Министерство образования и науки Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

ФАКУЛЬТЕТ ДИСТАНЦИОННОГО ОБУЧЕНИЯ (ФДО)

Э. К. Ахтямов, Ю. П. Ехлаков

ОСНОВЫ ГИПЕРТЕКСТОВОГО ПРЕДСТАВЛЕНИЯ ИНТЕРНЕТ-КОНТЕНТА

Учебное пособие

Томск
2017

УДК [004.451.55+ 004.738.1](075.8)
ББК 32.973.202я73
А 957

Рецензенты:

О. Б. Фофанов, канд. техн. наук, доцент кафедры программной инженерии Национального исследовательского Томского политехнического университета;

Д. Н. Бараксанов, канд. техн. наук, начальник Центра веб-технологий и информационных ресурсов (ЦВТиИР) Томского государственного университета систем управления и радиоэлектроники

Ахтямов Э. К., Ехлаков Ю. П.

А 957 Основы гипертекстового представления интернет-контента : учебное пособие / Э. К. Ахтямов, Ю. П. Ехлаков. – Томск : Эль Контент, 2017. – 181 с.

ISBN 978-5-4332-0257-3

Приведены современные тенденции развития веб-технологий. Описаны возможности использования языков разметки и построения веб-сайтов, оформления внешнего вида веб-страниц с использованием каскадных таблиц стилей.

Для студентов, обучающихся по направлениям подготовки 09.03.04 «Программная инженерия» и 38.03.04 «Бизнес-информатика».

ISBN 978-5-4332-0257-3

Оглавление

Введение

Интернет перевернул все представления о компьютере и связи, как ни одна другая технологическая новинка [1]. Это одновременно способ глобальной передачи информации и механизм для взаимодействия людей по всему миру. Вряд ли Вы сейчас встретите человека, не знакомого с поисковыми системами, облачными хранилищами, социальными сетями, виртуальными магазинами, но, к сожалению, немногие знают об инструментах для создания этих технических чудес и людях, которые стоят за их разработкой.

Все более ускоряющееся развитие Интернета и сетевых технологий повлекло за собой появление соответствующих профессий в сфере IT: бэкенд-разработчик, веб-аналитик, CEO, контент-менеджер и многие другие. Одной из таких профессий является фронтенд-разработчик – специалист по созданию клиентской части сайтов и веб-приложений.

Дисциплина «Основы гипертекстового представления информации» направлена на изучение принципов построения и организации веб-страниц, особенностей веб-дизайна и знакомство с инструментами разработки сайтов.

Изучение процесса написания кода и отладки в рамках данной дисциплины осуществляется в текстовом редакторе Sublime Text и в браузерах на платформах Gecko (Mozilla Firefox 52) и Blink (Chrome 57). В качестве языка разметки выбран HTML 5.1, языком описания внешнего вида является CSS 3, а для программирования используется язык JavaScript 1.8.

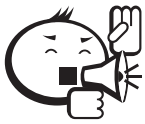
В данной дисциплине акцент делается на оформлении внешнего вида документа и взаимодействии с пользователем, поэтому основной объем работ связан с языками CSS и JavaScript; HTML изучается как основа для представления данных.

В результате изучения дисциплины студенты получают представление о процессе разработки клиентской части сайта, знакомятся с типами цветовых моделей и форматами изображений, изучают структуру DOM и овладевают инструментами работы с ним, учатся оптимизировать работу компонентов, используемых при разработке, овладевают навыками семантической разметки, узнают механизм работы потока в документе и порядок вывода страницы на экран. Знания, полученные в процессе изучения курса, позволят в дальнейшем использовать такие инструменты веб-разработки, как node.js, Angular 2, React, JQuery и др.

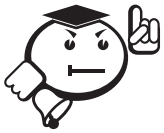
Соглашения, принятые в учебном пособии

Для улучшения восприятия материала в данном учебном пособии используются пиктограммы и специальное выделение важной информации.

Эта пиктограмма означает определение или новое понятие.



Эта пиктограмма означает «Внимание!». Здесь выделена важная информация, требующая акцента на ней. Автор может поделиться с читателем опытом, чтобы помочь избежать некоторых ошибок.



В блоке «На заметку» автор может указать дополнительные сведения или другой взгляд на изучаемый предмет, чтобы помочь читателю лучше понять основные идеи.

Контрольные вопросы по главе

1 История развития веб-технологий. Основы разметки информации с помощью HTML

История развития современных веб-технологий берет свое начало в 1989 г., когда сотрудник ЦЕРН[1] Тим Бернерс-Ли показал миру первый веб-сайт (рис. 1.1) и браузер, а также описал концепцию будущей Всемирной паутины, представляющую собой набор документов, связанных гиперссылками, и основу языка HTML [1].

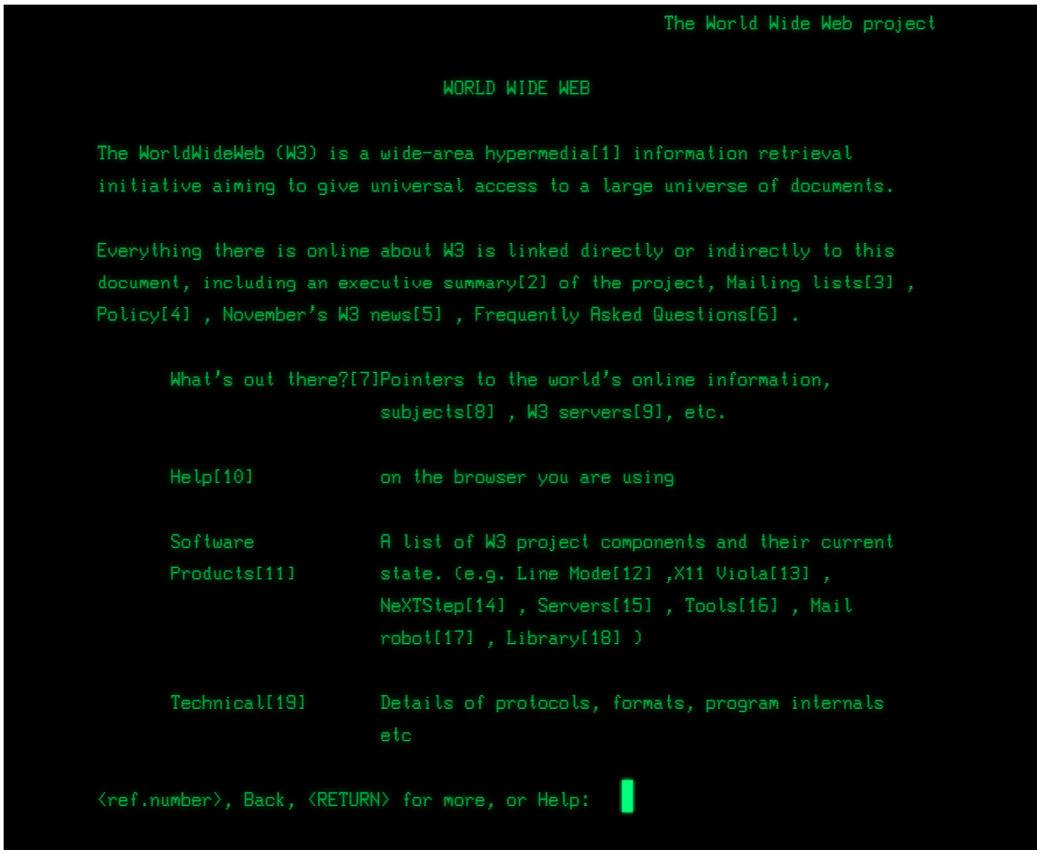


Рис. 1.1 – Первоначальный вариант сайта info.cern.ch

В первые пять лет (1990–1995 гг.) HTML несколько раз был доработан и испытал несколько расширений. С созданием Консорциума Всемирной паутины (W3C) разработчики HTML снова изменили направление своей деятельности. В первую очередь были прекращены попытки продолжения разработки HTML 1995 г., известного как HTML 3.0, затем сменили подход на более праг-

[1]От фр. CERN – Conseil Européen pour la Recherche Nucléaire (Европейский совет по ядерным исследованиям).

матичный, результатом чего стало появление HTML 3.2, который был готов в 1997 г. HTML 4.01 был разработан в следующие несколько лет.

В последующие годы члены W3C решили остановить развитие HTML и вместо этого начали работу над XML-подобным эквивалентом, получившим название XHTML. Разработка началась со слияния HTML 4.01 и XML, продуктом которого стал XHTML 1.0. После XHTML 1.0 разработчики W3C сместили фокус на его облегчение для разных рабочих групп с целью расширения. Параллельно с этим консорциум также работал над новым языком XHTML 2.0, который был не совместим с ранними HTML и XHTML.

Развитие HTML было приостановлено в 1998 г., после этого были определены части API для языка, разработанные создателями браузеров и опубликованные под названиями DOM level 1 (1998), DOM level 2 Core и DOM level 2 html (начат в 2000 г., максимальное развитие получил в 2003 г.).

Идея о том, что развитие HTML должно продолжаться, была рассмотрена в 2004 г. на мозговом штурме W3C, где некоторые принципы, лежащие в основе работы HTML, а также предварительный проект предложения, охватывающий только функции, связанные с формами, были представлены W3C вместе Mozilla и Opera. Предложение было отклонено на том основании, что оно противоречило ранее выбранному направлению развития Сети, поэтому сотрудники и члены W3C проголосовали за продолжение разработки замен на основе XML.

Немного позже Apple, Mozilla и Opera совместно анонсировали свое намерение продолжить работу над проектом под эгидой нового объединения, названного WHATWG[1]. Публичный почтовый список был создан, и черновики опубликовали на сайте WHATWG. Впоследствии авторские права были изменены, чтобы они находились в совместном владении всех трех поставщиков и позволяли повторно использовать спецификацию.

Последнее требование, в частности, направлено на то, чтобы область спецификации HTML включала сведения, ранее указанные в трех отдельных документах: HTML 4.01, XHTML 1.1 и DOM Level 2 HTML.

В 2006 г. W3C проявил интерес к участию в разработке HTML 5.0 и в 2007 г. сформировал рабочую группу для создания совместно с WHATWG спецификации HTML. Apple, Mozilla и Opera позволили W3C публиковать специ-

[1]От англ. Web Hypertext Application Technology Working Group – сообщество людей и компаний, заинтересованных в развитии Интернета.

фикацию под копирайтом W3C, сохраняя при этом версию с меньшее ограниченной лицензией на сайте WHATWG [3].

Несколько лет обе группы работали вместе под руководством Иана Хиксона. В 2011 г. группы пришли к выводу, что у них разные цели: W3C хотели устанавливать границы дозволенного для нововведений Рекомендации к HTML 5.0, в то время как WHATWG хотели продолжить работу над непрерывно обновляемой документацией HTML, постоянно обновляя спецификацию и вводя новые функции. В середине 2012 г. новый редактор команды был приглашен в W3C, чтобы позаботиться о создании Рекомендации к HTML 5.0 и подготовить Рабочий Черновик для следующей версии HTML.

С 1 ноября 2016 г. действует спецификация HTML 5.1.

1.1 Базовые понятия HTML

Кирпичиком разметки HTML документа является элемент.

Элемент *– основная единица HTML.*

Каждый элемент имеет своё предназначение. Например, абзац:

```
<p class="foo">CERN</p>
```

Каждый элемент состоит из:

- открывающего тега (или дескриптора) – `<p>`;
- закрывающего тега – `</p>`;
- атрибутов – `class="foo"`;
- и содержимого – `CERN`.

Тег *– имя элемента, заключенное в угловые скобки < >. Теги бывают открывающимися и закрывающимися.*

Весь документ состоит из вложенных друг в друга элементов. Чтобы один элемент мог вести себя по-разному, существуют атрибуты. Атрибут состоит из имени атрибута (`class`) и значения `foo`, окруженного кавычками `" "`. Есть атрибуты, которые можно применить ко всем элементам (например, `class`, `id`, `style` и т. д.), а есть специфичные – `href`, `src`. Бывают атрибуты без значений, например `disabled`.

Атрибуты предоставляют дополнительную информацию об элементе.

Синтаксически все элементы можно разделить на пять групп:

1. Void elements (пустые элементы) – не имеют содержимого и закрывающего тега, только атрибуты. Таких элементов всего 15: `<area>`, `<base>`, `<br>`, `<col>`, `<embed>`, `<hr>`, `<img>`, `<input>`, `<keygen>`, `<link>`, `<meta>`, `<param>`, `<source>`, `<track>` и `<wbr>`.

2. Raw text elements (элементы сырого текста – `script`, `style`) – в качестве содержимого могут иметь только текст, хотя на это существуют ограничения, описанные ниже.

3. Foreign elements (внешние элементы) – элементы из сторонних спецификаций со своим синтаксисом, например SVG или MathML.

4. Escapable raw text elements (элементы мнемонизируемого сырого текста) – могут содержать текст и ссылки на символы, но этот текст не должен содержать неоднозначный амперсанд (&). Имеются также дополнительные ограничения. Таких элементов два: `textarea`, `title`.

5. Normal elements (обычные элементы) – в качестве содержимого могут иметь как текст, так и другие вложенные элементы. Например элемент `<p>`, описывающий параграф текста:

```
<p>
    <a href="http://home.web.cern.ch/">CERN</a>
     is a European research organization that operates
     the largest particle physics laboratory in the
     world.
</p>
```

Внутри элемента `<p>` (paragraph) лежит элемент `<a>` (anchor) и текст.

Содержимое элемента сырого текста и элементы мнемонизируемого сырого текста не должны содержать никаких строк `"</"` (U+003C LESS-THAN SIGN, U+002F SOLIDUS) с последующими символами, которые регистронезависимо совпадают с именем тэга элемента, одним из следующих: "tab" (U+0009), "LF" (U+000A), "FF" (U+000C), "CR" (U+000D), U+0020 SPACE, ">" (U+003E) или "/" (U+002F) [4].

В коде HTML документа (как и в языках программирования) можно оставлять комментарии:

```
<!-- comment -->
```

Обычно комментарии оставляют для пояснения кода, но они могут использоваться и для специальных указаний браузеру:

```
<!--[if IE 8]><link rel="stylesheet" href="ie8.css"/>
<![endif]-->
```

Благодаря такому комментарию, браузер Internet Explorer понимает, что файл со стилями оформления ie8.css нужно загрузить, только если версия браузера равна 8.

Текст в HTML разрешён только внутри элементов, значений атрибутов и комментариях. Между атрибутами его размещать нельзя.

Новая строка может быть обозначена как символом CR (carriage return) – возврат каретки, так и LF (line feed) – подача на строку, так и двумя сразу – CR LF.

Чтобы вставить особые символы, например ©, для удобства можно использовать символы-мнемоники (Entity Characters):

`<p>©</p>` `<!-- © -->` Указание `&` и `;` – обязательно.

1.2 Типы разметки

По назначению разметку делят на четыре группы:

1) описательная (Descriptive Markup) – определяет метаинформацию документа. Например, ключевые слова, автора документа, заголовок и кодировку;
2) структурная (Structural Markup) – определяет структуру и назначение текста;
3) визуальная (Presentational Markup) – определяет то, как элемент будет выглядеть;
4) ссылочная (Hypertext Markup) – обозначает части документа как ссылки. В спецификации HTML5 такой элемент только один – `<a>`.

Визуальная разметка появилась до широкого распространения стилей CSS, но на сегодняшний день уже рекомендуется её не использовать. Вместо элементов

```
<p><center>centered text</center></p>
```

рекомендуется использовать стили

```
<p style="text-align: center">centered text</p>
```

1.3 Основные теги

В основе документа лежит корневой элемент `html`. Он определяет границы документа:

`<html lang="ru"> </html>`

Рекомендуется указывать у него атрибут языка `lang="ru"`. Атрибут языка помогает инструментам синтеза речи для невидящих людей правильно выбирать произношение.

Для того чтобы отделить описательную разметку от структурной, метаинформацию для роботов, браузеров и поисковых систем от содержимого для пользователей есть два элемента – `body` и `head`.

Элемент `head` хранит в себе набор элементов, определяющих метаинформацию документа для роботов, браузеров и поисковых систем (например, ключевые слова, автора документа, заголовок). Этот элемент не является обязательным и его можно пропустить.

```
<html lang="en">
     <head>
          <title>Example</title>
     </head>
</html>
```

Элемент `body` хранит в себе набор элементов, определяющих содержимое документа для пользователей, и также как и `head` является необязательным.

```
<html lang="en">
     <body>
       <p>CERN is a European research organization that
       operates the largest particle physics laboratory
       in the world.</p>
     </body>
</html>
```

Хотя теги `html`, `head` и `body` являются необязательными, хорошим стилем считается их использование, чтобы обозначить структуру документа.

Так как элементов в HTML очень много, для удобства их можно разделить на несколько больших групп:

- метаэлементы;
- секционные элементы;
- заголовочные элементы;

- группирующие элементы;
- текстовые элементы;
- табличные элементы;
- ссылочные элементы;
- элементы форм;
- элементы встраиваемого содержимого.

За каждым элементом скрывается вполне определённая задача. Чем точнее будет подобран элемент под задачу, тем дружелюбнее будет разметка как для пользователей, так и для поисковых систем.

Метаэлементы

<title></title> – определяет заголовок документа. Не виден в тексте, но используется для отображения на вкладке браузера или в истории посещений. Это единственный обязательный тег, и он не должен быть пустым. Поэтому минимально возможный корректный HTML-документ выглядит так:

```
<!DOCTYPE html>
<title> </title> <!-- Название состоит из одного пробела -->
```

<link> – устанавливает связь с внешним документом, например с CSS-файлом (где хранятся стили). Атрибут `href` описывает путь к файлу стилей, `rel` (relationship) описывает тип связи «по ссылке файл со стилями», и без этих атрибутов использовать элемент нельзя.

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

<script></script> – предназначен для описания скриптов, может содержать ссылку на скрипт или его текст на определённом языке. Атрибут `type` имеет значение `text/javascript` по умолчанию, и его можно не указывать. Ранее предполагалось, что JavaScript будет не единственный скриптовым языком для браузеров. Использовался VBScript (язык от Microsoft), его тип – `text/vbscript`. На сегодняшний день для использования в клиентской части сайта доступен только JavaScript.

```
<script src="path/to/my-jquery-plugin.js" type="text/javascript"></script>
```

<style></style> – предназначен для определения стилей в коде.

```
<style>
    p { color: red; }
</style>
```

<meta> – метатег – определяет метаинформацию документа. Например, ключевые слова, автора документа, заголовок, кодировку.

```
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="keywords" content="CERN, European, phys-
ics">
  </head>
</html>
```

Атрибут `name` задаёт название, `content` – значение. Наиболее известны:

- `author` – имя автора документа;
- `description` – описание текущего документа;
- `keywords` – список ключевых слов.

Особый метатег **<meta charset="utf-8">** задаёт кодировку всего документа. Его рекомендуется располагать до любого текста на странице, включая `<title>`.

В исходной спецификации HTML5 не так много метатегов, поэтому компания Facebook расширила его спецификацией Open Graph. Результат использования OpenGraph можно встретить при размещении ссылок в социальных сетях: благодаря такой разметке рядом со ссылкой появляется изображение, краткое описание и другая полезная информация.

Помимо Facebook разметку Open Graph распознают также Яндекс, ВКонтакте, Google+, Twitter, LinkedIn, Pinterest и др.

Кроме информационных метатегов есть технические. Технические метатеги фактически копируют HTTP-заголовки и влияют на поведение документа. Вместо `name` в них просто используется атрибут `http-equiv` (HTTP Header Equivalent).

Секционные элементы

Секционные элементы необходимы для деления документа на смысловые части: основное содержимое, неосновное, навигация, шапка, подвал. К этим элементам относятся: `article`, `section`, `aside`, `header`, `footer`, `nav`, `address`.

Элемент **<article></article>** обозначает законченную самодостаточную часть документа, которая вполне может существовать независимо: пост на форуме или в блоге, новостная статья или виджет (например, календарь),

комментарий. Элемент может содержать вложенные <article>. Так, пост в блоге может включать в себя комментарии посетителей, каждый из которых является <article>.

Элемент **<section></section>** определяет раздел в документе, такой как глава, шапка, подвал или любая другая часть документа. Блоки article и section предполагают наличие хотя бы одного заголовка.

Элемент **<nav></nav>** определяет блок с навигацией по связанным документам. Его рекомендуется использовать только для главной навигации. Например, для ссылок в подвале документа лучше использовать элемент <footer>, а не <nav>. Браузер может использовать ссылки в этом элементе, чтобы предсказывать дальнейшие действия посетителей и предзагружать страницы.

Элемент **<aside></aside>** определяет блок с сопутствующим содержимым. Например, для обозначения сносок или цитат. Также этот элемент может использоваться для группировки нескольких элементов <nav>.

Элемент **<header></header>** определяет шапку всего документа или шапку секции.

Элемент **<footer></footer>** определяет подвал всего документа или подвал секции.

Элемент **<address></address>** определяет блок с контактной информацией, относящейся к ближайшему <article> или <body>. <address> не должен содержать другой информации, кроме контактной: почтовый адрес, электронная почта, домашняя страницы в интернете.

Заголовочные элементы

Элементы <h1>, <h2>, <h3>, <h4>, <h5>, <h6> определяют заголовок для секционных элементов или <body>. Число определяет уровень: 1 – наивысший, 6 – наименьший.

```
<header>
     <h1>Грамматика немецкого языка</h1>
</header>
<article>
  <h2>Имя существительное (Substantiv)</h2>
     <section>
          <h3>Определенный и неопределенный артикли</h3>
```

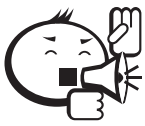
```
            <p>В немецком языке не бывает просто дерева. Мо-
            жет быть либо ein Baum [айн баум] - одно (какое-
            либо) дерево, либо der Baum [дэа баум] - то (са-
            мое) дерево.</p>
        </section>
    </article>
```

Группирующие элементы

Группирующие элементы объединяют логические блоки внутри секций – `main`, `p`.

Элемент **`<main></main>`** определяет главное содержимое страницы. Согласно спецификации W3C данный элемент может быть только один (в версии WHATWG такого ограничения нет).

Элемент **`<p></p>`** (paragraph) определяет параграф текста. Внутри `<p>` можно размещать только текстовые элементы:



```
<!-- Хорошо -->
<p>
  <mark>Яндекс.Браузер</mark>     предостерегает
  пользователей, когда они начинают вводить па-
  роль на подозрительных страницах.
</p>

<!-- Плохо -->
<p>
<h1>Заголовок первого уровня</h1>
</p>
     Последний пример браузер поймёт как:
<p>Список покупок</p>
     <h1>Заголовок первого уровня</h1>
<p></p>
```

Не рекомендуется использовать `<p>`, если есть более подходящий элемент. Например, для электронной почты лучше использовать `<address>`.

Элемент **`<hr>`** (horizontal rule) разделяет параграфы текста, например когда сменилась тема повествования. По умолчанию браузер рисует горизонтальную линию.

Элемент **<pre></pre>** (preformatted text) определяет блок предварительно форматированного текста. Такой текст отображается моноширинным шрифтом, сохраняя всё форматирование: пробелы, отступы.

Элемент используется когда необходимо вывести участок кода или ASCII графику:

```
<pre>
    <code>function Panel(element, canClose) {
    this.element = element;
    this.canClose = canClose;
    }</code>
</pre>
```

Элемент **<blockquote></blockquote>** предназначен для выделения длинных цитат внутри документа. Атрибут `cite` определяет ссылку на источник цитаты.

```
<blockquote cite="http://example.com">
    Life is what happens to you while you’re busy making
    other plans. // Джон Леннон
</blockquote>
```

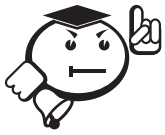
Элемент **<ol>** (ordered list) определяет упорядоченный список – при выводе на экран каждый элемент списка будет пронумерован. Элемент `li` (list item) определяет элемент списка.

Элемент **<ul>** (unordered list) определяет НЕупорядоченный список – при выводе на экран каждый элемент списка будет обозначен точкой или любым другим символом. Списки могут быть вложены друг в друга.

Элемент **<dl></dl>** (definition list) задаёт список «термин – описание». Внутри него задается термин элементом **<dt></dt>** (definition term) и описание термина – элементом **<dd></dd>** (definition description).

```
<dl>
    <dt>GIF</dt>
    <dd>Формат графических файлов, широко применяемый
    при создании сайтов.
    GIF использует 8-битный цвет и эффективно сжимает
    сплошные цветные области,
    при этом сохраняя детали изображения.</dd>
</dl>
```

Каждому термину `<dt>` может соответствовать несколько описаний `<dd>`.



. . . . . .

Этот элемент также может определять любые списки типа «ключ – значение», а не только «термин – описание».

```
<dl>
    <dt>Дата</dt>
    <dd>20 сентября 2015</dd>
    <dt>Автор</dt>
    <dd>Артур</dd>
</dl>
```

. . . . . .

Элемент **<figure></figure>** используется для группирования любых элементов и добавления подписи к ним с помощью элемента **<figcaption></figcaption>**. Чаще всего используется для добавления подписей к изображениям или коду.

Элемент ** ** (division, раздел) – универсальный группирующий элемент с целью изменения вида содержимого через стили.

```
<div style="color: red;">
<p>Как только пользователь устанавливает курсор мыши в поле для ввода пароля на любом сайте, которого нет в списке, активируется система защиты.</p>
</div>
```

Элемент не несёт смысловой нагрузки, поэтому его не рекомендуется использовать, если есть более подходящий.

Табличные элементы

Элемент **<table></table>** предназначен для разметки табличных данных.

<thead></thead> обозначает шапку таблицы.

<tr></tr> обозначает строку данных.

<th></th> обозначает ячейку шапки.

<tbody></tbody> обозначает данные таблицы.

<td></td> обозначает ячейку таблицы.

```
<table>
   <thead>
     <tr>
       <th>Game name</th>
       <th>Game publisher</th>
```

```
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Diablo</td>
      <td>Blizzard</td>
    </tr>
    <tr>
      <td>Portal</td>
      <td>Valve</td>
    </tr>
  </tbody>
</table>
```

В результате работы кода будет выведена таблица из двух столбцов и трех строк (рис. 1.2). Вообще элемент `<tbody></tbody>` является необязательным, однако при выводе в браузере, если открыть «Панель разработчика», можно заметить, что этот элемент присутствует. Такова особенность вывода таблиц, поэтому хоть этот элемент и является необязательным, во избежание недоразумений его лучше использовать.

Game name	Game publisher
Diablo	Blizzard
Portal	Valve

Рис. 1.2 – Результат выполнения кода

Текстовые элементы

Элемент **** (emphasis, акцент) предназначен для акцентирования текста.

```
<p><em>Cats</em> are cute animals.</p>
```

`<em>` обозначает ключевую часть предложения (не обязательно важного), в то время как `<strong>` обозначает именно важную информацию.

Элемент **** предназначен для обозначения важной информацию.

```
<p><strong>Внимание! Идут учения!</strong></p>
```

Элемент **<ins></ins>** (inserted text) предназначен для выделения текста, который был добавлен к текущему. Атрибут `datetime` указывает на время добавления текста. По умолчанию браузер выделяет его подчёркиванием.

Элемент **<del></del>** (deleted text) предназначен для выделения текста, который был удалён. Атрибут `datetime` указывает на время удаления текста. По умолчанию браузер выделяет его зачёркиванием.

Элементы `<del>` и `<ins>` чаще всего используются для того, чтобы показать, что изменилось в документе с момента последней его редакции.

Элемент **<mark></mark>** (marked text) помечает текст как выделенный. По умолчанию браузер выделяет его жёлтым маркером. Удобно использовать этот элемент и для выделения участка кода:

```
<pre>
   <code>function <mark>Panel</mark>(element, canClose) {
    this.element = element;
    this.canClose = canClose;
   }</code>
</pre>
```

Элемент **<small></small>** предназначен для различных оговорок. Например, для условий и правовых ограничений в рекламе. По умолчанию браузер выделяет его мелким шрифтом.

Элемент **<cite></cite>** (отсылки) предназначен для выделения названий книг, песен, фильмов; имён авторов (то есть отсылок к ним).

```
<p>Роман Замятина <cite>Мы</cite> – одна из лучших книг-антиутопий</p>
```

Элемент **<q></q>** (quote, цитата) предназначен для выделения цитат или обозначения прямой речи.

```
<p><q>Le général Koutouzoff,</q> – сказал Болконский, ударяя на последнем слоге zoff, как француз</p>
```

Элемент **<dfn></dfn>** (definition, определение) предназначен для выделения определений, терминов.

```
<p><dfn>Гипотенуза</dfn> – сторона прямоугольного треугольника, лежащая против прямого угла.</p>
```

Элемент **<abbr></abbr>** (abbreviation) предназначен для выделения аббревиатур. Браузер при наведении покажет подсказку с расшифровкой.

```
<abbr title="Conseil Européen pour la Recherche Nucléaire">CERN</abbr>
```

Элемент **<ruby></ruby>** (ruby annotations, сноски) предназначен для добавления небольшой аннотации сверху или снизу от заданного текста.

Элемент **<time></time>** предназначен для выделения дат.

```
<article>
   <p><time>1969-07-21</time>  высадка  человека  на
Луну.</p>
</article>
```

Элемент **<samp></samp>** (sample output) используется для отображения текста, который является результатом вывода компьютерной программы или скрипта.

Элемент **<kbd></kbd>** (keyboard) используется для обозначения текста, который набирается на клавиатуре, или для названия клавиш.

```
<article>
   <p>Нажмите <kbd>CTRL</kbd> + <kbd>S</kbd> на клавиату-
ре.
   <p>Компьютер должен ответить <samp>Document was saved
successfuly</samp>.</p>
</article>
```

Элемент **<code></code>** используется для отображения текста, который является результатом вывода компьютерной программы или скрипта.

Элемент **<var></var>** (variable) используется для выделения переменных в компьютерных программах или математических выражениях.

```
<p>
   Формула эквивалентности массы и энергии:
   <var>E</var> = <var>m</var> <var>c</var><sup>2</sup>
</p>
```

Элемент **** (superscript) обозначает текст как верхний индекс. Например, для обозначения степени:

```
<p>
   Формула эквивалентности массы и энергии:
   <var>E</var> = <var>m</var> <var>c</var><sup>2</sup>
</p>
```

Элемент **** (subscripts) обозначает текст как нижний индекс.

```
<p>
   Формула воды: H<sub>2</sub>O.
</p>
```

Элемент **
** (break line) устанавливает перевод строки в том месте, где он находится.

```
<p>
   Варкалось. Хливкие шорьки<br>
   Пырялись по наве,<br>
   И хрюкотали зелюки,<br>
   Как мюмзики в мове.<br>
</p>
```


 не рекомендуется использовать для разделения текста на тематический группы, для этого используется элемент <p> (параграф).

Элемент **<wbr>** (word break) указывает браузеру место, где допускается делать перенос строки в длинном слове:

```
<p>метоксихлор<wbr>диэтиламино<wbr>метил<wbr>бутил<wbr>амино<wbr>акридин</p>
```

Элемент **** (span, интервал) – универсальный текстовый элемент, используемый для изменения вида содержимого через стили. Этот элемент не несёт смысловой нагрузки, поэтому его не рекомендуется использовать, если есть более подходящий.

Визуальная разметка i, b, center, s, u

**** (bold) выделяет текст жирным шрифтом.

**** (italic) выделяет текст курсивом.

** ** размещает текст по центру.

<s></s> (strikethrough) выводит текст зачёркнутым.

<u></u> выводит текст подчёркнутым.

Напоминаем, что визуальная разметка появилась до широкого распространения стилей CSS, и на сегодняшний день использовать её не рекомендуется.

1.4 Интерактивные элементы

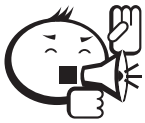
Элемент **<a>** (anchor) является основой любого HTML-документа. Он позволяет связать HTML-документы и другие ресурсы сети друг с другом, образуя сеть Всемирной паутины. Ссылки – очень гибкий элемент и имеет много разновидностей.

Например, ссылки могут вести к другому HTML-документу в Сети:

```
<a
   href="http://home.web.cern.ch/"
   hreflang="en"
   target="_blank"
   title="Официальный сайт CERN">
   CERN
</a>
```

или к электронной почте, номеру телефона:

```
<a href="mailto:press.office@cern.ch">Написать нам письмо</a>
<a href="tel:+41 (0)22 767 2757">Или позвонить</a>
```



Самый важный атрибут ссылки – `href` (hyperlink reference). Он задаёт собственно ссылку на другой документ или ресурс. В качестве значения атрибута используется URL (Uniform Resource Locator) – единообразный указатель (адрес) ресурса в сети Интернет.

Рассмотрим в качестве примера адрес расписания факультета систем управления: https://timetable.tusur.ru/faculties/fsu.

Любой URL состоит из двух частей: метод доступа к ресурсу https и адрес ресурса //timetable.tusur.ru/faculties/fsu, разделённые двоеточием «:».

Метод доступа обычно представляет собой идентификатор протокола, который определяет, какая программа будет обрабатывать событие нажатия на ссылку. Подробнее о методах доступа можно узнать из таблицы 1.1.

Таблица 1.1 – Методы доступа к ресурсу

Протокол	**Пример**	**Приложение**	**Действие**
http	http://yandex.ru/support/search	Браузер	Загрузит новый HTML документ по протоколу HTTP (HyperText Transfer Protocol)
https	https://timetable.tusur.ru/faculties/fsu	Браузер	Загрузит новый HTML документ по защищённому протоколу HTTPS (HyperText Transfer Protocol Secure)

Протокол	Пример	Приложение	Действие
mailto	mailto:press.office@cern.ch	Почтовый клиент	Откроет форму создания письма
skype	skype:i-dont-like-skype	Skype	Откроет чат с пользователем
tel	tel:+43-000-00-000	Skype, FaceTime	Попробует позвонить абоненту по номеру
file	file://home/Users/you/.doc	Браузер	Попробует открыть файл на Вашем компьютере локально

Протокол http является протоколом по умолчанию и его можно не указывать.

Разберём адрес HTML документа в сети Интернет чуть подробнее. Полный формат выглядит так:

https://<логин>:<пароль>@<хост>:<порт>/<путь>?<параметры>#<якорь>

Например, адрес

https://admin:123456@yandex.ru:443/search?text=paraweb#resutls

подробно разобран в таблице 1.2.

Таблица 1.2 – Разбор адреса HTML документа

Параметр	Пример	Назначение
<логин>	admin	Имя пользователя для доступа к ресурсу (если необходимо)
<пароль>	123456	Пароль для доступа к ресурсу (если необходим)
<хост>	yandex.ru	Доменное имя сервера (Domain Name), где расположен ресурс
<порт>	443	Номер порта, по которому доступен ресурс. По умолчанию 80 для HTTP и 443 для HTTPS – если так, его можно не указывать. На одном и том же сервере по разным портам можно получать разные ресурсы

Параметр	Пример	Назначение
<путь>	/search	Путь до ресурса. В примере форма поиска по Интернету. Это может быть HTML-страница или файл
<параметры>	text=paraweb	Параметры доступа к ресурсу. В примере запрос «paraweb» в форме поиска
<якорь>	results	Прокрутить страницу до элемента с `id="results"`

__Доменное имя__ или __домен__ (Domain Name) – это уникальное виртуальное имя сервера в сети Интернет в рамках DNS (Domain Name System). DNS – компьютерная распределённая система для получения физического адреса сервера (IP-адреса) в сети Интернет по виртуальному и человекопонятному адресу (доменному имени).

Таблицы соответствий IP-адреса домену хранятся на специальных DNS-серверах по всему миру.

Когда Вы в браузере набираете yandex.ru, то он в первую очередь спрашивает DNS-серверы и получает от них в ответ IP-адрес 77.88.55.66. Затем у сервера с этим IP-адресом браузер уже запрашивает HTML-документ.

Доменные имена позволяют запрашивать ресурсы сети Интернет в человекопонятном виде, а также размещать сразу несколько ресурсов на одном сервере (с одним IP). Сервер по доменному имени сам поймёт, какой ресурс отдать пользователю.

Все ссылки с протоколом http можно разделить на четыре вида:

1. Абсолютные ссылки href="http://yandex.ru/support/search" или href="//yandex.ru/support/search" – обязательно содержат <хост>. Если протокол не указан, будет использован протокол текущего документа.
2. Относительные ссылки href="/support/search" – в качестве <хост> используют доменное имя текущего документа.
3. Якоря href="#result" – при клике прокручивают до элемента (переходят к элементу) с `id`, равным идентификатору, указанному после #.

4. Ссылки на скачивание <a href="https://whatwg.org/pdf" download> – задаются при помощи атрибута `download`. Он указывает браузеру, что документ необходимо именно скачать, а не показать (а современные браузеры умеют показывать даже PDF). С помощью атрибута `type` можно подсказать браузеру и пользователю MIME-тип скачиваемого файла type="application/pdf".

MIME-тип *– идентификатор типа файла согласно спецификации MIME (Multipurpose Internet Mail Extensions) – стандарт, описывающий передачу различных типов данных (изначально для передачи в тексте почтовых сообщений).*

Идентификатор MIME имеет следующий формат:

<базовый тип>/<тип>

Например:

`application/zip` – архив zip;

`application/pdf` – PDF-файл;

`text/css` – файл с CSS-стилями;

`video/mp4` – mp4-видеофайл.

Базовые типы представлены в таблице 1.3.

Таблица 1.3 – Базовые типы

Тип	Описание
application	Прикладные программы
audio	Аудиофайлы
image	Изображения
message	Текстовый сообщения
model	3D-модели
multipart	Несколько типов файлов в одном
text	Текстовый файлы
video	Видеофайлы

Элементы встраиваемого содержимого

Если в HTML-документ вставить содержимое других типов, кроме текста мы можем воспользоваться группой элементов встраиваемого содержимого. С помощью них Вы можете вставлять аудио, видео, изображения и др.

Такими типами являются `<svg>` (для вставки векторной графики) и `<math>` (для размещения математических формул). Рассмотрим ряд других.

