Введение в HTML5

ИНТУИТ

НАЦИОНАЛЬНЫЙ ОТКРЫТЫЙ УНИВЕРСИТЕТ

Введение в HTML5

2-е издание, исправленное

Национальный Открытый Университет "ИНТУИТ"
2016

Введение в HTML5/ - М.: Национальный Открытый Университет "ИНТУИТ", 2016

В курсе рассказывается об истории создания HTML5, возможностях, совместимости с другими стандартами, основных структурных элементах и их использовании в современных браузерах.
Демонстрируются новые элементы разметки и атрибуты, улучшающие формы, элемент video и создание индивидуального видеоплеера, "холст", API сокеты, кэширование приложение и хранилища данных на клиентских компьютерах. Дается введение в технологию многопоточного выполнения кода Web Workers и описываются основные возможности геолокации.

Знакомство с HTML5!

Основные вехи истории создания HTML5. Перспективы языка, сравнение с другими стандартами. Проблемы совместимости. Обсуждаются новые свойства языка разметки, а также технологии, не имеющие прямого отношения к стандарту HTML5, но тесно с ним связанные. API браузеров. JavaScript/DOM. Сокеты. Автономные web приложения, кэши приложений и локальная база данных Web SQL. Увеличение производительности за счет компонента Web Workers. Геолокация.

Крис Миллз · 14 января 2011 г.

Введение

Большая часть Учебного курса по стандартам web (ссылка: http://www.opera.com/wsc - http://www.opera.com/wsc) создана на последней стабильной версии языка HTML — HTML 4.01. Спецификация HTML 4.01 была завершена в 1999 г., более 10 лет назад на момент написания этой статьи! Но жизнь не стоит на месте и вам должно быть хорошо известно, что в настоящее время существует новая действующая версия HTML — HTML5!

Почему же мы, несмотря на это, учим вас HTML 4.01? В данной статье мы ответим на этот вопрос и на многие другие. Мы сообщаем все необходимые основные сведения о том, почему появился HTML5, и где он находится в настоящий момент. Мы обсудим, как он может вписаться в процесс обучения прямо сейчас, даже если вы являетесь новичком дизайна или разработки web, и мы рассмотрим некоторые основные свойства HTML5, чтобы вы могли увидеть, что он добавляет к уже достаточно мощному языку HTML.

Почему HTML5?

Когда HTML 4 приближался к завершению, консорциум W3C решил (на рабочем семинаре в 1998 г. - ссылка: http://www.w3.org/MarkUp/future/ - http://www.w3.org/MarkUp/future/), что с точки зрения языков разметки будущим Web является XML и XHTML, а не HTML. Поэтому W3C

подвел черту под HTML 4.01, и сконцентрировался после этого на спецификации XHTML 1.0, законченной в начале 2000 г. Язык XHTML 1.0 почти такой же, как и HTML 4.01, за исключением того, что использует правила синтаксиса разметки из XML. Вскоре последовал язык XHTML 2.0 (ссылка: http://www.w3.org/TR/xhtml2/ - http://www.w3.org/TR/xhtml2/), который добавил целый пакет новых мощных средств и тип mime только для XML, с целью стать следующей основой Web.

Проблема с XHTML 2.0 состояла в том, что он не являлся обратно совместимым с уже имеющейся в Web разметкой – элементы работали по-разному, тип mime XHTML (application/xhtml+xml) вообще не работал в IE, который по-прежнему удерживает большую долю рынка браузеров на время написания, доступные инструменты разработчиков были не готовы для работы с XML, который на самом деле не отражал, что разработчики Web ДЕЙСТВИТЕЛЬНО делали там, в ДИКОЙ, ДИКОЙ сети Web.

В 2004 г., группа одинаково мыслящих разработчиков и исполнителей (включая представителей Opera, Mozilla, и немного позже, Apple) собрались и сформировали отколовшуюся группу по спецификации с названием WHATWG (ссылка: http://www.whatwg.org/ - http://www.whatwg.org/), с целью написать лучшую спецификацию разметки HTML, которая сможет помочь созданию нового поколения приложений Web, не разрушая – что было критически важно – обратной совместимости.

В результате появилась спецификация Web Applications 1.0 (ссылка: http://www.whatwg.org/specs/web-apps/2005-09-01/ - http://www.whatwg.org/specs/web-apps/2005-09-01/), которая документирует существующие совместимые поведение и свойства браузеров, а также новые свойства стека Web, такие как API и новые правила синтаксического анализа DOM. После множества дискуссий между членами консорциума W3C 7 марта 2007 г. работа над HTML была возобновлена новой Рабочей группой HTML в виде открытого процесса участия. За несколько первых дней к процессу присоединились сотни участников для продолжения работы над следующей версией HTML. Одним из первых решений Рабочей группы HTML было принятие спецификации Web Applications 1.0, которую

назвали HTML5.

Язык HTML5 является действительно хорошей вещью для разработчиков и дизайнеров Web, так как он:

- Большей частью обратно совместим с тем, что там уже есть — не требуется учить совершенно новые языки для использования HTML5. Новые средства разметки работают таким же образом, как и старые (хотя семантика некоторых элементов изменилась – мы рассмотрим эти различия в будущей статье), и новые API основываются большей частью на том же JavaScript/DOM, который разработчики программировали в течение многих лет.
- Добавляет новые мощные средства в HTML, которые были ранее доступны в Web только с помощью технологии плагинов, такие как Flash, или с помощью сложного кода JavaScript и специальных приемов. Проверка форм и видео являются лучшими примерами.
- Лучше подходит для написания динамических приложений, чем предыдущие версии HTML (HTML был создан первоначально для создания статических документов).
- Имеет четко определенный алгоритм синтаксического анализа, так что все браузеры, реализующие HTML5, будут создавать одинаковое дерево DOM из одной и той же разметки, независимо от правильности. Это огромный выигрыш для совместимости.

Что HTML5 означает для меня?

Для начала давайте ответим на вопрос, который должен вертеться в голове с начала чтения этой статьи - зачем изучать стандарты Web с помощью HTML 4.01, если на горизонте виднеется HTML5? Прежде всего, когда Учебный курс по стандартам Web был впервые опубликован в 2008 г., HTML5 был по большей части в постоянном изменении, в отличие от сегодняшней ситуации, и было бы нежелательно учить что-то, что может измениться в ближайшем будущем.

Второе и более важное, HTML5 обратно совместим – в практическом смысле это означает, что весь материал из HTML 4.01 находится также в HTML5. Поэтому, изучая HTML 4.01, вы изучаете также большую часть HTML5. Эта статья и другие из этого раздела нацелены на то, чтобы

заполнить пробелы в ваших знаниях, излагая дополнительные возможности, которые HTML5 добавляет поверх HTML 4.01, например, дополнительные семантические элементы, и новые API, которые позволяют управлять воспроизведением видео и рисовать на холсте. Я предоставлю в конце статьи множество ссылок на дополнительную информацию о таких средствах.

Некоторые части HTML5 реализованы достаточно стабильным образом на различных браузерах, так что их можно безопасно использовать даже на производственном сайте (как обычно, вы должны будете решить это самостоятельно в зависимости от целевой аудитории сайта и свойств). Кроме того, если свойство не поддерживается в некоторых браузерах, это можно обойти – в статьях будет показано, как это сделать, в соответствующих местах.

Краткий заключительный ответ состоит в том, что HTML5 является будущим Web, и большой частью вашего будущего как дизайнера и разработчика Web. Я бы рекомендовал начать изучение HTML5, как только вы будете готовы – многие новые свойства сделают вашу работу значительно легче, и это будет будущее доказательство ваших знаний. Если вы студент и ваш преподаватель не знает еще о HTML5, или не хочет рассматривать его в своем курсе, покажите ему эту статью.

Свойства HTML5

Язык HTML5 содержит много новых свойств, что делает HTML значительно более мощным и удобным для создания приложений Web. В приведенном ниже списке суммируются основные свойства, о которых действительно необходимо знать.

Некоторые из перечисленных ниже свойств не являются на самом деле частью самой спецификации HTML5, но определены в тесно связанных спецификациях, поэтому они все еще являются допустимыми частями нового движения в направлении современных Web-приложений, и о них полезно знать.

Новые семантические элементы: Как вы возможно уже знаете, семантика является очень важной в HTML — мы всегда должны использовать для работы подходящий элемент. В HTML 4.01 мы имеем

проблему — да, существует много элементов для определения специальных средств, таких как таблицы, списки, заголовки, и т.д., но существует также много общих свойств web-страницы, которые не имеют элемента для их определения. Представьте верхние и нижние колонтитулы сайта, навигационные меню, и т.д. — до сих пор мы определяли их с помощью `<div id="xxx"></div>`, которые мы можем понять, но машины не могут, кроме того, различные разработчики web будут использовать различные ID и классы. К счастью, HTML5 содержит новые семантические элементы, такие как `<nav>`, `<header>`, `<footer>` и `<article>`. Мы познакомимся с ними подробнее в лекции "Новые структурные элементы HTML5".

Новые свойства форм: HTML 4.01 уже позволяет создавать удобные, доступные web-формы, но некоторые общие свойства форм являются не слишком удобными и требуют специальных усилий для реализации. HTML5 предоставляет стандартизованный, простой способ реализации таких свойств, как выбор даты, ползунки и клиентская проверка. Формы HTML5 подробно рассматриваются в лекции "Новые свойства форм в HTML5".

Собственная поддержка видео и аудио: В течение многих лет видео и аудио в Web делалось, вообще говоря, с помощью Flash. Фактическая причина, почему технология Flash стала так популярна на заре 21 века, состояла в том, что открытые стандарты не смогли предоставить для различных браузеров совместимый механизм реализации таких вещей, поэтому различные браузеры реализовали различные конкурирующие способы выполнения одних и тех же вещей (например, `<object>` и `<embed>`), делая тем самым весь процесс действительно сложным. Flash предоставлял высококачественный, легкий способ реализации работы видео в различных браузерах.

HTML5 содержит элементы `<video>` и `<audio>` для простой реализации собственных видео и аудио плееров с помощью только открытых стандартов, и также содержит API, позволяющий легко реализовать индивидуальные элементы управления плеером. На сайте dev.opera.com имеется много статей, посвященных видео и аудио в HTML5, но лучшим местом для начала является лекция Брюса Лоусона и Пэто Лауке "Введение в видео HTML5". Также полезно познакомиться с лекцией "Улучшение доступности видео-плеера HTML5".

API рисования на холсте: Элемент `<canvas>` и соответствующий API позволяют определить на странице область для рисования, и использовать команды JavaScript для рисования линий, фигур и текста, импорта и манипуляций с изображениями и видео, экспорта в различные форматы изображений, и многих других вещей. Больше о холсте в HTML5 можно узнать в лекции Миаи Сукан "Основы холста HTML 5".

Сокеты Web: Этот API (определенный в спецификации Сокеты Web - ссылка: http://www.w3.org/TR/websockets/ - http://www.w3.org/TR/websockets/, отдельно от спецификации HTML5) позволяет открывать постоянное соединение между сервером и клиентом на определенном порте, и посылать данные в обоих направлениях, пока порт не будет закрыт. Это существенно улучшает эффективность приложений web, так как данные могут непрерывно и аккуратно передаваться между клиентом и сервером без постоянной перезагрузки страницы, и без постоянного опроса сервера, чтобы проверить, нет ли доступных обновлений.

Лекция "Введение в сокеты Web" будет хорошим началом.

Автономные приложения web: HTML5 предоставляет ряд свойств, позволяющих приложениям web выполняться в автономном режиме. Кэши приложений позволяют сохранить копию всех ресурсов и других файлов, необходимых для локального выполнения приложения web, и базы данных Web SQL позволяют сохранить локальную копию данных приложения web. Совместно они позволяют продолжать использовать приложение, когда отсутствует соединение с сетью, и затем синхронизируют изменения с основной версией на сервере, когда сеть снова становится доступной.

Хранилище Web: Cookies предоставляют в какой-то степени локальное хранилище данных, но их использование довольно ограничено. Web хранилище HTML5 позволяет хранить значительно больше данных, и делать с ними значительно больше. Прочтите лекцию Шветанка Диксита "Хранилище Web: более удобное и мощное клиентское хранилище данных", чтобы больше об этом узнать.

Web workers: Общая проблема, встающая перед приложениями web, состоит в том, что их производительность страдает, когда требуется

обработать много данных, в связи с тем, что все происходит в одной нити/процессе (только одна последовательность обработки может выполняться в текущий момент). Web Workers смягчают эту проблему, позволяя создавать фоновые процессы для выполнения значительного объема вычислений, позволяя основному процессу продолжить выполнение других задач. Узнать больше о Web Workers можно в лекции Дениела Дэвиса "Web Workers за работой!".

Геолокация: Спецификация геолокации (ссылка: http://dev.w3.org/geo/api/spec-source.html - http://dev.w3.org/geo/api/spec-source.html) (также не являющаяся частью спецификации HTML5) определяет API, который позволяет приложению web легко получить доступ к данным в любом местоположении, которое стало доступным, например, с помощью средств GPS устройства. Это позволяет добавлять в приложения различные полезные свойства, связанные с местоположением, например, выделить контент, который больше подходит для местоположения. Чтобы получить общее представление о возможностях геолокации, прочтите лекцию "Как использовать API геолокации W3C".

Примечание: CSS3 СОВЕРШЕННО ОПРЕДЕЛЕННО НЕ является HTML5, и никогда не будет. Не позволяйте отделу маркетинга говорить вам обратное.

Заключение

На этом завершается краткое введение в HTML5. Будем надеяться, что оно будет полезно, и что вы продолжите чтение других указанных выше статей о HTML5.

Новые структурные элементы в HTML5

В лекции рассматриваются структурные элементы HTML5. Демонстрируется, каким образом новые свойства языка взаимодействуют друг с другом в контексте реальной web-страницы. Использование тегов логической компоновки документа: колонтитулов, навигационных панелей, автономных фрагментов контента, блоков для иллюстраций, разметки времени и даты. Специфика описания типа документа (DTD - Document Type Definition). Как новые средства языка поддерживаются в популярных браузерах. Методики, позволяющие заставить старые клиентские программы отображать неизвестные элементы разметки.

Крис Миллз, Брюс Лоусон · 21 декабря 2010 г.

Введение

HTML5 содержит две новые вещи: новые API, которые добавляют существенно важные новые свойства к открытым стандартам модели разработки web, и новые структурные элементы, которые определяют специальные свойства страницы Web со значительно более точной семантикой, чем это было доступно в HTML 4. Статьи, посвященные многим новым API, можно найти на Dev.Opera (ссылка: http://dev.opera.com/articles/tags/html5/ - http://dev.opera.com/articles/tags/html5/) с меткой HTML5.

Данная статья рассматривает другую составляющую – мы кратко опишем, как были выбраны новые семантические элементы, какие новые свойства имеются, и как они используются, как в HTML5 работают заголовки, и поддержку в браузерах новых элементов, включая их поддержку в старых браузерах.

Введение в структурные элементы HTML5

HTML4 имеет множество семантических элементов, позволяющих четко определять различные свойства страницы Web, такие как формы, списки, параграфы, таблицы и т.д. Однако он имеет и свои недостатки. Существенно используются элементы `<div>` и `<span>` с различными

атрибутами `id` и `class` для определения различных других свойств, таких как навигационные меню, верхние и нижние колонтитулы, основной контент, окна предупреждения, боковые панели, и т.д. Что-нибудь типа `<div id="header">` будет понятно разработчикам и дизайнерам, знающим, для чего это предназначено, и умеющих использовать CSS и JavaScript для применения собственных стилей и поведения, чтобы сделать это понятным для конечных пользователей.

Все могло бы быть значительно лучше, но с такими настройками по-прежнему существуют проблемы:

- Люди могут понять разницу между различным контентом, но машины не могут — браузер не видит различные `div` как верхние и нижние колонтитулы и т.д. Он видит их как различные `div`. Не будет ли полезнее, если браузеры и считыватели экрана смогут явно идентифицировать, скажем, навигационное меню, чтобы пользователям с недостатками зрения было легче его найти, или различные элементы новостей в пакете блогов, чтобы их было легче объединить в ленту RSS без дополнительного программирования?
- Даже если для решения некоторых из этих проблем используется дополнительный код, вы можете сделать это надежно только для собственных web-сайтов, так как другие разработчики web будут использовать другие имена классов и ID, особенно, если рассмотреть международную аудиторию — различные разработчики web в разных странах будут использовать различные языки для записи имен своих классов и id.

Поэтому имеет смысл определить согласованное множество элементов для общих структурных блоков, которые часто появляются на большинстве Web-сайтов. В данной статье будут рассмотрены следующие новые элементы HTML5:

- `<header>`: Используется для верхнего колонтитула сайта.
- `<footer>`: Используется для нижнего колонтитула сайта.
- `<nav>`: Содержит навигационные функции страницы.
- `<article>`: Содержит автономный фрагмент контента, который будет иметь смысл, если используется как позиция RSS, например,

новостное сообщение.

- `<section>`: Используется либо для объединения в группу различных статей с различной целью или по различным темам, или для определения различных разделов одной статьи.
- `<time>`: Используется для разметки времени и даты.
- `<aside>`: Определяет блок контента, который связан с основным контентом, но не входит в его основной поток.
- `<hgroup>`: Используется в качестве оболочки скрытия более одного заголовка, если требуется, чтобы учитывался только один заголовок в структуре заголовков страницы.
- `<figure>` и `<figcaption>`: Используется для инкапсуляции рисунка как единого элемента, и содержит, соответственно, подпись для рисунка.

Как были выбраны названия элементов?

Во время создания HTML5 редактор Ян Хиксон использовал инструменты Google для извлечения данных более чем миллиарда web-страниц, определяя, какие имена ID и class использовались наиболее часто в реальной сети web. Можно посмотреть один из подготовленных обзоров, опубликованный на сайте Google Code (ссылка: http://code.google.com/webstats/2005-12/classes.html - http://code.google.com/webstats/2005-12/classes.html). Выражаясь короче, имена элементов, указанные в этой статье, были взяты из 20 наиболее популярных имен ID и class, представленных в обзорах Хиксона.

Примечание: Компания Opera выполнила аналогичное исследование 3.5 миллионов URL, назвав его MAMA. MAMA использует меньшее множество URL, но рассматривает более широкий набор статистики Web-страниц. Дополнительную информацию можно найти на домашней странице проекта MAMA (ссылка: https://dev.opera.com/tags/mama/ - https://dev.opera.com/tags/mama/).

Почему нет элемента <content>?

Хотя это может казаться кричащим упущением, на самом деле это не

так. Основной контент будет блоком верхнего уровня контента, который не является `<header>`, `<nav>` или `<footer>`, и в зависимости от конкретных обстоятельств, может иметь больше смысла пометить контент с помощью `<article>`, `<section>`, или даже `<div>`. Брюс Лоусон называет это "алгоритмом Scooby Doo", но чтобы понять почему, вы должны спросить его самого в Twitter или на конференции.

Представляем пример страницы HTML5

Рассмотрев в некоторой степени предпосылки и увидев, какие новые элементы предлагаются, давайте теперь перейдем к примеру и посмотрим детально, как использовать их в контексте реальной страницы.

Посмотрите на мою страницу A history of Pop Will Eat Itself (ссылка: http://dev.opera.com/articles/view/new-structural-elements-in-html5/pwei_sample_html5.html - http://dev.opera.com/articles/view/new-structural-elements-in-html5/pwei_sample_html5.html) – историю и дискографию одной из моих любимых музыкальных групп 80/90-х (если вам нравится альтернативная музыка, пожалуйста, подтвердите это). Я использовал исходную разметку страницы Wikipedia Pop will Eat Itself (ссылка: http://en.wikipedia.org/wiki/Pop_Will_Eat_Itself - http://en.wikipedia.org/wiki/Pop_Will_Eat_Itself), почистил ее, и превратил в HTML5. Давайте внимательно посмотрим на то, что было сделано.

Держите пример страницы открытым в отдельной вкладке во время чтения статьи – вы можете захотеть вернуться к ней.

Пример использует традиционную проверенную оболочку `<div>` для размещения контента по центру, но Крок Камен опубликовал интересную статью о создании центрированного дизайна без оболочки `<div>` (ссылка: http://camendesign.com/code/developpeurs_sans_frontieres - http://camendesign.com/code/developpeurs_sans_frontieres), поэтому я подумал, что покажу ее здесь также. Также полезно порекомендовать не использовать элементы HTML5 `<section>` в качестве оболочки на страницах HTML5 – это просто совершенно неверно!

Некоторые мета-различия

Первое, что вы заметите, состоит в том, что doctype значительно проще, чем в более старых версиях HTML:

```
<!DOCTYPE html>
```

Создатели HTML5 выбрали самую короткую возможную строку doctype для этой цели — в конце концов, почему разработчик должен помнить длинную строку, содержащую множество URL, когда в действительности doctype присутствует здесь только для того, чтобы задать для браузера стандартный режим (в противоположность необычному режиму)?

Затем, я хотел бы привлечь ваше внимание к кажущимся "слабым синтаксическим требованиям" HTML5. Я добавил кавычки вокруг всех значений атрибутов, и написал все элементы строчными буквами, но это, на самом деле, связано с привычкой писать по правилам XHTML. Но для кого-то может показаться удивительным открытие, что в HTML5 можно при желании игнорировать эти правила. Фактически не нужно даже беспокоиться о том, чтобы использовать элементы `<head>`, `<body>`, или `<html>`, все будет по-прежнему допустимо!

Примечание: Это не так, если вы переключаетесь на использование XHTML (HTML работающий с doctype XHTML— `application/xhtml+xml`)

Дело в том, что такие элементы подразумеваются браузером в любом случае. Если создать пример страницы HTML5 без этих элементов, загрузить ее в браузер и посмотреть исходный код загруженной страницы, то можно заметить, что они будут автоматически вставлены браузером. Иначе можно воспользоваться утилитой-просмотрщиком Яна Хиксона Live DOM (ссылка: http://software.hixie.ch/utilities/js/live-dom-viewer/ - http://software.hixie.ch/utilities/js/live-dom-viewer/), чтобы увидеть состояние DOM.

Примечание: На самом деле можно также отправить на проверку документ HTML4, не содержащий `<head>`, `<body>`, или `<html>`, но это, тем не менее, стоит здесь отметить.

Необходимо также упомянуть, что спецификация HTML5 строго определяет, как обрабатывать плохосформированную разметку

(например, неправильно вложенные элементы, или незакрытые элементы), впервые определяя алгоритм синтаксического разбора. Это означает, что даже если разметка выполнена неправильно, DOM будет согласован с поддерживающими HTML5 браузерами.

Не означает ли это, что нам не нужно больше беспокоиться о проверке правильности и эффективных методах работы? ВОВСЕ НЕТ! Проверка правильности по-прежнему является очень полезным инструментом для создания страниц насколько возможно хорошими. Даже если DOM совместим с браузерами, он по-прежнему может вести себя, прежде всего не так, как требуется, создавая проблемы в CSS и JavaScript! И как вы увидите при дальнейшем изучении HTML5, существуют очень серьезные причины для проверки, что такие свойства документа как `<html>` объявляются явно. Например, вы можете захотеть объявить язык документа в элементе <html> для интернационализации и улучшения доступности, и это требуют также некоторые связанные технологии. Хорошим примером является AppCache (ссылка: http://dev.opera.com/articles/view/offline-applications-html5-appcache/ - http://dev.opera.com/articles/view/offline-applications-html5-appcache/).

Для проверки документов HTML5 можно использовать валидатор W3C (ссылка: http://validator.w3.org/ - http://validator.w3.org/), который может проверять HTML5, а также большой спектр других разновидностей языков разметки. Или использовать специальный валидатор HTML5 (+ WAI-ARIA и MathML) по адресу validator.nu (ссылка: http://validator.nu/ - http://validator.nu/).

В конце этого раздела я хочу привлечь ваше внимание к следующей строке:

```
<meta charset="utf-8" />
```

Вам необходимо объявить множество символов документа среди первых 512 байтов, чтобы защититься от серьезной угрозы безопасности. Если нет действительно серьезной причины не делать это, то рекомендуется использовать множество символов UTF-8.

