A BOOK APART
*Актуальные книги для тех, кто создает сайты*
№ 6

Джереми Кит

# HTML5 ДЛЯ ВЕБ-ДИЗАЙНЕРОВ

Предисловие Джеффри Зельдмана

**Кит Джереми**

# HTML5 для веб-дизайнеров

Серия «Актуальные книги для тех, кто создает сайты», книга 6

*Текст предоставлен издательством*
*http://www.litres.ru/pages/biblio_book/?art=4570175*
*HTML5 для веб-дизайнеров / Джереми Кит: Манн, Иванов и Фербер; Москва; 2013*
*ISBN 978-5-91657-596-5*

## Аннотация

Джереми Кит обладает способностью писать легко и доступно о сложных вещах и сразу выделять те определенно важные моменты, которые имеют значение для дизайнеров-разработчиков. В книге «HTML5 для веб-дизайнеров» он рассказывает о современных тенденциях в области web-разработок.

В HTML5 появилось много интересных тэгов, в том числе поддержка аудио– и видеофайлов. Теперь вам не надо тратить время на установку плагинов для проигрывания музыки или видео – просто воспользуйтесь одним из новых тегов.

Автор убеждает, что можно использовать структурные элементы HTML5 прямо сейчас, например назначить стиль любому элементу, который вы захотите изобрести, или начать использовать доступные вам дополнительные уровни заголовков.

Книга Джереми Кит – настоящая инструкция по использованию HTML5.

# Содержание

# Кит Джереми
# HTML5 для веб-дизайнеров

## Предисловие

Когда мы с Мэнди Браун и Джейсоном Санта-Мария организовали издательство A Book Apart, мы считали особенно животрепещущей одну конкретную тему, и был только один автор, который мог бы с ней справиться.

Ни одна другая тема, ни «полноценные шрифты», ни CSS3, не волнуют сообщество разработчиков, работающих по стандартам, больше, чем неминуемое появление HTML5. Эта новая вариация общего языка веба, зародившаяся из-за неудовлетворенности медленным темпом развития и консервативной политикой W3C, задуманная для Сети, состоящей из приложений (а не только документов), – в равной мере воодушевляет, злит и запутывает сообщество веб-разработчиков.

У Джереми Кита есть уникальная способность разъяснять HTML5 и писать сразу о том, что имеет значение для дизайнеров/разработчиков, стремящихся делать доступный для технологий специальных возможностей и основанный на стандартах дизайн. Джереми уже предельно доступно описал DOM и JavaScript и делает то же самое в этой книге, в которой ровно столько слов и иллюстраций, сколько необходимо.

О HTML5 есть другие книги, а будет их гораздо больше. Появятся написанные техническим языком пятисотстраничные книги для разработчиков приложений, потребности которых во многом стимулировали развитие HTML5. Будут и еще более объемные книги – для разработчиков браузеров, в которых будут даваться решения технических проблем, о которых – хвала небесам! – нам с вами никогда не придется думать.

Но эта книга для вас – человека, который создает контент для веба, который делает осмысленную, семантическую разметку веб-страниц, который разрабатывает доступные для технологий специальных возможностей интерфейсы. Можно назвать эту книгу инструкцией по использованию HTML5. Ее цель – как и всех книг, которые выходят в каталоге A Book Apart, – пролить ясный свет на запутанный предмет, и сделать это быстро, чтобы вы могли сразу вернуться к работе.

*Джеффри Зельдман*

# 1. Краткая история разметки

HTML – связующий язык Всемирной паутины. С помощью простых тегов, которые содержит этот язык, род человеческий сумел создать ошеломительно разнообразную сеть документов, связанных между собой гиперссылками – от Amazon, eBay и Wikipedia до личных блогов и страничек, посвященных котикам, похожим на Гитлера.

HTML5 – последняя на данный момент итерация этого лингва-франка, и хотя это и самое амбициозное изменение в нашем Всеобщем Наречии, но обновляется HTML не впервые. Язык начал развиваться с самого начала.

Как и собственно веб, гипертекстовый язык разметки (HyperText Markup Language, HTML) был детищем сэра Тима Бернерса-Ли, который в 1991 году составил документ под названием HTML Tags, предложив в нем около 20 элементов, которые можно было использовать для написания веб-страниц.

Не сэр Тим придумал использовать теги, состоящие из слов в угловых скобках; такие теги уже существовали в формате SGML (Standard Generalized Markup Language, стандартный обобщенный язык разметки). Вместо того чтобы изобретать новый стандарт, сэр Тим увидел все преимущества того, чтобы разрабатывать язык как надстройку к уже существующему стандарту, – эта тенденция заметна и сейчас, в разработке HTML5.

# От IETF до W3C: путь к HTML 4

Такой вещи, как HTML 1, никогда не было. Первой официальной стала спецификация HTML 2.0, опубликованная IETF (Инженерный совет Интернета, Internet Engineering Task Force). Многие из пунктов появились в этой спецификации потому, что они уже существовали на практике. Например, лидировавший на рынке веб-браузер Mosaic уже в 1994 году позволял авторам веб-страниц вставлять в документы картинки с помощью тега `<img>`. Впоследствии элемент `img` появился в спецификации HTML 2.0.

На смену IETF пришел W3C, Консорциум Всемирной паутины (World Wide Web Consortium), который публиковал последующие обновления стандарта HTML на сайте http://www.w3.org. Во второй половине девяностых появился целый шквал исправлений в спецификации, пока в 1999 году не была наконец опубликована спецификация HTML 4.01.

В этот момент HTML подошел к своей первой развилке.

# XHTML 1: HTML по правилам XML

Следующая после HTML 4.01 версия языка называлась XHTML 1.0. «X» означало «экстремальный», и каждый веб-разработчик, когда начинал произносить название языка, был строго обязан скрещивать руки в форме буквы «X».

Ладно, на самом деле нет. «X» значило eXtensible, «расширяемый», а скрещивать руки, в общем, было необязательно.

Содержимое спецификации XHTML 1.0 было совершенно идентично спецификации HTML 4.01. Не было добавлено никаких новых элементов и атрибутов. Единственная разница заключалась в синтаксисе языка. Если HTML давал авторам большую свободу в том, как писать элементы и атрибуты, то XHTML требовал следовать правилам XML, гораздо более строгого языка разметки, на основе которого W3C строил большинство своих технологий.

Введение более строгих правил было не так уж и плохо. Это способствовало тому, что авторы документов стали вынуждены придерживаться общего стиля написания. Если раньше теги и атрибуты можно было писать прописными, строчными буквами или любой их комбинацией, то для того чтобы документ XHTML 1.0 проходил валидацию, требовалось, чтобы все его теги и атрибуты были написаны в нижнем регистре.

Публикация стандарта XHTML 1.0 совпала с началом поддержки CSS в браузерах. По мере того как веб-разработчики стали принимать только что появившиеся веб-стандарты (это началось с Web Standards Project), более строгий синтаксис XHTML стал рассматриваться как передовая практика в написании разметки.

Потом W3C опубликовал спецификацию XHTML 1.1.

Если XHTML 1.0 – это был простой HTML, пересказанный средствами XML, то XHTML 1.1 стал настоящим XML, беззаветно и полностью. Таким образом, сервер не мог отдавать его с MIME-типом `text/html`. Но если же авторы публиковали документ с MIME-типом XML, то самый распространенный браузер в мире на тот момент – Internet Explorer – вовсе не мог отобразить документ.

Казалось, что W3C стал утрачивать чувство реальности и того, что действительно происходит с публикацией документов в вебе.

# XHTML 2: терпению пришел конец

Если бы персонаж Дастина Хоффмана в фильме «Выпускник» был веб-разработчиком, W3C сказал бы ему одно слово, ровно одно: XML.

С точки зрения W3C разработка HTML закончилась на версии 4. Они начали работать над XHTML 2, который был спроектирован так, чтобы привести веб к светлому, основанному на XML будущему.

И хотя название XHTML 2 звучало достаточно похоже на XHTML 1, между ними не было ничего общего. В отличие от XHTML 1, в XHTML 2 не предусмотрено обратной совместимости с существующим веб-содержимым и даже с предыдущими версиями HTML. Это должен быть чистый язык, неотягощенный неряшливой историей предыдущих спецификаций.

Наступила полная катастрофа.

# Раскол: WHATWG TF?

Внутри W3C назрело восстание. Со стороны казалось, что консорциум формулировал теоретически чистые стандарты, никак не связанные с нуждами веб-разработчиков. Представители Opera, Apple и Mozilla были недовольны этим направлением развития. Им хотелось, чтобы большее внимание уделялось технологиям, позволяющим создавать веб-приложения.

Конфликт перешел в критическую фазу на семинаре в 2004 году. Ян Хиксон (Ian Hickson), который в то время работал в Opera Software, предложил идею расширения HTML с целью сделать возможным создание веб-приложений. Это предложение было отвергнуто.

Недовольные повстанцы организовали свою собственную группу: рабочую группу по разработке гипертекстовых приложений для веба (Web Hypertext Application Technology Working Group, или сокращенно WHATWG).

# От веб-приложений 1.0 к HTML5

С самого начала WHATWG стала работать совершенно не так, как W3C. В W3C использовался подход, основанный на согласии: вопрос поднимается, обсуждается, затем по нему голосуют. В WHATWG вопросы тоже поднимаются и обсуждаются, а окончательное решение по тому, что войдет в спецификацию, а что нет, принимает редактор – Ян Хиксон.

В теории процесс W3C выглядит более демократическим и честным. На практике же политические распри и внутренние перебранки могут очень сильно замедлять продвижение. В WHATWG, где кто угодно может внести свое предложение или мнение, но последнее слово остается за редактором, все движется быстрее. Но и у редактора все же нет абсолютной власти: собранный по личным приглашениям управляющий комитет может запустить процедуру импичмента редактора в маловероятном случае, если он поведет себя как доктор Стрейнджлав.

Сначала основной объем работы в WHATWG был разбит на две спецификации: веб-форм (Web Forms 2.0) и веб-приложений (Web Apps 1.0). Обе спецификации должны быть расширениями для HTML. Со временем они объединились в одну спецификацию, которая называлась просто HTML5.

# Объединение

Пока в WHATWG разрабатывали HTML5, W3C продолжала работать над спецификацией XHTML 2. Нельзя сказать, что она летела по шоссе в никуда. Она ехала в никуда очень-очень медленно.

В октябре 2006 года сэр Тим Бернерс-Ли написал пост в блоге, в котором признал, что попытка заставить веб перейти с HTML на XML не имеет шансов на успех. Несколько месяцев спустя W3C выпустил новый договор для рабочей группы HTML. Вместо того чтобы начинать с нуля, они мудро решили, что в качестве фундамента для любой будущей версии HTML нужно использовать наработки WHATWG.

Эта остановка и новый запуск привели к несколько запутанной ситуации. Получилось, что W3C одновременно работал над двумя разными, несовместимыми типами разметки: XHTML 2 и HTML 5 (обратите внимание на пробел перед пятеркой). В то же время отдельная организация, WHATWG, работала над спецификацией под названием HTML5 (без пробела), которая должна быть использована в качестве основы для одной из спецификаций W3C!

Если вы относитесь к тем веб-разработчикам, которые пытаются в этом разобраться, то знайте, что проще расшифровать смысл фильмов «Мементо», «Детонатор» и всей фильмографии Дэвида Линча, даже если смотреть их подряд.

# XHTML умер: да здравствует синтаксис XHTML

Туман неразберихи начал рассеиваться в 2009 году. W3C объявил, что договор на XHTML 2 не будет продлеваться. Формат был мертвым уже несколько лет, и это объявление стало только официальным свидетельством о смерти.

Как ни странно, смерть XHTML 2 не прошла незамеченной. Напротив, противники XML отреагировали на нее злорадно и использовали это объявление для того, чтобы высмеять всех, кто когда-либо использовал XHTML 1, – даже несмотря на то, что между XHTML 1 и XHTML 2 не было практически ничего общего.

В то же время авторы, которые писали на XHTML 1 с тем, чтобы следовать более строгому стилю написания кода, стали волноваться, что HTML5 будет означать возвращение к небрежной разметке.

Как вы скоро увидите, это не обязательно так. Вы можете писать на HTML5 и небрежно, и строго – как захотите.

# Развитие HTML5

Текущее состояние HTML5 не такое запутанное, как было когда-то, но все равно не до конца понятное.

Над HTML5 работают две группы. WHATWG создает спецификацию HTML5 в рамках процесса «утвердить, потом пересмотреть». Рабочая группа по HTML W3C берет эту спецификацию и проводит ее через процесс «пересмотреть, потом утвердить». Как вы легко можете представить, это непростой политический союз. По крайней мере, кажется, наконец появилось единодушие по этому назойливому вопросу – с пробелом или без пробела? (На тот случай, если вам вдруг интересно, HTML5 решили писать без пробела.)

Пожалуй, самая сбивающая с толку проблема для тех веб-разработчиков, которые пробуют кончиком ноги воду HTML5, – ответ на вопрос «когда он будет готов?»

Ян Хиксон в интервью сказал, что ожидает, что HTML5 получит статус предложенной рекомендации[1] в 2022 году. За этим последовала волна общественного негодования от ряда веб-разработчиков. Они не понимали, что значит «предложенная рекомендация», но уж точно знали – на руках нет столько пальцев, чтобы пересчитать, сколько лет пройдет до 2022 года.

Для негодования не было повода. В данном случае для того чтобы получить статус «предложенной рекомендации», нужно иметь две полных реализации HTML5. Учитывая объем спецификации, эта дата невероятно амбициозна. В конце концов, у браузеров не самая лучшая репутация в плане реализации существующих стандартов. Internet Explorer потребовалось больше десятилетия, чтобы добавить поддержку – всего-то – элемента `abbr`.

Та дата, которая действительно имеет значение для HTML5, – 2012 год. В этом году спецификация должна стать кандидатом в рекомендации. Это на жаргоне стандартов значит «сделано и отшлифовано».

Но даже эта дата не особенно важна для веб-разработчиков. Имеет значение тот момент, когда браузеры начнут поддерживать функциональность HTML5. Мы начали использовать части спецификации CSS 2.1, как только начали выпускаться браузеры с поддержкой этих частей. Если бы мы ждали, пока каждый браузер начнет полностью поддерживать CSS 2.1, и только потом стали его использовать, мы ждали бы до сих пор.

С HTML5 ровно то же самое. Не будет никакого четкого момента, когда мы могли бы объявить, что язык готов к использованию. Однако, мы можем начинать использовать части спецификации по мере того, как веб-браузеры начинают поддерживать эти функции.

Помните, что HTML5 – это не новый язык, созданный с нуля. Это скорее эволюционное, чем революционное изменение в продолжающейся истории разметки. Если сейчас вы создаете сайты на любой версии HTML, вы уже используете HTML5.

[1] Описание различных этапов, которые проходят рекомендации W3C, см.: http://ru.wikipedia.org/wiki/Рекомендации_W3C. *Прим. ред.*

# 2. Устройство HTML5

Эпоха Великой французской революции стала временем огромных политических и социальных перемен. Революционному пылу было подвластно и само время. На короткий период Французская республика ввела десятичную систему измерения времени: каждый день разделялся на десять часов, а каждый час – на сто минут. Это была очень логичная система, во всех отношениях превосходящая шестидесятеричную.

Десятичное время было полным провалом. Никто не стал использовать эту систему. То же можно сказать о XHTML 2. W3C на своем опыте усвоила урок революционной Франции: изменять существующее поведение в высшей степени сложно.

# Принципы устройства

WHATWG, намеревавшаяся избежать ошибок прошлого, обозначила ряд принципов и правил для разработки HTML5. Один из ключевых таких принципов: «поддерживать существующее содержимое». Это означает, что с HTML5 не начинается новая эра.

Если XHTML 2 намеревался отбросить все то, что было до него, то HTML5 основывается на уже существующих спецификациях и реализациях. Бóльшая часть HTML 4.01 вошла в HTML5.

В числе других принципов разработки есть, например, такие: «Не изобретайте велосипед» и «Асфальтируйте тропинки» – то есть если среди веб-разработчиков есть широко распространенный способ выполнять задачу – даже если он необязательно лучший, – именно этот способ должен быть записан в HTML5. Сказать по-другому: «Если ничего не ломается, не начинай чинить».

Многие из этих принципов дизайна могут быть вам знакомы, если вы когда-либо заглядывали в сообщество по микроформатам (http://microformats.org). HTML5-сообщество разделяет этот же самый прагматичесный подход: нужно запустить формат в реальный мир, а не волноваться слишком сильно из-за теоретических проблем.

Эта точка зрения четко выражена в принципе устройства, озаглавленном «Приоритет пользователей», в котором сказано: «В случае конфликта ставьте интересы пользователей выше разработчиков, разработчиков – выше конкретных реализаций, реализации – выше спецификаций, спецификации – выше теоретической чистоты».