При помощи элемента **`<img>`** в страницу встраивается изображение:

```
<img
     src="//yastatic.net/morda-logo/i/bender/logo.svg"
     alt="Yandex">
```

Путь до изображения указывается в атрибуте src, альтернативный текст в атрибуте `alt` для пользователей с ограничением зрения или тех, у кого картинки отключены.

Так как в современном мире достаточно большое разнообразие типов устройств, необходимо следить за тем, чтобы изображения выглядели хорошо везде. Устройства различаются разрешением экрана и плотностью точек (retina, 4k).

Для этого мы можем использовать атрибут `srcset`:

```
<img
     src="//yastatic.net/logo.png"
     alt="Yandex"
     srcset="//yastatic.net/logo-2x.png 2x"
>
```

Более гибкий элемент для этого – **`<picture></picture>`**, но он пока поддерживается не всеми браузерами.

Элемент **`<iframe>`** позволяет встроить один HTML-документ в другой. Например, встроить систему покупки билетов на сайт кинотеатра.

```
<iframe
src="//booking.yandex.ru/organizations?permalink=1225142754"
width="800" height="800">
```

Элемент **`<audio>`** позволяет вставлять на страницу аудиопроигрыватель, а `video` – соответственно видеопроигрыватель. Эти элементы пришли на смену Flash в спецификации HTML5.

1.5 Элементы форм

Если необходимо не только давать, но и принимать информацию от пользователя, то есть взаимодействовать с ним, то понадобится группа элементов для описания форм.

Элемент **<form></form>** представляет собой набор инструментов для ввода информации и кнопок для различных действий с этой информацией, чаще всего для её отправки на сервер.

Например, форма редактирования данных о книге «Война и мир»:

```
<form action="/book/war-and-peace" method="POST">
    <!-- Поле для ввода заголовка книги -->
    <input name="title" value="Война и мир">
    <!-- Поле для ввода автора книги -->
    <input name="author" value="Толстой Л.Н.">
    <!-- Кнопка сохранения -->
    <!-- При нажатии на неё браузер соберёт все данные и
отправит по адресу /book/war-and-peace -->
    <button type="submit">Сохранить</button>
<form>
```

В данной форме представлено два поля и одна кнопка. Данные формы отправляются по нажатию кнопки с атрибутом и значением `type="submit"` либо по нажатию клавиши ENTER.

После этого браузер соберёт все введённые в поля данные в объект. Для этого из атрибута `name` он возьмёт название поля, а из `value` – значения:

```
{
    title: 'Война и мир',
    author: 'Толстой Л.Н.'
}
```

Затем он запакует их в виде строки текста в формате `<поле1>=<значение1>&<поле2>=<значение2>` и специальным образом закодирует значения:

```
title=%D0%92%D0%BE%D0%B9%D0%BD%D0%B0%20%D0%B8%20%
D0%BC%D0%B8%D1%80&author=%D0%A2%D0%BE%D0%BB%D1%
81%D1%82%D0%BE%D0%B9%20%D0%9B.%D0%9D.
```

Часть символов `a-Z 0-9 - _ . ~` не кодируется. А остальные – кодируются при помощи Percent-encoding – специального механизма кодирования символов в формат %<номер> согласно их номеру в utf-8.

Например, в таблице UTF-8 encoding table в колонке utf-8 представлен номер буквы В – d0 92 и после кодировки получается %D0%92.

Наконец, браузер отправит HTTP-запрос с закодированными данными по адресу /book/war-and-peace, указанному в атрибуте `'action'`.

У формы есть ещё один важный атрибут – `method`. В нём указан метод HTTP-протокола, или, простыми словами, тип действия.

Так как у ресурса в Сети обычно один адрес: /book/war-and-peace, разработчикам необходимо различать, что с ним делать: смотреть, редактировать или удалять. Для этого были разработаны методы HTTP-протокола (табл. 1.4).

Таблица 1.4 – Основные методы HTTP-протокола

Метод	**Назначение**
GET	Прочитать ресурс
PUT	Создать ресурс
POST	Отредактировать данные ресурса
DELETE	Удалить ресурс

Например, HTTP на запрос вида DELETE /book/war-and-peace удаляет книгу.

К сожалению, HTML поддерживает только два метода: GET и POST. Остальные эмулируются при помощи скрытых полей `<input name="http-method" type="hidden" value="DELETE">`.

Вторым важным атрибутом формы является атрибут `enctype`. С его помощью можно кодировать данные (табл. 1.5).

Таблица 1.5 – Кодирование данных с помощью атрибута `enctype`

Значение	**Описание**
application/x-www-form-urlencoded	Вместо пробелов ставится +, символы кодируются при помощи Percent-encoding
multipart/form-data	Данные не кодируются. Это значение применяется при отправке файлов

Значение	Описание
text/plain	Пробелы заменяются знаком +, буквы и другие символы не кодируются

Выше был рассмотрен вид кодирования application/x-www-form-urlencoded. Он используется по умолчанию, поэтому не был указан в примере.

Кодирование multipart/form-data необходимо, когда надо передать в одном запросе несколько типов данных. Например, текстовые данные и файлы.

Данные в этом случае будут передаваться как есть, но будут разделены специальными, случайным образом сгенерированными разделителями.

`Content-Disposition: form-data; name="author"`

Каждый блок соответствует своему полю и содержит: имя поля `Content-Disposition: form-data; name="author"`; MIME-тип поля `Content-Type: image/png` (если отличается от `text/plain` – простой текст) и сами данные.

Основным элементом, позволяющим получать данные от пользователя, является **`<input>`**. Он позволяет добавить в форму поля различных типов. Атрибут `name` задаёт название поля, `value` – его текущее значение, а `type` – тип. По умолчанию поле является текстовым, т. е. предназначенным для ввода любого небольшого текста.

Рассмотрим ещё примеры типов полей:

- `type="file"` – поле для выбора файла, в котором можно ограничить MIME-типами в атрибуте «accept» типы выбираемых файлов;
- `type="checkbox"` – поле для ответа на вопрос в формате «да/нет». Можно поставить галочку (означает «да») нажатием на `label` или `input`. Это поле можно использовать для выбора нескольких значений. Атрибут `checked` обозначает уже выбранное поле;
- `type="hidden"` – скрытое поле. Оно никак не отображается на странице и в основном используется для отправки технических данных;
- `type="pass"` – поле для ввода пароля. При вводе пароль не виден, посторонние не смогут его подсмотреть;
- `type="radio"` – поле для выбора одного значения из предложенных. Значения группируются с помощью атрибута `name`.

В HTML5 добавлены новые типы:

- `type="color"` – поле для ввода цвета;
- `type="date"` – поле для ввода даты при помощи календаря. С помощью атрибутов `'min'` и `'max'` можно ограничить вводимую дату;
- `type="email"` – поле для ввода электронной почты;
- `type="number"` – поле для ввода числа. С помощью атрибутов `'min'` и `'max'` можно ограничить вводимое число. С помощью атрибута `'step'` можно задать шаг приращения числа;
- `type="tel"` – поле для ввода телефона;
- `type="url"` – поле для ввода URL;
- `type="time"` – поле для ввода времени.

Для каждого поля можно вывести подпись при помощи элемента `label`. Чтобы связать их, у элемента `label` в атрибуте `for` должно быть то же значение, что и у элемента `input` в атрибуте `id`. При клике на `label` браузер переводит фокус (выделяет и устанавливает курсор) на связанное поле.

Если подпись необходима, но места для неё нет, можно использовать атрибут `placeholder` у самого элемента `input`. Тогда подпись будет прямо в поле ввода. Атрибут может быть использован только в подходящих для этого типах полей.

Практически всегда необходимо не только давать пользователю вводить данные HTML5, но и проверять их (валидировать). Для этого есть ряд специальных атрибутов:

- `required` – обозначает поле как обязательное. Если значение не будет введено, то форма не отправится, а напротив этого поля возникнет подсказка с ошибкой;
- `pattern="[0-9]{6}"` – ограничивает ввод регулярным выражением. Если введённое значение не будет ему удовлетворять, то форма не отправится, а напротив этого поля возникнет подсказка с ошибкой;
- `readonly` – обозначает поле как «только для чтения»;
- `disabled` – обозначает поле как «отключенное». В отличие от `readonly`, значение этого поля не будет отправлено совсем.

Для отправки данных нужна кнопка **`<button></button>`**. Кнопку можно заблокировать атрибутом `disabled`.

Если необходимо, чтобы после загрузки формы фокус был сразу на кнопке, можно указать атрибут `autofocus`.

Существует ряд полей, которые исторически определяются другими элементами, например **`<textarea></textarea>`**. Как и у элемента `<input>`, у него есть все необходимы атрибуты: `disabled`, `readonly`, `maxlength`, `required`.

Как и элемент `<input type="radio">`, элемент `<select>` предлагает пользователю выбрать один вариант из предложенных в удобном и компактном виде.

По умолчанию всегда выбран первый вариант, но можно пометить необходимый атрибутом `selected` (будет выбран последний с таким атрибутом).

1.6 Глобальные атрибуты

Как было сказано выше, у каждого элемента есть набор атрибутов, которые задают его свойства и поведение. Например, у элемента `<button>` есть атрибут `disabled`, который делает кнопку неактивной.

Но есть небольшой набор атрибутов, которые есть у всех элементов:

- если необходимо задать пояснение к элементу используется атрибут `title`. Браузеры выводят его при наведении на элемент;
- если необходимо обратиться к элементу в CSS, чтобы задать стили, или в JS, чтобы задать поведение, используется атрибут `id` (identificator). С его помощью задается идентификатор элемента, уникальный для всего документа. Не рекомендуется использовать `id` для добавления стилей к элементу, лучше всего для этого подходит другой атрибут – `class`;
- атрибут `class` позволяет задать класс набору элементов, сгруппировать их и задать общее оформление в CSS и поведение в JS;
- если необходимо задать оформление прямо в html используется атрибут `style`. Не рекомендуется использовать этот атрибут в пользу отдельных CSS-файлов со стилями и атрибута `class`, если это возможно.

1.7 Семантическая разметка

Хорошая разметка должна не только определять структуру, но и делать это осмысленно – закрепляя за элементами определённый смысл: вот здесь рас-

положена навигация, а здесь – адрес. Такая разметка называется семантической (Semantic Markup).

Данная разметка помогает браузерам и поисковым системам лучше ориентироваться в Вашем документе, например, чтобы автоматически сформировать версию для чтения или помочь выделить из документа адрес и телефон, или информацию по фильму, товару, книге. Это очень важно, если Вы хотите, чтобы Ваш документ было легко найти среди миллионов других.

HTML5 предоставляет очень много средств для семантической (осмысленной) разметки. Так, есть элементы:

- `<abbr>` – для аббревиатур и определений;
- `<address>` – для обозначения адреса;
- `<dfn>` – для терминов и определений;
- `<code>` – для кода программ;
- `<nav>` – для навигации.

Семантичных элементов в HTML5 много, но недостаточно для того разнообразия информации, которое сейчас наблюдается. Нет элементов, которые могли бы указать, что здесь описание фильма `<movie>`, а здесь – анкета человека `<person>`.

В 2011 г. по инициативе компаний Bing, Google и Yahoo! был создан проект Schema.org, в рамках которой поисковые системы поддержали ряд дополнительных возможностей для семантической разметки.

Существует менее популярная альтернатива Scheme.org – микроформаты (Microformats). В отличии Scheme.org, она опирается не на атрибуты, а на классы.

Микроформаты, так же как и Scheme.org, широко поддерживаются поисковыми системами, но их меньше и полей в каждой схеме тоже немного.

Рекомендуется использовать Scheme.org, так как она использует только атрибуты, которые логично задают дополнительные свойства элемента. Микроформаты, напротив, используют классы, которые нужны ещё и для визуального оформления, что может приводить к путанице.

1.8 Интернационализация

Так как Интернет – Всемирная паутина, важно помнить, что пользователи разных стран говорят на разных языках, и предусмотреть это в разработке разметки, которая должна поддерживать многоязычие.

В первую очередь в разметке необходимо указать кодировку документа.

Кодировка *(или, вернее, набор символов, character set, charset) – таблица, где каждому символу сопоставляется последовательность нулей и единиц (битов).*

Согласно кодировке устройство преобразует последовательность нулей и единиц в символы, которые видит пользователь. Чтобы браузер правильно интерпретировал HTML-документ, `charset` указывать обязательно.

Иначе вместо:

لغةC++ &lm; هي لغة برمجة تستخدم.

Пользователь увидит нечто такое:

Ù„ØºØ© C++&lm; Ù‡ÙŠ Ù„ØºØ© Ø□Ø±Ù…Ø¬Ø© Ø³ØªØ®Ø®Ø¯Ù….

Рекомендуется использовать кодировку utf-8 (Unicode Transformation Format, 8-bit) – кодировку текста, которая позволяет хранить символы Юникода. Юникод (Unicode) – стандарт кодирования, позволяющий представить знаки почти всех письменных языков.

Существует несколько способов указать `charset`:

1. Используя технический метатег

```
<meta http-equiv="content-type" content="text/html; charset=utf-8">
```

2. Используя специальный метатег (рекомендуемый)

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

Если документ формата XHTML или XHTML5 в начале документа:

```
<?xml version="1.0" encoding="utf-8"?>
```

Кодировка по умолчанию в HTML – ISO 8859-1, XHTML – utf8.

Затем необходимо указать язык документа. Существует несколько способов сделать это:

1. Используя технический метатег

```
<meta http-equiv="Content-Language" content="en-US, ru">
```

2. Используя атрибут `lang` у элемента `<html>` (рекомендуемый)

```
<html lang="ru">
```

Если документ формата XHTML или XHTML5, используя атрибут `xml:lang` у элемента `<html>`:

```
<?xml version="1.0" encoding="utf-8"?>
<html xml:lang="ru">
```

Если в документе используется несколько языков, можно указывать атрибут `lang` для отдельных элементов.

Если необходимо обозначить какой-либо текст непереводимым, можно использовать атрибут `translate`. Он указывает автоматическим переводчикам, нужно ли переводить это текст: `yes`, по умолчанию, или `no`.

Атрибут поддерживается online-переводчиками Microsoft, Google и Яндекс. Он влияет не только на текст внутри элемента, но и на атрибуты `title` и `alt`.

```
<p>Нажмите кнопку <img title="RESUME" translate="no" src=«resume-button.jpg">, чтобы продолжить</p>
```

Всплывающая подсказка при наведении на изображение не будет переведена.

1.9 Тип HTML-документа

С момента появления Интернета прошло немало лет, и за это время в разные годы было создано огромное количество HTML-документов по разным спецификациям: HTML 3.2, HTML 4.0, XHTML 1.0, XHTML 1.1, HTML 5 и др.

Чтобы правильно интерпретировать HTML-документ, браузеру необходимо знать, по какой спецификации автор документа его создавал. Каким образом автор может указать это?

В начале HTML-документа обязательно размещение специального элемента `<!DOCTYPE>`, указывающего на тип документа, иными словами, на спецификацию, по которой он был создан.

Синтаксис у этого элемента следующий:

```
<!DOCTYPE [Корневой элемент] [Публичность] "[Регистрация]//[Организация]//[Тип] [Имя]//[Язык]" "[URL]">
```

Корневой элемент для HTML – это всегда элемент `html`.

Публичность определяет, является ли документ публичным (значение PUBLIC) или системным ресурсом (значение SYSTEM). Для HTML всегда указывается значение PUBLIC.

URL – адрес документа с формальным описанием спецификации. Например, http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd.

Тип – тип документа с формальным описанием спецификации. Для HTML/XHTML значение указывается DTD.

Имя – уникальное имя документа с формальным описанием спецификации. Например, XHTML 1.0 Strict.

Язык – язык документа с формальным описанием спецификации. Например, EN.

Организация – уникальное название организации, разработавшей DTD. Официально HTML/XHTML публикует W3C.

Тип организации: «–» сообщает, что разработчик DTD НЕ зарегистрирован в международной организации по стандартизации (ISO); «+» – зарегистрирован.

Распространенные в настоящее время типы спецификаций представлены в таблице 1.6.

Таблица 1.6 – Типы используемых спецификаций

DOCTYPE	**Описание**
HTML 4.01	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd"> Строгий синтаксис HTML
	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"> Переходный синтаксис HTML
	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd"> В HTML-документе применяются фреймы
XHTML 1.0	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"> Строгий синтаксис XHTML
	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> Переходный синтаксис XHTML
	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd"> Документ написан на XHTML и содержит фреймы

DOCTYPE	Описание
XHTML 1.1	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"> Строгий синтаксис XHTML 1.1.

DTD *(Document Type Definition) – формальная схема спецификации языка, предназначенная для автоматического разбора браузером или программой (например, валидатором документов).*

DTD определяет структуру и синтаксис языка – описывает, какие элементы есть, какие атрибуты и значения у них доступны. Важно заметить, что DTD не описывает семантику, в ней нет никакой информации о том, что означает элемент `<h1>`, как правильно его отображать и чем он отличается от `<h2>`.

Большинство современных документов создаётся по спецификации HTML5. Для него по историческим причинам нет формального описания DTD, поэтому HTML5-документ обозначается очень коротко: `<!DOCTYPE html>`.

Контрольные вопросы по главе 1

1. Чем элемент отличается от тега?
2. На какие группы можно разделить все HTML-элементы? Приведите пример для каждой группы.
3. Какое минимальное количество кода необходимо для создания веб-страницы?
4. Как выглядит процесс отправки данных с формы?
5. Перечислите методы HTTP-протокола.
6. Почему несмотря на ошибку в синтаксисе браузер отобразит представленный ниже список?

```
<p>
  <ul>
      <li>1</li>
      <li>2</li>
  </ul>
</p>
```

2 Photoshop для верстки. Графический контент

Графика – вторая по важности и по значимости после текста составляющая содержания сайтов. Графические материалы можно использовать не только для оформления веб-страницы, но и для представления на ней различного рода визуальной информации.

Графический формат *– это способ записи графической информации. Графические форматы файлов предназначены для хранения изображений, таких как фотографии и рисунки.*

2.1 Векторная графика

Векторная графика *– это изображения, созданные при помощи математических описаний элементарных геометрических объектов, таких как:*

- *точки;*
- *линии и ломаные линии;*
- *многоугольники;*
- *окружности и эллипсы;*
- *сплайны;*
- *кривые Безье;*
- *многоугольник Безье (Безигон);*
- *текст.*

Это далеко не полный список, фактически всё, что можно описать математическими формулами, можно применить для описания векторной графики. Например, для того чтобы построить прямую на экране, нужно всего лишь знать координаты точек начала и конца прямой и цвет, которым ее нужно нарисовать, а для построения окружности – координаты центра, длину радиуса, цвет заливки и, если необходимо, цвет обводки.

Такой подход к описанию графики является одним из преимуществ этого типа. Данный формат позволяет легко масштабировать изображение как в сто-

рону уменьшения, так и в сторону увеличения без потери качества, так как положение и размеры будут пересчитываться. Кроме того, примитивы можно поразному группировать и изменять их форму для создания новых изображений из тех же объектов.

Основные достоинства векторных изображений:

- масштабируемость;
- компактность: размер конечной картинки зависит только от количества и свойств фигур, а не от физического размера изображения;
- изменяемость: параметры объектов, использованных в изображении, могут быть заменены на другие значения.

Одно из основных достоинств – описательное хранение информации, одновременно может стать и отрицательным качеством. Это можно легко отследить на сложных изображениях, с применением большого количества сложных объектов. В этом случае код, описывающий изображение, становится несоразмерно большим. А при масштабировании векторного изображения в браузере происходит пересчет координат, что дает дополнительную нагрузку на систему.

Форматы векторной графики

Самым популярным форматом векторной графики в Вебе на данный момент является SVG (Scalable Vector Graphics).

SVG *– язык разметки масштабируемой векторной графики, предназначенный для описания двумерной векторной и смешанной векторно-растровой графики в формате XML.*

Это открытый стандарт, он рекомендован консорциумом W3C. Отображением (форматированием и декодированием) SVG-элементов можно управлять с помощью таблицы стилей CSS 2.0 и её расширений либо напрямую с помощью атрибутов SVG-элементов.

Достоинством SVG является еще и то, что это по сути текстовый файл, и при наличии определенных навыков возможно редактировать и создавать векторное изображение в обычном текстовом редакторе.

Файл SVG-изображения имеет расширение .svg или .svgz, описывается по всем синтаксическим правилам XML, а именно:

1. Начинается с заголовка (объявления `XML`) и указания `DOCTYPE` (не обязательно), например:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

2. Продолжается объявлением корневого элемента (в данном случае `<svg>`). Этот элемент является непосредственным холстом рисунка, в параметрах задаются пространство имен, ширина, высота документа.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
    <svg xmlns="http://www.w3.org/2000/svg" ver-
sion="1.1" height="400px" width="400px">
    </svg>
```

3. Внутри корневого элемента вставляются элементы (теги), описывающие само изображение. Каждому тегу могут быть присвоены атрибуты. В зависимости от атрибутов тега, фигура, описываемая этим тегом, будет иметь те или иные свойства (положение по оси *X* и *Y*, цвет фона, рамки и т. д.).

В HTML5 была внедрена `inline` поддержка SVG, таким образом, рисунок может быть вставлен на странице как обычный тег (`<svg>...</svg>`).

Любому элементу могут быть присвоены фильтры:

- `translate` – перенести;
- `rotate` – повернуть;
- `scale` – масштабировать;
- `scewX`, `scewY` – исказить;
- `matrix` – смешанная трансформация.

Все эти фильтры описываются в атрибуте `transform`.

Кроме SVG существуют и другие векторные форматы (список). Вот некоторые из них:

- Encapsulated PostScript (EPS) – один из первых векторных форматов, разработанных компанией Adobe;
- CDR – векторное изображение или рисунок, созданный с помощью программы CorelDRAW;
- WMF (англ. Windows MetaFile) – универсальный формат векторных графических файлов для Windows-приложений;

- SWF (ShockWaveFlash) – формат Flash, продукт компании Macromedia, позволяющий разрабатывать интерактивные мультимедийные приложения;
- FLA – внутренний формат программы для создания интерактивной анимации Flash.

2.2 Растровая графика

Растровый формат характеризуется тем, что все изображение по вертикали и горизонтали разбивается на достаточно мелкие прямоугольники – так называемые элементы изображения, или пикселы.

В файле, содержащем растровую графику, хранится информация о цвете каждого пиксела данного изображения. Чем меньше прямоугольники, на которые разбивается изображение, тем больше разрешение, то есть тем более мелкие детали можно закодировать в таком графическом файле. Размер изображения, хранящегося в файле, задается в виде числа пикселов по горизонтали и вертикали.

Растровые изображения обладают множеством характеристик. Размеры изображения и расположение пикселов в нем – это две основные характеристики, которые файл растровых изображений должен сохранить, чтобы создать картинку.

Кроме размера изображения важной является информация о количестве цветов, закодированных в файле.

Глубина цвета определяет то количество оттенков, в диапазоне которых точка может изменять свой цвет. Цвет каждого пиксела кодируется определенным числом бит. В зависимости от того, сколько бит отведено для цвета каждого пиксела, возможно кодирование различного числа цветов. Нетрудно сообразить, что если для кодировки отвести лишь один бит, то каждый пиксел может быть либо белым (значение 1), либо черным (значение 0). Такое изображение называют монохромным.

Далее, если для кодировки отвести четыре бита, то можно закодировать 16 различных цветов, отвечающих комбинациям бит от 0000 до 1111. Если отвести 8 бит – то такой рисунок может содержать 256 различных цветов (от 00000000 до 11111111), 16 бит – 65 536 различных цветов (так называемый High Color). И, наконец, если отвести 24 бита, то рисунок может содержать 16 777 216 различных цветов и оттенков. В последнем случае кодировка называется 24-bit True Color. Но даже если в файле и отводится 24 бита на каждый

пиксел, это еще не означает, что Вы действительно сможете насладиться такой богатой палитрой – ведь технические возможности мониторов ограничены.

Плюсы растрового изображения:

- высокая реалистичность изображения;
- растровая графика позволяет воспроизвести изображение любой сложности.

Недостатки растровых изображений:

- растровые изображения плохо масштабируются, можно уменьшить изображение, однако увеличить его без потери качества невозможно (к потере качества относится заметное увеличение размытия изображения после увеличения рисунка);
- исходное растровое изображение можно редактировать только целиком (в отличие от векторного его нельзя разбить на части);
- конечный размер картинки зависит от количества пикселов, т. е. от разрешения изображения. Поэтому зачастую файл с растровым изображением имеет больший размер по сравнению с векторным.

Форматы растровой графики

BMP (от англ. Bitmap Picture) – формат, который изначально мог хранить только аппаратно-зависимые растры (англ. Device Dependent Bitmap, DDB), но с развитием технологий отображения графических данных формат BMP стал преимущественно хранить аппаратно-независимые растры.

С форматом BMP работает огромное количество программ, так как его поддержка интегрирована в операционные системы Windows и OS/2. Файлы формата BMP могут иметь расширения .bmp, .dib и .rle.

В формате BMP можно сохранять черно-белые, серые полутоновые, индексные цветные и цветные изображения системы RGB (но не двухцветные или цветные изображения системы CMYK). Недостаток этих графических форматов: большой объем. Следствие – малая пригодность для использования в Вебе.

GIF (англ. Graphics Interchange Format – формат для обмена изображениями) способен хранить сжатые данные без потери качества в формате не более 256 цветов. Независящий от аппаратного обеспечения формат GIF был разработан в 1987 г. (GIF87a) фирмой CompuServe для передачи растровых изображений по сетям. GIF использует LZW-компрессию, что позволяет неплохо сжимать файлы, в которых много однородных заливок (логотипы, надписи, схемы). Стандарт разрабатывался только для поддержки 256-цветовой палитры. Один

из цветов в палитре может быть объявлен «прозрачным». В этом случае в программах, которые поддерживают прозрачность GIF (например, большинство современных браузеров) сквозь пикселы, окрашенные «прозрачным» цветом, будет виден фон.

Формат GIF допускает чересстрочное хранение данных (Interlaced GIF). При этом строки разбиваются на группы, и меняется порядок хранения строк в файле. При загрузке изображение проявляется постепенно, в несколько проходов. Благодаря этому, имея только часть файла, можно увидеть изображение целиком, но с меньшим разрешением.

Формат GIF поддерживает анимационные изображения. Фрагменты представляют собой последовательности нескольких статичных кадров, а также информацию о том, в течение какого времени каждый кадр будет показан на экране.

TIFF (Tagged Image File Format) создан объединенными силами Aldus, Microsoft и Next специально для хранения сканированных изображений. Исключительная гибкость формата сделала его действительно универсальным. TIFF – один из самых древних форматов в мире микрокомпьютеров, на сегодняшний день он является самым гибким, универсальным и активно развивающимся. В нем можно хранить графику в любом режиме: от битового и индексированных цветов до Lab, CMYK и RGB (кроме дуплексов и многоканальных документов).

Версии формата существуют на всех компьютерных платформах, что делает его исключительно удобным для переноса растровых изображений между ними. TIFF поддерживает монохромные, индексированные, полутоновые и полноцветные изображения в моделях RGB и CMYK с 8- и 16-битными каналами.

Большим достоинством формата остается поддержка практически любого алгоритма сжатия. Наиболее распространенным является сжатие без потерь информации по алгоритму LZW (Lempel Ziv Welch), обеспечивающему очень высокую степень компрессии.

JPEG (произносится «джейпег», от англ. Joint Photographic Experts Group, по названию организации-разработчика) – один из популярных графических форматов, созданный для хранения фотографий. Файлы, содержащие данные JPEG, обычно имеют расширения .jpeg, .jfif, .jpg, .JPG или .JPE, самое популярное – .jpg.

Алгоритм JPEG является алгоритмом сжатия данных с потерями, и в наибольшей степени пригоден для сжатия фотографий и картин, содержащих реа-

листичные сцены с плавными переходами яркости и цвета. Наибольшее распространение JPEG получил в цифровой фотографии и для хранения и передачи изображений по Сети.

Уменьшение размера файла достигается сложным математическим алгоритмом удаления информации – чем заказываемое качество ниже, тем коэффициент сжатия больше, файл меньше. Главное – подобрать максимальное сжатие при минимальной потере качества. Происходят идентификация и отбрасывание данных, которые человеческий глаз не в состоянии увидеть (незначительные изменения в цвете не различаются человеком, тогда как улавливается даже малейшая разница в интенсивности, поэтому JPEG меньше подходит для обработки черно-белых полутоновых изображений), что приводит к существенному уменьшению размера файла. В результате применения технологии JPEG данные теряются навсегда. Так, файл, однажды записанный в формате JPEG, а затем переведенный, скажем, в TIFF, уже не будет таким же, как оригинал.

С другой стороны, JPEG малопригоден для сжатия чертежей, текстовой и знаковой графики, где резкий контраст между соседними пикселами приводит к появлению заметных артефактов. Такие изображения целесообразно сохранять в форматах без потерь, таких как TIFF, GIF, PNG или RAW. JPEG (как и другие методы искажающего сжатия) не подходит для сжатия изображений при многоступенчатой обработке, так как искажения в изображения будут вноситься каждый раз при сохранении промежуточных результатов обработки (см. рис. 2.1).

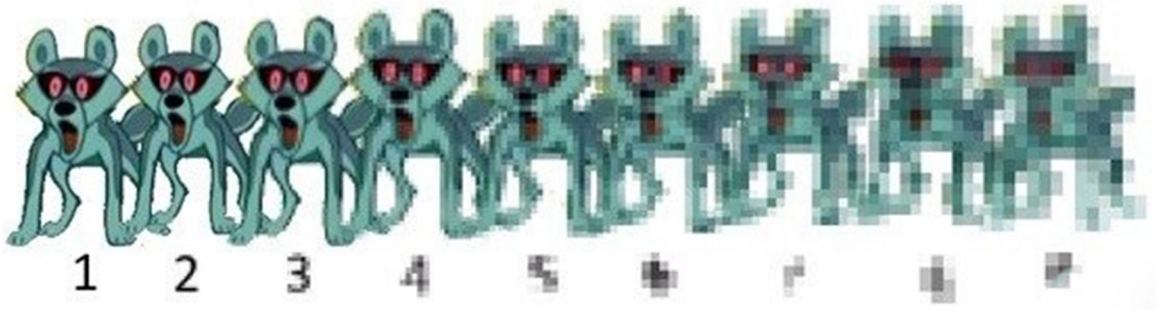


Рис. 2.1 – JPEG, сохраненный несколько раз

JPEG не должен использоваться и в тех случаях, когда недопустимы даже минимальные потери, например, при сжатии астрономических или медицинских изображений. В таких случаях может быть рекомендован предусмотренный стандартом JPEG режим сжатия Lossless JPEG или стандарт сжатия JPEG-LS.

Недостатки:

- появление на изображениях при высоких степенях сжатия характерных артефактов: изображение рассыпается на блоки размером 8×8 пикселов (этот эффект особенно заметен на областях изображения с плавными изменениями яркости), в областях с высокой пространственной частотой (например, на контрастных контурах и границах изображения) возникают артефакты в виде шумовых ореолов;
- форматом не поддерживаются анимация и прозрачный цвет.

Однако, несмотря на недостатки, JPEG получил очень широкое распространение из-за достаточно высокой (относительно существовавших во время его появления альтернатив) степени сжатия, поддержки сжатия полноцветных изображений и относительно невысокой вычислительной сложности.

Формат JPEG может хранить изображения с глубиной цвета 24 бит/пиксел. Такой глубины цвета достаточно для практически точного воспроизведения изображений любой сложности.

Формат предназначен для представления сложных фотоизображений. Разновидность Progressive JPEG позволяет сохранять изображения с выводом за указанное количество шагов (от 3 до 5 в Photoshop) – сначала с маленьким разрешением (плохим качеством), на следующих этапах первичное изображение перерисовывается все более качественной картинкой.

PNG (англ. portable network graphics) – растровый формат хранения графической информации, использующий сжатие без потерь по алгоритму Deflate. Файлы формата PNG имеют расширение .PNG (.png). Формат поддерживает прозрачность и позволяет выбирать палитру сохранения – серые полутона, 256 цветов, true color. Сжатие производится без потери качества по горизонтали и вертикали, используя собственный алгоритм, параметры которого не настраиваются. Анимация не поддерживается.

Формат PNG спроектирован для замены устаревшего и более простого формата GIF, а также, в некоторой степени, для замены значительно более сложного формата TIFF и позиционируется в первую очередь как формат для использования в веб-среде и редактирования изображений.

Формат *RAW* разработан для цифровых фотоаппаратов. Это точная копия картинки, запечатленной на матрице во время съемки, представляет собой три фотографии, снятые в красных, синих и зеленых цветах. Расширения RAW-файлов у разных производителей могут отличаться, и их далеко не всегда получается открыть с помощью программ для обработки изображений.

2.3 Цветовые модели

Цветовая модель *– термин, обозначающий абстрактную модель описания представления цветов в виде кортежей чисел, обычно из трёх или четырёх значений, называемых цветовыми компонентами или цветовыми координатами. Вместе с методом интерпретации этих данных множество цветов цветовой модели определяет цветовое пространство.*

Модель RGB

RGB-модель – наиболее распространенный способ кодирования цвета. При этом способе кодирования любой цвет представляется в виде комбинации трех цветов (каналов): красного (Red), зеленого (Green) и синего (Blue), взятых с разной интенсивностью.

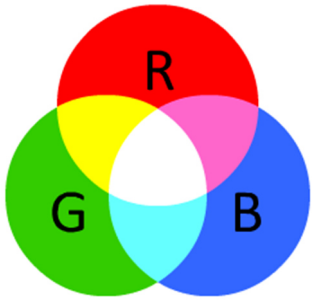


Рис. 2.2 – Модель RGB

Интенсивность каждого из трех цветов – это один байт (т. е. число в диапазоне от 0 до 255). Белый цвет можно записать в следующем виде: R – 255, G – 255, B – 255; чёрный: R – 0, G – 0, B – 0; красный: R – 255, G – 0, B – 0; желтый: R – 255, G – 255, B – 0.

Также допустимо задавать цвет в процентном отношении, при этом 100% будет соответствовать числу 255: R – 100%, G – 50%, B – 50%.

Кроме того, RGB хорошо представляется двумя шестнадцатеричными цифрами (числом от 00 до FF). Таким образом, цвет удобно записывать тремя парами цифр в системе счисления с основанием 16:

белый – #ffffff

красный – #ff0000

черный – #00000

желтый – #ffff00

Такой формат записи называется HEX. Если оба числа каждый пары одинаковы (например, в первой паре ff), то допускается использовать в hex-записи по одному числу вместо пары. Например, #ff00cc можно записать, как #f0c.

Функциональные нотации rgb() и rgba() позволяют указывать RGB-палитру, используя десятичные значения. rgb() принимает на вход указанные через запятую степени заполнения каждого из каналов. Например, для красного цвета: color: rgb(255, 0, 0).

rgba() отличает от rgb() возможность указания альфа-канала (степень прозрачности цвета). Альфа-канал указывается в виде десятичной дроби в диапазоне от 0 до 1, где 0 соответствует полной прозрачности, а 1 – непрозрачности: rgb(0, 0, 0, .8).

sRGB является стандартом представления цветового спектра с использованием модели RGB. sRGB создан совместно компаниями HP и Microsoft в 1996 г. для унификации использования модели RGB в мониторах, принтерах и интернет-сайтах.

Модель CMYK

В отличие от модели RGB, модель CMY описывает цвета, полученные в результате отражения света объектами, то есть она полностью противоположна модели RGB. Данная модель является субтрактивной (вычитающей), поскольку цвета в ней образуются путем вычитания из черного цвета базовых цветов: голубого (Cyan), пурпурного (Magenta), желтого (Yellow).

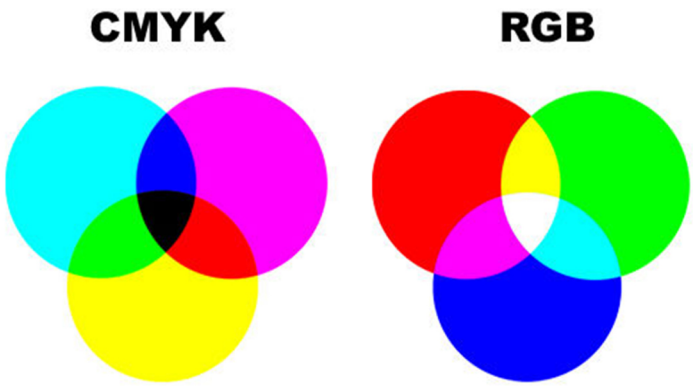


Рис. 2.3 – Модель представления CMYK в сравнении с RGB

Эти цвета образуют так называемую полиграфическую триаду и называются триадными. В цветовой модели CMY уровень составляющих задается значениями в диапазоне от 0 до 100 (величина 100 в этой модели соответствует

255 единицам модели RGB), где эти числа называются «частями» или «пропорциями». Поскольку цветовая модель CMY является обратной модели RGB, то при смешивании двух субтрактивных цветов результирующий цвет оказывается более темным, чем исходные, а при смешивании всех трех составляющих должен получаться черный цвет. Соответственно белый цвет – это полное отсутствие краски (значения всех цветовых составляющих равны 0).

Может показаться, что такая модель больше всего подходит для печатной продукции – ведь мы видим цвет, отраженный от поверхности. Однако модель CMY не годится для печатных процессов, так как на практике ничего идеального не бывает. Теоретически смесь трех базовых красок должна давать глубокий черный цвет, но в реальности так не получается, поскольку при смешивании трех данных красок образуется не черный, а грязно-коричневый цвет. Для устранения этого недостатка к трем краскам добавили четвертую, черную (Black), и цветовая модель получила название CMYK – Cyan, Magenta, Yellow, BlacK. В слове Black используется не первая буква, а последняя, чтобы не путать с цветом Blue модели RGB. Таким образом, черный цвет в модели CMYK образуется с помощью только одной составляющей – черной (0,0,0,100), хотя иногда применяется и более глубокий черный. Область применения цветовой модели CMYK – полноцветная печать. Именно с этой моделью работает большинство устройств печати.

Модели HSL и HSV

HSV (или HSB) означает Hue, Saturation, Value (еще может именоваться Brightness), где:

- Hue – цветовой тон, т. е. оттенок цвета;
- Saturation – насыщенность. Чем выше этот параметр, тем «чище» будет цвет, а чем ниже, тем ближе он будет к серому;
- Value (Brightness) – значение (яркость) цвета. Чем выше значение, тем ярче будет цвет, а чем ниже, тем темнее (0% – черный).

HSL – Hue, Saturation, Lightness:

- Hue – цветовой тон, т. е. оттенок цвета;
- Saturation – насыщенность. Аналогично с HSV;
- Lightness – это светлота цвета (не путать с яркостью). Чем выше параметр, тем светлее цвет (100% – белый), а чем ниже, тем темнее (0% – черный).

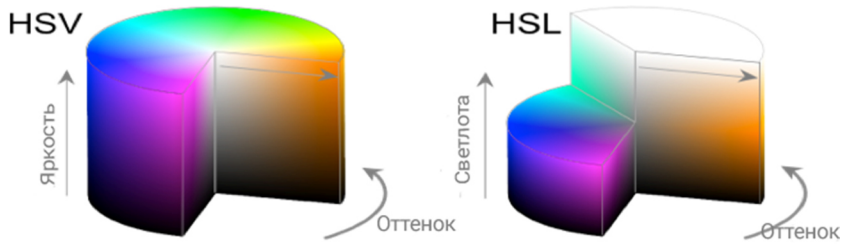


Рис. 2.4 – Трехмерная реализация HSL- (слева) и HSV- (справа) моделей

Более распространенная модель HSV, она часто используется вместе с моделью RGB, где HSV показана в визуальном виде, а числовые значения задаются в RGB.

Такая модель чаще всего используется в простой обработке изображений, т. к. при помощи неё удобно регулировать основные параметры фотографий, не прибегая к куче различных фильтров или отдельных настроек. Например, в Photoshop присутствуют обе модели, только одна из них находится в редакторе выбора цвета, а другая – в окне настроек Hue/Saturation.

Существует возможность задать цвет HSL в CSS:

```
body {
    color: hsl(120,100%,25%); /* зеленый */
}
```

Формат HSLA похож по синтаксису на HSL, но включает в себя альфа-канал, задающий прозрачность элемента. Значение 0 соответствует полной прозрачности, 1 – непрозрачности, а промежуточное значение вроде 0.5 – полупрозрачности.

Недостаток HSB-модели в том, что она также зависит от аппаратной части. Она просто не соответствуют восприятию человеческого глаза, т. к. он воспринимает цвета с разной яркостью (например, синий воспринимается более темным, чем красный), а в этой модели у всех цветов одинаковая яркость. У HSL аналогичные недостатки. Во избежание таких недостатков была создана новая модель, названная Lab – именно она используется по умолчанию в программе Adobe Photoshop и, как правило, используется для правильной цветопередачи в изображениях. В веб-разработке данная цветовая модель не используется.

2.4 Архивация и компрессия

Архивация, или сжатие графических данных («сжатие без потерь»), возможна как для растровой, так и для векторной графики. При этом способе сжатия программа анализирует наличие в архивируемых данных некоторых одина-

ковых последовательностей и исключает их, записывая вместо повторяющегося фрагмента ссылку на предыдущий такой же (для последующего восстановления). Такими одинаковыми последовательностями могут быть пикселы одного цвета, повторяющиеся текстовые данные или некая избыточная информация, которая в рамках данного массива данных повторяется несколько раз. Например, растровый файл, состоящий из подложки строго одного цвета (например, серого), имеет в своей структуре очень много повторяющихся фрагментов.

Компрессия (конвертирование) данных («сжатие с потерями») – это способ сохранения данных таким образом, при использовании которого не гарантируется (хотя иногда возможно) полное восстановление исходных графических данных. При таком способе хранения данных обычно графическая информация немного «портится» по сравнению с оригинальной, но этими искажениями можно управлять, и при их небольшом значении ими вполне можно пренебречь. Обычно файлы, сохраненные с использованием этого способа хранения, занимают значительно меньше дискового пространства, чем файлы, сохраненные с использованием простого сжатия. Как правило, при сохранении данных с использованием компрессии имеется возможность компромисса между размером выходного файла и его качеством. Понятно, что возможна оптимизация только по одному параметру (чем меньше качество, тем меньше объем выходного файла, и наоборот).

2.4.1 Алгоритмы сжатия без потерь

Метод RLE

RLE (run-length encoding) – метод сжатия данных, при котором одинаковые последовательности одних и тех же байт заменяются однократным упоминанием повторяющегося байта (или целой цепочки байтов) и числа его повторений в исходных данных.

Например, строка типа 0100 0100 0100 0100 0100 0100 0100 0100, описывающая некую группу пикселов, будет заменена на запись типа 0100 x 8, и т. д.

Применяется этот тип сжатия в тех случаях, когда изображение имеет большие участки одинакового цвета, цифровое представление которых идентично.

В основном, этот тип сжатия применим для монохромных изображений, сохраненных в цветовой модели Bitmap, где при сжатии данных с его использованием можно добиться наилучших результатов.

Для сжатия других типов данных (в том числе и не графических) алгоритм применим, но малоэффективен, так как сжимаемые данные должны иметь простую повторяющуюся структуру).

Этот алгоритм имеет еще одно важное преимущество, заключающееся в его относительной простоте, что позволяет быстро производить распаковку из этого формата и упаковку в этот формат.

Этот метод сжатия графических данных используется для файлов форматов PSD, BMP и др.

Метод CCITT

CCITT Group 3, CCITT Group 4 – два похожих метода сжатия графических данных, работающие с однобитными изображениями, сохраненными в цветовой модели Bitmap. Основаны на поиске и исключении из исходного изображения дублирующихся последовательностей данных (как и в типе сжатия RLE). Различием является лишь то, что эти алгоритмы ориентированы на упаковку именно растровой графической информации, так как работают с отдельными рядами пикселов в изображении.

Этот алгоритм подходит для сжатия изображений с большими одноцветными областями. Его достоинством является скорость выполнения, а недостатком – ограниченность применения для компрессии графических данных (не все данные удается таким образом эффективно сжать).

Этот метод сжатия графических данных используется в файлах формата PDF, PostScript (в инкапсулированных объектах) и др.

Метод LZW

LZW (Lemple – Zif – Welch) – алгоритм сжатия данных, основанный на поиске и замене в исходном файле одинаковых последовательностей данных для их исключения и уменьшения размера «архива». Относится к формату сжатия без потерь.

Данный тип сжатия подходит для обработки растровых данных любого типа – монохромных, черно-белых или полноцветных.

Сжимает данные путём поиска одинаковых последовательностей (они называются фразы) во всем файле. Выявленные последовательности сохраняются в таблице, им присваиваются более короткие маркеры (ключи). Так, если в изображении имеются наборы из розового, оранжевого и зелёного пикселов, повторяющиеся 50 раз, LZW выявляет это, присваивает данному набору от-

дельное число (например, 7) и затем сохраняет эти данные 50 раз в виде числа 7. Метод LZW, так же, как и RLE, лучше действует на участках однородных, свободных от шума цветов, он действует гораздо лучше, чем RLE, при сжатии произвольных графических данных, но процесс кодирования и распаковки происходит медленнее.

Этот метод сжатия графических данных используется в файлах формата TIFF, PDF, GIF, PostScript (в инкапсулированных объектах) и др.

Метод ZIP

ZIP – метод сжатия данных, аналогичный методу, использованному в популярном алгоритме архивации PKZip.

В основу метода сжатия положен метод, аналогичный LZW. Как и метод сжатия данных LZW, этот способ не вносит искажений в исходный файл (сжатие без потерь) и лучше всего подходит для обработки графических данных с одинаковыми одноцветными или повторяющимися областями.

Этот метод сжатия графических данных используется в файлах формата PDF, TIFF и некоторых других.

2.4.2 Алгоритмы сжатия с потерями

Метод JPEG

JPEG (Joint Photographic Experts Group) – метод, используемый для хранения полутоновых и полноцветных изображений, позволяющий добиться наивысшей степени сжатия и минимального размера выходного файла. Алгоритм основан на особенностях восприятия человеческим глазом различных цветов и занимает много процессорного времени.

Происходит кодирование файла в несколько этапов:

- изображение условно разбивается на несколько цветовых каналов;
- изображение разбивается на группы, по 64 пиксела в каждой группе, которые представляют из себя квадратные участки изображения размером 8×8 пикселов;
- цвет пикселов специальным образом кодируется, исключается дублирующая и избыточная информация, причем при описании цвета большее внимание уделяется скорее яркостной, чем цветовой составляющей, так как человеческий глаз воспринимает больше изменения яркости, чем конкретного цветового тона;

- полученные данные сжимаются по RLE- или LZW-алгоритму, для получения еще большей компрессии.

В результате на выходе формируется файл, иногда в десятки раз меньший, чем его неконвертированный аналог. Однако чем меньше размер выходного файла, тем меньше степень «аккуратности» при работе программы-конвертора и, соответственно, ниже качество выходного изображения.

Обычно в программах, позволяющих сохранять растровые данные, возможно задание некоего компромисса между объемом выходного файла и качеством изображения. При наивысшем качестве объем выходного файла в 3–5 раз меньше исходного незапакованного. При наименьшем – меньше исходника в десятки раз, но, как правило, при этом качество изображения не позволяет его где-либо использовать.

Графический формат JPEG 2000

JPEG 2000 (или jp2) – графический формат, который вместо дискретного косинусного преобразования, характерного для JPEG, использует технологию интегрального вейвлет-преобразования, основывающуюся на представлении сигнала в виде суперпозиции некоторых базовых функций – волновых пакетов. В результате такой компрессии изображение получается более гладким и чётким, а размер файла по сравнению с JPEG при одинаковом качестве уменьшается ещё на 30%.

JPEG 2000 полностью свободен от главного недостатка своего предшественника: благодаря использованию вейвлетов изображения в этом формате не содержат знаменитой «решётки» из блоков по 8 пикселов. Новый формат, так же как и JPEG, поддерживает так называемое прогрессивное сжатие, позволяющее по мере загрузки видеть сначала размытое, но затем всё более чёткое изображение.

Большая степень сжатия: артефакты незначительны. Большая степень сжатия достигается благодаря использованию дискретного вейвлет-преобразования и более сложного энтропийного кодирования.

Возможность последовательной сборки: JPEG 2000 обеспечивает возможность последовательного декодирования и вывода изображения сверху вниз без необходимости буферизации всего изображения.

Гибкий формат файла: форматы файлов JP2 и JPX обеспечивают хранение информации о цветовых пространствах, метаданных и информации для со-

гласованного доступа в сетевых приложениях, взаимодействующих с помощью протокола JPEG Part 9 JPIP.

Оптимизация

Уменьшения размера некоторых векторных изображений можно достичь за счет сокращения кода, с помощью которого это изображения задается.

Рассмотрим на примере файла формата SVG. На рисунке 2.5 записан логотип компании «Паравеб». Это изображение было экспортировано из программы Avocode.

Рис. 2.5 – Логотип, записанный в формате SVG

Принцип работы инструментов оптимизации заключается в удалении «ненужной» информации из SVG, тем самым уменьшается размер самого файла. Для примера, мы можем уменьшить файл SVG, приведенный выше, используя инструмент svgo. В результате визуально запись изображения сокращается более чем в 2 раза (рис. 2.6).

Что касается размера файла, то согласно результатам работы программы svgo, он уменьшился на 64%: с 8 до 3 Кб.

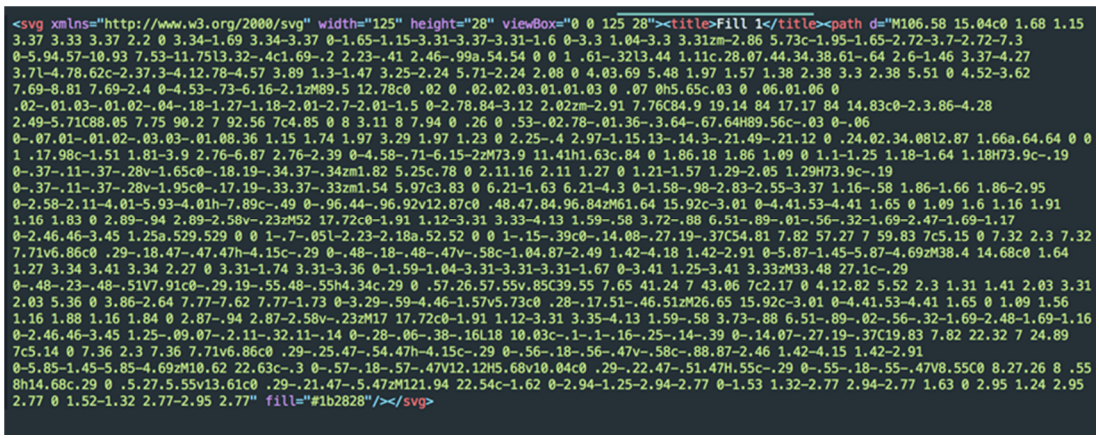


Рис. 2.6 – Запись логотипа после применения инструмента оптимизации

Кроме того, поскольку SVG – это формат на основе XML, мы можем применить сжатие GZIP для уменьшения его размера при передаче.

Оптимизация растровых изображений происходит с применением различных методов сжатия. Также уменьшить размер растрового изображения можно в момент его сохранения, путем удаления лишней метаинформации.

2.5 Графические редакторы. Photoshop

Графический редактор *– это приложение, созданное для работы с графикой. К ним относятся:*

- *Adobe Photoshop;*
- *GIMP;*
- *Krita;*
- *Sketch и др.*

Adobe Photoshop – один из наиболее популярных графических редакторов для создания дизайн-макета для веб-страницы. Он разработан фирмой Adobe Systems. Макеты, созданные в Photoshop, имеют расширение *.psd.

В данном параграфе не рассматривается полный цикл создания дизайн-макета для сайта или веб-страницы, представлен лишь алгоритм работы с уже готовым макетом.

Первое, на что следует обратить внимание после открытия макета, это направляющие (extras). Они всегда присутствуют в сложном макете.

Зачастую они представляют собой сетку и используются для выравнивания элементов макета. Чтобы посмотреть на макет, отключите их в меню приложения → View → Extras.

Под меню приложения длинная горизонтальная область – в ней отображаются настройки активного инструмента.

Внешний вид редактора также настраивается: можно перетаскивать панели, включать и отключать их видимость (в пункте меню приложения Window), сворачивать и разворачивать (двойной клик по названию панели), сворачивать в иконки.

Необходимые и желательные для разработчика панели:

- Layers (панель слоёв) папки и слои макета;
- Character – данные о выделенном текстовом слое или тексте (шрифт, цвет, размер, интерлиньяж и др.);
- Info (панель информации) – данные о цвете, положении курсоров по координатам и при выделении, размере выделенного фрагмента;
- History – панель истории.

Рабочие панели

В панели Info для того, чтобы цвета отображались в привычном для нас виде HEX, для цвета выбираем WebColor (рис. 2.7).

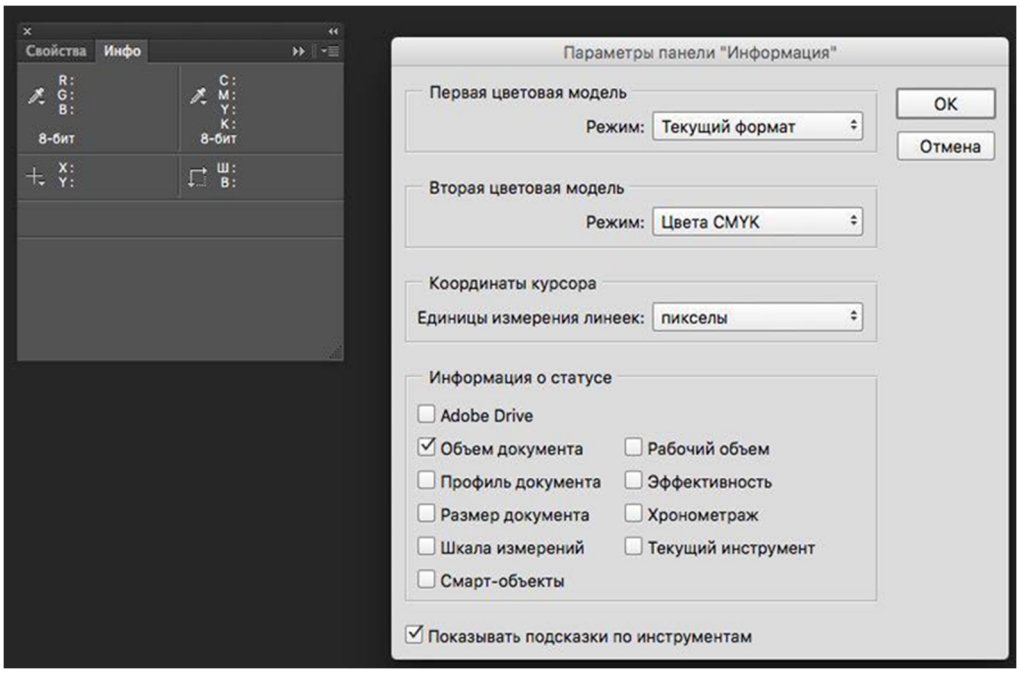


Рис. 2.7 – Параметры панели «Информация»

В панели слоев устанавливается настройка, которая позволяет видеть миниатюру (preview, превью) для каждого слоя. В «Параметрах панели» выбирается превью побольше (если нужно), «Контент миниатюр» устанавливается в

положение «Границы слоев», это позволит видеть на миниатюре только содержимое слоя (рис. 2.8).

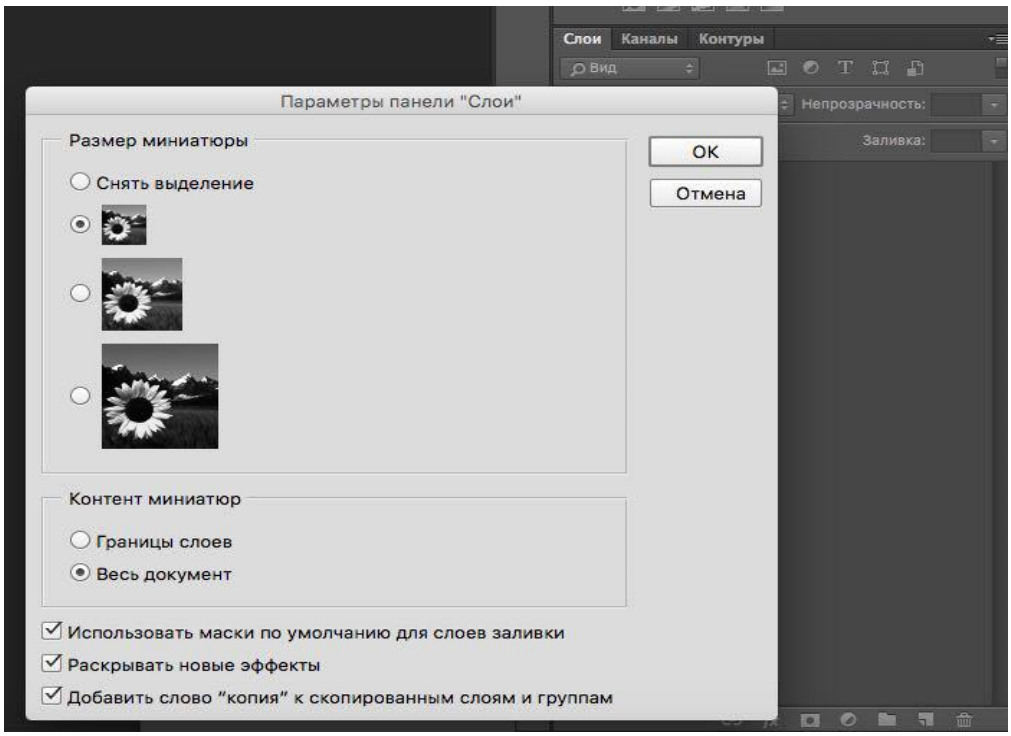


Рис. 2.8 – Параметры панели «Слои»

В Photoshop есть три основных способа организации макета:

1. Использование слоев. Слои – это графическая аналогия наложения блоков. Каждый слой содержит в себе часть общего изображения. Слои удобно использовать для редактирования макета по частям.
2. Использование групп. Слои можно объединять в группы (папки). Также можно использовать цветовое кодирование для групп: для пометки принадлежности группы к той или иной странице (если в макете несколько страниц).
3. Использование окна «Композиции слоев». Этот инструмент применяется для запоминания состояний. Когда в одном макете находится несколько страниц, можно запомнить состояние включенных/выключенных слоев для каждой страницы. И потом переключаться только по сохраненным состояниям. Чтобы включить панель, воспользуйтесь меню: «Окно» → «Композиции слоев» (рис. 2.9).

Можно поискать нужный слой в панели слоёв или зажать ⌘/Ctrl и кликнуть на слой. Зажатие кнопки временно активирует инструмент «Перемещение», который работает в случае выбора любого инструмента кроме инстру-

мента «Рука» (по зажатию Ctrl включается инструмент масштабирования) и самого инструмента «Перемещения». Чтобы это работало, необходимо убедиться, что настройки инструмента «Перемещения» (это панель под меню приложения, когда инструмент выбран) выставлены следующие: выбран «Автовыбор» и в выпадающем списке рядом – «Слой» (рис. 2.10).

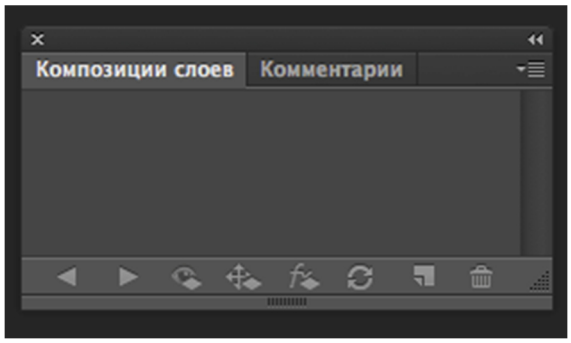


Рис. 2.9 – Панель композиции слоев

Рис. 2.10 – Настройки инструмента «Перемещение»

Показать или скрыть какие-либо слои просто – необходимо кликнуть на иконке «глаз» этого слоя в панели слоёв (рис. 2.11).

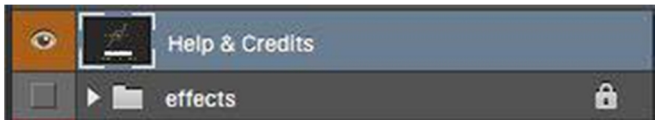


Рис. 2.11 – Панель слоев со скрытым слоем

Клик по иконке «глаз» в панели слоёв с зажатой кнопкой Alt позволяет показывать только один этот слой, прочие будут скрыты, повторный клик позволяет снова показать все остальные слои.

Двойной клик по миниатюре текстового слоя позволяет редактировать слой; выставление текстового курсора в нужное место позволяет узнать шрифт, размер, интерлиньяж, трансформации, кернинг, спейсинг и цвет.

Если параметры «Горизонтальное масштабирование» или «Вертикальное масштабирование» отличаются от 100%, нужно экспериментировать с CSS3-свойством transform у блока, в который включать только этот текст, и налаживать взаимодействие дизайнера и верстальщика, если это контентный текст.

Двойной клик по миниатюре слоя с цветом, градиентом, заливкой текстурой позволяет вызвать модальное окно с данными слоя (рис. 2.12).

Если у слоя справа есть курсивная надпись «fx» (и иконка, открывающая список), значит у него есть эффекты. При клике на открывающую иконку можно увидеть список эффектов (можно отключить их показ с помощью клика на иконки глаза рядом с эффектами).

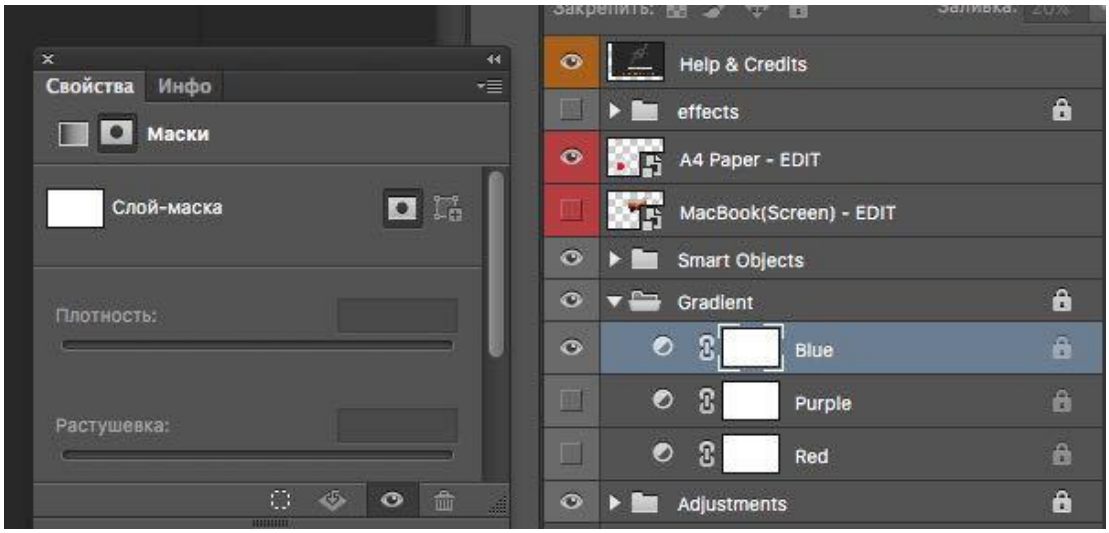


Рис. 2.12 – Модальное окно с данными слоя

Небольшое отступление: в верхней левой части панели слоёв есть выпадающий список – это режим наложения слоя. Если его значение отличается от режима «Обычные», то необходимо обратиться к дизайнеру, так как слои, отличающееся от режима «Обычные» и не являющиеся частью сложных коллажей/картинок (целиком сохраняются в единое изображение), нереально корректно сверстать (как в макете), в любом случае это требует дополнительных трудозатрат с стороны исполнителя (рис. 2.13).

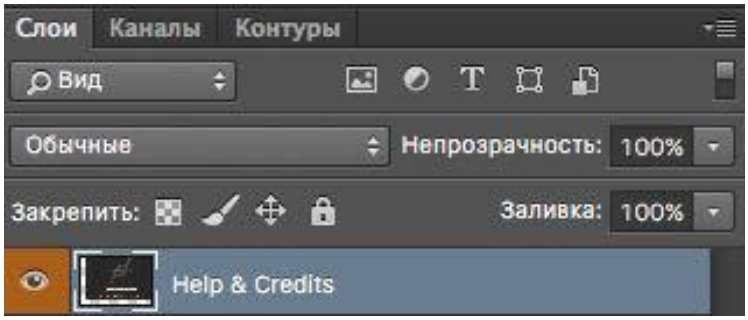


Рис. 2.13 – Режим наложения слоя не должен быть никаким другим, кроме режима «Обычные»

Цвет в макете настраивается при помощи инструмента «Пипетка» (в настройках необходимо указать «Размер образца» → «Точка»). При клике по произвольному пикселу в панели цвета (под всеми инструментами) появляется цвет пиксела (рис. 2.14).

Для определения размера в макете используется инструмент «Линейка» – необходимо нажать на него, тащить мышью, отпустить – в панели настроек (под меню приложения, параметр L1) появляется измеренное расстояние. Если тащить с зажатым Shift, измеритель перемещается строго горизонтально, строго вертикально или под углом 45° (рис. 2.15).

Рис. 2.14 – Инструмент «Пипетка»

Рис. 2.15 – Инструмент «Линейка»

Можно использовать инструмент «Прямоугольная область», создавая выделение (размер выделения будет показан рядом с выделением и в панели Info). Убрать получившееся после измерения выделение можно с помощью ⌘/Ctrl + + D.

Для кадрирования используется инструмент «Прямоугольное выделение», он выбирается из меню приложения «Изображение» → «Кадрировать». Для сохранения воспользуйтесь меню «Файл» → «Сохранить»/«Сохранить как…».

Для сохранения графики для Веба в Photoshop правильным будет использовать специально созданный для этого инструмент – «Сохранить для Web» (через меню «Файл» → «Сохранить для Web...»).

В левом верхнем углу области предварительного просмотра представлена серия из четырех вкладок. По умолчанию выбрана вкладка «Оптимизация», исходное изображение не видно (рис. 2.16). Вместо этого показан предварительный просмотр того, как изображение выглядит с текущими настройками оптимизации.

Первое, что нужно сделать, – выбрать правильный формат файла для изображения. Большие фотографические (многоцветные) изображения лучше

сохранять в JPEG. Любые картинки, где нужны полупрозрачность или отсутствие искажений, следует сохранять в формате PNG-24.

Если отключить метаданные (Metadata), это позволит уменьшить размер изображения, а убедиться в том, что картинка сохраняется в цветовом пространстве sRGB позволит включение опции «Преобразовать в sRGB».

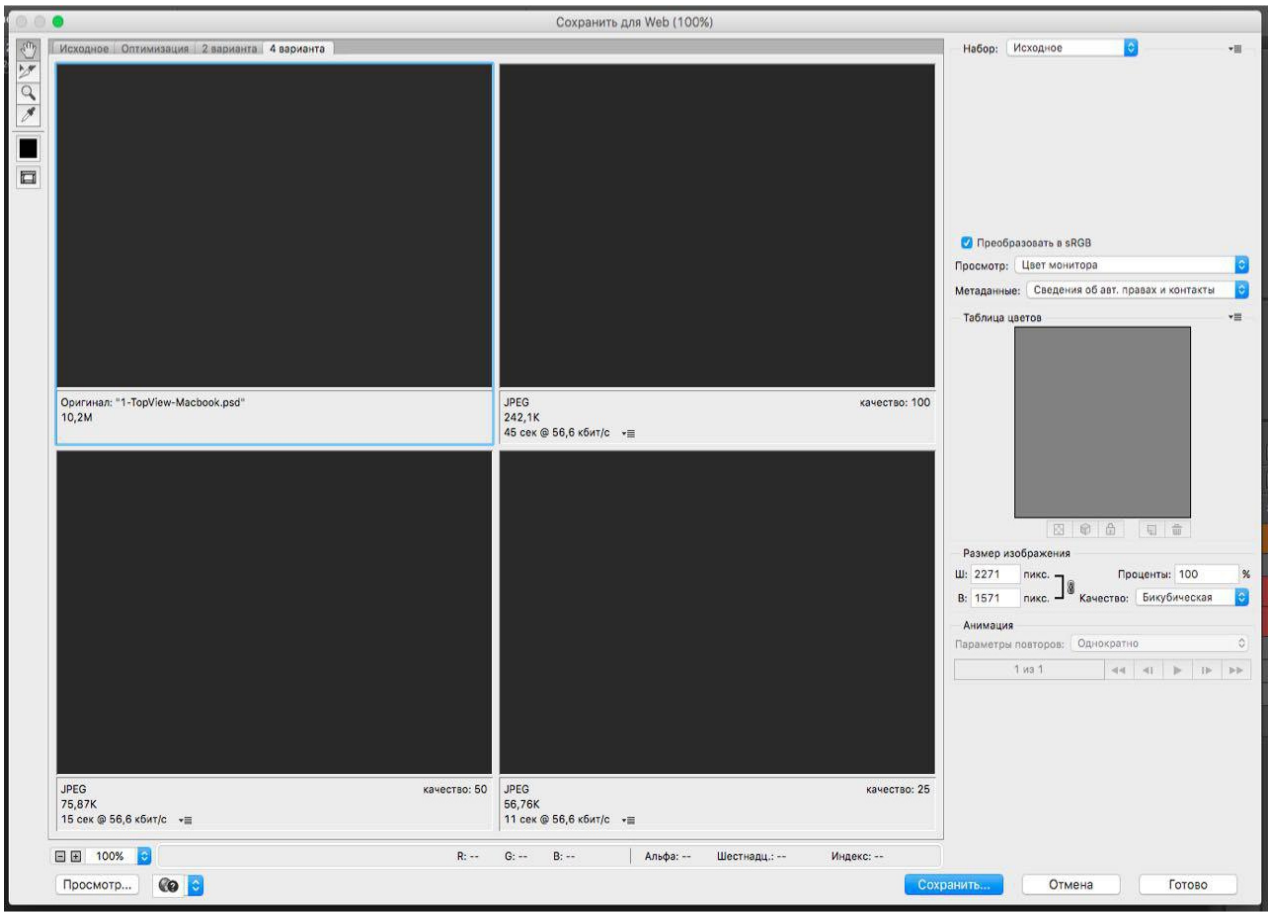


Рис. 2.16 – Окно «Сохранить для Web»

При работе с макетом необходимо подумать:

- о структуре будущей верстки;
- о деградации (для более старых браузеров);
- о шрифтах, используемых в макете;
- о реальных данных (если в макете использована «рыба» – наброски макета).

В идеале, если наложить «картинку» сверстанной HTML-страницы на «картинку» оригинального PSD-макета, то обе должны совпасть. Совместиться должны все элементы картинок – текст, изображения, графические элементы. Такая техника верстки имеет специальное название: Pixel Perfect. Реализация этой техники осуществляется при помощи соответствующих плагинов под

браузеры или специализированных скриптов. Для браузера Firefox имеется плагин Pixel Perfect для одноименной проверки сверстанной страницы.

Один из способов оптимизации изображений – это использование спрайтов. Это прием, позволяющий объединить много изображений в одно, что помогает:

- сократить количество обращений к серверу;
- загрузить несколько изображений сразу, включая те, которые понадобятся в будущем;
- если у изображений сходная палитра, объединённое изображение будет меньше по размеру, чем совокупность исходных картинок.

При этом изображение вставляется на страницу через css-свойство background, а не через элемент img.

Суть CSS-спрайта в том, что задается элемент фиксированного размера и сдвигается его background ровно на ту величину, пока не станет видно нужную картинку. Спрайтом также называют картинку, состоящую из объединенных изображений. Чаще всего в спрайты объединяют иконки и другие мелкие изображения.

Еще один из способов оптимизации – использование Base64.

Base64 – это способ вставить картинку прямо в веб-страницу, без обращений к внешним файлам. Все такие «встроенные» изображения используют схему data:URL.

Синтаксис у data:URL следующий:

```
data:[<тип данных>][;base64],<данные>
```

В случае простых изображений Вам нужно указать mime-тип для них (например, image/gif), затем base64-представление бинарного файла с изображением. Такие изображения, внедренные в HTML-страницы, не кешируются для повторного использования, а также от странице к странице (они будут кешироваться только с HTML, их содержащим). Однако CSS кешируется браузерами, и такие изображения могут быть повторно использованы вместе с использующим их селектором. Но отсюда получается и главное неудобство использования base64 в css: нужно пересчитывать base64-представление изображений и редактировать CSS-файл каждый раз, когда само изображением меняется.

Контрольные вопросы по главе 2

1. Какие цветовые модели существуют?
2. Какие форматы являются векторными?
3. В чем основное преимущество растровых форматов над векторными?
4. Какие алгоритмы сжатия изображения Вы знаете?
5. Чем архивация отличается от компрессии?

3 Модульные сетки

В HTML вывод элементов на странице происходит построчно сверху вниз, и поэтому слой, размещенный в самом верху кода, отобразится раньше слоя, который расположен в коде ниже. Такой механизм делает результат вывода элементов предсказуемым и позволяет на него влиять. Порядок вывода объектов на странице называется «потоком».

Всего существует три вида потоков вывода, которые определяются по свойствам элементов [6]:

1) нормальный поток (normal flow);
2) плавающий (float);
3) абсолютное позиционирование (absolute positioning).

Нормальный поток документа *– модель, по которой элементы располагаются на странице в соответствии с CSS-спецификацией и своим расположением в исходном коде страницы (рис. 3.1).*

Рис. 3.1 – Элементы в нормальном потоке

В нормальном потоке все элементы расположены последовательно, причем каждый блочный элемент выводится с новой строки, а строчные – в одной строке.

CSS-свойства `float` и `position` (кроме `position: relative`) вырывают элементы из нормального потока и отображают в соответствии со своими правилами, поэтому для соседних элементов из нормального потока они становятся «невидимыми». Это значит, что блоки больше не оказывают на них влияние: не отталкивают с помощью внешних отступов, не занимают место [5].

Поток документа *– это воспроизведение текста, написанного на западных языках, слева направо и сверху вниз, и обычная схема компоновки текста традиционных HTML-документов. Для восточных языков направление потока может меняться.*

В момент загрузки страницы движок рендеринга в браузере определяет размеры и положение элементов на странице и выделяет им прямоугольную область. Эту прямоугольную область (или просто блок) описывает *блочная модель CSS*.

Блочная модель *CSS описывает прямоугольный блок, генерируемый для элемента в дереве документа и выводящийся согласно визуальной модели форматирования* [6].

По сути это коробка, обтекающая каждый элемент на странице, а состоит она из четырех слоев: внешние отступы, рамка, внутренние поля и внутренняя область элемента (`content area`). Последняя содержит текст или любые другие элементы, расположенные внутри. Зачастую эта область имеет фон или изображение, а её размеры называют шириной контента и высотой контента. На рисунке 3.2 представлена блочная модель прямоугольника с шириной содержимого 360px, внутренними полями – по 15px каждый, рамкой шириной в 2px и внешними отступами.

Ширину и высоту содержимого можно задать явно с помощью css-правил. Также можно ограничить диапазон возможной ширины (и высоты), используя свойства `min/max-width`.

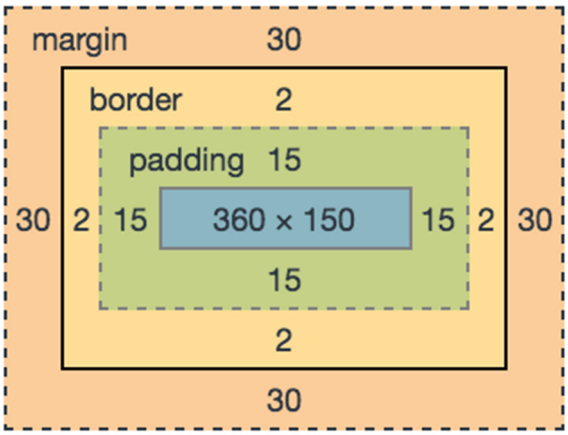


Рис. 3.2 – Блочная модель элемента

В CSS используются два способа расчета размеров блока: стандартный и использовавшийся в Internet Explorer 6.

Ширина элемента в стандартной модели рассчитывается по формуле:

`width` = `content-width` + `padding-left` + `padding-right` + + `border-left` + `border-right`,

где `content-width` – ширина содержимого;

`padding-left` – левое внутреннее поле;

`padding-right` – правое внутреннее поле;

`border-left` – ширина левой рамки;

`border-right` – ширина правой рамки.

Соответственно высота элемента рассчитывается следующим образом:

`height` = `content-height` + `padding-top` + `padding-bottom` + + `border-top` + `border-bottom`,

где `content-height` – высота содержимого;

`padding-top` – верхнее внутреннее поле;

`padding-bottom` – нижнее внутреннее поле;

`border-top` – ширина верхней рамки;

`border-bottom` – ширина нижней рамки.

А вот в браузере Internet Explorer ширина рассчитывалась уже с учетом внутренних отступов и рамок. До сих пор такой способ расчета считается лучше, т. к. при стандартном способе расчета элемент с явно заданной шириной может вылезти за границы своего родителя.

Чтобы изменить алгоритм расчета в CSS существует свойство `box-sizing`, которое может принимать значения:

- `content-box` – значение по умолчанию. Основывается на спецификации CSS 2.1, расчет ширины и высоты ведется в зависимости от ширины и высоты контента;
- `border-box` – ширина рассчитывается как сумма ширины контента, полей и рамки. При указании явной ширины и полей контент занимает только то пространство, которое составляет разница этих значений;
- `padding-box` (не входит в официальную спецификацию, т. к. поддерживается только в браузере Firefox) – ширина и высота включают в себя значения полей, но не рамки и внешних отступов.

В некоторых случаях, при отсутствии явно заданной ширины, блок может заполнить все свободное пространство в строке. В остальном он зависит от ши-

рины содержимого. С высотой дело обстоит иначе: если не задано css-свойство, то высота равна высоте содержимого, вплоть до того, что может быть равна 0.

На размеры блока также влияют внешние отступы (`margin`), поля (`padding`) и рамка. Наличие внешних отступов сообщает, какое пустое пространство должно располагаться между текущим блоком и соседними, и задается с помощью общего свойства `margin` или отдельно для каждой стороны.

Поля элемента (`padding area`) – отступ от контента до внешних границ блока. Размеры полей, как и внешних отступов, задаются по отдельности с разных сторон или общим свойством `padding`.

Область рамки (`border area`) окружает поля элемента. Размеры элемента с учетом полей и рамки иногда называют внешней шириной/высотой элемента.

Рамка задается с помощью свойства общего свойства `border` или же отдельных свойств для цвета, стиля и ширины каждой стороны по отдельности. Помимо свойства `border` существует также свойство `outline` – внешняя рамка. В отличие от обычной рамки, внешняя не входит ни в один способ расчета ширины, соответственно никак не влияет на размеры блока (рис. 3.3).

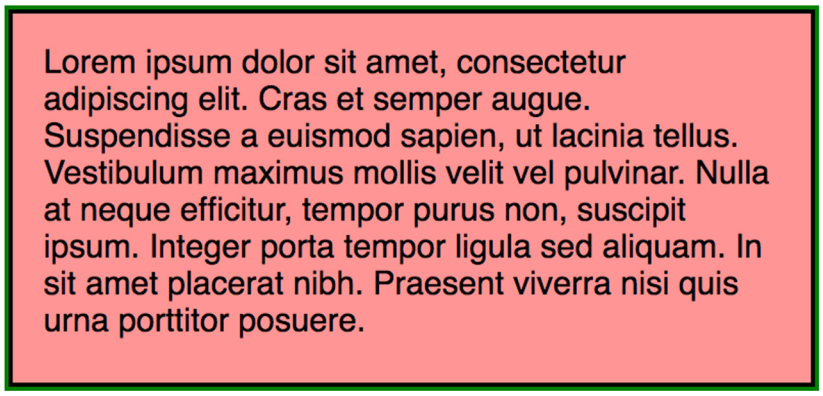


Рис. 3.3 – Блок с полями размером 15px, черной рамкой и зеленой внешней рамкой

Теперь стоит поподробнее рассказать о блочных и строчных элементах, упомянутых в самом начале главы.

Каждый элемент по умолчанию имеет тип отображения. Самые распространенные типы отображения – это блок или строка [7].

3.1 Блочные элементы

Для блочных элементов доступны к изменению все слои блочной модели: ширина, высота, внутренние отступы, внешние отступы, рамка. Значения мож-

но указывать в процентах, `em`, `rem`, `vw`, `vh` и т. д. Важно отметить, что значение внешних отступов, указанное в процентах, рассчитывается от ширины родителя (в том числе верхние и нижние отступы). Это сделано для того, чтобы не уйти в рекурсию по высоте.

Если расположить блочные элементы друг за другом и указать им отступы, то расстояние между элементами будет равно значению наибольшего отступа, т. е. они схлопнутся. Эффект схлопывания также можно наблюдать между родителями и потомками: внешний отступ дочернего элемента выходит за границы родителя. Чтобы этого избежать, нужно у родительского элемента указать поле или рамку больше нуля. Также с помощью отступов можно добиться расположения элемента по центру родительского элемента: нужно указать значения `margin-left` и `margin-right` равным `auto`. Однако этот способ работает только в том случае, когда ширина элемента задана явно.

3.2 Строчные элементы

Строчные (`inline`) элементы ведут себя как слова в предложении: располагаются друг за другом в одной строке, при необходимости строка переносится.

Одной из особенностей строчных элементов является то, что их ширина и высота зависят только от содержимого, а свойства `width` и `height` на них не действуют. Примером инлайн-элемента является `span`.

На размеры элемента можно влиять за счет изменения размера шрифта или высоты строки (`line-height`).

3.3 Блочно-строчные элементы

Помимо строчных и блочных элементов в CSS есть тип отображения, который совмещает в себе поведение строчного и блочного элемента: им можно задавать размеры, рамки, отступы, но ширина по умолчанию зависит от содержания (а не растягивается на весь родительский элемент) и несколько строчно-блочных элементов могут располагаться на одной строке.

3.4 Свойство display

Для того чтобы присвоить элементу блочно-строчное поведение, необходимо указать CSS-свойство `display: inline-block`. Соответственно для блочного элемента – `display: block`, а для строчного – `display: inline`.

Помимо этих трех, свойство `display` также принимает значения: `flex`, `inline-flex`, `inline-table`, `list-item`, `run-in`, `table`, `table-caption`, `table-column-group`, `table-header-group`, `table-footer-group`, `table-row-group`, `table-cell`, `table-column`, `table-row`, `none`, `initial`, `inherit`. Значение `none` удаляет элемент из потока, а `inherit` позволяет наследовать значение `display` от родителя.

3.5 Управление потоком. Построение сетки

До появления блочно-строчных элементов такого же эффекта добивались с помощью плавающего потока: элементам присваивались значения `float: left|right`, а для того, чтобы элемент полностью не выпадал из потока, применялось свойство `clear`. Со временем такой способ управления потоком стал избыточным, поэтому на данный момент использование плавающих элементов для построения сетки не рекомендуется. Взамен предлагается использовать гибкие блоки.

Гибкие блоки (flexbox – флексбоксы) – инструмент, позволяющий максимально рационально использовать свободное пространство за счет способности изменять ширину/высоту и порядок элементов. Однако самым главным преимуществом сетки, построенной на гибких блоках, является независимость от направления, что делает этот способ незаменимым при создании мобильных интерфейсов или адаптивной верстке.

В первую очередь флексбоксы – это целый модуль, состоящий из совокупности множества свойств. Некоторые из них применяются к родительскому контейнеру, называемому «гибкий (или flex-) контейнер», а некоторые – к дочерним элементам (соответственно «гибкие (flex-) элементы»). Для того чтобы создать flex-контейнер, достаточно присвоить блоку значение свойства `display` равным `flex` или `inline-flex`. Разница между этими значениями только в способе взаимодействия с окружающими элементами (как у `block` и `inline-block`).

Система флексбоксов построена на собственном механизме раскладки. Flex-элементы располагаются уже не в обычном, а гибком потоке, состоящем из главной (main axis) и поперечной оси (cross axis). По умолчанию главная ось располагается слева направо, поперечная – сверху вниз (рис. 3.4).

Элементы будут располагаться в основном вдоль главной оси. Но это не значит, что она всегда располагается по горизонтали: направление главной и

поперечной оси можно менять – для этого существует свойство `flex-direction` со значениями:

- `row` – главная ось располагается слева направо;
- `row-reverse` – главная ось располагается справа налево;
- `column` – главная ось располагаемся сверху вниз;
- `column-reverse` – главная ось располагает снизу вверх (рис. 3.5).

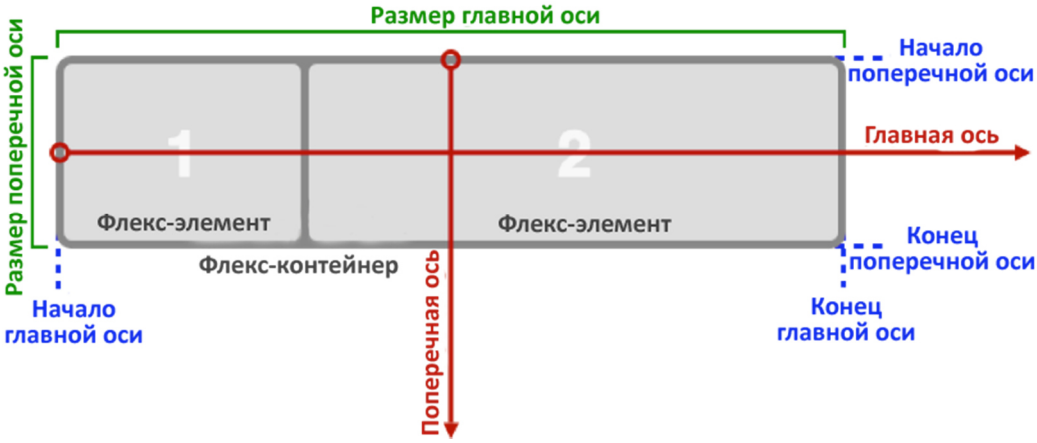


Рис. 3.4 – Механизм раскладки гибких блоков

flex-direction: row;

1 block 2 block 3 block 4 block

flex-direction: column;

1 block
2 block
3 block
4 block

Рис. 3.5 – Расположение элементов
при изменении направления главной оси

Выравнивать элементы можно как по главной, так и по поперечной оси. Для выравнивания элементов по главной оси применяется свойство `justify-content`, а для выравнивания по поперечной – `align-items` (рис. 3.6).

Свойство `justify-content` может принять одно из пяти значений:

- `flex-start` – выравнивает блоки у начала оси;
- `flex-end` – выравнивает блоки в конце оси;
- `center` – выравнивает блоки в центре оси;

- `space-between` – равномерно выравнивает блоки вдоль оси, крайние блоки прижимаются вплотную к краям контейнера;
- `space-around` – блоки выравниваются таким образом, что свободное пространство равномерно распределяется между ними.

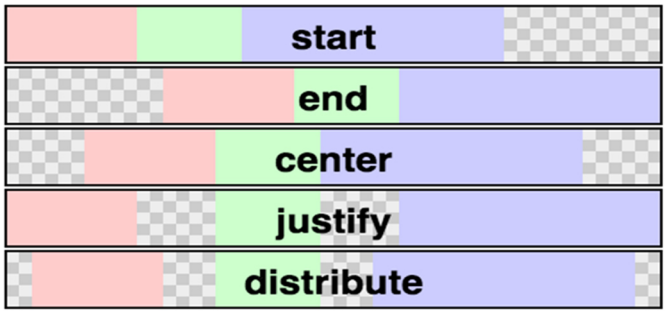


Рис. 3.6 – Выравнивание вдоль главной оси

Значения свойства `align-items` во многом похожи на `justify-content`:

- `flex-start` – все элементы прижимаются к началу строки;
- `flex-end` – элементы прижимаются к концу строки;
- `center` – элементы выравниваются по центру строки;
- `baseline` – элементы выравниваются по базовой линии текста;
- `stretch` (`default`) – элементы растягиваются, заполняя полностью строку (рис. 3.7).

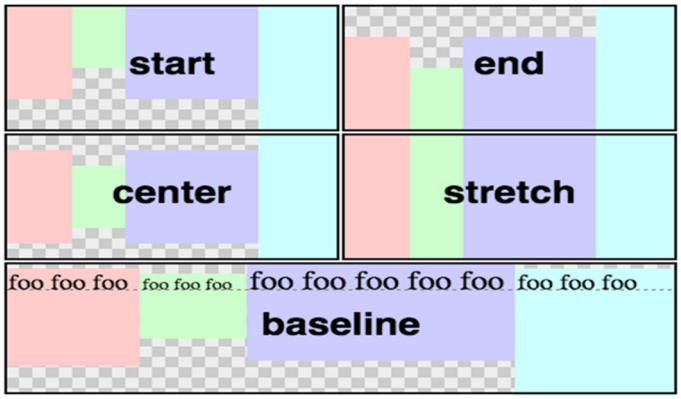


Рис. 3.7 – Выравнивание элементов вдоль поперечной оси

Элементы могут не помещаться во всю строку. Для того, чтобы можно было переносить гибкие элементы на новую строку, используется свойство `flex-wrap: wrap`. Соответственно, если переноса блоков не планируется, то можно оставить значение по умолчанию – `no-wrap`.

Все вышеперечисленные свойства применяются к гибкому контейнеру (хоть и влияют на элементы). Непосредственно для самих элементов также существуют свойства:

- `align-self` – выравнивание по поперечной оси отдельного элемента;
- `order` – применяется для управления порядком блоков. Принимает целочисленные значения.

Контрольные вопросы по главе 3

1. Какие виды потока документа Вы знаете? Какие у них плюсы и минусы?
2. Что такое флексбокс? Назовите ключевые особенности.
3. Как выглядит блочная модель?
4. Чем блочные элементы отличаются от блочно-строчных?
5. Для чего предназначено свойство `box-sizing`?
6. Назовите способы управления потоком.

4 Декоративные элементы

4.1 Шрифт

Шрифт *– графический рисунок начертаний букв и знаков, составляющих единую стилистическую и композиционную систему, набор символов определенного размера и рисунка.*

Компьютерный шрифт *– это файл, содержащий в себе набор графических символов и соответствующих им кодов. Символы могут быть различными по назначению: языковые знаки, числа, графические, специальные и др.*

В эпоху матричных принтеров и мониторов с низким разрешением царствовали растровые шрифты. Такие шрифты представляли собой набор растровых изображений каждого символа в виде битмапа, состоящего из цветных пикселов. При увеличении размера шрифта его детализация не увеличивалась, и пикселизация была видна еще больше.

Позже появились векторные шрифты, описывающие символы с помощью математических формул, что позволяет их масштабировать без потери качества, как и любое другое векторное изображение. Небольшой издержкой отрисовки такого шрифта является несколько большая нагрузка на процессор, т. к. перед растеризацией необходимо также рассчитать различные кривые, сглаживание линий и т. д.

Шрифты обладают обширным количеством параметров и особенностей, комбинированием которых решаются те или иные задачи. В процессе развития типографики появились шрифты и предоставилась возможность их классифицировать, большое значение имеет задание параметров, таких как:

Гарнитура – типографский термин, объединяющий набор шрифтов, которые отличаются по размеру, начертанию, наличию или отсутствию засечек на концах линий, пропорциям символов, соотношению размера высоты прописных и строчных знаков, величине верхних и нижних выносных элементов, плотности, то есть близких по характеру и отличительным знакам рисунка. Примеры известных гарнитур: Arial, Times New Roman.

Базовая линия – воображаемая прямая линия, проходящая по нижнему краю прямых знаков без учёта свисаний и нижних выносных элементов.

В строке символы текста стоят на базовой линии, а нижние выносные элементы текста «свисают» с неё.

Высота строчных знаков – расстояние от базовой линии до верхней линии строчных, то есть высота строчных букв без свисаний и выносных элементов.

Высота заглавной буквы – расстояние от базовой линии до верхней линии прописных, то есть высота прописных букв без учета свисаний.

Свисание – выступающая вниз за базовую линию (нижнее свисание) или вверх за верхнюю линию (верхнее свисание) часть контура круглого или остроконечного знака. Применяется для оптического выравнивания высоты знака по отношению к соседним.

Интерлиньяж, межстрочный интервал – расстояние между базовыми линиями соседних строк. Так как в Вебе данный параметр называется *высотой строки*, дальше мы будем использовать именно его.

Кегельная площадка. В металлическом наборном шрифте: верхняя прямоугольная часть ножки литеры, на которой расположено выпуклое (печатающее) изображение знака. В цифровом шрифте: прямоугольник, в который вписывается изображение знака.

CSS и шрифт

С помощью `font-size` можно определить размер шрифта элемента. Размер шрифта определяется как высота от базовой линии до верхней границы кегельной площадки. Он может быть установлен несколькими способами. Набор констант (`xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`) задает размер, который называется абсолютным. На самом деле они не совсем абсолютны, поскольку зависят от настроек браузера и операционной системы.

Другой набор констант (`larger`, `smaller`) устанавливает относительные размеры шрифта, зависящие от размера шрифта родителя. Изменение составляет примерно 20%. Например, если размер шрифта-родителя 20px, то размер шрифта-потомка с `larger` будет составлять 24px.

Другой более популярный и рекомендуемый способ – использование единиц измерения CSS:

- **`em`**. Установка размера шрифта в зависимости от размера шрифта-родителя. Например, если у родителя размер шрифта 16px, то значение 1.25em будет рассчитываться по формуле 16×1.25 и составит

20 пикселов. Обычно такой подход используется, когда необходимо, чтобы блок подстраивался под различных родителей;

- **rem**. Подобно em, но зависит не от родительского размера шрифта, а от размера шрифта корневого элемента (html);
- **%**, как и em, устанавливает размер шрифта, зависящий от родителя. Различия от em могут наблюдаться в редких случаях и только при изменении размера шрифта в настройках браузера;
- **px** позволяет указать размер шрифта в пикселах. Так как задание динамического размера требуется нечасто, эта единица измерения является наиболее популярной;
- **ex** соответствует ширине символа x. Не рекомендуется к использованию;
- **ch** соответствует ширине символа 0 (ноль). Не рекомендуется к использованию;
- *абсолютные единицы измерения* **pt**, **pc**, **cm**, **mm**, **in** пришли из типографики и также не рекомендованы к использованию для размера шрифта, т. к. могут очень по-разному выглядеть в зависимости от монитора/устройства.

Единица измерения em может быть использована не только для указания размера шрифта. Если указать отступ в em, то он также будет зависеть от размера шрифта родителя.

Следует учитывать, что связь свойства font-size с тем, что отображается на экране, в действительности определяется разработчиком шрифта. Размер шрифта характеризуется кегельной площадкой (em square) (некоторые называют ее кегельным квадратом) шрифта. Кегельная площадка (и соответственно размер шрифта) не задается границами каких-либо символов шрифта. Эта величина определяется расстоянием между базовыми линиями, если шрифт задан без дополнительных межстрочных интервалов. Шрифты могут иметь символы, размер которых превышает стандартное расстояние между базовыми линиями. Впрочем, шрифт может быть определен так, что все его символы будут меньше, чем его кегельная площадка, что и наблюдается у многих шрифтов. Таким образом, действие font-size состоит в задании размера кегельной площадки используемого шрифта. Это не гарантирует, что любой из фактически отображаемых символов будет иметь такой размер.

Высота строки (межстрочный интервал, интерлиньяж) – расстояние между базовыми линиями соседних строк. Устанавливается с помощью свойства `line-height` с помощью трех типов значений: множителя, или единиц длины и процентов.

Множитель устанавливает интерлиньяж равным размеру шрифта, умноженному на множитель:

```
<p style="font-size: 20px; line-height: 1.5;">
С другой стороны укрепление и развитие структуры обеспечивает широкому кругу (специалистов) участие в формировании направлений прогрессивного развития.
</p>
```

В результате интерлиньяж будет равен 10px × 1.5 = 15px.

Пикселы, пункты, сантиметры и подобные им единицы длины позволяют указать независимое значение интерлиньяжа. Значение, указанное в `em` или в процентах, так же как и множитель, зависит от размера шрифта блока (умножается на него). Различие между множителем и `em`/`%` заключается в том, как они наследуются дочерним элементам. Значение в `em`/`%`, унаследованное от родителя, рассчитывается умножением этого значения на размер шрифта родителя. А множитель, унаследованный от родителя, рассчитывается умножением размера шрифта дочернего элемента на множитель.

Разрядка *– способ выделения текста, широко использующийся в традиционной русской типографике; заключается в увеличении интервала между буквами.*

В Вебе разрядка применяется редко, как правило в заголовках. Для изменения разрядки предусмотрено свойство `letter-spacing`. Оно позволяет как увеличивать, так и уменьшать это расстояние.

В качестве значений можно указывать любые единицы длины, кроме процентов. Современные браузеры позволяют указывать дробные значения, например 0.5px.

Расстояния между словами можно изменить с помощью свойства `word-spacing`. В качестве значения можно указать любые единицы длины, кроме процентов. Следует иметь в виду, что это свойство влияет только на обычные и неразрывные пробелы.

Капитель – это начертание в гарнитуре, в которой строчные знаки выглядят, как уменьшенные прописные. Установить такое начертание можно с помощью параметра `font-variant` со свойством `small-caps`. Значение по умолчанию `normal`.

Наклон используется, чтобы выделить важные слова и фразы, часто применяется в цитатах как речь автора. Существует два типа наклона: курсив (`italic`) и наклонный шрифт (`oblique`). Если курсив представляет собой специально спроектированный шрифтовой набор, имеющий лишь отдаленное сходство с соответствующим прямым шрифтом, то наклонный представляет собой лишь модифицированный прямой, слегка «заваленный» вправо. Наклон осуществляется с помощью свойства `font-style` со значениями `italic` или `oblique`.

Насыщенность устанавливается при помощи свойства `font-weight`.

У этого свойства существуют две категории значений: числа и ключевые слова. Числа – это значения от 100 до 900, кратные ста, т. е. 100, 200, 300, 400, 500, 600, 700, 800 или 900. Каждое число является степенью насыщенности. При этом 400 – это обычный вид текста и значение по умолчанию. 700 соответствует стилю текста «жирный» в текстовых процессорах. Следует отметить, что степени, отличные от 400 и 700, встречаются значительно реже. Как правило, это специализированные шрифты.

Часто для указания жирности используются ключевые слова. Среди них: `normal`, `bold`, `bolder` и `lighter`. `bold` (соответствует 700) является самым популярным и доступен для использования в каждом из системных шрифтов.

`bolder` и `lighter` устанавливают степень жирности на один шаг в большую или меньшую сторону относительно уровня родителя. При этом шаг не обязательно равен +100 или –100, а зависит от доступных уровней жирности. Например, если родитель с `font-family: Arial` имеет `font-weight: normal`, то значение `bolder` у потомка будет эквивалентно значению `bold`, т. к. Arial имеет лишь градации `normal` (400) и `bold` (700).

Браузер не пытается сам отрисовать градации, отличные от `bold`. Если шрифт поставляется только со степенями `normal` и `bold`, то при использовании числовых значений свойства `font-weight` браузер будет вести себя следующим образом: значения 100–500 округляются до значения `normal` (400), 600–900 округляются до `bold` (700).

Чтоб иметь возможность использовать множество градаций, следует подключать шрифты особым образом, о чем будет рассказано ниже.

Существуют элементы, имеющие `font-weight: bold` по умолчанию, например: `<strong>`, `<b>`.

Плотность *– один из признаков начертания шрифта, по которому шрифты в зависимости от зрительного соотношения ширины знаков с их высотой делятся на шрифты узкого, нормального и широкого начертания (а также ротации). Например, в семействе шрифтов Arial существует модификация Arial Narrow, символы которой узкие.*

В CSS для указания плотности используется свойство `font-stretch`, значениями которого являются градации плотности в виде ключевых слов: `ultra-condensed`, `extra-condensed`, `condensed`, `semi-condensed`, `normal`, `semi-expanded`, `expanded`, `extra-expanded`, `ultra-expanded`.

Браузер не рендерит это свойство своими силами, поэтому, чтоб оно работало, шрифт с каждой градацией нужно подключать отдельно.

Категории шрифтов в Вебе

В мире существует множество типов шрифтов и их различных иерархических классификаций.

В Вебе утверждено пять видов шрифтов: с засечками, без засечек, моноширинные, рукописные, декоративные.

Наиболее распространенные из них – шрифты с засечками и без засечек. Шрифт без засечек, как правило, состоит из линий одинаковой толщины. У шрифта с засечками есть эти самые засечки, которые располагаются на концах линий. Толщина линии часто варьируется в местах сгибов и не только.

Для различных ситуаций требуются гарнитуры с различными свойствами. Например, в книжном и газетном деле наиболее распространены гарнитуры с засечками. Хотя об этом идет много споров, но многими считается, что шрифт с засечками менее утомителен при чтении.

Остальные типы шрифтов используются редко, для определенных ситуаций.

В поставке каждой операционной системы имеется набор шрифтов. Существует ряд шрифтов, которые есть практически в каждой из них, таким обра-

зом их использование считается наиболее безопасным. Среди шрифтов без засечек таким шрифтом являются Arial. Среди шрифтов с засечками – Times New Roman, среди моноширинных – Courier New.

Шрифт блока на сайте можно указать с помощью свойства `font-family`, например:

```
.block {
    font-family: 'Arial';
}
```

Имеется также возможность указать несколько шрифтов через запятую.

```
.block {
    font-family: Arial, Helvetica; /* Шрифты можно указывать без кавычек */
}
```

Если браузер не находит в системе первый в списке шрифт, он будет пытаться найти следующий, пока в конце концов не использует шрифт браузера по умолчанию (обычно это Times New Roman). В последнем случае имеет смысл сообщить браузеру о категории шрифта, чтобы он мог выбрать более подходящий из имеющихся в системе. Для этого существуют пять ключевых слов: serif (с засечками), sans-serif (без засечек) и monospace (моноширинный), cursive (рукописный) и fantasy (декоративный).

Окончательный вариант оптимального указания шрифта будет выглядеть так:

```
.block {
    font-family: Arial, Helvetica, sans-serif;
}
```

Существует возможность встраивать нестандартные (то есть отсутствующие в системе) шрифты на страницу. Например, такая потребность может возникнуть из-за необходимости отобразить символ рубля.

Для подключения шрифтов используется правило `@font-face`, в котором необходимо указать название шрифта и адрес источника шрифта:

```
@font-face {
    font-family: myCustomFont;
    src: url(myCustomFont.woff); /* .woff – один из популярных форматов веб-шрифтов */
}
```

Затем для применения шрифта к блоку просто указывают его имя в font-family (а также запасные варианты на случай, если хост с источником недоступен):

```
body {
    font-family: myCustomFont, sans-serif; /* либо serif, если подключаемый шрифт с засечками */
}
```

Как было сказано ранее, наклонный стиль и степени жирности шрифта (а также их комбинации) являются отдельными файлами. Таким образом, чтобы задействовать курсивную версию шрифта, понадобится соответствующий файл шрифта.

Однако такой вариант использования является неоптимальным, т. к. увеличение количества названий заставляет запоминать название и его предназначение. Кроме того, нарушается семантика описания стиля шрифта, ведь более логичным было бы использовать font-style: italic вместо font-face. Для получения такой возможности необходимо описать в @font-face свойства шрифта, характеризующие его внешний вид:

```
@font-face {
    font-family: myCustomFont;
    src: url(myCustomItalicFont.woff);
    font-style: italic;
}
```

Таким образом, все шрифтовые CSS-свойства, описанные в @font-face, указывают браузеру, что myCustomItalicFont.woff необходимо применять только тогда, когда описанные в @font-face свойства шрифта совпадают со свойствами шрифта, описанными в блоке:

```
em{
    font-style: italic;
}
```

В теге будет использован именно myCustomItalicFont.woff, т. к. унаследует от <body> свойство font-family: myCustomFont, а font-style: italic укажет браузеру, что необходимо применить курсивную версию гарнитуры.

Подключение шрифта в @font-face без указания свойств font-style, font-stretch и font-weight эквивалентно подключению с указанием font-style: normal; font-weight: normal; font-stretch: normal.

Base64 – один из способов кодирования различной информации в Вебе. Помимо прочего, он позволяет кодировать файлы шрифтов и тем самым интегрировать их прямо в описание стилей, непосредственно вставляя закодированный шрифт в поле `src` правила `@font-face`:

```
@font-face {
    font-family: 'latoregular';
    src:    url(data:application/x-font-woff;charset=utf-
8;base64,d09GRgABAAAAAHwwABMAAAAA4IwAAQA==)
format('woff');
    /* На деле кода base64 значительно больше, для при-
мера приведена лишь часть */
}
```

Такой вариант подключения может существенно увеличить размер CSS-файла, однако избавляет браузер от необходимости делать дополнительный запрос за файлами шрифта. Польза этого приема заметна при использовании мобильного Интернета, где достаточно быстрое скачивание, но не самая быстрая обработка запросов.

Что произойдет, если для обычного начертания указать наклонный стиль отображения `font-style: italic`? В дело вступают так называемые щадящие алгоритмы – браузер попытается «наклонить» обычное начертание, используя для этого свой упрощенный алгоритм наклона. Нужно понимать, что разница с настоящим наклонным шрифтом может быть весьма заметна. С жирностью ситуация аналогична.

Существуют сервисы, позволяющие быстро подключить на страницу различные бесплатные шрифты. Наиболее известным является Google Fonts. После выбора шрифта сервис предлагает вставить на страничку ссылку на стиль вида `<link href='https://fonts.googleapis.com/css?family=Open+ +Sans' rel='stylesheet' type='text/css'>`. В подключаемом CSS-файле находится набор правил `@font-face` с указанием источников шрифтов на серверах Google.

Некоторое время назад браузеры поддерживали различные по отношению друг к другу форматы шрифтов. Для кроссбраузерного подключения приходилось использовать несколько форматов: .svg, .ttf, .woff, .eot. В настоящее время при использовании последних версий браузеров достаточно использовать лишь .woff. Однако версии Android 4.3 и ниже не понимают формат .woff и для них необходимо подключить также .ttf. Таким образом рекомендуется использовать

по крайней мере два этих формата. Чтоб подключить более одного формата, необходимо перечислить источники через запятую:

```
@font-face {
     font-family: myCustomFont;
     src: url('myCustomFont.woff') format('woff'),
          url('myCustomFont.ttf') format('truetype');
}
```

Более распространенный формат лучше указать раньше, т. к. браузер будет использовать первый встреченный им и понятный ему формат.

Существует несколько способов получения .woff и других форматов. Наиболее простой и быстрый вариант – использование соответствующих веб-сервисов, наиболее популярным из которых является fontsquirrel. Сервис предлагает загрузить в него имеющийся шрифт, после чего он генерирует архив с нужными форматами. Наибольшим преимуществом этого сервиса является возможность указания необходимых символов (или их диапазона), что может существенно уменьшить размер файлов шрифтов. Как правило, достаточно выбрать кириллический, латинский диапазоны, а также набор типографических символов. Главным недостатком такого рода сервисов является не всегда предсказуемый и приемлемый результат генерации, заключающийся главным образом в некотором искажении рендеринга символов. Результат может зависеть от конкретного сервиса.

Старые версии браузеров, а также Internet Explorer сначала отображали запасной шрифт, а после загрузки шрифта применяли новый. Момент подмены одного шрифта другим называется FOUT (flash of unstyled text).

Современные браузеры при обнаружении использования нестандартного шрифта не отображают текст с этим шрифтом вообще, пока шрифт не будет загружен.

Загрузка файлов шрифта и его отображение на странице – достаточно трудоемкий для браузера процесс, поэтому его следует оптимизировать.

Прежде всего необходимо максимально уменьшить размер шрифта путем исключения из него всех неиспользуемых символов. Как было сказано ранее, сделать это можно с помощью специальных веб-сервисов. Помимо этого существует CSS-свойство `unicode-range`, позволяющее сообщить браузеру, какой диапазон необходимо загрузить, но оно работает не во всех браузерах.

Использование архивации gzip может сэкономить до 60% трафика при загрузке шрифта.

Полезно знать, что можно включить кэширование ресурсов на сервере таким образом, чтоб браузер не выкачивал шрифты заново при каждом новом заходе на страницу, а лишь проверял, не изменился ли файл шрифтов. Возможно также настроить более агрессивное кеширование, когда браузер даже не делает запрос, проверяющий изменения. Обычно такая проверка не нужна, а запрос занимает время, и шрифт не отобразится, пока такой запрос не будет выполнен.

С недавних пор современные браузеры начали поддерживать усовершенствованную версию формата .woff. Его главным преимуществом перед первой версией являются улучшенные алгоритм компрессии, за счет чего файл может «весить» на 30–50% меньше. Для наилучшей совместимости с браузерами при подключении следует указывать обе версии шрифта. Как было сказано ранее, браузер загружает самый первый из понятных ему форматов. Поэтому стоит указать .woff2 раньше всех остальных. Затем .woff и самый старый и «тяжелый» .ttf:

```
@font-face {
     font-family: myCustomFont;
     src: url('myCustomFont.woff2') format('woff2'),
          url('myCustomFont.woff') format('woff'),
          url('myCustomFont.ttf') format('truetype');
}
```

Современные браузеры не начинают загружать шрифт, пока не обнаружат случай его использования на странице. Например, если Ваш CSS-файл «весит» 100 кБ и вызов `font-family: myCustomFont` находится в конце этого файла, то браузер не начнет загружать шрифт, пока не распарсит весь этот файл. Поэтому имеет смысл поместить `@font-face` и использующее CSS правило как можно выше в файле. И сам файл разместить как можно выше в блоке `<head>`.

Свойство `font` позволяет определить множество параметров текста в одном правиле.

```
font:     [font-style||font-variant||font-weight]||[font-stretch] font-size [/line-height] font-family
```

Полный пример использования:

```
p {
     font:   italic   small-caps   bold   semi-condensed   20px/24px Arial;
}
```

`font-style`, `font-variant`, `font-weight` и `line-height` необязательны, можно их не указывать. `font-size` и `font-family` обязательны:

```
p {
     font: 20px Arial;
}
```

Существует несколько способов отобразить символ на странице.

Наиболее простейший и очевидный: набрать нужный символ при помощи клавиатуры. При этом существует возможность при помощи клавиатурных сочетаний набирать не только цифры, буквы и стандартные символы пунктуации, но и более специализированные, например, €, ←, →, ↑, ∞ и т. д. Однако стоит отметить, что многие из таких символов отобразятся в браузере, только если кодировка страницы utf-8.

Более безопасным способом отобразить спецсимвол является использование так называемых именованных символьных сущностей, например: `©` – знак копирайта (©), `—` – длинное тире (—), `≤` – меньше или равно (≤).

Именованная сущность начинается с амперсанда, далее следует название сущности и в конце ставится точка с запятой. При отрисовке страницы браузер заменит такие вхождения на соответствующие символы. Стоит отметить, что не у каждого символа есть именной вариант.

Точка с запятой в конце вызова символа является необязательной, но рекомендуется всегда ее ставить во избежание ошибок.

Unicode

Юникод – самый распространенный и универсальный стандарт кодирования символов в мире, позволяющий представить знаки почти всех письменных языков. Помимо этого в нем закодировано огромное количество различных символик: флаги, логотипы, смайлики и пр. Символы в юникоде представлены в десятичной (decimal) и шестнадцатеричной системах (hex). Например, знак копирайта в десятичной системе имеет вид `#169`, а в шестнадцатеричной – `#xA9`. «x» перед hex-представлением указывает, что далее следует шестнадцатеричное число.

Вызов юникод-символов в HTML также начинается с амперсанда и заканчивается точкой с запятой. Например, знак копирайта © вызывается как `©` или `©`.

Использование юникода гарантирует корректность отображения символа вне зависимости от кодировки страницы при условии, что символьная таблица операционной системы располагает нужным символом. Например, в Windows

XP могут не отобразиться какие-то особо редкие символы, добавленные в unicode после окончания его поддержки компанией Microsoft.

Существуют ситуации, когда необходимо отобразить не сам символ, а то, как он закодирован, например, `©` вместо ©. Для этого необходимо закодировать символы, составляющие код. Для отображения кода копирайта в браузере необходимо написать в коде `&#169`. Таким же образом можно отобразить, например, HTML-теги. Для этого пригодятся кодировки знаков «больше» и «меньше»: `<` и `>`.

В CSS существует псевдоэлемент `::before`, который в свою очередь имеет обязательное свойство `content`, позволяющее вставлять символы. Если страница представлена в utf-8 кодировке, то спецсимволы можно использовать в этом свойстве как есть:

```
div::before{
    content: "©";
}
```

Использование юникода также возможно, но формат его вставки отличается от формата вставки в HTML и допускается только hex-кодирование. Перед кодом ставится символ обратной косой черты, например:

```
div:before{
    content: '\A9';
}
```

Как и во всех текстовых процессорах, текст в Вебе можно выравнивать по левому и правому краям, по центру, а также по ширине. Осуществляется это с помощью свойства `text-align`, которое принимает соответствующие значения: `left`, `right`, `center` и `justify`. При выравнивании по ширине пробелы в каждой строке увеличиваются таким образом, чтоб правый край последнего символа строки был прижат к правому краю.

Выравнивание можно применить только к блочным элементам, оно действует на все строчные элементы. Так, в следующем примере это свойство не будет иметь никакого эффекта, т. к. `<span>` – строчный элемент.