<header>

Верхний колонтитул документа выглядит следующим образом:

```
<header>
  <hgroup>
    <h1>A history of Pop Will Eat Itself</h1>
    <h2>Introducing the legendary Grebo Gurus!</h2>
  </hgroup>
</header>
```

Назначение элемента `<header>` состоит в создании оболочки вокруг раздела контента, который формирует верхний колонтитул страницы, содержащий обычно логотип компании/графику, заголовок основной страницы, и т.д.

<hgroup>

Вы можете заметить, что в приведенном выше коде, единственным контентом колонтитула является элемент `<hgroup>`, содержащий два заголовка. Я хочу здесь определить заголовок документа верхнего уровня, плюс подзаголовок/ключевую фразу. Мне нужно, чтобы только заголовок верхнего уровня учитывался в иерархии заголовков документа, и именно это делает элемент `<hgroup>` - он позволяет учитывать группу заголовков как один заголовок в структуре документа. Подробнее о работе иерархии заголовков в HTML5 будет написано ниже в разделе "Разбивка HTML5 и алгоритм заголовков HTML5".

<footer>

Внизу документа можно увидеть следующий код:

```
<footer>
  <h3 id="copyright">Copyright and attribution</h3>
</footer>
```

`<footer>` необходимо использовать для размещения контента нижнего колонтитула сайта — если просмотреть нижнюю часть какого-то количества сайтов, то легко увидеть, что нижние колонтитулы могут

содержать различные вещи, от уведомления об авторских правах и контактной информации, до заявления о доступности, информации о лицензировании и множества других второстепенных ссылок.

Примечание: Не существует ограничения только на один верхний и нижний колонтитул на страницу — страница может содержать несколько статей, и иметь для каждой статьи верхний и нижний колонтитул.

<nav>

Двигаясь дальше по документу, мы встречаемся со следующей структурой:

```
<nav>
  <h2>Contents</h2>
    <ul>
      <li><a href="#Intro">Introduction</a></li>
      <li><a href="#History">History</a>

      <!—другие навигационные ссылки ... -->

    </ul>
</nav>
```

Элемент `<nav>` предназначен для разметки навигационных ссылок или других конструкций (например, формы поиска), которые направляют вас на различные страницы текущего сайта, или различные области текущей страницы. Другие ссылки, такие как рекламные ссылки, не учитываются. Можно, конечно, включать заголовки и другие структурные элементы внутрь `<nav>`, но это не обязательно.

<aside>

Сразу под заголовком документа имеется следующий код:

```
<aside>
```

```
  <table>
    <!—- множество кратких фактов в этом месте -->
  </table>
</aside>
```

Элемент `<aside>` предназначен для разметки фрагментов контента, которые имеют отношение к основному контенту, но не вписываются явно в основной поток изложения. Например, в данном случае мы имеем пакет кратких интересных фактов и статистики о музыкальной группе, которые не очень хорошо подходят для основного контента. Другими подходящими кандидатами для элементов `<aside>` являются списки ссылок на внешний связанный контент, справочная информация, цитаты, и боковые панели.

<figure> и <figcaption>

Динамический дуэт из `<figure>` и `<figcaption>` был создан для решения достаточно специального множества проблем. Прежде всего, не казалось ли всегда немного семантически сомнительно и неясно размечать рисунок и его подпись как два параграфа, или как пару списка определений, или как-то иначе? И второе, что делать, когда требуется рисунок, состоящий из изображения, или двух изображений, или двух изображений и некоторого текста? Элемент `<figure>` хорошо подходит, чтобы объединить весь контент, из которого вы хотите составить один рисунок, будет ли это текст, изображения, SVG, видео, или что-то другое. Элемент `<figcaption>` затем помещается внутри элемента `<figure>`, и содержит описательный заголовок для этого рисунка. В мой пример включен простой рисунок, чтобы просто показать, как это используется:

```
<figure>
  <img src="pwei.png" alt="Old poppies logo" />
  <figcaption>
    The old poppies logo, circa 1987.<br /> <a href="http://www.flickr.com/photo
    Original picture on Flickr</a>, taken by bobcatrock.
  </figcaption>
</figure>
```

<time>

Элемент `<time>` позволяет определить точно выраженное значение даты и времени, которое одновременно понятно человеку и машине. Например, я пометил даты выпуска синглов музыкальной группы следующим образом:

```
<time datetime="1989-03-13">1989</time>
```

Текст между открывающим и закрывающим тегами может быть любым, подходящим для людей, посещающих сайт. При желании можно сделать это следующим образом:

```
<time datetime="1989-03-13">13th March 1989</time>
<time datetime="1989-03-13">March 13 1989</time>
<time datetime="1989-03-13">My nineteenth birthday</time>
```

С другой стороны дата внутри атрибута `datetime` представлена в стандарте ISO (см. дополнительные сведения в "Советы W3C: Используйте международный формат даты (ISO)" - ссылка: http://www.w3.org/QA/Tips/iso-date - http://www.w3.org/QA/Tips/iso-date), и является машинно-читаемой датой, поэтому мы получаем двойную выгоду. Можно также добавить время в конце стандартного представления ISO следующим образом:

```
<time datetime="1989-03-13T13:00">One o'clock in the afternoon, on the 13th
```

Можно добавить также настройку часового пояса, поэтому, например, чтобы представить последний пример в стандартном тихоокеанском времени, можно сделать следующее:

```
<time datetime="1989-03-13T13:00Z-08:00">One o'clock in the afternoon, on
```

<article> и <section>

Теперь обратим наше внимание к двум, возможно, наиболее неправильно понимаемым элементам HTML5 - `<article>` и `<section>`. При первом знакомстве различие может показаться

нечетким, но на самом деле не все так плохо.

В основном элемент `<article>` предназначен для независимых фрагментов контента, которые будут иметь смысл вне контекста текущей страницы, и могут хорошо объединяться. Такие фрагменты контента включают публикации в блоге, видео и его текстовая запись, новостная история, или одна часть серийной истории.

Элемент `<section>`, с другой стороны, предназначен для разбиения контента страницы на различные функциональные или тематические области, или разбиения статьи или истории на различные части. Поэтому, например, в моей истории PWEI, структура выглядит следующим образом:

```
<article>
  <section id="Intro">
    <h2>Introduction</h2>
  </section>

  <section id="History">
    <h2>History</h2>
  </section>

  <section id="Discography">
    <h2>Discography</h2>
  </section>
</article>
```

Но можно также структурировать следующим образом:

```
<section id="rock">
  <h2>Rock bands</h2>
  <!--multiple article elements could go in here -->
</section>

<section id="jazz">
  <h2>Jazz bands</h2>
  <!--multiple article elements could go in here -->
</section>
```

```
<section id="hip-hop">
  <h2>Hip hop bands</h2>
  <!--multiple article elements could go in here -->
</section>
```

Где остается ?

Итак, со всеми этими прекрасными новыми элементами, которые можно использовать на страницах Web, дни простого элемента `<div>` сочтены, не так ли? НЕТ. Фактически, `<div>` по-прежнему имеет совершенно законное применение. Его следует использовать, когда не существует другого более подходящего доступного элемента для объединения области контента, что часто происходит, когда вы используете элемент только для объединения контента в группу с целью стилевого или визуального оформления. Примером в моей истории PWEI является элемент `<div id="wrapper">`, который использован для создания оболочки вокруг всего контента. Единственная причина его использования здесь в том, чтобы я мог использовать CSS для выравнивания контента по центру в браузере:

```
#wrapper {
  background-color: #ffffff;
  width: 800px;
  margin: 0 auto;
}
```

<mark>

Элемент `<mark>` предназначен для выделения терминов, значимых в данный момент, или выделения частей контента, к которым вы просто хотите привлечь внимание, но не хотите изменять семантическое значение. Это похоже на то, как при просмотре напечатанной статьи вы выделяете важные для вас строки с помощью цветного маркера. Поэтому, например, вы можете захотеть использовать этот элемент для выделения строк в wiki, которые требуют редакторской правки, или выделения экземпляров термина поиска, который пользователь только

что искал на странице, и задание затем для них подходящего оформления в CSS.

Атрибут hidden

Атрибут `hidden`, когда применяется к любому элементу, скрывает его полностью от любых форм представления/медиа, и должен использоваться, если вы намерены показать контент позже (например, используя JavaScript для удаления этого атрибута), но не хотите, чтобы он отображался в данный момент. Он не должен использоваться для скрытия такого контента, как скрытые вкладки интерфейса с вкладками, так как это совершенно другой способ представления контента в меньшем пространстве, а не скрытие контента вообще.

Разбивка HTML5 и алгоритм заголовков HTML5

Прежде чем продолжить путешествие в направлении более глубокого овладения HTML5, необходимо обсудить одно важное отличие, которое существует между HTML5 и предыдущими версиями спецификации. В HTML мы имеем концепцию разбивки документа, которая является в основном разбивкой документа по его заголовкам, и их иерархии относительно друг друга, точно таким же образом, как при записи документа в текстовом процессоре вы просматриваете документ в виде плана-конспекта. На самом деле я фактически создал план-конспект этого документа, вкладывая списки друг в друга для создания оглавления в начале статьи. План-конспект этой статьи выглядит примерно следующим образом:

- Введение в структурные элементы HTML5
  - Как были выбраны названия элементов?
  - Почему нет элемента `<content>`?
- Представляем пример страницы HTML5
  - Некоторые мета-различия
  - `<header>`
  - `<hgroup>`
  - `<footer>`
  - `<nav>`

    - <aside>
    - <figure> и <figcaption>
    - <time>
    - <article> и <section>
        - Где остается ?
        - <mark>
        - Атрибут hidden
- Разбивка HTML5 и алгоритм заголовков HTML5
- Как заставить это работать в старых браузерах
- Заключение

Поэтому "Новые структурные элементы в HTML5" будет <h1>, "Введение в структурные элементы HTML5" будет <h2>, и т.д. В HTML4 мы привыкли к тому факту, что имеется шесть возможных уровней заголовков, и каждый уровень заголовков диктуется реально используемым элементом, что означает, что вполне возможно получить совершенно искаженную иерархию заголовков, если использовать неправильные уровни заголовков, или даже если часть вашего контента объединяется в другой CMS (Системе управления контентом).

HTML5 решает эту проблему, создавая иерархию заголовков на основе относительного вложения различных разделов документа. Новый раздел документа создается, когда вы используете так называемый разделяющий контент – элементы <article>, <section>, <nav>, и <aside>. Возьмем следующий пример.

```
<h1>My title</h1>
<div>
  <h2>My subtitle</h2>
</div>
```

Посмотрите первый пример разбивки в действии (ссылка: http://dev.opera.com/articles/view/new-structural-elements-in-html5/outlining1.html - http://dev.opera.com/articles/view/new-structural-elements-in-html5/outlining1.html).

HTML 4 будет считать это заголовком первого уровня, за которым следует заголовок второго уровня, но HTML5 будет считать это как два заголовка первого уровня. Почему? Так как не является

разделяющим элементом, то новый раздел в иерархии не создается. Чтобы исправить это, необходимо заменить `<div>` на разделяющий элемент:

```
<h1>My title</h1>
<section>
  <h2>My subtitle</h2>
</section>
```

Посмотрите на второй пример разбивки в действии (ссылка: http://dev.opera.com/articles/view/new-structural-elements-in-html5/outlining2.html - http://dev.opera.com/articles/view/new-structural-elements-in-html5/outlining2.html).

Ни один из браузеров в настоящее время не реализовал алгоритм HTML5 для разбивки, но вы уже можете получить представление о том, как это работает, используя Расширение Opera HTML5 Outliner (ссылка: https://addons.opera.com/addons/extensions/details/html5-outliner/1.0/?display=en - https://addons.opera.com/addons/extensions/details/html5-outliner/1.0/?display=en), сетевой HTML5 outliner Джефри Шеддона (ссылка: http://gsnedders.html5.org/outliner/ - http://gsnedders.html5.org/outliner/), или Google HTML5 outliner (ссылка: http://code.google.com/p/h5o/ - http://code.google.com/p/h5o/). Попробуйте пропустить приведенные выше примеры через один из этих инструментов, если вы мне не верите на слово. И в будущем в действительности не нужно будет беспокоиться об иерархии h1, h2, h3, и т.д., так как независимо от реально используемых элементов заголовков, алгоритм будет создавать одну и ту же иерархию на основе вложенности разделов документа. Но пока об этом необходимо волноваться, так как ни один браузер (или считыватель экрана) этого не поддерживает!

Поэтому возникает естественный вопрос – "Зачем вообще об этом беспокоиться"? На самом деле этот новый способ создания план-конспекта документа/иерархии заголовков имеет два основных преимущества по отношению к старому:

1. Можно иметь любое количество уровней заголовков — количество не ограничивается шестью.

2. Если контент переносится в какую-то другую CMS (Систему управления контентом), что приводит к тому, что уровни заголовков h1, h2, h3, и т.д.становятся неправильными, алгоритм будет по-прежнему создавать правильную иерархию несмотря ни на что.

Примечание: Иерархия заголовков HTML5 является в действительности достаточно старой идеей, первоначально намеченной Тимом Бернерс-Ли еще в 1991 г. (ссылка: http://lists.w3.org/Archives/Public/www-talk/1991SepOct/0003.html - http://lists.w3.org/Archives/Public/www-talk/1991SepOct/0003.html):

На самом деле, я бы предпочел вместо `<H1>`, `<H2>` и т.д. для заголовков [которые возникают из AAP DTD] иметь допускающий вложение друг в друга элемент `<SECTION>..</SECTION>`, и базовый `<H>..</H>`, который на любом уровне в разделах создавал бы заголовок требуемого уровня.

Как заставить это работать в старых браузерах

Старые браузеры: всегда бедствие для самого нашего существования, когда пытаешься справиться с новыми блестящими игрушками в Web! Фактически проблема здесь со всеми браузерами – ни один браузер в настоящее время по существу не распознает и не поддерживает новые структурные элементы HTML5. Но не надо бояться, вы можете, тем не менее, заставить их работать сегодня в разных браузерах с минимальными усилиями.

Прежде всего, если вы помещаете неизвестный элемент на web-страницу, по умолчанию браузер будет интерпретировать его просто как `<span>`, т.е. анонимный встроенный элемент. Предполагается, что большинство рассмотренных в этой статье элементов HTML5, ведут себя как блочные элементы, поэтому простейший способ заставить их вести себя правильно в старых браузерах, состоит в задании для них `display:block;` в коде CSS:

```
article, section, aside, hgroup, nav, header, footer, figure, figcaption {
  display: block;
```

```
}
```

Это решает все проблемы для всех браузеров, за исключением одного. Хотите угадать какого? … Удивительно, не правда ли, что IE оказался сложнее других браузеров, и отказывается оформлять элементы, которые не распознает? Исправление для IE кажется нелогичным, но к счастью достаточно простым. Для каждого используемого элемента HTML5 необходимо вставить строку JavaScript в заголовок документа следующим образом:

```
<script>
    document.createElement('article');
    document.createElement('section');
    document.createElement('aside');
    document.createElement('hgroup');
    document.createElement('nav');
    document.createElement('header');
    document.createElement('footer');
    document.createElement('figure');
    document.createElement('figcaption');
</script>
```

Теперь IE будет магическим образом применять стили для этих элементов. Печально, что приходиться использовать JavaScript, чтобы заставить работать CSS, но, по крайней мере, мы продвинулись вперед. Почему это все-таки работает? Никто, с кем я говорил об этом, в действительности не знает. Существует также проблема с этими стилями, которые не выводятся на принтер при печати документов HTML5 из IE.

Примечание: Проблема с печатью в IE может быть решена с помощью библиотеки JavaScript HTML5 Shiv (ссылка: http://code.google.com/p/html5shiv/ - http://code.google.com/p/html5shiv/), которая справляется также с добавлением строк `document.createElement`. Необходимо поместить ее в Условные комментарии (ссылка: http://dev.opera.com/articles/view/supporting-ie-with-conditional-comments/ - http://dev.opera.com/articles/view/supporting-ie-with-conditional-comments/) для IE меньше IE9, чтобы современные браузеры не выполняли JS, который им не нужен.

Заключение

На этом завершается обсуждение новых структурных элементов в HTML5. Если вам требуется дополнительная помощь с HTML5, многое можно найти на сайте dev.opera.com, и можно также проконсультироваться на сайте HTML5 doctors (ссылка: http://html5doctor.com/ - http://html5doctor.com/).

Новые свойства форм в HTML5

Рассказывается о новых элементах разметки и атрибутах, улучшающих формы. Примеры новых элементов числового ввода, ползунков, списков выбора даты и времени, выбора цвета из цветовой палитры, индивидуального поиска по сайту, элементов ввода со списком вариантов, поля для телефонных номеров, e-mail- и url-адресов. Новые механизмы вывода информации: вывод результатов вычислений, панель индикатора выполнения. Встроенная в HTML5 проверка заполнения формы.

Патрик Х. Лауке, Крис Миллз · 16 декабря 2010 г.

Введение

Язык HTML5 содержит много новых свойств, которые должны существенно облегчить создание форм в Web, и сделать их значительно более мощными и совместимыми в Web. Эта статья делает краткий обзор некоторых новых элементов управления форм и функций, которые были введены.

Плохая форма?

Посмотрим правде в лицо – формы HTML всегда были проблемой. Они не слишком интересны для разметки, и могут требовать кропотливой работы для оформления – особенно, если вы хотите, чтобы они были совместимы с разными браузерами и соответствовали общему виду вашего сайта. И они могут создавать трудности при заполнении для конечных пользователей, даже если они были созданы с заботой и вниманием, чтобы сделать их как можно удобнее и доступнее. Создание хорошей формы состоит в большей степени в ограничении ущерба, чем в обеспечении приятного взаимодействия пользователя.

Ранее, когда HTML 4.01 стал стабильной рекомендацией, сеть web была по большей части стабильным местом. Да, существовала странная форма для комментариев или сценарий для гостевой книги, но обычно web-сайты предназначались посетителям для простого чтения. С тех пор сеть web существенно изменилась. Для многих людей браузер уже

стал окном в мир сложных приложений на базе web, которые стараются обеспечить взаимодействие почти на уровне настольного компьютера.

Несколько сложных элементов управления формой, созданных с помощью JavaScript

Чтобы удовлетворить потребность в более развитых элементах управления, которые нужны для таких приложений, разработчики использовали библиотеки JavaScript и рабочие среды (такие как jQueryUI или YUI). Эти решения, конечно, существенно развились за прошедшие годы, добавляя богатую функциональность на страницы Web, начиная включать даже поддержку считывателей экрана с помощью таких технологий как WAI-ARIA. Но, по сути, они являются обходными путями, чтобы компенсировать ограниченные возможности элементов управления форм, которые имеются в HTML. Они "имитируют" более сложные виджеты (такие как выбор даты и ползунки), создавая дополнительный слой разметки (не всегда семантической), и дополнительный код JavaScript поверх страниц.

HTML5 нацелен на стандартизацию некоторых из наиболее распространенных элементов управления форм, делая их отображение в браузерах простым, исключая необходимость решений, которые существенно используют сценарии.

Знакомство с примером

Чтобы проиллюстрировать некоторые новые свойства, статья содержит

элементарный пример формы HTML5 (ссылка: http://dev.opera.com/articles/view/new-form-features-in-html5/html5-forms-example.html). Он не предназначен для представления "реальной" формы (так как трудно было бы, пожалуй, найти ситуацию, где могут потребоваться все новые свойства в одной форме), но она должна показать, как выглядят и ведут себя различные новые возможности.

Примечание: Внешний вид новых элементов управлении форм и контрольные сообщения варьируются в различных браузерах, поэтому оформление форм для согласованного представления в различных браузерах будет по-прежнему требовать тщательного рассмотрения.

Новые элементы управления форм

Так как формы являются основным инструментом ввода данных в приложениях Web, а требуемые для сбора данные стали более сложными, возникла необходимость создать элементы ввода с дополнительными возможностями, чтобы собирать эти данные с дополнительной семантикой и лучшим определением, и предоставить более простое, более эффективное управление ошибками и проверкой.

<input type="number">

Первый новый тип элемента ввода, который мы рассмотрим, будет тип ввода `number` (числа):

```
<input type="number" … >
```

Он создает специальный вид поля ввода для числа – в большинстве поддерживающих браузеров оно появляется как текстовое поле ввода со счетчиком, который позволяет увеличивать и уменьшать его значение.

Тип элемента ввода number

<input type="range">

Создание элемента управления ползунка, позволяющего выбирать в диапазоне значений, было обычно сложным, семантически неясным предложением, но в HTML5 все стало легко – вы используете просто тип ввода `range` (диапазон):

```
<input type="range" … >
```

Тип элемента ввода range

Отметим, что, по умолчанию, этот элемент ввода обычно не показывает выбранное в данный момент значение, или даже диапазон значений, который он охватывает. Автору нужно будет предоставить их с помощью других средств – например, для отображения текущего значения можно было бы использовать элемент вывода вместе с некоторым кодом JavaScript для обновления вывода, когда пользователь взаимодействует с формой:

```
<output onforminput="value=weight.value"></output>
```

<input type="date"> и другие элементы управления датой/временем

HTML5 имеет ряд различных типов элементов ввода для создания сложных средств выбора даты/времени, например, показанное ниже средство выбора даты имеется почти на каждом существующем сайте бронирования самолета/поезда. Такие средства обычно создавались с помощью несемантического путаного кода, поэтому здорово, что теперь существует стандартизованный простой способ сделать это. Например:

```
<input type="date" … >
<input type="time" … >
```

Соответственно эти коды создают полностью функциональное средство

выбора даты, и элемент ввода текста, содержащий разделитель для часов, минут и секунд (в зависимости от указанного атрибута `step`), который позволяет вводить только значение времени.

Типы элементов ввода date и time

Но на этом все не кончается – доступен ряд других связанных типов элементов ввода:

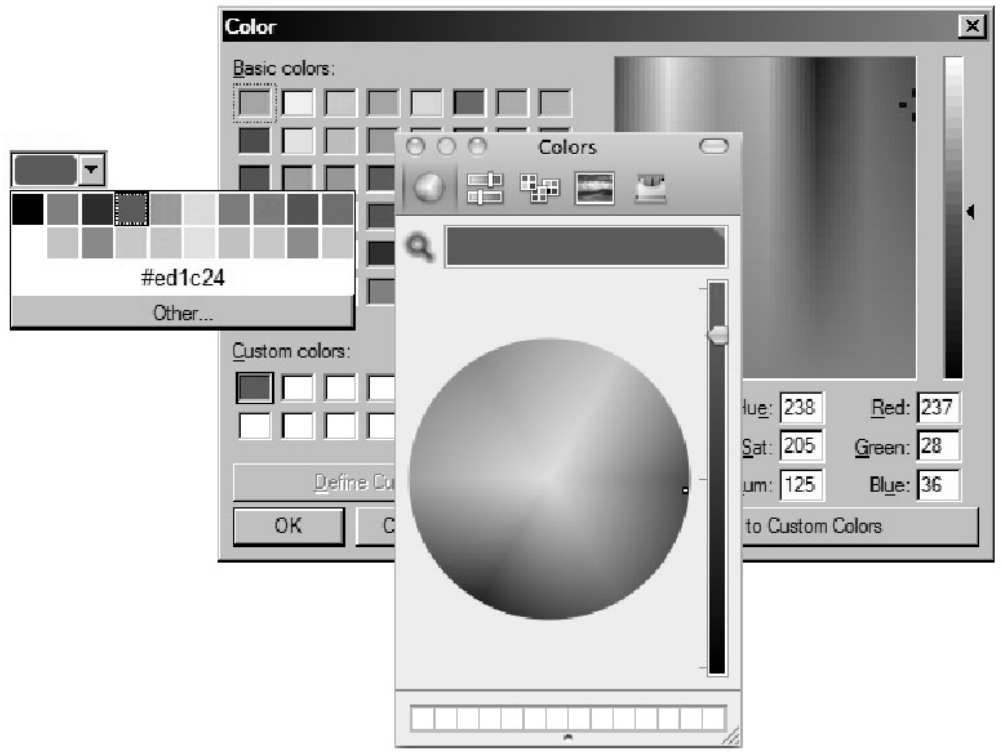
`datetime`: объединяет функции обоих рассмотренных выше элементов, позволяя выбрать дату и время.

