**Глава 4. ЭЛЕМЕНТ VIDEO**

**4.1. Назначение элемента VIDEO и принципы его применения**

Элемент <video> – новый элемент языка HTML5, предназначенный для воспроизведения видеозаписей. До появления HTML5 и <video> для этого в большинстве случаев применялся плагин мультимедийной платформы Adobe Flash Player [9], который может быть встроен в html-страницу с помощью элемента <object>.

Помимо собственно html-элемента <video> HTML5 предоставляет программный интерфейс позволяющий разработчику на JavaScript управлять воспроизведением, создавать пользовательские интерфейсы управления и интегрировать видео с остальными html-элементами и программными интерфейсами (например, с Canvas API).

В простейшем случае элемент <video> в исходном коде html-страницы выглядит так, как это показано на рис. 4.1. Элемент содержит только два атрибута: идентификатор (id) и путь к видеофайлу (src).



Рис. 4.1. Простейший вид элемента <video>

Обратите внимание: в элементе <video> (рис. 4.1) даже не указывается размеры окна для воспроизведения видеофайла – в этом случае, автоматически создается окно, размеры которого определяются шириной и высотой кадра записанного видео. Получить эти характеристики в Windows можно, просмотрев вкладку «Подробно», свойств видеофайла, указанного атрибутом src (рис. 4.2). Кроме размеров окна, в свойствах видеофайла можно найти еще много полезной информации: продолжительность видео, частоту кадров, скорость передачи потоков видео и аудиоданных и пр. Вся эта информация используется элементом <video> при воспроизведении по умолчанию.

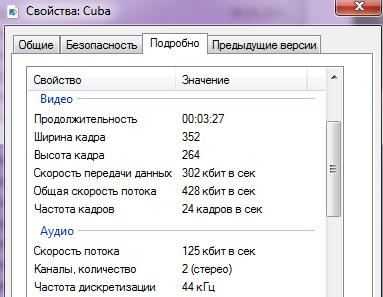


Рис. 4.2. Фрагмент скриншота окна свойств видеофайла

В окне браузера элемент <video> (рис. 4.1) отобразится примерно так, как это продемонстрировано на рис. 4.3. Внешний вид видеоплеера зависит от браузера

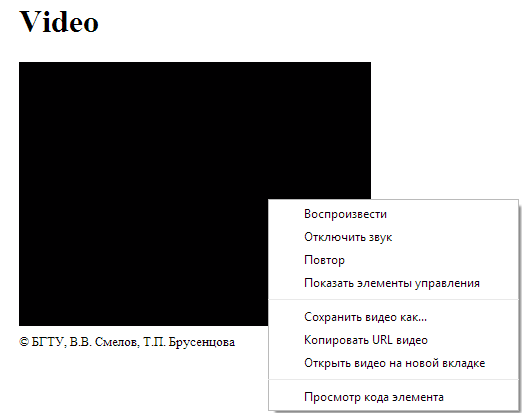


Рис. 4.3. Отображение браузером html-файла на рис. 4.1

Обратите внимание на следующее: браузер не отображает никаких элементов управления воспроизведением (пуск, стоп, пауза и т. д.); размеры окна воспроизведения, скорость воспроизведения и пр. определяются метаинформацией, записанной в видеофайле; для управления воспроизведением доступно контекстное меню.

**4.2. Форматы видеофайлов**

Видеофайл в любого формата следует воспринимать как сжатый файл, содержащий видео-поток и аудио-поток. В табл. 4.1 приведены три наиболее распространенных видео-формата.

Таблица 4.1. Наиболее распространённые форматы

видеофайлов

|  |  |  |
| --- | --- | --- |
| Видео-формат | Стандарт видео | Стандарт аудио |
| mp4 | H.264 | AAC |
| ogg/ogv | Theora | Vorbis |
| webm | VP8 | Vorbis |

Каждая часть видеофайла (видео-поток и аудио-поток), представляет собой закодированные с целью уменьшения размера данные. Стандарты видео и аудио данных, приведенные в табл. 4.1, по сути, определяют алгоритм их сжатия. Устройства или программы, способные кодировать исходный сигнал в последовательность бит и, наоборот, декодировать последовательность бит в вид готовый для воспроизведения, называется кодеком. Файл, содержащий закодированные с помощью определенных кодеков видео- и аудио-данные называется контейнером. Поэтому, когда говорят о видео-формате, по сути, имеют ввиду формат контейнера.

