // 1
  console.log(btns2.length); // 2
</script>
```

HTMLCOLLECTION

Как было понятно из названия метода, `getElements ByTagName()` возвращает не один элемент, а некий список элементов. Причем это не массив, а специальный объект, подобный массиву (похож на `arguments`), прототипом которого является `HTMLCollection`.

ПРОВЕРЯЕМ НА ПУСТОТУ

У такого объекта определено свойство `length`. С помощью него можно определить, нашлось ли вообще что-то, или найденная коллекция пустая (`length` равно `0`).

```
const btns = document.getElementsByTagName('button');
if (btns.length) {
  console.log('Найдена хотя бы одна кнопка');
} else {
  console.log('Нет ни одного тега <button>');
}
```

ИТЕРИРУЕМ КОЛЛЕКЦИИ

Как итерироваться по этой коллекции?

Можно использовать цикл `for`, как мы это делали выше.

Кроме этого, `HTMLCollection` реализует протокол для итерации (определен метод `[Symbol.iterator]` возвращающий итератор). Значит, коллекцию `HTMLCollection` можно перебирать с помощью `for-of` цикла.

```
const btns = document.getElementsByTagName('button');
for (const btn of btns) {
  console.log(btn);
}
```

ИЗБЕГАЙТЕ `for-in`

Использовать цикл `for-in` не следует, потому что в списке перечисляемых свойств есть и само свойство `length`, а не только индексы.

ПРИВОДИМ `HTMLCollection` К МАССИВУ

Как и в случае с `arguments`, в объекте `HTMLCollection` нет привычных методов для массива — `forEach()`, `map()`, `reduce()` и т.д. Чтобы использовать их, необходимо сначала привести `HTMLCollection` к массиву, например, с помощью `Array.from()`:

```
1  const btns = document.getElementsByTagName('button');
2  const buttonsArray = Array.from(btns);
3
4  buttonsArray.forEach(() => { /* */ })
5  buttonsArray.map(() => { /* */ })
6  buttonsArray.reduce(() => { /* */ })
```

`getElementsByClassName()`

Метод `getElementsByClassName` во всем похож на `getElementsByTagName` кроме того, что поиск происходит по имени класса, а не по имени тега.

ИЩЕМ ПО ИМЕНИ КЛАССА

```
<div class="selected">Только selected</div>
<div class="selected red">selected и red</div>
<script>
  const elementsRed =
  document.getElementsByClassName('red');
  console.log(elementsRed.length); // 1

  const elementsSelected =
  document.getElementsByClassName('selected');
  console.log(elementsSelected.length); // 2

  const elementsBlue =
  document.getElementsByClassName('blue');
  console.log(elementsBlue.length); // 0
</script>
```

`document` И ОТДЕЛЬНЫЕ РОДИТЕЛИ

Этот метод так же можно вызывать у всего документа — `document`, или у конкретного объекта узла, тогда поиск будет происходить только по его потомкам.

ИЩЕМ ПО КЛАССУ ВНУТРИ РОДИТЕЛЯ

```
<div id="div1">
  <div class='red'>red 1</div>
</div>
<div id="div2">
  <div class='red'>red 2</div>
  <div class='red'>red 3</div>
</div>
<script>
  const div1 = document.getElementById('div1');
  const div2 = document.getElementById('div2');

  console.log(div1.getElementsByClassName('red').length);
  // 1
  console.log(div2.getElementsByClassName('red').length);

```

ИЗМЕНЕНИЕ КЛАССОВ ТЕГОВ

РАБОТАЕМ С КЛАССАМИ ЧЕРЕЗ `className`

Для доступа к атрибуту `class` любого тега в объекте-узле есть свойство-строка с именем `className`.

Почему не просто `class`? В JavaScript слово `class` является зарезервированным словом, и начиная с **ES2015** с его помощью действительно можно создавать классы.

Поэтому вместо `class` мы используем `className`.

`className` НА КЛИК

Установим свойство `className` на клик по элементу:

```
<style>.red {color: red;}</style>

<button id='btn1'>Нажми и станет красным</button>

<script>
  const btn1 = document.getElementById('btn1');
  btn1.onclick = () => {
    if (btn1.className !== 'red') {
      btn1.className = 'red';
    } else {
      btn1.className = '';
    }
  };
</script>
```

РАБОТАЕМ С НЕСКОЛЬКИМИ КЛАССАМИ

Использовать `className` в случае одного класса достаточно удобно. Усложним ситуацию и добавим еще класс.