`month`: позволяет выбрать месяц, сохраняя его внутренне как число между 1-12, хотя различные браузеры могут предоставлять более развитый механизм выбора, например, как прокручиваемый список названий месяцев.

`week`: позволяет выбрать неделю, сохраняемую внутренне в формате 2010-W37 (37 неделя 2010 года), и выбирать с помощью элемента выбора даты, аналогичного тому, что мы уже видели.

<input type="color">

Этот тип элемента ввода предоставляет элемент выбора цвета. Реализация в Opera позволяет пользователю выбрать из набора цветов, ввести шестнадцатеричные значения прямо в текстовое поле, или вызвать собственную процедуру выбора цвета операционной системы.



Элемент ввода color, и собственные инструменты выбора цвета в Windows и OS X

<input type="search">

Тип элемента ввода `search` является, возможно, ничем иным, чем иначе оформленным текстовым элементом ввода. Предполагается, что браузеры будут оформлять эти элементы ввода таким же образом, как оформляются любые функции поиска в конкретной операционной системе. Тем не менее, помимо этих чисто эстетических соображений, важно отметить, что явная разметка полей поиска открывает для браузеров, вспомогательных технологий, и автоматизированных

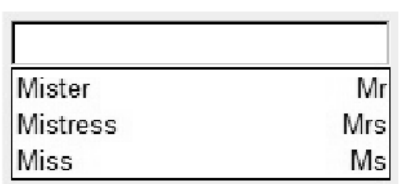
сканеров возможность сделать в будущем что-то полезное с помощью этих элементов ввода – например, браузер мог бы, возможно, предложить пользователю возможность автоматически создать индивидуальный поиск для определенного сайта.

Элемент ввода search, как он выглядит в браузере Opera в OS X

Элемент <datalist> и атрибут list

До сих пор мы привыкли использовать элементы `<select>` и `<option>` для создания раскрывающихся списков вариантов выбора для пользователей. Но как быть, если мы захотим создать список, который позволял бы пользователям выбирать из списка предложенных вариантов, а также позволял вводить свои собственные? Это требует кропотливого создания сценария – но теперь можно просто использовать атрибут `list`, чтобы соединить обыкновенный элемент ввода со списком вариантов, определенным в элементе `<datalist>`.

```
<input type="text" list="mydata" … >
<datalist id="mydata">
   <option label="Mr" value="Mister">
   <option label="Mrs" value="Mistress">
   <option label="Ms" value="Miss">
</datalist>
```



Создание элемента ввода с предложениями с помощью datalist

<input type="tel">, <input type="e-mail"> и <input type="url">

Как предполагают названия, эти новые типы элементов ввода имеют отношение к телефонным номерам, адресам e-mail, или URL. Браузеры будут отображать их как обычные текстовые поля ввода, но явное указание, какой вид текста ожидается в этих полях, играет важную роль в проверке формы на стороне клиента. Кроме того, на некоторых мобильных устройствах браузер будет переключаться из своей обычной экранной клавиатуры для ввода обычного текста в более контекстно-подходящие варианты. Здесь также возможно, что в будущем браузеры будут использовать дополнительные преимущества этих явно размеченных элементов ввода, чтобы предложить новые возможности, такие как автозавершение адресов e-mail и телефонных номеров на основе списка контактов пользователя или адресной книги.

Новые атрибуты

Кроме явно новых типов элементов ввода HTML5 определяет ряд новых атрибутов для элементов управления форм, которые помогают упростить некоторые общие задачи и дополнительно уточнить ожидаемые значения для определенных полей ввода.

placeholder

Обычный удобный прием в формах Web состоит в размещении замещаемого контента в текстовых полях ввода – например, чтобы предоставить дополнительную информацию об ожидаемом типе информации, ввод которой должен выполнить пользователь – который исчезает, когда элемент управления формы получает фокус. Хотя это обычно требует использования JavaScript (удаление контента поля формы в фокусе и восстановление текста по умолчанию, если пользователь покидает поле, не вводя ничего), мы можем теперь использовать просто атрибут `placeholder`:

```
<input type="text"… placeholder="John Doe">
```

Текстовое поле ввода с замещаемым текстом

autofocus

Другим обычным свойством, которое ранее требовало использования сценария, является автоматическое помещение поля формы в фокус при загрузке страницы. Теперь это можно сделать с помощью атрибута `autofocus`:

```
<input type="text" autofocus … >
```

Помните, что вы не должны иметь на странице более одного элемента управления формы с автофокусом. Необходимо также использовать функциональность такого вида с осторожностью в ситуациях, когда форма представляет основную область интереса на странице. Страница поиска является хорошим примером – при условии, что нет большого количества контента и пояснительного текста, имеет смысл задавать фокус автоматически на текстовом поле ввода формы поиска.

min и max

Как предполагают имена, эта пара атрибутов позволяет задать нижнюю и верхнюю границы для значений, которые могут вводиться в числовое поле формы, например, типы элементов ввода `number`, `range`, `time` или `date` (можно даже использовать их для задания верхней и нижней границы дат – например, в форме бронирования путешествия можно ограничить элемент выбора даты, позволяя выбирать только будущие даты). Для элементов ввода `range`, `min` и `max` необходимо, на самом деле, определять, какие значения возвращаются при отправке формы. Код вполне простой и не требует пояснений:

```
<input type="number" … min="1" max="10">
```

step

Атрибут `step` можно использовать с числовым значением ввода, чтобы определить точность возможных вводимых значений. Например, может потребоваться, чтобы пользователи вводили определенное время, но

только с шагом 30 минут. В этом случае можно использовать атрибут `step`, помня о том, что для элементов ввода `time` значение атрибута задается в секундах:

```
<input type="time" … step="1800">
```

Новые механизмы вывода

Кроме новых элементов управления форм, с которыми может взаимодействовать пользователь, HTML5 определяет ряд новых элементов, предназначенных специально для вывода и визуализации информации для пользователя.

<output>

Мы уже упоминали об элементе вывода `output`, когда говорили об элементе ввода `range` – этот элемент служит средством вывода "результата вычислений", или в более общем смысле для предоставления явно определенного вывода сценария (вместо простого выталкивания некоторого текста в случайный `span` или `div` с помощью `innerHTML`, например). Чтобы еще более явно указать, каким конкретно элементам управления формы соответствует `output`, можно – аналогично тому как для label – передать список ID в дополнительном атрибуте `for` элемента.

```
<input type="range" id="rangeexample" … >
<output onforminput="value=rangeexample.value" for="rangeexample"></output>
```

<progress> и <meter>

Два этих новых элемента очень похожи, оба в результате создают индикатор/панель, выводимую пользователю. Отличие их состоит в их назначении. Как предполагает название, *`progress`* предназначен для представления индикатора выполнения, чтобы указать процент завершения определенной задачи, в то время как *`meter`* является более общим индикатором скалярного измерения или дробного значения.

Панель индикатора выполнения

Проверка

Проверка формы является очень важной, как на стороне клиента, так и на стороне сервера, чтобы помочь законным пользователям избежать и исправить ошибки, и чтобы остановить злоумышленных пользователей от отправки данных, которые могут нанести вред приложению. Так как браузеры могут теперь иметь представление о том, какой тип значений ожидается из различных элементов управления форм (по их собственному типу, или каким-либо верхним/нижним границам, заданным для числовых значений, дат и времени), они могут также предложить собственную проверку формы – еще одна утомительная задача, которая, до сих пор, требовала от авторов написания большого объема кода JavaScript или использования каких-либо готовых сценариев/библиотек проверки.

Примечание: Чтобы можно было проверять элементы управления формы, они должны иметь атрибут `name`, так как без него они в любом случае не могут быть отправлены как часть формы.

required

Одним из наиболее общих аспектов проверки формы является принудительное заполнение требуемых полей – недопущение отправки формы, пока определенные элементы информации не будут введены. Это можно теперь легко реализовать, добавляя атрибут `required` к элементам `input`, `select` или `textarea`.

```
<input type="text" … required>
```

Клиентская проверка в браузере Opera в действии, показывающая ошибку для требуемого поля, которое было оставлено пустым

type и pattern

Как мы видели, авторы могут теперь определять тип записи, который они ожидают из полей формы – вместо простого определения текстового элемента ввода, авторы могут явно создавать элементы ввода для таких вещей как числа, адреса e-mail и URL. Как часть клиентской проверки, браузеры могут теперь проверять, что ввод пользователя в эти более специфические поля соответствует ожидаемой структуре – по существу, браузеры оценивают значения ввода в соответствии со встроенным шаблоном, который определяет, как выглядят допустимые значения в полях ввода этого типа, и будут предупреждать пользователя, когда его ввод не соответствует критериям.

Сообщение об ошибке браузера Opera для неверного адреса e-mail в поле ввода

Для других текстовых полей ввода, которые, тем не менее, должны следовать определенной структуре (например, формы регистрации, где имя пользователя может содержать только определенную последовательность букв нижнего регистра и цифр), авторы могут использовать атрибут `pattern` для определения своего собственного регулярного выражения.

```
<input type="text" … pattern="[a-z]{3}[0-9]{3}">
```

Стилевое оформление с помощью CSS3

Соответственно встроенной проверке форм HTML5, Модуль Базового Интерфейса Пользователя CSS3 (ссылка: http://www.w3.org/TR/css3-ui/ -

http://www.w3.org/TR/css3-ui/) определяет последовательность новых псевдо-классов, которые могут использоваться для различения требуемых полей и для динамического изменения внешнего вида элементов управления формы на основе того, были они заполнены правильно или нет:

- `:required` и `:optional` позволяют явно оформить элементы управления на основе того, имеют они или нет требуемый атрибут
- `:valid` и `:invalid` будут применять стили к элементам управления формы в зависимости от клиентской проверки
- `:in-range` и `:out-of-range` работает специально с элементами управления, которые имеют атрибут `min` и/или `max`.

В нашем примере формы мы применяем специальное оформление проверки и диапазона на элементах ввода текста, e-mail, url и чисел, только когда они имеют в текущий момент фокус, поэтому пользователи будут получать некоторую немедленную визуальную реакцию во время ввода информации в эти поля, не влияющую на общий вид формы в целом.

```
input[type=text]:focus:valid,
input[type=e-mail]:focus:valid,
input[type=number]:focus:in-range { outline: 2px #0f0 solid; }

input[type=text]:focus:invalid,
input[type=e-mail]:focus:invalid,
input[type=number]:focus:out-of-range { outline: 2px #f00 solid; }
```

Стили :valid и :invalid, применяемые динамически во время ввода адреса e-mail

Как обстоят дела для различных браузеров?

На настольном компьютере браузер Opera в настоящее время имеет наиболее полную реализацию новых типов элементов ввода и собственную клиентскую проверку, но уровень поддержки растет также во всех основных браузерах, поэтому пройдет не так много времени, и мы сможем использовать новые мощные средства в собственных проектах. Но как обстоит дело в старых браузерах?

Предполагается, что браузеры, которые не понимают новые типы элементов ввода (такие как `date` или `number`) будут просто интерпретировать их как стандартные текстовые элементы ввода – не так как их более развитые в отношении HTML5 аналоги, но, по крайней мере, они позволяют заполнить формы. Однако, как и в отношении большей части новых функций HTML5, можно предложить лучшие резервные варианты для пользователей браузеров, которые недостаточно развиты для работы с формами HTML5, выполнять базовую проверку свойств на основе JavaScript и, только в случае необходимости, загружать внешние библиотеки JavaScript, чтобы "имитировать" поддержку более сложных виджетов элементов управления форм и проверку. Таким образом можно уже программировать будущее, не отвергая существующих пользователей.

Введение в видео HTML5

В данной лекции описывается элемент <video> и некоторые связанные с ним API. Перечислены основные поддерживаемые видеоформаты. Автор рассказывает о наиболее важных способах, с помощью которых можно управлять видео через JavaScript и возможностях создания собственных элементов управления проигрывателем. Как элемент <video> сочетается с другими элементами web страницы. Оформление видео с помощью каскадных таблиц стилей (CSS3). Особое внимание уделяется совместимости новых свойств языка со старыми браузерами.

Брюс Лоусон, Патрик Х. Лауке · 11 февраля 2010 г.

Введение

Много лет тому назад в галактике, которая кажется находящейся очень далеко отсюда, мультимедиа в Web была ограничена мелодиями MIDI и анимированными изображениями GIF. По мере того как пропускающая способность сети увеличивалась, и технологии сжатия улучшались, музыка MP3 вытеснила MIDI, и стало делать успехи реальное видео. Всевозможные проприетарные плееры боролись за победу — Real Player, Windows Media Player, и т.д. — пока в 2005 г. не появился победитель: Adobe Flash. Это было обусловлено в большой степени вездесущностью его плагина и того факта, что он был выбран механизмом доставки для YouTube; Flash стал фактическим стандартом доставки видео в Web.

Одним из наиболее интересных новых свойств HTML5 является появление элемента `<video>`, который позволяет разработчикам включать видео непосредственно в свои страницы, без использования каких-либо решений на основе плагина. Компания Opera предложила стандартный элемент `<video>` как часть HTML5 в 2007 г., и он сделал свой официальный дебют в браузере Opera версии 10.50.

Эта статья знакомит с элементом `<video>` и некоторыми из связанных с ним API. Мы увидим, почему так важна собственная поддержка видео в браузерах, предоставим обзор разметки элемента, и опишем наиболее важные способы, с помощью которых можно управлять видео через

JavaScript.

Зачем нужен элемент <video>?

До сих пор, если нужно было включить в Web-страницу видео, требовалось создавать достаточно сложную разметку. Вот пример, взятый прямо с сайта YouTube:

```
<object width="425" height="344">
  <param name="movie"
    value="http://www.youtube.com/v/9sEI1AUFJKw&hl=en_GB&fs=1&"></pa
  <param name="allowFullScreen"
    value="true"></param>
  <param name="allowscriptaccess"
    value="always"></param>
  <embed src="http://www.youtube.com/v/9sEI1AUFJKw&hl=en_GB&fs=1&"
    type="application/x-shockwave-flash" allowscriptaccess="always"
    allowfullscreen="true" width="425"
    height="344"></embed>
</object>
```

Прежде всего, имеется элемент `<object>` — общий контейнер для "чужих объектов" — чтобы включить фильм Flash. Чтобы обойти несовместимость браузеров, мы включили также элемент `<embed>` как запасной контент и дубликат большинства параметров `<object>`. Получившийся код достаточно громоздкий и не очень легко читается, и имеет другие проблемы — контент плагина не очень хорошо сочетается с другими технологиями на странице и имеет неустранимые проблемы доступности (о чем будет сказано позже).

Анатомия элемента <video>

По сравнению со сложной конструкцией, необходимой для включения Flash на страницу, основная разметка необходимая для `<video>` в HTML5 удивительно проста:

```
<video src=myVideo.ogv
  width=320
```

```
    height=240
    controls
    poster=image.jpg>
  </video>
```

Отметим, что в примере мы воспользовались более свободным синтаксисом HTML5 — не требуется использовать кавычки вокруг значений атрибутов, и можно использовать простые логические атрибуты, такие как *autoplay*, в виде одиночных слов. При желании можно, конечно, использовать синтаксис XHTML `controls="controls"` и кавычки вокруг значений атрибутов. Очевидно, имеет смысл выбрать стиль кодирования, который лучше всего вам подходит, и придерживаться его для согласованности и облегчения обслуживания. В HTML5 требуется использовать синтаксис XHTML (и требуется также подавать свои страницы как код XML с правильным типом MIME, который в настоящее время не работает в Internet Explorer).

Атрибуты элемента `<video>`, использованные в примере кода, вполне понятны:

src

Источник элемента, задающий URL видео-файла.

width и height

Как и для элементов `img`, можно явно задавать размеры видео изображения — иначе элемент будет по умолчанию использовать просто собственную ширину и высоту видео файла. Если задать одно из значений, то браузер задаст размер неопределенной размерности, чтобы сохранить пропорции изображения видео.

controls

Если этот логический атрибут присутствует, браузер будет отображать свои собственные элементы управления для воспроизведения и громкости. Если он отсутствует, пользователь будет видеть только первый кадр (или определенное изображение `poster`) и не сможет воспроизвести видео, если только воспроизведение фильма не включается где-то в коде JavaScript или не создаются собственные индивидуальные элементы управления, как мы покажем позже в этой статье.

poster

Атрибут указывает на изображение, которое браузер будет использовать во время загрузки видео, или пока пользователь не начнет воспроизведение видео. Если он отсутствует, вместо него будет использоваться первый кадр видео.

Для браузеров Web, которые в данный момент не поддерживают `<video>`, можно включить альтернативный контент — по крайней мере, он мог бы содержать некоторый текст и ссылку на видео файл, чтобы пользователь мог загрузить и воспроизвести его в приложении медиаплеера:

```
<video src=myVideo.ogv
  width=320
  height=240
  controls
  poster=image.jpg>
  Download the <a href=myVideo.webm>webm</a> or <a href=myVideo.mp4>
```

Существуют и другие атрибуты, которые не были использованы в примерах. Это:

autoplay

Этот атрибут дает браузеру указание, начинать воспроизведение видео автоматически. Используйте его с осторожностью, так как это может быть крайне неприятно, если не просто проблематично, особенно для

пользователей, использующих такие вспомогательные технологии, как считыватели экрана, или имеющих недостаточно быстрые соединения (такие как мобильные устройства).

autobuffer

Если вы совершенно уверены, что пользователь захочет активировать видео (например, он специально перешел на страницу, и это единственная причина для посещения страницы), но вы не хотите использовать *autoplay*, можно задать логический атрибут `autobuffer`. Он говорит браузеру, чтобы загрузка медиа начиналась немедленно, предполагая, что пользователь будет воспроизводить видео. (Эта часть спецификации пока еще в стадии изменения; и поэтому не реализована в браузере Opera).

loop

Атрибут `loop` является еще одним логическим атрибутом. Как можно было бы догадаться, он приводит к циклическому повторению воспроизведения видео. (В настоящее время это не реализовано в браузере Opera)

Кодеки — ложка дегтя в бочке меда

В данный момент потребительские версии браузера Opera поддерживают только не требующий лицензии видео-кодек Ogg Theora, открытый формат сжатия видео некоммерческой организации Xiph.org (ссылка: http://theora.org/ - http://theora.org/).

Браузеры Firefox и Chrome также поддерживают кодек Theora. Однако браузер Safari не поддерживает, предпочитая вместо этого использовать собственную разработку для кодека H.264 (который браузер Chrome также поддерживает). Компания Microsoft также объявила о поддержке элемента `<video>`, используя при этом H.264. Как можно понять, все это создает некоторые проблемы, связанные с необходимостью представления видео в нескольких форматах для различных браузеров.

В попытке решить эту проблему компания Google недавно выпустила на безвозмездных условиях видео-кодек VP8 и формат контейнера WebM (ссылка: http://www.webmproject.org/ - http://www.webmproject.org/) с целью создать высококачественный, открытый видео-формат, доступный для различных браузеров и платформ. Это было объявлено публично на конференции 2010 Google I/O (ссылка: http://code.google.com/events/io/2010/ - http://code.google.com/events/io/2010/).

В день объявления компания Opera выпустила экспериментальную сборку, поддерживающую WebM. Эта возможность является теперь частью базовой функциональности браузера Opera 10.60 и всех будущих версий настольных браузеров (ссылка: http://www.opera.com/browser/ - http://www.opera.com/browser/).

Вернемся к нашему примеру, мы кодируем видео дважды — один раз как Theora и второй как H.264 — добавляем к видео альтернативные элементы `<source>` с подходящими атрибутами `type` и позволяем браузеру загрузить тот формат, который он может отобразить. Отметим, что в этом случае мы не используем атрибут `src` в самом элементе `<video>`:

```
<video width=320 height=240 controls poster=image.jpg>
  <source src=myVideo.webm type=video/ogg>
  <source src=myVideo.mp4 type=video/mp4>
  Download the <a href=myVideo.webm>webm</a> or <a href=myVideo.ogg>r
```

Тип файлов MP4 также может воспроизводиться плагином плеера Flash, так что это можно использовать в сочетании как запасной вариант для текущей версии Internet Explorer и более старых версий браузеров. Посмотрите отличную реализацию этой техники Кроком Кейменом в его статье Видео для всех! (ссылка: http://camendesign.com/code/video_for_everybody - http://camendesign.com/code/video_for_everybody), в которой он объединяет `object` и старый `embed` в части альтернативного контента элемента `<video>`.

Конечно, если браузеры, которые не поддерживают элемент `<video>`, возвращаются к использованию плагинов Quicktime или Flash, мы в действительности оказываемся там, где начинали, и мы не сможем

воспользоваться никакими новыми свойствами и улучшениями, которые собираемся описать ниже. Какой же тогда в этом смысл? Необходимо сказать, что это переходное решение, пока собственная поддержка видео не будет реализована во всех основных браузерах, и форматы не будут согласованы. Это случай постепенного ухудшения – пользователи могут получить слегка урезанную версию вашей страницы, но, по крайней мере, они смогут увидеть ваши фильмы.

В Web больше нет граждан второго сорта

Мы видели, что разметка для нового элемента HTML5 `<video>` является на порядок величины более легко читаемой и понятной по сравнению с тем, что необходимо делать в настоящее время, чтобы добавить фильмы Flash в разметку. Но, несмотря на устрашающий вид старого способа кодирования, в большинстве случаев он работает, не правда ли? Почему же мы должны отказываться от этого подхода, использующего отображение видео с помощью внешнего плагина, такого как Flash?

Основное преимущество элемента HTML5 `<video>` состоит в том, что, наконец, видео является полноценным содержанием Web, а не отодвигается вглубь элемента `object` или непроверяемого элемента `embed` (хотя это теперь проверяется в HTML5). Давайте рассмотрим некоторые реальные преимущества.

Готовая доступность с клавиатуры

Одной из серьезных неразрешенных проблем использования Flash является доступность с клавиатуры. За исключением Internet Explorer в Windows, обычно не существует для пользователя клавиатуры способа как-то переместить свой фокус в фильм Flash. И даже если эти пользователи как-то смогут поместить свой фокус в фильм (используя дополнительные вспомогательные технологии), не существует простого способа переместить фокус оттуда в другое место (если только в фильм не был добавлен специальный код ActionScript для программного возврата фокуса браузера из плагина на страницу). В противоположность этому, так как браузер непосредственно отвечает за

элемент <video>, он может использовать элементы управления фильма, как если бы они были обычными кнопками на Web-странице, и включать их в свою обычную последовательность перемещения фокуса.

Поддержка клавиатуры для встроенного видео в настоящее время реализована не во всех браузерах … однако она уже прекрасно работает в браузере Opera 10.50+.

Элемент <video> хорошо сочетается с остальной страницей

Говоря коротко, когда на странице используется плагин, то для изображения резервируется определенная область, которую браузер передает плагину. Что касается браузера, то для него область плагина остается черным ящиком — браузер не обрабатывает и не интерпретирует ничего из того, что там происходит.

Обычно это не является проблемой, но могут возникать проблемы, когда компоновка перекрывает область рисования плагина. Представьте, например, сайт, который содержит видео Flash, но имеет также раскрывающиеся меню на основе JavaScript или CSS, которое необходимо развернуть поверх видео. По умолчанию область рисования плагина расположена поверх Web-страницы, т.е. эти меню будут странным образом появляться позади видео. Аналогичный уродливый результат возникает в случае, когда страница использует `lightbox` — любое видео Flash будет по-прежнему появляться плавающим поверх затененного представления страницы. Именно поэтому большинство готовых сценариев `lightbox` стремятся решить эту проблему, удаляя сначала со страницы все объекты плагинов, прежде чем выполнять затенение, и возвращая их, когда затенение прекращается.