```
<span style="text-align: right;">
    Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim.
</span>
```

Вместо `<span>` следует использовать блочный тег, например `<p>`.

Подчеркивание в Вебе по умолчанию применяется у ссылок. Таким образом, его принято применять к активным (т. е. кликабельным) элементам. Реже применяется для увеличения акцента к словам и фразам. Для придания подчеркивания используется свойство `text-decoration` со значением `underline`.

Перечеркивание, как правило, используется, чтоб показать изменения, произошедшие в тексте. Например, старая и новая цены товаров. Чтоб отобразить перечеркивание у свойства `text-decoration` следует указать значение `line-through`.

Надчеркивание (значение `overline`) в основном используется в математических нотациях.

Значением по умолчанию является `none`. Стоит отметить, что `text-decoration` не наследуется дочерним элементам и его нельзя отменить для отдельных слов, если всему блоку указано подчеркивание или перечеркивание. Например:

```
<p style="text-decoration: line-through;">
    Текст перечеркнут. <span style="text-decoration:
none;">Текст по-прежнему перечеркнут</span>.
<p>
```

Преобразование текста позволяет управлять типом букв (строчные/прописные). Для этого используется свойство `text-transform`.

Абзацный отступ *– это сдвиг первой строки абзаца для логического разделения текста на части. Для задания величины отступа необходимо воспользоваться свойством* `text-indent`.

Указание цвета текста задается с помощью свойства `color`. Цвет в Вебе может быть представлен в различных палитрах. Цвет, указанный в `color`, влияет не только на цвет самого текста. В него окрашиваются линии подчеркивания, надчеркивания и перечеркивания. Помимо этого он наследуется как цвет по умолчанию для многих составляющих элемента, например для бордеров, тени блока и др. Пример:

```
p{
    color: red;
    border: 1px solid; /* Цвет бордера будет красным */
    box-shadow: 2px 2px 3px; /* красная тень блока */
}
```

При этом элементы, которые не наследуют цвет от `color` автоматически, можно «окрасить» вручную при помощи ключевого слова `currentColor`:

```
p{
     color: red;
     background-color: currentColor; /* Цвет фона станет
красный */
}
```

`Color` является наследующимся свойством, поэтому если у дочерних элементов не указан свой `color`, то `currentColor` будет взят от цвета родительского элемента.

Можно добавить к тексту *тень* с различными параметрами: смещение, размытие, цвет. В этом поможет свойство `text-shadow`:

```
text-shadow: [сдвиг по оси X] [сдвиг по оси Y]
                [радиус размытия] [цвет];
```

`[радиус размытия]` является необязательным параметром.

Сдвиг может быть в том числе отрицательным. Можно также указывать две и более нотации теней, перечислив их через запятую:

```
p{
     text-shadow: 2px 2px 3px red,
          -2px -2px 3px green;
}
```

Многоколоночность

Многоколоночность *– возможность разбивать блоки текста на колонки. Этот прием часто применяется при верстке газет и журналов в связи с тем, что читать широкие блоки текста некомфортно.*

Обычное оформление текста:

```
<p>
     В европейских языках чтение текста происходит слева
направо, в то время как есть языки, где текст читается
справа налево. При смешении в одном документе разных по
написанию символов (русского с ивритом, к примеру) в
системе юникод, их направление определяется браузером из
характеристик и содержимого текста.
</p>
```

Чтоб разбить текст на колонки, необходимо указать нужное количество колонок в свойстве column-count:

```
p{
    column-count: 2; /* В браузерах Firefox и Chrome (и др. на его основе) свойство работает только с префиксами. */
}
```

Если необходимо, чтобы текст разбивался на колонки только тогда, когда родительский блок становится достаточно широким, можно прибегнуть к свойству column-width. Оно задает минимальную ширину колонки. Если блок не может вмесить в себя нужное количество колонок, т. е. ширина блока меньше, чем количество колонок, умноженное на column-width, то браузер уменьшает количество колонок до количества, способного поместиться в блок. Предположим, есть блок с текстом шириной 1000px. Ему заданы 5 колонок с помощью column-width: 5 и размер колонки 300px (column-width: 300px). Расчетная ширина колонок получается равной 1500px, что превышает ширину родительского блока. Влезть могут только три колонки, и именно столько покажет браузер в такой ситуации. Таким образом значения этих двух свойств принято называть не точными, а оптимальными.

Для изменения расстояния между колонками используется свойство column-gap. Его величина влияет на расчет количества колонок.

По умолчанию разделитель между колонками отсутствует. С помощью свойства column-rule можно его добавить, задав необходимый вид, а именно ширину, тип и размер через пробел, например: column-rule: 1px solid #ccc.

Свойство column-span со значением all позволяет расположить текст поверх колонок (например, заголовок):

```
div{
    column-count: 2;
}
h3{
    column-span: all;
}
<div>
```

В европейских языках чтение текста происходит слева направо, в то время как есть языки, где текст читается справа налево.

При смешении в одном документе разных по написанию символов (русского с ивритом, к примеру) в системе юникод их направление определяется браузером из характеристик и содержимого текста.

Перенос *– это разрыв части текста, при котором её начало оказывается на одной строке, а конец – на другой.*

По умолчанию в Вебе переносы могут быть в местах символов пространства (пробелы, табуляция и прочие) и дефисов. В зависимости от браузера они могут быть и около других символов. Например, и Firefox, и Chrome переносят текст перед символом тире, но только Chrome переносит текст и после.

Для того чтоб перенести текст в произвольном месте на следующую строку, необходимо указать тег `<br>` в месте переноса:

```
<p>
    Далеко-далеко за словесными горами<br>в стране гласных и согласных живут рыбные тексты.
</p>
```

Переносы внутри слов используются для того, чтобы привести к минимуму разброс ширины пробелов у выровненного по ширине текста, а также чтоб сократить «лесенки» правого края текста, выровненного по левому краю.

Чтобы задействовать автопереносы, необходимо воспользоваться свойством `hyphens`, выставив ему значение `auto`. В этом случае браузер будет расставлять переносы своими силами. Для этого ему необходимо подсказать правильный язык документа или блока с текстом с помощью атрибута `lang`:

```
<p lang="ru">
    Далеко-далеко за словесными горами в стране гласных и согласных живут рыбные тексты.
    Вдали от всех живут они в буквенных домах на берегу Семантика большого языкового океана.
</p>
```

В большинстве современных европейских языков перенос обозначается дефисом после начальной части разорванного слова.

Правила переноса букв могут различны в разных языках. Например, в русском языке буквы переносятся по слогам. При этом слоги из одной буквы переносить недопустимо. В английском языке некоторые переносы могут требовать дублирования буквы, после которой происходит перенос, например: eighteen → eight-//teen.

Возможность побуквенного переноса появилась в браузерах сравнительно недавно, поэтому почти во всех из них `hyphens` работает только с префиксами. И поддержка того или иного языка, в зависимости от браузера, может отсутствовать. Например, автопереносы в русском языке пока что не работают в Chrome.

Существует возможность «подсказать» браузеру, где допустимо делать переносы. Для этого необходимо расставить служебные символы `` в соответствующих местах:

```
<p>
    <-- Далеко-далеко за словесными горами -->
    Да&shy;ле&shy;ко-да&shy;ле&shy;ко за
сло&shy;вес&shy;ны&shy;ми го&shy;ра&shy;ми.
</p>
```

Чтобы мягкие переносы работали, свойство `hyphens` должно иметь значение `manual`. Оно является значением по умолчанию, таким образом указывать его необязательно.

В редких случаях в конце строки и при переносе слова может понадобиться добавить мягкий перенос, не добавляющий символ. Тогда вместо `` необходимо использовать тег `<wbr>`. Такой мягкий перенос может пригодиться при переносе веб-адресов, т. к. символ дефиса может быть воспринят как часть адреса:

```
    <p>http://this<wbr>.is<wbr>.a<wbr>.really<wbr>.long<wbr>.
example</p>
```

Однако расставлять такие теги вручную неудобно. В большинстве случаев более удобным будет воспользоваться свойством `word-break` со значением `break-all`.

Существует ряд ситуаций, когда необходимо запретить перенос в тех местах, где по умолчанию текст переносится. Например, в типографике принято, чтоб в конце строк не оставалось висящих предлогов. Рассмотрим следующий пример:

```
<p style="width: 100px;">
    Вдали от всех живут они в
    буквенных домах.
</p>
```

Предлог «в» некрасиво «подвисает» в конце строки. Чтоб решить эту проблему, необходимо, чтобы пробел после предлога не приводил к переносу

текста. Тогда предлог и следующее слово будут неразрывны. Добиться этого можно несколькими способами:

1. Использование символа неразрывного пробела ` ` (Non-breaking space). Это специальный символ, который выглядит, как пробел, но предотвращает перенос текста там, где он расположен:

```
<p style="width: 100px;">
    Вдали от всех живут они
    в буквенных домах.
</p>
```

Так как сочетание «в буквенных» стало единым целым, оно перенесется полностью.

Существуют сервисы (а также приложения), позволяющие расставить в правильных местах текста неразрывные пробелы. Одним из таких сервисов является «Типограф».

2. Использование CSS-свойства `white-space` со значением `nowrap`, которое предотвращает перенос текста. Так как необходимо предотвратить перенос только у фразы «в буквенных», нужно обернуть её в какой-нибудь строчный тег с этим свойством:

```
<p style="width: 100px;">
    Вдали от всех живут они
    <span style="white-space: nowrap;">в буквенных
    </span> домах.
</p>
```

Свойство `white-space` определяет, как ведут себя символы пространства (пробелы, табуляция, разрывы строки и пр.).

Символ разрыва строки вставляется, если нажать Enter на клавиатуре. Перенос при этом будет виден в редакторах кода, но в браузере перенос будет только внутри тега `<pre>` и в особых значениях свойства `white-space`. Возможные значения:

- **`normal`** – значение по умолчанию. Любая подряд идущая последовательность символов пространства схлопывается в один пробел. Символы разрыва строки не переносят текст;
- **`nowrap`** – любая подряд идущая последовательность символов пространства схлопывается в один пробел. Символы пространства не вызывают потенциального переноса текста. Перенос можно осуществить только с помощью тега `<br>`;

- **pre** – подряд идущие символы пространства не схлопываются. Символы пространства не вызывают потенциального переноса текста. Аналогично работе тега <pre>;
- **pre-line** – работает, как normal (пробелы схлопываются в один), но символы разрыва строки приводят к переносу;
- **pre-wrap** – работает, как pre, но текст переносится автоматически, если не помещается в строку. Свойство может быть применено ко всем типам элементов.

Иногда возникают ситуации, когда текст значительно превышает ширину родительского блока, и может случиться так, что появится запись с очень длинной фамилией. Данная ситуация называется «переполнение». Конечно, переполнение контента следует скрыть при помощи правила overflow: hidden, но может случиться так, что совершенно непозволительно обрежутся посередине буквы. Чтобы такое не произошло, предусмотрено свойство переполнения текста text-overflow. Единственным (помимо значения по умолчанию clip) значением является ellipsis. Оно помещает символ многоточия в месте, где текст «обрезается».

Таким образом, пользователю становится более очевидно, что в ячейке присутствует какой-то длинный текст. При этом если выделить такое слово вместе с символом многоточия и скопировать в буфер обмена, то слово скопируется целиком, что удобно.

Псевдоэлементы текста

CSS позволяет задать оформление первой строки блока с помощью специального псевдоэлемента ::first-line. Возможности оформления при этом ограничиваются в основном видоизменением свойств шрифта и текста и представлены следующими свойствами: font properties, color properties, background properties, word-spacing, letter-spacing, text-decoration, vertical-align, text-transform, line-height, clear.

Например:

```
<style>
p::first-line{
     color: red;
}
</style>
```

```
<p>
    Первая строка красная.<br>Вторая строка черная.
</p>
```

Псевдоэлемент `::first-letter` позволяет указать стили для первой буквы блока.

```
<style>
p::first-letter{
    color: #fff;
    background: #f00;
    padding: 0 4px;
    margin: 0 2px 0 0;
}
</style>

<p>
    Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Aenean commodo ligula eget dolor.
</p>
```

В отличие от `::first-line` в селекторе можно также указывать отступы, бордюры, обтекания и некоторые другие свойства.

4.2 Позиционирование

Очень часто при верстке веб-страниц возникает задача создать или расположить элементы необычным способом, при этом воспользоваться нормальным потоком в этой ситуации не представляется возможным. Тогда на помощь приходит метод позиционирования элементов.

Всего в CSS есть четыре варианта позиционирования:

- `static` – вариант, используемый по умолчанию: элементы участвуют в нормальном потоке, все манипуляции с ними влияют на рядом расположенные элементы;
- `relative` – относительное позиционирование – промежуточный вариант между `static` и `absolute`: элементы можно сдвигать с помощью свойств `top`, `left`, `right` и `buttom` (при этом образующееся пустое место не заполняется), а также менять порядок наложения этого элемента с помощью свойства `z-index` (переместить на задний план или, наоборот, ближе к зрителю);

- `absolute` – абсолютное позиционирование – полная противоположность `static`: элемент выпадает из потока и располагается относительно ближайшего родительского элемента со значением свойства `position` отличным от `static` (по умолчанию это элемент `body`). Для этого элемента становятся доступны свойства: `top`, `left`, `right`, `buttom`, `z-index`;
- `fixed` – фиксированное позиционирование – вариант позиционирования, при котором элемент выпадает из потока и фиксируется относительно окна браузера, причем его положение не меняется даже при прокрутке;
- `sticky` – «прилипающее» позиционирование. Этот вариант возник совсем недавно на фоне популярности различных «прилипающих» элементов веб-страницы (например, шапки или подвала). Это значит, что пока элемент не пересечет некоторую невидимую линию, он остается абсолютно позиционированным. После пересечения – фиксированным.

Рассмотрим подробнее механизм расположения элемента при относительном позиционировании. Положение блока вычисляется относительно нормального потока. После этого блок смещается относительно своего нормального расположения и при этом не влияет на позиционирование других блоков. Когда блоку задано относительное позиционирование, то его положение считается так, будто смещения не было.

Одним из самых популярных применений относительного позиционирования является создание *содержащего блока*.

Содержащий блок *– это пространство, в пределах которого ведется расчет позиционирования элементов. Таким блоком является ближайший элемент со значением свойства* `position absolute`, `relative`, `fixed` *или* `sticky`.

Сдвиги задаются с помощью свойств `top`, `left`, `right`, `bottom` – вверх, влево, вправо и вниз соответственно. Для относительного позиционирования нельзя задать одновременно `left` и `right`, `top` и `bottom` – будут применены только `left` и `top`. Также если `left` или `right auto`, то это эквивалентно значению `left/right = 0`, у `top/bottom` работает то же правило. Стоит за-

метить еще один нюанс использования свойств `top` и `bottom`: если содержащему блоку не задана высота, то значение этих свойств в процентах будет равно нулю [14].

4.3 Контекст наложения

Одним из интересных свойств позиционированных элементов является возможность изменять их расположение по z-оси. Таким образом можно добиться создания эффекта объема, но также могут возникнуть и некоторые проблемы, связанные с непониманием механизма работы этого свойства. Порядок наложения регулируется с помощью свойства `z-index`: чем больше значение свойства, тем ближе он по отношению к зрителю.

Что же такое контекст наложения? Это концепция расположения HTML-документов вдоль оси Z по отношению к пользователю, находящемуся перед экраном. HTML-элементы занимают это место по порядку, основанному на атрибутах элемента.

Чтобы создать контекст, нужно выполнение одного из условий:

- элемент является корневым (`<html></html>`);
- элементу задано абсолютное позиционирование (`position: absolute`) или относительное позиционирование (`position: relative`) с z-index значением, отличным от `auto`;
- flex-элемент с z-index, отличным от `auto`, чей родительский элемент имеет свойство `display: flex|inline-flex`;
- элементы с `opacity` меньше чем 1;
- элементы с `transform`, отличным от `none`;
- элементы с `mix-blend-mode` значением, отличным от `normal`;
- элементы с `filter`, значение которого отлично от `none`;
- элементы с `isolation`, установленным в `isolate`;
- элементу задано фиксированное позиционирование;
- у элемента установлен атрибут `will-change`;
- элементы с `-webkit-overflow-scrolling`, установленным в `touch` [8].

Контекст наложения влияет на потомков, а значит, порядок наложения будет работать только между соседними элементами, и при этом перекрываться элементами с z-index большим, чем у их родителя. Чтобы представить себе это визуально, рассмотрим рисунок 4.1. Здесь блок `div#4` является дочерним эле-

ментом `div#3`, поэтому перекрывается элементом `div#1`, хотя его z-index больше, а все потому, что родительский элемент `div#3`, создающий контекст наложения, имеет значение z-index меньшее, чем его сосед `div#1`.

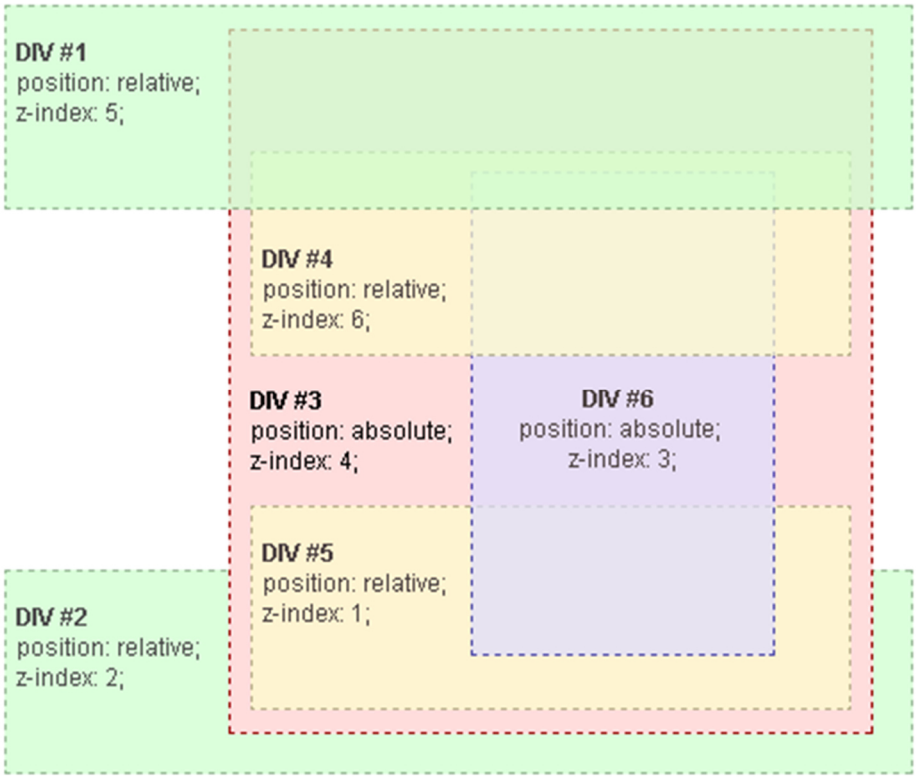


Рис. 4.1 – Элементы с разным контекстом наложения

Контрольные вопросы по главе 4

1. Что такое контекст наложения?
2. Что такое содержащий блок?
3. Какие типы шрифта существуют?
4. Что можно сделать, если текст слишком большой и не помещается в блок?
5. Как подключить нестандартный шрифт? Сформулируйте правило `@font-face`.
6. Какое минимальное количество шрифтов нужно подключить к странице, чтобы покрыть большую часть популярных браузеров?

5 Каскадная таблица стилей

CSS *(Cascading Style Sheets – каскадные таблицы стилей) – формальный язык описания внешнего вида документа, написанного с использованием языка разметки. Преимущественно используется как средство описания, оформления внешнего вида веб-страниц, написанных с помощью языков разметки HTML и XHTML, но может также применяться к любым XML-документам, например к SVG или XUL.*

В ранние годы Всемирной паутины (1990–1993 гг.) HTML был довольно бедным языком. Он практически целиком состоял из структурных элементов, полезных для описания абзацев, гиперссылок, списков и заголовков. В нем не было ничего, даже отдаленно напоминающего таблицы, фреймы или сложную разметку – того, что считается абсолютно необходимым для создания веб-страниц. HTML изначально задумывался как структурный язык разметки, применяемый для описания различных частей документа. О том, как должны отображаться эти части, говорилось совсем немного. Язык не затрагивал описание внешнего вида, он был лишь небольшой схемой разметки.

С нарастающей популярностью Веба появлялась необходимость большего управления в оформлении представлений. Так появились атрибуты и теги, призванные менять внешний вид документа.

```
<font size="+3" face="Helvetica" color="red">Заголовок страницы</font>
```

При этом тег **** никак не влияет на структуру документа. Содержимое страниц стало превращаться в смесь полезных данных и «полезного» оформления.

Пример кода без оформления:

```
<body>
    <h1>Основной заголовок</h1>
    <p>Текст параграфа 1</p>
    <h2>Подзаголовок 1</h2>
    <p>Текст параграфа 2</p>
</body>
```

Та же самая разметка, но уже с добавленными тегами оформления:

```
<body bgcolor="#f1f1f1">
    <h1 align="center"><font size="16px" color="red" face="Tahoma">Основной заголовок</font></h1>
    <p><font size="8px" color="gray" face="Arial">Текст параграфа 1</font></p>
    <h2><font size="10px" face="Tahoma">Подзаголовок 1</font></h2>
    <p><font size="8px" color="gray" face="Arial">Текст параграфа 2</font></p>
</body>
```

Конечно, проблема загрязнения HTML-разметкой представления не осталась незамеченной W3C, который приступил к поиску решения. В 1995 г. консорциум начал публикацию рабочего варианта стандарта, наз-ванного CSS. К 1996 г. он получил статус рекомендации, такой же значимой, как и сам HTML, причиной этого стал ряд достоинств:

- обеспечение «богатого» представления документа;
- простота применения;
- возможность применения стилей к нескольким страницам.

Но главной заслугой CSS стало разделение описания логической структуры от описания внешнего вида. Посмотрим на нашу разметку после добавления CSS.

```
<body>
    <h1>Основной заголовок</h1>
    <p>Текст параграфа 1</p>
    <h2>Подзаголовок 1</h2>
    <p>Текст параграфа 2</p>
</body>
<style>
    body {
        background: #f1f1f1;
    }
    h1 {
        font-size: 34px;
        color: red;
        font-face: Tahoma;
        text-align: center;
    }
```

```
        h2 {
             font-size: 24px;
             font-face: Tahoma;
        }
        p {
             font-size: 18px;
             color: gray;
             font-face: Arial;
        }
</style>
```

5.1 Селекторы

Одно из основных преимуществ CSS (особенно для разработчиков) – это возможность легко применять набор стилей (правил) ко всем однотипным элементам в документе. Отредактировав всего одну строку CSS, можно, например, изменить цвет всех заголовков в документе.

Чтобы лучше понять, как формируются правила, давайте разберем их структуру. Каждое правило имеет две основные части: селектор и блок объявлений. Блок объявлений состоит из одного или более объявлений, а каждое объявление представляет собой сочетание свойства и значения. Каждая таблица стилей образуется из наборов правил.

Селектор, расположенный в левой части правила, определяет, на какие элементы документа распространяется правило. На рисунке 5.1 выбраны элементы `h1`. Если бы селектором был `p`, то выбирались бы все элементы `p` (абзацы). В правой части правила находится блок объявлений, образованный одним или несколькими объявлениями. Каждое объявление представляет собой сочетание свойства CSS и значения этого свойства. На рисунке представлен блок, содержащий два объявления. Первое определяет, что согласно правилу цвет (`color`) указанных элементов документа будет синим (`blue`), а второе, что размер шрифта (`font-size`) этих элементов будет 12px. Таким образом, все элементы `h1` (определенные селектором) этого документа будут выводиться синим текстом размером в 12px (рис. 5.1).

В блок объявлений входит одно или несколько объявлений. Формат объявлений обычно такой: имя свойства, за которым следует двоеточие, затем значение и точка с запятой. После двоеточия и точки с запятой может быть произвольное количество пробелов (в частности, возможно отсутствие пробела). Практически во всех случаях значение – это или отдельное ключевое слово, или

список из нескольких допустимых для данного свойства ключевых слов, разделенных пробелами. Если указать неверное свойство или значение, будет проигнорировано все объявление целиком. Поэтому следующие два объявления не будут выполнены:

```
font-saze: 12px; /* неизвестное значение (правильно font-size) */
color: ultraviolet; /* неизвестное значение (нет такого цвета) */
```

Рис. 5.1 – Селектор

Бывают случаи, когда допускается указывать в качестве значения свойства более одного ключевого слова, обычно их разделяют пробелами. Не все свойства могут принять несколько ключевых слов. Для примера рассмотрим свойство `font`.

```
h1 { font: italic 40px Tahoma; } /* три значения для одного свойства */
```

Необходимо обратить внимание на пробел между значениями `italic 40px` и `Tahoma`, каждое из которых является частью значения свойства `font` (первое – начертание шрифта, второе – размер шрифта, а третье – фактическое имя шрифта). Благодаря пробелу браузер различает этот набор значений и применяет их. Точка с запятой указывает на завершение объявления.

Группировка селекторов

Если необходимо, чтобы текст элементов `h1` и абзацев был серого цвета, это достигается посредством следующего объявления:

```
h1, p { color: gray; }
```

Поместив в левую часть разделенные запятой селекторы `h1` и `p`, определяется правило, в котором находящийся в правой части стиль (`color: gray;`) применяется к элементам, обозначенным обоими селекторами. Запятая сообщает браузеру о том, что в правило включены два разных селектора. Если запятую опустить, правило приобретет совершенно другое значение (такой селектор станет селектором потомка)!

Универсальный селектор

Универсальный селектор (universal selector) записывается символом «звездочка» (`*`). Этот селектор соответствует любому элементу почти так же, как подстановочный символ в маске имени файла. Например, чтобы сделать все элементы документа красными, можно написать:

```
* { color: red; }
```

Это описание эквивалентно групповому селектору, в котором перечислен каждый содержащийся в документе элемент (как будто каждому элементу документа назначили значение red). Хотя универсальный селектор удобен, его не рекомендуют к применению, т. к. он выбирает абсолютно все элементы. К тому же использование универсального селектора приводит к снижению скорости рендеринга страницы.

Селекторы классов

Кроме простых селекторов элементов документа еще есть селекторы классов (class selectors) и селекторы идентификаторов (ID selectors), позволяющие назначать стили элементам независимо от их типа. Эти селекторы могут применяться самостоятельно или в сочетании с селекторами элементов. Однако работают они только в том случае, если документ размечен соответствующим образом, поэтому их применение подразумевает некоторое предварительное планирование.

Самый распространенный способ применения стилей без учета элементов состоит в том, чтобы обратиться к селектору класса. Однако сначала придется изменить разметку документа таким образом, чтобы обеспечить возможность работы этого селектора.

Чтобы связать стили селектора класса с элементом, необходимо присвоить соответствующее значение атрибуту `class` данного элемента. В CSS возможна очень краткая запись, в которой имени класса предшествует точка.

```
.warning { font-weight: bold; color: crimson; }
```

Ранее были рассмотрены примеры, где значения атрибута `class` состояли из одного слова. В HTML значением `class` может быть и разделенный пробелами список слов. Например, если Вы хотите обозначить конкретный элемент и как важное сообщение, и как предупреждение, можно было бы написать:

```
.strong.attention { font-weight: bold; color: crimson; }
```

Порядок слов на самом деле не имеет значения. Подошло бы и `attention strong`. Для того чтобы все элементы, у которых атрибут `class` имеет

значение strong, были выделены полужирным шрифтом, а те элементы, атрибут class которых имеет значение attention, были выделены красным цветом, а элементы, имеющие оба значения, получили желтый фон, это должно быть записано следующим образом:

```
.strong { font-weight: bold; }
.attention { color: crimson; }
.strong.attention { background: yellow; }
```

Объединяя два селектора класса, можно выбрать только те элементы, которые имеют оба имени класса, стоящие в любом порядке.

Селекторы идентификаторов

В некотором смысле селекторы идентификаторов аналогичны селекторам классов, но есть два существенных отличия. Во-первых, перед селекторами идентификаторов вместо точки ставится «решетка» (#). Таким образом, возможно и такое правило:

```
#first { font-weight: bold; }
```

Оно устанавливает полужирный шрифт для любого элемента, у которого атрибут id имеет значение first. Второе отличие – вместо значений атрибута class в селекторах идентификаторов используются значения атрибутов id.

Назначать классы можно любому количеству элементов. Идентификаторы в HTML-документе используются только один раз, поэтому если в документе есть элемент со значением атрибута id, ни один другой элемент этого документа не может (на самом деле может, и CSS будет работать нормально, но тогда идентификатор теряет свой основной смысл) иметь id с таким же значением.

В отличие от селекторов класса селекторы идентификаторов не могут объединяться, поскольку в атрибуты id нельзя помещать разделенный пробелами список.

Еще одно отличие между именами class и id состоит в том, что идентификаторы имеют больший вес.

5.2 Селекторы атрибутов

И в селекторах классов, и в селекторах идентификаторов речь на самом деле идет о выборе значений атрибутов. Селектор .strong применит правила к тем же элементам, что и селектор [class="strong"].

Для того чтобы выбрать элементы с определенным атрибутом независимо от значения этого атрибута, можно обратиться к простому селектору атрибутов. Например, чтобы выбрать все элементы, имеющие атрибут `title` с любым значением, и сделать их текст красным, можно написать такое правило:

```
[title] { color: red; }
```

В дополнение к выбору элементов по атрибутам можно еще более сузить выбор, чтобы охватить только те элементы, атрибуты которых имеют определенное значение. Например, можно выделить полужирным шрифтом ссылки, указывающие на главную страницу сайта.

```
[href="https://ya.ru/"] { color: red; }
```

Поиск элементов по атрибуту не ограничивается точным совпадением. Можно написать селектор, который будет искать совпадение определенной части атрибута. Рассмотрим вариант, когда ссылка на страницу может быть по протоколам http и https. В этом случае селектор на основании конкретного значения не найдет все ссылки на страницу, а вот селектор на основании частичного поиска справится с этой задачей:

```
[href*="ya.ru"] { color: red; text-decoration: none; }
```

Необходимо обращать внимание на наличие в селекторе звездочки (`*`). Это ключ для осуществления выбора на основании частичного совпадения. Если пропустить звездочку, то получится требование точного соответствия конкретному значению.

Селекторы атрибутов по подстроке

`[href^="https"]` выбирает любой элемент, значение атрибута `href` которого начинается с `"https"`.

`[href$="ru"]` выбирает любой элемент, значение атрибута `href` которого заканчивается `"ru"`.

`[class~="strong"]` выбирает любой элемент, значение атрибута `class` которого содержит `"strong"` как отдельное слово.

Селекторы псевдоклассов

Псевдоклассы определяют динамическое состояние элементов, которое изменяется с помощью действий пользователя, а также положения в дереве документа [9].

Есть два типа псевдоклассов: динамические и структурные.

Примером динамического псевдокласса состояния служит текстовая ссылка, которая меняет свой цвет в зависимости от того, была ли ранее посещена страница, на которую указывает ссылка. При этом HTML никак не изменяется, потому что в нем нет никаких указателей на то, что по ссылке уже был переход.

:link ссылается на любую гиперссылку (т. е. имеющую атрибут `href`) и указывает на адрес, который не был посещен.

:visited ссылается на любую гиперссылку, указывающую на уже посещенный адрес.

```
a { color: red; }
a:visited { color: gray; }
```

CSS поддерживает псевдоклассы, которые могут изменять внешний вид документа в результате действий пользователя. Эти динамические псевдоклассы традиционно применяются для оформления ссылок и кнопок, но их возможности намного шире.

:focus соответствует элементу, которому в настоящий момент принадлежит фокус ввода, т. е. который готов принимать ввод с клавиатуры или быть активированным некоторым образом.

```
input:focus { background: blue; }
```

:hover соответствует элементу, над которым размещен указатель некоторого устройства, например ссылка, по которой проводят курсором мыши.

```
a:hover { color: red; }
```

:active соответствует элементу, который был активирован пользователем, например кнопка, по которой щелкает пользователь в течение того времени, когда удерживается кнопка мыши.

```
button:active { background: green; }
```

:target соответствует элементу, на который можно перейти по хеш-ссылке.

```
section:target { background: red; }
```

Для примера рассмотрим меню, которое помогает переходить к нужному разделу на странице. Каждая ссылка содержит хеш, который является связывающим ключом с определенной секцией. По клику на ссылку соответствующая область будет выделена красным цветом.

Список псевдоклассов состояния:

- **:enabled** соответствует элементам форм, которые являются доступным (незаблокированным) для изменения состояния. По умолчанию,

все элементы форм являются доступными, если в коде HTML к ним не добавляется атрибут `disabled`;

- **`:disabled`** соответствует элементам форм, которые не являются доступным для изменения состояния;
- **`:checked`** соответствует элементам форм, таким как переключатели (checkbox) и флажки (radio), когда они находятся в положение «включено»;
- **`:indeterminate`** соответствует элементам форм, таким как переключатели (checkbox) и флажки (radio), когда они находятся в неопределенном состоянии.

Структурные псевдоклассы позволяют выбирать элементы в зависимости от их положения в дереве элементов.

Псевдокласс **`:first-child`** применяется для выбора элементов, представляющих собой первые дочерние элементы других элементов. Для примера напишем селектор, который будет выбирать только первый элемент в списке.

```
li:first-child { color: green; }
```

Данный код можно прочитать справа налево как «выберем каждый первый элемент `li`, который является дочерним по отношению к элементу `ul`».

Самая распространенная ошибка – полагать, что такой селектор, как **`li:first-child`**, выберет первый дочерний элемент элемента `li`. На самом деле он выберет элемент `li`, причем только в том случае, если этот элемент является первым дочерним элементом по отношению к своему родительскому элементу.

Список структурных псевдоэлементов:

- **`:root`** – соответствует корневому элементу документа. В HTML этот селектор всегда соответствует элементу;
- **`:nth-child()`** – соответствует элементам на основе заданной нумерации в дереве элементов;
- **`:nth-last-child()`** – работает так же, как и `:nth-child()`, но в отличие от него отсчет ведется не от первого элемента, а от последнего;
- **`:nth-of-type()`** – соответствует элементам указанного типа на основе нумерации в дереве элементов;

- **:nth-last-of-type()** – работает так же, как и :nth-of-type(), но в отличие от него отсчет ведется не от первого элемента, а от последнего;
- **:first-child** – соответствует первому дочернему элементу своего родителя;
- **:last-child** – соответствует последнему дочернему элементу своего родителя;
- **:first-of-type** – соответствует первому элементу указанного типа в списке дочерних элементов своего родителя;
- **:last-of-type** – соответствует последнему элементу указанного типа в списке дочерних элементов своего родителя;
- **:only-child** – соответствует дочернему элементу, только если он единственный у родителя;
- **:only-of-type** – соответствует дочернему элементу указанного типа, только если он единственный у родителя;
- **:empty** – соответствует пустому элементу (которые не содержат дочерних элементов, текста или пробелов).

Селекторы псевдоэлементов

Почти так же, как псевдоклассы назначают фантомные классы для ссылок, псевдоэлементы вводят фиктивные элементы в документ, чтобы достигнуть определенных эффектов.

Псевдоэлемент **:first-letter** участвует в стилевом оформлении первой буквы блочного элемента.

```
p:first-letter { color: red; }
```

Согласно этому правилу первая буква каждого абзаца будет окрашена в красный цвет.

Аналогичным образом **:first-line** может применяться для оформления первой строки текста элемента. Например, можно сделать первую строку каждого абзаца документа красной:

```
p:first-line { color: red; }
```

Применение специальных стилей до и после элементов.

В CSS существует возможность добавлять контент до (:before) и после (:after) содержимого элементов. Для примера добавим строки «начало» и «конец» элементу body:

```
body:before { content: '=начало='; color: green; }
body:after { content: '=конец='; color: red; }
```

Обратите внимание на то, что псевдоэлементы `before` и `after` можно применять только к парным тегам.

Мощь CSS базируется на родительско-дочерних отношениях (parent-child relationship) элементов. HTML-документы строятся на основании иерархии элементов, которую можно показать в форме древовидного представления документа.

В этой иерархии каждый элемент тем или иным образом вписывается в общую структуру документа. Каждый элемент является или родительским (parent), или дочерним (child) элементом другого элемента, а зачастую выполняет обе эти роли. Элемент является родителем другого элемента, если в иерархии документа он находится прямо над этим элементом. Например, первый элемент `p` является родителем для элементов `em` и `strong`, тогда как `strong` – родитель элемента `a`, который в свою очередь является родителем другого элемента `em`. И наоборот, элемент является дочерним элементом другого элемента, если он находится прямо под этим элементом. Таким образом, элемент `a` – это дочерний элемент элемента `strong`, который в свою очередь является потомком элемента `p` и т. д.

Термины «родительский элемент» и «дочерний элемент» – это частные случаи терминов «предок» (ancestor) и «потомок» (descendant). Между ними существует разница: если в представлении в виде древовидного списка элемент находится ровно на один уровень выше другого, между ними существуют родительско-дочерние отношения. Если путь от одного элемента к другому пересекает два или более уровней, между элементами существуют отношения «предок – потомок», но не родительско-дочерние (конечно, дочерний элемент также является потомком, а родитель – предком).

На рисунке 5.2 первый элемент `ul` – это родитель двух элементов `li`, но первый `ul` также является предком всех элементов, происходящих от его элемента `li`, вплоть до самых глубоко вложенных элементов `li`. Также на рисунке показан элемент `a`, который является дочерним по отношению к `strong`, но еще и потомком абзаца, элементов `body` и `html`. Элемент `body` – предок всего, что браузер будет отображать по умолчанию, а элемент `html` – предок всего документа. Поэтому элемент `html` также называют корневым элементом (root element).

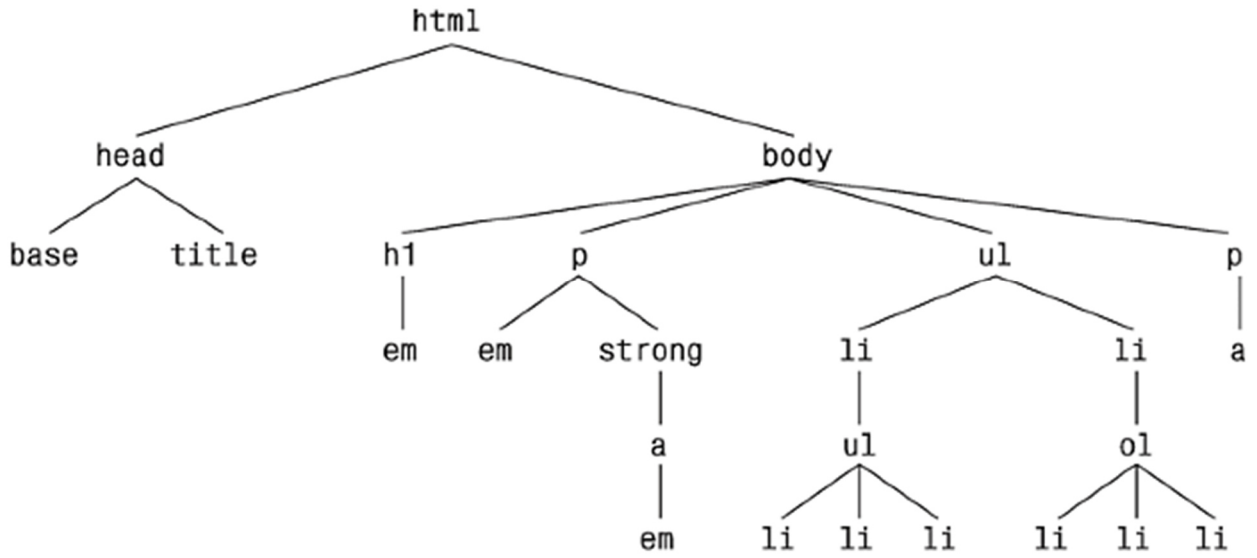


Рис. 5.2 – Пример наследования

Селекторы потомков

Определение селекторов потомков – это создание правил, действующих лишь в рамках заданной структуры. Например, требуется задать стили только для тех элементов `a`, которые происходят от элемента `nav`. Для этого можно объявить правила, которые соответствуют только элементам `a`, находящимся внутри элементов `nav`.