```
<style>
  .red {color: red;}
  .selected {border: 5px solid green;}
</style>

<button id='btn1' class='red'>Нажми на меня</button>
```

ЕЩЕ ОДИН КЛАСС

Live Demo

```
const btn1 = document.getElementById('btn1');

function toggleSelectedClass() {
  const classNames = btn1.className.split(' ');
  const index = classNames.indexOf('selected');
  if (index === -1) {
    classNames.push('selected');
  } else {
    classNames.splice(index, 1);
  }
  btn1.className = classNames.join(' ');
}

btn1.onclick = toggleSelectedClass;
```

`classList`

Из-за того, что `className` — строка, работать с несколькими классами не очень удобно. Поэтому появился новый интерфейс для работы с ними — через свойство `classList` и его методы.

ОГРАНИЧЕНИЕ БРАУЗЕРОВ

`classList` не доступен в старых версиях Internet Explorer (9 и ниже). Более подробную информацию можно посмотреть по ссылке caniuse.com/#feat=classlist.

МЕТОДЫ `classList`

Вся работа происходит через методы объекта `classList`, вот основные:

- `add()` – для добавления класса;
- `remove()` – для удаления класса;
- `contains()` – для проверки, установлен ли такой класс или нет;
- `toggle()` – для переключения класса (если он уже был, то будет удален, если его не было – будет добавлен).

ОБНОВИМ ФУНКЦИЮ

Тогда функцию `toggleSelectedClass()` из прошлого примера можно заметно укоротить:

```
function toggleSelectedClass() {
  btn1.classList.toggle('selected');
}
```

УСЛОЖНЯЕМ ЗАДАЧУ

Давайте придумаем пример посложнее. Будем добавлять класс `selected` только тем объектам-узлам, где уже есть класс `red`.

```
<style>
  .red {
    color: red;
  }
  .selected {
    border: 5px solid green;
  }
</style>

<button class='red'>Нажми на меня</button>
<button>Нажми на меня</button>
```

ОБНОВИМ СКРИПТ

Live Demo

```
const buttons = document.getElementsByTagName('button');

function addSelectedClassIfRed() {
  if (this.classList.contains('red')) {
    this.classList.add('selected');
  }
}

for (const btn of buttons) {
  btn.onclick = addSelectedClassIfRed;
}
```

АУДИО И ВИДЕО

ВОСПРОИЗВОДИМ МУЗЫКУ И ВИДЕО

В JavaScript существуют специальные теги `<audio>` и `<video>` для воспроизведения музыки и просмотра видео соответственно. Мы познакомимся с их основными атрибутами и провзаимодействуем с ними из JavaScript.

ТЕГ `<audio>`

Для того, чтобы на странице отобразился простейший аудиоплеер, достаточно одной строчки.

```
<audio controls src="http://bit.ly/NetoSong"></audio>
```

Live Demo

АТРИБУТЫ ТЕГА `<audio>`

На предыдущем слайде мы видим тег `<audio>` с атрибутом `src`, в котором указан путь до звукового файла. Атрибут `controls` нужен для отображения простеших элементов управления — *ползунка*, кнопок воспроизведения и паузы.

ЕСЛИ БРАУЗЕР НЕ УМЕЕТ...

Есть браузеры, которые вообще не поддерживают воспроизведение звуковых файлов. Тогда для них можно воспользоваться следующим приемом:

```
<audio controls src="http://bit.ly/NetoSong">
    Ваш браузер не умеет воспроизводить музыку
</audio>
```

Если браузер пользователя не поддерживает тег аудио, то вместо плеера будет виден текст «Ваш браузер не умеет воспроизводить музыку».

ФОРМАТЫ МУЗЫКИ

Типы музыкальных файлов не ограничиваются **.mp3**. В спецификации HTML5 заявлена поддержка трех звуковых форматов: **.mp3**, **.wav** и **.ogg**. И их все можно указать одновременно с помощью нескольких тегов `<source>`.

ПОДКЛЮЧАЕМ НЕСКОЛЬКО ФОРМАТОВ

Структура очень простая. Есть тег `<audio>`, а внутри один или несколько тегов `<source>` с двумя атрибутами `src` и `type`. В `src` путь до звукового файла, в `type` указан **MIME** этого звукового файла:

```
<audio controls>
  <source src="music.mp3" type="audio/mpeg">
  <source src="music.ogg" type="audio/ogg">
  <source src="music.wav" type="audio/wav">
</audio>
```

БРАУЗЕР ВЫБИРАЕТ

Не каждый браузер поддерживает все эти форматы. В итоге он будет проигрывать первый формат, который умеет воспроизводить. В основном все современные браузеры умеют воспроизводить **.mp3**.

`controls`

Вернемся к атрибуту `controls`. Можно использовать тег `<audio>` и без `controls`, но в этом случае плеер будет невидим на странице.

СКРЫВАЕМ ПЛЕЕР

Можно, например, скрыть плеер по нажатию на кнопку. Live Demo

```
<audio id="player" controls>
  <source src="music.mp3" type="audio/mpeg">
</audio>

<button id="toggle_player">Скрыть/показать плеер</button>

<script>
  const btn = document.getElementById('toggle_player');
  const player = document.getElementById('player');

  btn.onclick = () => {
    player.controls = !player.controls;
  };
</script>
```

`autoplay`

У тега `<audio>` есть еще один нужный атрибут — `autoplay`. Если он указан, то воспроизведение аудио файла начнется автоматически, после того как страница и звуковой файл будут загружены.

```
<audio controls autoplay src="http://bit.ly/NetoSong">
</audio>
```

`loop`

Также у тега `<audio>` есть атрибут `loop` — отвечает за воспроизведения в цикле. Если установлен этот атрибут, то песни или песня будут играть по кругу.