В конкретном случае плагина Flash, разработчики могут исправить эту проблему отображения, добавляя атрибут `wmode='opaque'` в свой элемент `<object>` и эквивалентную конструкцию `<param name='wmode' value='opaque'>` в свой элемент `<embed>`. Однако, это приводит также к тому, что плагин становится полностью недоступным для пользователей со считывателями экрана, и поэтому

лучше этого не делать.

Проблемы и странное поведение могут возникать также, если страница испытывает динамическое изменение компоновки. Если размеры области рисования плагина изменяются, то это может иметь иногда непредвиденные последствия — видео, воспроизводимое в плагине, может не изменить размер, но может быть вместо этого просто обрезано, или будет выводиться дополнительное пустое пространство.

В случае собственного элемента `<video>` сам браузер заботится об отображении. Поэтому элемент `<video>` ведет себя также как и любой другой элемент компоновки страницы. Его можно позиционировать, сделать плавающим, перекрывающимся или изменяющим динамически размеры, без каких-либо дополнительных средств. Можно даже получить интересные эффекты, такие как полупрозрачное видео, задавая просто плотность элемента с помощью CSS. Совершенно новый мир удивительного видео ожидает нас.

Оформление <video> с помощью CSS

Теперь видео является частью множества технологий Открытой Web, поэтому мы можем использовать CSS для надежного стилевого оформления элемента.

(Прочтите наше руководство по переходам CSS3 и 2D-преобразованиям (ссылка: http://dev.opera.com/articles/view/css3-transitions-and-2d-transforms/ - http://dev.opera.com/articles/view/css3-transitions-and-2d-transforms/)).

Объединение <video> и <canvas>

Когда браузер реализует размещение и воспроизведение видео, мы легко можем размещать и объединять другие элементы поверх него. В данном примере `<canvas>` накладывается поверх видео.

Отметим, что `<canvas>` не полностью покрывает видео. Мы сделали холст на 40 пикселей короче, чем высота видео, чтобы область видео, где появляются элементы управления, не закрывалась. Это гарантирует, что, если пользователь переместит мышь в нижнюю часть видео, будет

достаточно области элемента `<video>`, выглядывающего из-под холста, чтобы получить сообщение события зависания курсора, и заставить его отобразить элементы управления. Доступ с клавиатуры к элементам управления должен работать независимо от закрытия элементов, однако поддержка клавиатуры в настоящее время варьируется в различных браузерах.

Создание собственных элементов управления

Элементы `<video>` и `<audio>` (который будет рассмотрен в будущей статье) являются экземплярами новых элементов медиа из HTML5 DOM (ссылка: http://www.w3.org/TR/html5/video.html#media-elements - http://www.w3.org/TR/html5/video.html#media-elements), которые предоставляют мощный API, позволяющий разработчикам управлять воспроизведением видео с помощью целого множества новых методов JavaScript и свойств. Давайте рассмотрим некоторые из наиболее подходящих для создания простого индивидуального средства управления:

play() и pause()

Вполне очевидно, что эти методы начинают воспроизведение видео и делают паузу. Метод `play()` будет всегда начинать воспроизведение видео с текущей позиции воспроизведения. Когда видео загружается впервые, это будет первый кадр видео. Отметим, что не существует метода `stop()` — если вы хотите остановить воспроизведение и "перемотать" в начало видео, то нужно будет сделать паузу с помощью `pause()` и программным путем самостоятельно изменить текущую позицию воспроизведения.

volume

Этот атрибут можно использовать для считывания или задания громкости аудио дорожки видео, может принимать значения типа `float` в диапазоне от 0.0 (тишина) до 1.0 (самый громкий).

muted

Независимо от `volume`, звук в видео можно заглушить.

currentTime

При чтении этот атрибут возвращает текущую позицию воспроизведения в секундах, также выраженную как `float`. Задание этого атрибута будет — если возможно — перемещать позицию воспроизведения в указанную позицию времени. (Отметим, что задание `currentTime`, или поиск, в настоящее время не реализовано в браузере Opera.)

Кроме того, элементы медиа также порождают ряд событий как часть своей модели обработки, которые можно отслеживать и перехватывать. Для нашего примера мы рассмотрим только несколько из них:

loadeddata

Браузер загрузил достаточно видео данных, чтобы начать воспроизведение в текущей позиции.

play и pause

Воспроизведение было начато или остановлено на паузу. Если мы управляем видео из JavaScript, мы можем следить за этими значениями, чтобы убедиться, что вызов метода `play()` или `pause()` завершился успешно.

timeupdate

Текущая позиция воспроизведения изменилась, потому что видео воспроизводится, сценарий изменил его программным путем, или

пользователь решил переместиться в другую позицию видео.

ended

Мы достигли конца видео, и элемент `<video>` не задан для циклического воспроизведения или обратного воспроизведения (не рассматривается в этой статье).

Теперь мы имеем все основные строительные блоки, необходимые для создания простого средства управления. Вначале сделаем предостережение: если мы создаем свой собственный элемент управления на основе JavaScript, мы, очевидно, хотим исключить все собственные элементы управления браузера. Однако мы можем захотеть предоставить эти элементы управления как запасной вариант, в том случае, если пользователи отключают JavaScript в своем браузере. В связи с этим мы сохраним атрибут `controls` в нашей разметке, и программным путем удалим его во время выполнения сценария. Альтернативно мы могли бы также задать значение атрибута как `false` — оба подхода будут допустимы. Так как наш индивидуальный элемент управления сам опирается для работы на сценарий, мы будем создавать разметку самого элемента управления с помощью JavaScript.

Посмотрите пример со сценарием собственных элементов управления видео в HTML5 в действии. Сценарий достаточно многословен, и требует небольшой чистки перед использованием в производственной среде, но будем надеяться, что он помог проиллюстрировать некоторые новые возможности, которые открывает видео в HTML5. Разработчики Web с небольшими знаниями JavaScript теперь легко могут создавать специальные элементы управления видео, которые отлично дополняют их сайты, не требуя для этого создания заказных видео-плееров Flash.

Дополнительное чтение

1. Все, что нужно знать об аудио и видео в HTML5 (ссылка: http://my.opera.com/core/blog/2010/03/03/everything-you-need-to-know-about-html5-video-and-audio-2 - http://my.opera.com/core/blog/2010/03/03/everything-you-need-to-know-

about-html5-video-and-audio-2) – и это действительно все!

2. Спецификация элемента `<video>` (ссылка: http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#video - http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#video)
3. Как элемент `<video>` реализован в браузере Opera (ссылка: http://my.opera.com/core/blog/2009/12/31/re-introducing-video - http://my.opera.com/core/blog/2009/12/31/re-introducing-video)
4. Доступное видео в HTML5, использующее титры на основе JavaScript (ссылка: http://dev.opera.com/articles/view/accessible-html5-video-with-javascripted-captions/ - http://dev.opera.com/articles/view/accessible-html5-video-with-javascripted-captions/)

Улучшение доступности видео-плеера HTML5

Лекция представляет собой описание проекта создания индивидуального видеоплеера HTML5 с улучшенной доступностью. На примерах рассматривается разработка элементов управления плеером: кнопок и ползунков. Создание титров и стенограмм. Описывается проблемы, связанные с управлением плеером посредством клавиатуры. Использование JavaScript-библиотеки jQuery для программирования индивидуальных особенностей видеоплеера. Приводится подборка полезных ресурсов в сети интернет.

Кристиан И. Колсериу · 11 ноября 2010 г.

Введение

В моей последней статье я реализовал настраиваемый, совместимый с различными браузерами видео-плеер, использующий элемент `<video>` из HTML5 (ссылка: http://dev.opera.com/articles/view/custom-html5-video-player-with-css3-and-jquery/ - http://dev.opera.com/articles/view/custom-html5-video-player-with-css3-and-jquery/). Он является хорошим решеним по многим причинам, включая доступность – элемент `<video>` в HTML5 значительно более доступен, чем альтернативные решения на основе плагинов, например, с точки зрения готовой клавиатурной доступности, и легкости настройки, не требуя при этом дорогостоящего IDE.

Но на этом работа не заканчивается. Построенное до сих пор решение, как и другие виджеты на основе JavaScript, по-прежнему имеет ряд проблем доступности с точки зрения семантики и возможности обнаружения, что можно было бы рассмотреть с помощью спецификации WAI-ARIA (ссылка: http://www.w3.org/TR/wai-aria/ - http://www.w3.org/TR/wai-aria/) Инициативы W3C по доступности Web.

В этой статье я покажу, как решать такие проблемы с помощью WAI-ARIA, и будут добавлены еще некоторые усовершенствования в плеер, такие как титры.

Мультимедиа

До появления в HTML5 собственных элементов `<video>` и `<audio>`, все мультимедийные возможности, применяемые в Web, полагались в браузерах на плагины сторонних разработчиков. Эти плагины создают множество проблем для доступности, от трудностей с клавиатурной навигацией до отсутствия непосредственной доступности для вспомогательных технологий.

Решения для этих проблем постоянно развиваются создателями и разработчиками плагинов, но они не являются стандартными, и не очень широко распространены.

Текущее состояние собственных элементов управления браузера

Все последние версии популярных Web-браузеров поддерживают элементы `<audio>` и `<video>`, и каждый браузер предоставляет набор кнопок управления для этих элементов. Проблема в том, что не все браузеры предоставляют клавиатурный доступ к этим элементам управления:

- В сентябре 2010, Opera 10.6 являлся, видимо, единственным браузером, который предоставлял полный клавиатурный доступ ко всем отдельным элементам управления плеера (см. Рисунок 5.1).
- Firefox 3.6.10 также предоставляет полную поддержку для управления поведением плеера, но пользователь, по сути, не может получить доступ к отдельным элементам управления через клавиатуру. Вместо этого, пользователи могут задать фокус на элементе, затем запустить событие воспроизведения с помощью клавиши пробела, искать с помощью клавиш правой и левой стрелок, и изменять уровень громкости с помощью клавиш со стрелками вверх и вниз.
- Браузер Internet Explorer 9 beta использует систему почти идентичную той, которая применяется в Firefox 3.6.10.
- Safari 5 и Chrome 6 не могут получать доступ к плееру через клавиатуру.

Рис. 5.1. Собственные элементы управления видео в браузере Opera 10.63, с фокусом на кнопке громкости

Индивидуальный видеоплеер HTML5 с улучшенной доступностью

Решением этих проблем является создание своего собственного плеера мультимедиа, и отображение доступных с клавиатуры элементов управления, используя API элементов медиа (ссылка: http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#media-elements - http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#media-elements).

В этой статье мы сосредоточимся на адаптации специального видео-плеера из моей последней статьи с помощью jQuery и CSS3, делая его насколько возможно доступным, и добавляя попутно новые свойства, такие как скрытые титры и стенограммы.

Прогрессивное улучшение

Наш плагин jQuery создан с учетом прогрессивного улучшения — именно поэтому он создан поверх стандартного элемента видео. В этом

случае, если JavaScript отключен, пользователь получит стандартные элементы управления, предоставляемые браузером. Это должно обеспечить базовый уровень доступности, даже если сейчас не все браузеры предоставляют это "свойство", а некоторые даже не показывают их, когда JavaScript отключен.

Элементы управления плеера

Исходная разметка для элементов управления выглядит следующим образом:

```
<div class="ghinda-video-controls">
  <a class="ghinda-video-play" title="Play/Pause"></a>
  <div class="ghinda-video-seek"></div>
  <div class="ghinda-video-timer">00:00</div>
  <div class="ghinda-volume-box">
    <div class="ghinda-volume-slider"></div>
    <a class="ghinda-volume-button" title="Mute/Unmute"></a>
  </div>
</div>
```

Первая версия была по большей части проверкой концепции, и разметка, конечно, может быть улучшена в смысле семантики и доступности. Давайте перепишем это сейчас, используя более содержательные элементы для каждого элемента управления:

```
<div class="acorn-video-controls">
  <button class="acorn-play-button" title="Start the playback" aria-controls="vide

  <input type="range" class="acorn-seek-slider" title="Video seek control" value=
   max="150" step="0.1" aria-controls="video1"/>

  <span class="acorn-timer">00:00</span>

  <div class="acorn-volume-box">
    <button class="acorn-volume-button" title="Mute volume" aria-controls="vide
    <input type="range" class="acorn-volume-slider" title="" value="1" min="0"
      max="1" step="0.05" aria-controls="video1"/>
```

```
    </div>
  </div>
```

Прежде всего, мы заменили элементы `<a>`, которые ведут себя как кнопки, на реальные элементы `<button>`, так что разметка показывает их смысл, а считыватели экрана интерпретируют их правильно.

Вместо бессодержательных `<div>` для ползунков, мы используем собственные ползунки HTML5: `<input type="range">`.

Можно также видеть новый атрибут, кроме обычно используемых. Атрибут `aria-controls` является частью спецификации WAI-ARIA, и определяет, какой элемент контролируется. Можно определить ID элемента или список ID. Наше значение в данный момент `video1`, но в реальном продукте мы бы использовали ID для видео, или сгенерировали какое-то уникальное значение, если значение не было бы предоставлено. Основная разметка выглядит как на Рисунке 5.2.

Рис. 5.2. Элементы управления индивидуального видео плеера с новой разметкой

Если вы не знакомы с WAI-ARIA, то я настоятельно рекомендую

прочитать статью Введение в WAI ARIA (http://dev.opera.com/articles/view/introduction-to-wai-aria/) на сайте Dev.Opera (ссылка: http://dev.opera.com/ - http://dev.opera.com/).

Кнопки

Все наши кнопки имеют содержательные метки и атрибуты `title`. Атрибуты `title` предоставляют также всплывающие подсказки, которые будут особенно полезны, если разработчик решит показывать только кнопки, без меток. Они также полезны для пользователей считывателей экрана, предоставляя им более длинное объяснение, что реально делает элемент управления.

В предыдущей разметке можно увидеть, что метка для кнопки воспроизведения Play была "Play/Pause", а метка для кнопки приглушения Mute была "Mute/Unmute", но в новой версии мы помечаем их просто "Play" и "Mute". Это связано с тем, что мы теперь используем JavaScript для изменения метки на каждой кнопке после ее нажатия. Поэтому после нажатия на кнопке Play, метка на ней станет "Pause".

Мы также добавляем различные классы в каждую кнопку после нажатия.

Ползунки

Для ползунков поиска и громкости мы использовали тип элемента формы HTML5 `<input type="range">`. К сожалению, все не так просто, как мы надеялись:

- Opera является единственным браузером, который полностью поддерживает элемент, с клавиатурной доступностью для горизонтальных и вертикальных ползунков.
- Браузер Chrome поддерживает его с клавиатурной доступностью, но не поддерживает вертикальные ползунки. Браузер Safari ведет себя почти также, но без клавиатурной доступности.
- Браузеры Firefox (версия 3.6 и Minefield 4.0b7) и Internet Explorer 9

Beta не поддерживают ползунки вообще, а используют обычные текстовые элементы ввода.

В связи с этими проблемами мы будем по-прежнему использовать jQuery UI для создания ползунков, но, тем не менее, предоставляем собственные ползунки в качестве варианта для плагина, на тот случай, когда они будут правильно реализованы во всех браузерах.

В предыдущей версии плагина мы вызывали плагин ползунка UI на пустом `<div>`, но теперь мы собираемся вызывать его на элементе `<input>`. У нас теперь имеется проблема, так как плагин добавляет классы к ползунку, и присоединяет к нашему элементу другие элементы, такие как регулятор ползунка. Это не будет работать, так как `<input>` является линейным (inline), поэтому не может включать другие элементы.

Поэтому перед вызовом плагина ползунка мы должны удалить `<input>`, заменить его элементом `<div>` (или `<p>` или любым другим обычным элементом HTML) и вызвать на нем плагин.

WAI-ARIA

При переписывании исходной разметки можно было бы сохранить исходные элементы и добавить к ним роли ARIA, например, добавить `role="button"` к элементам `<a>`. Для пользователей вспомогательных технологий это могло бы заставить их вести себя почти как элементы `<button>`. Однако при таком подходе возникает ряд проблем.

Проблемой могут быть семантические конфликты. Элемент `<div>`, например, является семантически нейтральным и может получить любую роль, но другие элементы имеют свою собственную внутреннюю семантику, и добавление к ним случайных ролей может приводить к конфликту между ролью ARIA, пытающейся описать элемент, и его врожденной семантикой.

В связи с этими конфликтами была создана концепция "неявной семантики ARIA". Таким способом спецификация описывает

примененную семантику, и ограничение внутренней семантики для большинства элементов. Например, элементы `<article>`, `<aside>` и `<section>` имеют следующую примененную семантику: "article role", "note role" и "region role".

Использование элементов с собственной ролью неизмеримо лучше, чем его "подделка", если такой элемент доступен. Об этом можно дополнительно почитать в спецификации HTML5 в разделе, который связан с WAI-ARIA (Примечания для продуктов вспомогательных технологий - ссылка: http://dev.w3.org/html5/spec/Overview.html#annotations-for-assistive-technology-products-aria - http://dev.w3.org/html5/spec/Overview.html#annotations-for-assistive-technology-products-aria).

Ползунки ARIA

Возвращаясь к плееру, мы теперь разрешили проблемы с кнопками, но, принимая во внимание, что мы не всегда используем собственные ползунки браузера, нам по-прежнему требуется кое-что сделать с ползунками jQuery UI.

Ползунки UI предоставляют отличную клавиатурную доступность, но не имеют никакой поддержки ARIA (в версии 1.8.4). И, считая, что ползунок является элементом `<a>`, указывающим на `href="#"` в том же документе, считыватели экрана будут интерпретировать это как посещенную ссылку, а не как управляющий ползунок. Чтобы обойти эту проблему, мы создадим небольшую функцию, чтобы добавить в ползунок ARIA и возможность получать фокус.

Чтобы элемент интерпретировался вспомогательными технологиями как ползунок, необходимо использовать на элементе роль *`slider`*. Но ползунок состоит из двух элементов: регулятора и направляющей. ARIA требует соединения с атрибутом только одного элемента, поэтому надо выбрать, какой использовать.

С учетом того, что только один элемент (для ползунка) должен получить фокус, и UI jQuery предоставляет клавиатурное управление для ползунка, когда фокус находится на регуляторе, то это будет лучшим

выбором.

Все элементы формы должны опознаваться пользователями, поэтому нужно пометить ползунок. Для этого можно использовать различные подходы. Можно было бы использовать атрибут `aria-labelledby`, использовать элемент `<label>` и заставить его указывать на ползунок, или использовать атрибут `title`. Для нашего случая атрибут `title` является, вероятно, лучшим решением.

Кроме атрибута `role`, ползунок ARIA требует следующие атрибуты:

aria-valuenow

Текущее значение ползунка.

aria-value-min

Минимальное значение.

aria-value-max

Максимальное значение, которое может иметь ползунок.

По умолчанию вспомогательные технологии используют атрибут `aria-valuenow`, но значение должно быть числом – если бы мы хотели предоставить пользователю более удобное для восприятия значение, мы могли бы использовать вместо этого другое свойство ARIA, `aria-valuetext`. Это значение является строкой, поэтому, например, когда пользователь использует ползунок поиска, то вместо вывода значения "23", можно было бы выводить "23 секунды".

С учетом всего здесь рассмотренного, наша функция выглядит следующим образом:

```
var initSliderAccess = function (elem, opts) {
  var accessDefaults = {
   'role': 'slider',
   'aria-valuenow': parseInt(opts.value),
   'aria-valuemin': parseInt(opts.min),
   'aria-valuemax': parseInt(opts.max),
```

```
    'aria-valuetext': opts.valuetext,
    'tabindex': '0'
   };
   elem.attr(accessDefaults);
 };
```

Эта функция адаптирована для тех средств, которые используются при вызове ползунка UI jQuery. Мы имеем в функции два объекта в качестве параметров, первый является элементом, с которым выполняется манипуляция, а второй является объектом свойств, которые используются для значений.

Можно видеть, что кроме атрибутов `role` и `aria`, мы добавляем атрибут `tabindex` . Это делается, чтобы гарантировать, что регулятор будет доступен с клавиатуры. Мы используем также функцию `parseInt()` на некоторых значениях, чтобы не передавать в систему вспомогательных технологий такие значения, как "1.54321".

В предыдущей статье, чтобы фактически заставить ползунки управлять видеоплеером, мы присоединили функции к порождаемым jQuery событиям *`slide`* и `stop`.

И так как мы всегда хотим информировать пользователя о текущем значении ползунка, мы должны изменить значение `"aria-valuenow"` и `"aria-valuetext"` на *`slide`*`()`.

Мы добавим небольшой код в функции. Для ползунка поиска мы будем использовать следующий код:

```
sliderUI.attr("aria-valuenow", parseInt(ui.value));
sliderUI.attr("aria-valuetext", ariaTimeFormat(ui.value));
```

Объект `sliderUI` является элементом регулятора ползунка, а `ui.value` является текущим значением ползунка, возвращаемым плагином. Можно видеть, что мы используем для атрибута `aria-valuetext` другую функцию, функцию `ariaTimeFormat()`.

Вот как выглядит функция `ariaTimeFormat()`:

```
var ariaTimeFormat = function(sec) {
```

```
    var m = Math.floor(sec/60)<10?"" + Math.floor(sec/60):Math.floor(sec/60);
    var s = Math.floor(sec-(m*60))<10?"" + Math.floor(sec-(m*60)):Math.floor(se
    var formatedTime;

    var min = 'minutes';
    var sec = 'seconds';

    if(m==1) min = 'minute';
    if(s==1) sec = 'second';

    if (m!=0) {
      formatedTime = m + ' ' + min + ' ' + s + ' ' + sec;
    } else {
      formatedTime = s + ' ' + sec;
    };

    return formatedTime;
  };
```

Эта функция получает текущее значение ползунка (число секунд, где расположен регулятор) и преобразует его в понятную человеку строку. Если ползунок расположен на 72 (секунде), например, функция вернет "1 minute 12 seconds".

Это особенно полезно для длинных видеофильмов, где без такой функции смещение указателя поиска на 25 минут будет сообщаться как 1500 секунд, делая это достаточно неудобным, особенно для пользователей считывателей экрана.

Мы будем использовать ту же самую функцию `ARIAfication` на ползунке громкости, чтобы сделать его доступным, и модифицируем функцию изменения громкости, чтобы изменять значение `aria-valuenow` и `aria-valuetext`, когда ползунок перемещается:

```
  video.$volume.$handle.attr("aria-valuenow", Math.round(volume*100));
  video.$volume.$handle.attr("aria-valuetext", Math.round(volume*100) + ' percer
```

Можно видеть, что мы умножаем наше значение на 100. Это связано с тем, что значение громкости находится между 0 и 1, а не между 0 и 100.

Также больше смысла имеет предоставление пользователю "процентного" значения для громкости, а не однозначное число, именно поэтому мы добавляем символ процентов к `aria-valuetext`, а также умножаем и округляем значение.

Титры и стенограммы

Как крайне важное свойство доступности, каждый плеер видео/мультимедиа должен предоставлять поддержку для титров и/или стенограмм. К сожалению, текущая спецификация W3C HTML5 не содержит ничего с этим связанного. С другой стороны спецификация WHATWG HTML5 (да, существует две спецификации HTML5) недавно добавила элемент `<track>` для титров. Это "позволяет авторам определить явные внешние размеченные по времени дорожки для элементов медиа ". Поэтому, по существу, вы можете определить внешний файл, содержащий титры, подзаголовки, описания или другие размеченные по времени дорожки. Можно определить несколько элементов `<track>` для различных дорожек, таких как различные языки.

Текущий формат файла называется WebSRT (ссылка: http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#websrt - http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#websrt), и является, по сути, улучшенной версией формата SRT для SubRip.

Можно прочитать больше об этом в спецификации WHATWG HTML5 (ссылка: http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#the-track-element - http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#the-track-element).