Ян Хиксон несколько раз говорил, что настоящие судьи в споре того, что окажется в HTML5, а что нет, – разработчики браузеров. Если компания-разработчик браузера откажется поддерживать какое-либо предложение, нет смысла добавлять это предложение в спецификацию, потому что тогда спецификация окажется всего лишь фиктивным документом. Согласно приоритету пользователей наш (веб-разработчиков) голос имеет еще больший вес. Если мы откажемся использовать какую-либо часть спецификации, это также сделает спецификацию фиктивной.

# Ближе к реальности

Определяющим фактором в разработке HTML5 стало постоянное внутреннее напряжение. С одной стороны, эта спецификация должна быть достаточно мощной, чтобы поддерживать создание веб-приложений. С другой стороны, HTML5 должна поддерживать существующее содержимое даже с учетом того, что бóльшая часть существующего содержимого – неудобоваримая каша. Если спецификация слишком отклонится в одном направлении, ее постигнет та же судьба, что и XHTML 2. Но если она пойдет слишком далеко в другом направлении, тогда выйдет, что спецификация будет рекомендовать использование тегов `<font>` и таблиц для разметки – поскольку в конце концов именно с использованием этих приемов построено огромное количество веб-страниц.

Здесь нужно выдержать очень тонкий баланс, и это требует прагматического, уравновешенного подхода.

# Обработка ошибок

Спецификация HTML5 не просто объявляет, что должны делать браузеры, когда они обрабатывают синтаксически правильную разметку. Впервые за всю историю HTML спецификация также объявляет, что́ браузеры должны делать, когда им встречаются документы с ошибками разметки.

До сих пор разработчикам браузеров приходилось разбираться, что делать с ошибками, каждому самостоятельно. Обычно это означало, что нужно было применять реверс-инжиниринг и реализовывать примерно то, что делал в случае ошибок самый популярный браузер. Не самое продуктивное использование времени разработчиков браузеров. Гораздо лучше было бы не тратить время на то, чтобы дублировать то, как конкурент обрабатывает ошибочную разметку, а разрабатывать вместо этого новые функции.

Определение того, как нужно обрабатывать ошибки, в HTML5 – невероятно амбициозная задача. Даже если бы в HTML5 были только элементы и атрибуты из HTML 4.01, без добавления каких бы то ни было новых возможностей определить то, как нужно обрабатывать все ошибки, к 2012 году, – все равно было бы геркулесовым трудом.

Возможно, обработка ошибок не очень-то заинтересует веб-разработчиков, особенно если мы сразу настраиваемся на то, что пишем валидные и синтаксические корректные документы, но для разработчиков браузеров это очень важно. Если предыдущие спецификации разметки писались для авторов, то HTML5 написан и для авторов, и для разработчиков реализаций. Держите это в уме, когда штудируете спецификацию. Это объясняет, почему спецификация HTML5 настолько велика и почему она написана с таким уровнем детализации, который, кажется, обычно пишется для филателистов, любящих играть в шахматы, перебирая свою коллекцию игрушечных поездов.

# Доктайп, скажите честно, я буду жить?

Декларация типа документа, или сокращенно «доктайп», обычно используется для того, чтобы определить, какой именно версией разметки написан документ.

Доктайп для HTML 4.01 выглядит так (переносы строки обозначены»):

```
<!DOCTYPE HTML PUBLIC"-//W3C//DTD HTML 4.01//EN""http://
www.w3.org/TR/html4/strict.dtd">
```

Вот доктайп XHTML 1.0:

```
<!DOCTYPE html PUBLIC"-//W3C//DTD XHTML 1.0 Strict //
EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Не сильно человекочитаемо, но по-своему эти доктайпы просто говорят: «этот документ написан на HTML 4.01» и «и этот документ написан на XHTML 1.0».

Наверное, вы ожидаете, что в доктайпе, объявляющем «этот документ написан на HTML5», где-то будет цифра «пять». Не будет. Доктайп для HTML5 выглядит так:

```
<!DOCTYPE html>
```

Он настолько короткий, что я даже могу его запомнить.

Но это же безумие! Если в доктайпе нет номера версии, как мы сможем определить следующие версии HTML?

Когда я в первый раз увидел доктайп HTML5, я подумал, что это верх гордыни. «Неужели они действительно думают, – спросил я себя, – что это будет последняя спецификация разметки, написанная на Земле?»

В общем, казалось, что это случай из учебника по мышлению «с нуля».

На самом деле, однако, доктайп HTML5 весьма прагматичен. Так как HTML5 должен поддерживать существующее содержимое, этот доктайп может быть применен и к существующему документу на HTML 4.01 или XHTML 1.0. Любая будущая версия HTML тоже должна будет поддерживать существующее содержимое, написанное на HTML5, так что сам концепт применять номера версий к документам разметки имеет значительный изъян.

На деле доктайпы не имеют принципиального значения. Например, вы поставили в документ доктайп HTML 4.01. Если в этом документе окажется элемент из другой спецификации – например, из HTML 3.2 или из HTML5, – браузер все равно отобразит эту часть документа. Браузеры поддерживают функциональность, а не доктайпы.

Декларации типа документа предназначались не для браузеров, а для валидаторов. Единственный случай, в котором браузер обращает какое-либо внимание на доктайп, – когда он «переключает доктайп», – это маленький умный хак, который переключает режим отображения между нестандартным (quirks mode) и стандартным режимами в зависимости от присутствия подходящего доктайпа.

Минимальная информация, необходимая для того, чтобы браузер точно отобразил страницу в стандартном режиме, – и есть доктайп HTML5. На самом деле это вообще единственная причина включать какой-либо доктайп. HTML-документ без доктайпа HTML5 все равно вполне может быть валидным HTML5.

# Будем проще

Доктайп – не единственная вещь, оказавшаяся упрощенной в HTML5.

Если вы хотите особо указать кодировку вашего документа разметки, лучший способ сделать это – проверить, что ваш сервер посылает правильный HTTP-заголовок `Content-Type`. Если вы хотите быть вдвойне уверенным, можно также определить кодировку с помощью тега `<meta>`. Вот как выглядит декларация meta для документа, написанного на HTML 4.01:

```
<meta http-equiv="Content-Type" content="text/html;charset=UTF-8">
```

Вот гораздо более легкий для запоминания способ сделать то же самое в HTML5:

```
<meta charset="UTF-8">
```

Как и с доктайпом, это упрощенное объявление кодировки содержит минимальный набор символов, который необходим браузерам для правильной интерпретации.

Тег `<script>` – еще одно место, где мы можем позволить себе немножко сбросить вес. Обычная практика – добавлять к элементам `script` атрибут `type` со значением `"text/javascript"`:

```
<script type="text/javascript" src="file.js"></script>
```

Браузерам этот атрибут не нужен. Они и так примут за данность, что этот скрипт написан на JavaScript, самом популярном языке скриптов в вебе (давайте будем честными – на единственном языке скриптов в вебе):

```
<script src="file.js"></script>
```

Точно также не нужно указывать значение `type` – `"text/css"` каждый раз, когда вы делаете ссылку на CSS-файл:

```
<link rel="stylesheet" type="text/css" href="file.css">
```

Можно просто написать:

```
<link rel="stylesheet" href="file.css">
```

# Синтаксис: размечайте, как хотите

Некоторые языки программирования, например Python, обязывают писать инструкции специфическим образом. Обязательно использовать пробелы для отступа кода – пробелы и переносы строк имеют значение. Другие языки программирования (например, JavaScript) не обращают никакого внимания на форматирование – сколько пробелов в начале строки, совершенно неважно.

Если хотите бесплатно развлечься вечером, соберите в одной комнате несколько программистов и произнесите слова: «табы или пробелы». Ближайшие несколько часов можете греться от жарких споров, которые разгорятся немедленно.

В сердце спора о значимых пробелах лежит фундаментальный философский вопрос: должен ли язык навязывать определенный стиль написания кода – или авторы должны иметь возможность писать в любом стиле, в каком хотят?

Пробелы и переносы строк не важны для разметки. Если вы хотите ставить перенос строки и отступ при каждом вложенном элементе, пожалуйста, но ни браузеры, ни валидаторы этого не требуют. Это не значит, впрочем, что разметку можно писать совсем уж как угодно. Некоторые версии разметки обязывают к более строгому стилю написания, чем другие.

До XHTML 1.0 не имело никакого значения, пишете вы теги в верхнем или нижнем регистре. Не имело значения, закавычивали вы атрибуты или нет. Для некоторых элементов даже не имело значения, ставите ли вы закрывающий тег.

XHTML 1.0 обязывает следовать синтаксису XML. Все теги должны быть написаны в нижнем регистре. Все атрибуты должны быть в кавычках. У всех элементов должен быть закрывающий тег.

В особенном случае самостоятельных элементов, например `br`, требование закрывающего тега заменяется требованием закрывающей косой черты: `<br/>`.

В случае HTML5 все подходит. Прописные, строчные буквы, в кавычках, без кавычек, самозакрывающиеся элементы или нет – решение здесь полностью за вами.

Я использовал доктайп XHTML 1.0 в течение многих лет. Мне нравится, что я должен писать в каком-то одном специфическом стиле, и мне нравится, что валидатор W3C обязывает меня писать в этом стиле. Теперь, когда я использую HTML5, я сам должен обязать себя писать в том стиле, в каком хочу.

Я понимаю, почему некоторым людям не нравится нетребовательность синтаксиса HTML5. Получается, что мы как будто закрываем глаза на годы, за которые накопились передовые практики. Некоторые даже говорят, что нестрогий синтаксис HTML5 поощряет плохую разметку. Я не думаю, что это так, но могу понять, почему это причина для волнения. Случилось то же самое, как если бы язык программирования, который обязывал использовать значимые пробелы и переводы строк, внезапно переключился бы на правила, которые позволили бы делать это не всегда, а с какими-то исключениями.

Лично у меня нет проблем с бессистемностью синтаксиса HTML5. Я смирился с тем, что мне придется самому обязывать себя писать так, как я хочу. Но мне хотелось бы видеть больше инструментов, которые позволили бы мне проверять, насколько моя разметка соответствует тому или иному стилю. В мире программирования такие инструменты называются «линтерами» – программы, которые отмечают ненадежные места в коде. Линтер для разметки отличается от валидатора, который проверяет соответствие разметки доктайпу; но было бы замечательно, если бы оба они могли быть соединены в одну подкачавшуюся и готовую работать машину для линтирования и валидации.

Кто напишет такую программу, заслужит вечное уважение и восхищение веб-разработчиков по всему миру.

## Мы так не разговариваем

В предыдущих версиях HTML, когда из спецификации удалялся ранее существовавший элемент или атрибут, этот процесс назывался исключением (deprecation). Веб-разработчикам рекомендовалось не использовать исключенный элемент, не посылать ему открытки на Новый год и вообще не говорить о нем в приличном обществе.

В HTML5 нет исключенных элементов или атрибутов. Но зато есть огромное количество устаревших элементов и атрибутов.

Нет, это не очередной случай выжившей из ума политкорректности. «Устаревший» имеет несколько иное значение, чем «исключенный».

Поскольку HTML5 стремится быть обратно совместимым с существующим контентом, спецификация должна учитывать существующие элементы даже в том случае, если эти элементы больше не входят в HTML5. Это приводит к несколько странной ситуации, когда в спецификации написано «авторы, не используйте этот элемент», а дальше «браузеры, вы должны отображать этот элемент вот так». Если бы элемент был исключен, он вовсе не упоминался бы в спецификации, но поскольку элемент является устаревшим, он включается в спецификацию для браузеров.

Если вы не разрабатываете браузер, вы можете относиться к устаревшим элементам и атрибутам так же, как относились бы к исключенным: не используйте их на веб-страницах, не приглашайте их на коктейль-вечеринки.

Если вы будете настаивать на использовании устаревшего элемента или атрибута, ваш документ станет «несоответствующим». Браузеры будут отображать все как и прежде, но вы, может случиться, заметите, что сайты по соседству поглядывают на вас неодобрительно.

## Было приятно познакомиться, чао

Устаревшими стали элементы `frame`, `frameset` и `noframes`.

Никто не будет по ним скучать.

Устарел элемент `acronym`, освободив таким образом годы времени на споры, которые можно использовать с бóльшим толком: хотя бы рассчитать наконец теоретически возможную плотность одновременного количества ангелов на булавочной головке стандартного размера[2]. Не плачьте по элементу `acronym` – используйте вместо него элемент `abbr`. Да, я знаю, что между акронимами и аббревиатурами есть разница: акронимы произносятся как одно целое слово (например: НАТО, ЮНЕСКО), но просто запомните: все акронимы – аббревиатуры, но не все аббревиатуры – акронимы.

Элементы, относящиеся исключительно к представлению, такие как `font`, `big`, `center` и `strike`, также являются устаревшими в HTML5. В действительности они являются устаревшими уже несколько лет: гораздо проще добиться того же самого эффекта в оформлении с помощью CSS-свойств: например, `font-size` и `text-align`. Точно также атрибуты, относящиеся к представлению: `bgcolor`, `cellspacing`, `cellpadding` и `valign`, являются устаревшими. Просто используйте CSS.

---

[2] В крупнейшем схоластическом труде Средневековья, «Сумме теологии» Фомы Аквинского, содержится ряд логических умозаключений о природе мира, Бога и в том числе ангелов (например: «Может ли ангел переместиться из одной точки в другую, не проходя нигде в середине между ними?»). Мыслители эпохи Просвещения, критиковавшие абсурдные с их точки зрения построения томизма, сочинили иронический «вопрос»: «Сколько ангелов могут одновременно танцевать на кончике иглы, не задевая друг друга?» Хотя у Фомы Аквинского нигде нет подобного образа, схожий («в раю тысяча душ может поместиться на кончике одной иглы») встречается в одном из немецких мистических текстов XIV в. *Прим. перев.*

Не все элементы, относящиеся к представлению, являются устаревшими. Некоторые из них прошли процесс профессиональной переподготовки, и теперь у них есть еще один шанс.

## Перемен, мы ждем перемен!

Элемент `big` является устаревшим, а вот элемент `small` – нет. Чтобы это не выглядело непоследовательным, было решено переопределить, что значит в данном случае «маленький». Раньше мы понимали «маленький» как термин, связанный только с представлением: «это нужно отображать шрифтом маленького размера». Вместо этого появилось семантическое значение: «это то, что набирается мелким шрифтом», то есть текст для юридических нюансов или условий использования.

Конечно, в девяти случаях из десяти вы будете отображать «мелкий шрифт» как раз маленьким шрифтом, но смысл в том, что чисто оформительское значение элемента уступило место семантическому.

Элемент `b` раньше означал: «это нужно отобразить полужирным шрифтом». Теперь его можно использовать, чтобы текст «стилистически отличался от обычного текста, не передавая при этом семантики дополнительной важности». Если этот фрагмент текста более важен, чем окружающий текст, тогда больше подойдет элемент `strong`.

Точно также элемент `i` не значит больше «отобразить текст курсивом». Теперь этим элементом описывается текст, «произнесенный другим голосом или с другим настроением». Опять же, этот элемент не предполагает дополнительной важности или акцента на текст. Если вы хотите, чтобы акцент был, используйте элемент `em`.

Эти изменения могут показаться простой игрой в определения. Отчасти это так и есть, но, кроме этого, они повышают независимость HTML5 от конкретных устройств. Когда вы представляете себе слова «жирный» или «курсив», то ясно, что они имеют смысл только в визуальной среде – например, на экране или на странице. Убрав перекос в сторону визуального из определений этих элементов, спецификация остается релевантной и для устройств, лишенных визуального слоя: например, для программ, читающих с экрана. Эти изменения также побуждают разработчиков думать не только о визуальных средах отображения документов.

## Анонимная цитата

В HTML5 изменено определение элемента `cite`. Раньше он означал «отсылку к другим источникам», а теперь – «название работы, к которой идет отсылка». Достаточно часто ссылка на источник цитаты и есть название работы (скажем, книги или фильма), но настолько же часто источником может быть и человек. До HTML5 вы могли разметить имя этого человека с помощью `cite`. Теперь это однозначно запрещено – прощай, обратная совместимость.

Оправдывают это изменение примерно следующим: браузеры выделяют текст внутри тега `<cite>` курсивом; названия работ обычно выделяют курсивом[3], а имена людей – нет, таким образом, элемент `cite` не должен использоваться для того, чтобы размечать имена авторов.

Это просто неправильно. Я полностью за то, чтобы HTML5 ориентировалась на существующие в браузерах реалии, но здесь явный случай того, когда хвост виляет собакой.

К счастью, ни один валидатор не отличит, относится ли текст между открывающими и закрывающими тегами `<cite>` к человеку или нет, так что ничто не мешает нам,

[3] В англоязычной традиции. Отсутствие такой практики на других языках (в частности, на русском) – еще один аргумент в пользу точки зрения автора. *Прим. перев.*

веб-разработчикам, использовать элемент `cite` имеющим смысл образом, к тому же поддерживающим обратную совместимость.

## Элемент a на стероидах

Если изменения в уже существующих элементах включают в себя креативную игру в определения, один элемент в HTML5 обновился полностью.

Элемент `a`, без сомнения, самый важный элемент в HTML. Он превращает наш текст в гипертекст. Это соединительная ткань Всемирной паутины.

Элемент `a` всегда был строчным (inline) элементом. Если вы хотели сделать заголовок и абзац гиперссылками, нужно было использовать несколько элементов `a`:

```
<h2><a href="/about">Обо мне</a></h2>
<p><a href="/about">Узнайте, почему я такой.</a></p>
```

В HTML5 вы можете обернуть несколько элементов в один элемент `a`:

```
<a href="/about">
<h2>Обо мне</h2>
<p>Узнайте, почему я такой.</p>
</a>
```

Единственная оговорка – вы не можете поместить элемент `a` внутри другого элемента `a`.

Может показаться, что оборачивать несколько элементов в один элемент `a` – очень серьезное изменение, но большинству браузеров не придется очень много делать для того, чтобы поддерживать эту новую модель ссылок. На самом деле они уже поддерживают ее – даже несмотря на то, что такая разметка вплоть до HTML5 технически никогда не была разрешенной.

Это кажется немножко противоречащим здравому смыслу: наверное, браузеры должны реализовывать уже имеющуюся спецификацию? Но получается наоборот: новейшая спецификация документирует то поведение браузеров, которое уже наличествует.

# Новые игрушки! API JavaScript

Если вы хотите почитать документацию по CSS, то отправляетесь смотреть спецификацию CSS. Если ищете документацию по разметке, обращаетесь к спецификации HTML. Но где можно найти спецификацию по различным API JavaScript, таким как `document.write`, `innerHTML` и `window.history`? Спецификация JavaScript касается только языка программирования – вы не найдете в ней никаких браузерных API.

Вплоть до настоящего момента браузеры создавали и реализовывали API JavaScript независимо друг от друга, заглядывая друг другу через плечо, чтобы посмотреть, что делают другие. HTML5 задокументирует эти API раз и навсегда, что должно обеспечить лучшую совместимость.

Кажется странным, что документация по JavaScript находится в спецификации разметки, но не забывайте, что HTML5 начал свое существование как спецификация для веб-приложений (Web Apps 1.0). JavaScript – неотъемлемая часть разработки веб-приложений.

Ряд разделов спецификации HTML5 целиком посвящен новым API для создания веб-приложений. Описан, например, менеджер отмены (`UndoManager`), который позволяет браузеру отслеживать изменения документа. Есть отдельный раздел по созданию офлайновых веб-приложений с помощью использования манифеста кэширования. Детально описан процесс перетаскивания объектов.

Как всегда, если уже существует реализация, спецификация будет опираться на нее, а не изобретать велосипед. В Internet Explorer уже несколько лет существует API для перетаскивания объектов, поэтому она и стала фундаментом для перетаскивания в HTML5. К сожалению, у API Microsoft – как бы помягче сказать – есть свои проблемы. Может быть, иногда не так уж плохо заново изобретать велосипед, если у тебя есть только велосипед с квадратными колесами.

API в HTML5 могут очень многое. И еще они полностью за гранью моего понимания. Я предоставлю возможность писать о них разработчикам, которые умнее меня. Эти API заслуживают своей собственной, отдельной книги.

В то же время в HTML5 есть еще очень много нового, что приведет нас, веб-разработчиков, в полный восторг. И этот восторг начинается прямо в следующей главе.

# 3. Мультимедиа

История веба пестрит технологическими нововведениями. Одним из самых ранних добавлений к HTML стал элемент `img`, и это фундаментально изменило веб. Затем появление JavaScript позволило вебу стать более динамической средой. Наконец, быстрый рост количества Ajax-приложений позволил вебу стать средой, в которой возможны полноценные приложения.

Веб-стандарты продвинулись настолько, что сейчас возможно разработать практически все что угодно с помощью HTML, CSS и JavaScript.

В разнообразном наборе веб-стандартов есть пробелы. Если вы хотите опубликовать текст и картинки, вам ничего не нужно, кроме HTML и CSS. Но если вы хотите опубликовать аудио и видео, вам понадобится технология, существующая в виде плагинов, – например, Flash или Silverlight.

«Плагин» (дословно «подключаемый модуль») – вполне точный термин для этого типа технологий: они помогают заткнуть дырки в вебе. С помощью плагинов относительно просто выложить онлайн-игры, фильмы и музыку. Но эти технологии не являются открытыми. Они не создаются сообществом разработчиков, а контролируются отдельными компаниями.

Flash – мощная технология, но когда используешь ее, иногда кажется, что ты заключил договор с дьяволом. Мы получили возможность публиковать мультимедиа в вебе, но при этом в какой-то степени потеряли независимость.

HTML5 заполняет эти пробелы. По сути, язык становится прямым конкурентом проприетарных технологий (таких, как Flash и Silverlight). Но вместо того чтобы быть зависимыми от плагинов, мультимедиа-элементы в HTML5 являются встроенными в браузеры.

# Canvas

Когда браузер Mosaic добавил возможность встраивать на веб-страницы картинки, это дало вебу огромный импульс для развития. Но с тех пор картинки оставались статическими. Вы можете создать анимированные гифки. Можете обновлять стили картинки при помощи JavaScript. Можете генерировать картинку динамически на сервере. Но после того как браузер загрузил картинку, ее содержимое больше нельзя обновить.

Элемент `canvas` – среда для создания динамических картинок.

Сам по себе элемент очень прост. Все, что вам нужно определить в открывающем теге, – его размеры:

```
<canvas id="my-first-canvas" width="360" height="240">
</canvas>
```

Если вы напишете что-нибудь между открывающим и закрывающим тегом, то это будет отображено только в тех браузерах, которые не поддерживают работу с Canvas (**рис. 3.01**):

```
<canvas id="my-first-canvas" width="360" height="240">
<p>Браузер не поддерживает canvas? Тогда картинка,по-старинке:</p>
<img src="puppy.jpg" alt="очаровательный щенок">
</canvas>
```

Браузер не поддерживает canvas? Тогда картинка по старинке:

**Рис. 3.01.** Пользователи, браузеры которых не поддерживают canvas, увидят картинку очаровательного щенка

Вся сложная работа делается на JavaScript. Сначала вам нужно будет создать переменную, указывающую на Canvas и его контекст. Слово «контекст» в данном случае означает просто API. В настоящий момент контекст есть только один – двумерный:

```
var canvas = document.getElementById('my-first-canvas');
var context = canvas.getContext('2d');
```

Теперь вы можете начать рисовать на двумерной поверхности элемента canvas, используя API, задокументированное в спецификации HTML5 по адресу: http://bkaprt.com/html5/.[4]

В 2D API есть довольно большое количество тех же самых инструментов, которые есть в графическом редакторе (например, Adobe Illustrator), – обводка, заливка, градиент, тень, формы, кривые Безье. Разница в том, что вместо того чтобы использовать графический интерфейс, вам нужно писать все на JavaScript.

## Танец вокруг архитектуры: как рисовать с помощью кода

Вот так вы определяете, что цвет обводки должен быть красным:

```
context.strokeStyle = '#990000';
```

Теперь у всего, что вы нарисуете, будет красный контур. Например, если вы хотите нарисовать прямоугольник, используйте такой синтаксис:

```
strokeRect (left, top, width, height)
```

Если вы хотите нарисовать прямоугольник размерами 100×50 пикселей, расположенный в 20 пикселях от левого края и в 30 пикселях от верхнего края элемента canvas, вы напишете так (**рис. 3.02**):

```
context.strokeRect(20,30,100,50);
```

Это очень простой пример. 2D API предоставляет очень много методов: `fillStyle`, `fillRect`, `lineWidth`, `shadowColor` и многие другие.

**Рис. 3.02.** Прямоугольник, нарисованный на canvas

В теории любое изображение, которое можно реализовать в программе, аналогичной Illustrator, можно создать внутри элемента `canvas`. На практике делать это очень утомительно и, скорее всего, приведет к безумно длинному коду на JavaScript. Да и вообще смысл Canvas несколько не в этом.

[4] Полная ссылка: http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html

## Canvas. Ага! И для чего он нужен?

Создавать картинки на лету с использованием JavaScript и Canvas – это все здорово и прекрасно, но если вы не убежденный мазохист, то зачем?

Истинная сила Canvas заключается в том, что его содержимое может быть обновлено в любой момент, на нем можно нарисовать новое содержимое в зависимости от действий пользователя. Эта способность реагировать на события, вызванные действиями пользователя, делает возможным создавать инструменты и игры, для которых раньше потребовалась бы технология плагина, например Flash.

Одна из первых флагманских демонстраций возможностей Canvas была разработана в Mozilla Labs. Приложение Bespin (https://bespin.mozilla.com) – редактор кода, работающий внутри браузера (**рис. 3.03**).

Он очень мощный. Очень впечатляющий. Но это прекрасный пример того, чего с Canvas делать как раз совершенно не нужно.

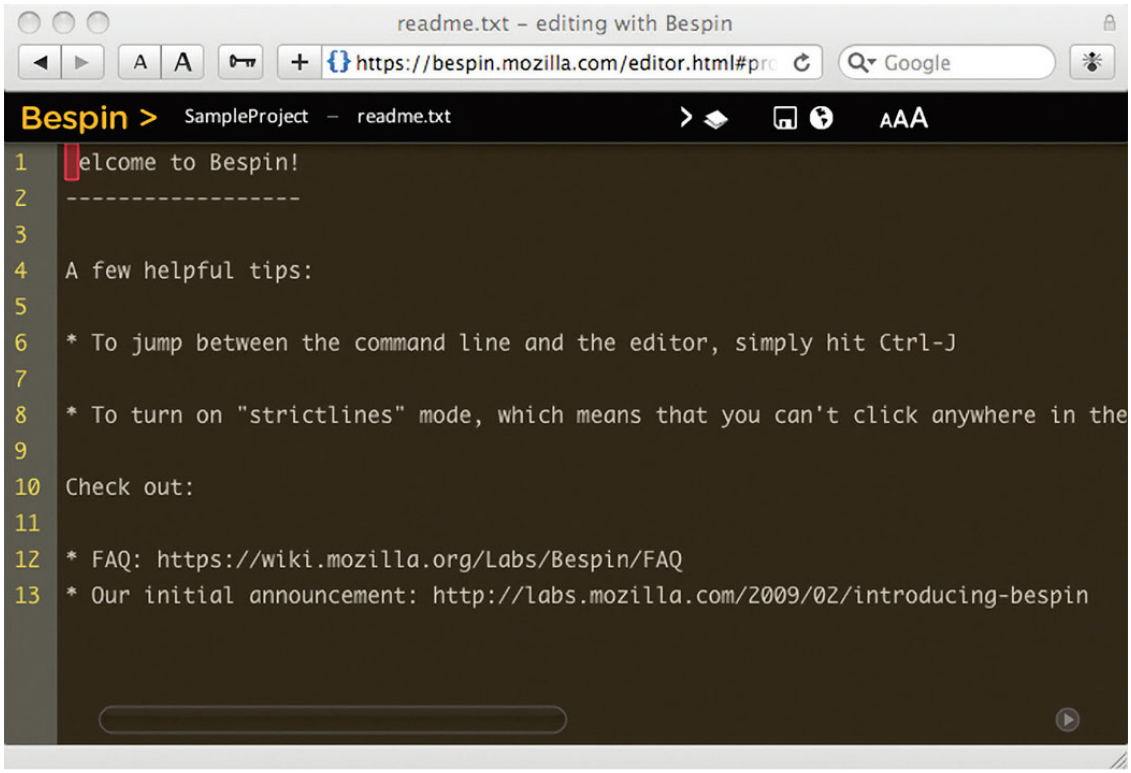


**Рис. 3.03.** Приложение Bespin, разработанное на Canvas

## Доступ запрещен

Редактор кода по своей природе имеет дело с текстом. Редактор Bespin работает с текстом внутри элемента `canvas` – вот только на самом деле это уже не текст; это набор фигур, которые выглядят как текст.

Каждый документ в вебе можно описать объектной моделью документа (Document Object Model, DOM). DOM может содержать большое количество различных узлов, самыми важными из которых являются узлы элементов, текстовые узлы и атрибуты. У элемента canvas нет DOM. Содержимое, нарисованное внутри `canvas`, нельзя представить как дерево узлов.

Программы, читающие с экрана, и другие технологии специальных возможностей разбирают документ благодаря тому, что имеют доступ к объектной модели документа. Нет DOM – доступа тоже нет.

Недоступность содержимого Canvas для технологий специальных возможностей – большая проблема для HTML5. К счастью, очень умные люди работают вместе в рамках рабочей группы, которая может предложить решение этой проблемы (http://bkaprt.com/html5/2)[5].

Доступ к Canvas – очень важный вопрос, и я не хотел бы, чтобы какие-либо внесенные предложения принимались поспешно. С другой стороны, мне не хотелось бы также, чтобы Canvas задерживал все остальное в спецификации HTML5.

## Умный Canvas

Пока проблема с доступом технологий специальных возможностей не решена, может показаться, что Canvas – неактуальная технология для веб-разработчиков. Но это не на сто процентов верно.

Когда я использую на сайте JavaScript, я использую его не как основную функциональность, а как дополнение к уже имеющейся. Посетителям, у которых нет JavaScript, все равно будет доступно все содержимое, но оно будет вести себя несколько менее динамично, чем в среде с включенным JavaScript.

Этот многоуровневый подход, называющийся еще «ненавязчивый JavaScript», можно применить и к Canvas. Вместо того чтобы использовать Canvas для создания содержимого, используйте его, чтобы иначе отобразить существующее содержимое.

Предположим, у вас есть таблица с данными. Скажем, вы хотите проиллюстрировать аналитические выводы из этих данных в диаграмме. Если данные статичны, то вы можете сгенерировать картинку диаграммы – например, используя Google Chart API. Если же данные редактируемы, если они меняются в ответ на события, вызванные действиями пользователя, тогда Canvas – отличный инструмент для того, чтобы сгенерировать изменившуюся диаграмму. Ключевой момент здесь тот, что то содержимое, которое выводится внутри элемента `canvas`, уже доступно в существующем элементе `table`.

Умные парни из Filament Group разработали jQuery-плагин как раз для такой ситуации (**рис. 3.04**; http://bkaprt.com/html5/3)[6].

[5] Полная ссылка: http://www.w3.org/Wai/pf/html-task-force

[6] Полная ссылка: http://www.filamentgroup.com/lab/update_to_jquery_visualize_accessible_charts_with_html5_from_designing_with/

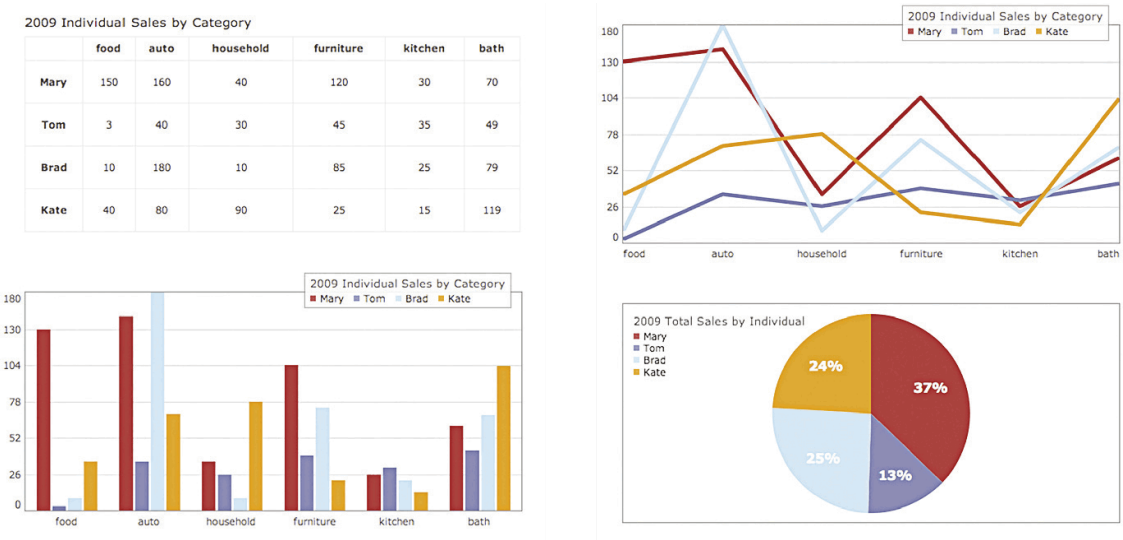


**Рис. 3.04.** Сгенерированная с помощью Canvas диаграмма из данных, введенных пользователями

Есть и другой вариант. Canvas – не единственная API для генерации динамических картинок. SVG (Scalable Vector Graphics, масштабируемая векторная графика) – XML-формат, в котором можно описать те же самые формы, что и в Canvas.

Поскольку XML – текстовый формат данных, содержимое SVG теоретически доступно программам, читающим текст на экране.

На практике SVG не захватило воображение разработчиков настолько, насколько это получилось у Canvas. Хотя Canvas – новый паренек в классе, у него уже отличная браузерная поддержка. Safari, Firefox, Opera и Chrome поддерживают Canvas. Есть даже JavaScript-библиотека, которая добавляет поддержку Canvas в Internet Explorer (http://bkaprt.com/html5/4)[7].

Учитывая мантры WHATWG – «асфальтируйте тропинки» и «не изобретайте велосипед», – может показаться странным, что при этом они выступают за включение Canvas в HTML5, когда уже существует стандарт SVG. Как часто и бывает, спецификация HTML5 только документирует то, что браузеры уже поддерживают. Элемент canvas не был задуман специально для HTML5; он был разработан Apple и реализован в Safari. Другие производители браузеров увидели, что делает Apple, им это понравилось, они это скопировали.

Звучит как-то несуразно, но зачастую именно так рождаются наши веб-стандарты. Например, в конце XX века Microsoft создала объект `XMLHttpRequest` для Internet Explorer 5.

Десятилетие спустя все браузеры поддерживают эту функцию, и в W3C она находится в статусе последней версии рабочего черновика.

В развивающемся по законам Дарвина мире веб-браузеров Canvas распространяется вширь и вдаль. Если он сможет приспособиться к технологиям специальных возможностей, выживание ему обеспечено.

[7] Полная ссылка: http://code.google.com/p/explorercanvas/

# Audio

Первым сайтом, который я сделал в жизни, был маленький сайт-визитка моей группы. Я хотел, чтобы посетители сайта могли слушать песни, которые исполняет моя группа. Так начался мой спуск в подземное царство огромного количества форматов и музыкальных проигрывателей. Многие из них боролись за мое внимание: QuickTime, Windows Media Player, Real Audio, – и я потратил слишком много времени, переживая из-за доли каждого на рынке и кросс-платформенной совместимости.

В последующие годы битву за повсеместное распространение выиграл формат MP3. Но для того чтобы простым способом дать послушать звуковой файл посетителю сайта, все еще нужна проприетарная технология. Эту битву выиграл проигрыватель Flash.

Теперь на ринг, чтобы сразиться с действующим чемпионом, входит HTML5.

Встроить аудиофайл в HTML5-документ очень просто:

```
<audio src="witchitalineman.mp3">
</audio>
```

Наверное, это даже слишком просто. Наверняка вы хотите несколько более конкретно указать, что должен делать элемент `audio`.

Предположим, что живет на свете злой урод, который ненавидит веб и всех пользователей Интернета. Этому господину наверняка плевать, что встраивать на страницу аудиофайл, который начинает проигрываться автоматически, невероятно грубо и глупо. С помощью атрибута `autoplay` можно удовлетворить его глубоко порочные желания.

```
<audio src="witchitalineman.mp3" autoplay>
</audio>
```

Если вы когда-нибудь будете так использовать атрибут `autoplay`, знайте: я вас найду.

Обратите внимание, что у атрибута `autoplay` нет значения. Такие атрибуты называются булевыми, в честь Джорджа Буля, великого математика из Корка[8].

Компьютерная логика целиком основана на булевой логике: электрический ток либо течет, либо нет; двоичное значение – либо единица, либо ноль; результат расчета – либо истина (true), либо ложь (false).

Не перепутайте булевы атрибуты и булевы значения. Вас можно понять, если вы подумаете, что булев атрибут будет принимать значения true и false. В действительности булево по природе само существование элемента: присутствует он или нет. Даже если напишете атрибуту значение, никакого эффекта это иметь не будет. Написать `autoplay="false"` или `autoplay="no thanks"` – то же самое, что написать `autoplay`.

Если вы используете синтаксис XHTML, можете написать `autoplay="autoplay"`. Этот синтаксис используется в официальных документах Управления по управлению управлением избыточной информацией.

Если ставить аудиофайл на автоматическое проигрывание не кажется вам достаточно зловредным, вы можете принести еще больше скорби в этот мир, заставив аудиофайл бесконечно повторяться. Другой булев атрибут, `loop`, позволит вам совершить этот низкий поступок:

```
<audio src="witchitalineman.mp3" autoplay loop>
</audio>
```

[8] Джордж Буль (1815–1864) – один из предтеч математической логики, был профессором математики в университете Корка с 1849 года. *Прим. перев.*

Использование атрибута `loop` вместе с атрибутом `autoplay` заставит меня искать вас с удвоенной энергией.

## Вырваться из-под контроля

Элемент `audio` можно использовать не только для злых, но и для благих целей. Дать пользователю контроль над управлением проигрывания аудиофайла – здравая идея, которую легко осуществить с помощью булева атрибута `controls`:

```
<audio src="witchitalineman.mp3" controls>
</audio>
```

Присутствие атрибута `controls` заставляет браузер отобразить встроенные элементы управления того, чтобы проигрывать аудиофайл и ставить его на паузу – и для того, чтобы настраивать громкость (**рис. 3.05**).

**Рис. 3.05.** Используйте атрибут controls, для того чтобы отобразить элементы управления «проиграть», «пауза» и управления громкостью

Если вам не нравятся встроенные в браузер элементы управления, можете создать свои собственные. С помощью JavaScript вы можете взаимодействовать с `Audio` API, которое дает вам доступ к методам (`play` и `pause`) и свойствам (`volume` и др.). Вот быстрый и простой пример, как использовать элементы `button` и ужасные обработчики событий, написанные прямо в коде страницы (**рис. 3.06**):

```
<audio id="player" src="witchitalineman.mp3">
</audio>
<div>
<buttononclick="document.getElementById('player').
play()">Проиграть
</button>
<buttononclick="document.getElementById('player').
pause()">Пауза
</button>
<buttononclick="document.getElementById('player').   volume+=
0.1">
Громче
</button>
<buttononclick="document.getElementById('player').   volume-=
0.1">
Тише
</button>
</div>
```

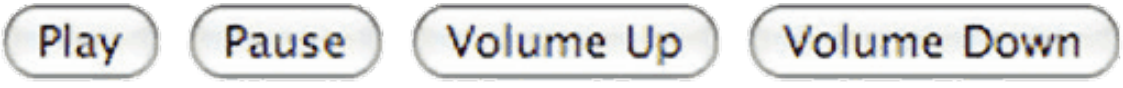


**Рис. 3.06.** Элементы управления (Проиграть, Пауза, Громче, Тише), сделанные с помощью элементов button

## Буферизация

В какой-то момент спецификация HTML5 включала еще один булев атрибут для элемента `audio`. Атрибут `autobuffer` был более вежливым и продуманным вариантом неприятного атрибута `autoplay`. Он позволял авторам сообщить браузеру, что хотя аудиофайл и не нужно начинать проигрывать автоматически, скорее всего в какой-то момент пользователь начнет его проигрывать, поэтому браузеру стоит начать подгружать файл в фоновом режиме.

Это был бы полезный атрибут, но, к сожалению, Safari сделал лишний шаг вперед. Этот браузер начал подгружать аудиофайлы вне зависимости от того, присутствует атрибут `autobuffer` или нет. Не забывайте, что из-за того, что `autobuffer` – булева переменная, не было никакого способа сказать Safari, что подгружать аудиофайл не нужно: `autobuffer="false"` – то же самое, `что autobuffer="true"` или любое другое значение (http://bkaprt.com/html5/5)[9].

В данный момент атрибут `autobuffer` заменен атрибутом `preload`. Это не булев атрибут. Он принимает одно из трех возможных значений: `none`, `auto` и `metadata`. Написав `preload="none"`, вы можете явным образом указать браузеру, что подгружать аудиофайл заранее не нужно:

```
<audio src="witchitalineman.mp3" controls preload="none">
</audio>
```

Если у вас на странице только один элемент audio, возможно, стоит использовать `preload="auto"`, но чем больше элементов audio появляется, тем больше интернет-канал посетителей вашей странички будет загружен из-за излишней предварительной подгрузки.

## Его вам сразу включ́ат, а может быть, включа́т

Элемент `audio` выглядит практически идеальным. Где-то должен быть подвох, правда? Он есть.

Проблемы с элементом `audio` не в спецификации. Главная проблема – с форматами аудиофайлов.

Хотя формат MP3 и распространен повсеместно, это не открытый формат. Из-за того, что на этот формат навешано множество патентов, нельзя написать декодер MP3, не заплатив отчислений по патенту. У корпораций вроде Apple или Adobe с этим нет проблем, но для маленьких компаний или опенсорс-групп это не так просто. Поэтому Safari будет с удовольствием проигрывать MP3-файлы, а Firefox – нет.

На свете есть и другие аудиоформаты. Кодек Vorbis – обычно для него используется файл с расширением `.ogg` – никакими патентами не обременен. Firefox поддерживает Ogg Vorbis, а Safari – нет.

К счастью, есть способ использовать элемент `audio`, не делая при этом «выбор Софи»[10] между форматами файлов. Вместо того чтобы использовать атрибут `src` в открывающем теге `<audio>`, можно указать несколько форматов файлов с помощью элемента source:

```
<audio controls>
<source src="witchitalineman.ogg">
```

---

[9] Полная ссылка: https://bugs.webkit.org/show_bug.cgi?id=25267

[10] В фильме «Выбор Софи» (Sophie's Choice, 1982 год) главная героиня (Мэрил Стрип, получившая за эту роль «Оскар»), мать двоих детей, во время Второй мировой войны должна сделать выбор, кто из ее детей останется жить, а кто отправится в газовую камеру (если она откажется сделать выбор – погибнут оба). *Прим. пер.*

```
<source src="witchitalineman.mp3">
</audio>
```

Браузер, который может проигрывать файлы Ogg Vorbis, не станет смотреть дальше первого элемента `source`. Браузер, который может проигрывать файлы MP3, но не может Ogg Vorbis, пропустит первый элемент `source` и проиграет файл во втором элементе `source`.

Можно помочь браузерам и указать MIME-типы для каждого исходного файла:

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

Элемент `source` – самостоятельный (или «пустой») элемент, так что если вы используете синтаксис XHTML, не забудьте включить закрывающий слэш в конца каждого тега `<source />`.

## Запасной вариант

Возможность указывать несколько элементов `source` очень удобна. Но есть браузеры, которые пока не поддерживают элемент `audio` совсем. Угадаете, на который браузер я намекаю?

Internet Explorer и его родню нужно кормить аудиофайлами с ложечки, по старинке, через Flash. Модель содержимого элемента `audio` позволяет это сделать. Все, что находится между открывающим и закрывающим тегами `<audio>` – и что при этом не является элементом `source` – будет показываться браузерам, которые не понимают элемента `audio`:

```
<audio controls>
<source src="witchitalineman.ogg» type="audio/ogg">
<source src="witchitalineman.mp3» type="audio/mpeg">
<object type="application/x-shockwave-
flash"data="player.swf?soundFile=witchitalineman.mp3">
<param name="movie"value="player.swf?
soundFile=witchitalineman.mp3">
</object>
</audio>
```

В этом примере элемент `object` будет доступен только тем браузерам, которые не поддерживают элемент `audio`.

Можно пойти еще дальше. Элемент `object`, включающийся при «запасном варианте», тоже предоставляет вам возможность включить содержимое. Это значит, что, если больше ничего не срабатывает, можно дать старый проверенный вариант – гиперссылку:

```
<audio controls>
<source src="witchitalineman.ogg" type="audio/ogg">
<source src="witchitalineman.mp3" type="audio/mpeg">
<object type="application/x-shockwave-flash"data="player.swf?
soundFile=witchitalineman.mp3">
<param name="movie"value="player.swf?
soundFile=witchitalineman.mp3">
<a href="witchitalineman.mp3">Скачать песню</a>
</object>
</audio>
```

В этом примере четыре уровня постепенной деградации.

1. Браузер поддерживает элемент `audio` и формат Ogg Vorbis.
2. Браузер поддерживает элемент `audio` и формат MP3.
3. Браузер не поддерживает элемент `audio`, но в нем установлен Flash-плагин.
4. Браузер не поддерживает элемент `audio`, и в нем не установлен Flash-плагин.

## Доступ на все уровни

Модель содержимого элемента `audio` очень удобна для предоставления «запасного варианта» содержимого. Запасное содержимое – не то же самое, что содержимое для технологий специальных возможностей.

Предположим, что вместе с аудиофайлом идет его транскрипция. Вот так не нужно размечать эти данные:

```
<audio controls>
<source src="witchitalineman.ogg" type="audio/ogg">
<source src="witchitalineman.mp3" type="audio/mpeg">
<p>I am a lineman for the county…</p>
</audio>
```

Транскрипция в этом примере будет видна только тем браузерам, которые поддерживают элемент `audio`. Размечать незвуковое содержимое таким образом никак не поможет глухому пользователю с хорошим браузером. Кроме того, так называемое содержимое для технологий специальных возможностей часто полезно для всех – так что зачем его прятать?

```
<audio controls>
<source src="witchitalineman.ogg" type="audio/ogg">
<source src="witchitalineman.mp3" type="audio/mpeg">
</audio>
<p>I am a lineman for the county…</p>
```

# Video

Если родное для браузера воспроизведение аудио – это воодушевляюще, то перспектива родного отображения видео в браузере заставляет веб-разработчиков пускать слюнки от нетерпения. По мере того как пропускная способность интернет-каналов возросла, видеосодержимое начало становиться все более и более популярным. Сейчас главная технология для показа видео в вебе – Flash-плагин. Но HTML5 может все это изменить.

Элемент video работает примерно так же, как элемент `audio`. У него есть необязательные атрибуты `autoplay`, `loop` и `preload`. Вы можете указать расположение видеофайла либо через атрибут `src` элемента `video`, либо с помощью элементов source, вложенных внутри открывающих и закрывающих тегов `<video>`. Вы можете разрешить браузеру отобразить пользовательский интерфейс с помощью атрибута `controls` или написать свои собственные управляющие элементы.

Главная разница между аудио– и видеосодержимым состоит в том, что фильмы по своей природе будут занимать больше места на экране, поэтому, скорее всего, вам стоит определить размеры элемента:

```
<video src="movie.mp4" controls width="360" height="240">
</video>
```

Вы можете выбрать подходящее изображение для видеофайла и указать браузеру, что нужно его отобразить, через атрибут poster (**рис. 3.07**):

```
<video src="movie.mp4" controls width="360"height="240"
poster="placeholder.jpg">
</video>
```


**Рис. 3.07.** Через атрибут poster показывается картинка-заполнитель

Поле битвы конкурирующих видеоформатов «залито кровью» еще сильнее, чем в мире аудио. Из больших игроков нужно назвать MP4 – по уши увязшего в патентах – и Theora

Video (здесь все проще). И снова вам нужно будет указать альтернативные форматы и содержимое, которое выводится в том случае, если HTML5 video не поддерживается:

```
    <video                  controls                  width="360"
height="240"poster="placeholder.jpg">
    <source src="movie.ogv" type="video/ogg">
    <source src="movie.mp4" type="video/mp4">
    <object       type="application/x-shockwave-flash"width="360"
height="240"data="player.swf?file=movie.mp4">
    <param name="movie"value="player.swf?file=movie.mp4">
    <a href="movie.mp4">Скачать фильм</a>
    </object>
    </video>
```

Авторы спецификации HTML5 изначально надеялись установить некий единый формат видео, который бы поддерживали все. К сожалению, производители браузеров не смогли договориться о едином формате.

## Нативный режим

Возможность нативного встраивания видео в веб-страницы – пожалуй, самое заманчивое добавление к HTML со времен введения элемента `img`. Большие игроки, как, например Google, не стесняются выражать свой энтузиазм на этот счет. Вы можете взглянуть на то, что они приготовили для YouTube, по адресу: http://youtube.com/html5.

Одной из проблем отображения мультимедиа в плагине является то, что содержимое плагина находится в «песочнице», отдельно от всего остального документа. Нативные мультимедиа-элементы в HTML смогут работать в комплексе с остальными браузерными технологиями – JavaScript и CSS. Элементом `video` можно не только управлять через JavaScript, можно также назначать ему стили (**рис. 3.08**).


**Рис. 3.08.** Элемент video с примененными стилями

Попробуйте-ка сделать это с плагином.

Аудио и видео – долгожданные дополнения к HTML5, но веб – не среда вещания, а интерактивная среда. Самый старый и самый мощный способ обеспечивать интерактивность – формы. В главе 4 мы посмотрим на то, какое обновление в HTML5 получили формы.

# 4. Веб-формы 2.0

Когда в веб-браузерах появился JavaScript, его немедленно стали использовать для двух задач: изменения картинки при наведении мышью и улучшения форм. Когда же в CSS появился псевдокласс `:hover`, веб-разработчикам перестало быть нужным использовать JavaScript для того, чтобы добиться просто изменения картинки при наведении.

И эта картина зачастую повторяется. Если какое-то типовое действие достаточно популярно, практически наверняка развитие будет идти так, что вместо того чтобы писать собственный скрипт, достаточно будет более декларативного решения. Вот почему в CSS3 появляется еще больше возможностей для анимации из числа тех, что раньше требовали JavaScript.

Когда дело касается улучшения форм, у CSS есть ряд ограничений. Здесь в дело вступает HTML5. Следуя той же самой проторенной дорожкой от программного решения к декларативному, спецификация HTML5 вводит много новых улучшений форм.

Эти функции изначально были частью спецификации WHATWG, которая называлась Веб-формы 2.0 и основывалась на уже проделанной работе в W3C. Эта спецификация теперь включена в HTML5.

# Placeholder

Вот типичная задача, которая часто используется для поисковых форм и обычно решается написанием скрипта для DOM:

1. Если у поля формы нет значения, вставить туда текст-заполнитель.
2. Когда пользователь перемещает фокус на это поле, убрать текст-заполнитель.
3. Если пользователь выходит из поля и в нем по-прежнему нет значения, снова поставить текст-заполнитель.

Текст-заполнитель обычно показывается шрифтом более светлого цвета, чем шрифт самого значения поля формы. Это реализуется с помощью CSS, JavaScript или обоими этими инструментами вместе.

В документе HTML5 вы можете просто использовать атрибут placeholder (**рис. 4.01**):

