Живой стандарт: HTML5, "HTML5" и HTML

В настоящий момент все ведущие браузеры полностью поддерживают новый стандарт HTML5 (Mozilla Firefox, Google Chrome, Opera и даже Inetrnet Explorer теперь располагает поддержкой html5).

Термин HTML5 стал общим названием для всех развивающихся технологий. Разработчик JavaScript Петер-Пауль Кох (Peter Paul Koch) кратко описал этот феномен в заметке в своем блоге в январе 2010 года: "Термином HTML5 можно назвать все что угодно, если вы хотите, чтобы это звучало новомодно и круто".

1 Синтаксис HTML5

Ослабленные правила

HTML 5 будет иметь два синтаксиса - "custom" HTML и XML.

XML синтаксис совместим с документами XHTML1 и его реализациями. Чтобы использовать этот синтаксис нужно объявить MIME тип XML, а элементы должны быть выстроены согласно спецификации XML. Ниже приведен пример, который соответствует синтаксису XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Example document</title>
 </head>
 <body>
  <p>Example paragraph</p>
 </body>
</html>
```

При нашем первом знакомстве с разметкой HTML5 мы узнали, что использования элементов <html>, <head> и <body> не является обязательным для этой разметки. Но ослабление правил в HTML5 на этом не заканчивается.

В нем также разрешается использовать в тегах как прописные, так и строчные буквы, как в следующем примере:

<**P**>В тегах <**EM**>можно использовать</**eM**> как прописные, так и строчные буквы.</**p**>

Также можно не использовать закрывающую обратную косую черту в пустых элементах, т.е. элементах без содержимого, таких как <img> (изображение),
 (разрыв строки) или <hr> (горизонтальная линия). Далее приведены три равнозначных способа вставить разрыв строки:

Пример <**br** />
разрыва <**br**>
строки <**br**/>

В HTML5 также подверглись изменениям правила для атрибутов. Значения атрибутов больше не требуется брать в кавычки, но только при условии, что они не содержат запретных символов (обычно это символы >, = или пробел). Вот пример использования элемента <img> таким образом:

<**img** alt="Стандартное значение атрибута" src=myimage.jpg>

Также разрешены атрибуты без значений. Таким образом, если для установки флажка в XHTML требуется несколько повторяющийся синтаксис:

<!-- XHTML -->

<**input** type="checkbox" checked="checked" />

то в HTML5 это можно делать в традициях HTML 4.01, указывая только одно название атрибута:

<input type="checkbox" checked >

Далее дается краткое изложение основных принципов хорошего стиля создания разметки HTML5:

- Использование элементов <html>, <body> и <head>. В элементе <html> удобно размещать определение естественного языка страницы, а элементы <head> и <body> позволяют отделить информацию о странице от собственно содержимого страницы.
- Строчные буквы в тегах. Использование строчных букв в тегах не является обязательным, но такие теги намного больше распространены, их легче вводить (т. к. не требуется задействовать клавишу <Shift>), а также не так режут глаз, как теги с прописными буквами.
- Взятие в кавычки значений атрибутов. Значения атрибутов берутся в кавычки потому, что на это есть причина — помочь избежать ошибок, которые в противном случае очень легко допустить. Без кавычек один неправильный символ значения атрибута может испортить всю страницу.

Проверка кода HTML5

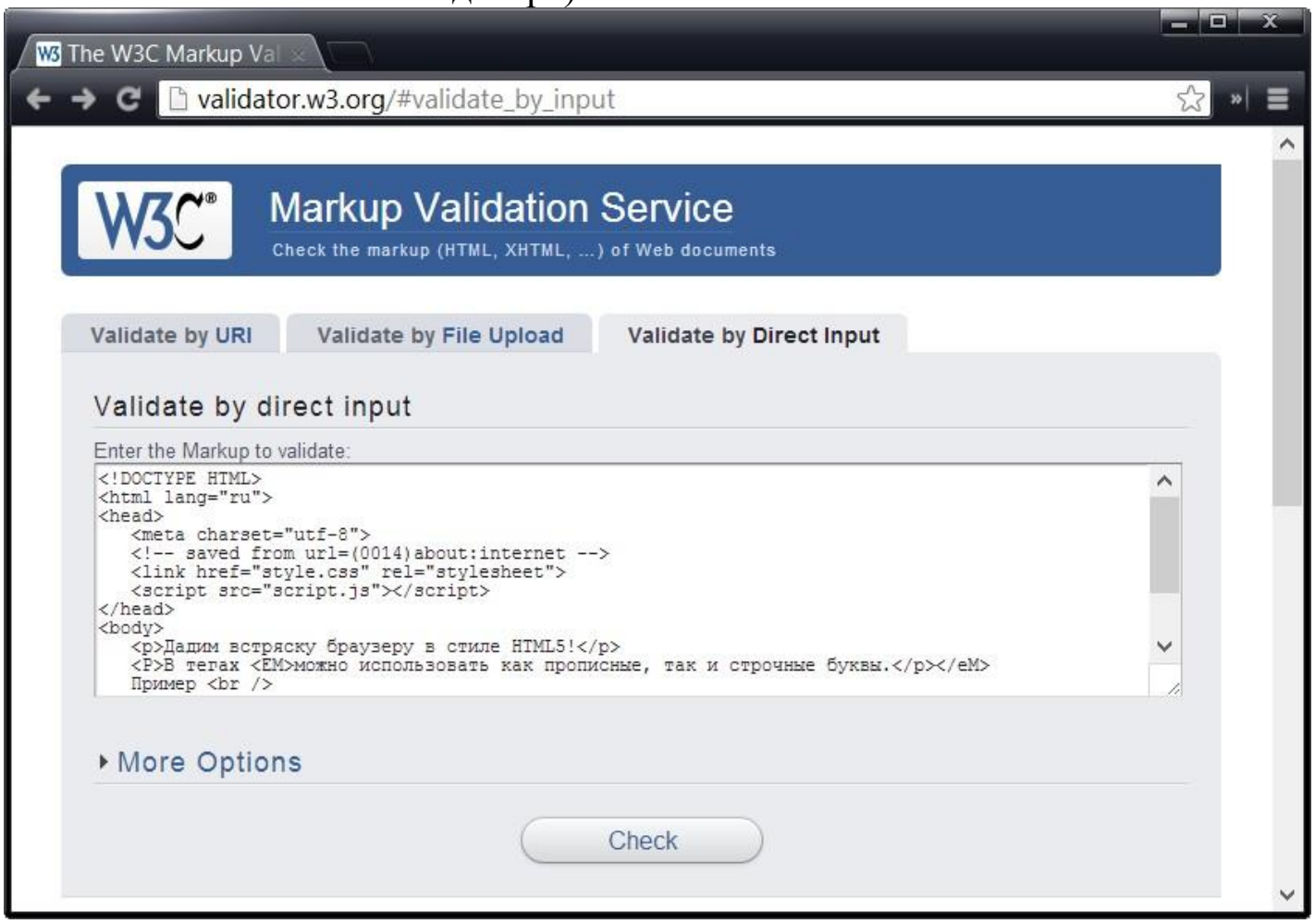
Инструмент для проверки правильности разметки, называющийся валидатором, может обнаружить код, который не соответствует рекомендуемым стандартам HTML5.

Некоторые из возможных проблем, которые валидатор в состоянии уловить, включают следующие:

- отсутствие обязательных элементов (например, элемента <title>);
- отсутствие закрывающего тега;
- неправильно внедренные теги;
- отсутствие атрибутов у тегов, для которых они обязательны (например, атрибута src тега <img>);
- неправильное расположение элементов или содержимого (например, текста в блоке <head>).

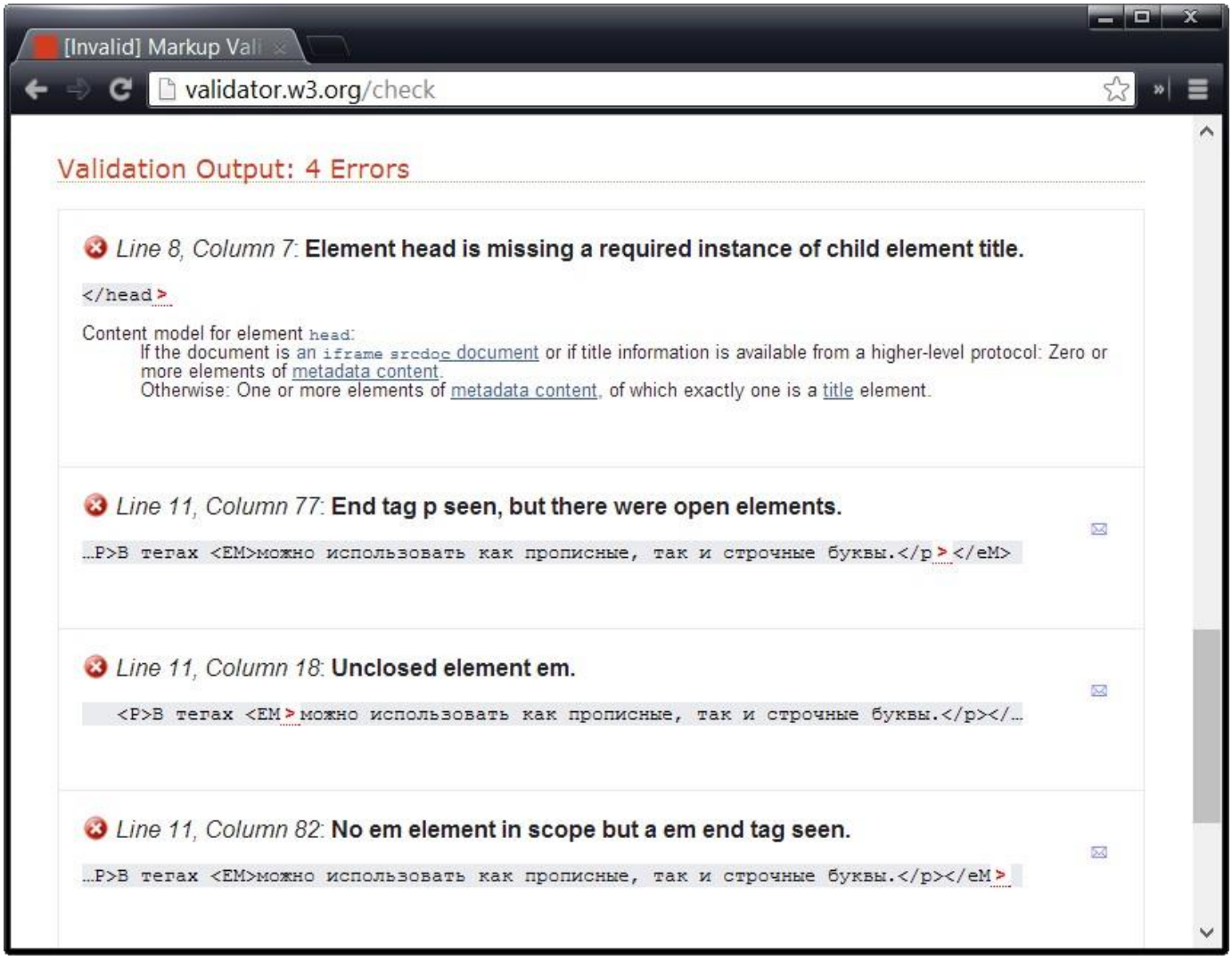
Инструменты для разработки веб-страниц, такие как Dreamweaver и Expression Web, оснащены собственными валидаторами, но только самые последние версии поддерживают HTML5. В таком случае можно воспользоваться одним из онлайновых валидаторов. Далее даются инструкции по использованию популярного валидатора от организации W3C:

1. Откройте в своем браузере страницу W3C Markup Validation Service. Валидатор предложит три способа проверки разметки, каждая на своей вкладке: Validate by URI (для страницы, которая уже размещена в интернете), Validate by File Upload (для страницы, сохраненной в файле на вашем компьютере) и Validate by Direct Input (для кода, вводимого или вставляемого в окно валидатора):



2. Выберите нужную вкладку и предоставьте свою HTML-разметку. Нажмите кнопку Check.

Ваш код будет отправлен на проверку, и после короткого ожидания в браузере будет выведен отчет с результатами валидации. Если код не прошел проверку, то в отчете будут указаны выявленные валидатором ошибки:



Даже для полностью правильного HTML-документа в отчете может быть указано несколько предупреждений (хотя полностью безобидных), включая такие, что кодировка была определена автоматически и услуга валидации кода HTML5 является экспериментальной и не совсем доведенной до логического конца.

Как можно видеть на рисунке, валидатор выявил в документе четыре нарушения правил HTML5, являющиеся результатом двух ошибок в коде. Первая ошибка — отсутствует обязательный элемент <title>. Вторая — элемент <p> закрывается до закрытия вложенного в него элемента . Тем не менее, несмотря на эти ошибки, эта разметка правильная, и все браузеры будут отображать эту страницу должным образом.

Новые элементы текстовой разметки и переопределение старых

Прежде чем переходить к обзору новых элементов html5, перенесём своё внимание на теги, которые постигла участь переопределения. Существуют теги-близнецы для выделения текста жирным шрифтом.

** **

С точки зрения визуального восприятия текст, заключённый в любую пару таких тегов, выглядит абсолютно неразличимо, если только дополнительно не изменить стили с помощью CSS. В этом плане наличие

двух пар тегов вносило не только некоторую путаницу, но и вызывало определённое недоумения. Зачем иметь два разных тега, если можно обойтись всего одним?

В html5 этот вопрос отпал, теперь по-прежнему визуальное различие отсутствует, однако вышеупомянутые теги имеют совершенно различное семантическое значение.

Фрагмент текста, выделенный тегом **strong**, принимает более важное значение, чем окружающий текст. Таким образом, это не просто выделенный жирным шрифтом текст, а важная смысловая единица страницы. Иными словами, тег **strong**, заключая внутрь себя часть текста, придаёт ему более высокий вес, нежели окружающий контент. К слову сказать, поисковые системы так уделяют семантике немалое значение и умеют оценивать вес текстовых фрагментов, так что пренебрегать семантическому значению блоков не стоит.

Тег **b** сохранил свою оформительную функцию, и по-прежнему выделяет текст жирным шрифтом, однако сохраняет вес такого фрагмента по отношению к окружающему тексту. Например, вы можете использовать тег **b** для выделения терминов в статье.

Аналогично в html сосуществует и другая пара сходных между собой по смыслу тегов:

```
<i></i>  <em></em>
```

Опять же, любая из этих пар тегов по умолчанию визуально выделяет текст с помощью курсива, так что раньше вы могли выбирать для использования любой из вариантов. Аналогично тегам выделения текста жирностью, html5 чётко разделил теги **i** и **em** по семантическому назначению.

Текст, выделенный при помощи **em**, получает дополнительное эмоциональное ударение, но в отличие от **strong** тег **em** означает не важность фрагмента, а делает акцент на нём при произношении.

Слова, обрамлённые тегом **i**, произносятся обычно, без логического ударения, но вместе с тем выделяются на странице курсивом. Чтобы было понятнее, приведу пример использования тегов **i** и **em**.

```
<em>Никогда</em> не говори <i>никогда</i> !
```

Первое слово обрамлено в **em**, что однозначно акцентирует на нём ударение, второе слово «никогда» лишь выделяется визуально на странице, как слово, о котором и идёт речь в предложении. При этом оба варианта на странице отображаются в виде курсивного выделения.

Наконец обратимся к последней паре:

```
<s></s>  <strike></strike>
```

Визуально любая из этих пар тегов заставляет браузер выводить текст в зачёркнутом виде, но как вы уже догадались, в html5 каждой из пар соответствует своя семантика. Оба элемента отображаются браузерами в виде зачёркнутого текста, но тег **s** семантически означает фрагмент, который утратил своё значение или же стал в корне не верным.

Раз уж заговорили об элементе html5 **del**, то нельзя не упомянуть о парном ему теге **ins**. Элемент **ins** является для **del** комплементарным и служит для указания фрагмента текста, который был вставлен в документ позже. Пример:

```
Для зачёркнутого текста следует использовать
<del> тег s </del> <ins>тег del</ins>,
для ознакомления читайте руководство
по html5 и комментарии.
```

Соответственно, в браузере фрагмент будет выглядеть так:

Для зачёркнутого текста следует использовать ~~тег s~~ тег del, для ознакомления читайте руководство по html5 и комментарии.

Как видите, фрагмент, вставленный фрагмент, выделенный тегом ins, визуально отображается в виде подчёркнутого текста.

У элементов **del** и **ins** есть два возможных атрибута: **cite** и **time**. Как можно было бы подумать, в атрибут **cite** следует заключить текстовую цитату, однако вопреки предположению данный атрибут имеет формат URL, указывающий на документ, согласно которому данный фрагмент был вставлен или же наоборот потерял актуальность (соответственно для **ins** и **del**). Атрибут **time** имеет смысл времени, когда фрагмент потерял актуальность или же был добавлен.

```
<del cite="http://www.example.com/tra-la-la.html" datetime="2010-08-08T08:08:08Z">Этот фрагмент потерял актуальность</del>
```

Тег **small** по-прежнему заставляет браузеры отображать текст более мелким шрифтом, но его значение в html5 полностью изменилось. Теперь он означает текст, который не связан с содержанием страницы, и вместе с тем должен быть приведён в силу тех или иных обстоятельств. Например, в футере страницы для размещения информации о номере лицензии организации. Ещё один пример использования – предупреждение о том, что при тех или иных обстоятельствах сайт ответственности не несёт.

```
<small>Обращаем ваше внимание, что сайт не
несёт ответственности за высказывания авторов
публикаций!</small>
```

Элемент **address** согласно html5 теперь может использоваться несколько раз на странице. Если вы помните, тег address служит для указания контактного адреса автора (документа или статьи). В каждом блоке **article** допускается указать свойства **address**. При этом значение тега **address** относится только к текущему блоку **article** и не относится к вложенным **article**. Если вы хотите определить адрес во вложенном **article**, определите для него собственный **address**.

Интересная возможность для определения списков с помощью тега **ol**. Спецификация html5 позволяет указать стартовое значение индекса элементов списка.

```
<ol start="7">
    <li>И.А.Крылов</li>
    <li>М.Ю.Лермонтов</li>
    <li>А.С.Пушкин</li>
    <li>Н.В.Гоголь</li>
    <li>Н.А.Некрасов</li>
    <li>М.Е.Салтыков-Щедрин</li>
  </ol>
```

В браузере Opera такой фрагмент выглядит следующим образом:

7. И.А.Крылов
8. М.Ю.Лермонтов
9. А.С.Пушкин
10. Н.В.Гоголь
11. Н.А.Некрасов
12. М.Е.Салтыков-Щедрин

Второй интересный атрибут – **reversed**, позволяющий задать нумерацию списка в обратном порядке. В настоящее время работа атрибута reversed реализована во всех популярных браузерах.

Также у элементов списка **li** может использоваться атрибут **value**, позволяющий переопределить значение элемента.

Интересна и новая особенность всем знакомого тега ссылки (анкор). Если несколько элементов ссылались на один и тот же адрес, то каждый из них нужно было оформить в виде отдельного тега. Например, если одна ссылка должна представлять собой фрагмент текста, а другая рисунок, то раньше такую конструкцию можно было соорудить так:

```
<a  href=”http://adnotes.ru”>Читайте наш блог</a>
<a  href=”http://adnotes.ru”><img src=”image15.gif”></a>
```

Теперь с переходом на html5 внутрь тега ссылки может быть заключён более сложный комплексный элемент. Например, предыдущий пример в стиле html5 можно исполнить следующим образом:

```
<a href=”http://adnotes.ru”>Читайте наш блог<img src=”image15.gif”></a>
```

Необходимо заметить, что вложенные друг в друга теги ссылки ***не допускаются***.

Сейчас мы поговорим о переопределении заголовков и логической структуре документа.

Как мы помним, в html ещё с давних пор заголовки формируются с помощью специальных тегов: H1,H2,...,H6. Теги перечислены в порядке убывания их значения и уменьшения отображаемого шрифта по умолчанию. В заголовочных тегах вес определяется номером Hn и издавна рекомендовалось использовать на страницах нумерацию заголовочных тегов последовательно и без пропусков.

Рассмотрим, пример, демонстрирующий старую трактовку структуры документа.