Этот элемент отсутствует в спецификации W3C, в связи с "политическими" проблемами, в основном, потому что текущий предложенный формат (WebSRT) конфликтует с примерно 50 другими форматами, включая форматы W3C, такие как smilText и TTML.

Одной из основных проблем является то, что он не реализован сейчас ни в одном браузере, но не бойтесь – мы собираемся реализовать

элемент самостоятельно, используя JavaScript. Это лучший защищенный от будущих изменений способ использовать титры сейчас. Когда однажды проблемы разрешатся, и браузеры реализуют элемент, нам не понадобиться изменять существующую разметку для использования собственных титров.

Техника, которую мы собираемся использовать, является в какой-то степени обратной версией техники создания титров Брюса Лоусона. Если вы не знакомы с ней, прочтите статью Улучшение доступности видео в HTML5 с помощью титров на основе JavaScript (ссылка: http://dev.opera.com/articles/view/accessible-html5-video-with-javascripted-captions/ - http://dev.opera.com/articles/view/accessible-html5-video-with-javascripted-captions/) на сайте Dev Opera.

Описанная в этой статье техника описывает использование разметки HTML для определения титров, использование специальных атрибутов `data` для задания смещения времени для каждого титра. Затем выполняется синтаксический анализ элементов и создается объект JavaScript, который используется для отображения каждого титра в правильное время. Она использует также генерируемый с помощью CSS контент для вставки отметок времени в контент.

Мы собираемся изменить технику на прямо противоположную, и интерпретировать элементы `<track>` таким же образом, как может браузер.

Прежде всего, нам потребуется интерпретатор для файлов, так как мы собираемся использовать анализатор SRT Сильвии Пфайффер (ссылка: http://silvia-pfeiffer.de/ - http://silvia-pfeiffer.de/) – посмотрите его обсуждение в следующей статье (ссылка: http://blog.gingertech.net/2009/07/29/first-experiments-with-itext/ - http://blog.gingertech.net/2009/07/29/first-experiments-with-itext/), и демонстрационный пример анализатора SRT (http://www.annodex.net/~silvia/itext/).

Теперь в функции инициирования титров мы собираемся найти элементы `<track>`. Если мы найдем более одного элемента, мы сгенерируем разметку и позволим пользователю выбирать титры из списка. Спецификация включает атрибут `label` для элемента `<track>`, определяя его как "читаемый пользователем заголовок

дорожки", который "используется агентами пользователя при перечислении дорожек подзаголовков, титров, и аудио описаний в своем пользовательском интерфейсе". Поэтому мы собираемся использовать этот атрибут в своем UI.

```
<ul>
  <li>
    <label>
      <input type="radio" name="acornCaptions" checked="checked" />
      None
    </label>
  </li>
  <li>
    <label>
      <input type="radio" name="acornCaptions" />
      English <!-- это атрибут "label" элемента <track> -->
    </label>
  </li>
  <li>
    <label>
      <input type="radio" name="acornCaptions" />
      Romani <!-- это атрибут "label" элемента <track> -->
    </label>
  </li>
</ul>
```

Мы используем неупорядоченный список с кнопками `<input type="radio">`.

Мы помещаем элементы `<input>` и текст в элементы `<label>`, чтобы вспомогательные технологии ассоциировали метку с кнопкой, не определяя `<label>` отдельно и присваивая ему уникальный ID и кнопку.

Реальный текст метки является контентом атрибута `label` элемента `<track>`.

Когда пользователь выбирает титры, мы загружаем их с помощью вызова Ajax, анализируем, создаем с каждым титром привязанный к

времени объект, и генерируем также стенограмму.

Это звучит сложнее, чем является в действительности.

```
$.ajax({
  url: url,
  success: function(data) {
    // используем анализатор SRT на загруженных данных
    captions = parseSrt(data);

  // находим управляющую кнопку стенограммы и отображаем ее

    video.$transcript = video.$container.next('.acorn-transcript');
    video.$transcriptBtn.show();

    // генерируем разметку для стенограммы и добавляем ее

    var transcriptText = '';
    $(captions).each(function() {
      transcriptText += '' + this.content.replace('"','"') + '';
    });
    video.$transcript.html(transcriptText);

    captionsActive = true;
    video.$captions.show();

    // в случае паузы видео и
// при отключенном timeUpdate,
// мы будем updateCaption (обновлять титры) вручную

if(video.$self.attr('paused')) updateCaption();

    video.$captionsBtn.addClass('acorn-captions-active').removeClass('acorn-ca
  },
  error: function() {

    // в случае ошибки при загрузке титров
// не отображайте ничего и покажите сообщение
// об ошибке, если присутствует консоль
```

```
    captionsActive = false;
    captions = '';
    video.$transcriptBtn.hide();
    video.$captionsBtn.removeClass('acorn-captions-active').removeClass('acorn

    if(console) console.log('Error loading captions');
  }
});
```

Код достаточно просто понять, поэтому я собираюсь сосредоточиться больше на стенограммах. Можно видеть, что мы генерируем разметку и используем функцию jQuery `html()`, чтобы добавить их в контейнер стенограмм. Мы используем такую же разметку, как и техника Брюса Лоусона, но теперь для стенограммы, а не для титров.

Таким образом, мы генерируем стенограмму на основе титров, и можем иметь столько версий стенограммы, сколько имеется версий титров.

Клавиши доступа?

Большинство доступных плееров мультимедиа и RIA реализуют некоторую форму клавиш доступа, либо с помощью стандартного атрибута `accesskey`, либо присваивая сложные комбинации клавиш с помощью JavaScript. Хотя это может показаться отличным решением для большинства разработчиков, обследования и примеры использования говорят об обратном.

Это "свойство", нацеленное на улучшение доступа к страницам и приложениям, оказались плохо спроектированными и реализованными, создавая путаницу у пользователей вспомогательных технологий, а не помогая им.

Именно поэтому я предпочел вообще не реализовывать клавиши доступа, и сделать перемещение по элементам управления плеера с помощью стандартной навигации с помощью клавиши TAB.

Окончательная отделка

Наш плагин берет стандартный элемент `<video>` из HTML5 и создает для него доступные элементы управления, поддержку титров и стенограммы, и другие средства, но он не предоставляет никакой поддержки для описания видео. Это необходимо сделать без использования плагина.

Я предлагаю использовать элемент `<figure>` из HTML5 вместе с `<figcaption>`, и использовать атрибут `aria-describedby` для соединения элемента `<video>` с описанием. В этом случае, когда, например, считыватель экрана достигнет видео, он получит также его описание.

```
<figure>
  <video controls="controls" width="300" height="200" preload="metadata" aria-
    <source src="/path/to/video.webm" />
    <track src="/path/to/caption.srt" kind="captions" srclang="rom" label="Roman
  </video>
  <figcaption id="videodescription">
    Трейлер короткого анимационного фильма "Sintel", проекта Durian Ope
    Дополнительная информация на сайте http://durian.blender.org. Это опи
  </figcaption>
</figure>
```

Запасной вариант

Также как и в предыдущей статье о видео плеере, я не собираюсь создавать механизм запасного варианта, так как каждый метод запасного варианта имеет свои собственные проблемы доступности, будет ли это Flash, Java, Silverlight или что-то другое. Разработчик должен определить для себя лучший подход.

Заключение

Посмотрите демонстрационный пример видео плеера HTML5 с повышенной доступностью (ссылка: http://dev.opera.com/articles/view/more-accessible-html5-video-player/demos.html - http://dev.opera.com/articles/view/more-accessible-html5-video-player/demos.html), дополненный всем, что было описано в этой статье.

Описанные в этих статьях технические приемы для элемента `<video>` из HTML5 привели к созданию готового к реальному использованию плагина jQuery. Самую последнюю версию и дальнейшие разработки можно посмотреть в репозитории Github плеера Acorn Media (ссылка: http://github.com/ghinda/acornmediaplayer - http://github.com/ghinda/acornmediaplayer).

По мере того как HTML5 переносится на все большее количество платформ и устройств, крайне важно, чтобы разработчики и создатели учитывали проблемы доступности во время реализации новых свойств, а не впоследствии, после их создания.

Дальнейшее чтение

Дополнительная информация об элементе <video>

1. Все необходимое о видео и аудио в HTML5 (ссылка: http://my.opera.com/core/blog/2010/03/03/everything-you-need-to-know-about-html5-video-and-audio-2 - http://my.opera.com/core/blog/2010/03/03/everything-you-need-to-know-about-html5-video-and-audio-2)
2. Улучшение доступности видео в HTML5 с помощью титров на основе JavaScript (ссылка: http://dev.opera.com/articles/view/accessible-html5-video-with-javascripted-captions/ - http://dev.opera.com/articles/view/accessible-html5-video-with-javascripted-captions/)

Учебные материалы компании Opera по доступности

1. Основы доступности (ссылка: http://dev.opera.com/articles/view/25-accessibility-basics/ - http://dev.opera.com/articles/view/25-accessibility-basics/)
2. Тестирование доступности (ссылка: http://dev.opera.com/articles/view/26-accessibility-testing/ - http://dev.opera.com/articles/view/26-accessibility-testing/)

Ресурсы

1. Доступность в HTML5 (ссылка: http://html5accessibility.com/ - http://html5accessibility.com/)
2. Ресурсы ARIA (ссылка: http://wiki.codetalks.org/wiki/index.php/ARIA_Resources - http://wiki.codetalks.org/wiki/index.php/ARIA_Resources)
3. Семантика в HTML 5 (http://www.alistapart.com/articles/semanticsinhtml5)
4. Встроенные роли доступности в HTML5 (ссылка: http://hsivonen.iki.fi/html5-roles/ - http://hsivonen.iki.fi/html5-roles/)
5. Самодельные элементы управления для улучшения доступа к UI с помощью ARIA и HTML (ссылка: http://www.w3.org/2010/Talks/www2010-dsr-diy-aria/#(1) - http://www.w3.org/2010/Talks/www2010-dsr-diy-aria/#(1))
6. Террилл Томпсон: Создание собственного медиа плеера HTML5 с улучшенной доступностью (ссылка: http://terrillthompson.blogspot.com/2010/08/creating-your-own-accessible-html5.html - http://terrillthompson.blogspot.com/2010/08/creating-your-own-accessible-html5.html)
7. Результаты исследования пользователей считывателей экрана (ссылка: http://webaim.org/projects/screenreadersurvey2/ - http://webaim.org/projects/screenreadersurvey2/)
8. HTML5, роли ARIA, и считыватели экрана в мае 2010 (ссылка: http://www.accessibleculture.org/research/html5-aria/ - http://www.accessibleculture.org/research/html5-aria/)
9. Wikipedia о клавишах доступа (ссылка: https://secure.wikimedia.org/wikipedia/en/wiki/Access_key - https://secure.wikimedia.org/wikipedia/en/wiki/Access_key)

Холст HTML5 – основы

Основы использования элемента "холст". Из данной лекции вы узнаете, как создавать на холсте графические примитивы в виде прямоугольных и треугольных областей со свойствами заливки и обводки. Даны особенности рисования линий и штрихов. Рисование фигур с помощью путей. Вставка в холст других изображений, сформированных элементами img и canvas. Манипуляции с пикселями изображения. Методики добавления на холст текста, теней и градиентов.

Михаил Сучан · 8 января 2009 г.

Введение

Спецификация HTML5 (ссылка: http://www.whatwg.org/specs/web-apps/current-work/multipage/ - http://www.whatwg.org/specs/web-apps/current-work/multipage/) включает множество новых свойств, одним из которых является элемент `canvas` (ссылка: http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html - http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html). Холст (`canvas`) HTML5 предоставляет простой и мощный способ создания графических изображений с помощью JavaScript. Для каждого элемента `canvas` можно использовать "контекст" (представьте страницу в альбоме для рисования), в который можно выполнять команды JavaScript для рисования. Браузеры могут реализовать несколько контекстов холстов и различные API предоставляют функции для рисования.

Большинство основных браузеров включают возможности контекста холста 2D - Opera, Firefox, Konqueror и Safari. Кроме того существуют экспериментальные сборки браузера Opera, которые включают поддержку контекста холста 3D, и дополнение, которое обеспечивает поддержку холста 3D в Firefox:

- Ссылка для загрузки сборки Opera с поддержкой холста 3D, HTML видео и в/в файлов (ссылка: http://labs.opera.com/news/2008/11/25/ - http://labs.opera.com/news/2008/11/25/).
- Узнайте больше об использовании контекста холста 3D в Opera

(ссылка: http://my.opera.com/timjoh/blog/2007/11/13/taking-the-canvas-to-another-dimension - http://my.opera.com/timjoh/blog/2007/11/13/taking-the-canvas-to-another-dimension).

- Узнайте больше о получении и использовании контекста холста 3D в Firefox (ссылка: http://blog.vlad1.com/2007/11/26/canvas-3d-gl-power-web-style/ - http://blog.vlad1.com/2007/11/26/canvas-3d-gl-power-web-style/).

В данной статье рассматриваются основы реализации контекста холста 2D и использования основных функций холста, включая линии, примитивы форм, изображения, текст и другие. Предполагается, что вы уже знакомы с основами JavaScript.

Отметим, что код всех примеров можно загрузить одним файлом zip-архива (ссылка: http://dev.opera.com/articles/view/html-5-canvas-the-basics/canvas-primer.zip - http://dev.opera.com/articles/view/html-5-canvas-the-basics/canvas-primer.zip), а также просмотреть их в действии, используя ссылки ниже.

Основы использования холста

Создание контекста холста на странице состоит просто в добавлении элемента `<canvas>` в документ HTML следующим образом:

```
<canvas id="myCanvas" width="200" height="150">
"Резервный контент на случай отсутствия поддержки
холста в браузере."
<span style="display:block;width:150px;height:100px;background:#00f"> </span
</canvas>
```

Необходимо определить ID элемента, чтобы можно было найти элемент позже в коде JavaScript, и нужно также определить ширину и высоту холста.

Мы создали альбом для рисования, так что теперь давайте коснемся пером бумаги. Чтобы рисовать на холсте, нужно использовать JavaScript. Сначала мы находим элемент холста с помощью `getElementById`, затем инициализируем требуемый контекст. После этого можно начинать рисовать на холсте с помощью доступных в API контекста

команд. Следующий сценарий (попробуйте выполнить пример - ссылка: http://www.robodesign.ro/coding/canvas-primer/20081208/example-using-canvas.html - http://www.robodesign.ro/coding/canvas-primer/20081208/example-using-canvas.html) рисует простой прямоугольник на определенном выше холсте:

```
window.addEventListener('load', function () {
  // Получаем ссылку на элемент.
  var elem = document.getElementById('myCanvas');
  if (!elem || !elem.getContext) {
    return;
  }

  // Получаем контекст 2d.
  var context = elem.getContext('2d');
  if (!context) {
    return;
  }

  // Все сделано! Теперь можно нарисовать синий прямоугольник.
  context.fillStyle = '#00f';
  context.fillRect(0, 0, 150, 100);
}, false);
```

Можно включить этот сценарий в заголовок документа или во внешний файл - все определяется вашими предпочтениями.

API контекста 2D

Теперь, когда мы создали свое первое базовое изображение на холсте, давайте посмотрим немного глубже на API холста 2D, и посмотрим, что нам доступно для использования.

Основные линии и штрихи

Мы уже видели в примере выше, что действительно легко рисовать прямоугольники, раскрашенные требуемым образом.

С помощью свойств `fillStyle` и `strokeStyle` можно легко задать цвета, используемые для изображения заполненных фигур и штрихов. Значения цветов, которые можно использовать, такие же как и в CSS: шестнадцатеричные коды, `rgb()`, `rgba()` и даже `hsla()`, если браузер поддерживает это (например, это свойство поддерживается в Opera 10.00 и более поздних версиях).

С помощью `fillRect` можно рисовать заполненные прямоугольники. С помощью `strokeRect` можно рисовать прямоугольники, используя только границы, без заполнения. Если вы хотите очистить некоторую часть холста, можно использовать `clearRect`. Эти три метода используют одинаковые аргументы: `x`, `y`, `width`, `height`. Два первых аргумента сообщают координаты `(x,y)`, а два последних аргумента задают ширину и высоту прямоугольника.

Чтобы изменить толщину линий, можно использовать свойство `lineWidth`. Давайте посмотрим на пример, который использует `fillRect`, `strokeRect`, `clearRect` и другие возможности:

```
context.fillStyle = '#00f'; // синий
context.strokeStyle = '#f00'; // красный
context.lineWidth = 4;

// Рисуем несколько прямоугольников.
context.fillRect (0, 0, 150, 50);
context.strokeRect(0, 60, 150, 50);
context.clearRect (30, 25, 90, 60);
context.strokeRect(30, 25, 90, 60);
```

Этот пример создает вывод, который можно видеть на рисунке 6.1.

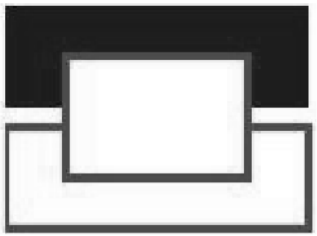


Рис. 6.1. Пример использования fillRect, strokeRect и clearRect

Пути

Пути холста позволяют рисовать произвольные фигуры. Вы чертите сначала "контур", затем выбираете для рисования штрих и в конце заполняете при желании фигуру. Создание специальной фигуры выполняется легко – чтобы начать рисовать путь, используйте `beginPath()`, затем начертите путь, который формирует фигуру с помощью прямых линий, кривых и других примитивов. Когда закончите, вызовите `fill` и `stroke`, если хотите заполнить фигуру или нарисовать штрихи, затем вызовите `closePath()`, чтобы закончить рисование фигуры.

Следующий пример показывает рисование треугольника:

```
// Задаем свойства стиля оформления.
context.fillStyle = '#00f';
context.strokeStyle = '#f00';
context.lineWidth = 4;

context.beginPath();
// Начинаем с верхней левой точки.
context.moveTo(10, 10); // задаем координаты (x,y)
context.lineTo(100, 10);
context.lineTo(10, 100);
context.lineTo(10, 10);

// Готово! Теперь заполните фигуру, и начертите штрихи.
// Примечание: фигура будет невидимой, пока не будет вызван
// любой из этих трех методов.
context.fill();
context.stroke();
context.closePath();
```

Это порождает вызов, показанный на рисунке 6.2.

Рис. 6.2. Простой треугольник

Я приготовил также пример более сложных путей, использующих прямые линии, кривые, и дуги (ссылка: http://www.robodesign.ro/coding/canvas-primer/20081208/example-paths.html - http://www.robodesign.ro/coding/canvas-primer/20081208/example-paths.html) – можно его посмотреть.

Вставка изображений

Метод `drawImage` позволяет вставлять другие изображения (элементы `img` и `canvas`) в контекст холста. В браузере Opera можно также рисовать изображения SVG внутри холста. Это достаточно сложный метод, который получает три, пять или девять аргументов:

- Три аргумента: Базовый вариант `drawImage` использует один аргумент для указания на включаемое изображение, и два для определения координат места назначения внутри контекста холста.
- Пять аргументов: Второй вариант использования `drawImage` включает приведенные выше три аргумента, плюс два для определения ширины и высоты вставляемого изображения (если вы захотите изменить его размер).
- Девять аргументов: Самый развитый вариант использования `drawImage` включает кроме пяти аргументов два значения для координат внутри исходного изображения, и два значения для ширины и высоты внутри исходного изображения. Эти значения позволяют динамически обрезать исходное изображение перед вставкой в контекст холста.

Следующий пример кода показывает все три типа `drawImage` в

действии (ссылка: http://www.robodesign.ro/coding/canvas-primer/20081208/example-drawimage.html - http://www.robodesign.ro/coding/canvas-primer/20081208/example-drawimage.html):

```
// Три аргумента: элемент, координаты места назначения (x,y).

context.drawImage(img_elem, dx, dy);

// Пять аргументов: элемент, координаты места назначения (x,y)
// и ширина и высота места назначения (если вы хотите изменить // раз

context.drawImage(img_elem, dx, dy, dw, dh);

// Девять аргументов: координаты места назначения (x,y),
// ширина и высота источника (для обрезки),
// координаты места назначения (x,y),
// ширина и высота места назначения (изменение размера).

context.drawImage(img_elem, sx, sy, sw, sh, dx, dy, dw, dh);
```

Это должно выводиться, как показано на рисунке 6.3.

Рис. 6.3. Пример использования drawImage

Манипуляции с пикселями

API контекста 2D предоставляет три метода, которые помогают рисовать с точностью до пикселя: `createImageData`, `getImageData`, и `putImageData`.

Пиксели хранятся в объектах типа `ImageData`. Каждый объект имеет

три свойства: `width, height` и `data`. Свойство `data` имеет тип `CanvasPixelArray`, имеющий количество элементов равное `width*height*4` ; это означает, что для каждого пикселя определяются значения красного, зеленого, синего цветов и alpha, в том порядке, в котором они должны появиться (все значения находятся в диапазоне от 0 до 255, включая alpha!). Пиксели упорядочиваются слева направо, ряд за рядом, сверху вниз.

Чтобы лучше понять, как все это работает, рассмотрим пример, который рисует блок красных пикселей (ссылка: http://www.robodesign.ro/coding/canvas-primer/20081208/example-imagedata2.html - http://www.robodesign.ro/coding/canvas-primer/20081208/example-imagedata2.html).

```
// Создаем объект ImageData

var imgd = context.createImageData(50,50);
var pix = imgd.data;

// Цикл по всем пикселям и задание прозрачного красного цвета

for (var i = 0, n = pix.length; i < n; i += 4) {
pix[i ] = 255; // канал red (красный)
pix[i+3] = 127; // канал alpha
}

// Отображение объекта ImageData в заданных координатах (x,y).

context.putImageData(imgd, 0,0);
```

Примечание: Не все браузеры реализуют `createImageData`. В таких браузерах необходимо получить объект `ImageData` с помощью метода `getImageData`. Посмотрите пример кода по следующей ссылке (ссылка: http://www.robodesign.ro/coding/canvas-primer/20081208/example-imagedata2.html - http://www.robodesign.ro/coding/canvas-primer/20081208/example-imagedata2.html).

С помощью возможностей ImageData можно делать значительно больше, чем только это. Например, можно выполнить фильтрацию изображения,

или можно сделать математическую визуализацию (представьте фракталы или что-то еще). Следующий код показывает, как создать простой фильтр для инверсии цвета (ссылка: http://www.robodesign.ro/coding/canvas-primer/20081208/example-imagedata.html - http://www.robodesign.ro/coding/canvas-primer/20081208/example-imagedata.html):

```
// Получаем CanvasPixelArray из заданных координат
// и размеров.

var imgd = context.getImageData(x, y, width, height);
var pix = imgd.data;

// Цикл по всем пикселям для инверсии цвета.

for (var i = 0, n = pix.length; i < n; i += 4) {
pix[i ] = 255 - pix[i ]; // red
pix[i+1] = 255 - pix[i+1]; // green
pix[i+2] = 255 - pix[i+2]; // blue
// i+3 будет alpha (четвертый элемент)
}

// Выводим ImageData в заданных координатах (x,y).
context.putImageData(imgd, x, y);
```

Рисунок 6.4 показывает фильтр инверсии цвета, примененный к графическому изображению Opera (сравните с рисунком 6.3, который показывает исходную цветовую схему графического изображения Opera).

Рис. 6.4. Фильтр инверсии цвета в действии

Текст

API текста доступен только в недавних сборках WebKit, и в ночных сборках Firefox 3.1, но я решил включить его здесь для полноты изложения.

На объекте `context` имеются следующие свойства текста:

- `font`: Определяет шрифт текста, таким же образом как свойство CSS `font-family`)
- `textAlign`: Определяет горизонтальное выравнивание текста. Значения: `start`, `end`, `left`, `right`, `center`. Значение по умолчанию: `start`.
- `textBaseline`: Определяет вертикальное выравнивание текста. Значения: `top`, *`hanging`*, `middle`, *`alphabetic`*, *`ideographic`*, `bottom`. Значение по умолчанию: *`alphabetic`*.

Для отображения текста имеется два метода: `fillText` и `strokeText`.