```
nav a { color: red; text-decoration: none; }
```

Это правило сделает цвет ссылок, которые являются потомками элемента `nav`, красным. Цвет других элементов `a`, например находящихся в абзаце или блоке, не будет выбран этим правилом.

В селекторе потомков часть правила, соответствующая селектору, состоит из двух или более разделенных пробелами селекторов. Пробел между селекторами – это пример комбинатора (combinator). Каждый комбинатор – пробел, если читать его справа налево, он может быть истолкован как «находящийся в», «который представляет собой часть» или «являющийся потомком». Таким образом, `nav a` можно прочитать как: «любой элемент `a`, который является потомком элемента `nav`». (Если прочитать селектор слева направо, может получиться примерно следующее: «для любого `nav`, содержащего `a`, к `a` будут применены следующие стили».) Конечно, два селектора не предел. Например:

```
article p a abbr { color: red; text-decoration: none; }
```

В некоторых случаях не требуется выбирать любой элемент-потомок. Напротив, необходимо сузить диапазон выбора до дочернего элемента другого элемента. Предположим, надо выбрать элемент `strong`, только если он являет-

ся дочерним элементом (а не просто потомком) элемента h1. Для этого используется символ-комбинатор селектора дочерних элементов, которым является символ «больше» (>):

```
h1 > strong { color: red; }
```

Это правило сделает красным элемент strong для первого из следующих далее h1 и не сделает для второго:

```
<h1>Это <strong>очень</strong> важно.</h1>
<h1>Это <em>действительно <strong>очень</strong></em> важно.</h1>
```

Прочитанный справа налево селектор h1 > strong имеет значение «выбираем любой элемент strong, являющийся дочерним элементом h1». Комбинатор селектора дочерних элементов может быть окружен пробелами. Таким образом, селекторы: h1 > strong и h1 > strong эквивалентны.

На этом фрагменте дерева можно без труда выделить родительско-дочерние отношения. Например, элемент a – родитель элемента strong, он же и дочерний элемент элемента p. Можно было бы сопоставить элементы этого фрагмента с селекторами p > a и a > strong, но не с селектором p > > strong, поскольку strong является потомком p, а не дочерним элементом.

Рассмотрим еще один пример. Предположим, у нас есть блок комментариев и нам необходимо сделать внутри каждого комментария красными только те ссылки, которые являются прямыми потомками.

```
article > a { color: red; }
```

Для того чтобы выбрать элемент, который расположен сразу за другим элементом и имеет того же родителя, применяется комбинатор селектора соседних элементов (adjacent-sibling combinator), представляемый в виде знака «плюс» (+). Как и комбинатор селектора дочерних элементов, этот символ может быть окружен пробелами.

Чтобы удалить верхний отступ абзаца, следующего за элементом h1, запишем:

```
h1 + p { margin-top: 0; }
```

Этот селектор означает следующее: «выбираем любой абзац, расположенный за элементом h1, имеющий общих родителей с элементом p». Наглядно представить себе, как работает этот селектор, проще всего, еще раз рассмотрев фрагмент дерева документа.

```
<div>
     <ul>
          <li></li>
          <li></li>
          <li></li>
     </ul>
     <ol>
          <li></li>
          <li></li>
          <li></li>
     </ol>
</div>
```

В этом фрагменте от элемента `div` происходит пара списков, один нумерованный, а другой нет, каждый из которых содержит по три элемента списка. Списки представляют собой сестринские элементы, и сами элементы списков тоже сестринские элементы. Однако элементы первого списка не являются сестринскими для элементов второго списка, поскольку их родительские элементы разные.

```
li + li { font-weight: bold; }   /* выбрать li, перед которым есть li */
ol + ul { background: yellow; }  /* выбрать ol, перед которым есть ul */
```

Из двух сестринских элементов одним комбинатором выбирается только второй элемент. Поэтому если Вы записываете:

```
li + li { font-weight: bold; }
```

полужирным шрифтом будут выделены только второй и третий элементы каждого списка. Первые элементы списков останутся нетронутыми.

Для обеспечения правильной работы CSS требует, чтобы два элемента были приведены в «исходном порядке». В данном примере за элементом `ol` следует элемент `ul`. Это позволяет выбрать второй элемент с помощью селектора `ol + ul`, но первый элемент посредством аналогичного синтаксиса выбрать не удастся.

Рассмотрим другой пример. Допустим, что необходимо разделить комментарии пользователей красной линией, при этом необходимо выделить весь блок комментариев черными линиями (перед первым и после последнего комментария).

```
h3 + article { border-top: 2px solid black; }
/* добавляем черную двухпиксельную линию для article, перед которым есть h3 */

article + article { border-top: 1px solid red; }
/* добавляем красную однопиксельную линию для article, перед которым есть article */

article + footer { border-top: 2px solid black; }
/* добавляем черную двухпиксельную линию для footer, перед которым есть article */
```

Чтобы выбрать все элементы определенного типа, расположенные на одном уровне с другим элементом и имеющие того же родителя, применяется комбинатор селектора сестринских элементов (General sibling combinator), представляемый в виде знака «тильда» (~).

Селектор выбора сестринских элементов очень похож на селектор выбора соседних элементов с той лишь разницей, что селектор выбора сестринских элементов выбирает не соседний элемент, а все последующие.

Для примера представлена форма регистрации, в которой текстовые поля будут показаны только после перевода `checkbox` в состояние `checked`.

```
[type="text"] { visibility: hidden;}
/* по умолчанию все текстовые поля скрыты*/
[type="checkbox"]:checked ~ [type="text"] { visibility: visible; }
/* показываем все текстовые поля, которые расположены после checkbox и на одном уровне с ним, когда checkbox в состоянии checked */
```

Селектор отрицания

Псевдокласс `:not` соответствует элементам, которые не содержат указанный селектор. Допустим, есть таблица, и необходимо задать зеленый фон ячейкам, которые содержат внутри текст и красный фон ячейкам, которые пустые.

```
td:empty { background: red; } /* красный фон для пустых ячеек */
td:not(:empty) { background: green; } /* зеленый фон для ячеек, которые не являются пустыми */
```

5.3 Специфичность

Как было отмечено выше, существует множество способов выбора необходимых элементов. Фактически один и тот же элемент может быть выбран двумя и более правилами, каждое из которых имеет собственный селектор.

```
h1 { color: red; }
body h1 { color: green; }
```

Очевидно, что применено будет только одно из двух правил каждой пары, поскольку сопоставляемый элемент может быть только одного цвета. А как узнать, какое из правил применится?

Ответ кроется в специфичности (specificity) каждого селектора. Для каждого правила браузер вычисляет специфичность селектора (его вес) и прикрепляет ее к каждому объявлению правила. Если элемент имеет несколько конфликтующих объявлений одного свойства, выигрывает то, которое имеет наибольшую специфичность.

Специфичность селектора определяется компонентами самого селектора. Значение специфичности состоит из четырех частей: `0, 0, 0, 0`. Реальная специфичность селектора определяется следующим образом:

- для каждого указанного в селекторе значения идентификатора к специфичности добавляется `0, 1, 0, 0`;
- для каждого указанного в селекторе имени класса, псевдокласса или атрибута к специфичности добавляется `0, 0, 1, 0`;
- для каждого заданного в селекторе элемента и псевдоэлемента к специфичности добавляется `0, 0, 0, 1`.

Универсальный селектор не учитывается.

```
h1 { color: red; }  /* специфичность = 0, 0, 0, 1 */
p em { color: purple; }  /* специфичность = 0, 0, 0, 2 */
.grape { color: purple; }  /* специфичность = 0, 0, 1, 0 */
* .bright { color: yellow; }  /* специфичность = 0, 0, 1, 0 */
p.bright em.dark { color: maroon; }  /* специфичность = 0, 0, 2, 2 */
#id216 { color: blue; }  /* специфичность = 0, 1, 0, 0 */
div#sidebar [href] { color: silver; }  /* специфичность = 0, 1, 1, 1 */
* { color: yellow; }  /* специфичность = 0, 0, 0, 0 */
h1 {color: red;}      /* 0, 0, 0, 1 */
```

```
body h1 {color: green;}  /* 0, 0, 0, 2 (победитель) */
```

Вес значения растет слева направо. Специфичность `1, 0, 0, 0` возьмет верх над любой специфичностью, которая начинается с `0`, независимо от того, какими будут остальные числа. Таким образом, `0, 1, 0, 1` выигрывает у `0, 0, 1, 7`, потому что `1`, стоящая на втором месте первого значения, побеждает `0` на этом месте во втором значении.

Чтобы определить специфичность, браузер пользователя должен рассматривать правило так, как будто бы оно разделено на отдельные «разгруппированные» правила.

```
h1, h2.section { color: silver; background: black; }

h1 { color: silver; background: black; } /* 0, 0, 0, 1 */
h2.section { color: silver; background: black; } /* 0, 0, 1, 1 */
```

Важно понимать разницу в специфичности селектора идентификатора (ID selector) и селектора атрибутов (attribute selector), в котором указан атрибут `id`.

```
#uniq { color: green; }      /* 0, 1, 0, 0 */
[id='uniq'] { color: red; }  /* 0, 0, 1, 0 */
```

Все рассмотренные до сих пор значения специфичности начинались с нуля. Дело в том, что этот первый нуль зарезервирован для встроенных объявлений стилей, специфичность которых превосходит специфичность всех остальных объявлений. Рассмотрим следующее правило и фрагмент разметки:

```
<style>
    #uniq { color: green; }      /* 0, 0, 0, 1 */
    [id='uniq'] { color: red; }  /* 0, 1, 0, 0 */
</style>

<h1 id="uniq" style="color: blue;">CSS</h1> /* 1, 0, 0, 0 */
```

Исходя из того, что это правило применяется к элементу `h1`, можно ожидать, что текст `h1` будет синим. Так и происходит, объясняется это тем, что специфичность данного объявления равна `1, 0, 0, 0`. Это значит, что даже элементы с атрибутами `id`, которые сопоставляются с правилом, подчиняются встроенным в тег стилям.

Иногда важность объявления настолько велика, что перевешивает все остальные факторы. В CSS их называют важными объявлениями (important

declarations). Важность добавляется путем введения в объявление ключевого слова `!important` прямо перед завершающей точкой с запятой:

```
#uniq {
    color: green !important;
    font-size: 20px;
}
```

Здесь значение цвета #333 отмечено как `!important`, тогда как размер шрифта – нет. Если необходимо сделать важными оба объявления, каждому из них понадобится собственный маркер `!important`:

```
#uniq {
    color: green !important;
    font-size: 20px !important;
}
```

Ключевое слово `!important` всегда располагается в конце объявления, прямо перед точкой с запятой.

Объявления, отмеченные как `!important`, не имеют особого значения специфичности, они рассматриваются отдельно от остальных. Фактически все объявления `!important` группируются вместе, и тогда уже их конфликты специфичностей разрешаются относительно друг друга. Аналогично группируются все остальные объявления, и конфликты свойств разрешаются с помощью специфичностей. В любом случае, когда имеет место конфликт важного и неважного объявления, всегда побеждает важное.

5.4 Наследование

Наследование – это механизм, с помощью которого стили применяются не только к указанным элементам, но также и к их потомкам. Например, если цвет применен к элементу `p`, то этот цвет будет применен ко всему тексту в `p` и даже к тому, который заключен в дочерние элементы этого `p`.

```
<style>
    p { color: green; }
</style><p><strong>Каскадные таблицы стилей</strong> – мощный механизм</p>
```

И обычный текст `p`, и текст `strong` окрашены в зеленый цвет, потому что элемент `strong` наследует значение `color`. Если бы значения свойств не наследовались элементами-потомками, текст `strong` был бы черным, а не зеленым, и пришлось бы окрашивать этот элемент отдельно.

Лучше всего принцип работы наследования иллюстрирует древовидное представление документа. На рисунке 5.3 показана древовидная схема простого документа, содержащего два списка – ненумерованный и нумерованный. Когда к элементу `ul` применяется объявление `color: green`; он принимает это объявление. Затем это значение передается вниз по дереву элементов-потомков до тех пор, пока не останется потомков, которые могли бы наследовать это значение. Значения никогда не передаются вверх по иерархии, т. е. элемент никогда не передает значение своим предкам.

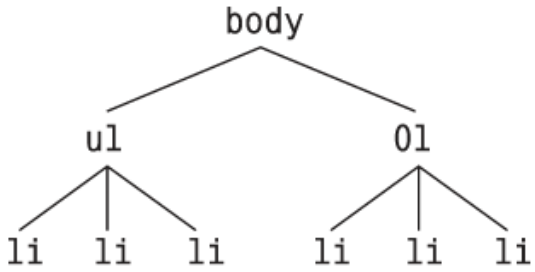


Рис. 5.3 – Древовидная схема простого документа с двумя списками

Но не все свойства наследуются. Это обусловлено тем, что наследование таких свойств (`border`, `padding` и др.) может создать много проблем.

С точки зрения специфичности унаследованные значения вообще не имеют веса, даже нулевого. Рассмотрим следующие правила и фрагмент разметки:

```
<style>
     * { color: gray; }
     p { color: green; }
</style>

<p><strong>Каскадные таблицы стилей</strong> (CSS – Cascading Style Sheets) …</p>
```

Поскольку универсальный селектор применяется ко всем элементам и имеет нулевую специфичность, заданный в нем серый цвет для элемента `strong` одерживает верх над унаследованным значением `green`, которое вообще не имеет никакой специфичности. Следовательно, цвет элемента `strong` будет серым, а не зеленым.

5.5 Каскад

Часто на практике можно встретить ситуацию применения правил с одинаковыми свойствами и специфичностью, но с разными значениями свойств.

```
h1 { color: red; }
h1 { color: blue; }
```

Какое из них победит? Специфичность обоих равна `0, 0, 0, 1`, у них равные шансы, и они оба должны быть применены. Для решения таких ситуаций существует каскад. CSS основывается на методе каскадирования стилей, реализация которого стала возможной благодаря сочетанию наследования и специфичности. Правила каскадирования очень просты:

1. Найти все правила, содержащие селектор, сопоставляемый с данным элементом.

2. Провести сортировку согласно явной приоритетности всех применяемых к элементу объявлений. Правилам, отмеченным как `!important`, присваивается более высокий приоритет, чем остальным. Все применяемые к данному элементу объявления сортируются согласно их источнику. Существует три возможных источника правил: автор, читатель и браузер. В общем случае стили автора приоритетней стилей читателя. Но стили читателя, отмеченные как `!important`, сильнее, чем все остальные стили, включая и те стили автора, которые отмечены как `!important`. И стили автора, и стили читателя замещают применяемые по умолчанию стили браузера.

3. Провести сортировку всех объявлений, применяемых к элементу, согласно их специфичности. Элементы с более высокой специфичностью имеют больший приоритет по сравнению с теми, специфичность которых ниже.

4. Провести сортировку всех объявлений, применяемых к элементу, в соответствии с очередностью расположения. Чем позже объявление появляется в таблице стилей или документе, тем больший приоритет ему присваивается. Считается, что объявления, находящиеся в импортированных таблицах стилей, располагаются перед всеми объявлениями импортировавшей их таблицы стилей.

Согласно второму правилу, если к элементу применяются два правила и одно из них отмечено как `!important`, то оно побеждает.

Несмотря на то что цвет задан в атрибуте `style` абзаца, побеждает правило с пометкой `!important`, и текст абзаца становится серым. Этот серый цвет также наследуется элементом `strong`. Более того, учитывается источник правила. Если с элементом сопоставляются стили с обычной приоритетностью из таблицы стилей автора и таблицы стилей читателя, то применяются стили автора. Предположим, что следующие стили происходят из указанных источников:

```
p strong { color: black; }   /* таблица стилей автора */
p strong { color: yellow; } /* таблица стилей читателя */
```

В данном случае выделенный текст параграфа закрашивается черным цветом, а не желтым, потому что стили автора с обычной приоритетностью побеждают обладающие обычным приоритетом стили читателя.

Однако если оба правила отмечены как `!important`, ситуация меняется:

```
p strong { color: black !important; }  /* таблица стилей автора */
p strong { color: yellow !important; }  /* таблица стилей читателя */
```

Теперь выделенный текст параграфа будет желтым, а не черным. Так сложилось, что применяемые по умолчанию стили браузера, которые обычно отражают предпочтения пользователя, учитываются именно так. Применяемые по умолчанию объявления стилей вообще относятся к категории правил, обладающей наименьшим влиянием. Поэтому если заданное автором правило применяется к тегам `a` (например, объявляет, что они будут белыми), то оно замещает стандартные настройки браузера.

С точки зрения приоритетности объявлений выделены пять уровней. В порядке уменьшения приоритетности это:

5. Важные объявления читателя.
4. Важные объявления автора.
3. Обычные объявления автора.
2. Обычные объявления читателя.
1. Объявления браузера.

Согласно третьему правилу, если к элементу применяются конфликтующие объявления и все они имеют одинаковую приоритетность, они должны сортироваться в соответствии со специфичностью. Побеждает объявление, обладающее наибольшей специфичностью.

```
<style>
    .content { color: green; }
    p { color: gray; }
</style>

<p class="content"><strong>Каскадные таблицы стилей</strong> (CSS - Cascading Style Sheets) ...?>
```

Исходя из приведенных правил текст параграфа будет окрашен в зеленый цвет, потому что специфичность `.content` (`0, 1, 0, 0`) превышает специ-

фичность `p` (`0, 0, 0, 1`), даже несмотря на то, что последнее правило расположено в таблице стилей позже.

И наконец, согласно четвертому правилу, если два правила имеют совершенно одинаковую приоритетность, источник и специфичность, тогда побеждает то, которое расположено в таблице стилей ниже. Вернемся к нашему предыдущему примеру, в котором мы нашли следующие два правила таблицы стилей документа:

```
h1 { color: red; }
h1 { color: blue; }
```

Значение `color` для всех элементов `h1` документа будет `blue`, а не `red`, поскольку именно это правило стоит в таблице стилей ниже.

5.6 Значения и единицы измерения

Единицы измерения (units) применяются во многих свойствах для задания цвета, расстояний и размеров. Без единиц измерения нельзя объявить о том, что текст абзаца должен быть нужного размера или что вокруг изображения должно быть пустое пространство в 10 пикселов.

Числа

В CSS существует два типа чисел: целые (integer) и вещественные (real). Эти типы чисел в большинстве случаев служат базой для всех остальных типов значений.

Процентное значение (percentage value) – это вычисляемое вещественное число, за которым следует знак процента (%). Процентные значения практически всегда выражены относительно другого значения, которым может быть все что угодно, включая значение другого свойства того же элемента, значение, унаследованное от родительского элемента, или значение элемента предка. Любое свойство, принимающее значения, задаваемые в процентах, определяет свои ограничения на допустимый диапазон процентных значений и точность представления относительных процентных значений. Для примера рассмотрим следующее правило:

```
p {
     font-size: 20px;
     width: 50%;
     line-height: 150%;
}
```

Ширина `p` будет равна половине ширины его родителя (`width: 50%`), потому что процентное значение ширины в данном случае рассчитывается от значения ширины родителя. В тоже время значение `line-height`, заданное как 150%, будет равно 30px, так как рассчитывается от своего собственного значения элемента `font-size`, которое равно 20px.

Цвета в CSS можно задавать различными способами. Самый интуитивно понятный способ задать значение цвета, это использовать ключевое слово, например red, green, black.

Существует два варианта задания цвета, основанных на функциональном формате записи RGB (functional RGB notation). Обобщенный синтаксис кодировки цвета – rgb(color), где color представляет собой комбинацию трех процентных значений или целых чисел. Допустимый диапазон процентных значений – от 0 до 100%, а диапазон целых значений – от 0 до 255. Следовательно, код, задающий белый и черный цвета с помощью процентных значений, будет таким: `rgb(100%, 100%, 100%) rgb(0%, 0%, 0%)`. А вот те же цвета, представленные записью из целых чисел (integertriplet notation): `rgb(255, 255, 255) rgb(0, 0, 0)`.

Модель RGBA – вариант модели RGB, но с дополнительным параметром (альфа-канал), который задает прозрачность цвета.

CSS позволяет определять цвет с помощью шестнадцатеричной записи. При такой форме записи цвет задается посредством объединения трех шестнадцатеричных чисел в диапазоне от 00 до FF. Обобщенный синтаксис для этой формы записи – `#RRGGBB`. Обратите внимание, что здесь между числами нет ни пробелов, ни запятых, ни каких-либо других разделителей.

```
.red { color: #ff0000; }
.orange { color: #eea837; }
```

Для шестнадцатеричных чисел, составленных из трех согласованных пар символов, CSS допускает более короткую запись. Обобщенный синтаксис для этой формы записи – `#RGB`.

```
.red { color: #f00; }       /* #f00 = #ff0000 */
.gray { color: #888; }      /* #888 = #888888 */
```

Для измерения длины существует множество способов. Единицы измерения длины могут быть представлены как положительные или отрицательные числа (хотя некоторые свойства будут принимать только положительные значения), за которыми следует обозначение единиц измерения. Числа могут быть и вещественными, т. е. содержать дробную часть, например 10.5 или 4.561. Все

значения длины сопровождаются двухбуквенной аббревиатурой, представляющей единицы измерения, например in (дюймы) или pt (пункты). Единственное исключение из этого правила – нулевая длина (0), для которой не надо указывать единицы измерения.

Различают два типа единиц измерения длины: абсолютные единицы измерения (absolute length units) и относительные (relative length units).

К абсолютным единицам измерения относятся пункты (pt) и пики (pc). Эти термины пришли в веб-разработку из типографии. Традиционно дюйму соответствуют 72 пункта, или 6 пик.

Абсолютные единицы измерения намного удобнее при определении таблиц стилей для печатных документов, где измерения в дюймах, пунктах и пиках – обычное дело, а вот для веб-разработки они не подходят.

Относительные единицы измерения получили такое название потому, что они измеряются относительно других единиц измерения. Измеряемое ими фактическое (или абсолютное) расстояние может меняться под действием внешних факторов, таких как разрешение экрана, ширина области просмотра, предпочтительные настройки пользователя и массы других параметров. Кроме того, для некоторых относительных единиц измерения их размер практически всегда измеряется относительно использующего их элемента и соответственно будет меняться от элемента к элементу:

- **`px`** – пиксел – это точка на экране. Может показаться, что это самый простой и понятный способ задать значение CSS-свойству, потому что нет необходимости производить расчеты относительно другого значения. Но почему тогда пиксел относительная единица измерения? Дело в том, что размер пиксела зависит от разрешения устройства и его технических характеристик;
- **`em`** – в CSS один «em» – это значение свойства `font-size` заданного шрифта. Если для элемента `font-size` равен 14 пикселам, тогда для него же 1em равен 14 пикселам. Это значение может меняться от элемента к элементу. Кроме того, значение `em` меняется в зависимости от размера шрифта родительского элемента. Так, если элементу задать `font-size`, равный 20px, то при задании дочернему элементу значения шрифта 1.5em по факту на экране можно увидеть текст со значением в 30px (20 × 1.5). Но если у дочернего элемента будут свои дочерние элементы, для них значение 1em будет равно уже 30px;

- **rem** – это единица измерения em, отсчет которой ведется относительно значения font-size элемента body.

Ключевые слова

Очень распространенный пример – ключевое слово none, отличающееся от 0 (нуля). Так, чтобы удалить подчеркивание ссылок в HTML-документе, можно написать:

```
a { text-decoration: none; }
```

Аналогично, если бы потребовалось подчеркнуть ссылки, можно было бы указать ключевое слово underline.

Если свойство допускает применение ключевых слов, то его ключевые слова определены только для этого свойства. Если одно и то же слово задано как ключевое для двух свойств, то действие ключевого слова для одного свойства никак не будет связано с его действием в рамках другого свойства. В качестве примера можно взять слово normal. Определенное для свойства letter-spacing, оно означает нечто совершенно отличное от того, что задает normal для свойства font-style.

5.7 Способы добавления CSS на страницу

CSS можно добавить на страницу несколькими способами:

- Первый способ: использование связанных стилей.

```
<link rel="stylesheet" type="text/css"
href="https://goo.gl/vyqNwv">
```

При использовании связанных стилей описание селекторов и их значений располагается в отдельном файле, а для связывания документа с этим файлом применяется тег link. Данный тег помещается в контейнер head.

Значение href задаёт путь к CSS-файлу, он может быть задан как относительно, так и абсолютно. Таким образом можно подключать таблицу стилей, которая находится на другом сайте.

Файл со стилем не хранит никакие данные, кроме синтаксиса CSS. В свою очередь и HTML-документ содержит только ссылку на файл со стилем, т. е. таким способом в полной мере реализуется принцип разделения кода и оформления сайта. Поэтому использование связанных стилей является наиболее универсальным и удобным методом добавления стиля на сайт.

- Второй способ: использование глобальных стилей.

```
<style type="text/css">
```

```
    h1 { font-size: 20px; }
    body { background: yellow; }
  </style>
```

При использовании глобальных стилей свойства CSS описываются в самом документе и располагаются в заголовке веб-страницы. По своей гибкости и возможностям этот способ добавления стиля уступает предыдущему, но также позволяет хранить стили в одном месте, в данном случае прямо на той же странице с помощью контейнера `style`. В нем будут содержаться применяемые к документу стили, но он также может включать ссылки на внешние таблицы стилей с помощью директивы `@import`.

- Третий способ: использование директивы `@import`.

```
@import url(https://goo.gl/Y32anZ);
```

Так же как и `link`, `@import` может указывать браузеру на необходимость загрузки внешней таблицы стилей и использования ее стилей при формировании представления HTML-документа. Единственное основное отличие заключается в синтаксисе и размещении команды. Директива `@import` может находиться только в контейнере `style`. Она должна располагаться перед всеми остальными правилами CSS, иначе не будет работать. В документе может быть несколько директив выражения `@import`, как и тегов `link`.

- Четвертый способ: внутренние стили.

```
<h1 style="text-align: center">CSS</h1>
```

Внутренний, или встроенный, стиль является по существу расширением для одиночного тега, используемого на текущей странице. Для определения стиля используется атрибут `style`, а его значением выступает набор стилевых правил.

Внутренние стили рекомендуется применять на сайте ограниченно или вообще отказаться от их использования. Дело в том, что добавление таких стилей может увеличить общий объём файлов, что ведет к повышению времени их загрузки в браузере и усложнению редактирования документов для разработчиков.

5.8 Типы устройств

Одной из особенностей CSS является возможность подключения тех или иных стилей в зависимости от типа устройства. Благодаря ей можно сделать отдельные таблицы стилей для мониторов, принтеров, мобильных телефонов, звуковых или тактильных браузеров и т. д.

`all` – таблица стилей используется для всех устройств;

`aural` – для речевых синтезаторов;

`braille` – устройства для слепых людей;

`embossed` – страничные принтеры для слепых людей;

`handheld` – для устройств с небольшими экранами (мобильные телефоны, карманные компьютеры и т. д.);

`print` – используется при выводе документа на печать;

`projection` – для проектора;

`screen` – для экрана компьютерного монитора;

`tty` – для устройств, использующих символьную сетку экрана фиксированного шага, например телетайпа;

`tv` – для экранов, подобных телевизионным (низкое разрешение, ограниченная цветопередача, отсутствует прокрутка и т. д.).

Отдельные таблицы стилей могут быть удобны по многим причинам. Например, при выводе на принтер можно убрать меню сайта или рекламные блоки. Также можно изменить шрифт, так как из-за особенностей зрения человека текст, написанный шрифтами без засечек (Arial, Verdana и т. д.) гораздо удобней читать с монитора компьютера, а вот с засечками (Times, Times New Roman и т. д.), наоборот, с бумаги.

Есть несколько способов подключения стилей для конкретных устройств. Рассмотрим один из них. Выбор устройств с помощью тегов `link`. Данный вид подключения основан на добавлении атрибута `media` с указанием типа устройства.

```
<html>
  <head>
    <meta  http-equiv="Content-Type"  content="text/html;
charset=utf-8">
    <link rel="stylesheet" type="text/css"
href="all.css">
    <link  rel="stylesheet"  type="text/css"  href="tv.css"
media="tv">
    <link rel="stylesheet" type="text/css"
href="print.css" media="print">
  </head>
  <body>
    <p>Текст параграфа.</p>
```

```
  </body>
</html>
```

В этом примере стили из файла all.css будут использоваться на всех устройствах. Стили из файла tv.css будут применяться при просмотре страницы на телевизорах, а стили из файла print.css при печати документа.

Контрольные вопросы по главе 5

1. Какие существуют способы подключения CSS к HTML-странице?
2. В чем заключается принцип наследования? Приведите пример.
3. В чем разница между абсолютными и относительными единицами измерения?
4. В чем суть каскада?
5. В каких случаях лучше использовать идентификатор, а в каких класс?

6 Анимации

За всю историю Интернета у разработчиков было несколько вариантов добавления анимации на свои сайты.

Начиная с 1987 г. разработчики могли добавить анимацию с помощью GIF-изображения. Это самая простая и примитивная анимация в рамках одного изображения.

Программы Adobe Flash позволяли создавать сложную анимацию, включая игры и веб-приложения. Недостатком данной технологии является то, что ее необходимо изучать, она не позволяет взаимодействовать с другим HTML-кодом на странице (будь то заголовок или абзац) и часто является источником угрозы безопасности сайта (это одна из причин, почему Apple отказались от Flash на iOS).

JavaScript позволяет анимировать все элементы на странице, но ценой вычислительных мощностей, что порой приводит к неоправданным замедлениям работы сайта.

Консорциум W3C осознал потребность разработчиков в удобном нативном инструменте для создания анимаций на странице и в спецификации CSS3 представил три модуля, реализующие эту возможность: CSS Transforms, CSS Transitions и CSS Animations. Сегодня мы можем перемещать, масштабировать, вращать и наклонять любые HTML-элементы на странице на чистом CSS.

6.1 Преобразования

Перед рассмотрением свойств, отвечающих непосредственно за анимацию, давайте познакомимся с возможностями CSS3, связанными с преобразованиями элементов. На текущий момент нам доступны:

- перемещение;
- масштабирование;
- вращение;
- наклон (перекашивание вдоль горизонтальной или вертикальной оси).

Преобразования позволяют слегка увеличивать пункт меню или элемент галереи при проходе над ним указателя мыши. Можно сочетать несколько преобразований и добиться по-настоящему выразительных эффектов.

Основным CSS-свойством получения перечисленных преобразований является `transform`. В качестве значения указываются тип и степень желаемого

преобразования. Например, для увеличения элемента предоставляется ключевое слово `scale`, далее в скобках указывается коэффициент увеличения: `transform: scale(2);`.

Показанное объявление приведет к увеличению элемента в два раза.

Преобразования обладают одним уникальным свойством – они не влияют на окружающие элементы. Сначала браузеры выделяют элементу то пространство, которое он занимал бы в потоке при обычных обстоятельствах (до преобразования), а затем занимаются преобразованием элемента.

Если поместить увеличивающийся в два раза элемент в один ряд с другими, то после увеличения он наложится на элементы, расположенные выше его, ниже или по бокам. Браузер сохранит остальные части страницы в том виде, в котором они находились, если бы элемент не был увеличен.

Рассмотрим подробнее все имеющиеся у нас в арсенале преобразования.

Перемещение

С помощью функции `translate` свойства `transform` можно перемещать элемент из его текущей позиции по системе координат влево/вправо и вверх/вниз. Как уже было сказано, преобразованный элемент не влияет на окружение и занимает на странице то пространство, которое он занимал бы, будучи показанный без преобразования.

Например, с помощью функции `translate` можно перенести изображение из текста вправо и вниз. Браузер оставляет пустое пространство там, где изображение отрисовалось бы при обычных обстоятельствах, а затем рисует элемент в его новой позиции. В результате такого перемещения на странице остается пустое место – незаполненное пространство между заголовком и параграфом текста.

Можно заметить, что подобного эффекта можно добиться, используя абсолютное или относительное позиционирование. Однако у перемещения с помощью `transform` есть свои преимущества – такие перемещения не вызывают лишних работ по отрисовке в браузере.

В функцию `translate` передаются два аргумента. Первый определяет величину горизонтального, а второй – вертикального перемещения.

Если необходимо переместить элемент влево или вверх, то следует использовать отрицательные значения.

Функция `translate` может перемещать элемент на указанное количество px, em или процентов. В качестве единиц измерения подойдут любые зна-

чения, указывающие длину в CSS. Следует обратить внимание, что значение перемещения, указанное в процентах, вычисляется от текущих размеров элемента. Зачастую эта особенность используется для выравнивания элемента произвольных размеров по центру.

В CSS3 также имеются две дополнительные функции для горизонтального – `translateX` – и вертикального перемещения – `translateY`.

Масштабирование

Функция `scale` позволяет увеличить или уменьшить элемент в размерах. Если необходимо увеличить элемент в полтора или два раза, нам следует добавить следующие объявления:

```
.biceps_x15 {
    transform: scale(1.5);
}
.biceps_x2 {
    transform: scale(2);
}
```

Как уже упоминалось выше, увеличенный в размерах элемент не двигает другие элементы со своего пути.

До этого момента в функцию `scale` передавался один параметр – коэффициент масштабирования. Это число, на которое умножаются текущие размеры элемента. При передаче числа 1 элемент не изменяется (масштабирование отсутствует), при передаче 0.5 элемент уменьшается вдвое, при передаче 4 – увеличивается в четыре раза.

Таким образом числа между 0 и 1 приводят к уменьшению элемента, а числа больше 1 – к увеличению. С помощью `scale(0)` можно фактически скрыть элемент на странице.

При изменении размеров элемента увеличивается или уменьшается как сам элемент, так и все его содержимое, включая текст, изображения, вложенные элементы.

Функция `scale` может принимать два параметра. В таком случае первое число будет относиться к горизонтальному, а второе – к вертикальному масштабированию.

Например, можно уменьшить элемент вдвое по ширине и увеличить вдвое по высоте. В таком случае потребуется следующее объявление:

```
.uncle_stepan {
    transform: scale(.5, 2);
}
```

Что будет, если применить к функции `scale` отрицательные значения? Получится интересный визуальный эффект поворота элемента вокруг его горизонтальной или вертикальной оси. Чтобы повернуть элемент вокруг обеих его осей, необходимо использовать объявление `scale(-1)`. `scale(-1, 1)` поворачивает элемент вокруг его вертикальной оси, а `scale(1, -1)` – вокруг горизонтальной.

Вращение

Функция `rotate` позволяет вращать элемент на указанное количество градусов. Чтобы задать значения угла, используется число, за которым следует единица измерения deg. Чтобы повернуть элемент на 180 градусов, потребуется следующее объявление:

```
.dislike {
    transform: rotate(180deg);
}
```

Отрицательное значение угла вращает элемент против часовой стрелки. Значение `0deg` не придает вращение. Значения, кратные 360, вращают элемент на полные обороты, и внешне это выглядит как отсутствие вращения. Однако использование таких значений оправдано, например, для создания анимации вращения кнопки на один или несколько оборотов во время проведении указателя мыши над кнопкой.