```
<H1>Разметка HTML</H1>
<div>
  <H1>Структурные теги</H1>
</div>
```

Как видим, на странице размещены два заголовочных тега **H1**, один из которых вложен в семантически безликий элемент **DIV**. Если говорить о структуре, то вероятно с точки зрения привычных канонов выглядит следующим образом

```
1.Разметка HTML
2.Структурные теги
```

Таким образом, имеем список заголовков, состоящий из двух элементов, при этом оба элемента с точки зрения иерархии расположены на одинаковом уровне. Возможно, ранее об этом задумываться приходилось мало, поскольку основное внимание было приковано к визуальному отображению.

Теперь давайте посмотрим, что изменится, если семантически нейтральный **DIV** мы заменим на один из новых блокообразующих элемента, скажем, **article**. Создадим страницу с похожим на предыдущий фрагментом и посмотрим, что изменится с точки зрения структуры.

```
<H1>Разметка HTML</H1>
<article>
   <H1>Структурные теги</H1>
</article>
```

Что изменилось от того, что теперь вместо **div** в коде красуется **article**? Если судить по отображению страницы, то никаких изменений не наблюдается. Тогда в чём, собственно, смысл менять шило на мыло? А суть в том, что замена блока **div** на казалось бы эквивалентный **article** полностью изменило логическую структуру документа. Теперь она выглядит совершенно другим образом.

1.Разметка HTML
    1.Структурные теги

Точно так же список содержит два элемента, но теперь один из них вложен в другой. Совершенно поразительно! Если продолжить мысль, то с привычных позиций общепринятого понимания вложенный элемент списка даже больше напоминает нам элемент, выполненный в виде **H2**.

Более того, в данном случае используемый уровень заголовка не столь уж и важен, **H1** во вложенном **article** мы могли спокойно заменить заголовком с другим уровнем, скажем, **H5**, что совершенно не поменяло бы картину. По-прежнему структура выглядела бы аналогично: два заголовка, один из которых вложен в другой.

Элемент **article** можно было бы заменить на другой из новых блокообразующих, например, **section** или **nav**. Совершенно очевидно, что разнообразие новых блокообразующих элементов, о которых мы рассказывали выше, совместно с тегами заголовков способно создать структуру самой высокой сложности.

Более того, с точки зрения старых общепринятых представлений структура документа с использованием новых элементов может выглядеть просто парадоксально! Рассмотрим следующий пример.

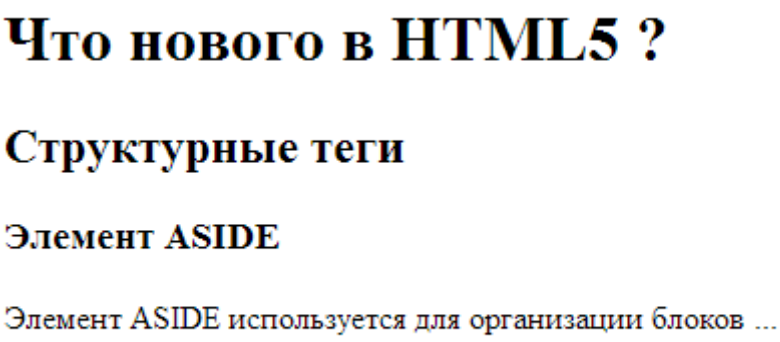
```
<H1>Что нового в HTML5 ?</H1>
   <section>
      <H2>Структурные теги</H2>
      <article>
         <H1>Элемент ASIDE</H1>
         Элемент ASIDE используется для организации блоков ...
      </article>
   </section>
```

Как будет выглядеть логическая структура данного документа? Как и предполагается, благодаря использованию section и article, а также в соответствии с вложенностью заголовков, структура будет следующей:

1.Что нового в HTML5 ?
    1.Структурные теги
        1.Элемент ASIDE

Налицо три разных уровня вложенности, и это несмотря на то, что два элемента выполнены в виде заголовка одинакового уровня **H1**. Но это одинаковый он чисто формально, на деле первый **H1** расположен в структуре выше.

Кстати сказать, даже без внесения изменений в таблицу стилей CSS, браузер уже выделяет заголовки разного значения шрифтами различного размера. Так, в браузере Chrome предыдущий пример отображается следующим образом:



Как видим, размер шрифта уменьшается в соответствии с уровнем вложенности в логической структуре документа, а не в зависимости от используемого уровня заголовочного тега. Так, заголовок в теге **H2** оказался крупнее, чем вложенный в него заголовок с использованием тега **H1**.

С тегами заголовков H1-H6 напрямую связан появившийся в html5 новый элемент – **HGROUP**. Давайте рассмотрим типичный пример, когда в шапке сайта размещён заголовок. Очень часто в заголовке наряду с H1 используется и H2, и это очень распространённый случай. Заголовок **H2**, как правило, выводится меньшим размером шрифта, но при этом является такой же составляющей частью заголовка, как и заголовок **H1**. Иными словами, общий заголовок представляет собой нечто целое и неразделимое, что нельзя разбить на две части. В таких случаях оба тега из заголовка можно объединить в один общий элемент **hgroup**.

```
<hgroup>
  <H1>Блокнот молодого админа</H1>
  <H2>Как сделать сайт?</H2>
</hgroup>
```

Отображаться такой комплексный заголовок будет традиционно, но с точки зрения логической структуры представляет собой единый элемент.

Интересно, что в схему документа будут включаться не оба заголовка (H1 и H2), а только первый. Общее правило таково, что в схеме отразится тот элемент, который имеет больший заголовочный уровень. Так, если бы в воображаемом общем заголовке **hgroup** были бы заключены три заголовка: H2, H4 и H5, то в общей схеме документа отразился бы лишь H2. Другое дело, что на практике такие расклады вряд ли распространены, обычно достаточно бывает ограничиться лишь H1 и H2. Кстати сказать, вспоминая общие принципы формирования структуры документа, теперь те же самые H1 и H2 можно свободно применять и для организации заголовков в основном содержании сайта.

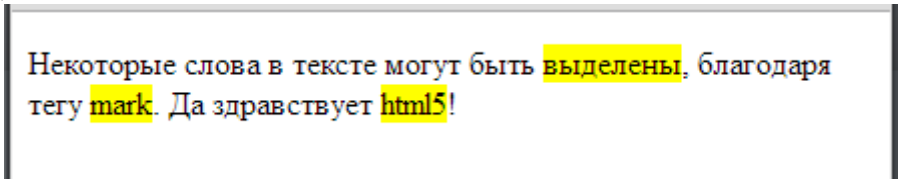
Новые тэги форматирования

Наконец переходим к рассмотрению новых элементов, появившихся в html5. Список новых тегов html5 не ограничивается лишь блокообразующими, о которых было сказано выше (nav, aside, section, footer и т.д.). В html5 достаточно новшеств и по части оформления текстовых фрагментов.

Новый тег **mark** выделяет текстовый фрагмент или отдельные слова. Тег **mark** не имеет атрибутов, он просто помечает текст путём его выделения.

```
Некоторые слова в тексте могут быть <mark>выделены</mark>,
благодаря тегу <mark>mark</mark>.
Да здравствует <mark>html5</mark>!
```

Выделение внешне происходит путём изменения фона, на который выводятся символы.



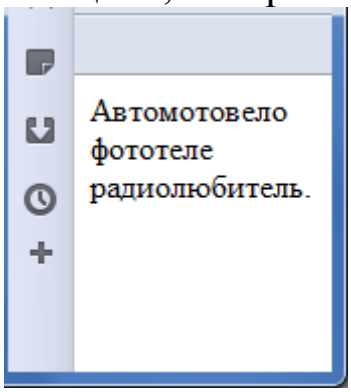
При этом выделенный с помощью **mark** фрагмент не получает дополнительного семантического значения, mark служит только для улучшения визуального восприятия.

Одно из полезных самых очевидных предназначений элемента разметки **mark** – его использование в результатах поиска. Благодаря mark искомые слова отлично выделяются визуально на фоне остального текста, что является несомненным плюсом.

Рассмотрим последний среди плеяды тестовых тегов, речь идёт об элементе **wbr**. Вероятно, вам знакома ситуация, когда в тексте встречается длинное слово, а размер контейнера не позволяет целиком отобразить его в одну строчку. В этом случае перенос может произойти в совершенно неожиданном для вас месте, которое вполне может не соответствовать вашим представлениям о правильности. Именно в этом случае и поможет новый тег **wbr**, результат работы которого удобнее продемонстрировать на примере.