Первый из них изображает форму текста, заполненную с помощью текущего стиля `fillStyle`, в то время как второй изображает контур/границу текста используя текущий стиль `strokeStyle`. Оба получают три аргумента: текст для вывода, и координаты (x,y), определяющие место отображения. Существует также необязательный четвертый аргумент – максимальная ширина. Это приводит к тому, что браузер, если понадобится, сжимает текст, чтобы разместить его внутри заданной ширины.

Свойства выравнивания текста влияют на положение текста относительно координат (x,y), заданных в методах отображения.

Теперь пора обратиться к практике – следующий код является примером создания на холсте простого текста "hello world".

```
context.fillStyle = '#00f';
context.font = 'italic 30px sans-serif';
```

```
context.textBaseline = 'top';
context.fillText ('Hello world!', 0, 0);
context.font = 'bold 30px sans-serif';
context.strokeText('Hello world!', 0, 50);
```

Рисунок 6.5 показывает вывод этого примера.

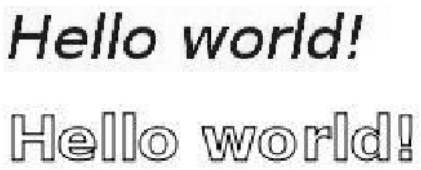


Рис. 6.5. Отображение простого текста на холсте

Тени

API теней предоставляет четыре свойства:

- `shadowColor`: Задает желательный цвет тени. Допустимые значения такие же как и значения цвета в CSS.
- `shadowBlur`: Задает величину размытости на тени в пикселях. Чем меньше значение размытости, тем более резкой будет тень. Создает эффект очень похожий на размытость по Гауссу в Photoshop.
- `shadowOffsetX` и `shadowOffsetY`: Определяет смещение тени по x и y в пикселях.

Использование вполне очевидно, как показано в следующем примере кода тени холста (ссылка: http://www.robodesign.ro/coding/canvas-primer/20081208/example-shadows.html - http://www.robodesign.ro/coding/canvas-primer/20081208/example-shadows.html):

```
context.shadowOffsetX = 5;
context.shadowOffsetY = 5;
context.shadowBlur = 4;
context.shadowColor = 'rgba(255, 0, 0, 0.5)';
context.fillStyle = '#00f';
```

```
context.fillRect(20, 20, 150, 100);
```

Этот код отображается, как показано на рисунке 6.6.

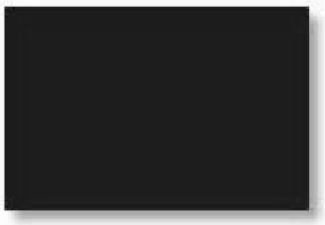


Рис. 6.6. Пример тени холста – синий прямоугольник с красной тенью

Градиенты

Свойства `fillStyle` и `strokeStyle` могут также иметь присвоенные им объекты `CanvasGradient` вместо строк цвета CSS – это позволяет использовать цветовые градиенты для окрашивания линий и заполнений вместо однородных цветов.

Для создания объектов `CanvasGradient` можно использовать два метода: `createLinearGradient` и `createRadialGradient`. Первый из них создает линейный градиент – линии цвета все идут в одном направлении – в то время как второй создает радиальный градиент – цветовые окружности, расходящиеся из одной точки.

Когда имеется объект градиента можно добавить опорные точки цвета вдоль градиента, используя метод объекта `addColorStop`.

Следующий код показывает, как использовать градиенты:

```
// Необходимо предоставить для градиента
// координаты (x,y) источника и места назначения
// (откуда он начинается и где заканчивается).

var gradient1 = context.createLinearGradient(sx, sy, dx, dy);

// Теперь можно добавить в градиент цвета.
```

```
// Первый аргумент задает позицию цвета в градиенте.
// Диапазон допустимых значений от 0 (начало градиента)
// до 1 (конец градиента).
// Второй аргумент сообщает требуемый цвет,
// используя формат цвета CSS.

gradient1.addColorStop(0, '#f00'); // красный
gradient1.addColorStop(0.5, '#ff0'); // желтый
gradient1.addColorStop(1, '#00f'); // синий

// Для радиального градиента также нужно задать
// источник и радиус окружности назначения.
// Координаты (x,y) определяют центр окружности
// (начало и окружности назначения).

var gradient2 = context.createRadialGradient(sx, sy, sr, dx, dy, dr);

// Добавление цветов для радиального градиента делается
// как добавление цветов к линейным градиентам.
```

Я подготовил также более сложный пример, который использует линейный градиент, тени и текст (ссылка: http://www.robodesign.ro/coding/canvas-primer/20081208/example-gradients.html - http://www.robodesign.ro/coding/canvas-primer/20081208/example-gradients.html). Пример создает вывод, показанный на рисунке 6.7.

Рис. 6.7. Пример с использованием линейного градиента

Сетевые примеры использования холста

Если вы хотите узнать, что другие сделали с помощью холста, то можно посмотреть следующие проекты:

- Виджеты Opera:
- SimAquarium (ссылка: http://widgets.opera.com/widget/5040/ - http://widgets.opera.com/widget/5040/)
- Artist's Sketchbook (ссылка: http://widgets.opera.com/widget/4647/ - http://widgets.opera.com/widget/4647/)
- Spirograph (ссылка: http://widgets.opera.com/widget/5218/ - http://widgets.opera.com/widget/5218/)
- Сетевые проекты и демонстрации
- Полином Ньютона (ссылка: http://www.benjoffe.com/code/demos/interpolate/ - http://www.benjoffe.com/code/demos/interpolate/)
- Canvascape - "3D Walker" (ссылка: http://www.benjoffe.com/code/demos/canvascape/ - http://www.benjoffe.com/code/demos/canvascape/)
- Paint.Web – демонстрация рисования, открытый исходный код (ссылка: http://code.google.com/p/paintweb - http://code.google.com/p/paintweb)
- Полет среди звезд (ссылка: http://arapehlivanian.com/wp-content/uploads/2007/02/canvas.html - http://arapehlivanian.com/wp-content/uploads/2007/02/canvas.html)
- Интерактивный blob (ссылка: http://www.blobsallad.se/ - http://www.blobsallad.se/)

Заключение

Холст является одним из наиболее интересных свойств HTML5, и он готов к использованию в большинстве современных браузеров. Он предоставляет все необходимое для создания игр, усовершенствований интерфейса пользователя, а также других вещей. API контекста 2D включает множество функций помимо рассмотренных в этой статье. Я надеюсь, что вы получили хорошее представление о возможностях холста и жажду узнать больше.

Введение в сокеты Web

Приводится описание API сокетов HTML5 с примерами использования. Основные протоколы для обмена информацией между клиентскими и серверными приложениями. Методы, свойства и события объекта WebSocket. Описаны конструкторы для создания соединения с сервером, использующие различные настройки. Открытие и закрытие соединений. Характеристика сообщений от сервера, обработка основных ошибок, возникающих при работе с сокетами. Методики проверки поддержки сокетов в браузере клиента.

Брюс Лоусон, Майк Тейлор · 22 ноября 2010 г.

Сокеты Web отключены по умолчанию в связи со спецификацией связанной с проблемами безопасности. Тем не менее, можно включить их в браузере Opera с помощью `opera:config#UserPrefs|EnableWebSockets`.

Введение

Бета-версия Opera 11 продемонстрировала поддержку сокетов Web, свойства, которое было ранее частью спецификации HTML5, известное как TCPConnection (ссылка: http://www.w3.org/TR/2008/WD-html5-20080610/comms.html#tcpconnection - http://www.w3.org/TR/2008/WD-html5-20080610/comms.html#tcpconnection). Однако сегодня сокеты Web определяются в двух местах: Web Sockets API (ссылка: http://www.w3.org/TR/websockets/ - http://www.w3.org/TR/websockets/) поддерживает редактор HTML5 Ян Хиксон, в то время как протокол (ссылка: http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol - http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol) редактирует Ян Фетте. Сокеты Web позволяют приложению передавать информацию между сервером и браузером, а не опрашивать сервер с заданным интервалом времени или использовать нестандартные приемы с помощью `<iframe>` — это позволяет разработчикам web уменьшить ненужный трафик HTTP и сложность программирования.

В этой статье мы объясняем предпосылки создания сокетов Web, показываем, почему это полезное свойство, и демонстрируем несколько

примеров простого использования.

История сокетов Web

Существует городская легенда, что сокеты Web были изобретены Яаном Хиксоном в качестве способа управления моделью поезда из браузера (ссылка: http://ln.hixie.ch/?start=1113762425 - http://ln.hixie.ch/?start=1113762425). К сожалению, это неправда, так как по словам самого Хиксона (частное сообщение):

Я думаю, что работа над сокетами Web (TCPConnection в то время) предшествовала моим мыслям об управлении моим макетом поезда с помощью сокетов Web, но сокеты Web несомненно могли бы сделать это намного легче. Проблема состоит в том, что обычное приложение Web управляет одиночной системой: имеется один компьютер, соединенный через последовательный порт с устройством Marklin Interface (ссылка: https://secure.wikimedia.org/wikipedia/en/wiki/Marklin_Digital - https://secure.wikimedia.org/wikipedia/en/wiki/Marklin_Digital), и требуется написать страницу Web, которая взаимодействует с этим компьютером, посылает инструкции и получает обновления, не опрашивая, имеет несколько соединений, использует потоки `<script>` в `<iframe>` и т.д.

Почему сокеты Web полезны

API сокетов Web позволяет открывать соединение с сервером, используя новый протокол *ws*, который остается открытым в течение времени жизни сеанса. Он является полнодуплексным, т.е. допускает коммуникацию в обоих направлениях одновременно. Он также имеет значительно меньшие накладные расходы, чем повторяющиеся обращения к серверу для отслеживания изменений. Раньше такие функции были доступны только с помощью технологии плагинов типа Flash.

Тестирование, выполненное компанией Kaazing Corp (ссылка: http://www.websockets.org/ quantum.html - http://www.websockets.org/quantum.html), которая была тесно вовлечена в процесс спецификации, показали, что " Сокеты Web из HTML5 могут обеспечить пятисоткратное — в зависимости от размера заголовков

HTTP — и даже тысячекратное сокращение бесполезного трафика заголовков HTTP и трехкратное сокращение времени задержки".

Короче говоря: Сокеты Web могут сделать приложения быстрее, более эффективными, и более масштабируемыми.

Как работают сокеты Web

Сокеты Web включают в себя API JavaScript, описанный ниже, и протокол `ws:` — или `wss:` для шифрованной передачи. Браузер Opera поддерживает версию -00 протокола (обозначаемую также -76). То же самое относится к Chrome 6, Safari 5.0.2 и Firefox 4 beta. Протокол стандартизован в IETF (ссылка: http://www.ietf.org/ - http://www.ietf.org/) и может изменяться; однако изменение API маловероятно. Это означает, что приложение может прекратить работать, если оно общается с сервером, который использует другую версию протокола, но для относительно стабильного клиентского API вам, вероятно, не понадобится изменять клиентский код.

API сокетов Web

Для соединения с сервером сокета Web, используется конструктор WebSocket следующим образом:

```
var ws = new WebSocket('ws://example.org:12345/demo');
```

Можно также запросить специальные субпротоколы с помощью второго параметра:

```
var ws = new WebSocket('ws://example.org:12345/demo', 'my-chat-protocol');
```

Если требуется более одного субпротокола, можно передать их в массиве строк (будет поддерживаться в будущей версии браузера Opera):

```
var ws = new WebSocket('ws://example.org:12345/demo', ['chat-protocol-v2-w
```

Сервер будет выбирать наиболее совместимую версию, которую можно затем проверить, считывая свойство `ws.protocol`.

Можно пытаться соединиться с любым хостом и любым портом, за исключением блокированных портов, однако сервер должен поддерживать сокеты Web и будет ожидать соединение от определенной страницы, которая, как ожидается, открывает соединение. Субпротокол, если используется, является произвольной строкой (состоящей из печатных символов ASCII), которая гарантирует, что клиент и сервер посылают сообщения, которые являются взаимно понятными. Например, Caucho Technology (ссылка: http://blog.caucho.com/?p=500 - http://blog.caucho.com/?p=500) пишет, что субпротоколы полезны "чтобы гарантировать, что клиент Quake/2.0 [WebSocket] не будет путаться, общаясь с сервером Quake/1.0 [WebSocket]".

Если соединение установлено, объект `WebSocket` получает событие `open`, и после этого можно начинать отправлять и получать сообщения. В контексте сокетов Web, сообщение является просто строкой текста, например, в формате JSON. Вы посылаете сообщение с помощью метода `send()`, и обрабатываете входящие сообщения с помощью перехватчика событий `onmessage`:

```
ws.onopen = function(e) {
// соединение теперь установлено
// давайте пошлем простое сообщение
this.send('{"username": "Bruce", "message": "Hello!"}');
}
ws.onmessage = function(e) {
// получить сообщение с сервера
var msg = JSON.parse(e.data);
alert(msg.message);
}
```

Если сервер отказывает в соединении, или если соединение закрывается по какой-то причине, объект `WebSocket` получает сообщение `close`.

```
ws.onclose = function(e) {
alert('WebSocket closed :-(');
}
```

Соединение можно закрыть с помощью метода `close()`:

```
ws.close();
```

Если сервер посылает кадры, которые браузер не понимает (может быть потому что сервер поддерживает только более новую версию протокола), то вы получите событие `error`. Если вы получаете события `error`, но не события `message`, вы можете попробовать закрыть соединение и вернуться к использованию другого средства, например, длинные опросы `XHR` или сервер Comet.

Функция обнаружения сокетов Web устроена просто — выполните следующее:

```
if ('WebSocket' in window) {
// Сокеты Web поддерживаются
} else {
// Сокеты Web не поддерживаются
}
```

Или если вы уже используете Modernizr (ссылка: http://www.modernizr.com/ - http://www.modernizr.com/) в своем приложении, то обнаружение будет таким же простым:

```
if (Modernizr.websockets) {
// Сокеты Web поддерживаются
} else {
// Сокеты Web не поддерживаются
}
```

Как быть в случае неподдерживающих сокеты Web браузеров?

Вики HTML5 Cross browser Polyfills (ссылка: https://github.com/Modernizr/Modernizr/ wiki/HTML5-Cross-browser-Polyfills - https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills) перечисляет три сценария, которые можно использовать в качестве средства симуляции сокетов Web для более старых, неподдерживающих сокеты Web браузеров (отметим, что мы не проверяли все такие браузеры). Проект Socket.IO (ссылка: http://socket.io/ - http://socket.io/) является особенно хорошим решением, так как он соединяет сервер

сокета Web Node.js с унифицированным, абстрактным API с автоматическим откатом к сокетам Flash, длинному опросу AJAX, многокомпонентному потоковому AJAX, потоковому `<iframe>` и JSONP для более слабых клиентов. Socket.IO распространяется с открытым исходным кодом и работает с `Node.js` (ссылка: http://nodejs.org/ - http://nodejs.org/), с несколькими другими совместимыми реализациями для других серверных технологий.

Демонстрация

Следующие приложения используют сокеты Web для предоставления многопользовательского взаимодействия (почти) в реальном времени:

- Mr. Doob's Multiuser Sketchpad (ссылка: http://mrdoob.com/projects/multiuserpad/ - http://mrdoob.com/projects/multiuserpad/): В этом многопользовательском приложении рисования с помощью <canvas>, сокеты Web используются для передачи координат линий, которые другие пользователи рисуют на каждом клиенте как получится.
- Rumpetroll (ссылка: http://rumpetroll.com/ - http://rumpetroll.com/): В качестве головастика вы можете плавать и общаться с другими головастиками. Сокеты Web предназначены для определения местонахождения головастиков и сообщений, которые передаются каждому пользователю.
- wordsquared.com (ссылка: http://wordsquared.com/ - http://wordsquared.com/): В этой многопользовательской сетевой игре со словами сокеты Web используются, чтобы пользователи видели карточки других игроков во время игры.

Кроме этих демонстраций Ericsson Labs имеет интересный видеофильм, демонстрирующий преимущества сокетов Web по сравнению с длинными опросами HTTP для приложений реального времени:

- Отличие WebSocket (ссылка: https://labs.ericsson.com/developer-community/blog/video-websocket-difference - https://labs.ericsson.com/developer-community/blog/video-websocket-difference)

Ресурсы

- Симон Питерс. Сокеты Web в браузере Opera (ссылка: http://my.opera.com/core/blog/websockets - http://my.opera.com/core/blog/websockets)
- Пакет тестирования сокетов Web (ссылка: http://testsuites.opera.com/websockets/ - http://testsuites.opera.com/websockets/)
- Спецификация сокетов Web (ссылка: http://dev.w3.org/html5/websockets/ - http://dev.w3.org/html5/websockets/)
- Протокол сокетов Web (ссылка: http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol - http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol)

Автономное выполнение приложений Web с помощью HTML5 AppCache

В данной лекции речь идет о методах кэширования приложений на клиентских компьютерах. Отличия кэша браузеров от кэша приложений HTML5. Файл манифеста и его основные директивы. Подключение манифеста к web приложению. Явное определение файлов для кэширования. Предоставление пользователю резервного контента. Использование API кэша приложений и событий для проверки использования актуальных версий файлов. Проверка поддержки технологии браузером. Статусы, события и обработчики событий кэша приложений.

Шветанк Диксит · 1 июля 2010 г.

Введение

Приложения Web стали существенной частью жизни людей в такой степени, что многие используют их все время. Не будет ли полезно, если мы сможем использовать их, даже когда не подключены к сети? До недавних пор не существовало никакого способа это сделать – однако, с появлением в W3C HTML5 кэша приложений (ссылка: http://dev.w3.org/html5/spec/offline.html - http://dev.w3.org/html5/spec/offline.html), стало возможно выполнение приложения Web в автономном режиме, также как и в сети. Давайте посмотрим, как это реализуется.

Зачем приложение выполняется в автономном режиме?

Приложения Web становятся с каждым днем все сложнее и мощнее. Существует множество примеров приложений web в различных областях выполняющих такую же работу, как и настольные приложения (подумайте о Google Docs, Picasa, и т.д.). Однако один существенный недостаток состоит в том, что они не могут работать, когда пользователь не соединен с Интернет.

Здесь на сцене появляется новое автономное хранилище HTML5. Оно

пытается избавиться от этого недостатка, определяя метод сохранения файлов в кэш-памяти, чтобы когда пользователь не подключен к сети, браузер имел, тем не менее, доступ к необходимым файлам. Это могут быть файлы HTML, CSS или JavaScript, или любые другие ресурсы необходимые сайту для работы.

Сохранение файлов в кэше приложений для автономного использования

HTML5 для автономных приложений web использует новое средство, называемое кэшем приложений, или коротко AppCache. Сохраняемые в этом AppCache файлы доступны для приложения, даже когда пользователь не находится в сети. С помощью файла манифеста можно определить, какие файлы необходимо сохранить в AppCache.

В чем отличие кэша приложений от обычного кэша браузера?

Существует ряд особенностей, которые отличают AppCache от обычного кэша браузера. Прежде всего, они решают различные задачи. AppCache предназначен для подходящих приложений Web, в то время как кэш браузера предназначен вообще для обычных страниц web. Обычная кэш-память будет кэшировать почти любую страницу, в то время как AppCache будет кэшировать только те страницы, которые специально указаны в файле манифеста. Кроме того, обычный кэш ненадежен, так как мы не знаем, какие страницы (и какие ресурсы этих страниц) будут наверняка доступны.

Преимущество использования AppCache состоит в том, что теперь разработчик имеет значительно больше программного контроля над кэшем, что означает значительно больше уверенности и контроля над тем, как приложения Web будут вести себя в автономном режиме. Отметим также, что можно иметь несколько страниц, совместно использующих одну память AppCache. Также с помощью AppCache можно использовать API для определения состояния памяти AppCache, и затем даже заставить ее обновиться.

Файл манифеста

Этот файл находится на сервере и указывает, какие файлы должны храниться на клиентской стороне в AppCache браузера, готовые к тому, что пользователь переходит в автономный режим. Давайте рассмотрим подробнее, как он работает.

Для файла манифеста можно задать любое имя, но рекомендуется давать ему расширение

`.manifest`. Каждый файл манифеста должен начинаться с `CACHE MANIFEST`, после чего перечисляются файлы, которые желательно сохранить и сделать доступными для автономного использования. Можно задавать комментарии, помещая # в начале строки. Очень простой файл манифеста выглядит следующим образом:

```
CACHE MANIFEST
#Можно также использовать заголовок раздела CACHE:,
#чтобы явно объявить три следующих файла.

style.css
script.js
index.htm
```

Файл манифеста должен иметь правильный тип MIME, который должен быть `text/cache-manifest`. Чтобы сделать это, можно задать расширение `.manifest` для файла манифеста, и добавить следующую строку в файл `.htaccess` на сервере:

```
AddType text/cache-manifest .manifest
```

Соединение файла HTML с файлом манифеста

Теперь, после создания файла манифеста, говорящего какие файлы необходимо кэшировать в приложении, нужно сообщить странице HTML, чтобы она использовала этот кэш. Для этого нужно соединить страницу с файлом манифеста, включая атрибут

`manifest` в тег `<HTML>` страницы. Например:

```
<html manifest="demo.manifest">
```

Если приложение Web имеет более одной страницы, необходимо сделать так, чтобы все страницы соединялись с файлом манифеста таким образом, иначе они не будут частью AppCache, и не будут работать автономно.

Использование заголовков разделов для улучшения контроля AppCache

До сих пор мы видели самый простой пример использования файла манифеста. С помощью заголовков разделов можно в действительности точно задать, что определенный файл должен кэшироваться или нет.

Явное определение файлов для кэширования

Можно использовать заголовок раздела `CACHE`: для явного объявления, какие файлы необходимо кэшировать. Например, предыдущий пример файла манифеста можно написать следующим образом, и он будет функционировать точно таким же образом:

```
CACHE MANIFEST

CACHE:
style.css
script.js
index.htm
```

Единственное различие состоит в том, что в этом примере мы явно объявили, что все эти файлы будут частью кэша приложений.

Вот очень простой пример страницы, которая использует заголовок раздела `CACHE`: (ссылка: http://people.opera.com/shwetankd/demos/2/index.htm - http://people.opera.com/shwetankd/demos/2/index.htm).

Важно отметить, что указанный для файлов путь доступа должен задаваться относительно расположения файла манифеста. В примере

здесь мы предполагаем, что упомянутые файлы находятся в том же каталоге, что и файл манифеста. Можно использовать как относительные, так и абсолютные URL при указании файлов в файле манифеста.

Файлы, определенные как часть `CACHE:`, будут загружаться из AppCache (а не с сервера), даже если вы находитесь в сети, при условии, что в файле манифеста нет изменений. Если, однако, браузер, найдет обновленный файл манифеста, то новый кэш будет снова загружен однократно, в соответствии с тем, что говорит новый файл манифеста. Поэтому AppCache может быть неподходящим для сайтов с быстро изменяющимся контентом, таким как новостные блоги, например, но может быть очень полезен для приложений Web, которые делают определенную работу и могут работать автономно (например, приложение календаря, или список дел, и т.д.).

Как быть, если требуется загрузить файл непосредственно с сервера, а не из кэша?

Если страница связана с файлом манифеста, то только упомянутые в манифесте файлы будут пытаться загрузиться, независимо от того находится пользователь в сети или нет. Однако могут возникать ситуации, когда требуется, чтобы какой-то файл не соединялся с кэшем, когда пользователь находится в сети, а чтобы он соединился и загрузил свежие данные с сервера вместо кэша (Например, некоторый динамический контент из сценария CGI).