```
.go-away {
    transform: rotate(90deg);
}

.taxi {
    transform: rotate(-30deg);
}
```

Помимо градусов (`deg`, полный круг равен `360deg`), можно использовать другие единицы измерения: `grad` (полный круг `400grad`), `rad` (полный круг `6.2832rad`) и `turn` (равен одному повороту).

В CSS3 есть дополнительные функции для вращения элемента относительно осей X и Y: `rotateX()` и `rotateY()`. С помощью них можно создавать эффекты флип-вращения.

Наклон

Наклон элемента можно выполнить по горизонтальной и вертикальной осям. Чтобы наклонить все вертикальные линии элемента на 45 градусов влево (против часовой стрелки), потребуется следующее объявление:

```
.skew_45_0 {
    transform: skew(45deg, 0);
}
```

Для наклона всех горизонтальных линий на 45 градусов по часовой стрелке потребуется объявление:

```
.skew_0_45 {
    transform: skew(0, 45deg);
}
```

Можно наклонить элемент по двум осям одновременно. Для этого применяется наклон на 25 градусов для вертикальных и горизонтальной линий с помощью объявления:

```
.skew_25 {
    transform: skew(25deg, 25deg);
}
```

Как и в случае с `translate` и `scale`, в CSS3 предлагаются отдельные функции для осей X и Y: `skewX` и `skewY`.

Множественные преобразования

Выше были рассмотрены четыре доступных в CSS3 вида преобразований. До сих пор в качестве значения свойства `transform` указывалась только одна функция. Не стоит ограничиваться одним видом преобразования, можно одновременно вращать, наклонять, передвигать и масштабировать элемент. Для этого надо просто добавить дополнительные функции к свойству `transform` через пробел. Можно увеличить элемент в два раза и повернуть на 180 градусов с помощью объявления:

```
.scale_rotate {
    transform: scale(2) rotate(180deg);
}
```

Следующее объявление применит все четыре вида преобразования к элементу:

```
.all_of_them {
    transform: scale(1.5) skew(45deg, 0) trans-
late(100px, 200px) rotate(180deg);
}
```

Браузер применит все функции к элементу в том порядке, в котором они были перечислены. Порядок играет роль, если используются наклон и вращение. Дело в том, что данные функции наклоняют и вращают не только сам элемент, но и систему координат. Например, можно повернуть элемент на 90 градусов, тем самым поменять оси местами. После такого вращение функции перемещения и наклона будут работать в относительно новой, преобразованной, системе координат.

Для вышерассмотренного примера справедливо, если `translateX(100px)` переместит элемент вниз, а не вправо, и сработает, фактически, как `translateY(100px)` для первоначального элемента, к которому не применялось вращение. Для лучшего понимания работы преобразований после изменения системы координат рассмотрено (или представлено) следующее объявление:

```
.rotate_translate_skew {
    transform: rotate(45deg) skew(20deg, 0) translate(200px, 100px);
}
```

Учитывая все вышеперечисленные особенности, старайтесь внимательно использовать множественные преобразования.

Исходная точка

Все преобразования браузер выполняет относительно точки, которая по умолчанию находится в центре элемента. Однако эту точку можно двигать с помощью свойства `transform-origin`. Это свойство работает так же, как и свойство `background-position`. В качестве значения принимает ключевые слова, абсолютные или относительные единицы измерения (em, %).

Для поворота элемента относительно его левой верхней точки используют объявление:

```
.rotate_lt {
    transform: rotate(45deg);
    transform-origin: left top;
}
```

или

```
.rotate_lt {
    transform: rotate(45deg);
    transform-origin: 0 0;
}
```

или

```
.rotate_lt {
     transform: rotate(45deg);
     transform-origin: 0% 0%;
}
```

Чтобы повернуть элемент относительно центральной нижней точки, используют объявление:

```
.rotate_cb {
     transform: rotate(45deg);
     transform-origin: center bottom;
}
```

или

```
.rotate_cb {
     transform: rotate(45deg);
     transform-origin: 50% 100%;
}
```

При передаче двух аргументов ставится пробел: это связано с тем, что первое значение отвечает за горизонтальную, а второе – за вертикальную позицию. Свойство `transform-origin` не влияет на элементы, которые подвергаются только перемещению с помощью функции `translate`.

3D

Все рассмотренные выше преобразования применяются в двумерном пространстве – системе координат XY. В CSS3 есть возможность имитировать трехмерное пространство на плоском экране монитора, используя трехмерные преобразования (3D transform).

Модель 3D-трансформаций CSS3 является логическим продолжением модели 2D-трансформаций, но с одним важным дополнением – наличием перспективы. Перспектива создает иллюзию глубины и позволяет перемещать в двумерном пространстве экрана точку вдоль и вокруг оси Z (как бы вглубь экрана и из него).

Перспектива позволяет создать иллюзию трехмерного пространства, когда в наличии только двумерное. Сначала берется точка в плоскости, которая называется «точкой схождения линий», рисуется линия от этой точки до краев плоскости, с которой работает исполнитель. В дальнейшем эти линии определят относительный размер всех объектов, которые планируется нарисовать, создавая у смотрящего иллюзию глубины и расстояния до объекта.

3D-преобразования работают похожим образом. В качестве плоскости используется система координат, и тогда точкой схождения линий будет точка 0 на этой системе, а точнее точка (0,0,0), где все значения на осях X, Y и Z равны 0.

Можно управлять расстоянием от точки просмотра до трансформируемого элемента. Для это есть свойство `perspective`. При уменьшении значения перспективы добавляется глубина элемента (увеличиваются его размеры по оси Z). Сам элемент при этом становится визуально меньше. Чем меньше значение, тем ближе Z-пространство к зрителю и тем больше эффект, заданный с помощью свойства `transform`. Свойство `perspective` действует только на дочерние элементы, сам элемент не получает «перспективный» вид. Если единица измерения не указана, по умолчанию она считается в px.

Возможно, все это звучит сложно, но на деле это не так.

```
.3d_perspective_200 {
    perspective: 200px;
}
.3d_perspective_400 {
    perspective: 400px;
}
.3d_perspective_600 {
    perspective: 600px;
}
.3d_perspective_800 {
    perspective: 800px;
}
```

Помимо удаленности можно изменять расположение точки обзора. Для этого в CSS3 имеется свойство `perspective-origin`. Свойство работает только с заданным свойством `perspective`, отличным от нуля. Аналогично `background-position` и `transform-origin` в качестве значения принимаются ключевые слова, абсолютные или относительные единицы измерения (em, %).

Во всех следующих примерах значение перспективы равно 400.

```
.3d_perspective {
    /* по умолчанию: perspective-origin: 50% 50%; */
}
.3d_perspective_lt {
    perspective-origin: 0 0;
```

```
}
.3d_perspective_ct {
    perspective-origin: 50% 0%;
}
.3d_perspective_rt {
    perspective-origin: right top;
}
```

По умолчанию все трехмерные преобразования работают в двумерной системе координат (X и Y) и не имеют глубины (Z). Для имитации глубины необходимо явно изменить это поведение. Сделать это можно с помощью свойства `transform-style`, выставив ему значение `preserve-3d`. По умолчанию установлено второе допустимое значение – `flat`.

Реализуются три элемента: первый выдвигается по оси Z вперед (от экрана), второй остается в плоскости экрана, а третий отодвигается назад, вглубь экрана. Родительский элемент также немного поворачивается. Если оставить значение по умолчанию, то глубина картины не предстанет. В отличие от этого, значение `preserve-3d` создаст псевдоглубину.

Также стоит упомянуть о свойстве `backface-visibility`, которое контролирует видимость задней части элемента, когда он повернут. Значение `hidden` отменяет видимость по умолчанию и скрывает обратную сторону преобразованного элемента.

Все функции преобразования, кроме наклона, имеют аналоги в 3D-трансформациях: `translate3d(x,y,z)`, `translateZ(z)`, `scale3d(x,y,z)`, `scaleZ(z)`, `rotate3d(x,y,z,угол)`, `rotateZ(угол)`.

Матрица преобразований

Помимо четырех функций преобразования: `translate`, `scale`, `rotate` и `skew` CSS3 позволяет самостоятельно описать трансформацию с помощью матрицы преобразования. В данном случае имеется ограничение – все линии остаются параллельными, таким образом нельзя описать перспективу с помощью матрицы.

Матрица имеет размер 3×3. Иногда третью колонку опускают, так как она не влияет на конечный результат.

Чтобы рассчитать новые координаты элемента после применения матрицы трансформации, браузер использует следующие формулы:

$$x^{new} = ax + cy + t_x,$$

$$y^{new} = bx + dy + t_y,$$

где a – изменение размера по горизонтали `scaleX()`;

b – наклон по горизонтали `skewX()`;

c – наклон по вертикали `skewY()`;

d – изменение размера по вертикали `scaleY()`;

t_x – смещение по горизонтали в пикселах `translateX()`;

t_y – смещение по вертикали в пикселах `translateY()`.

Матрица задается с помощью ключевого слова `matrix` свойства `transform`. В функцию `matrix` передаются аргументы – коэффициенты матрицы преобразования.

```
.matrix {
     transform: matrix(a, c, b, d, tx, ty);
}
```

Порядок перечисления коэффициентов имеет принципиальное значение.

Для описания 3D-преобразований используется матрица 4×4 и ключевое слово `matrix3d` свойства `transform`.

Переходы

Преобразования позволяют создавать забавные эффекты, но по-настоящему оживить страницу они могут в сочетании с CSS3-переходами.

Переход – это анимация от одного набора CSS свойств к другому. Для работы перехода необходимо:

- два набора свойств. Первый набор описывает начальное состояние элемента (например, `color: red`), а второй – конечное (`color: blue`). Процессом анимации от одного набора к другому (переход цвета от красного до синего) занимается браузер;
- свойство `transition`. Краеугольным камнем перехода является описание свойств и характеристик анимации перехода. Как правило, описание применяется к исходному набору свойств;
- инициатор. Действие, которое вызывает изменение от одного набора свойств к другому. Часто в качестве инициатора используются псевдоклассы `:hover`, `:target`, `:focus`, `:active`. Например, с помощью `:hover` можно анимировать изменение внешнего вида элемента от первоначального появления на странице до проведения над элементом указателя мыши. Конечно, в качестве инициатора может выступать JavaScript: применение инлайновых стилей к элементу или до-

бавление класса с новым набором CSS-свойств. После пропадания инициатора (пользователь увел указатель мыши с элемента) браузер возвращает внешний вид элемента к первоначальному и анимирует этот процесс.

Необходимо единожды установить переход в CSS, а все хлопоты по созданию анимации от одного набора свойств к другому и обратно браузер возьмет на себя.

Поддерживаемые свойства

Браузер не может анимировать переходы для всех CSS-свойств, но и список доступных для анимации свойств достаточно велик:

- все преобразования: `translate`, `rotate`, `scale`, `skew`;
- `color`;
- `background-color`;
- `border-color`;
- `width`;
- `height`;
- `border-width`;
- `margin`;
- `padding`;
- `font-size`;
- `line-height`;
- `letter-spacing`;
- `word-spacing`;
- `top`;
- `right`;
- `bottom`;
- `left`;
- `opacity`.

Создание перехода

В основе CSS-перехода лежат четыре свойства: `transition-property` (свойства, требующие анимации), `transition-duration` (продолжительность

анимации), `transition-timing-function` (тип анимации), `transition-delay` (задержка выполнения анимации).

Для перехода сначала нужно задать два набора свойств: для первоначального вида элемента и конечного.

```
.button {
    /* набор свойств для старта анимации*/
}
.button:hover {
    /* стили, которые применятся к элементу, пока над ним выполняется действие*/
    transform: scale(1.2);
}
```

Эти стили работают мгновенно, кнопка увеличивается в 1.2 раза моментально при проходе курсора мышки над кнопкой.

Чтобы масштаб изменился плавно и анимация изменения длилась половину секунды, нужно к стилю кнопки добавить два новых свойства:

```
.button {
    transition-property: transform;
    transition-duration: .5s;
}
.button:hover {
    transform: scale(1.2);
}
```

В `transition-property` указано свойство, которое нужно анимировать, а в `transition-duration` – время, в течение которого будет осуществляться анимация.

В качестве значения для `transition-property` можно указывать не только конкретное свойство, но и:

- ключевое слово `all` (в таком случае будут анимироваться все изменяемые CSS-свойства);
- список анимируемых свойств через запятую (`transition: width, height, ...`).

Свойство `transition-duration` принимает значение в секундах или миллисекундах (`transition-duration: .5s` эквивалентно `transition-duration: 500ms`). Если в `transition-property` указано несколько CSS свойств, в `transition-duration` есть возможность задать продолжительность анимации для каждого свойства отдельно. Для этого надо перечислить

значения через запятую. Порядок перечисления продолжительности анимации совпадает с порядком следования анимируемых свойств.

При необходимости можно добавить изменение цвета фона для используемой кнопки с той же продолжительностью, что и масштаб.

```
.button {
    background-color: green;
    transition-property: transform, background-color;
    transition-duration: .5s, .5s;
}
.button:hover {
    transform: scale(1.2);
    background-color: blue;
}
```

Можно задержать время начала анимации перехода, воспользовавшись свойством `transition-delay`. Например, если надо подождать полсекунды, прежде чем начать анимацию фона, можно добавить следующий код:

```
.button {
    background-color: green;
    transition-property: transform, background-color;
    transition-duration: .5s, .5s;
    transition-delay: 0s, 1s;
}
.button:hover {
    transform: scale(1.2);
    background-color: blue;
}
```

Теперь при проходе указателя мыши над кнопкой сначала плавно увеличивается масштаб кнопки, а после окончания этой анимации плавно изменяется цвет фона. Аналогично `transition-duration`, для каждого свойства можно указать свое время задержки. Порядок перечисления показателей времени задержки должен соответствовать порядку перечисления этих свойств в `transition-property`.

И последнее свойство, описывающее CSS-переход, – это `transition-timing-function`. Это свойство описывает скорость хода анимации. В его предназначении легко запутаться: в отличие от `transition-duration`, которое управляет продолжительностью анимации, `transition-timing-function` описывает, как быстро по времени меняется указанное через `tran-`

sition-property значение свойства. Например, можно начать изменение свойства быстро в начале и медленно в конце, или наоборот.

В качестве значений для данного свойства есть 8 ключевых слов: linear, ease, ease-in, ease-out, ease-in-out, step-start, step-end и step. По умолчанию используется метод ease, при котором анимация начинается медленно, ускоряется к середине и снова замедляется в конце. Используя разные методы, можно варьировать общий вид анимации.

Помимо ключевых слов свойство transition-timing может воспринимать кубическую кривую Безье. Эта кривая описывает график хода анимации по времени. Кривизной линии можно управлять путем настройки двух контрольных точек: чем круче линия, тем быстрее анимация, а более пологая линия означает более медленный ход анимации.

На самом деле все ключевые слова (linear, ease, ease-in, ease-out и ease-in-out) – это частные и наиболее популярные кривые. Например, метод ease-in можно задать и с помощью кубической кривой Безье:

```
.button {
    ...
    transition-timing-function: cubic-bezier(0.420,
0.000, 1.000, 1.000);
}
```

С помощью кривых Безье можно добиться крайне интересных эффектов в Ваших анимациях.

Для указания всех четырех свойств можно использовать лишь одно – transition. Это свойство объединяет все вышеперечисленные в одно. Для этого нужно перечислить через пробел:

- transition-property;
- transition-duration;
- transition-delay;
- transition-timing-function.

Обязательными считаются анимируемое свойство (или ключевое слово all) и продолжительность анимации, а задержку и функцию распределения скорости по времени можно опустить. По умолчанию задержка равна нулю (отсутствует), а функция распределения – ease. Следующее объявление включит анимацию изменения всех свойств в течение одной секунды.

```
.transition-all {
    transition: all 1s;
}
```

Если требуется анимация нескольких CSS-свойств, можно перечислить их (вместе с характеристиками перехода) через запятую:

```
.button {
    transition: transform .5s ease-in, background-color
.5s ease-in 1s;
}
```

Переходы позволяют создавать красивые анимации от одного набора свойств к другому. Помимо переходов в CSS3 есть механизм анимаций, позволяющий анимировать переходы от одного набора свойств к другому, затем к третьему и так далее. Кроме того, можно задать повторяющуюся анимацию, повернуть ход анимации вспять, приостановить и возобновить анимацию.

6.2 Анимация

В отличие от переходов, анимация не требует наличия инициатора. Можно запустить анимацию сразу при загрузке страницы и тем самым, например, привлечь внимание к какому-либо элементу.

Для создания анимации необходимо:

- определить ключевые кадры. Ключевой кадр – это отдельный кадр анимации, определяющий внешний вид сцены. Например, в качестве первого кадра можно указать зеленый цвет фона для элемента, в качестве второго – синий. После этого браузер создаст анимацию перехода цвета фона от зеленого к синему (от первого кадра ко второму). В переходы заложен тот же механизм. Однако если в переходах можно задать «два кадра» анимации (внешний вид до и после применения инициатора), то в механизме анимации можно определять множество ключевых кадров. Например, переход от зеленого цвета фона к синему, потом к желтому и в конце – к красному;
- применение анимации к элементу. После определения ключевых кадров можно назначать созданную анимацию любому количеству элементов на странице. При этом продолжительность, задержка и другие свойства анимации могут варьироваться от элемента к элементу.

Ключевые кадры создаются с помощью правила `@keyframes`, за которым следует наименование анимации. Это имя затем используется для применения анимации к элементам. Хорошим тоном считается называть анимацию осмысленно (например, `fadeIn/fadeOut`).

Ключевое слово `@keyframes` подобно конструкциям `@import` и `@media` не является CSS-свойством, а считается так называемым эт-правилом (по названию символа @ – «эт»).

Пример задания анимации:

```
@keyframes имяАнимации {
     from {
          /* CSS свойства для первого кадра */
     }
     to {
          /* CSS свойства для второго кадра */
     }
}
```

Анимация содержит минимум два ключевых кадра. В примере выше использованы ключевые слова `from` и `to` для создания начального и конечного кадров соответственно. На самом деле описание начального кадра можно опустить, тогда в качестве начальных CSS-свойств для анимации будут использованы текущие свойства элемента.

Каждый ключевой кадр, фактически, это простой набор CSS-свойств, описывающий внешний вид элемента.

Рассмотрим процесс создания анимации для попапа (popup) – первоначально скрытого окошка, которое отображается по центру экрана после определенных действий (в нашем случае после клика на кнопку).

Создадим анимацию перехода от абсолютной прозрачности к абсолютной непрозрачности.

```
@keyframes show {
     from {
          opacity: 0;
     }
     to {
          opacity: 1;
     }
}
```

После создания анимация готова к применению. Ее можно добавить к любому стилю любого элемента на странице. Если просто добавить анимацию к элементу, то она запустится при загрузке страницы. Можно привязать анимацию в какому-либо действию, добавив ее к одному из псевдоклассов: `:hover`, `:active`, `:target` или `:focus`.

В CSS3 имеется несколько свойств, позволяющих управлять способом и временем проигрывания анимации. Для работы анимации как минимум необходимо указать имя (которое задается в правиле `@keyframes`) и продолжительность анимации.

В нашем случае после клика на кнопку элементу с попап-окном добавляется класс `b-popup_show`. Для проигрывания анимации классу `b-popup_show` необходимо указать следующие CSS-свойства:

```
.b-popup_show {
    animation-name: show;
    animation-duration: 1s;
}
```

Свойство `animation-name` сообщает браузеру, какую анимацию нужно применить. Здесь указывается то самое имя, которое использовалось при создании анимации.

Свойство `animation-duration` устанавливает время анимации (в секундах или миллисекундах).

Как видно, анимация задается в одном месте (в правиле `@keyframes`), а применяется в другом (в стиле элемента). Благодаря этому можно применять одну и ту же анимацию для разных элементов, меняя параметры (продолжительность и др.). Таким образом, созданная анимация `show` может применяться для отображения других элементов на странице.

Нет ограничений по указанию одной анимации для нашего попапа. Предполагается, что есть вторая анимация моргания фонового цвета элемента с синего на зеленый:

```
@keyframes blink {
    from {
        background-color: blue;
    }
    to {
        background-color: green;
    }
}
```

Чтобы применить к элементу более одной анимации, нужно указать список имен с запятой в качестве разделителя и список времен длительности через запятую.

Как и в случае с переходами, порядок применения параметров времени совпадает с порядком перечисления имен анимаций. В данном случае к попап-

окну применится анимация появления длительностью 1 секунда и анимация моргания цвета фона, длительностью 3 секунды.

Исполнение не ограничено использованием двух ключевых кадров. С помощью процентных значений можно указать несколько кадров. Значение в процентах указывает место описываемой кадром сцены на общей продолжительности анимации. Вводится третий кадр к имеющейся анимации бликования цвета фона. Пусть фон изменяет цвет от синего к красному, а затем от красного к зеленому.

```
@keyframes blink {
     from {
          background-color: blue;
     }
     50% {
          background-color: red;
     }
     to {
          background-color: green;
     }
}
```

Теперь в середине анимации цвет фона станет красным. Ключевые слова `from` и `to` являются синонимами для процентных значений 0% и 100%. Можно добавить сколь угодно много ключевых кадров:

```
@keyframes blink {
     0% {
          background-color: blue;
     }
     25% {
          background-color: green;
     }
     50% {
          background-color: red;
     }
     75% {
          background-color: yellow;
     }
     100% {
          background-color: grey;
     }
}
```

Если анимация, указанная выше, длится 10 секунд, тогда на нулевой секунде выставится синий цвет фона. Затем в течение 2.5 секунд цвет фона будет плавно меняться с синего на зеленый и в отметке 2.5 секунды станет зеленым. Затем до 5 секунд будет происходить плавная смена цвета до красного, потом до 7.5 секунд до желтого и последние 2.5 секунды цвет фона будет меняться с желтого на серый. В конце анимации, на 10-й секунде, цвет фона станет серым.

Можно усложнить использование процентных значений, добавив несколько таких значений к одному набору CSS-свойств.

```
@keyframes blink {
    0%, 50% {
        background-color: blue;
    }
    25%, 75% {
        background-color: green;
    }
    100% {
        background-color: grey;
    }
}
```

Таким образом, фоновый цвет попапа будет синим на 0-й и 5-й секунде, зеленым на отметках 2.5 и 7.5 секунд и станет серым в конце анимации на 10-й секунде. С помощью такого трюка можно притормозить смену какого-то свойства, сделав паузу в анимации:

```
@keyframes blink {
    0% {
        background-color: blue;
    }
    25%, 75% {
        background-color: green;
    }
    100% {
        background-color: grey;
    }
}
```

В таком случае с 25 до 75% от всего времени анимации цвет фона будет оставаться зеленым. На 25% хода анимации цвет фона станет зеленым, затем от отметки 25% до 75% будет оставаться зеленым, и с 75% начнем меняться на серый, пока не станет чисто серым к 100%.

Пока что все рассмотренные примеры включали изменение только одного CSS-свойства. Естественно, никто не ограничивает использование в анимации сразу нескольких свойств.

```
@keyframes show {
    0% {
        opacity: 0;
        background-color: blue;
    }
    50% {
        background-color: green;
    }
    100% {
        opacity: 1;
        background-color: red;
    }
}
```

Аналогично `transition-delay` свойство `animation-delay` задает возможное время ожидания перед воспроизведением анимации. По умолчанию анимация выполняются сразу, как только она применяется к элементу. Допустимо указывать задержку в секундах и миллисекундах.

Как и в случае с переходами, можно указывать функцию распределения скорости анимации по времени. Свойство `animation-timing-function` принимает пять ключевых слов: `linear`, `ease`, `ease-in`, `ease-out`, `ease-in-out` или функцию Безье. Все это работает точно так же, как в рассмотренном случае с переходами.

Можно управлять функцией распределения скорости по времени не только между началом и завершением анимации, но и между отдельными ключевыми кадрами.

Предположим, что исполнитель реализует анимацию цвета фона с тремя ключевыми кадрами. Ему необходимо замедлить превращение первого цвета во второй с помощью функции `ease-in`, а затем равномерно продолжить анимацию от середины до ее окончания. Это можно сделать добавлением двух функций распределения скорости по времени для первого ключевого кадра (она будет управлять анимацией от 0 до 50%) и второго (для управления анимацией от 50 до 100%).

```
@keyframes show {
    0% {
```

```
            background-color: blue;
            animation-timing-function: ease-in;
        }
        50% {
            background-color: green;
            animation-timing-function: linear;
        }
        100% {
            background-color: red;
        }
    }
```

В переходах работа происходила с однократным изменением свойств элемента (например, на `:hover`). В анимациях благодаря свойству `animation-iteration-count` можно проигрывать анимацию один, два и более раз. Допустим, он имеет анимацию масштаба элемента от единицы до двух и хочет повторить анимацию 10 раз.

```
.b-circle {
    animation-name: scale;
    animation-duration: 2s;
    animation-iteration-count: 10;
}
@keyframes scale {
    0% {
        transform: scale(1);
    }
    100% {
        transform: scale(2);
    }
}
```

В качестве значения свойства `animation-iteration-count` может выступать ключевое слово `infinite`, которое запускает анимацию бесконечное количество раз.

При многократном запуске анимации браузер запускает каждое повторение с самого начала. В связи с этим может происходить неприятный визуальный эффект. В примере выше браузер анимирует масштаб элемента от единицы до двух, после чего резко возвращает масштаб к единице и снова анимирует его до двух.

В отличие от переходов, где «из коробки» после пропадания инициатора браузер воспроизводил анимацию в обратном направлении, в CSS3-анимациях необходимо указать свойство `animation-direction`, равное `alternate`. В таком случае браузер будет воспроизводить нечетные итерации в прямом направлении, а четные – в обратном.

```
.b-circle {
    animation-name: scale;
    animation-duration: 2s;
    animation-iteration-count: infinite;
    animation-direction: alternate;
}
```

Эффект резкого перехода между последним и первым ключевым кадром пропадает, если необходимо выполнить анимацию несколько раз и вернуться к его первоначальному виду, используется четное количество итераций и устанавливается `animation-direction: alternate`.

Следующий, режущий глаз, эффект возникает в момент завершения всех повторений анимации (если только свойство `animation-iteration-count` не равно `infinite`). По завершении браузер отобразит элемент в его первоначальном виде. В примере с анимацией масштаба от единицы до двух устанавливается количество повторений, равное трем.

```
.b-circle {
    animation-name: scale;
    animation-duration: 2s;
    animation-iteration-count: 3;
    animation-direction: alternate;
}
```

Свойство `animation-direction` работает без видимых резких изменений в масштабе между итерациями. Однако, учитывая нечетное количество повторений, в конечном итоге анимация завершатся исполнителем со значением масштаба, равным двум. После этого браузер резко вернет элемент к первоначальному виду и масштабу, создав еще один резкий визуальный эффект.

Исполнитель может заставить браузер сохранять внешний вид элемента, который он приобретает по завершении анимации. Для этого в CSS3 есть свойство `animation-fill-mode`, которому следует выставить значение `forwards`.

```
.b-circle {
    animation-name: scale;
```

```
    animation-duration: 2s;
    animation-iteration-count: 3;
    animation-direction: alternate;
    animation-fill-mode: forwards;
}
```

Теперь по окончании трех повторений элемент сохранил масштаб, равный двум.

CSS3-анимации предлагают множество свойств по управлению анимациями. К счастью, имеется свойство `animation`, в котором сочетаются такие свойства, как:

- `animation-name`;
- `animation-duration`;
- `animation-timing-function`;
- `animation-iteration-count`;
- `animation-direction`;
- `animation-delay`;
- `animation-fill-mode`.

Таким образом, следующие объявления полностью эквиваленты.

```
.long {
    animation-name: scale;
    animation-duration: 2s;
    animation-timing-function: ease-in-out;
    animation-iteration-count: 3;
    animation-direction: alternate;
    animation-delay: 5s;
    animation-fill-mode: forwards;
}
.short {
    animation: scale 2s ease-in-out 3 alternate 5s for-
wards;
}
```

Для указания нескольких анимаций используется разделитель – запятая.

```
.b-multi {
    animation: scale 2s ease-in-out 3 alternate 5s for-
wards, fadeIn 2s;
}
```

Как уже упоминалось выше, CSS3-анимации имеют механизм остановки и возобновления анимации. Для этих целей предлагается свойство `animation-play-state`, которое принимает одно из двух значений: `running` или `paused`.

Например, имеется анимация пульсирующего сердца, и исполнитель хочет остановить пульсацию при проходе указателя мыши над элементом.

```
.b-heart {
    ...
    animation: heartBeat 1s ease infinite 0s;
}
.b-heart:hover {
    animation-play-state: paused;
}
```

При наведении указателя мыши на сердце анимация замирает. Если указатель убрать – свойство `animation-play-state` примет значение по умолчанию `running` и анимация продолжится. При создании сложных анимаций можно добавить кнопку «Пауза» и останавливать анимацию по требованию пользователя.

CSS-анимации активно используются разработчиками для создания визуальных эффектов. С помощью анимаций можно в несколько строчек CSS-кода заметно улучшить визуальный вид активного пункта меню, отмеченной фотографии в галерее, сделать красивый спиннер или прелоадер. Имея фантазию и умение работать с анимациями, можно создавать по-настоящему эффектные сайты.

В настоящий момент все современные браузеры поддерживают CSS3-трансформации, переходы и анимации. Только IE требует вендорного префикса для некоторых свойств 3D-трансформаций.

Анимации на CSS просты, декларативны, задействуют аппаратное ускорение на видеокарте, за счет чего выигрывают в производительности.

Контрольные вопросы по главе 6

1. Каково главное преимущество анимации на CSS перед анимацией на JavaScript?
2. Как выглядит форма записи для свойства `transition`?
3. Можно ли зациклить анимацию, созданную с помощью `transition`?
4. Как создать покадровую анимацию?
5. Что делает свойство `transform`?

7 Введение в JavaScript

JavaScript *– это интерпретируемый язык программирования с объектно-ориентированными возможностями, появившийся в 1995 г. Причиной возникновения языка называют потребность в создании динамических веб-страниц при минимальных знаниях в сфере программирования. Использовавшийся для этих же целей язык Java требовал серьезной профессиональной подготовки, и на фоне этого новый язык начал завоевывать популярность.*

Несмотря на синтаксическую схожесть с C, C++ и Java, JavaScript не считается их близким родственником. Семантически язык гораздо ближе к Self, Smalltalk и Лиспу, а некоторые инструменты в JavaScript (JS) созданы по образу и подобию языка Perl.

На сегодняшний день JavaScript уже вышел за рамки языка клиентской стороны. Появились возможности использовать JS и в серверной части, с помощью таких разработок, как Node.js, v8cgi, Jaxer. Также JavaScript используется в серверной части проектов компании Google.

7.1 Основы JavaScript

Группа слов, чисел и операторов, выполняющих специфичную задачу, называется утверждением (statement). В JavaScript утверждение может выглядеть, например, так:

```
a = b * 2;
```

Символы `a` и `b` здесь переменные, а символы `=` и `*` являются операторами – они совершают действия над значениями и переменными. Большинство утверждений в JavaScript содержат `;` в конце (хотя наличие `;` в конце утверждения является опциональным, его использование все же помогает избежать ряда проблем). Таким образом, программа является просто набором утверждений, которые описывают шаг за шагом, что нужно сделать для достижения цели.

Утверждения состоят из одного и более выражений.

Выражение *(expression) – это ссылка на переменную или значение, а также набор переменных и значений, совмещенный с операторами.*

```
a = b * 2;
```

Это утверждение состоит из четырёх выражений:

- `2` – это выражение значения-литерала;
- `b` – это выражение переменной, которое говорит «получи текущее значение переменной»;
- `b * 2` – это арифметическое выражение выполнения умножения;
- `a = b * 2` – это выражение присвоения.

Отдельно стоящее выражение называют выражением-утверждением (expression statement). Например, оно может выглядеть так:

```
b * 2;
```

Само по себе утверждение «получить значение переменной `b` и умножить его на `2`» бесполезно, т. к. мы не сохраняем результат этого действия. Поэтому чаще всего мы можем увидеть выражения-утверждения в виде вызовов функции:

```
alert( a ).
```

Основные способы ввода/вывода данных

Вывод осуществляется с помощью функций: `console.log()`, `alert()` или через элементы форм. Ввод – через `promt()` или также через элементы формы.

Распространенные операторы:

- математические: `+`, `-`, `*`, `/`;
- присвоение и комбинированное присвоение: `=`, `+=`, `-+`, `*=`, `/=`;
- инкремент, декремент: `++`, `--`;
- сравнение: `==`, `===`, `!=`, `!==`, `<`, `>`, `<=`, `=>`;
- логические и оператор ветвления: `&&`, `||`, `if () {} else {}`;
- цикла: `while () {}`, `do {} while ()`, `for () {}`;
- комментариев: `//`строчный, `/*`блочный`*/`.

Переменная состоит из имени и выделенной области памяти, которая ему соответствует. Для объявления переменной используется ключевое слово `var`:

```
var foo; //объявляем переменную
foo = 42; //присваиваем значение переменной
```

Эти данные будут сохранены в соответствующей области памяти и в дальнейшем доступны при обращении по имени.

Имя переменной должно начинаться с буквы, символа _ или символа $. Остальная часть может состоять из букв, цифр, символов _ или $. Запрет на использования цифр в качестве первого символа идентификатора необходим для избежания путаницы с числовыми литералами. В EcmaScript 3 идентификаторы могут содержать буквы и цифры из полного набора символов Unicode, более старые же версии ограничены набором символов ASCII. Еще одно ограничение на идентификатор заключается в том, что они не могут совпадать с зарезервированными словами языка (`break`, `default`, `function`, `return`, `var`, `case`, `delete`, `if`, `switch`, `void`, `catch`, `do`, `in`, `this`, `while`, `const`, `else`, `instanceof`, `throw`, `with`, `continue`, `finally`, `let`, `try`, `debugger`, `for`, `new`, `typeof`).

Во многих языках программирования помимо переменных существуют константы. В JavaScript, увы, изначально не было способа создания констант, поэтому для того чтоб отличить их от переменных, константы обычно именуют заглавными буквами:

```
var IamVariable = 42;
var I_AM_CONSTANT = 42;
```

И, наконец, самый главный момент: как запустить код? На сегодняшний день одной из самых популярных сред, где исполняется JavaScript код, помимо браузера, является NodeJS. Подробную инструкцию по установке, равно как и документацию, можно найти на официальном сайте.

Типы данных и работа с ними

Встроенные типы данных:

- `null` – специальное значение, которое имеет смысл «ничего» или «значение неизвестно». В этом JavaScript отличается от других языков, где `null` является «нулевым указателем» или «ссылкой на несуществующий элемент»;
- `undefined` – значение не присвоено. Это значение появляется у переменной, которая объявлена, но в нее ничего не записано.

  ```
  var x;
  console.log( x ); // выведет "undefined"
  ```

В явном виде `undefined` обычно не присваивают, так как это противоречит его смыслу. Для записи в переменную «пустого» или «неизвестного» значения используется `null`.

Для того чтобы узнать тип аргумента, используется оператор `typeof`. У него есть два синтаксиса, которые работают одинаково: со скобками `(typeof x)` и без `(typeof(x))`:

```
typeof undefined // "undefined"
typeof 0 // "number"
typeof true // "boolean"
typeof "foo" // "string"
typeof null // "object"
```

Результат `typeof null == "object"` – это официально признанная ошибка в языке, которая сохраняется для совместимости. На самом деле `null` – это не объект, а отдельный тип данных.

Для работы с переменными, со значениями JavaScript поддерживает все стандартные операторы, большинство которых есть и в других языках программирования. Несколько операторов – это обычные сложение `+`, умножение `*`, вычитание и так далее.

Операнд – то, к чему применяется оператор. Например: `5 * 2` – оператор умножения с левым и правым операндами. Другое название: «аргумент оператора». Унарным называется оператор, который применяется к одному выражению. Например, оператор унарный минус «-» меняет знак числа на противоположный:

```
var x = 1;
x = -x;
```

Бинарным называется оператор, который применяется к двум операндам. Тот же минус существует и в бинарной форме:

```
var x = 1, y = 3;
console.log( y - x ); // 2, бинарный минус
```

Важно помнить, что в операции сложения, если хотя бы один из операндов является строкой, то второй будет также преобразован к строке! Причем неважно где находится операнд-строка. Например:

```
console.log( '1' + 2 ); // "12"
console.log( 2 + '1' ); // "21"
```

В других арифметических операциях (вычитание, умножение, деление) операнды-строки, напротив, будут преобразованы к числам:

```
console.log( 2 - '1' ); // 1
console.log( 6 / '2' ); // 3
console.log( '4' - '1' ); // 3
```

Унарный, то есть применённый к одному значению, плюс ничего не делает с числами. С точки зрения математики такое изобилие плюсов может показаться странным. С точки зрения программирования – никаких разночтений: сначала выполнятся унарные плюсы, приведут строки к числам, а затем бинарный + их сложит.

```
var apples = "2";
var oranges = "3";
console.log( +apples + +oranges ); // 5, число, оба операнда предварительно преобразованы в числа
```

Остальные операторы:

- оператор взятия остатка «%» интересен тем, что, несмотря на обозначение, никакого отношения к процентам не имеет. Его результат `a % b` – это остаток от деления `a` на `b`:
  ```
  console.log( 5 % 2 ); // 1, остаток от деления 5 на 2
  ```
- инкремент «++» увеличивает на 1:
  ```
  var i = 2;
  i++;    // более короткая запись для i = i + 1
  console.log(i); // 3
  ```
- декремент «--» уменьшает на 1:
  ```
  var i = 2;
  i--;    // более короткая запись для i = i - 1
  console.log(i); // 1
  ```

Вызывать эти операторы можно не только после, но и перед переменной: `i++` (постфиксная форма) или `++i` (префиксная форма). Обе эти формы записи делают одно и то же: увеличивают на 1.

Тем не менее, между ними существует разница. Она видна только в том случае, когда мы хотим не только увеличить/уменьшить переменную, но и использовать результат в том же выражении.

```
var i = 1;
var a = ++i; // (*)
console.log(a); // 2
```

Постфиксная форма `i++` отличается от префиксной `++i` тем, что возвращает старое значение, бывшее до увеличения.

```
var i = 1;
```

```
var a = i++; // (*)
console.log(a); // 1
```

Инкремент/декремент можно использовать в любых выражениях.

- Сокращённая арифметика с присваиванием:

```
var n = 2;
n = n + 5;// -> n += 5;
n = n * 2;// -> n *= 2
```

Так можно сделать для операторов `+`, `-`, `*`, `/` и бинарных `<<`, `>>`, `>>>`, `&`, `|`, `^`. Вызов с присваиванием имеет в точности такой же приоритет, как обычное присваивание, то есть выполнится после большинства других операций.

Один из самых необычных операторов – запятая `','`. Его можно вызвать явным образом, например:

```
var a = (5, 6);
```

Запятая позволяет перечислять выражения, разделяя их запятой `','`. Каждое из них – вычисляется и отбрасывается, за исключением последнего, которое возвращается. Запятая – единственный оператор, приоритет которого ниже присваивания. В выражении `a = (5,6)` для явного задания приоритета использованы скобки, иначе оператор `'='` выполнился бы до запятой `','`, получилось бы `(a=5), 6`. Зачем же нужен такой странный оператор, который отбрасывает значения всех перечисленных выражений, кроме последнего? Обычно он используется в составе более сложных конструкций, чтобы сделать несколько действий в одной строке. Например:

```
// три операции в одной строке
for (a = 1, b = 3, c = a*b; a < 10; a++) {
 ...
}
```

- Операторы сравнения: больше/меньше: `a > b`, `a < b`; больше/меньше или равно: `a >= b`, `a <= b`; равно `a == b` (в отличие от оператора присваивания, для сравнения используется два символа равенства); «не равно» – «`!=`» (в математике он пишется как ≠). В отличие от оператора равенства, данный оператор не приводит операнды к одному типу.