СпецификацияHTML5 допускает применение любого видео-формата. Все зависит от реализации браузера, которая определяет перечень поддерживаемых им форматов (табл. 4.2).

Таблица 4.2. Поддержка форматов браузерами

|  |  |  |  |
| --- | --- | --- | --- |
| Браузер | mp4 | webm | ogg |
| Internet Explorer | + | - | + |
| Chrome | + | + | + |
| Firefox | + | + | + |
| Safari | + | - | - |
| Opera |  | + | + |

Отчасти такое положение с форматами видеофайлов скрашивается возможностью указать несколько источников в одном элементе <video> с помощью вложенных элементов <source> (рис. 4.4). В этом случае, заданные источники будут проверяться на возможность использования в порядке их перечисления в теле тега <video>. Если, ни один из перечисленных источников не может быть применен или элемент <video> вообще не поддерживается, в окно браузера выведется текстовое сообщение, расположенное ниже тегов <source>.

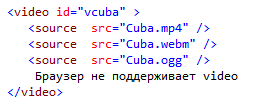


Рис. 4.4. Элемент <video> может указывать на

несколько видеофайлов

Для более полного знакомства с форматами видеофайлов рекомендуется источник [10].

**4.3. Потоковое видео**

В приведенных ранее примерах (рис. 4.1, 4.3) значение атрибута src (элементов <video> или <source>) представляет собой путь к локальному видеофайлу. Однако, если html-страница является компонентой web-приложения, видеофайл расположен на сервере и путь к нему должен быть представлен в формате URL (рис. 4.5). В этом случае доставка содержимого видеофайла будет осуществляться через сеть по протоколу HTTP.

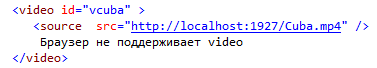


Рис. 4.5. В элементе <video> путь к исходному файлу может быть указан в формате URL

Если к html-странице, содержащий фрагмент подобный представленному на рис. 4.5, будет выполнено несколько http-запросов, в ответ на каждый будет осуществляться доставка содержимого одного и того же видеофайла. Учитывая, что общая скорость потока (битрейт) для каждого запроса должна составлять 428 килобит в секунду (рис. 4.2) несложно подсчитать, что при пятидесяти одновременно работающих подключениях к серверу, суммарный поток сервера, доставляющий видео должен составлять более 21 мегабайт в секунду. Кроме того, на стороне сервера не обязательно может быть видеофайл. Сервер может транслировать содержимое видеокамеры или другого устройства генерирующего видеопоток в реальном режиме времени. Очевидно, что количество возможных подключений к такому серверу будет ограничено и следует применять другие способы передачи видеопотоков.

Технологии потокового видео не рассматриваются в данном пособии, но отметим, что на сегодняшний день наибольшее распространение получили три технологии: HTTP Live Streaming компании Apple [11], Smooth Streaming от Microsoft [12] и HTTP Dynamic Streaming от Adobe.

**4.4. Применение элемента VIDEO**

Возьмем за основу html-страницу, представленную на рис. 4.1и 4.3, и поясним принципы применения элемента <video> и Video API, развивая это простейшее одностраничное web-приложение.

**4.4.1. Атрибут poster**

На рис. 4.6 представлен фрагмент html-страницы с элементом <video> , использующим атрибут poster.