```
Авто<wbr>мото<wbr>вело<wbr>фото<wbr>теле<wbr>радиолюбитель.
```

Может быть, слово не самое подходящее для примера и не самое длинное, но вполне позволяет проверить в работе и понять смысл тега **wbr**. Попробуйте уменьшать размеры окна до минимально возможного, и вы увидите, как трансформируется отображение путём переноса слова именно в тех позициях, которые мы пометили тегом.



Поиграйтесь с изменением размеров окна, только сильно не увлекайтесь, поскольку мы ещё рассказали не о всех возможностях html5.

От текстовых тегов плавно переходим к изображениям. Наверное, вряд ли кто-то поспорит, что тег размещения на странице изображений **img** по большей части вполне даже удовлетворяет основную массу потребностей. У тега имеются атрибуты **title** и **alt**, позволяющие сопровождать изображения на сайте всплывающей подсказкой и альтернативным текстом в случае, когда отображение картинок либо по какой-то причине отключено, либо картинка не может загрузиться. Таким образом, даже если изображение не выводится на страницу, его замещает надпись из атрибута **alt**, а всплывающая подсказка из атрибута **title** позволяет отображать дополнительную информацию.

Одна из распространённых конфигураций – размещение на странице изображения одновременно с текстовой информацией под ним. Это может быть блок-схема с поясняющей надписью, либо чертёж, либо изображение репродукции картины с наименованием и фамилией автора и т.д. Всё это можно реализовать при помощи тега img и CSS, что собственно до и происходило до сих пор.

Появление нового тега **figure** в html5 позволяет ещё больше упростить реализацию размещения изображений на странице. Тег **figure** используется не вместо img, а совместно с ним. Кроме того, для организации подписей к изображениям применяется тег **figcaption**.

Пример кода

```
<figure>
    <img src="images/html5wow.png" alt="html5 - это супер!">
    <figcaption>
      <small>Обязательно изучите, что нового появилось в html5!</small>
    </figcaption>
</figure>
```

Данный фрагмент кода обеспечивает вывод на страницу, как самого изображения, так и подписи под ним. Тег **figure** как бы заключает в себя все детали, относящиеся к изображению. К таким деталям относятся вложенный img, определяющий непосредственно изображение, и **figcaption**, формирующий подпись.

Обязательно изучите, что нового появилось в html5!

Норвые атрибуты

От рассмотрения новых тегов html5 перемещаемся к новым атрибутам. Одним из самых неоднозначных является атрибут **hidden**. Он позволяет разместить на странице элемент, но при этом не заставляет браузер не отображать его. Иными словами, элемент на странице становится скрытым, что и объясняет происхождение наименования тега.

```
<p hidden>Этот текст браузер вообще не отобразит,
хотя в коде страницы он будет присутствовать.</p>
```

Такой фрагмент не будет визуально отображён на странице. Сразу стоит сделать замечание, что поскольку в коде страницы элемент с атрибутом hidden всё равно присутствует, то не следует использовать его для размещения конфиденциальных данных. Стоит только нажать в браузере комбинацию Ctrl+U, и все ваши секреты тут же станут общеизвестными.

Вообще споры по поводу использования атрибута дополнительно подогреваются в связи с наличием в CSS стилей

```
display: none;
visibility: hidden;
```

Но в отличие от стилей CSS атрибут html5 **hidden** придаёт фрагменту ещё и определённое семантическое значение. Если говорить коротко, то общие рекомендации по использованию данного атрибута сводятся к тому, что там, где решаются чисто задачи отображения, применяем CSS, где важна семантика – пользуемся **hidden**.

Атрибут **draggable** придаёт элементам свойство быть перетаскиваемым. Ещё один интересный атрибут html5 – **contenteditable** . Он заставляет элемент вести себя иначе и быть редактируемым со стороны пользователя.

```
<p contenteditable>А вы видели когда-нибудь редактируемый абзац?</p>
```

Атрибут может принимать любое из двух значений: **true** или **false**. По умолчанию используется значение true, т.е. если вы определите элемент просто с атрибутом **contenteditable**, это будет то же самое, что и с явно заданным значением true. Для того, чтобы элемент не был редактируемым, либо нужно совсем не упоминать атрибут **contenteditable**, либо указать его значение в виде false.

```
<p contenteditable="false">Этот текст нельзя редактировать </p>
```

HTML 5 вводит несколько новых атрибутов для элементов, которые уже входили в HTML 4:

- элементы **a** и **area** получили новый признак ping, который определяет список URI адресов, которые должны пропинговаться при переходе по гиперссылке. Принцип функционирования пока до конца не ясен.
- элемент **area** теперь имеет атрибуты hreflang и rel
- **base** получил атрибут target
- атрибут **value** для **li** и атрибут **start** для элемента **ol** больше не deprecated
- **meta** получил атрибут **charset**

- новый атрибут **autofocus** может быть определен у **input** (кроме тех случаев, когда **type** атрибут - **hidden**), **select**, **textarea** и **button**. Это обеспечивает способ передачи управления форме во время загрузки страницы
- атрибут **form** для **input**, **output**, **select**, **textarea**, **button** и **fieldset** позволяет связать элемент с более чем одной формой
- **input**, **button** и **form** получили атрибут **replace**, который определяет, что будет с элементом после отправки формы
- **form**, **select** и **datalist** имеют атрибут **data**, который учитывает автоматическое предзаполнение, в случае заполнения данными с сервера
- новый атрибут **required** применяется к **input** (кроме тех случаев, когда **type** атрибут - **hidden**, **image** или кнопка) и **textarea**. Он указывает обязательные для заполнения поля
- **input** и **textarea** имеют новый атрибут **inputmode**, который дает подсказку пользовательскому интерфейсу относительно того, какие данные ожидаются для ввода
- теперь можно **disable** (отключить) сразу целый **fieldset**, что не было возможно прежде
- элемент **input** имеет несколько новых атрибутов для определения ограничений: **autocomplete**, **min**, **max**, **pattern** и **step**, а также **list**, который может использоваться вместе с элементами **select** и **datalist**
- **input** и **button** также получили новый атрибут **template**, который может использоваться для шаблонов повторения
- элемент **menu** имеет три новых атрибута: **type**, **label** и **autosubmit**
- **script** имеет новый атрибут **async**, который влияет на загрузку и выполнение сценария
- элемент **html** имеет новый атрибут **manifest**, который указывает на кэш приложений, используемый вместе с API для автономных Web приложений

Несколько атрибутов из HTML 4 применяют ко всем элементам, поэтому их называют глобальными атрибутами: **class**, **dir**, **id**, **lang**, **tabindex** и **title**.

Появились также несколько новых глобальных атрибутов:

- атрибут **contenteditable** указывает, что элемент доступен для редактирования
- **contextmenu** может использоваться для указания на контекстное меню, созданное автором
- **draggable** может использоваться вместе с новым drag&drop API
- **irrelevant** указывает, что элемент еще или больше не актуален

Атрибуты для модели повторения (repetition model):

- **repeat**
- **repeat-start**

- **repeat-min**
- **repeat-max**

Отмененные элементы

Следующие элементы не включены в HTML5, потому что их эффект достигается использованием CSS:

- **basefont**
- **big**
- **center**
- **font**
- **s**
- **strike**
- **tt**
- **u**

Следующие элементы не включены в HTML5, потому что их использование негативно сказывалось на удобстве и доступности:

- **frame**
- **frameset**
- **noframes**

Следующие элементы не включены, потому что использовались редко или они могут быть заменены другими элементами:

- **acronym**
- **applet** замещен **object**
- **isindex**
- **dir** замещен **ul**

Наконец **noscript** остался только в синтаксисе HTML, поскольку его использование предполагает разбор с помощью HTML парсера.

Отмененные атрибуты

- **accesskey** для **a**, **area**, **button**, **input**, **label**, **legend** и **textarea**
- **rev** и **charset** для **link** и **a**
- **shape** и **coords** для **a**
- **longdesc** для **img** и **iframe**
- **target** для **link**
- **nohref** для **area**
- **profile** для **head**
- **version** для **map**, **img**, **object**, **form**, **iframe**, **a**
- **scheme** для **meta**
- **archive**, **classid**, **codebase**, **codetype**, **declare** и **standby** для **object**
- **valuetype** и **type** для **param**
- **charset** и **language** для **script**
- **summary** для **table**
- **headers**, **axis** и **abbr** для **td** и **th**
- **scope** для **td**

Кроме того, HTML5 не имеет следующих атрибутов, поскольку они лучше обрабатываются CSS:

- **align** для **caption**, **iframe**, **img**, **input**, **object**, **legend**, **table**, **hr**, **div**, **h1-h6**, **p**, **col**, **colgroup**, **tbody**, **td**, **tfoot**, **th**, **thead**, **tr** и **body**
- **alink**, **link**, **text** и **vlink** для **body**
- **background** для **body**
- **bgcolor** для **table**, **tr**, **td**, **th** и **body**
- **border** для **table**, **img** и **object**
- **cellpadding** и **cellspacing** для **table**
- **char** и **charoff** для **col**, **colgroup**, **tbody**, **td**, **tfoot**, **th**, **thead** и **tr**
- **clear** для **br**
- **compact** для **menu**, **ol** и **ul**
- **frame** на **table**
- **frameborder** приписывают на **iframe**
- **height** для **iframe**, **td** и **th**
- **hspace** и **vspace** для **img** и **object**
- **marginheight**, **marginwidth** и **scrolling** для **iframe**
- **noshade** для **hr**
- **nowrap** для **td** и **th**
- **rules** для **table**
- **size** для **hr**, **input** и **select**
- **style** для всех элементов
- **type** для **li**, **ol** и **ul**
- **valign** для **col**, **colgroup**, **tbody**, **td**, **tfoot**, **th**, **thead** и **tr**
- **width** для **hr**, **table**, **td**, **th**, **col**, **colgroup**, **iframe** и **pre**

k) HTML 5: отличия от HTML 4

- **figure** может использоваться для связи заголовка с медиа контентом:

<figure>
<video src=ogg>…</video>
<legend>Example</legend>
</figure>

- **audio** и **video** для мультимедиа.

Таким образом разработчики могут писать скрипты собственного пользовательского интерфейса, но также предусмотрен способ вызова стандартного API пользовательского агента. Вместе с этими элементами может быть использован **source**, если есть возможность организовать параллельные потоки.

- **embed** используется для контента plugin'ов.
- **meter** - для представления единиц измерений.
- **time** - дата и/или время.

- **canvas** используется для динамической отрисовки графики.
- **command** представляет команду, которую может вызвать пользователь.
- **datagrid** - интерактивное представление списка типа "дерево" или табличных данных.
- **details** представляет дополнительную информацию, которую пользователь может получить по требованию.
- **datalist** вместе с новым атрибутом **list** используется чтобы сделать combobox:

```
<input list=browsers>
<datalist id=browsers>
<option value="Safari">
<option value="Internet Explorer">
<option value="Opera">
<option value="Firefox">
</datalist>
```

- **datatemplate**, **rule**, и **nest** обеспечивают механизм шаблонов (templating mechanism) для HTML.
- **event-source** используется для перехвата событий, посланных сервером.
- **output** представляет определенный тип вывода, например, от вычислений, сделанных через скрипт.
- **progress** представляет ход выполнения задачи, например, загрузки.

API

HTML 5 вводит множество API, которые должны помочь в создании Web приложений. Они могут использоваться вместе с новыми элементами.

- 2D drawing API, который может использоваться с новым элементом **canvas**
- API для проигрывания видео и аудио, который может использоваться с новыми элементами **video** и **audio**
- выделенная область памяти (Persistent storage) с поддержкой данных в виде ключ / значение и SQL данных
- API, который допускает автономную работу web приложений
- API, который позволяет web приложений регистрировать себя для определенных протоколов или типов MIME
- Editing API в сочетании с новым глобальным атрибутом contenteditable
- Drag&drop API в сочетании с атрибутом **draggable**
- Network API

- API, который выстраивает историю посещения, чтобы предотвратить нарушение функционирования back кнопки (Этот API имеет необходимые ограничения безопасности)
- Cross-document messaging (Передача сообщений между документами)
- события сервера (Server-sent events) в сочетании с новым элементом **event-source**

WebGL

Библиотека Web-based Graphics Library, или WebGL, расширяет возможности JavaScript по созданию интерактивной трехмерной графики в браузере. WebGL реализована как контекст HTML-элемента Canvas. 3 марта 2011 года спецификация достигла версии 1.0. Разработку ведет некоммерческая организация Khronos Group.

Геолокация

API Geolocation представляет собой стандартный интерфейс для получения данных о географическом местоположении устройства. Он предоставляет доступ к объекту window.geolocation, который, в свою очередь, обладает методами, которые помогают определить местоположение устройства на основании информации со специальных серверов. Информация о местоположении собирается из множества источников, включая характеристики IP-адреса, встроенный приемник GPS, MAC-адреса Wi-Fi и Bluetooth, радиочастотные метки RFID, и место подключения к Wi-Fi.

SVG

Пожалуй, самая странная технология, попавшая под бренд HTML5, — это стандарт векторной графики SVG (Scalable Vector Graphics). SVG представляет собой синтаксис векторной графики на базе XML. Спецификация SVG разрабатывается W3C с 1999 года, поэтому включение этой технологии как «новой» составляющей HTML5 является большим лукавством.

Тем временем новый всплеск внимания к SVG вполне оправдан, так как в настоящее время существует реальная тенденция к внедрению этого стандарта. Определенный уровень поддержки уже доступен в новейших версиях всех основных браузеров, а API для более старых версий Internet Explorer реализуется через библиотеки, подобные Raphael.js.

Контекст Canvas 2D

Одной из первых звезд эры HTML5 стал элемент Canvas и связанная с ним библиотека API, которые начали свое существование в качестве расширения Apple для HTML. Как уже отмечалось ранее, эта технология начала свою жизнь в рамках спецификации HTML5. Теперь же она разрабатывается W3C в отдельном документе.

Canvas 2D предоставляет программируемый интерфейс для рисования двумерных изображений и картинок прямо в браузере. Этот элемент уже применяется для широкого круга задач — от рисования карт до игр, от популярной системы поддержки пользовательских шрифтов до переноса языка программирования Processing на JavaScript. К радости тех, кто хочет использовать Canvas сегодня, этот элемент до некоторой степени поддерживается и в старых версиях Internet Explorer благодаря библиотеке ExplorerCanvas.

Web Storage

Спецификация Web Storage определяет API для постоянного хранения данных в веб-браузере в виде пар ключ/значение. Эта спецификация похожа на cookies, но значительно превосходит ее по возможностям.

Хранение данных происходит в двух видах: sessionStorage и localStorage. Оба они предоставляют одинаковые методы для управления элементами (setItem(), removeItem() и getItem()), а также clear() для очистки всего хранилища. Вариант SessionStorage предназначен для хранения информации, которая нужна только во время текущей сессии браузера. Вариант LocalStorage предназначен для длительного хранения настроек сайта или других данных пользователя. Существует также событие storage, которое можно использовать для мониторинга и реагирования на действия с хранилищем.

Тем, кто хочет использовать упомянутые функции уже сейчас, библиотека Persist.js предоставляет надежное решение для различных платформ, позволяющее использовать Web Storage или ее эквиваленты во всех популярных браузерах.

Еще две спецификации идут рука об руку с Web Storage:

- **IndexedDB**— это связанная, но еще незрелая спецификация, которая предоставляет еще больше возможностей для продолжительного хранения данных в браузере, включая возможность делать запросы к базе данных, а также, что важно, хранить сложные объекты, а не только простые строки.
- **Web Workers**— это удивительно мощная спецификация, определяющая API, в котором предусмотрена возможность запуска фоновых «исполнителей», выполняющих код параллельно с машиной JavaScript на главной странице. Эта функциональность позволяет разработчикам выводить ресурсоемкие задания обработки данных в фоновые процессы, освободив браузер для дальнейшего взаимодействия с пользователем в основном контексте.

File API

Эта спецификация предоставляет API для работы с файлами прямо в веб-приложениях. Вместе с несколькими развивающимися и уже

принятыми технологиями, такими как XMLHttpRequest, drag-and-drop и Web Workers, библиотека File API позволит производить значительно более сложные взаимодействия между веб-ресурсом и рабочим столом пользователя, чем это возможно сегодня. Вместо простого элемента загрузки файла, который передает файл на веб-сервер для обработки, или сложного интерфейса на базе Flash, библиотека File API позволит напрямую обращаться к содержимому файла в браузере.

WebSocket

WebSocket API разработан для одновременного двустороннего взаимодействия между браузером и сервером через один сокет TCP. На самом деле внедрение WebSocket должно было продвинуться значительно дальше благодаря поддержке в последних версиях Firefox, Opera, Chrome и Apple Safari, однако обнаруженная уязвимость в системе безопасности привела к тому, что в настоящее время поддержка этой технологии в Firefox и Opera по умолчанию отключена.

Server-sent Events

Эта спецификация определяет API для создания HTTP-соединения с целью получения push-уведомлений в форме событий DOM. Эта спецификация заменяет собой текущий формат регулярного опроса серверов с целью получения обновлений, исключая необходимость в бесчисленном количестве ненужных запросов, а также связанных с ними нагрузок на процессор и канал передачи данных.

XMLHttpRequest Level 2

Спецификация XMLHttpRequest Level 2 вносит улучшения в объект XMLHttpRequest, расширяя его возможности. Наиболее интересным из новшеств, возможно, является функция Cross-Origin Resource Sharing, которая предоставляет безопасный способ обхода политики единого источника запросов, позволяя XMLHttpRequest взаимодействовать со сторонним сервером. В настоящее время XMLHttpRequest может контактировать в рамках одного протокола только с одним сервером.

Учитывая повсеместное распространение JSON и JSONP, Cross-Origin Sharing может уже быть не так актуальна, как ранее, но она все же открывает множество путей для развития архитектуры сайтов и гибридных приложений, снимая ограничения политики единого источника.

Расширение HTMLDocument

HTML 5 расширил интерфейс HTMLDocument. Интерфейс теперь реализован на всех объектах интерфейса Document. Его новые методы:

- **getElementsByClassName()**
- **activeElement** и **hasFocus**
- **getSelection()**
- **designMode** и **execCommand()**, которые используются главным образом для редактирования документов

Расширение к HTMLElement

Интерфейс HTMLElement также получил несколько расширений:

- **getElementsByClassName()**
- **innerHTML**
- **classList** введен для удобства доступа к **className**. Возвращаемый объект имеет методы **has()**, **add()**, **remove()** и **toggle()** для манипуляции классами элемента

Формы

Формы являются наиболее популярным средством для интерактивного взаимодействия с пользователем. Они могут содержать такие элементы управления как кнопки, переключатели, поля для ввода текста и т. д. Эти элементы представляют возможность посетителю сайта ответить на предлагаемые вопросы и отправить результаты ответов на сервер или по электронной почте.

1. Описание формы

Для описания формы служит тег **<FORM>…</FORM>**, который является контейнером для элементов управления. Он определяет:

- Макет формы (задается содержимым тега).
- Программу, которая будет обрабатывать заполненную и переданную форму.
- Метод отправки данных на сервер.

Синтаксис этого тега:

<FORM ACTION= “Обработчик форм” METHOD= “Метод”>
…содержимое формы…
</FORM>

Атрибут **ACTION** задает агента для обработки формы. Например, значением может быть **mailto: адрес** (для отправки формы по электронной почте) или адрес обрабатывающей программы (для передачи формы в программу).

Атрибут **METHOD** определяет метод HTTP, используемый для передачи данных формы. Возможные значения (обязательно в нижнем регистре) – **“post”** (для передачи по почте) и **“get”** (для передачи программе обработки).

Например, если форма будет отправляться по электронной почте, необходимо указать:

<FORM ACTION= “mailto:znanie_kov@mail.ru” METHOD= “post”>

2 Однострочные текстовые поля

Содержимое формы задаётся с помощью тегов управляющих элементов. Наиболее часто используется для этого тег **<INPUT>**, атрибут **TYPE** которого определяет тип создаваемого элемента:

<INPUT TYPE= тип элемента SIZE= размер VALUE= значение NAME= имя элемента>

Для создания поля ввода под одну строку текста применяется атрибут **TYPE= “TEXT”.**

Размер текстового поля можно устанавливать посредством атрибута **SIZE** (в символах).

Необязательный атрибут **VALUE** задаёт строку, которая будет выведена в поле браузером.

Атрибут **NAME** используется для присвоения данному полю имени, которое может применяться при отправке формы или доступа к полю из программ и таблиц стилей.

<INPUT TYPE= “TEXT” SIZE= 40 VALUE= “Текст…” NAME= Имя>

Можно создать также текстовое поле пароля, для этого используется атрибут **TYPE= “PASSWORD”**.

В данном случае вводимые символы отображаются на экране в виде условных значков (звездочек), таких как * или ·. Делается это для того, чтобы скрыть текст от любопытных глаз при вводе паролей с клавиатуры.

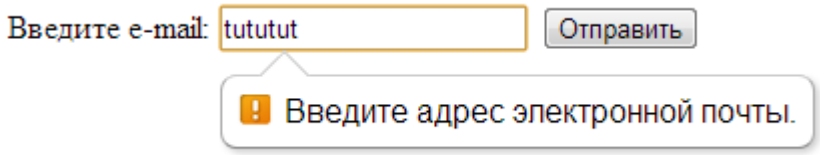
<INPUT TYPE= PASSWORD SIZE=12 NAME= Пароль>

В **HTML5** подавляющее число элементов создаётся при помощи элемента **INPUT** за счёт введения новых значений **TYPE**.

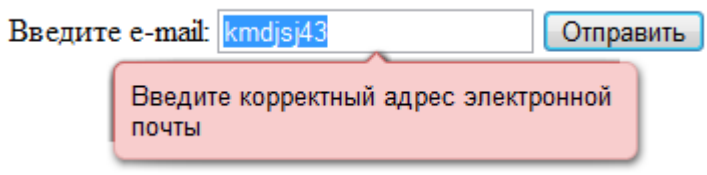
Для ввода адреса электронной почты **TYPE= “EMAIL”**