По сути, если страница связана с файлом манифеста, то весь сетевой трафик ее файлов блокируется, и файлы либо должны загружаться из AppCache, либо отказываться загружаться. Заголовок раздела `NETWORK:` делает исключения для этого правила. Можно использовать раздел заголовка `NETWORK:` для объявления, какие файлы не должны кэшироваться, чтобы они загружались с сервера, и никогда не были частью кэша приложений. Раздел заголовка `NETWORK:` принимает во внимание заголовок обычного кэша браузера. Поэтому, если предполагается, что файл будет кэшироваться обычным кэшем браузера, то он и будет им кэшироваться (также как и любой другой файл, не определенный в AppCache), даже если он определен под заголовком

раздела NETWORK:.

```
CACHE MANIFEST

CACHE:
style.css
script.js
index.htm

NETWORK:
style2.css
```

В примере выше style2.css всегда будет загружаться с сервера и никогда не будет частью кэша приложений. Помните, что могут быть ситуации, когда имеется слишком много файлов для перечисления, которые не должны использовать кэш, что делает запись всех этих файлов под заголовком раздела NETWORK: обременительным. В этом случае можно использовать символ звездочки (*), что разрешает всем URL выходить в сеть, если вы находитесь в сети.

Посмотрите на пример, который использует файл манифеста с заголовком раздела NETWORK: (ссылка: http://people.opera.com/shwetankd/demos/3/index.htm - http://people.opera.com/shwetankd/demos/3/index.htm). Можно заметить, что когда вы не в сети и перезагружаете страницу, страница перезагружается, но оформление фона исчезает. Это связано с тем, что оформление фона в этом примере находится в файле style2.css, который находится под заголовком раздела NETWORK:, что означает, что он не кэшируется и будет загружаться, только когда вы в сети и перезагружаете страницу.

Предоставление резервного контента

Раздел заголовка FALLBACK: используется для определения запасных ресурсов, которые будут использоваться вместо файлов, которые отказываются загружаться (или загружаются не полностью):

```
CACHE MANIFEST
```

```
CACHE:
style.css
script.js
index.htm

NETWORK:
style2.css

FALLBACK:
main_image.jpg backup_image.jpg
```

Предполагается, что резервный контент будет кэшироваться и будет использоваться только в том случае, когда основной контент не загружается. В приведенном выше примере файл `backup_image.jpg` кэшируется в AppCache, поэтому когда файл `main_image.jpg` не может загрузиться, на его место будет загружаться `backup_image.jpg`. Посмотрите пример с резервной копией манифеста (ссылка: http://people.opera.com/shwetankd/demos/4/index.htm - http://people.opera.com/shwetankd/demos/4/index.htm) — если перейти на эту страницу, и отсоединиться от Интернет, а затем перезагрузить страницу, браузер попытается загрузить изображение, но так как вы не в сети (и изображение не кэшировано) оно не будет загружено, и, следовательно, на его место будет загружен резервный контент. (Браузер сначала в течение некоторого времени будет пытаться загрузить основной контент, и только затем загрузит резервный контент ... так что будьте терпеливы!)

Файл манифеста, который используется в этом примере, предоставляет резервный контент для нескольких изображений (ссылка: http://people.opera.com/shwetankd/demos/5/index.htm - http://people.opera.com/shwetankd/demos/5/index.htm).

Использование API кэша приложений и событий для проверки, что кэш использует самые свежие файлы

Одним из больших достоинств кэша приложений является то, что теперь программист имеет доступ к тому, как себя должен вести кэш. Он

имеет доступ к событиям, которые могут сообщить о текущем состоянии кэша приложений, и имеет также функции для асинхронного обновления. Например, можно использовать функцию `window.applicationCache`, чтобы определить, поддерживает браузер кэш приложений или нет. Давайте посмотрим на некоторые другие способы, которыми можно получить программный контроль над кэшем приложений.

Статусы

Текущий статус кэша приложений можно проверить с помощью функции `window.applicationCache.status`, которая возвращает числовое значение, соответствующее следующим статусам:

0 - uncached (не кэшировано)

Если страница не соединена с кэшем приложений. Также при самой первой загрузке кэша приложений и во время загрузки AppCache будет иметь статус `uncached`.

1 – idle (не работает)

Когда браузер имеет самую последнюю версию AppCache, и нет обновленных версий для загрузки, то статус задается как `Idle`.

2 – checking (проверка)

В течение времени, когда страница проверяет наличие обновленного файла манифеста, статус задается как `Checking`.

3 – downloading (загрузка)

В течение времени, когда страница фактически загружает новый кэш (если был обнаружен обновленный файл манифеста), статус задается как `downloading`

4 – updateready (обновление готово)

Когда браузер заканчивает загрузку нового кэша, он готов к использованию (но все еще не используется). В течение этого времени

статус задается как `updateready`

5 – obsolete (устарел)

Когда файл манифеста невозможно найти, статус задается как `obsolete`, и кэш приложений удаляется. Важно знать, что когда файл манифеста (или любой из файлов, упомянутых в манифесте, за исключением имеющих резервные копии) отказывается загружаться, то это будет считаться ошибкой и будет продолжать использоваться старый кэш приложений.

События

В зависимости от того, что происходит с AppCache в данный момент, могут порождаться также некоторые события.

Checking (проверка)

Это событие порождается, когда браузер выполняет проверку, пытаясь загрузить манифест в первый раз, или проверяет, что имеется обновленная версия файла манифеста.

Noupdate (нет обновления)

Если на сервере отсутствует обновленная версия файла манифеста, порождается событие `noupdate`.

Downloading (загрузка)

Если браузер загружает кэш в первый раз, или загружает обновленную версию, порождается это событие.

Progress (успешно)

Это событие порождается для каждого файла, который загружается как часть AppCache.

Cached (кэшировано)

Это событие порождается, когда все ресурсы закончили загрузку, и

приложение будет кэшировано.

Updateready (обновление готово)

Когда ресурсы закончили перезагрузку обновленного кэшированного файла, вызывается `updateready`. Когда это произошло, можно использовать `swapCache()` (как объясняется далее в статье), чтобы заставить браузер использовать этот вновь обновленный кэш.

Obsolete (устарел)

Это событие порождается, если файл манифеста невозможно найти (ошибка 404 или 410).

Error (ошибка)

Это событие может порождаться по различным причинам. Если файл манифеста невозможно найти, процесс загрузки кэша приложений должен быть прерван, и в данном случае может порождаться это событие. Оно может порождаться в случае наличия файла манифеста, но какой-нибудь из файлов, упомянутых в файле манифеста, не может быть загружен правильно. Оно может порождаться даже в случае изменений файла манифеста во время выполнения обновления (в этом случае браузер будет ждать какое-то время, прежде чем попытаться еще раз), или в любом другом случае, где возникает фатальная ошибка.

Все обработчики событий для этих событий имеют префикс `'on'`. Например, `onchecking`, `onupdateready`, `onerror`, и т.д.

API кэша приложений имеет несколько особенностей, которые стоит упомянуть:

`window.applicationCache.update()`: Эта функция будет запускать процесс загрузки кэша приложений, который является почти тем же, что и перезагрузка страницы. Она просто проверяет, изменился ли манифест, и, если возможно, загружает свежую версию всего контента в кэше (учитывая все заголовки кэша). Отметим, что даже хотя при этом создается новый кэш, страница будут продолжать использовать старый кэш. Чтобы заставить страницу использовать только что загруженный новый кэш, необходимо использовать функцию

`swapCache()`.

`window.applicationCache.swapCache()`: Эта функция приказывает браузеру начать использовать данные нового кэша, если он доступен. Важно отметить, что даже если имеется новый файл манифеста, приложение будет по-прежнему продолжать использовать старый кэш (как определено в старом файле манифеста), пока не будет вызвана функция `swapCache()`. Когда вызывается `swapCache()`, кэш будет использоваться, как определено в новом файле манифеста.

Обычно не требуется использовать функцию `update()`, так как браузер должен автоматически делать это при перезагрузке страницы. Наиболее часто функцию `swapCache()` будет использоваться в соединении с событием `onupdateready` (обновление готово).

В следующем примере, если изменить файл манифеста и перезагрузить страницу, браузер загрузит новые файлы в кэш, и затем переключится на новый кэш (так как вызывается функция `swapcache()`):

```
<html manifest="demo.manifest">
<head>
<script type="text/javascript">
window.applicationCache.addEventListener('updateready', function(){
window.applicationCache.swapCache();
}, false);
</script>
</head>
<body>
...
</body>
</html>
```

Если созданная страница вряд ли будет перезагружаться пользователем в течение какого-то времени, то можно было бы периодически вызывать функцию `update()`, чтобы проверять наличие обновлений файла манифеста, и в положительном случае вызывать функцию `swapcache()` на событии `updateready`, чтобы загрузить и переключиться на новый кэш:

```
setInterval(function () { window.applicationCache.update(); }, 3600000); // Проверять обновления файла манифеста каждые 60 минут. Если он обновляется, загрузить новый кэш, как определено новым файлом манифеста.
```

```
window.applicationCache.addEventListener('updateready', function(){
// когда обновленный кэш загрузится и будет готов для использования
window.applicationCache.swapCache();
//переключитесь на самую новую версию кэша
}, false);
```

Этот код будет проверять наличие обновленной версии файла манифеста каждые 60 минут. Если он находит версию файла манифеста на сервере, отличную от встречавшейся ранее, он загрузит новый кэш на основе этого нового манифеста. Когда это произойдет, будет порождаться событие `updateready`, сообщающее, что обновленная версия кэша закончила загружаться и готова к использованию. Затем можно явно использовать функцию `swapCache()`, чтобы переключиться со старого кэша на новый, только что загруженный.

Таким образом можно гарантировать, что кэш пользователя будет оставаться обновленным.

Заключение

Введение кэша приложений W3C HTML5 (ссылка: http://dev.w3.org/html5/spec/offline.html - http://dev.w3.org/html5/spec/offline.html) предоставляет множество новых возможностей для разработчиков web. Приложения Web могут теперь кэшироваться для автономного использования, делая тем самым их еще более мощными и полезными чем раньше.

Хранилище Web: более удобное и мощное хранилище клиентских данных

Рассматриваются сессионные (Session Storage) и локальные (Local Storage) хранилища данных на стороне клиента. Сравнение технологий хранилищ HTML5 с технологией Cookie. Помещение и извлечение данных из сессионного и локального хранилищ. Удаление данных. Лимит хранилища. Использование событий хранилища. Вопросы безопасности и соответствующие рекомендации.

Шветанк Диксит · 2 марта 2010 г.

Введение

Приложения Web становятся все более развитыми с каждым днем, со все более продуманным использованием JavaScript, а также появляющихся стандартов и технологий. Мы все в большей степени полагаемся на эти приложения, многие из них становятся частью нашей повседневной жизни. Одной из областей, в которой разработка приложений Web отстает от потребностей, является хранение данных на стороне клиента. То есть, до сих пор.

Хранилище Web (ссылка: http://dev.w3.org/html5/webstorage/ - http://dev.w3.org/html5/webstorage/) является спецификацией W3C, которая предоставляет функции для сохранения данных на стороне клиента до конца сеанса (Session Storage – сессионное хранилище), или после завершения сеанса (Local Storage – локальное хранилище). Это значительно более мощное средство, чем традиционные cookies, и более простое в использовании. В этой статье мы посмотрим, почему это так, и как можно его использовать.

Существующая проблема: cookies могут разрушаться

Прежде чем двигаться дальше, давайте коротко разберем, почему текущий способ хранения данных на стороне клиента — cookies — является проблемой:

- Маленький размер: Cookies обычно имеют максимальный размер около 4KB, что не слишком хорошо подходит для хранения сложных данных любого вида.
- С помощью cookies трудно отслеживать две или больше транзакций на одном и том же сайте, которые могут происходить в двух или более различных вкладках.
- Cookies могут использоваться злонамеренно с помощью так называемой техники межсайтового скриптинга, что приводит к проблемам безопасности.

Другие (менее популярные) альтернативы для cookies включают методы, использующие строки запросов, скрытые поля форм, совместно используемые локальные объекты на основе flash, и т.д. Каждый со своим собственным набором проблем, связанным с безопасностью, легкостью использования, ограничениями на размер, и т.д. Поэтому до сих пор мы используем достаточно плохие способы хранения данных на стороне пользователя. Нам требуется улучшенный способ, и здесь на помощь приходит Хранилище Web.

Хранилище Web

Спецификация Хранилища Web W3C (ссылка: http://dev.w3.org/html5/webstorage/ - http://dev.w3.org/html5/webstorage/) была разработана для улучшения способа хранения данных на стороне клиента. Она имеет два различных типа хранилища: Session Storage (Сессионное хранилище) и Local Storage (Локальное хранилище).

Как сессионное, так и локальное хранилище будут иметь возможность хранить около 5Mb данных на домен, что значительно больше чем cookies. По мере дальнейшего чтения вы больше узнаете о них, и о том, что делает хранилище Web более удобным механизмом хранения.

Сессионное хранилище

Сессионное хранилище имеет единственное предназначение: Запоминать все данные сеанса и забывать их, как только закрывается используемая вкладка (или окно).

Задание и извлечение данных

Чтобы задать пару ключ значение в сессионном хранилище, необходимо написать просто строку следующего вида:

```
sessionStorage.setItem(yourkey, yourvalue);
```

Чтобы снова извлечь данные, необходимо написать:

```
var item = sessionStorage.getItem(yourkey);
```

Чтобы сохранить значение "This is a sample sentence" в сессионном хранилище, можно написать:

```
sessionStorage.setItem(1, 'This is a sample sentence');
```

Здесь значением ключа будет 1, но это не значит, что это вообще первое значение. Число 1 просто преобразуется в строку '1', которая используется в качестве ключа, но это не помещает эту пару ключ значение в первую позицию.

А чтобы извлечь это предложение в предупреждающем сообщении JavaScript, необходимо написать:

```
var item = sessionStorage.getItem(1);
alert(item);
```

Другим примером `setItem()` может быть:

```
sessionStorage.setItem('name', 'john');
```

а извлечь значение можно с помощью

```
var name = sessionStorage.getItem('name');
```

Удаление данных

Существуют также методы для удаления данных из сессионного хранилища. Метод `removeItem()` используется для удаления определенного объекта из списка:

```
var item = sessionStorage.removeItem(yourkey);
```

Помните, что можно также сослаться просто на ключ объекта и удалить его из списка следующим образом:

```
var items = sessionStorage.removeItem(1);
```

Метод `clear()` используется для удаления всех объектов в списке; он применяется следующим образом:

```
sessionStorage.clear();
```

Можно использовать также атрибут `length`, чтобы определить число пар ключ/значение в хранилище, следующим образом:

```
var no_of_items = sessionStorage.length;
```

Локальное хранилище

Локальное хранилище используется, если требуется, чтобы данные сохранялись более чем для одного сеанса. Простым примером использования будет подсчет количества посещений пользователем страницы Web. Когда страница использует локальное хранилище, страница (или окно) можно закрыть и снова открыть, и, тем не менее, показать сохраненные данные — т.е. обеспечивается постоянное хранение.

Сохранение и извлечение данных в локальном хранилище работает аналогично сессионному хранилищу: оно использует такие же имена функций `setItem()` и `getItem()`. Чтобы сохранить предложение в локальном хранилище, нужно написать что-нибудь следующего вида:

```
localStorage.setItem(1, 'This is a sample sentence');
```

а чтобы извлечь его:

```
var data = localStorage.getItem(1);
```

Также как и сессионное хранилище, локальное хранилище поддерживает атрибут `length`, и функции `removeItem()` и

`clear()`.

Как в сессионном хранилище, так и в локальном функция `clear()` имеет одну задачу – удалить все значения из списка. Это означает, что если вызвать, например, функцию `localStorage.clear()`, то она удалит все локальное хранилище из этого источника. Поэтому все данные локального хранилища из, скажем, (такие как www.example.org, www.example.org:80, www.example.org/abc/, www.example.org/xyz/) будут удалены. Тем не менее, хранилище, скажем, для abc.example.org этим затронуто не будет. Однако для сессионного хранилища она будет очищать хранилище только для текущей сессии.

Простой пример

Чтобы проиллюстрировать хранилище Web в действии, был создан небольшой пример, который использует, как локальное, так и сессионное хранилище. Посмотрите демонстрационную страницу хранилища Web, чтобы увидеть его в действии (ссылка: http://people.opera.com/shwetankd/external/demos/webstorage_demo.htm - http://people.opera.com/shwetankd/external/demos/webstorage_demo.htm).
Пример попросит ввести две строки, одну для сессионного хранилища, и другую для локального хранилища. Затем можно открыть Storage Inspector в Opera Dragonfly, чтобы получить доступ к хранилищу Web. Можно заметить, что если закрыть страницу, а затем снова ее открыть, данные, введенные для локального хранилища, сохраняются, в то время как для сессионного хранилища, это будет не так.

Использование событий хранилища

Спецификация предоставляет также событие хранилища (ссылка: http://dev.w3.org/html5/webstorage/#the-storage-event - http://dev.w3.org/html5/webstorage/#the-storage-event), которое будет порождаться, когда область хранилища изменяется. Оно имеет различные полезные атрибуты, такие как:

- storageArea: Говорит, какой это тип хранилища (сессионное или локальное)
- key: Изменяющийся ключ.

- oldValue: Старое значение ключа.
- newValue: Новое значение ключа.
- url: URL страницы, ключ которой изменился.

Если вызвать метод `clear()`, то атрибуты `key`, `oldValue` и `newValue` устанавливаются пустыми. Здесь имеется модифицированная версия упомянутого ранее демонстрационного примера страницы (ссылка: http://people.opera.com/shwetankd/external/demos/webstorage_demo2.htm - http://people.opera.com/shwetankd/external/demos/webstorage_demo2.htm), в этот раз использующая события хранилища, чтобы пользователи могли знать об изменениях в значениях. Если ввести значение, а затем снова его изменить, то можно будет видеть предупреждение, упоминающее новое и старое значения.

Где можно получить детальный доступ к данным хранилища Web в браузере?

В Opera 10.50+ существует несколько способов, как это можно сделать. Можно ввести `opera:webstorage`, а также `opera:config#PersistentStorage` в поле адреса, чтобы получить доступ к высокоуровневым данным хранилища Web (какой лимит хранилища, где оно находится, и т.д.), но для разработчиков существует лучший способ получить подробную информацию о хранилище Web для конкретной страницы — с помощью Storage Inspector в Opera Dragonfly, который предоставляет значительно более подробную информацию.

Opera 10.50+ имеет новую и улучшенную утилиту отладки Opera Dragonfly (которая была опубликована как проект с открытым исходным кодом - ссылка: http://www.opera.com/dragonfly/ - http://www.opera.com/dragonfly/). Среди исправлений, усовершенствований и новых свойств появился Storage Inspector (Инспектор хранилища). Он создает для разработчиков отдельную вкладку для доступа к информации о cookies и локальном и сессионном хранилище страницы. Откройте Opera Dragonfly и щелкните на вкладке Storage, чтобы получить к ней доступ.

Что нужно помнить при использовании хранилища Web

Хранилище на источник: Все данные из одного и того же источника будут совместно использовать одно пространство хранения. Источником является тройка схема/хост/порт (или глобально уникальный идентификатор). Например, ссылка: http://www.example.org - http://www.example.org и ссылка: http://abc.example.org - http://abc.example.org являются двумя отдельными источниками, также как ссылка: http://example.org - http://example.org и ссылка: https://example.org - https://example.org, а также ссылка: http://example.org:80 - http://example.org:80 и ссылка: http://example.org:8000 - http://example.org:8000

Лимит хранилища: В настоящее время большинство браузеров, которые реализовали хранилище Web, включая Opera, определяют лимит хранилища как 5 Mb на домен. Можно изменить этот лимит хранилища для каждого домена отдельно, сохраняя какие-то данные из домена в сеансовом или локальном хранилище, а затем переходя к `opera:webstorage`. Этот домен появится тогда в списке, и можно будет щелкнуть на кнопке, чтобы получить доступ к статистике и параметрам, включая размер данных, сохраненных для этого домена, какой имеется лимит хранилища, и что браузер будет делать, когда лимит будет исчерпан.

Вопросы безопасности и соответствующие рекомендации: Хранилище назначается на основе источника. Злоумышленник может использовать подмену DNS, чтобы представить себя определенным доменом, которым он фактически не является, получая тем самым доступ к области хранилища этого домена на компьютере пользователя. Можно использовать SSL, чтобы предотвратить такие действия, чтобы пользователи могли быть абсолютно уверены, что просматриваемый сайт находится в том же домене.

Где не надо использовать: Если два различных пользователя используют различные пути доступа на одном домене, они могут получить доступ к области хранения всего источника, и поэтому к данным друг друга. Поэтому в настоящее время не рекомендуется использовать хранилище Web на своих страницах пользователям свободных хостов, которые имеют свои сайты в различных каталогах

одного и того же домена (например, `freehostingspace.org/user1/` и `freehostingspace.org/user2/`).

Хранилище Web не является частью спецификации HTML5: Это целая спецификация сама по себе (ссылка: http://dev.w3.org/html5/webstorage/ - http://dev.w3.org/html5/webstorage/).

Web Workers за работой

Введение в технологию многопоточного выполнения кода Web Workers. Принципы работы и случаи использования. Какие стандартные объекты JavaScript доступны для Web Workers. Присущие ему ограничения. Поддержка в современных браузерах. Дополнительные ссылки по теме.

Дэниел Дэвис · 1 июля 2010 г.

Введение

Представьте следующее. Вы являетесь любимым руководителем малоизвестной страны ScravaJipt, безраздельно властвуя над всем, что обозреваете. У вас есть главный слуга, который ухаживает за вами, покупает одежду, нажимает кнопки на мобильном телефоне. Но иногда он со всем не справляется. Он бывает перегружен всеми этим повседневными заботами, так что приходится передавать часть работы специалистам (один для нажатия кнопок, еще один для покупки рубашек, другой для покупки брюк). К счастью для него имеется множество работников, на которых можно положиться. Аналогично, к счастью для разработчиков web существуют виртуальные цифровые специалисты, которые могут взять на себя некоторые задачи, когда процессор JavaScript становится перегруженным. Познакомьтесь с Web Workers, одной из множества технологий, которые, совместно с HTML5 формируют следующее поколение открытой Web.

Введение в технологию Web Workers

Встречали ли вы страницу, которая отображается частично, но не отвечает ни на какие щелчки? Или страницу, которая замораживается или приводит к аварийной остановке браузер?

Причина была, скорее всего, в JavaScript. Страницы Web становятся все в большей степени перегружены JavaScript, иногда в такой степени, что не могут двигаться. Вездесущность JavaScript является благом для разработчиков, но это означает, что язык может выполняться на широком множестве устройств, включая такие, которым не хватает

мощности для сегодняшних приложений Web. Существует несколько способов оптимизации JavaScript, но он все равно не будет работать так же быстро, как машинный код.

Разработчики Web не собираются (и не должны) в связи с этим сокращать свое использование JavaScript. Вместо этого, стандарты Web и браузеры, которые их реализуют, продвигаются вперед, чтобы выдержать нагрузку. Технология Web Workers является этому примером, вместе с различными другими API JavaScript, которые разработаны для повышения возможностей браузера.