Рис. 4.6. Применение атрибута poster элемента <video>

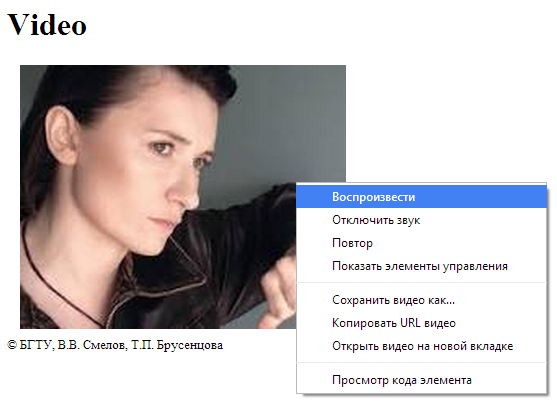


Рис. 4.7. Отображение браузером элемента <video> c атрибутом poster

Атрибут poster позволяет указать файл или URL картинки, которая будет отображаться в окне элемента <video> в начальный момент времени. Формат картинки, указываемой атрибутом poster, определяется применяемым браузером.

**4.4.2. Атрибут controls**

С помощью атрибута controls (рис. 4.8) в окно, отображаемое тегом <video> можно добавить элементы управления, позволяющие запускать, останавливать и позиционировать воспроизведение, регулировать звук и отображать текущую позицию видеофайла (рис. 4.9).



Рис. 4.8 Применение атрибута poster элемента <video>

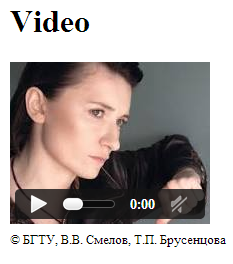


Рис. 4.7. Отображение браузером элемента <video> c атрибутом controls

**4.4.3. Атрибуты width и height**

Если в элементе <video> не используется атрибуты, задающие размеры окна для отображения видео, то по умолчанию будут установлены размеры, совпадающие с размерами кадра. Ширина и высота кадра видео является атрибутами видеофайла (рис. 4.2).

Размеры окна элемента <video> могут быть заданы атрибутами width (ширина) и height (высота). Какие бы значения для этих атрибутов не устанавливались, пропорции изображения при воспроизведении видео нарушены не будут.

Может потребоваться согласовать размеры окна <video> с размерами картинки-постера, чтобы панель элементов управления не «вылезла» за пределы стартовой картинки (рис. 4.8).

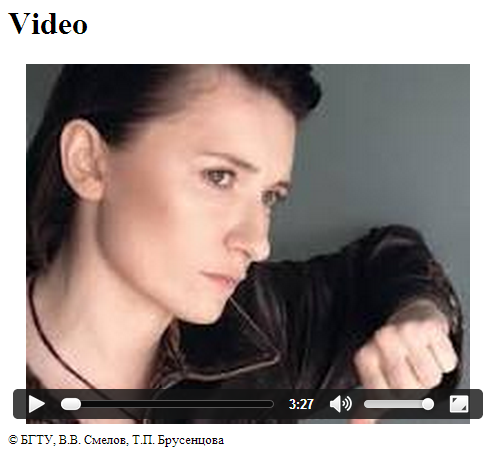


Рис. 4.8. Панель элементов управления тега <video> может «вылезти» за пределы картинки-постера

Кроме того, следует иметь ввиду, что если значения атрибутов width и height отличается от размера кадра, то при воспроизведении браузер будет постоянно (кадр за кадром) выполняет преобразование выводимого в окно изображения.

**4.4.4. Атрибуты preload, autoplay и loop**

На рис. 4.9 представлен элемент <video>, использующий атрибуты preload, autoplay и loop.

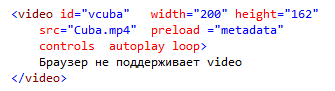


Рис. 4.9. Применение атрибутов preload, autoplay и loop

Атрибут preload используется для загрузки видео вместе с загрузкой html-страницей. Атрибут может принимать три значения: auto – указывает браузеру на необходимость загрузить видеофайл полностью; metadata – осуществляется загрузка только служебной информации (размеры кадра, продолжительность, скорость воспроизведения, продолжительность и пр.), none – значение по умолчанию, указывающее на то, что предварительную загрузки выполнять не надо.