```
<form>
   Введите e-mail: <input type="email" name="email">
   <input type="Submit">
</form>
```

В случае отправки формы с не соответствующими формату данными браузер выведет предупреждение.



В данном случае так отреагирует популярный браузер Chrome. У Opera сообщение об ошибке будет выглядеть несколько иначе.

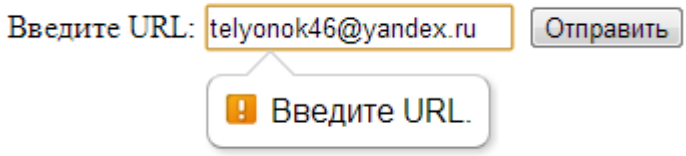


Отображение элементов форм производит браузер, поэтому визуально один и тот же элемент формы может отображаться в зависимости от браузера по-разному. Аналогично для сообщений об ошибке каждый

браузер выбирает свой стиль. Спишем это на молодость html5 и надеемся ситуация в скором будущем улучшится.

Для получения от пользователя адреса URL можно использовать одноимённый тип поля ввода **TYPE= “URL”**

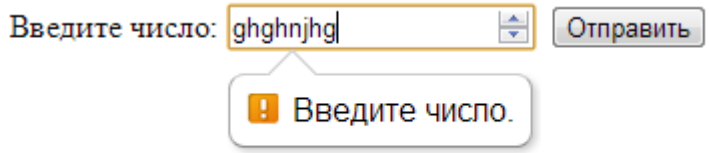
```
<form>
   Введите URL: <input type="url" name="adresurl">
   <input type="Submit">
</form>
```



Новый тип **number** позволяет производить проверку на предмет соответствия введённого значения числу.

```
<form>
   Введите число: <input type="number" name="num">
   <input type="Submit">
</form>
```

Как видим, у нас не только происходит проверка, но и появились дополнительные элементы для увеличения/уменьшения значения с помощью мыши.



Это скрин с **Chrome**, а вот то же самое для **Opera**.

Убеждаемся, что на текущий момент браузер **Opera** корректно отображает такое поле, однако должную проверку не производит.

3 Флажки и переключатели

Флажки - это переключатели типа вкл./выкл., которые могут переключатся пользователем. Несколько флажков в форме могут иметь одно и то же имя управляющего элемента. Таким образом, например, флажки позволяют пользователям выбрать несколько значений для одного и того же свойства. Для создания флажков используется тег **INPUT** с атрибутом **TYPE=”CHECKBOX”:**

<INPUT TYPE=”CHECKBOX” NAME=”Флаг 1” VALUE=”Школа”>Школа
<INPUT TYPE=”CHECKBOX” NAME=”Флаг 1” VALUE=”Техникум”> Техникум
<INPUT TYPE=”CHECKBOX” NAME=” Флаг 1” VALUE=”Институт”> Институт
<INPUT TYPE=”CHECKBOX” NAME=”Флаг 1” VALUE=”Аспирантура”> Аспирантура

Переключатели похожи на флажки за исключением того, что, если несколько переключателей используют одно и то же имя управляющего элемента, они являются взаимоисключающими: если один переключатель включен, другие обязательно выключены.

Для создания переключателей используются тег **INPUT** с атрибутом **TYPE=”RADIO”:**

<INPUT TYPE=”RADIO” NAME=”Пол” VALUE=”Мужчина” CHECKED>Мужчина
<INPUT TYPE=”RADIO” NAME=”Пол” VALUE=”Женщина”> Женщина

Атрибут CHECKED позволяет заранее ”включать” тот флажок или переключать, в теге которого он расположен.

4 Кнопки

В форме могут использоваться три типа кнопок:

- кнопки отправки - при активизации такой кнопки производится отправка формы;
- кнопки сброса- при активизации такой кнопки для всех управляющих элементов устанавливаются исходные значения;
- прочие кнопки - для таких кнопок действие по умолчанию не определено.

Кнопка отправки позволяет отправить данные, введенные в форму, на соответствующий сервер (по адресу, указанному в атрибуте **ACTION** тега **<FORM>).** Надпись на кнопке задается с помощью атрибута **VALUE**. Для создания кнопки отправки используется значение **SUBMIT** атрибута **TYPE**:
<INPUT TYPE=SUBMIT VALUE=”Отправить”>

Кнопка сброса позволяет вернуть все поля формы в исходное состояние, когда документ еще был только загружен в браузер. Вы можете выбрать для этой кнопки ту или иную надпись, воспользовавшись атрибутом **VALUE**. Кнопка сброса создается с использованием значение **TYPE=RESET**:
<INPUT TYPE=RESET VALUE=”Очистить”>

Авторы могут создавать прочие кнопки с помощью контейнера **<BUTTON>…</BUTTON>**, который позволяет встраивать в кнопку картинки, таблицы и т.д.

<BUTTON NAME=Имя TYPE=Тип>
…вложенные теги…
</BUTTON>

Атрибут **TYPE** может принимать значения SUBMIT или **RESET** при создании кнопок отправки или очистки.

Параметры для различных видов кнопок:
submit - кнопка отправки содержимого формы web-серверу. Ее параметры:

- type="submit" - тип кнопки,
- name - имя кнопки,
- value - надпись на кнопке.

image - графическая кнопка отправки содержимого формы web-серверу. Для ее использования необходимо подготовить картинку кнопки, а потом использовать ее в виде кнопки. Ее параметры:

- type="image" - тип графической кнопки,
- name - имя кнопки,
- src - адрес картинки для кнопки.

reset - кнопка, позволяющая восстановить все значения по умолчанию в форме. Ее параметры:

- type="reset" - тип кнопки очищения,
- name - имя кнопки,
- value - надпись на кнопке.

button - произвольная кнопка, ее действия назначаются вами, т.е. сама она делать ничего не умеет. Ее параметры:

- type="button" - тип произвольной кнопки,
- name - имя кнопки,
- value - надпись на кнопке.
- onclick - указывает, что делать при щелчке по кнопке.

Вообще, у этого типа кнопок есть и другие события (например, двойной щелчок), но здесь мы не будем их рассматривать.
Если на форме несколько кнопок, то они должны иметь разные названия.
Пример кода:

```
<form name="forma1">
  <input type="submit" name="submit" value="Отправить">
  <input type="image" name="but_img" src="but.gif">
  <input type="reset" name="reset" value="Очистить">
  <input type="button" name="button" value="Отправить">
</form>
```

Результат:
Начало формы

Конец формы

Кнопки можно задавать и по другому, при помощи тегов **<button> </button>.** Возможности у таких кнопок несколько шире, они могут иметь содержимое в виде текста или картинки. Этот тег имеет следующие параметры:

- type - тип кнопки, может принимать значения:
- reset - кнопка очистки формы,
- submit - кнопка отправки данных,
- button - кнопка произвольного действия.
- name - имя кнопки,
- value - надпись на кнопке.

Пример кода:

```
<form name="forma1">
  <button name="submit" type="submit">
   <img src="mail.gif" align="absmiddle">
   <font size="4"> Отправить </font>
  </button>
</form>
```

Результат:
Начало формы

 Отправить

5 Многострочные поля и меню

Управляющий элемент для многострочного ввода текста создается тегом **<TEXTAREA>…</TEXTAREA>.** В этом теге можно заранее задать текст, который будет выведен, как исходное значение управляющего элемента, причем текст выводится с учётом пробелов, табуляторов и символа конца строки.

<TEXTAREA NAME= address ROWS=3 COLS=20>
Текст…
</TEXTAREA>

ROWS – количество строк текста, видимых на экране
COLS – ширина создаваемого текстового поля в символах
NAME - имя поля,

WRAP - способ переноса слов:
off - переноса не происходит,
virtual - перенос отображается, но на сервер поступает неделимая строка,
physical - перенос и на экране и при поступлении на сервер.
DISABLED - неактивное поле,
READONLY - разрешено только чтение.

Открывающееся меню типа **«выбор одного пункта из многих»** создаётся тегом **<SELECT>…</SELECT>**. Тег SELECT должен включать в себя один или несколько тегов **<OPTION>**, описывающих отдельные пункты меню. Использование в теге **<OPTION>** атрибута **SELECTED**, позволяет заранее «выбрать» пункт меню.

