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, чтó 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

[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.*

`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.

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, 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>
```

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

```
    <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.

издательство
МАНН, ИВАНОВ И ФЕРБЕР
Максимально полезные книги