При наличии атрибута autoplay виде начинает воспроизводиться автоматически сразу после загрузки страницы. Следует отметить, что атрибут autoplay отменяет действие атрибута preload.

Атрибут loop позволяет зациклить воспроизведение видео: сразу после окончания, видео будет повторяться сначала.

**4.5. Применение Video API**

На рис. 4.10 представлен фрагмент html-страницы, который далее будет применяться для демонстрации применения Video API.



Рис. 4.10. Фрагмент html-страницы, демонстрирующий применение Video API

Помимо собственно элемента <video>, фрагмент (рис. 4.10) содержит два элемента <div>. Первый (c идентификатором videoproperties) содержит разметку для отображения браузером некоторых свойств элемента <video> доступных для чтения и изменения JavaScript-программисту. Второй (c идентификатором videomethode) содержит пять элементов <input type=”button”>, предназначенных для выполнения JavaScript-кода, демонстрирующего управление элементом <video>, а также его взаимодействие с элементом <canvas> и web-камерой. Будем при этом предполагать, что весь, применяемый далее JavaScript-код расположен в одном файле Video.js, который загружен с помощью элемента <script>.

В браузере фрагмент html-страницы (рис. 4.10) будет выглядеть примерно так, как на рис. 4.11.

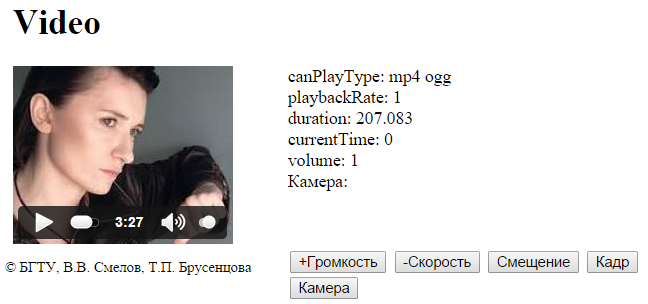


Рис. 4.11. Отображение браузером фрагмента html-страницы на рис. 4.10

Video API – набор JavaScript-функций, позволяющий считывать и изменять свойства, управлять воспроизведением, а также обрабатывать события элемента <video>.

Различные свойства и методы доступны на разных этапах работы элемента <video>. На рис. 4.12 представлен фрагмент кода файла Video.js, в котором определяется анонимная функция, выполняющаяся сразу после загрузки браузером html-страницы. Анонимная функция получает (с помощью функции getElementById) и записывает в глобальную переменную с именем video ссылку на объект элемента <video>, вызывает функцию WriteCanPlay, а также с помощью метода addEventListener динамически добавляет объекту элемента <video> обработчики двух событий: loadeddata и progress. Кроме того, обратим внимание, что для элемента <video> (рис. 4.10) указан атрибут onloadstart, указывающий на функцию WriteOnLoadStart, предназначенную для обработки события loadstart. Таким образом, для элемента <video> определены обработчики трех событий.

Первым из трех событий активизируется loadstart: сразу после того как браузер запрашивает ресурс видеофайла. Событие возникает только один раз.

Следующим будет обрабатываться событие progress: событие возникает после каждой считанной порции видеофайла.

Последним будет выполняться обработчик события loadeddata: событие возникает при каждом воспроизведении с текущей позиции видеофайла