```
<SELECT NAME= «Программы»>
<OPTION> Excel
<OPTION>Word
<OPTION SELECTED> Power Point
<OPTION> Access
</SELECT>
```

Многострочное меню типа «несколько пунктов из многих» создаётся аналогично, разница состоит в том, что в теге **<SELECT>** добавляются атрибуты **MULTIPLE** и **SIZE**:

```
<SELECT NAME= «Программы» SIZE=4 MULTIPLE>
```

SIZE - количество строк меню, видимых на экране.

HTML5-новые элементы

Атрибут **type** элемента **input** теперь имеет следующие новые значения:

- **datetime**
- **datetime-local**
- **date**
- **month**
- **week**
- **time**
- **number**
- **range**
- **email**
- **url**

Идея относительно этих новых типов состоит в том, что пользовательский агент может обеспечить интерфейс для таких объектов как календарь (выбор даты), интеграции с адресной книгой и предоставить серверу данные в определенном формате. Это дает определенные преимущества как пользователям, так и разработчикам, поскольку пользовательский ввод проверяется перед посылкой на сервер браузером. Это означает, что разработчикам нет необходимости расходовать ресурсы

на проверку введенных данных, что, в свою очередь, приводит к сокращению времени ожидания ответа.
Среди новых тегов особое внимание заслуживает **time**. Это совершенно новый тип элемента, ранее не имеющий аналога.

```
<time datetime="2012-12-21">21 декабря 2012 года</time>
```

Согласитесь, несколько необычно выглядит дублирование даты как внутри тега, так и в его атрибуте. Дело в том, что элемент **time** призван решать сразу две задачи одновременно: визуальное отображение даты и передачу значение даты, которая будет понятна машинам. Первое будет производиться над текстом, заключённым в тег **time**, а второе определяться путём задания атрибута **datetime**. В качестве атрибута можно передавать не только значение даты, но и даты и времени. При этом время должно указываться в 24-часовом формате. Значение даты задаётся в формате **YYYY-MM-DD**, для времени используется формат **hh:mm:ss**. Значение секунд является необязательным и его можно опустить. Если в значении указываются одновременно и дата, и время, то они отделяются друг от друга заглавным символом **T**, т.е. «Time».

В общем случае формат атрибута, следующий:

YYYY-MM-DDT hh:mm:ssTZ

Как видите, помимо даты и времени у элемента time есть место для указания временной зоны (Time Zone).
Пример использования тега **time**:

```
<time datetime =2013-06-06>Дата  матча</time>
<time datetime =2013-06-06T18:00> Время начала матча </time>
<time datetime =2013-06-06T18:00+03:00> Время начала матча </time>
<time datetime =18:00> Время начала матча </time>
<time datetime =18:00+03:00> Время начала матча </time>
```

У элемента **time** существует весьма полезный атрибут – **pubdate**, указав который можно превратить значение **time** в значение даты публикации материала.

Пример:

```
Статья опубликована  <time datetime =2011-07-07 pubdate> 7 июля 2011 года
</time>
```

Атрибут **pubdate** даст понять, что именно эти дата или время являются временем публикации материала.

Дата и время задается в международном формате ISO 8601. Примеры оформления приведены в табл. 2.
Для каждой единицы существует своя заданная форма и ограничения.
Год — задается четырьмя цифрами (1860).
Месяц — две цифры (01 — январь, 02 — февраль, 12 — декабрь).
День — две цифры от 01 до 31.
Час — две цифры от 00 до 23.
Минуты — две цифры от 00 до 59.
Секунды — две цифры от 00 до 59.
Часовой пояс — часы и минуты с указанием знака плюс или минус.
Дата и время разделяются между собой заглавной латинской буквой T. Часовой пояс при необходимости пишется после времени со знаком плюс или минус.
К примеру, для Москвы часовой пояс будет +03:00.

Табл. 2. Форматы даты и времени

Значение	Формат	Пример
Год	ГГГГ	2012
Месяц и год	ГГГГ-ММ	2012-12
Полная дата	ГГГГ-ММ-ДД	2012-12-23
Дата и время с минутами	ГГГГ-ММ-ДДТчч:мм	2004-07-24T18:18
Дата и время с секундами	ГГГГ-ММ-ДДТчч:мм:сс	2004-07-24T18:18:18
Дата и время с часовым поясом	ГГГГ-ММ-ДДТчч:мм:сс±чч:мм	2004-07-24T18:18:18+04:00

Элементы управления

range

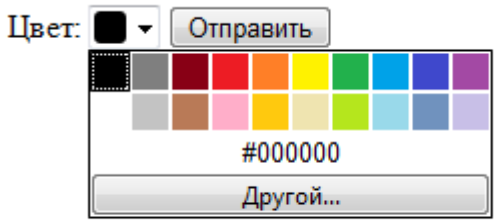
Тип **range** позволяет создать элемент управления в виде ползунка, с помощью которого пользователь будет изменять значение.

```
<form>
   Range: <input type="range" name="range">
   <input type="Submit">
</form>
```

Color

Теперь для выбора цвета не обязательно подключать дополнительные библиотеки в виде сторонних JavaScript, достаточно определить элемент **input** с типом **color**.

```
<form>
   Цвет: <input type="color" name="color">
   <input type="Submit">
</form>
```

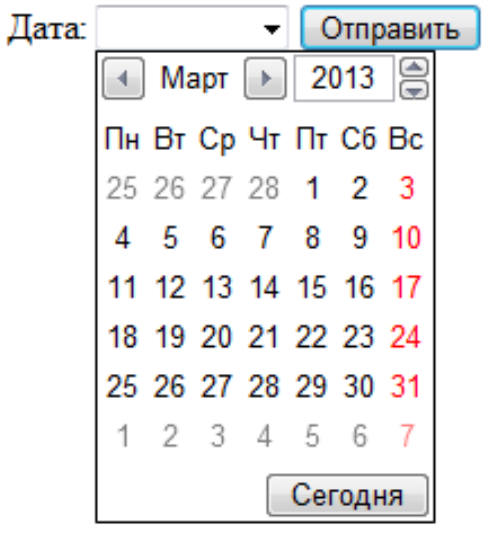


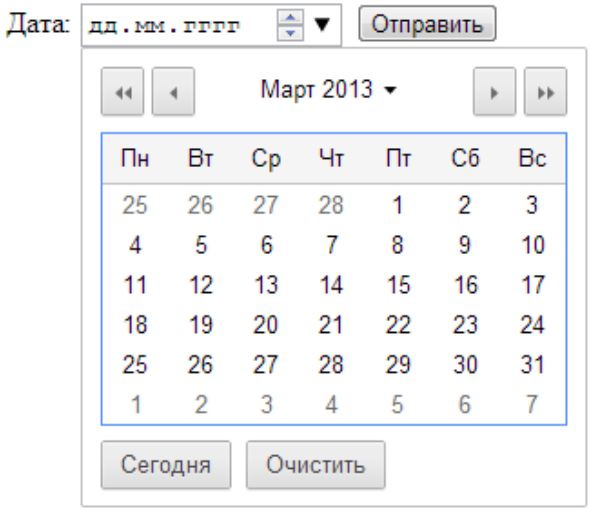
Date

Календарь теперь встроен в html5.

```
<form>
   Дата: <input type="date" name="date">
   <input type="Submit">
</form>
```

Реализации встроенного календаря у браузеров различаются.
Вот так выглядит поле ввода даты с календарём в **Opera**. А рядом вариант календаря с полем ввода одного из хромоподобных браузеров.





Tel

Тип **tel** был задуман как тип для ввода в форму номеров телефонов, однако на текущий момент вряд ли будет пользоваться популярностью. Формат номеров телефонов в общем случае разный, да и необходимой проверки, о которой мечтал пользователь, на сегодняшний день не осуществляется.

Search

Тип поля **search** предназначен для полей поиска. Несмотря на то, что в браузерах отображение поля с типом **Search** очень напоминает **Text**, в полях для организации строки поиска рекомендуется использовать именно **Search**.

Замечательной практической возможностью форм в **html5** является новый атрибут **placeholder**. С помощью него в поле ввода формы можно задать подсказку. Когда такое поле не заполнено, в нём отображается подсказка. Как только поле активизировано, или в нём введено какое-то значение, подсказка исчезает. Если значение поля вновь сделать пустым, то подсказка восстановится.

```
<form>
   Мы ищем: <input type="search" value="" required placeholder="Введите строку поиска">
   <input type="Submit">
</form>
```

Мы ищем: Введите строку поиска Отправить

Следующий на очереди у нас атрибут **autofocus**. Если элемент формы снабжён данным атрибутом, то после загрузки страницы именно этот элемент становится активным (получает фокус) Это очень удобно, если порядок расположения элементов формы не позволяет сделать нужный элемент активным. Раньше для этого приходилось применять дополнительные скрипты, теперь **autofocus** выделит нужное поле автоматом.

```
<form>
   Поле1: <input value="" placeholder="Поле без автофокуса">
   Поле2: <input value="" autofocus placeholder="Поле с автофокусом">
   Поле3: <input value="" placeholder="Поле без автофокуса">
   Поле4: <input value="" placeholder="Поле без автофокуса">
   <input type="Submit">
</form>
```

Как вы уже догадались, после рендеринга страницы в браузере именно **Поле2**получит фокус, несмотря на порядок следования полей в форме (**Поле2** в середине).

Поле1: Поле без автофокуса

Поле2:

Поле3: Поле без автофокуса

Поле4: Поле без автофокуса

Отправить

Важно!

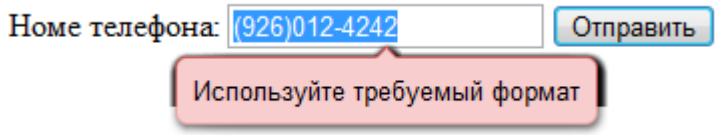
По поводу атрибута **autofocus** следует сделать одно важное замечание: на странице должно быть не более одного поля с атрибутом **autofocus**.

Ещё одно средство контроля правильности ввода значений поля – это атрибут **pattern**. С помощью этого атрибута и регулярных выражений можно задать требования к формату вводимых значений. В случае, если при отправке формы формат будет нарушен, браузер выведет сообщение об ошибке.

В качестве примера приведём поле ввода номера сотового телефон а с кодом оператора, заключённого в круглые скобки.