```
<label for="hobbies">Ваши хобби</label>
<input id="hobbies" name="hobbies"
type="text"placeholder="Стряхивать филинов с деревьев">
```


**Рис. 4.01.** «Стряхивать филинов с деревьев» появляется в поле input посредством атрибута placeholder

Атрибут `placeholder` прекрасно работает в браузерах, которые его поддерживают, но, увы, таких браузеров прямо сейчас не так-то много. Вы сами решаете, что вы хотите видеть в других браузерах, которые пока этот атрибут не поддерживают.

Вы можете вообще ничего не делать. В конце концов это функциональность из разряда «хорошо бы иметь», а не «обязательно иметь».

Либо же вы можете сделать запасное решение на JavaScript. В этом случае вам нужно убедиться, что скрипт на JavaScript будет исполняться только в браузерах, которые не понимают атрибут `placeholder`.

Вот маленькая JavaScript-функция общего характера, которая тестирует, поддерживается тот или иной атрибут или нет:

```
function elementSupportsAttribute(element,attribute) {
var test = document.createElement(element);
if (attribute in test) {
return true;
} else {
return false;
}
}
```

Работает это так: в памяти создается «фантомный» элемент (в вашем документе его нет), а затем проверяется, есть ли в прототипе этого элемента свойство с тем же названием, что и атрибут, который вы передаете в эту функцию. Функция вернет `true` или `false`.

С помощью этой функции вы можете удостовериться, что JavaScript-решение будет исполняться только в тех браузерах, которые не поддерживают `placeholder`:

```
if (!elementSupportsAttribute('input','placeholder')) {
// Код запасного решения на JavaScript.
}
```

# Autofocus

«Привет! Я автофокус. Может быть, вы помните меня по кнопочкам на сайтах: “Google: мне повезет” и “Twitter: что происходит?”»

Это простое типовое решение из одного шага, которое достаточно просто программируется на JavaScript:

1. Когда документ загрузился, автоматически поставить фокус на одно конкретное поле в форме.

HTML5 позволяет вам сделать это с помощью булева атрибута `autofocus`:

```
<label for="status">Что происходит?</label>
<input id="status" name="status" type="text" autofocus>
```

Единственная проблема с этим решением – оно может адски раздражать. Когда я хожу по страничкам в вебе, я часто нажимаю пробел, чтобы прокрутить страницу на экран вниз. На таких сайтах, как Twitter, которые используют автофокус, я понимаю, что занимаюсь тем, что не пролистываю экран, а впечатываю пробелы в поле формы.

Мне понятно, почему атрибут `autofocus` оказался добавлен HTML5 – снова принцип асфальтирования тропинок, – но меня беспокоит юзабилити этого решения: реализуется ли оно скриптом или средствами браузера. Эта функция может быть полезной, но может точно также и приводить в ярость. Пожалуйста, как следует подумайте перед тем, как применять его.

Одно из преимуществ того, чтобы переместить это решение из скрипта в разметку, в том, что теоретически браузеры могут включить опцию настройки, с помощью которой пользователи смогут отключить автоматическую фокусировку. На практике ни в одном браузере такой настройки пока нет, но введено это решение в HTML5 относительно недавно. На данный момент единственный способ отключить автофокус методами JavaScript – отключить сам JavaScript совсем. Это работает, но, надо сказать, это довольно радикальное решение – такое же, как, например, выдавить себе глаза, если вас раздражает яркий свет.

Как и в случае с атрибутом `placeholder`, вы можете протестировать, поддерживается ли `autofocus`, и, если нет, откатиться к решению на JavaScript:

```
if (!elementSupportsAttribute('input','autofocus')){
document.getElementById('status'). focus();
}
```

Атрибут `autofocus` работает не только на элементах `input`; его можно использовать на любом поле формы – как, например, `textarea` или `select`, но его можно использовать только один раз во всем документе.

## Required

Один из самых распространенных случаев использования JavaScript – валидация форм на стороне клиента. И снова HTML5 перемещает это решение из JavaScript в разметку. Просто добавьте булев атрибут required:

```
<label for="pass">Ваш пароль</label>
<input id="pass" name="pass" type="password" required>
```

Теоретически это дает указание браузерам не отправлять форму, если необходимые поля не заполнены. Даже несмотря на то, что пока браузеры этого не делают, уже сейчас можно использовать атрибут `required` в той валидации форм, которую вы пишете на JavaScript. Вместо того чтобы держать список всех требуемых полей в вашем скрипте или добавлять в разметку `class="required"`, теперь вы можете проверять существование атрибута `required`.

# Autocomplete

Браузеры не просто показывают веб-страницы. В большинстве браузеров есть дополнительные функции, предназначенные для улучшения юзабилити, безопасности или удобства, когда вы бродите по просторам веба. Одна из таких функций – автозаполнение форм. В большинстве случаев это очень полезно, но иногда может раздражать или даже представлять совершенно реальную опасность. Я не против, что браузер помнит мои контакты, но, пожалуй, мне не хотелось бы, чтобы он запоминал данные логина для моего банковского аккаунта – на тот случай, например, если мой компьютер украдут.

HTML5 позволяет вам отключить автозаполнение во всей форме или для какого-либо конкретного поля. Атрибут `autocomplete` не является булевым, но он может принимать только два возможных значения: “on” и “off ”:

```
<form action="/selfdestruct" autocomplete="off">
```

По умолчанию браузеры будут считать, что autocomplete имеет значение “on”, и тем самым у них есть возможность осуществить предварительное заполнение формы.

Если вы хотите, чтобы в ваших формах было два варианта автозаполнения, это тоже возможно. Если вы хотите, чтобы предварительное заполнение было включено в форме, но отключено для одного-двух полей, можно сделать так:

```
<input type="text" name="onetimetoken" autocomplete="off">
```

Никакого запасного варианта на JavaScript для браузеров, которые не поддерживают атрибут `autocomplete`, нет. В этом случае новый атрибут HTML5 дополняет существующее поведение браузеров, а не заменяет решение на JavaScript.

Возможность отключать автозаполнение в браузерах может показаться странным дополнением к спецификации HTML5. HTML5 предназначен для того, чтобы закрепить превалирующие решения, и в данном случае это не очень типичный пример. Но учитывая потенциальные риски безопасности, которые связаны с автоматическим заполнением форм, имеет смысл дать владельцам сайтов возможность переопределить именно эту функцию браузера.

## Datalist

Новый элемент `datalist` позволяет вам скрестить обычный элемент input с элементом select. С помощью атрибута `list` вы можете сопоставить с полем формы список опций (**рис. 4.02**):

```
    <label for="homeworld">Ваша родная планета</label>
    <input              type="text"              name="homeworld"
id="homeworld"list="planets">
    <datalist id="planets">
    <option value="Меркурий">
    <option value="Венера">
    <option value="Земля">
    <option value="Марс">
    <option value="Юпитер">
    <option value="Сатурн">
    <option value="Уран">
    <option value="Нептун">
    </datalist>
```

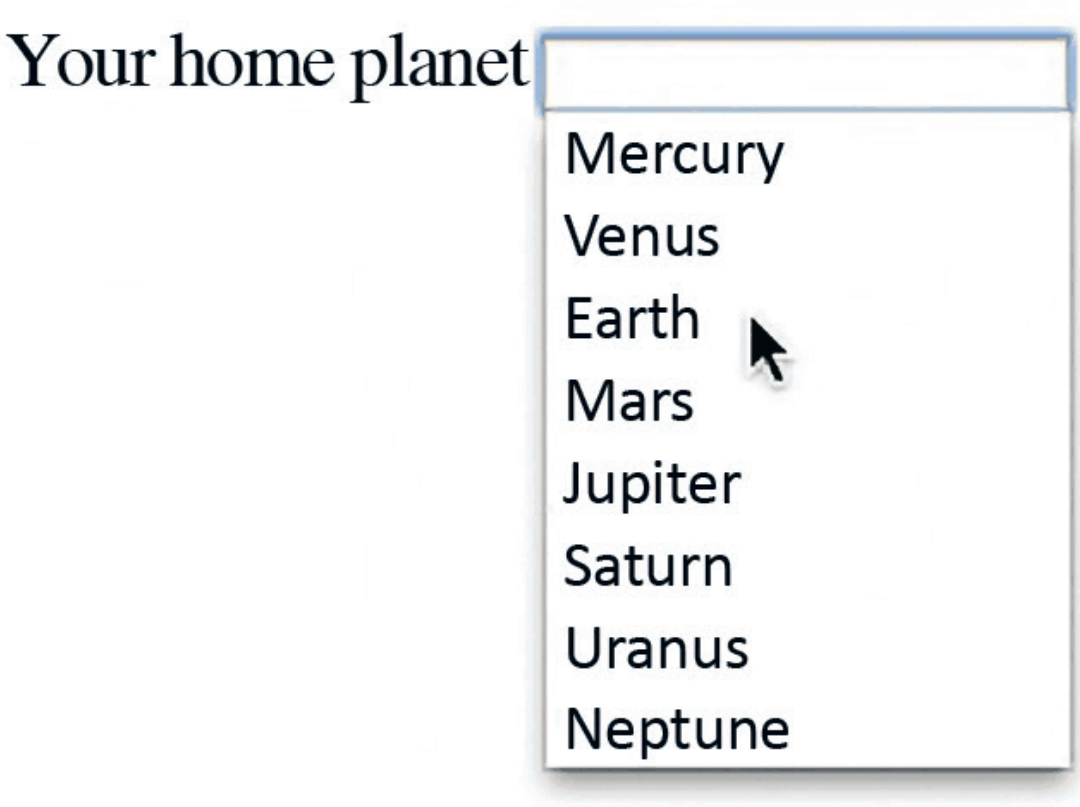


**Рис. 4.02.** Новый элемент datalist

Это позволяет юзерам выбрать опцию из приготовленного списка или ввести значение, которого в списке нет. Это очень полезно для ситуаций, которые обычно требуют отдельного поля в форме, озаглавленного: «если вы выбрали вариант “другое”, пожалуйста, укажите…» (**рис. 4.03**).


**Рис. 4.03.** Элемент datalist: показано, что юзер может ввести значение, которого нет в списке

Элемент `datalist` – отличное, ненавязчивое дополнение к полю формы. Если браузер не поддерживает `datalist`, то поле ведет себя как обычное поле ввода.

# Типы полей ввода

В HTML5 стало намного больше вариантов атрибута `type` элемента `input`. Здесь нужно заасфальтировать столько тропок, что это похоже на строительные работы после того, как по лесу в панике пробежала толпа дачников.

## Поиск

Элемент `input` со значением “search” в атрибуте `type` будет вести себя примерно так же, как элемент ввода со значением “text” атрибута `type`:

```
<label for="query">Поиск</label>
<input id="query" name="query" type="search">
```

Единственная разница между “text” и “search” состоит в том, что браузер может отображать поле для поиска иначе для того, чтобы соответствовать стилю полей поиска в операционной системе. Именно так делает Safari (**рис. 4.04**).


**Рис. 4.04.** Safari устанавливает стили полей поиска в соответствии с Mac OS

## Контакты

В HTML5 добавилось три новых типа type для особенных типов контактов: e-mail-адресов, веб-сайтов и номеров телефонов:

```
<label for="email">Email-адрес</label>
<input id="email" name="email" type="email">
<label for="website">Вебсайт</label>
<input id="website" name="website" type="url">
<label for="phone">Телефон</label>
<input id="phone" name="phone" type="tel">
```

И снова эти поля будут вести себя так же, как текстовые поля ввода, но у браузеров будет чуть больше информации о том, какой именно тип данных ожидается в этом поле.

Разработчики Safari уверяют, что их браузер поддерживает эти новые типы ввода, но при быстром взгляде на форму в десктоп-браузере не видно никаких различий с простым использованием `type="text"`. Однако, если вы начнете работать с этой же формой в Mobile Safari, различия станут очевидными. Браузер показывает разные экранные клавиатуры в зависимости от значения атрибута `type` (**рис. 4.05**).

Тонко сделано, Webkit, тонко.


**Рис. 4.05.** Mobile Safari показывает разные экранные клавиатуры в зависимости от значения атрибута type

## Ползунки

Большое количество JavaScript-библиотек предлагают набор виджетов, которые можно использовать в веб-приложениях. Они отлично работают – пока включен JavaScript. Было бы прекрасно, если бы пользователям не приходилось скачивать JavaScript-файл каждый раз, когда мы хотим добавить интересный элемент управления на нашу страницу.

Классический пример – ползунок. До сих пор нам приходилось использовать JavaScript для того, чтобы эмулировать этот в каком-то смысле интерактивный элемент. В HTML5 благодаря `type="range"` можно воспользоваться элементом управления, встроенным в браузер:

```
<label for="amount">Почем опиум для народа?</label>
<input id="amount" name="amount" type="range">
```

Safari и Opera на данный момент поддерживают этот тип элемента ввода, предлагая похожие элементы управления (**рис. 4.06**).


**Рис. 4.06.** Тип ввода range в Safari и Opera

По умолчанию элемент ввода принимает значения от нуля до ста. Вы можете поставить свои собственные минимальные и максимальные значения с помощью атрибутов min и max:

```
<label for="rating">Ваша оценка</label>
<input id="rating" name="rating" type="range" min="1" max="5">
```

Для пользователей Safari и Opera все здорово и прекрасно; другие браузеры просто будут выводить обычное текстовое поле. Это, наверное, нормально, но вы, пожалуй, захотите использовать запасное решение на JavaScript для браузеров, которые не поддерживают `type="range"`.

## Проверка

Для того чтобы проверить, есть ли в браузере встроенная поддержка типов ввода, нужен прием, похожий на проверку на поддержку атрибута. Опять же вам нужно будет создать в памяти «фантомный» элемент `input`. Затем вы устанавливаете атрибут `type` на то значение, которое хотите проверить.

После этого вы запрашиваете значение свойства `type`, и если получаете значение “text”, то вы знаете, что браузер не поддерживает то значение, которое вы установили.

Вот примерный код, хотя я уверен, что вы можете написать и что-то гораздо более элегантное:

```
function inputSupportsType(test) {
var input = document.createElement('input');
input.setAttribute('type',test);
if (input.type == 'text') {
return false;
} else {
return true;
}
}
```

Теперь вы можете использовать эту функцию, чтобы удостовериться, что JavaScript-widget будет исполняться только в тех браузерах, которые не поддерживают определенный тип элемента как встроенный:

```
if (!inputSupportsType('range')) {
// Код запасного решения на JavaScript.
}
```

Встроенный в браузер элемент ввода будет, разумеется, грузиться быстрее, чем решение на JavaScript, которому нужно ждать, пока загрузится вся DOM. Встроенный в браузер элемент управления будет также более доступен для технологий специальных возможностей, чем элемент, который написан на JavaScript, хотя – что весьма странно – элементом `range` в Safari на данный момент нельзя управлять с клавиатуры!

## Счетчики

Встроенный в браузер элемент управления `range` не показывает пользователю свое внутреннее значение. Вместо этого номер переводится в графическое представление ползунка. Это отлично для определенных типов данных. Другие типы данных предназначены для того, чтобы пользователь мог видеть и выбирать числовое значение. Здесь на помощь приходит `type="number"`:

```
<label for="amount">Почем опиум для народа?</label>
<input id="amount" name="amount" type="number" min="5" max="20">
```

Позволяя пользователю напрямую ввести значение в текстовое поле, браузеры также выводят элементы управления для того, чтобы пользователь мог уменьшить и увеличить значение (**рис. 4.07**).

Тип ввода `number` – гибрид `text` и `range`. Он позволяет пользователям вводить значения напрямую, как поле text, но также позволяет браузерам проверить, что в поле вводятся только численные значения, как в элементе управления `range`.

## Дата и время

Один из самых популярных JavaScript-виджетов – виджет выбора даты в календаре. Вы знаете, как это выглядит: вы бронируете билет на самолет или создаете мероприятие – и вам нужно выбрать дату. Выплывает небольшой календарик, из которого вы можете выбрать дату.

Все эти виджеты календаря выполняют одну и ту же функцию, но вы увидите, что они реализованы на каждом сайте слегка по-разному. Встроенный в браузер виджет календаря позволит сгладить несоответствия и уменьшить нагрузку на когнитивные способности пользователя во время процесса выбора даты.

HTML5 вводит целую уйму типов полей ввода специально для даты и времени:

• `date` предназначен для года, месяца и дня.

• `datetime` предназначен для года, месяца и дня вместе с часами, минутами, секундами и информацией о временной зоне.

• `datetime-local` – то же самое, но без информации о временной зоне.

• `time` – только часы, минуты и секунды.

• `month` – год и месяц, но без дня.

Все эти типы ввода будут записывать временные величины в какой-либо части стандартного формата YYYY-MM-DDThh: mm: ss.Z (Y – год, M – месяц, D – день, h – час, m – минута, s – секунда, а Z – временная зона). Возьмем, например, дату и время, когда закончилась Первая мировая война, в 11:11 11 ноября 1918 года:

• `date`: 1918-11-11

• `datetime`: 1918-11-11T11:11:00+01

• `datetime-local`: 1918-11-11T11:11:00

• `time`: 11:11:00

• `month`: 1918-11

Типа ввода `year` нет, хотя существует тип ввода `week`, который принимает число от 1 до 53 вместе с годом.

Использовать типы ввода даты и времени достаточно просто:

```
<label for="dtstart">Дата начала</label>
<input id="dtstart" name="dtstart" type="date">
```

Opera реализует эти типы ввода с помощью своей запатентованной технологии, заставляющей все выглядеть безобразно (**рис. 4.08**).

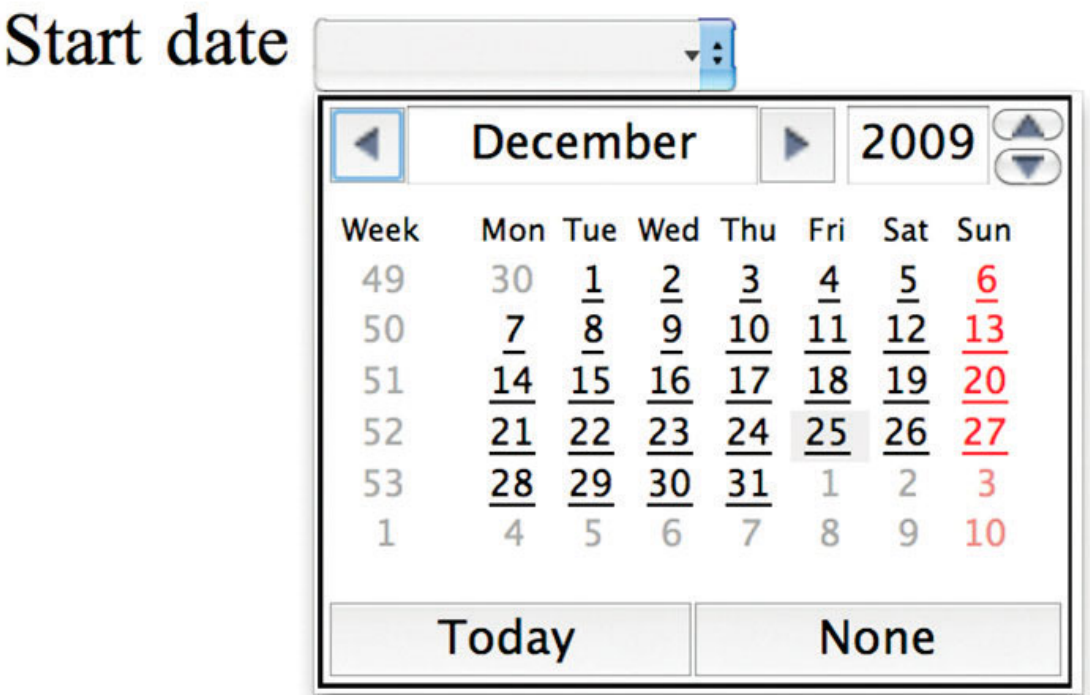


**Рис. 4.08.** Встроенное в Opera отображение календаря – совершенно безобразное

Как обычно, браузеры, которые не поддерживают эти типы ввода, откатятся на запасной вариант – покажут обычное текстовое поле ввода. В этом случае вы можете попросить своих пользователей вводить дату и время в формате ISO или же использовать JavaScript-библиотеку по выбору, чтобы сгенерировать виджет. Убедитесь, что вы сначала проверяете встроенную поддержку этого типа в браузере:

```
if (!inputSupportsType('date')) {
// Сгенерировать виджет календаря.
}
```

Даже самый элегантно написанный виджет для календаря на JavaScript требует довольно сложного кода, чтобы сгенерировать таблицу дней и обработать события по выбору даты. Встроенные в браузер виджеты календаря должны быть намного более легкими и быстрыми, не говоря уже о том, что они будут соответствовать друг другу на разных сайтах.

## Выбор цвета

Пожалуй, самый амбициозный элемент в HTML5, заменяющий JavaScript-виджет, – тип ввода `color`. Он принимает значение в знакомом шестнадцатеричном формате: `#000000` – черный, `#FFFFFF` – белый.

```
<label for="bgcolor">Цвет фона</label>
<input id="bgcolor" name="bgcolor" type="color">
```

План таков, что браузеры должны реализовать встроенные диалоги выбора цвета – точно такие же, какие есть практически в каждом другом приложении на вашем компьютере. Пока ни один браузер этого не реализовал, но когда они это сделают, будет, ну, решительно круто.

А пока вы можете использовать решение на JavaScript, но не забудьте проверять на встроенную поддержку, ваш код застрахован на будущее для браузеров завтрашнего дня.

## Сделай сам

Все эти новые типы данных выполняют две главные задачи: они позволяют браузерам выводить встроенные элементы управления, подходящие для ожидаемых данных ввода, и валидировать введенное значение. Эти дополнения в HTML5 покрывают большинство вариантов, но вы все равно можете обнаружить, что вам нужно провести валидацию для значения, которое не подпадает ни под одну из новых категорий.

Хорошие новости: вы можете использовать атрибут `pattern` для того, чтобы указать, какой именно род данных вы ожидаете. Плохие новости: придется использовать регулярные выражения:

```
<label for="zip">Почтовый индекс США</label>
<input id="zip" name="zip" pattern="[\d]{5}(-[\d]{4})">
```

Большую часть времени вам не придется использовать атрибут `pattern`. Что касается тех редких случаев, когда придется, – заранее вам сочувствую.

# В ожидании будущего

Формы в HTML5 получили огромное развитие. Большое количество тяжелой ноши, которую до этого нес на себе JavaScript, переходит на плечи разметки. Прямо сейчас мы находимся в переходной фазе, когда только некоторая часть этой функциональности поддерживается лишь некоторыми браузерами. Мы пока не можем выбросить наш JavaScript на помойку, но мы уже не так далеки от светлого будущего.

Валидация на стороне клиента станет гораздо проще – хотя на нее никогда нельзя полагаться; нужно всегда валидировать значения полей формы и на сервере. Для того чтобы создавать управляющие элементы в форму, вашим пользователям больше не нужно будет скачивать JavaScript-библиотеку – все это будет управляться встроенными возможностями браузера.

Я уверен, что вы понимаете преимущества встроенных в браузер управляющих элементов для календарей и ползунков, но уверен, что у вас возникает вопрос: «Могу ли я применять к ним свои стили?»

Это хороший вопрос. На данный момент ответ: «нет». Придется смириться с решением рабочей группы CSS.

Этот аргумент может быть для вас решающим. Если вам кажется, что реализация какого-либо элемента в формы в конкретном браузере далека от совершенства, вы можете использовать JavaScript-виджет, который даст вам больше возможностей по управлению этим элементом.

Я бы хотел, чтобы вы задали себе другой вопрос: «Нужно ли мне применять к элементам форм свои стили?»

Не забывайте, что смысл веба не в контроле за видом всех мелочей. Если посетитель вашего сайта привык к тому, как выглядят встроенные в его браузер поля форм, то вы не окажете ему никакой услуги, переопределив функциональность браузера своим собственным виджетом, даже если вам кажется, что ваш виджет выглядит лучше.

Лично я хотел бы посмотреть, как производители браузеров соревнуются за то, у кого окажутся более красивые и удобные для использования элементы управления для HTML5-форм. Вот та браузерная война, которую я бы поддержал.

Теперь давайте отложим формы в сторонку и посмотрим на новую сочную семантику, которая появилась в HTML5.

# 5. Семантика

HTML не дает нам огромного количества элементов для работы. Тот ассортимент, что у нас есть, скорее похож на ассортимент магазинчика на углу, а не гипермаркета.

У нас есть абзацы, списки и заголовки, но нет событий, репортажей и рецептов. HTML дает нам элемент, который позволяет разметить слово как аббревиатуру, но не дает элемента, чтобы разметить число как цену.

Но не сказать, чтобы это ограничение чему-либо помешало, – посмотрите хотя бы на огромное разнообразие сайтов в вебе. Даже несмотря на то, что HTML зачастую не дает специального элемента для разметки того или иного участка контента, он дает достаточно гибкости для того, чтобы быть «достаточно хорошим» инструментом для этой задачи.

Перефразируя Уинстона Черчилля, HTML – худшая форма разметки, если не считать всех прочих, что были испробованы человечеством.

## Расширяемость

Другие языки разметки позволяют вам изобретать любой элемент, какой пожелаете. В XML, если вы хотите, чтобы в вашем документе был элемент `event` или `price`, вы просто берете и создаете его. Недостаток этой свободы заключается в том, что вам нужно будет затем обучить парсер, что значит `event` и `price`. Достоинство ограниченного набора элементов HTML в том, что каждая программа, работающая с ним, знает о существовании каждого из этих элементов. В браузеры встроено знание HTML. Это было бы невозможно, если бы нам разрешалось придумывать названия элементов.

HTML предлагает очень удобный аварийный выход, который позволяет веб-разработчикам добавлять семантическое значение элементам – атрибут `class`. Этот атрибут позволяет нам отметить некоторые экземпляры элемента как относящиеся к особенному классу или типы этого элемента. То, что браузеры не понимают того словаря, который мы используем в своих атрибутах `class`, не влияет на отображение наших документов.

Если в этом месте вы думаете: «Погодите-ка, разве классы нужны не для CSS?», вы отчасти правы. CSS-селекторы по классам – один из примеров технологии, которая использует атрибут `class`, но это не единственная причина использовать классы. Классы могут использоваться при написании скриптов для DOM и даже могут использоваться браузерами, если названия классов следуют заранее согласованным правилам, как это происходит в случае микроформатов.

## Микроформаты

Микроформаты – набор договоренностей, согласованных внутри сообщества. Эти форматы используют атрибут `class` для того, чтобы заделать самые зияющие дыры в HTML: `hCard` – для контактов, `hCalendar` – для событий, `hAtom` – для новостных репортажей. Поскольку внутри сообщества существует договоренность о том, какие имена классов следует использовать, существуют парсеры и расширения браузеров, которые работают именно с этими шаблонами.

Микроформаты ограничены по самой своей задумке. Они не пытаются предложить решение для любого возможного сценария использования. Напротив, они нацеливаются на тот плод, что низко висит. Они предлагают решения для 80% сценариев использования, при этом на их создание затрачивается всего 20% усилий. Решить, что считается «плодом, что низко висит», довольно просто: нужно просто посмотреть на содержимое, которое люди уже размечают. Другими словами, заасфальтировать тропинки.

Звучит знакомо? Микроформаты и HTML5 построены на одной философии. По сути, то, как я описал микроформаты – договоренности, согласованные сообществом, – вполне применимо и к HTML5.

## Вскипятить океан

То, что микроформаты использовались в качестве модели для разработки HTML5, приходится не всем по вкусу. Хотя правило 80/20 достаточно хорошо работает для сермяжного мира наименований классов, действительно ли оно достаточно хорошо для самого важного языка разметки в мире?

Некоторые считают, что HTML должен быть бесконечно расширяемым. Это значит, что давать решения для большинства случаев недостаточно: язык должен предоставлять решения для любого возможного сценария.

Пожалуй, самый красноречивый аргумент такого типа привел Джон Олсоп (John Allsopp) в своей великолепной статье на A List Apart, «Семантика в HTML5» (http://bkaprt.com/html5/6)[11]:

> *Нам не нужно, чтобы в словарь HTML добавляли специфические термины, нам нужно добавить механизм, который позволит добавлять семантическую насыщенность к документу по мере необходимости.*

Уже существуют технологии для того, чтобы делать именно это. RDFa позволяют авторам встраивать в HTML-документы собственные словари. Но в отличие от микроформатов – которые просто используют заранее оговоренный набор наименований классов, – RDFa использует пространства имен для бесконечного разнообразия форматов. Так, там, где микроформат будет использовать примерно такую разметку: `<h1 class="summary">`, RDFa будет использовать: `<h1 property="myformat:summary">`.

Нет никакого сомнения в том, что RDFa – потенциально очень мощный инструмент, но его выразительность имеет свою цену. Пространства имен вводит дополнительный уровень сложности, который не очень согласуется с относительно простой природой HTML.

Спор о пространствах имен не нов. В записи в своем блоге несколько лет назад Марк Ноттингем (Mark Nottingham) размышлял о потенциально деструктивных побочных эффектах (http://bkaprt.com/html5/7)[12]:

> *Что мне представляется интересным относительно расширяемости HTML – то, что пространства имен были необязательными: Netscape добавила blink, MSFT – marquee и т. п. Я бы сказал, что если бы в HTML с самого начала были пространства имен, это имело бы следующий эффект: вместо того чтобы (в конце концов) сойтись на одном и том же решении, различия между разными браузерами были бы легитимизированы и окончательно закреплены.*

Помимо даже темы бесконечной расширяемости, это сильный аргумент за ограниченный словарь, построенный на согласии сообщества.

Скорее всего, HTML5 будет выпущен с каким-либо методом расширения его встроенной семантики. Конечно, атрибут class все еще на месте, поэтому микроформаты будут работать так, как работали. Возможно, HTML5 изменится, чтобы стать совместимым с RDFa, или, возможно, он будет использовать свой собственный словарь «микроданных».

В любом случае такая расширяемость, скорее всего, мало будет интересовать большинство веб-разработчиков. Что действительно имеет значение – это встроенная семантика, относительно которой существует согласие в сообществе и которая реализована производителями браузеров.

---

[11] Полная ссылка: http://www.alistapart.com/articles/semanticsinHTML5/.

[12] Полная ссылка: http://www.mnot.net/blog/2006/04/07/extensibility/.

# Новые элементы

HTML5 вводит несколько новых строчных элементов, чтобы расширить наш существующий арсенал, состоящий из `span`, `strong`, `em`, `abbr` и других. Ах да, больше мы не называем такие элементы «строчными» – теперь они описывают «семантику на уровне текста».

## mark

Когда вы пролистываете список результатов поиска, вы заметите, что зачастую поисковый запрос подсвечен внутри каждого из результатов поиска. Вы можете разметить каждое вхождение поискового запроса элементом `span`, но `span` – костыль, не имеющий никакого семантического значения и годящийся мало на что, кроме как предоставлять место, куда можно было бы сунуть классы для применения стилей.

Можно использовать `em` или `strong`, но это не будет семантически верным; вы не хотите придавать особую важность запросу поиска, просто хотите, чтобы он был как-то выделен.

На сцену выходит элемент `mark`:

```
<h1>Результаты поиска по запросу 'единорог'</h1>
<ol>
<li><a href="http://clearleft.com/">
Едем на <mark>единороге</mark> юзабилитипо радуге веба.
</a></li>
</ol>
```

Элемент `mark` не придает значения содержимому внутри него, а только показывает, что в данный момент он представляет интерес. Как говорит спецификация, `mark` означает «отрезок текста в одном документе, отмеченный или подсвеченный для справочных целей в связи с его релевантностью в другом контексте».

Элемент `mark` разрешается использовать и в других контекстах, кроме как в результатах поиска, но, убейте меня, я не могу придумать ни одного такого примера.

## time

hCalendar – один из самых популярных микроформатов, потому что он удовлетворяет общую для многих потребность: размечать события так, чтобы пользователи могли добавлять их напрямую в свой календарь.

Единственная сложная часть в hCalendar – описывать дату и время так, чтобы компьютер мог их прочитать. Люди любят описывать даты: «25 мая» или: «в следующую среду», но парсеры хотят видеть красиво отформатированную по ISO дату: YYYY-MM-DDThh: mm: ss.

Сообщество по микроформатам придумало несколько умных решений этой проблемы, например использование элемента `abbr`:

```
<abbr class="dtstart" title="1992-01-12">
12 января 1992
</abbr>
```

Если от того, что вы используете элемент `abbr` таким образом, вас начинает немножко мутить, есть много других способов размечать машиночитаемые даты и время в

микроформатах с помощью шаблона класс-значение. В HTML5 эта проблема разрешается новым элементом time:

```
<time class="dtstart" datetime="1992-01-12">
12 января, 1992
</time>
```

Элемент time может использоваться для обозначения даты, времени или того и другого вместе:

```
<time datetime="17:00">17 часов</time>
<time datetime="2010-04-07">7 апреля</time>
<time datetime="2010-04-07T17:00">в 17 часов, 7 апреля</time>
```

Вам необязательно ставить значение даты и времени в атрибут datetime, но если вы этого не делаете, то должны отобразить значение для пользователя:

```
<time>2010-04-07</time>
```

## meter

Элемент meter может использоваться для разметки любых измерений, если эти измерения являются частью шкалы с минимальным и максимальным значением.