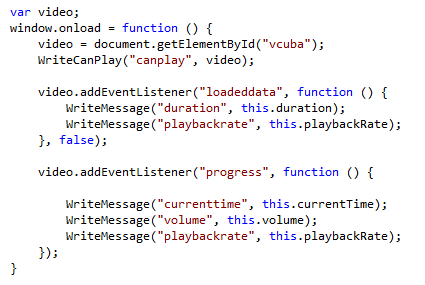


Рис. 4.12. Фрагмент файла Video.js: анонимная функция –

обработчик события load объекта window

Функция WriteCanPlay (рис. 4.13) предназначена для вывода форматов видеофайлов, поддерживаемых элементом <video> в текущем браузере. Результатом ее работы является строка, содержащая сокращенные названия разрешенных форматов. Функция принимает два параметра: идентификатор элемента <span> и ссылку на объект элемента <video>. Собственно проверка поддерживаемых типов браузером типов осуществляется методом canPlayType объекта элемента <video>. Метод canPlayType принимает один параметр: тип MIME [14] видеофайла. В том случае, если браузер не поддерживает заданный формат, метод возвращает пустую строку, иначе возвращается строковые значения maybe (поддержка, скорее всего, осуществляется) или probably (поддержка осуществляется). Заметим, что метод может выполняться еще до инициализации элемента <video>. Вывод результата осуществляется с помощью функции WriteMessage (рис. 4.13).

В обработчике события loadeddata (рис. 4.12) с помощью функции WriteMessage (рис. 4.13) отображаются свойства duration (продолжительность видео в секундах) и свойство playbackRate (коэффициент скорости воспроизведения). Начальное значение (по умолчанию) свойства playbackRate равно 1. Скорость воспроизведения видео элементом <video> вычисляется как произведение значение playbackRite на номинальную скорость потока (рис. 4.2).

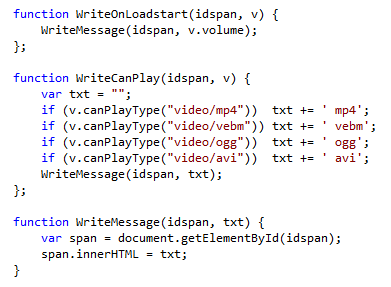


Рис. 4.13. Фрагмент файла Video.js:

функции WriteOnLoadStart и WriteCanPlay

В обработчике события progress (рис. 4.12), отображаются свойства currentTime (текущая позиция воспроизведения), volume (коэффициент громкости звука), а также уже рассмотренное выше текущее значение свойства playbackRate. Коэффициент volume, имеет начальное значение равное 1 и применяется при вычислении текущей громкости звука (аналогично свойству playbackRate). Номинальная громкость определяется установленным значением уровня громкости на компьютере.

Функции, представленные на рис. 4.14 вызываются при нажатии клавиш элементов <input type=”button”> (рис. 4.10, 4.11).

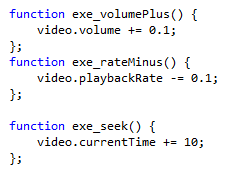


Рис. 4.14. Фрагмент файла Video.js:

функции exe\_volumePlus, exe\_rateMinus и exe\_seek

Функция exe\_volumePlus увеличивает значение свойства volume (коэффициент громкости звука) объекта элемента <video> на значение 0.1. Она вызывается при каждом нажатии клавиши «+Громкость» (рис. 4.10, 4.11) и увеличивает громкость звука.

Функция exe\_rateMinus уменьшает значение свойства playbackRate (коэффициент скорости воспроизведения) объекта элемента <video> на значение 0.1, вызывается при каждом нажатии клавиши «-Скорость» (рис. 4.10, 4.11) и замедляет скорость воспроизведения видео.

Функция exe\_rateMinus уменьшает значение свойства currentTime (текущая позиция воспроизведения) объекта элемента <video> на значение 10. Она вызывается при каждом нажатии клавиши «Смещение» (рис. 4.10, 4.11) и сдвигает воспроизведения видео на 10 сек. вперед.

Для более детального знакомства с возможностями Video API рекомендуется [15].

**4.6. Взаимодействие элементов VIDEO и CANVAS**

Элемент <video> , считывает указанный атрибутом src видеофайл, преобразует каждый считанный фрагмент этого файла в последовательность кадров (по сути, представляющих собой фотографии) и отображает их в окне элемента с заданной частотой (рис. 4.2). Каждый кадр, представляет собой набор пикселов. Поэтому он может быть использован в качестве источника для отображения на холсте элемента <canvas>.