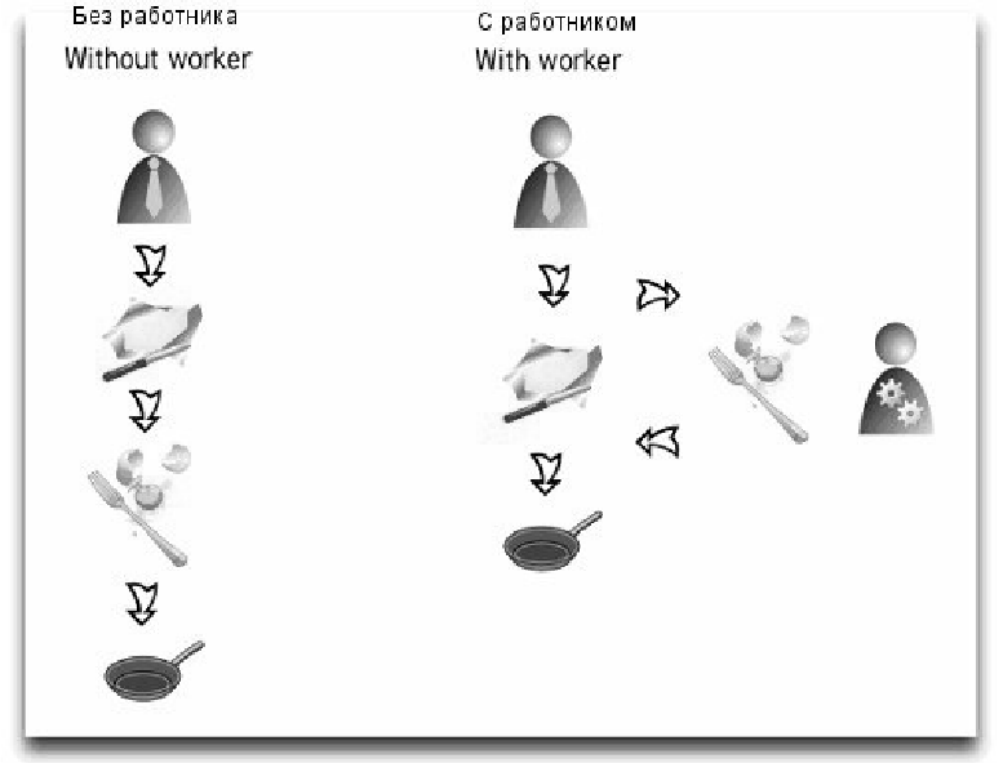
Как работают Web Workers

Большинство современных языков программирования являются мультипоточными, т.е. они могут выполнять несколько процессов одновременно. Превращение JavaScript в мультипоточный язык потребует значительных архитектурных изменений и фундаментального переосмысливания, поэтому Web Workers предлагают способ обойти эту проблему, позволяя расширить язык таким образом, что он может казаться в некоторых случаях мультипоточным. Другими словами может эффективно выполняться более одного процесса, но с некоторыми ограничениями. На самом деле достаточно много ограничений, поэтому они будут полезны только в определенных ситуациях.

Когда я могу их использовать?

Возвращаясь к нашей аналогии со "специалистами" можно сказать, что Web Workers могут делать только одну вещь, но они делают ее очень хорошо. Они прекрасно справляются с выполнением быстрых вычислений, но не могут сделать более сложную работу, такую как доступ к DOM. Если сравнить web-приложение с кухней, то основной поток JavaScript будет шеф-поваром, собирающимся приготовить омлет. Если он делает все самостоятельно, он должен взбить яйцо, подготовить сковородку, растопить масло и, наконец, приготовить омлет. Если он хочет повысить эффективность, он может получить помощь от кухонного работника. Работник может взбить яйцо, позволяя шеф-повару подготовить сковороду и масло, и затем приготовить омлет.

Работнику не разрешается прикасаться к сковороде или готовить омлет – он просто выполняет задачу, в то время как шеф-повар продолжает с другой работой.



Если бы Web Workers мог готовить, вот как бы он помог приготовить омлет

Использование Web Workers такое же. Если JavaScript содержит какие-то интенсивные вычисления с ресурсами, можно передать это Web Worker для обработки, в то время как основной процесс продолжает выполняться. Можно использовать более одного Web Worker, и Web Worker может делать более одной задачи. Давайте приготовим пример, чтобы увидеть это в действии.

Покажите мне просто код!

Сохраняйте спокойствие, мы прибываем на место! Работник (worker)

сам является просто некоторым кодом JavaScript в своем собственном файле. Также как концепция Web Workers является выполнением кода в отдельном потоке, так и код самого работника должен находиться в отдельном файле, или нескольких файлах, если используется больше одного работника. В нашем примере давайте начнем с создания пустого текстового файла, который назовем `worker.js`.

В нашем основном потоке JavaScript мы используем работника, создавая новый объект `Worker`:

Основной поток JavaScript

```
var worker = new Worker('worker.js');
```

Как и с кухонным помощником, мы передаем работнику что-то, он делает что-то с этим в фоновом режиме, и затем что-то нам возвращает. Коммуникация с работником осуществляется с помощью метода `postMessage`:

Основной поток JavaScript

```
// Создаем новый объект работника
var worker = new Worker('worker.js');

// Посылаем простое сообщение, чтобы запустить работника
worker.postMessage();
```

Можно также передать работнику переменную:

Основной поток JavaScript

```
// Создаем новый объект работника
var worker = new Worker('worker.js');

// Посылаем сообщение, чтобы запустить работника
// и передаем ему переменную
var info = 'Web Workers';
worker.postMessage(info);
```

В работнике, т.е. внутри файла `worker.js`, мы используем событие

`onmessage` для получения сообщения из основного потока и выполнения какой-то работы. Если передается переменная, то можно получить к ней доступ с помощью `event.data` следующим образом:

```
worker.js
// Получаем сообщение из основного потока
onmessage = function(event) {
// Выполняем что-то
var info = event.data;
};
```

Отправка сообщений из работника назад в основной поток использует те же методы:

```
worker.js
// Получить сообщение из основного потока
onmessage = function(event) {
// Сделать что-то
var info = event.data;
var result = info + ' rise up!';
postMessage(result);
};
Main JavaScript thread
// Создать новый объект worker
var worker = new Worker('worker.js');

// Послать сообщение, чтобы запустить работника и
// передать ему переменную
var info = 'Web Workers';
worker.postMessage(info);

// Получить сообщение от работника
worker.onmessage = function (event) {
// Сделать что-то
alert(event.data);
};
```

При желании можно загрузить следующий демонстрационный пример использования Web Workers (ссылка: http://dev.opera.com/articles/view/web-

workers-rise-up/WebWorkers_demo.zip - http://dev.opera.com/articles/view/web-workers-rise-up/WebWorkers_demo.zip).

Opera создана как однопоточный браузер с поддержкой множества платформ, поэтому текущая реализация Web Workers разделяет выполнение кода в одном потоке UI. Однако другие браузеры могут иметь мультипоточную архитектуру, которая позволяет одновременное выполнение различных потоков кода.

О чем надо помнить

Это, очевидно, очень простой пример, но когда вы задаете работникам более сложные задачи, такие как управление большими массивами, или вычисление точек в трехмерном пространстве для отображения в основном потоке, то это становится очень мощным средством. Основное, что надо помнить, однако, состоит в том, что работники не могут получить доступ к DOM. В примере выше, например, мы не можем вызвать внутри работника `alert()` или даже `document.getElementById()` - он может только получать и возвращать переменные, хотя это могут быть строки, массивы, объекты JSON, и т.д.

Вот сводка того, что доступно и недоступно для Web Workers.

Могут использовать:

- объект `navigator`
- объект `location` (только чтение)
- метод `importScripts()` (для доступа к файлам сценариев в том же домене)
- объекты JavaScript, такие как `Object`, `Array`, `Date`, `Math`, `String`
- `XMLHttpRequest`
- методы `setTimeout()` и `setInterval()`

Не могут использовать:

- DOM
- Порождающую работника страницу (только через `postMessage()`)

Поддержка браузеров

На момент написания не все браузеры поддерживали Web Workers, поэтому их надо использовать с осторожностью. Вместо того чтобы пытаться отследить, какие версии браузеров поддерживают и не поддерживают, проще включить проверку внутри сценария. Чтобы определить, поддерживает ли браузер пользователя Web Workers, можно проверить существование свойства *`Worker`* объекта `window`:

```
// Проверка, что имеется поддержка Web Workers
if (!!window.Worker) {
// Ура, можно передать утомительную работу!
}
```

Web Workers особенно подходят в тех ситуациях, где нежелательно заставлять пользователя ждать, пока выполняется какой-то код. Основной поток вычислений может сконцентрироваться на работе с UI, отображая его как можно быстрее, в то время как Web Workers могут в фоновом режиме обрабатывать данные, используя AJAX для коммуникации с сервером. Все счастливы, и все прекрасно в стране ScravaJipt.

Дополнительные ссылки по теме Web Workers

- Последняя опубликованная версия спецификации Web Workers (ссылка: http://www.w3.org/TR/workers/ - http://www.w3.org/TR/workers/)
- Реми Шарп, Простая демонстрация Web Workers (ссылка: http://html5demos.com/worker - http://html5demos.com/worker)
- Брэндон Аарон, Пример использования Web Workers (ссылка: http://brandonaaron.net/examples/connecting-the-dots-with-web-workers/1 - http://brandonaaron.net/examples/connecting-the-dots-with-web-workers/1)

- Николас К. Закас, Введение в API Web Workers (ссылка: http://answers.oreilly.com/topic/1358-introducing-the-web-workers-api/ - http://answers.oreilly.com/topic/1358-introducing-the-web-workers-api/)
- Марк Пилгрим, Обнаружение поддержки HTML5 и связанных API JavaScript (ссылка: http://diveintohtml5.org/detect.html - http://diveintohtml5.org/detect.html)

Как использовать API геолокации W3C

В завершающей курс лекции описываются основные возможности API геолокации. Как определяется местоположение пользователя. Стандартные предупреждения безопасности. Способы использования API геолокации в приложениях web. Прямое и обратное геокодирование. Геолокация на примере карт от Google.

Шветанк Диксит · 16 июня 2010 г.

Введение

Представьте себе первую поездку в незнакомый город. Вы голодны, но не знаете, где находится ближайший ресторан, и какой из них хороший. Не правда ли будет удобно, если какое-то приложение сможет определить ваше местоположение и предоставить список ближайших ресторанов, каждый вместе с рецензией и рейтингом?

И это не единственное соображение – сможет ли оно точно определить ваше местоположение, помня также о вашей приватности? Здесь на помощь приходит API геолокации консорциума W3C, поддержка которого включена в Opera 10.60.

Что такое геолокация?

Под "Геолокацией" мы понимаем использование в браузере сценариев для определения, где находится определенный пользователь. Это будет иметь бесчисленные выгоды в приложениях Web и является привлекательным свойством браузеров. Например, можно создать службу Web для предоставления информации о ближайшей больнице или с пошаговым навигационным маршрутом. Или как насчет приложения, которое сообщает обо всех интересных событиях, происходящих в вашей области в ближайшие дни с указанием точного местоположения? Вы взволнованы? Читайте дальше!

API геолокации консорциума W3C

Люди пытались определить местоположение пользователя в Web достаточно давно. Однако большинство старых методов недостаточно точны, и не следят за разрешением пользователя для доступа к этой информации. API геолокации консорциума W3C определен для предоставления разработчикам унифицированного способа создания приложений Web, использующих информацию о местоположении. Он значительно более точен и создан таким образом, что пользователь может контролировать, может ли информация о его местоположении быть доступна на данном сайте или нет.

Как определяется местоположение?

В браузерах, которые поддерживают геолокацию, местоположения пользователя определяются на основе комбинации данных о точках доступа wifi пользователя и IP-адресе пользователя.

Как посетители сайта подтверждают или отвергают доступ к информации о своем местоположении?

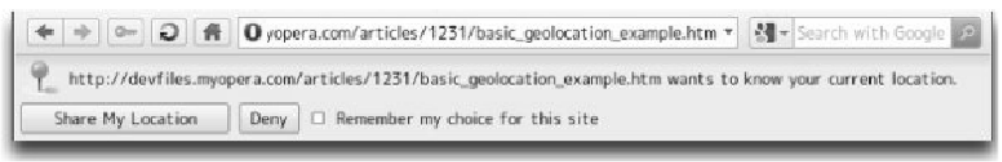


Рис. 11.1. Браузер предупреждает пользователя, что приложение пытается получить доступ к данным местоположения через API геолокации, и запрашивает разрешения на получение доступа

При обращении к странице, которая содержит код геолокации, появится небольшое уведомление вверху страницы, спрашивающее, хотите ли вы предоставить приложению информацию о местоположении или нет (см. рисунок 11.1). Если вы согласны, то информация будет предоставлена; можно также выбрать "запомнить мой выбор для этого сайта", чтобы в будущем больше не задавать этот вопрос, и приложение автоматически использовало бы сделанный в первый раз выбор. Аналогично можно блокировать информацию местоположения, предоставляя пользователю индивидуальный выбор в отношении приватности.

Как это используется в приложении Web?

Использовать API геолокации очень просто. Прежде всего, необходимо определить, что браузер пользователя действительно поддерживает геолокацию. Это можно сделать с помощью JavaScript, чтобы запросить браузер и проверить, что он поддерживает свойство `navigator.geolocation`. Для однократного определения местоположения пользователя, можно использовать функцию `navigator.geolocation.getCurrentPosition()`, которая, в случае успеха, вызовет выбранную функцию обратного вызова.

Можно сделать это следующим образом:

```
//Проверяем, что браузер поддерживает W3C Geolocation API
if (navigator.geolocation) {
navigator.geolocation.getCurrentPosition(successFunction, errorFunction);
} else {
alert('Похоже, что Geolocation, которая требуется для этой страницы, не в
Пожалуйста, воспользуйтесь браузером, который ее поддерживает.');
}
```

В производственной системе, необходимо рассмотреть откат, чтобы пользователи, использующие браузеры без геолокации, могли вручную ввести свое местоположение, например, вводя свой почтовый индекс или выбирая свое положение на карте.

Затем добавляется функция для успешного определения расположения:

```
function successFunction(position) {
var lat = position.coords.latitude;
var mylong = position.coords.longitude;
alert('Ваша широта:'+lat+' и долгота '+mylong);
}
```

Можно написать функцию обработки ошибки любым образом. Обратитесь к интерфейсу PositionError (ссылка: http://dev.w3.org/geo/api/spec-source.html#position_error_interface - http://dev.w3.org/geo/api/spec-source.html#position_error_interface) в спецификации за дополнительной информацией о кодах ошибки. Для

примера мы сохраним ее простой:

```
function errorFunction(position) {
alert('Error!');
}
```

Вот окончательный пример страницы (ссылка: http://devfiles.myopera.com/articles/1231/basic_geolocation_example.htm - http://devfiles.myopera.com/articles/1231/basic_geolocation_example.htm).

Если вы хотите продолжать наблюдение за местоположением пользователя и обновлять соответственно функцию, можно использовать функцию `navigator.geolocation.watchPosition()`. Она получает такие же параметры и работает почти таким же образом как и `getCurrentPosition()`. В обеих функциях `getCurrentPosisition()` и `watchPosition()` необходим только первый параметр. Второй параметр, который обрабатывает ошибку, является необязательным. Кроме того, существует третий необязательный параметр — объект, содержащий атрибуты, определяющие:

- более высокая точность (ссылка: http://www.w3.org/TR/geolocation-API/#high-accuracy - http://www.w3.org/TR/geolocation-API/#high-accuracy)
- время задержки (ссылка: http://www.w3.org/TR/geolocation-API/#timeout_error - http://www.w3.org/TR/geolocation-API/#timeout_error) между определением местоположения и вызовом функции обратного вызова при успехе.
- Максимальное время (ссылка: http://www.w3.org/TR/geolocation-API/#max-age - http://www.w3.org/TR/geolocation-API/#max-age) в течение которого приложение может кэшировать информацию о местоположении до следующего вызова `getCurrentPosition()` или `watchPosition()`.

Что дальше?

Возможность определить местоположение пользователя является очень

полезным свойством. Однако с помощью API вы получаете широту и долготу. Как определить чему соответствует на Земле пара широта/долгота? Здесь на помощь приходит обратное геокодирование.

Обратное геокодирование является процессом поиска, чему в точности соответствует пара широта/долгота. Например, для пары широта/долгота (38.898748, -77.037684) обратное геокодирование сообщает, что это соответствует Белому Дому в Вашингтоне, D.C.

Существует несколько способов выполнения обратного геокодирования, и все они требуют использования некоторой внешней службы или чего-то еще. Для этого можно использовать API GeoNames.org (ссылка: http://www.geonames.org/export/web-services.html - http://www.geonames.org/export/web-services.html), а также API WikiMapia или множество данных, называемое Nominatim (ссылка: http://wiki.openstreetmap.org/wiki/Nominatim#Reverse_Geocoding_.2F_Address_ - http://wiki.openstreetmap.org/wiki/Nominatim#Reverse_Geocoding_.2F_Address_ поддерживаемое добровольцами. Google Maps также предоставляет способ обратного геокодирования пары широта/долгота в реальные адреса для отображения на карте.

Другие API, где можно использовать пару широта/долгота включают API Flickr, который содержит методы для извлечения фотографий, сделанных в определенных координатах, API Meetup.com, и другие.

Находим себя на карте

Давайте быстро просмотрим небольшой пример, который использует Google Maps для отображения карты места, где в данный момент находится пользователь. Мы включаем сначала сценарий для загрузки Google Maps на страницу:

```
<script src="http://maps.google.com/maps/api/js?sensor=false"></script>
```

Затем мы включаем логику для определения местоположения и отображения карты, центрированной по местоположению:

```
<script type="text/javascript">
```

```
// Determine support for Geolocation
if (navigator.geolocation) {
// Locate position
navigator.geolocation.getCurrentPosition(displayPosition, errorFunction);
} else {
alert('Похоже, что Geolocation, которая требуется для этой страницы, не в
Пожалуйста, используйте браузер, который ее поддерживает.');
}

// Функция обратного вызова в случае успеха
function displayPosition(pos) {
var mylat = pos.coords.latitude;
var mylong = pos.coords.longitude;
var thediv = document.getElementById('locationinfo');
thediv.innerHTML = '<p>Your longitude is :' +
mylong + ' and your latitide is ' + mylat + '</p>';

//Загружаем Google Map
var latlng = new google.maps.LatLng(mylat, mylong);
var myOptions = {
zoom: 15,
center: latlng,
mapTypeId: google.maps.MapTypeId.HYBRID
};
var map = new google.maps.Map(document.getElementById("map_canvas"), my

//Добавляем маркер
var marker = new google.maps.Marker({
position: latlng,
map: map,
title:"You are here"
});
}

// Функция обратного вызова в случае ошибки
function errorFunction(pos) {
alert('Error!');
}
</script>
```

Наконец, код HTML и CSS для отображения данных имеет следующий вид:

```
<head>
<style type="text/css">
html, body {
width: 100%;
height: 100%;
}
#map_canvas {
height: 85%;
width: 100%;
}
</style>
</head>
<body>
<div id="map_canvas"></div>
<div id="locationinfo"></div>
</body>
```

Рисунок 11.2 показывает получающийся пример геолокации. Он спрашивает разрешения использовать местоположение, а затем выводит Google Map вашего местоположения.

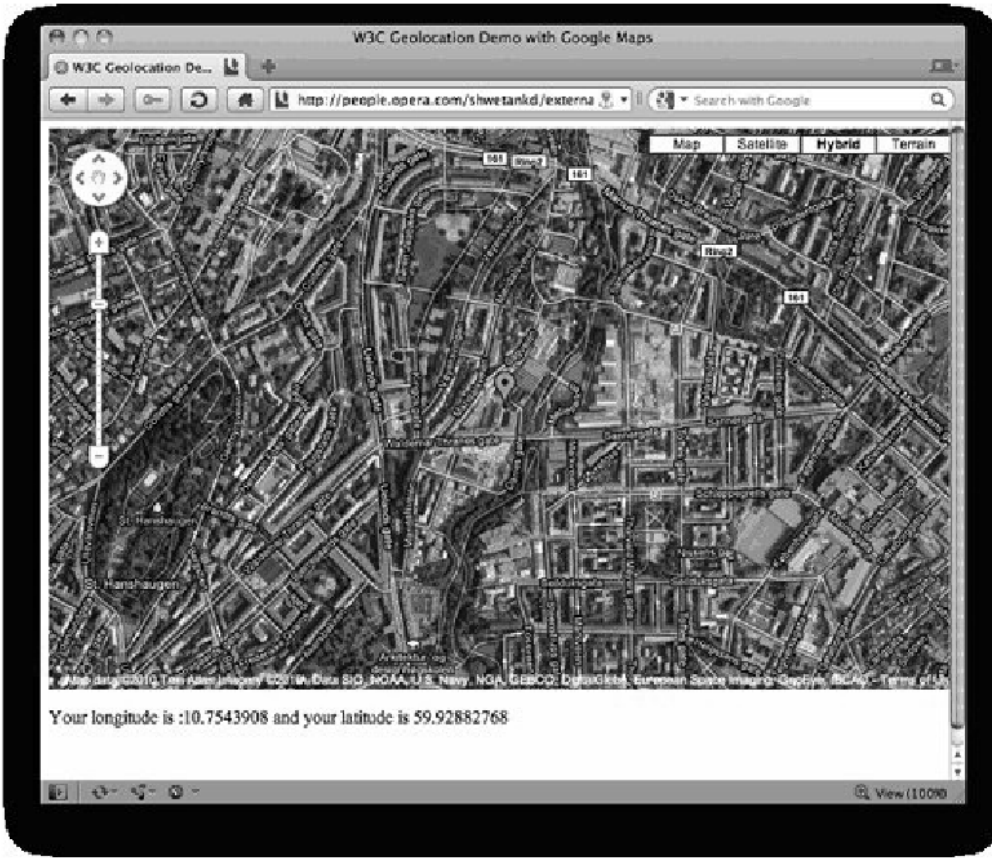


Рис. 11.2. Снимок с экрана страницы, которая показывает ваши координаты и карту, центрированную по этим координатам

На показанном выше снимке с экрана можно заметить справа в адресной строке небольшую иконку, представляющую геолокацию. Эта иконка будет присутствовать на любой странице, на которой используется геолокация. Если вы отклоняете права на геолокацию для страницы, которая ее запрашивает, то иконка не появится в адресной строке этой вкладки. Щелкните на иконке чтобы получить доступ к дополнительным настройкам, связанным с геолокацией, такими как отказ странице в доступе к вашему местоположению.

Заключение

API геолокации W3C предоставляет разработчикам Web интересный и

удобный способ использовать местоположение пользователя в своих приложениях. Сегодня это можно сделать точнее, чем раньше, и обеспечить пользователям более высокий уровень приватности. Теперь задача разработчиков создать приложения, которые в полной мере смогут использовать эти возможности.

Демонстрационные страницы

- Первый пример в статье показывает, как использовать функции для получения пары широта/долгота (ссылка: http://devfiles.myopera.com/articles/1231/basic_geolocation_example.htm - http://devfiles.myopera.com/articles/1231/basic_geolocation_example.htm).
- Второй пример в статье определяет ваше местоположение и использует Google Map для вывода вашего города (ссылка: http://people.opera.com/shwetankd/external/demos/demo_geo_googlemap.h - http://people.opera.com/shwetankd/external/demos/demo_geo_googlemap.h
- Вадим Макеев (ссылка: http://dev.opera.com/author/163437 - http://dev.opera.com/author/163437) создал простой пример, который масштабирует и выводит ваше точное местоположение (ссылка: http://dev.opera.com/articles/view/how-to-use-the-w3c-geolocation-api/GeolocationAPI.htm - http://dev.opera.com/articles/view/how-to-use-the-w3c-geolocation-api/GeolocationAPI.htm), используя API геолокации, Google Maps API v3, and Simple JS (точность, как всегда, будет зависеть от того, в какой степени данные геолокации доступны для вашей области).
- Более сложный пример геолокации (ссылка: http://experimenting.in/other/demo_geo_twitter_mashup.htm - http://experimenting.in/other/demo_geo_twitter_mashup.htm), который находит название вашего местоположения, выводит его карту, перечисляет твиты, упоминающие местоположение, а также все мероприятия, происходящие в ближайшие несколько дней в окрестностях этого местоположения. Используются данные из GeoNames.org (ссылка: http://geonames.org/ - http://geonames.org/), Google Maps (ссылка: http://maps.google.com/ - http://maps.google.com/), Twitter (ссылка: https://www.twitter.com/ - https://www.twitter.com/) и Meetup.com (ссылка: http://www.meetup.com/ - http://www.meetup.com/).

- Демонстрация геолокации в России (http://people.opera.com/pepelsbey/experiments/geo/), также Вадима Макеева, использует Yandex Maps (ссылка: http://maps.yandex.ru/ - http://maps.yandex.ru/) и Twitter (ссылка: https://www.twitter.com/ - https://www.twitter.com/) и работает аналогично предыдущему примеру.

Содержание