```
<audio controls autoplay loop
src="http://bit.ly/NetoSong">
</audio>
```

УПРАВЛЯЕМ МУЗЫКОЙ

Как управлять воспроиведением при помощи JavaScript?

Давайте рассмотрим основные методы `play()` и `pause()` на примере следующей разметки:

```
<audio id="player" controls>
  <source src="music.mp3" type="audio/mpeg">
</audio>

<button id="player_play">Play</button>
<button id="player_pause">Pause</button>
```

ПИШЕМ СКРИПТ

Live Demo

```
const player = document.getElementById('player');
const btnPlay = document.getElementById('player_play');
const btnPause = document.getElementById('player_pause');

btnPlay.onclick = () => {
  player.play();
};
btnPause.onclick = () => {
  player.pause();
};
```

СТОП!

Логично было бы предположить, что существует еще один метод: `stop()`, который останавливает воспроизведениие и отматывает время к нулевой отметке.

Такого метода на данный момент не существует. Но мы можем имитировать его поведение при помощи следующего приема:

```
player.pause();
player.currentTime = 0; // отматываем на начало
```

ДОБАВИМ КНОПКУ `Stop`

Live Demo

```
<button id="player_stop">Stop</button>
```

```
const btnStop = document.getElementById('player_stop');

btnStop.onclick = () => {
  player.pause();
  player.currentTime = 0;
};
```

ПРОСТОЙ ВИДЕОПЛЕЕР

В общем-то тег `<video>` очень похож на тег `<audio>`, за исключением некоторых моментов. Начнем с простейшего видеоплеера.

```
<video controls src="http://bit.ly/NetoVideo"></video>
```

ШИРИНА И ВЫСОТА

Первое существенное различие в том, что для тега `<video>` возможно использование атрибутов `width` и `height`, которые отвечают за размер видеоплеера.

ИЗМЕНЯЕМ РАЗМЕРЫ

Ширину, высоту и атрибут `src` можно установить из JavaScript:

```
<video controls></video>
<script>
  const videos = document.getElementsByTagName('video');
  if (videos.length) {
    const player = videos[0];
    player.onclick = () => {
      player.width = 640;
      player.height = 480;
      player.src = 'http://bit.ly/NetoVideo';
    };
  }
</script>
```

ФОРМАТЫ ВИДЕО

Как и для звуковых файлов, в спецификации HTML5 определены 3 видео формата. в случае с видео они следующие: **.MP4**, **.WebM** и **.Ogg**.

.MP4 является наиболее поддерживаемым форматом для современных браузеров (как **.mp3** для звуковых файлов).

ВИДЕО В РАЗНЫХ ФОРМАТАХ

С учетом разных видеоформатов пример видеоплеера будет иметь следующий вид:

```
<video width="640" heigth="480" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  Ваш браузер не умеет воспроизводить видео
</video>
```

Здесь указаны и размеры (`width` и `height`) и атрибут `controls` для отображения управления. Указаны два разных файла с помощью тегов `<source>` и есть текст для браузеров, которые не поддерживают тег `<video>`.

BOM И ДИАЛОГОВЫЕ ОКНА

- BOM — объектная модель браузера.
- `alert`, `confirm` и `prompt` почти не используются.

НОВЫЕ МЕТОДЫ ПОИСКА

- `getElementsByTagName()` возвращает коллекцию элементов.
- Его можно вызывать у любого объекта-узла.
- Прототипом коллекции является `HTMLCollections`.
- Коллекцию можно перебрать при помощи `for` и `for-of`, но избегайте `for-in`.
- Коллекцию можно привести к массиву при помощи `Array.from()`.
- Ищем по имени класса при помощи `getElementsByClassName()`.
- Методы `getElementsByTagName()` и `getElementsByClassName()` очень похожи.

РАБОТА С КЛАССАМИ ЭЛЕМЕНТОВ

- Работаем с одним классом при помощи `className`.
- Работаем с несколькими классами при помощи `classList`.
- `classList` отпадает, если нужен IE9-.
- Основные методы `classList`: `add()`, `remove()`, `contains()` и `toggle()`.

АУДИО

- Простейший аудиоплеер: `<audio controls src="файл"></audio>`
- Поддерживается 3 аудиоформата: .mp3, .wav и .ogg.
- Песня в нескольких форматах подключается при помощи вложенного тега `<source>`.
- `autoplay` для автоматического воспроизведения (Не надо так).
- `loop` для проигрывания по кругу.
- Есть методы `play()` и `pause()`, но нет метода `stop()`.

ВИДЕО

- Простой видеоплеер `<video controls src="файл"></video>`
- Видеоплееру можно задавать ширину и высоту.
- Поддерживаются 3 формата видео: .MP4, .WebM и .Ogg.
- `<video>` во всем похож на `<audio>`, кроме размеров.

нетология
университет интернет-профессий

Задавайте вопросы и напишите отзыв о лекции!

АРТЕМ ШАШКОВ


glomix@inbox.ru


Артем Шашков