Для демонстрации процедуры захвата кадра, воспроизводимого элементом <video>, и отображения его элементом <canvas> добавим на html-страницу (рис. 4.10) фрагмент, представленный на рис. 4.15.

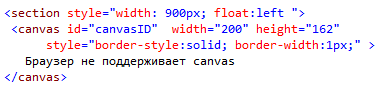


Рис. 4.15. Фрагмент html-страницы: элемент <canvas>

Обратите внимание: размер холста элемента <canvas> (рис. 4.15) совпадает с размерами окна элемента <video> (рис. 4.10).

В браузере измененный фрагмент html-страницы (рис. 4.10) будет выглядеть примерно так, как на рис. 4.16. При нажатии клавиши «Кадр» (рис. 4.10, 4.16) вызывается JavaScript-функция exe\_cadr (рис. 4.17).

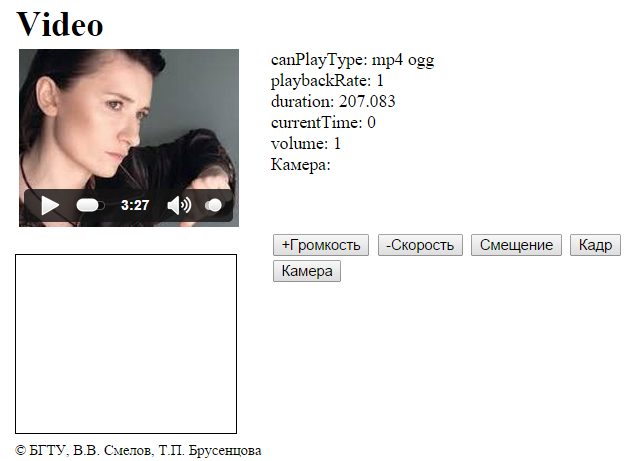


Рис. 4.17. Отображение браузером измененного

фрагмента html-страницы (рис. 4.10, 4.15.)

Функция exe\_cadr (рис. 4.17) принимает один параметр: идентификатор элемента <canvas>. Захват кадра, осуществляется с помощью метода drawImage, которой в качестве первого параметра указывается объект элемента <video>.

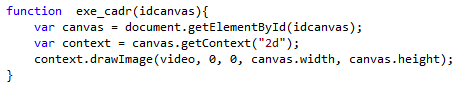


Рис. 4.18. Фрагмент файла Video.js: функция exe\_cadr

В результате выполнения функции exe\_cadr на холсте элемента <canvas> отобразится кадр, отображаемый в данный момент элементом <video>, примерно так, как это показано на рис. 4.19.



Рис. 4.19. Отображение браузером измененного

фрагмента html-страницы (рис. 4.10, 4.15.)

**4.7. Захват аудио и видеопотоков web-камеры**

На рис. 4.20 представлена функция exe\_camera, которая вызывается при нажатии клавиши «Камера» (рис. 4.10, 4.11).



Рис. 4.20. Фрагмент файла Video.js: функция exe\_camera

Для захвата аудио и видеопотоков в разных браузерах применяются функции, имеющие разные имена. Для обеспечения независимости от браузера может быть определена обобщенная функция navigator.getUserMedia (как это сделано на рис. 4.20), которая будет работать во всех браузерах. Функция принимает два параметра: объект со свойствами audio и video, функцию – обработчик потока.

Свойства первого параметра с помощью булевых значений указывают типы потоков, подлежащих захвату. В примере на рис. 4.20 захвату подлежат аудио и видеопотоки, формируемые микрофоном и камерой. При выполнении функции браузер запросит пользователя разрешение на использование камеры и микрофона.

Второй параметр задает функцию, вызываемую в случае подтверждения пользователем разрешение на применение камеры и микрофона. Функция принимает единственный параметр: ссылку на объект-поток.

Для отображения потока элементом <video>, с помощью функции createObjectURL формируется URL-объект, ссылка на который присваивается свойству src объекта элемента <video>. Если для <video> установлен атрибут autoplay, то видеопоток автоматически будет отображаться элементом <video>, иначе следует включить воспроизведение.