```
<meter>9 из 10 кошек</meter>
```

Если вы не хотите, можете не выводить максимальное значение, а использовать вместо него атрибут max:

```
<meter max="10">9 кошек</meter>
```

Имеется также соответствующий атрибут min. Есть еще атрибуты high, low и optimum, с которыми тоже можно поиграть. Если хотите, вы можете вообще спрятать сам результат измерения в атрибут value.

```
<meter low="-273" high="100" min="12" max="30"optimum="21"
value="25">
Для этого времени года довольно-таки тепло.
</meter>
```

## progress

Если meter хорошо подходит для описания чего-то, что уже было измерено, элемент progress позволяет вам разметить значение, которое меняется сейчас:

```
Ваш профиль заполнен на <progress>60%</progress>.
```

И снова у вас есть атрибуты min, max и value, если вы предпочитаете использовать их:

```
<progress min="0" max="100" value="60"></progress>
```

Элемент progress особенно полезен, когда используется вместе со скриптами для DOM. Вы можете использовать JavaScript для динамического обновления значения, позволяя браузеру тем самым сообщить это изменение пользователю – это весьма кстати для Ajax-загрузок файлов.

# Структура

Еще в 2005 году компания Google провела ряд исследований, чтобы обнаружить, какие именно низко висящие плоды можно найти на тропках веба (http://code.google.com/webstats/ ).

Парсер осмотрел более миллиарда веб-страниц и разметил самые частые названия классов. Результаты не принесли ничего неожиданного. Наиболее частыми были такие названия классов, как “header”, “footer” и “nav”. Эта зарождавшаяся в то время семантика отлично соотносится с новыми структурными элементами, появившимися в HTML5.

## section

Элемент `section` используется для группировки тематически связанного содержимого. Это звучит очень похоже на элемент `div`, который зачастую используется как наиболее общий контейнер содержимого. Разница в том, что у `div` нет никакого семантического значения, его наличие ничего не говорит вам о содержимом внутри него. Напротив, элемент `section` явно используется для группировки связанного содержимого.

Вы вполне можете заменить ряд элементов `div` в своем коде элементами `section`, но не забудьте спросить себя: «Все ли содержимое связано друг с другом?»

```
<section>
<h1>Скрипты для DOM</h1>
<p>Эта книга предназначена скорее для верстальщиков,чем для программистов.</p>
<p>автор: Джереми Кит</p>
</section>
```

## header

Спецификация HTML5 описывает элемент `header` как контейнер для «группы вводных или навигационных вспомогательных элементов». Звучит вполне разумно. Это как раз тот тип содержимого, который я ожидаю увидеть в шапке страницы, и слово header как раз традиционно используется как синоним термина «шапка».

Между элементом `header` в HTML5 и общепринятым использованием слов header и «шапка» есть принципиальная разница. На странице шапка обычно только одна, но в документе может быть много элементов `header`. Вы можете, например, использовать элемент `header` внутри элемента `section`. Пожалуй, это даже нужно делать. Спецификация определяет элемент `section` как «контейнер тематически сгруппированного содержимого, как правило, с заголовком».

```
<section>
<header>
<h1>Скрипты для DOM</h1>
</header>
<p>Эта книга скорее предназначена для дизайнеров, чем для программистов.</p>
<p>автор: Джереми Кит</p>
</section>
```

Элемент header обычно появляется в начале документа или секции, но это не обязательно должно быть так. header определяется в большей степени своим содержимым – вводными или навигационными вспомогательными элементами, – чем положением.

## footer

Как и элемент header, по названию элемента footer кажется, что это описание положения, но, как и в случае с header, это не так. Вместо этого элемент footer должен содержать информацию об элементе, который его содержит: кто его автор, информацию о копирайте, ссылки на связанное содержимое и т. п.

Это отлично согласовывается с той ментальной моделью, которая есть у веб-разработчиков для слова «подвал». Разница в том, что, хотя мы привыкли использовать один подвал на весь документ, HTML5 позволяет нам делать подвалы и внутри секций.

```
<section>
<header>
<h1>Скрипты для DOM</h1>
</header>
<p>Эта книга скорее предназначена для дизайнеров, чем для программистов.</p>
<footer>
<p>автор: Джереми Кит</p>
</footer>
</section>
```

## aside

Так же как элемент header соответствует концепту шапки документа, элемент aside соответствует концепту боковой колонки. Когда я говорю «боковая колонка», я говорю не о положении. Одного того, что какое-то содержимое появляется слева или справа от главного содержимого, недостаточно для того, чтобы использовать элемент aside. Опять же имеет значение содержимое, а не положение.

Элемент aside должен использоваться для не связанного напрямую содержимого. Если у вас есть блок содержимого, который вы считаете отдельным от основного содержимого, тогда, возможно, его следует заключить в элемент aside. Задайте себе вопрос: можно ли удалить содержимое элемента aside так, что при этом главное содержимое документа или секции не потеряет смысл?

Хороший пример не связанного напрямую содержимого – врезки; они хорошо смотрятся, но вы можете убрать их, и это никак не повредит пониманию основного содержимого.

Помните, если ваш дизайн ставит какое-либо содержимое в боковую колонку, это еще не означает, что это содержимое должно находиться именно в aside. Например, довольно часто в боковую колонку ставится биография автора. Этот тип данных лучше подходит для того, чтобы быть внутри элемента footer, – спецификация явным образом утверждает, что информация об авторстве подходит для подвалов (**рис. 5.01**).


**Рис. 5.01.** Текст «об авторе» в этом скриншоте должен быть размечен с помощью footer, а не aside

В девяноста процентах случаев шапки будут расположены сверху от вашего содержимого, подвалы – внизу, а боковые колонки – по одной из сторон. Но не расслабляйтесь. Держите ухо востро и не пропустите оставшиеся десять процентов.

## nav

Элемент `nav` выполняет ту самую функцию, которую, как вы предполагаете, он и должен выполнять. Он содержит навигационную информацию, обычно – список ссылок.

На самом деле давайте я лучше объясню. Элемент `nav` предназначен для очень важной информации по навигации. Одно то, что группа ссылок сгруппирована вместе в список, недостаточная причина для того, чтобы использовать элемент `nav`. С другой стороны, сквозная навигация по сайту почти наверняка должна содержаться в элементе `nav`.

Довольно часто элемент `nav` появляется внутри элемента `header`. Это вполне осмысленно, если вы вспомните, что элемент `header` может использоваться для «вспомогательной навигационной информации».

## article

Для лучшего понимания можно представить, что элементы `header`, `footer`, `nav` и `aside` – это специализированные формы элемента section. Секция – это общий блок связанного содержимого, а шапки, подвалы, навигационные блоки и боковые колонки – блоки особого связанного содержимого.

Элемент `article` – еще один особый вид `section`. Его следует использовать для самостоятельного связанного содержимого. Теперь сложная часть – это решить, что значит «самостоятельный».

Задайте себе вопрос, стали бы вы передавать это содержимое в RSS или Atom-ленте. Если в таком конктексте это содержимое имеет смысл, тогда, скорее всего, `article`

– нужный вам элемент. На самом деле элемент article спроектирован специально для агрегирования.

Если внутри `article` вы используете элемент time, можете добавить к нему необязательный булев атрибут `pubdate`, чтобы указать, что он содержит дату публикации:

```
<article>
<header>
<h1>Отзыв на книгу Скрипты для DOM</h1>
</header>
<p>Маленький маяк, который освещает длинную и зачастую темную дорогу в мире JavaScript.</p>
<footer>
<p>Опубликовано в
<time datetime=?2005-10-08T15:13? ачастую темную дорогу в мире JavaScript.</p>
<footer>
<p>Опубликовано в
<time datetime="2005-10-08T15:13" pubdate>
15:13 8 октября 2005 года.
</time>
автор: Гленн Джонс</p>
</footer>
</article>
```

Если внутри статьи у вас есть несколько элементов `time`, то атрибут `pubdate` может быть только у одного из них.

Элемент `article` особенно хорошо применим для записей в блогах, новостных репортажей, комментариев, оценок, записей на форумах. Он покрывает ровно те же сценарии, что и микроформат `hAtom`.

Но спецификация HTML5 идет дальше. Она также объявляет, что элемент `article` должен использоваться для самостоятельных виджетов: графиков акций, калькуляторов, часов, виджетов погоды и тому подобного. В этом случае элемент `article` пытается закрыть те же сценарии, что Web Slices компании Microsoft (http://bkaprt.com/html5/8)[13].

Мне кажется совершенно непонятным интуитивно, что элемент с названием «статья» должен применяться к концепту, известному как «виджет». Но, опять же, и статьи, и виджеты – самостоятельные, агрегируемые типы содержимого.

Что более проблематично – это то, что `article` и `section` очень во многом похожи. Все, что их разделяет, – слово «самостоятельный». Решить, какой элемент к чему относится, было бы просто при четких и быстрых правилах. В данном же случае все зависит от интерпретации. У вас может быть много статей внутри секции, много секций внутри статьи, можно создавать вложенные секции в секциях и статьи в статьях. Вы решаете, какой элемент больше подходит семантически в той или иной конкретной ситуации.

## Лекарство от избытка дивов?

HTML5 дает нам массу новых структурных элементов, которые описаны выше. Они особенно полезны, если вы разрабатываете обыкновенный сайт, например блог. Большинство блогов разработаны так, что сначала идет шапка, затем набор статей,

[13] http://www.ieaddons.com/en/webslices/

не имеющее прямого отношения содержимое в отдельном элементе и завершается все подвалом (**рис. 5.02**).

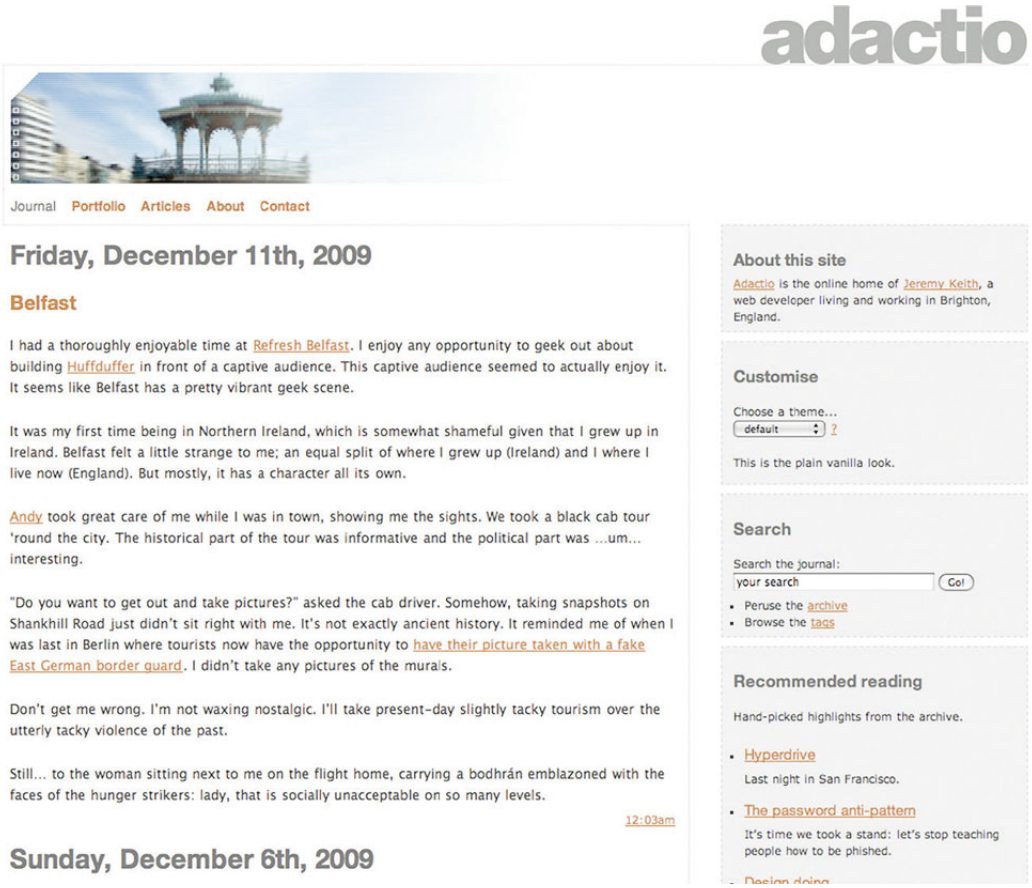


**Рис. 5.02.** Блог вашего покорного слуги

Теперь вы можете заменить какие-то из своих элементов `div` на структурные элементы с большей семантической точностью. Но не переходите границу. Есть шансы, что если вы используете `div` сегодня, будете использовать его же и завтра. Не заменяйте ваши элемент `div` на новые глянцевые элементы HTML5 просто ради того, чтобы их использовать. Подумайте о содержимом.

Новые элементы были созданы не просто для того, чтобы заменить элементы `div`. Они наделяют веб-браузеры совершенно новыми возможностями для понимания вашего содержимого.

# Модели содержимого

Предыдущие спецификации разметки разделяли элементы на строчные и блочные. HTML5 использует более тонкий подход, разделяя элементы на более широкий спектр категорий.

Строчные элементы теперь имеют модель содержимого «семантики на уровне текста». Многие блочные элементы теперь подпадают под категорию «группирующего содержимого»: абзацы, списки, дивы и т. п. У форм есть своя собственная модель содержимого. Картинки, звук, видео и Canvas относятся к встроенному содержимому. У новых структурных элементов появляется совершенно новая модель содержимого, которая называется «содержимое-разделитель».

## Содержимое, разбивающее на секции

Используя элементы заголовков, от `h1` до `h6`, можно создать содержание HTML-документа. Например, посмотрите вот на эту разметку:

```
<h1>An Event Apart</h1>
<h2>Города</h2>
<p>Присоединяйтесь к нам в 2010 году в этих городах.</p>
<h3>Сиэтл</h3>
<p>Идите в изумрудный город по дороге из желтого кирпича.</p>
<h3>Бостон</h3>
<p>Для друзей – Beantown.</p>
<h3>Миннеаполис</h3>
<p>Здесь так <em>мило</em>.</p>
<small>Размещение не предоставляется.</small>
```

Из этого получается следующее содержание:

- An Event Apart
- Города
- Сиэтл
- Бостон
- Миннеаполис

Это работает достаточно неплохо. Все содержимое, следующее за элементом заголовка, считается связанным с этим заголовком.

Теперь посмотрим на элемент `small`. Он должен быть связан со всем документом. Но браузер никак не может об этом узнать. Он ниоткуда не может узнать, что элемент `small` не должен относиться к заголовку «Миннеаполис».

Добавленное в HTML5 содержимое-разделитель позволяет вам явно размечать начало и конец взаимосвязанного содержимого:

```
<h1>An Event Apart</h1>
<section>
<header>
<h2>Города</h2>
</header>
<p>Присоединяйтесь к нам в 2010 году в этих городах.</p>
<h3>Сиэтл</h3>
<p>Идите в изумрудныйгород по дороге из желтого кирпича.</p>
<h3>Бостон</h3>
```

```
<p>Для друзей – Beantown.</p>
<h3>Миннеаполис</h3>
<p>Здесь так <em>мило</em>.</p>
</section>
<small>Размещение не предоставляется.</small>
```

Теперь ясно, что элемент `small` подпадает под заголовок “An Event Apart”, а не «Миннеаполис».

Я могу разделить это содержимое на еще более мелкие части: тогда каждый город окажется в своей собственной секции:

```
<h1>An Event Apart</h1>
<section>
<header>
<h2>Города</h2>
</header>
<p>Присоединяйтесь к нам в 2010 году в этих городах.</p>
<section>
<header>
<h3>Сиэтл</h3>
</header>
<p>Идите в изумрудныйгород по дороге из желтого кирпича.</p>
</section>
<section>
<header>
<h3>Бостон</h3>
</header>
<p>Для друзей – Beantown.</p>
</section>
<section>
<header>
<h3>Миннеаполис</h3>
</header>
<p>Здесь так <em>мило</em>.</p>
</section>
</section>
<small>Размещение не предоставлятся.</small>
```

Содержание при этом будет таким же:

- An Event Apart
- Города
- Сиэтл
- Бостон
- Миннеаполис

## Алгоритм содержания

Пока новое содержимое-разделитель не дает нам ничего такого, чего мы не могли бы делать с предыдущими версиями HTML. Вот самое интересное: в HTML5 у каждого содержимого-разделителя есть свое собственное внутреннее содержание. Таким образом, вам не нужно следить, на каком уровне заголовков вы находитесь, – можете просто каждый раз начинать с `h1`:

```
<h1>An Event Apart</h1>
<section>
<header>
<h1>Города</h1>
</header>
<p>Присоединяйтесь к нам в 2010 году в этих городах.</p>
<section>
<header>
<h1>Сиэтл</h1>
</header>
<p>Идите в изумрудныйгород по дороге из желтого кирпича.</p>
</section>
<section>
<header>
<h1>Бостон</h1>
</header>
<p>Для друзей – Beantown.</p>
</section>
<section>
<header>
<h1>Миннеаполис</h1>
</header>
<p>Здесь так <em>мило</em>.</p>
</section>
</section>
<small>Размещение не предоставляется.</small>
```

В предыдущих версиях HTML содержание было бы построено неправильно:

- An Event Apart
- Города
- Сиэтл
- Бостон
- Миннеаполис

В HTML5 содержание строится правильно:

- An Event Apart
- Города
- Сиэтл
- Бостон
- Миннеаполис

## hgroup

Бывают времена, когда вы хотите использовать элемент заголовка, но не хотите, чтобы его содержимое появлялось в содержании документа. Именно это позволяет вам сделать элемент `hgroup`:

```
<hgroup>
<h1>An Event Apart</h1>
<h2>Для людей, которые создают вебсайты</h2>
</hgroup>
```

В этом случае заголовок второго уровня («Для людей, которые создают веб-сайты») – это на самом деле слоган. В элементе `hgroup` только первый заголовок войдет в содержание. Первый заголовок не обязательно должен быть `h1`:

```
<hgroup>
<h3>Скрипты для DOM</h3>
<h4>Веб-разработка на JavaScriptдля Document Object Model</h4>
</hgroup>
```

## Корневые элементы разделов

Некоторые элементы не появляются в сгенерированном содержании. Другими словами, неважно, сколько заголовков вы используете внутри своих элементов, – в содержании документа они не появятся.

Элементы `blockquote`, `fieldset` и `td` являются исключениями для алгоритма составления содержания. Эти элементы называются «корневыми элементами разделов» – это не нужно путать с элементами-разделителями.

## Переносимость

Так как каждый элемент, относящийся к типу контента-разделителя, создает собственное содержание, теперь в вашем документе может быть гораздо больше заголовков, чем просто от `h1` от `h6`. Теперь нет никакого ограничения на то, как глубоко могут спускаться уровни ваших заголовков. Что более важно, теперь вы можете начать представлять свое содержимое в более модульном ключе.

Предположим, у меня есть запись в блоге под названием «Бутерброд с сыром». До HTML5 мне нужно было бы знать контекст, в котором расположена запись, чтобы определить, какой уровень заголовков использовать для заголовка записи. Если запись находится на главной странице, то она появляется вслед за элементом `h1`, который содержит название моего блога:

```
<h1>Мой клёвый блог</h1>
<h2><a href="cheese.html">Бутерброд с сыром</a></h2>
<p>Моя кошка съела бутерброд с сыром.</p>
```

Но если я публикую запись в блоге на отдельной странице, то я хочу, чтобы заголовок записи был заголовком первого уровня:

```
<h1>Бутерброд с сыром</h1>
<p>Моя кошка съела бутерброд с сыром.</p>
```

С HTML5 мне не нужно волноваться о том, какой уровень заголовков использовать. Мне нужно просто использовать содержимое-разделитель: в данном случае – элемент `article`:

```
<article>
<h1>Бутерброд с сыром</h1>
<p>Моя кошка съела бутерброд с сыром.</p>
</article>
```

Теперь это содержимое по-настоящему переносимо. Неважно, где оно появляется, на отдельной странице или на главной странице блога:

```
<h1>Мой клёвый блог</h1>
<article>
<h1>Бутерброд с сыром</h1>
<p>Моя кошка съела бутерброд с сыром.</p>
```

```
</article>
```

Новый алгоритм составления HTML5 выдает правильный результат:

- Мой клёвый блог
- Бутерброд с сыром

## Локальные стили

То, что у каждого элемента-разделителя есть свое собственное содержание, делает эти элементы прекрасно подходящими для решений на Ajax. И снова HTML5 показывает свое происхождение из спецификации для веб-приложений.

Однако если вы попытаетесь перенести кусочек содержимого из одного документа в другой, то столкнетесь с рядом проблем. CSS-правила, примененные к главному документу, будут применены и к вставленному содержимому. Это в настоящий момент одна из главных проблем с распространением виджетов в вебе.

HTML5 предлагает решение этой проблемы в виде атрибута `scoped`, который можно применить к элементу `style`. Все стили, объявленные внутри этого элемента `style`, будут применены только к ближайшему родительскому элементу-разделителю:

```
<h1>Мой клёвый блог</h1>
<article>
<style scoped>
h1 { font-size: 75% }
</style>
<h1>Бутерброд с сыром</h1>
<p>Моя кошка съела бутерброд с сыром.</p>
</article>
```

В этом примере только у второго элемента `h1` будет значение размера шрифта, равное 75%. По крайней мере теория такова. Ни один браузер еще не поддерживает атрибут `scoped`.

В этом-то и загвоздка. Перед тем как начать использовать какое-либо новое добавление в HTML5, вам нужно учесть, какие браузеры поддерживают эту функцию. Впрочем, какой бы ни была браузерная поддержка, у меня есть несколько стратегий, которые помогут вам начать работать с HTML5. В следующей, последней главе я хотел бы поделиться с вами этими стратегиями.

# 6. Использование HTML5 сейчас

Если вы хотите начать использовать новые структурные элементы HTML5 прямо сейчас, вам ничто не мешает. Большинство браузеров позволят вам назначать стили новым элементам. Дело не в том, что браузеры активно поддерживают эти элементы, а в том, что большинство браузеров позволяют использовать и назначать стили любому элементу, который вы захотите изобрести.

## Стили

Браузеры по умолчанию не применяют к новым элементам никаких CSS-стилей. Так что по крайней мере вы, скорее всего, захотите объявить, что все новые структурные элементы должны начинаться с новой строки:

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

Для большинства браузеров этого достаточно. Internet Explorer требует к себе особого внимания. Он решительно отказывается признать новые элементы, если только экземпляр каждого элемента не создан заранее с помощью JavaScript, вот так:

```
document.createElement('section');
```

Реми Шарп (Remy Sharp), гений JavaScript, написал очень полезный скриптик, который генерирует все новые элементы HTML5. Загрузите этот скрипт внутри условного комментария, так что он будет исполняться только в нуждающемся Internet Explorer:

```
<!-[if IE]>
<script src="http://html5shiv.googlecode.com/svn/trunk/
html5.js">
</script>
<![endif]->
```

Теперь вы можете назначать стили новым элементам сколько душе угодно.

## Заголовки

Браузеры еще не начали поддерживать новый алгоритм содержания в HTML5, но вы все равно можете начать использовать доступные вам дополнительные уровни заголовков.

Джоффри Снеддон (Geoffrey Sneddon) написал очень полезную онлайн-утилиту, которая сгенерирует содержание по спецификациям HTML5 (http://bkaprt.com/html5/9)[14].

Если вы последуете совету, изложенному в спецификации HTML5, и начнете с нуля, начиная с `h1`, внутри каждого раздела вашего содержимого CSS-правила, которые вы напишете, очень быстро станут очень запутанными:

```
h1 {
font-size: 2.4em;
}
h2, section h1, article h1, aside h1 {
font-size: 1.8em;
}
h3, section h2, article h2, aside h2,
section section h1, section article h1, section aside h1,
article section h1, article article h1, article aside h1,
aside section h1, aside article h1, aside aside h1 {
font-size: 1.6em;
}
```

Это только первые три уровня, и приведенный код не покрывает все возможные комбинации заголовков внутри содержимого разделов. К счастью, алгоритм содержания

[14] Полная ссылка: http://gsnedders.html5.org/outliner

HTML5 довольно гибок. Если вы хотите использовать уровни заголовков по старинке, это никоим образом не отразится на содержании.

# Aria

Новые структурные элементы HTML5 будут очень полезны для технологии специальных возможностей. Вместо того чтобы создавать ссылки «пропустить навигацию», все, что нужно делать, – правильно использовать элемент `nav`. Это позволит пользователям программ для чтения с экрана пропустить навигацию, а нам не придется создавать для этого отдельную ссылку.

По крайней мере план именно такой. Но прямо сейчас мы должны работать с технологиями, которые браузеры и программы чтения с экрана уже поддерживают.

К счастью, в настоящий момент отличную поддержку имеет стандарт ARIA (доступные насыщенные интернет-приложения, Accessible Rich Internet Applications).

На самом высоком уровне ARIA позволяет технологиям специальных возможностей в полной мере работать с самыми разнообразными Ajax-взаимодействиями с веб-приложением. На самом простом уровне ARIA позволяет нам добавить семантической насыщенности в наши документы.

Самая основная единица ARIA – атрибут `role`. Вы можете добавить `role="search"` к вашей форме поиска, `role="banner"` к красивому заголовку вашего сайта и `role="contentinfo"` – к подвалу страницы. Полный список значений в спецификации ARIA можно посмотреть здесь: http://bkaprt.com/html5/10[15].

Вы можете использовать эти значения ролей в HTML 4.01, XHTML 1.0 и любом другом типе разметки, но тогда ваш документ не будет проходить валидацию – если только вы не создадите собственный доктайп, а это страшное занятие.

Но роли ARIA – часть спецификации HTML5, так что можно и использовать ARIA, и проходить валидацию.

Вы также можете использовать дополнительную семантику атрибута `role` для добавления стилей. Здесь вам поможет селектор по атрибутам. Такие селекторы позволят вам отличать заголовки и подвалы документов от заголовков и подвалов внутри содержимого разделов:

```
header[role="banner"] { }
footer[role="contentinfo"] { }
```

[15] Полная ссылка: http://www.w3.org/TR/wai-aria/roles#role_definitions/.

# Валидация

Если использовать его с умом, валидатор – очень мощный инструмент для веб-разработчика. Если нет, то валидатор дает ботаникам и занудам повод показывать пальцем на чужую разметку и смеяться над ней.

Анри Сивонен (Henri Sivonen) разработал полнофункциональный валидатор HTML5. Он находится по адресу: http://validator.nu/.

Вам даже не нужно обновлять свои закладки, ведущие на валидатор W3C (http://validator.w3.org/). Этот валидатор тоже использует парсер Анри, как только находит доктайп HTML5.

## Тестирование функций

Если вы хотите начать использовать продвинутые типы ввода HTML5, вам нужен способ тестирования поддержки браузеров, чтобы вы могли вставить в код альтернативные варианты на JavaScript.

Modernizr – очень полезная JavaScript-библиотека, которая определяет поддержку различных типов ввода, а также `audio`, `video` и `canvas` (http:// www.modernizr.com/).

Этот скрипт создает в JavaScript объект с названием Modernizr. Запрашивая свойства этого объекта, вы можете определить, поддерживает браузер тот или иной тип ввода или нет:

```
if (!Modernizr.inputtypes.color) {
// Запасное решение на JavaScript.
}
```

Modernizr также исполнит маленький шулерский трюк, который позволит вам применять стили к новым структурным элементам Internet Explorer, – так что если вы используете Modernizr, то не нужно использовать вместе с ним скрипт Реми.

## Выберите собственную стратегию

Только вы решаете, как будете использовать HTML5 – осторожно или, наоборот, амбициозно.

По крайней мере что вы можете сделать без всяких проблем – это взять ваши текущие HTML– или XHTML-документы и изменить доктайп на:

```
<!DOCTYPE html>
```

Ну вот, вы только что сделали первый шаг в большой мир.

Теперь можете начать использовать роли ARIA – что вам, собственно, терять?

Если вы беспокоитесь по поводу новых структурных элементов, все равно можете начать привыкать к новой семантике, потренировавшись на именах классов:

```
<div class="section" >
<div class="header" >
<h1>Hello world!</h1>
</div><!- /.header - >
</div><!- /.section - >
```

По мере того как вы будете дальше продвигаться в мир HTML5, когда будете чувствовать себе более уверенно относительно новых элементов HTML5, можете заменить эти элементы `div` и имена классов соответствующими структурными элементами.

Если, пожалуй, еще слишком рано использовать более продвинутые типы ввода, такие как `date`, `range` и `color`, нет ничего плохого в том, чтобы использовать `search`, `url`, `email` и другие простые типы ввода. Не забывайте, что браузеры, которые не понимают этих значений, будут просто отображать поле ввода, как если бы оно было обозначено как `type="text"`.

Если вы чувствуете в себе дух искателя приключений, можете начать играть с `audio`, `video` и `canvas`. Может быть, они не вполне готовы для крупных аудиторий, но это точно забавные игрушки, с которыми можно поэкспериментировать на вашем личном сайте.

## Ресурсы

Я довольно часто пишу о HTML5 на своей `личной` страничке: http://adactio.com/journal/tag/html5

Я не единственный человек в мире, который с нетерпением ждет HTML5. Невероятный Брюс Лосон (Bruce Lawson) тоже записывает свои мысли: http://brucelawson.co.uk/category/html5/

Брюс – только один из активных участников HTML5 Doctor – отличного ресурса, сообщество которого пишет здесь множество прекрасных статей: http://html5doctor.com/

Если вы раздумываете, не взглянуть ли на более сложную сторону HTML5, то Реми Шарп выжимает из этого языка разметки все, что возможно: http://html5demos.com/

Марк Пилгрим (Mark Pilgrim) написал исчерпывающую книгу под названием «Погружение в HTML5» (Dive Into HTML5). Вы можете купить ее на сайте издательства O'Reilly или прочитать онлайн: http://diveintohtml5.org/

На тот случай, когда вам нужно отправиться напрямую к исходной точке, держите спецификацию HTML5 на кнопке быстрой загрузки: http://whatwg.org/html5

Спецификация HTML5 включает в себя большое количество информации, предназначенной для производителей браузеров. На сайте W3C есть актуальная версия спецификации специально для веб-разработчиков: http://www.w3.org/TR/html-markup

# Включайтесь!

Отправляясь в путешествие по стране HTML5, вам может показаться, что какие-то разделы спецификации запутанны. Это нормально. Более того, это будет очень полезной обратной связью.

Над HTML5 работают очень умные люди, но веб-разработчиков среди них не так много. Ваша точка зрения будет крайне ценной.

Вы можете вступить в рабочую группу HTML при W3C в качестве приглашенного эксперта – не обращая внимания на кафкианский язык приглашения, которое вам нужно будет выписать себе от своего же лица, – но я бы не советовал этого делать. В списке рассылки для экспертов огромное количество писем, и большинство из них касается политики и различных внутренних процедур.

Если вы хотите обсуждать именно спецификацию HTML5, вам нужен список рассылки WHATWG: http://www.whatwg.org/mailing-list#specs

Еще есть IRC-канал. Иногда хочется пойти туда, где каждый знает твой никнейм: irc://irc.freenode.org/whatwg

Не надо стесняться. Канал IRC – отличное место, где можно задать вопросы и получить ответы на них от Яна Хиксона, Анны ван Кестерен (Anne van Kesteren), Лахлана Ханта (Lachlan Hunt) и других членов WHATWG.

# Будущее

Я очень надеюсь, что эта быстрая прогулка вокруг HTML5 сподвигла вас захотеть начать изучать эту страшно интересную технологию. Я также надеюсь, что вы дадите знать WHATWG о плодах, которые это изучение принесет.

HTML – самый важный инструмент, которым владеет веб-разработчик. Без разметки не было бы веба. Мне кажется удивительным и прекрасным то, что каждый может внести свою лепту в эволюцию этой жизненно необходимой технологии. Каждый раз, когда вы создаете сайт, вы добавляете что-то к общему культурному наследству человечества. А выбирая HTML5, вы способствуете тому, чтобы будущее наступило раньше.

# Об авторе

Джереми Кит – ирландский веб-разработчик. Живет в Брайтоне (Англия), работает в компании по веб-консалтингу Clearleft. Автор уже двух книг, «Скрипты для DOM» и «Пуленепробиваемый Ajax». Мечтает стать режиссером. Его домашняя страничка – adactio.com, а последний проект – Huffduffer – сервис, который позволяет делать подкасты из звуков, найденных в Интернете. Когда Джереми не разрабатывает веб-сайты, он играет на бузуки в группе Salter Cane.