```
<form>
   Номер телефона: <input type="text" pattern="\([0-9]{3}\)[0-9]{3}-[0-9]{2}-[0-9]{2}" value="" >
   <input type="Submit">
</form>
```

Если мы введём номер (926)012-42-42, то браузер примет его, чего нельзя сказать, если мы будем вводить его как 926-012-42-42 или же (926)012-4242.



В данном случае чтобы пользователь не гадал о формате на кофейной гуще, желательно дополнить такое поле ввода подсказкой с помощью **placeholder**.

```
<form>
   Номе телефона: <input type="text" pattern="\([0-9]{3}\)[0-9]{3}-[0-9]{2}-[0-9]{2}" placeholder="(xxx)xxx-xx-xx" value="" >
   <input type="Submit">
</form>
```

http://adnotes.ru/uploads/html5intro/attr_pattern2.png

Совершенно аналогично атрибут **pattern** можно использовать и для организации других форматов полей. Новые возможности **html5** совместно со всей мощью регулярных выражений способны строить проверки форматов практически любой степени сложности.

Атрибут **required** указывает, что значение поля формы не может быть пустым.

required служит не для указания пользователю обязательности поля (такие поля обычно помечаются звёздочкой), он помогает браузеру выполнить контроль заполненности поля.

Пример:

```
<form>
   Поле с атрибутом required : <input type="text" value="" required >
   <input type="Submit">
</form>
```

Соответственно в случае, если в момент оправки формы поле заполнено не будет, браузер выдаст предупреждение:

Принцип автозаполнения уже прочно вошёл в нашу повседневную жизнь, и в этом плане **html5** тоже не остался в стороне. Итак, встречайте новый атрибут autocomplete. По умолчанию автозаполнение полей формы включено, и может быть переопределено в настройках браузера. Атрибут **autocomplete** позволяет переопределить настройку на уровне отдельного поля.

Если мы не можем допустить, что вводимые одним пользователем данные тот час же видел следующий, тогда надо отключить для таких полей автозаполнение.

```
<form>
   Поле ввода: <input type="text" autocomplete="off" value="" >
   <input type="Submit">
</form>
```

Атрибут **multiple** должен быть знаком многим ещё по старому стандарту html, однако в **html5** он может быть отнесён к полям загрузки файлов. Таким образом, в **html5** мультизагрузку файлов можно организовать без использования сторонних средств (Flash, Silverlight и т.д.).

```
<form enctype="multipart/form-data" method="post">
   Выберите файл: <input type="file" multiple>
   <input type="Submit">
</form>
```

Не забываем, что для загрузки файлов использовать метод отправки **post**, а также **multipart/form-data**.

Атрибут **list** используется совместно с новым элементом html5 – **datalist**. Пример.

```
<form>
   <label>Вы беситесь: <input type="text" list="spisok" ></label>
   <datalist id="spisok">
      <option value="с жиру">
      <option value="от безделья">
      <option value="постоянно">
      <option value="до умопомрачения">
   </datalist>
   <input type="Submit">
</form>
```

Поле ввода с вариантами выбора организовано с помощью элемента **input** с атрибутом **list** и элемента **datalist**. Так как эти два элемента не вложены друг в друга, то должен быть механизм, осуществляющий связь между ними. Таким механизмом служит атрибут **list** элемента **input**, указывающий на **id** списка (**datalist**). Напомним, что значение **id** уникально, поэтому неоднозначности не возникает.

В примере выше мы применили тип поля **text**, но совершенно так же можно было вместо него использовать тип **url** или **email**. Всё зависит от поставленной задачи, главное, чтобы тип элемента был уместен.

Связь элементов управления с формой можно организовать с помощью атрибута **form**. Мы ничего не перепутали, именно с атрибутом, а не одноимённым тегом!

Пример:

```
<form id="forma1">
   Поле1: <input value="ХА-ха-ха">
   Поле2: <input value="">
   Поле3: <input type="number" value="15">
</form>
   А теперь будет кнопка отправки той формы, что расположена выше.
   <input type="Submit" form="forma1">
```

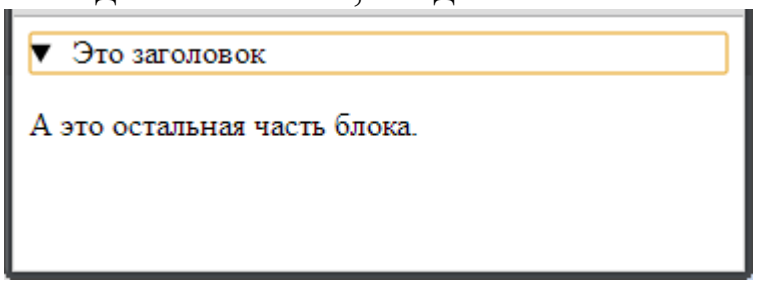
Всё очень просто, указываете значение **id** формы в атрибуте **form** элемента управления, и тогда этот элемент логически будет относиться к форме. Такая система даёт широкие возможности при написании кода страницы. Вы можете поместить элемент управления практически в любой части страницы, и при этом у него сохранится связь с формой. Обычно в роли таких вынесенных элементов выступают кнопки отправки данных форм, хотя с точки зрения правил **html5** это совсем не обязательно.

Элементы формы

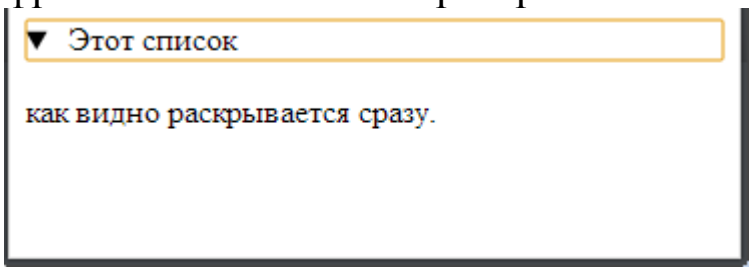
С помощью **details** создаётся фрагмент, часть содержания которого по умолчанию скрыта от пользователя. При нажатии на заголовок скрытая часть расправляется и демонстрируется пользователю. При создании таких конструкций совместно с **details** используется другой элемент – summary. Именно он и определяет заголовок, остальная часть блока до конца тега **details** будет скрыта и появится только после нажатия на заголовок.

```
<details>
  <summary>
    Это заголовок
  </summary>
    А это остальная часть блока.
</details>
```

В браузере Chrome выглядит такой блок следующим образом. Скриншот снят для состояния, когда блок полностью раскрыт.



У **details** есть очень важный атрибут – **open**, позволяющий отображать фрагмент в изначально раскрытом состоянии.



Чтобы завершить знакомство с новыми элементами html5, имеющими отношение к формам, расскажем ещё о **progress**, **meter** и **output**.

Элемент **progress** служит для конструирования полос выполнения работы (индикатор выполнения), позволяя организовать для пользователя

визуальное отображение процесса.

По умолчанию предполагаемым диапазоном значений элемента progress является интервал от 0 до 1. Однако с помощью соответствующих атрибутов, диапазон значений progress можно расширить, а также определить текущее значение. У **progress** есть пара атрибутов: **max**, означающее максимально возможное значение, и **value**, определяющее текущее состояние.

```
<form>
   <progress max="80" value="50">
   <input type="Submit">
</form>
```

Интересно, но браузер **Opera** отображает **progress** в виде статичной картинки,
в то время, как хромоподобные браузеры выводят индикатор выполнения в виде анимированный картинки.

Элемент **progress** служит для отображения состояния выполнения какого-то процесса, но не предназначен для визуализации долевых значений. Например, какая часть дискового пространства израсходована, или отобразить долю содержания кислорода в воздухе. Для таких целей предусмотрен специальный элемент под названием **meter**.

Элемент **meter** чем-то напоминает **progress** внешне, однако, они имеют совершенно разный смысл.

По сравнению с progress у **meter** возможных атрибутов больше. К ним относятся: **value**, **min**, **max**, **low**, **high** и **optimum**. Как вы уже догадались, значения атрибутов min, max и value служат для задания минимального, максимального и текущего значений соответственно.

```
<form>
   <meter min="0" max="100" value="75"></meter>
   <input type="Submit">
</form>
```

Вероятно, вы заметили, что визуально в браузере элемент **meter** отображается чуть иначе, нежели **progress**, рассмотренный нами выше.

Последним рассматриваемым на сегодня элементом является новый **output**.

Output – это некий контейнер, служащий для размещения в нём информации, полученной в результате вычисления или иным способом. Раньше для подобных целей использовался старый добрый **div** или же **span**, которые в общем случае можно использовать и сейчас.
Пример.

```
<form name="forma1" oninput="this.seconds.value=this.minutes.value*60">
  <label>Время в минутах: <input name="minutes" type="number" min="0" max="60" value="1" autofocus="autofocus" ></label>
  <label>Время в секундах: <output name="seconds" >60</output></label>
</form>
```

К форме привязан обработчик на javascript, который при каждом изменении значения поля времени в минутах вычисляет значение в секундах и выводит его в поле в поле **seconds** внутри **output**. Наглядный симбиоз новых для html5 типов элемента **input**, элемента **output** и **javascript**.

Html5 имеет свои средства проверки данных в формах, они устроены достаточно просто и удобны как для разработчика, так и для простого пользователя. Использование встроенных средств **html5** не только упрощает программирование, но и уменьшает размер служебного кода, что положительно сказывается на индексировании сайтов. Плюс грамотное использование семантического значения элементов. В общем, что касается форм, от html5 впечатление только положительное.