```
var a = 2;
var b = 3;
a == b; //false
a === b; //false
a > b; //false
```

```
a >= b; //false
a < b; //true
a <= b; //true
a != b; //true
a !== b; //true
```

Аналогом «алфавита» во внутреннем представлении строк служит кодировка, у каждого символа – свой номер (код). JavaScript использует кодировку Unicode. При этом сравниваются численные коды символов. В частности, код у символа Б больше, чем у А, поэтому и результат сравнения такой. В кодировке Unicode обычно код у строчной буквы больше, чем у прописной, поэтому регистр имеет значение:

```
'Вася' > 'Ваня'; //true т. к. 'с' > 'н'
```

В JS есть только один тип чисел – `number`. JS также руководствуется стандартом IEEE 754 (Стандарт двоичной арифметики с плавающей точкой), который использует двойную точность.

```
var n = 123;
n = 12.345;
```

Если целая часть равна 0, то необязательно его писать:

```
var a = 0.42;
var b = .42;
```

Также и для десятичной части:

```
var a = 42.0;
var b = 42.;
```

Очень большие и очень маленькие числа будут выводиться через экспоненту:

```
var a = 5E10;
a; // 50000000000
a.toExponential(); // "5e+10"
```

По умолчанию JavaScript переводит в экспоненциальную запись любые значения с плавающей точкой, содержащие как минимум шесть нулей после точки:

```
0.0000005; //5e-7
0.000005; //0.000005
```

Числа можно также записывать в других системах исчисления:

```
0xf3; // Шестнадцатеричная: 243
0Xf3; // то же самое
010; // восьмеричная: 8
08; // десятеричная: 8
```

Восьмеричная система определяется тем, что первым числом является 0, а затем идут числа от 1 до 7. Иначе будет считаться, что система десятеричная.

В процессе округления вещественных чисел могут возникнуть подобные ситуации:

```
0.1 + 0.2 === 0.3; // false
0.1 + 0.2; // 0.30000000000000004
```

Всё дело в том, что в стандарте IEEE 754 на число выделяется ровно 8 байт (= 64 бита), не больше и не меньше.

Число 0.1 (одна десятая) записывается просто в десятичном формате, а в двоичной системе счисления это бесконечная дробь. Такой же бесконечной дробью является 0.2 (=2/10). Двоичное значение бесконечных дробей хранится только до определенного знака, поэтому возникает неточность. Её даже можно увидеть:

```
console.log( 0.1.toFixed(20) ); // 0.10000000000000000555
```

Есть несколько способов этого избежать:

- привести к целым числам, провести операции и вернуться обратно:
  ```
  console.log( (0.1 * 10 + 0.2 * 10) / 10 ); // 0.3
  ```
- использовать эпсилон-окрестности (в ES6 есть константа):
  ```
  Number.EPSILON; //2.220446049250313e-16
  if (!Number.EPSILON) {
    Number.EPSILON = Math.pow(2,-52);
  }
  function numbersCloseEnoughToEqual(n1,n2) {
    return Math.abs( n1 - n2 ) < Number.EPSILON;
  }
  var a = 0.1 + 0.2;
  var b = 0.3;
  numbersCloseEnoughToEqual( a, b ); // true
  numbersCloseEnoughToEqual( 0.0000001, 0.0000002 );
// false
  ```

Из 64 бит, отведённых на число, сами цифры числа занимают до 52 бит. Остальные 11 бит хранят позицию десятичной точки и один бит – знак. Так что если 52 бит не хватает на цифры, то при записи пропадут младшие разряды. Интерпретатор не выдаст ошибку, но в результате получится не совсем то число.

Если математическая операция не может быть совершена, то возвращается специальное значение `NaN` (Not-A-Number). Например, деление 0/0 в математическом смысле неопределенно, поэтому его результат `NaN`:

```
0/0; // NaN
2/"foo"; // NaN
```

Значение `NaN` – единственное в своем роде, которое не равно ничему, включая себя.

Деление на ноль. Практически во всех языках программирования деление на ноль возвращает ошибку. Но создатели JavaScript решили пойти математически правильным путем, ведь чем меньше делитель – тем больше результат. При делении на очень маленькое число должно получиться очень большое. В математическом анализе это описывается через пределы, и если подразумевать предел, то в качестве результата деления на 0 мы получаем «бесконечность», которая обозначается символом ∞ или, в JavaScript, «Infinity».

```
1/0; // Infinity
-1/0; // -Infinity
```

В JavaScript 2 нуля, как бы это странно ни звучало.

```
var a = 0 / -3; // -0
a.toString(); // "0"
var a = 0;
var b = 0 / -3;
a == b; // true
-0 == 0; // true
a === b; // true
-0 === 0; // true
```

Нужно это в приложениях, где, например, в качестве направления берется знак числа, и если подставить 0, то знак не поменяется на противоположный.

Массивы

В отличие от строго типизированных языков, массивы в JS могут содержать значения разных типов: объекты, строки, числа и даже другие массивы (многомерные массивы).

```
var a = [ 1, "2", [3] ];
a.length; // 3
a[0] === 1; // true
a[2][0] === 3; // true
```

Необязательно указывать размер массива, нужно только лишь объявить его:

```
var a = [ ];
a.length; // 0
a[0] = 1;
a[1] = "2";
a[2] = [ 3 ];
a.length; // 3
```

Если будет использоваться строковое значение, которое может быть приведено к 10-значному числу, то обработчик подумает, что используется числовое значение в качестве ключа.

```
var a = [ ];
a.length; // 0
a['13'] = 1;
a.length; // 14
```

В JavaScript есть методы для записи нового элемента в конец массива – `push()` и для удаления последнего элемента – `pop()`

```
var fruits = ["Яблоко", "Апельсин", "Груша"];
fruits.pop(); // удалили "Груша"
fruits; // Яблоко, Апельсин
fruits.push("Груша"); // удалили "Груша"
fruits; // Яблоко, Апельсин, Груша
```

Массив содержит численную индексацию, но также может иметь строковые ключи, тогда длина массива не будет учитывать элементы со строковыми значениями:

```
var a = [ ];
a[0] = 1;
a["foobar"] = 2;
a.length; // 1
a["foobar"]; // 2
a.foobar; // 2
```

7.2 Функции

Один из способ объявления функции выглядит так:

```
function add(x, y) {
     return x + y;
}
```

Объявляется функция `add` с двумя параметрами – `x` и `y`. Функция возвращает сумму этих параметров.

Вызов функции выполняется следующим образом:

```
add(2, 3); // 5
```

Если явно не указывать возвращаемое значение, как в этом примере, функция вернёт `undefined`.

Функции можно вызывать с любым количеством аргументов. Если параметр не передан при вызове, ему будет присвоено значение `undefined`.

```
function asIs(x) {
    return y;
}
asIs(y); // undefined
```

При объявлении функции необязательные аргументы, как правило, располагают в конце списка.

Зачастую для задания значения по умолчанию используют более короткую форму с использованием оператора `||`.

```
function myMin(a, b) {
    b = b || Infinity;
    return a < b ? a : b;
}
myMin(2, 7); // 2
myMin(13, 7); // 7
myMin(-13); // -13
```

Этот способ следует использовать с осторожностью. Если параметр может принимать значения, приводимые к `false`, они будут перезаписаны значением по умолчанию.

```
function getSalary(rate, days) {
    days = days || 22;
    return rate * days;
}
getSalary(3, 10); // 30
getSalary(1); // 22
getSalary(2, 0); // 44
```

В некоторых языках (например, в Python) при вызове функции можно передавать именованные аргументы. В JavaScript можно имитировать именованные аргументы с помощью литерала объекта.

```
BMI({ m: 60, h: 1.7 }) // 20.7
```

На самом деле в функцию передается один аргумент – объект, свойствами которого являются как бы именованные аргументы обработчика.

Объявление функции будет выглядеть следующим образом:

```
function BMI(params) {
    var h = params.h;
    return params.m / (h * h);
}
```

Однако у такого подхода есть несколько недостатков:

- аргументы нужно вызывать через точку (как свойства объекта);
- по сигнатуре функции нельзя понять, с какими параметрами она вызывается.

Такой подход не следует использовать повсеместно. На практике его обычно применяют, когда функция принимает большое количество необязательных конфигурационных параметров.

Обработчик может передать больше аргументов, чем определено в функции. Избыточные аргументы будут проигнорированы.

```
function myMin(a, b) {
    return a < b ? a : b;
}

myMin(7, 5, 1); // 5
```

Внутри функции доступна специальная переменная `arguments`, которая позволяет получить доступ ко всем переданным аргументам.

```
function myMin(a, b) {
    var min = a < b ? a : b;
    var c = arguments[2];

    return c < min ? c : min;
}

myMin(2, 3, 4); // 2
myMin(20, 3, 4); // 3
myMin(20, 30, 4); // 4
```

С его помощью можно объявить функцию, принимающую любое количество аргументов.

По историческим причинам `arguments` – не массив, а так называемый `array-like object`. Поэтому у него нет методов массива, таких как `forEach` или `map`. К счастью, есть способ преобразовать его к массиву с помощью следующей конструкции:

```
var args = [].slice.call(arguments);
```

В этом случае помогает динамическая природа языка JavaSctipt. Обработчику (исполнителю) понадобится метод `slice`, который может использоваться для копирования массива.

Следующий пример: необходимо найти минимальное значение из элементов, которые лежат в массиве `numbers`.

```
Math.min(3, 5, 2, 6); // 2
var numbers = [3, 5, 2, 6];
Math.min(numbers); // NaN
```

Есть функция `Math.min`, но она принимает несколько аргументов. Если просто передать массив в функцию – результат будет отличаться от наших ожиданий. В данном случае можно использовать метод `apply`.

Первым аргументом метод принимает так называемый контекст – объект, на который будет указывать ключевое слово `this` внутри функции. Контекст и ключевое слово `this` подробно рассматриваться не будут, поэтому передадим первым аргументом `null`.

В качестве второго аргумента метод принимает массив, элементы которого будут переданы в функцию.

```
var numbers = [3, 5, 2, 6];
Math.min.apply(null, numbers); // 2
```

Другой полезный метод – `bind`, он позволяет выполнить так называемое частичное применение функции. Частичное применение – это процесс передачи части аргументов функции, который возвращает другую функцию, принимающую оставшиеся аргументы (функцию меньшей арности).

Рассмотрим функцию возведения в степень – `Math.pow`. Она принимает два аргумента:

```
Math.pow(2, 4); // 16
Math.pow(2, 10); // 1024
```

С помощью метода `bind` можно передать только первый аргумент и получить функцию, возводящую в степень определённой число, в нашем случае двойку.

```
var binPow = Math.pow.bind(null, 2);
binPow(4); // 16
binPow(10); // 1024
```

Функции в JavaScript являются объектами первого класса. Это значит, что функции являются таким же полноценным типом данных, как число, строка или объект.

Функцию можно положить в переменную:

```
function add(a, b) {
    return a + b;
}

var sum = add;
sum(1, 3); // 4
```

Функцию можно передать в качестве аргумента другой функции:

```
function multiplyBy2(x) {
    return x * 2;
}

var numbers = [3, 5, 9];

numbers.map(multiplyBy2); // [6, 10, 18]
```

Наконец, функция может быть возвращена в результате выполнения другой функции:

```
function getAdd() {
    function add(a, b) {
        return a + b;
    }

    return add;
}

var myAdd = getAdd();
myAdd(1, 3); // 4
```

В самом начале главы был рассмотрен один способ создания функции. Он называется function declaration. Функция, определённая таким образом, создаёт-

ся на этапе интерпретации программы, то есть ещё до начала выполнения кода. Это приводит к эффекту, называемому hoisting, то есть всплытие или подъём. Проявляется он в том, что функцию можно вызывать до её объявления.

```
add(2, 3); // 5

function add(x, y) {
    return x + y;
}
```

Это происходит потому, что интерпретатор поднимает объявление функции над выполняемым кодом и исходный код приобретает следующий вид:

```
function add(x, y) {
    return x + y;
}
add(2, 3);
```

Второй способ объявления функции называется function expression и выглядит так:

```
var add = function (x, y) {
    return x + y;
};
```

В этом примере создается функция без имени, так называемая анонимная функция, которая кладется в переменную `add`.

Несмотря на общую схожесть с предыдущим вариантом, есть одно существенное различие – эта функция создаётся в момент выполнения кода, поэтому её нельзя использовать до определения.

Функция, объявленная с помощью function expression, необязательно должна быть анонимной. Можно указать для неё идентификатор.

```
var add = function hidden() {
    return typeof hidden;
}

add(); // function
```

В отличие от function declaration, такой идентификатор будет доступен только внутри функции, но не виден снаружи. Это может быть полезно при определении рекурсивных функций. Такие функции могут вызывать сами себя.

Оба варианта создания функции полностью допустимы. В общем случае можно использовать первый способ – function declaration. Этот способ обладает более простым синтаксисом, а также позволяет использовать эффект всплытия.

Есть и ещё один способ объявить функцию, но он встречается достаточно редко.

```
var add = new Function('x', 'y', 'return x + y');
add(2, 3);
```

Можно использовать конструктор `Function`. Этот способ может пригодиться, если требуется создать тело функции на этапе выполнения кода.

Область видимости переменной – это часть кода, в пределах которой эта переменная доступна. В Javascript область видимости переменной ограничивается функцией. При определении функции внутри другой функции создается новая область видимости. Переменная из внешней области видимости доступна во вложенной:

```
function greet() {
    var text = 'Привет';
    function nested() {
        console.log(text);
    }
    nested();
}
greet(); // 'Привет'
```

Если во внутренней области видимости объявить переменную с тем же именем, что и во внешней, она перекроет переменную из внешней области видимости. Этот эффект называется shadowing (затенение).

В отличие от других языков, блок не создаёт область видимости:

```
function greet() {
    if (true) {
        var text = 'Привет';
    }

    console.log(text); // 'Привет'
}
```

В этом примере срабатывает механизм всплытия, о котором говорилось ранее.

Интерпретатор поднимает объявление переменной к началу области видимости, то есть к началу функции:

```
function greet() {
    var text;

    if (true) {
```

```
            text = 'Привет';
        }

        console.log(text); // 'Привет'
    }
```

Иногда возникает необходимость искусственно создать область видимости. Например, если необходимо ограничить доступ к некоторым переменным.

Для этого можно использовать способ, называемый «немедленно вызываемой функцией» или IIFE (immediately-invoked function expression).

```
function foo() {
    (function () {
        var bar = 4;
    }());
    console.log(bar); // Reference Error
}
```

Исполнитель создает функцию, чтобы ввести новую область видимости. И сразу же после создания вызывает её. Скобки вокруг функции нужны для того, чтобы интерпретатор понял, что можно воспользоваться function expression.

Если попробовать немедленно вызвать функцию, объявленную с помощью function declaration, получится ошибка синтаксиса – язык не разрешает такую конструкцию. Чтобы исправить ситуацию, необходимо поместить любой символ перед ключевым словом function. Сделать это можно любым способом:

```
!function () {
}();

void function () {
}();
```

7.3 JavaScript в разработке веб-сайтов

Основным инструментом работы и динамических изменений на странице является DOM (Document Object Model) – объектная модель в виде дерева, используемая для XML/HTML-документов и доступная для изменения через JavaScript. Чтобы посмотреть визуальное представление DOM, нужно открыть панель разработчика в браузере (рис. 7.1).

Узел (Node) – это объект со следующими полями:

- тип узла;
- набор атрибутов;

- ссылка на левого/правого соседа;
- ссылка на родительский узел;
- ссылка на массив дочерних узлов.

Рис. 7.1 – Представление DOM в браузере

Каждый узел имеет тип. Типов очень много, самые важные среди них:

- ELEMENT_NODE=1 – элемент;
- TEXT_NODE=3 – текст;
- COMMENT_NODE=8 – комментарий;
- DOCUMENT_NODE=9 – документ (ссылка на тег-html);
- DOCUMENT_TYPE_NODE=10 – DOCTYPE;
- DOCUMENT_FRAGMENT_NODE=11 – фрагмент.

Каждый HTML-элемент становится узлом дерева с типом «элемент», а текст – узлом с типом «текст».

Из списка типов элементов можно понять, что DOM сохраняет информацию о тексте и о комментариях. Наличие типа DOCUMENT_NODE говорит о том, что DOM также хранит информацию о корне графа отдельно.

Рассмотрим следующий пример:

```
<html>
  <body>
    <form>
      <input type='text' placeholder='Имя'/>
      <input type='text' placeholder='Фамилия'/>
      <button type='submit'>Отправить</button>
```

```
    </form>
  </body>
</html>
```

На первый взгляд в данном примере шесть элементарных узлов и один элемент текста. Однако не учтены пробельные символы между элементами, а значит, на самом деле текстовых узлов девять.

То есть соседний элемент у `'Фамилия'` – не `'Имя'`, а некоторый текстовый элемент. Поэтому многие разработчики перед отправкой клиенту минифицируют свой `html`, т. е. удаляют все лишние символы.

С DOM непрерывно связаны объекты Window и Document. Объект Window является глобальным объектом в Javascript, который содержится в свойстве window. Он единственный в своем роде, все остальные объекты содержатся в свойствах или возвращаются методами этого объекта или объекта более низкого уровня. Объект Document предоставляет информацию о документе и содержится в свойстве document объекта Window.

Поиск элементов

Первая задача, которая может возникнуть у веб-разработчика – поиск нужного узла.

Для начала добавим элементам идентификаторы:

```
<html>
  <body>
    <form id='form'>
      <input id='name' type='text' placeholder='Имя'/>
      <input id='surname' type='text' placeholder='Фамилия'/>
      <button id='submit' type='submit'> Отправить </button>
    </form>
  </body>
</html>
```

Для поиска по ID используется метод объекта document – `getElementByID`. `document` – это глобальный объект, который позволяет взаимодействовать с содержимым страницы. Свойства и методы объекта `document` позволяют получить информацию о документе и влиять на отображение его содержимого.

```
var form = document.getElementById('form');
```

```
var name = document.getElementById('name');
```

Такой способ поиска подходит для быстрого обнаружения уникальных элементов.

Помимо поиска по идентификатору, в JavaScript реализована возможность поиска по названию элемента – метод `getElementByTagName`.

Рассмотрим наш код:

```
<html>
  <body>
    <form>
      <input type='text' placeholder='Имя'/>
      <input type='text' placeholder='Фамилия'/>
      <button type='submit'>Отправить</button>
    </form>
  </body>
</html>
```

Поиск элемента `form` и `input` осуществляется при помощи следующего кода:

```
var forms = document.getElementsByTagName('form');
var input = document.getElementByTagName('input');
```

Этот метод возвращает не массив элементов, а некоторый похожий на массив (array-like) объект NodeList. NodeList – это живая коллекция, то есть она реагирует на изменения DOM. Метод `getElementsByTagName` доступен не только для объекта `document`, но и для других элементов.

Среди преимуществ такого метода поиска можно отметить возможность найти все нужные теги за один запрос, а также поиск в контексте определенного элемента. Но стоит помнить, что такой поиск сопряжен со сложностью поддержки, поэтому используется редко.

Еще один способ поиска элементов – это поиск по классу с помощью метода `getElementsByClassName`. Чтобы рассмотреть этот способ, к элементам кода добавляются классы метода `getElementsByClassName`:

```
<html>
  <body>
    <form class='b-form'>
      <input class='b-form__name' type='text' placeholder='Имя'/>
      <input class='b-form__surname' type='text' placeholder='Фамилия'/>
```

```
          <button    class='b-form__submmit'    type='submit'>
Отправить </button>
        </form>
      </body>
    </html>

    var form = document.getElementsByClassName('b-form')[0];
    var name = form.getElementsByClassName('form__name')[0];
```

Преимущества такого метода:

- способ найти все элементы, удовлетворяющие одному CSS-селектору;
- возможность гибко размечать нужные элементы.

Конечно, этот способ поиска не такой быстрый, как `getElementById`, однако на фоне преимуществ это не так значимо.

Последний способ поиска элементов: по селектору.

Для поиска по DOM интерпретатор предоставляет два очень удобных метода:

1) `querySelector` – поиск первого элемента по селектору;
2) `querySelectorAll` – поиск всех элементов, удовлетворяющих селектору.

```
var form = document.querySelector('.b-form');
var items = form.querySelectorAll('input');
```

Это наиболее универсальный метод поиска, хоть он и медленнее остальных.

Помимо глобального поиска, можно воспользоваться поиском близлежащих элементов. Например, чтобы найти родительский элемент, в JavaScript существуют свойства: `parentNode` и `parentElement`. Первое свойство ищет родительский узел произвольного типа, а второе – типа, отличающегося от комментария, текста или DOCTYPE.

```
var formsName = document.querySelector('.b-form__name');
var form = formsName.parentElement;
```

Методом `closest` можно воспользоваться, чтобы найти ближайшего родителя (включая самого себя), удовлетворяющего селектору. Если ни родитель, ни сам элемент не подходят под условие, то вернется `null`.

```
<div class='b-page'>
  <div class='controller'></div>
</div>
<div class='b-page'>
```

```
  <div class='another-controller'></div>
</div>
var controller = document.querySelector('.controller');
var page = controller.closest('.b-page');
```

Для поиска соседей существуют четыре свойства:

- `nextSibling` – возвращает предыдущего соседа (любого типа);
- `nextElementSibling` – возвращает предыдущий соседний узел-элемент;
- `previousSibling` – возвращает следующего соседа (любого типа);
- `previousElementSibling` – возвращает последующий соседний узел-элемент.

Для поиска потомков есть следующие свойства:

- `firstChild` – возвращает ссылку на первый дочерний узел;
- `firstElementChild` – возвращает;
- `lastChild` – возвращает ссылку на последнего ребенка;
- `lastElementChild`;
- `childNodes` – возвращает `live-array` со списком детей;
- `children`.

```
var form = document.querySelector('.b-form__name');
var name = form.firsElementChild;
var submit = form.lastElemenChild;
```

Чтобы проверить, подходит ли элемент DOM под селектор, используется метод `match`.

```
<div class='b-page'>
  <div class='controller current'></div>
</div>
<div class='b-page'>
  <div class='another-controller'></div>
</div>
var controller = document.querySelector('.controller');
controller.matches('.controller.current')
```

Работа с атрибутами

В некоторых случаях (например, для настройки работы некоторых HTML-элементов) необходимо получить значение атрибутов.

Для чтения данных из атрибутов нужно воспользоваться методом `getAttribute()`.

```
<html>
  <body>
    <form class='b-form'>
      <input class='b-form__name' type='text' placeholder='Имя'/>
      <input class='b-form__surname' type='text' placeholder='Фамилия'/>
      <button class='b-form__submmit' type='submit'>Отправить</button>
    </form>
  </body>
</html>
var name = document.querySelector('.b-form__name');
name.getAttribute('placeholder');
```

Для изменения значения атрибута используется метод `setAttribute()`

```
var name = document.querySelector('.b-form__name');
name.setAttribute('placeholder', 'Имя Вашего соседа');
```

И, наконец, для удаления атрибута также существует свой метод: `removeAttribute()`

```
var name = document.querySelector('.b-form__name');
name.removeAttribute('placeholder');
```

Разработчики веб-стандартов зарезервировали пространство имен для создания пользовательских атрибутов. Такие атрибуты начинаются с префикса `'data-'`.

```
<div class='page' data-server-time='01/09/2007'>
</div>
var page = document.querySearch('page');
page.getAttribute('data-search-time');
```

Для работы с пользовательскими атрибутами создано специальное свойство `dataset`, в котором для упрощения доступа к атрибуту используется его запись без префикса: `'data-'`.

Изменение класса

При создании веб-страниц множество компонентов могут иметь несколько визуальных состояний: открытое или закрытое модальное окно, активный элемент меню, в текстовое поле введены неверные данные и т. д.

Для управления визуальным состоянием можно воспользоваться следующими способами:

- перейти на страницу с уже измененным состоянием (устаревший способ);
- скрыть часть элементов из `html`;
- изменить значение атрибута `style` у нужных элементов;
- изменить значение атрибута `class`.

Остановимся подробнее на последнем способе. Так как `class` – это в пер-вую очередь атрибут, то можно попробовать изменить его через метод `setAttribute()`:

```
var track = document.querySelector('.track');
track.setAttribute('class', 'track track_lovely_yes');
```

Однако стоит помнить, что при работе с атрибутами Вы имеете дело со строкой, а значит, все операции по работе с классами нужно писать самим: удаление, замена, добавление. Можно поменять класс через свойство `className`, но и здесь возникает такая же ситуация, как и с методом `setAttribute()`.

```
var track = document.querySelector('.track');
track.className = 'track track_lovely_yes';
```

Следующий способ более удобный. В свое время разработчики вебстандарта решили уделить больше времени созданию удобного интерфейса для работы с классами, и так на свет появилось свойство `classList`.

Чтобы добавить новый класс элементу, нужно просто воспользоваться методом `add`:

```
var track = document.querySelector('.track');
track.classList.add('track_lovely_yes');
track.className;
```

Для удаления класса используется метод `remove`:

```
var track = document.querySelector('.track');
track.classList.remove('track_lovely_yes');
track.className;
```

Для замены класса в элементе создан метод `toggle`:

```
var track = document.querySelector('.track');
track.classList.toggle('hidden', 'shown');
track.className;
```

И наконец, чтобы проверить, есть ли данный класс у элемента, нужно воспользоваться методом `contain`:

```
var track = document.querySelector('.track');
track.classList.contain(‘track');
```

Изменение DOM

В самом начале развития веб-индустрии, чтобы изменить страницу нужно было перейти по ссылке на уже измененную страницу. Но со временем Веб стал динамичнее, и при работе с сайтом DOM может сильно изменяться: открываются модальные окна, обновляются страницы без перезагрузки данных, подгружаются все новые и новые изображения в галерее. Поэтому важным навыком является навык изменения (правки) DOM.

Как и в любом другом изменении, правка DOM не обходится без создания новых элементов. У объекта страницы есть метод `createElement`, создающий новый узел, название которого передается в параметре метода:

```
var track = document.createElement('div');
track.className = 'track'
```

Когда создается новый элемент, то он создается в «вакууме» и нигде не отображается. Чтобы он стал виден на странице, его нужно добавить к одному из элементу, который уже отображен на странице. Существует несколько способов сделать это:

- добавить последним дочерним элементом к какому-либо существующему с помощью метода `appendChild()`:

```
var track = document.createElement('div');
var label = document.createElement('div');
track.appendChild(label);
```

- вставить элемент в коллекцию дочерних элементов сразу перед нужным:

```
var track = document.createElement('div');
var label = document.createElement('div');
var duration = document.createElement(‘div’);
track.appendChild(duration);
track.insertBefore(label, duration);
```

- просто заменить старый элемент на новый:

```
var track = document.createElement('div');
var label = document.createElement('div');
track.appendChild(track);
track.replaceChild(document.createTextNode('Ваша любимая песня'), label);
```

Для создания текста существует отдельный метод `createTextNode` – из названия понятно, что создастся узел дерева с типом «текст»:

```
var text = document.createTextNode('Ваша любимая песня');
```

Все эти способы кажутся громоздкими, поэтому есть свойство `innerHTML`, которое позволяет проще создавать элементы.

`innerHTML` позволяет получить HTML-код дочерних элементов в виде строки и присвоить новый HTML:

```
var html = document.innerHTML;
var track = document.createElement('div');
track.innerHTML = "<div class='label'>Marilyn Manson</div><div class='star'>5</div>"
```

Использование этого свойства сопряженно с опасностью уязвимости (например XSS), т. к. не экранирует HTML. Но есть похожее свойство `innerText`, которое позволяет заменить все содержимое элемента на экранированный текст, т. е. если вставить код элемента, то и на экране появится код.

```
var track = document.createElement('div');
track.innerText = 'Пользовательский ввод';
```

И наконец, самый последний способ вставить элемент – это воспользоваться уже готовым шаблонизатором, которых существует огромное множество.

Иногда может возникнуть ситуация, когда нужно добавить не один элемент, а сразу несколько. Для этого в JavaScript существует несколько механизмов:

1. Добавление в цикле методом `appendChild()`

```
var track = document.querySelector('.track');
for(var i = 0; i < 10; i++) {
    var label = document.createElement('div');
    track.appendChild(label)
}
```

2. Создается элемент, в него вставляются новые элементы, и вся конструкция добавляется в DOM:

```
var track = document.querySelector('.track');
  var container = document.createElement('div');

  for(var i = 0; i < 10; i++) {
     var label = document.createElement('div');
     container.appendChild(label);
```

```
}
track.appendChild(container);
```

3. Использование предыдущего способа имеет свой минус: добавляется лишний элемент. Для решение этой проблемы есть специальный тип Node – DOCUMENT_FRAGMEN, который исчезает в DOM при присоединении к отображаемым элементам. Для создания `DocumentFragment` нужно воспользоваться методом `createDocumentFragment` у `document`

```
var track = document.querySelector('.track');
var container = document.createDocumentElement();
for(var i = 0; i < 10; i++) {
    var label = document.createElement('div');
    container.appendChild(label);
}
track.appendChild(container);
```

Умения вставлять элемент мало, нужно еще уметь его удалить. Удаление производится с помощью метода `removeChild()`.

```
var track = document.querySelector('.track');
var label = track.querySelector('.label');
track.removeChild(label);
```

События

Со временем HTML-страницы превратились из статичных научных статей в интерактивные площадки. Для взаимодействия с пользователями браузер оснастили механизмом, способным сообщать клиентскому коду о происходящем на странице, например, получить информацию о том, что пользователь нажал на кнопку, страница загрузилась и т. д.

Чтобы подписаться на событие, в JS есть несколько способов:

1. Самый старый и самый плохой способ – воспользоваться HTML-атрибутом. Для всех основных событий имя атрибута: on + имя события (`onclick`, `onblur`, ...). Обработчик события записывается в виде строки в разметку или же записывается имя глобальной функции

```
<button onclick='onClickByButton()'>First</button>
<button    onclick='(function(){alert(2);})()'>    Second
</button>
function onClickByButton() {
    alert(1);
}
```

2. В JS у всех dom-element есть свойства on + имя события. Чтобы подписаться на событие нужно присвоить функцию обработчика атрибуту.

Чтобы отписаться от события, нужно присвоить атрибуту значение null.

```
<button >First</button>
var button = document.querySelector('button');

button.onclick = function () {
    console.log('Click!!!');
}
```

3. Разработчики веб-стандарта предлагают пользоваться методом addEventListener, который есть у всех dom-element. Метод принимает три аргумента: eventName – имя события, listener – Ваш обработчик, useCapture – стадия активации обработчика.

```
<button >First</button>
    var button = document.querySelector('button');
    var callback = function (event) {
    alert('Click')
}
    button.addEventListener('click', callback, false);
```

Обработчик событий принимает объект event, который в себе содержит всю информацию о произошедшем событии:

- target – на каком элементе сработало событие;
- currentTarget – на каком элементе в данный момент обрабатывается событие;
- type – название события;
- и многие другие свойства.

У метода addEventListener есть напарник removeEventListener(), который имеет такую же сигнатуру и позволяет отписаться от события.

```
<button >First</button>
var button = document.querySelector('button');
var callback = function () {
    alert('Click')
};

button.addEventListener('click', callback, false);
button.removeEventListener('click', callback, false);
```

Важно обратить внимание на то, что при отписке от события нужно передать ту же функцию.

Следующий код не будет работать:

```
var button = document.querySelector('button');
button.addEventListener('click', function () {
        alert('Click')
    }, false);
button.removeEventListener('click', function () {
        alert('Click')
    }, false);
```

Отписка может понадобиться при изменении состояния приложения (например, кнопка заблокирована).

Механизм работы события таков:

1) кто-то создал событие (например, нажал мышкой по кнопке);
2) получаем список родителей;
3) начинаем стадию захвата;
4) бежим в цикле по родителям начиная с самого дальнего;
5) вызываем обработчики, которые подписаны на стадию захвата;
6) начинаем стадию цели;
7) вызываем все обработчики, которые подписаны на любую стадию;
8) начинаем стадию всплытия;
9) бежим в цикле по родителям начиная с самого ближнего;
10) вызываем обработчики, которые подписаны на стадию всплытия.

Что можно узнать из механизма?

1) `addEventListener` последним параметром принимает стадию захвата или всплытия;
2) список элементов, на которых будут вызваны обработчики фиксируется при возникновении события;
3) события на стадии захвата происходят чуть быстрее;
4) на цели обработчики вызываются в произвольном порядке.

Бывает ситуация, когда приходит осознание, что вызов дальнейших обработчиков не нужен. Тогда в помощь можно использовать метод `stopPropagation`, который есть у объекта `event`. Стоит помнить, что этот метод полностью останавливает обработку события.

```
<div class='container'>
    <div class='block'></div>
</div>
```

```
var container = document.querySelector('.container');
var block = document.querySelector('.block');
block.addEventListener('click', function (event) {
    console.log(1);
    event.stopPropagation();
}, false);
container.addEventListener('click', function (event) {
    console.log('этот текст никогда не распечатается');
}, false);
```

Некоторые HTML-элементы имеют некоторое действие по умолчанию, при возникновении определенных событий. Например, ссылка при клике осуществляет переход по ссылке.

Чтобы этого избежать, у объекта `event` есть метод `preventDefault`.

```
<a class='link' href='yadnex.ru'>
    какая классная ссылка
</a>
var link = document.querySelector('.link');

link.addEventListener('click', function (event) {
    event.preventDefault();
}, false);
```

Контрольные вопросы по главе 7

1. Какие типы данных существуют в JavaScript?
2. Как объявляется функция?
3. В чем заключается суть DOM-отображения?
4. Как осуществляется поиск по соседним элементам?
5. Какие типы узлов DOM существуют?

Заключение

Создание веб-сайтов – сложный и многогранный процесс. Каждый этап включает в себя работу нескольких разноплановых специалистов и их взаимодействие между собой. В зависимости от того, как построены коммуникационные процессы между ними, можно предсказать возможные проблемы и сложности.

Роль верстальщика в создании сайта немаловажна, ведь именно он – связующее звено между абстрактным мышлением дизайнера и прагматично-логическим мировоззрением бэкенд-разработчика. Умение найти правильные инструменты и адаптировать макет под требования заказчика – замечательное качество, которое нужно воспитывать в себе всем тем, кто хочет связать свою жизнь с разработкой веб-сайтов.

В курсе рассмотрены общие вопросы разработки веб-документов и некоторые моменты, которые возникают в профессиональной деятельности веб-разработчика. Для более серьезного овладения тематикой необходимо постоянное самообразование и наличие практики в виде реальных задач.

Помимо этого стоит помнить, что сфера веб-технологий – очень динамично развивающаяся отрасль. Настолько, что информация, актуальная еще два года назад, уже сегодня может стать безнадежно устаревшей, поэтому чтение новейшей документации и общение на профессиональных форумах также способствуют профессиональному росту фронтенд-разработчика.

Литература

1. Краткая история сети Интернет [Электронный ресурс] / Барри М. Лейнер, Винтон Дж. Серф, Дэвид Д. Кларк, Роберт Е. Кан, Леонард Клейнрок, Даниэл С. Линч, Джон Постел, Ларри Дж. Робертс, Стивен Вулф // Internet Society. – Режим доступа: http://u.to/Sha_Bw (дата обращения: 16.05.2017).
2. Background. HTML 5.1 [Electronic resource] // W3C Recommendation (1 November 2016). – URL: https://www.w3.org/TR/html51/introduction.html#background (дата обращения: 16.05.2017).
3. HTML 5.1 [Electronic resource] // W3C Recommendation (1 November 2016). – URL: https://www.w3.org/TR/html51/ (дата обращения: 16.05.2017).
4. HTML-синтаксис. HTML 5.1 [Электронный ресурс] // Рекомендации W3C от 28 октября 2014 г. – Режим доступа: http://spec.piraruco.com/html5/syntax.htm#cdata-rcdata-restrictions (дата обращения: 16.05.2017).
5. Порядок элементов set-ов в Python [Электронный ресурс] // Блог Павла Патрина. – Режим доступа: https://pavelpatrin.com/ (дата обращения: 16.05.2017).
6. Поток документа [Электронный ресурс] // htmlbook.ru. – Режим доступа: http://htmlbook.ru/samlayout/blochnaya-verstka/potok-dokumenta (дата обращения: 16.05.2017).
7. CSS Box Model [Electronic resource] // w3schools.com. – URL: https://www.w3schools.com/css/css_boxmodel.asp (accessed 16.05.2017).
8. HTML Block and Inline Elements [Electronic resource] // w3schools.com. – URL: https://www.w3schools.com/html/html_blocks.asp (дата обращения: 16.05.2017).
9. Контекст наложения (stacking context) [Электронный ресурс] // Mozilla Developer Network. – Режим доступа: https://developer.mozilla.org/ru/docs/Web/CSS/CSS_Positioning/Understanding_z_index/The_stacking_context (дата обращения: 16.05.2017).

Глоссарий

CSS (англ. *Cascading Style Sheets*) – каскадные таблицы стилей.

DTD (*Document Type Definition*) – формальная схема спецификации языка, предназначенная для автоматического разбора браузером или программой (например, валидатором документов).

HTML (англ. *HyperText Markup Language*) – язык гипертекстовой разметки документа.

IT (англ. *information technology*) – информационные технологии.

W3C (англ. *World Wide Web Consortium*) – Консорциум Всемирной паутины.

WHATWG (англ. *Web Hypertext Application Technology Working Group*) – сообщество людей и компаний, заинтересованных в развитии Интернета.

Анимация – последовательный показ (слайд-шоу) заранее подготовленных графических файлов, а также компьютерная имитация движения с помощью изменения (и перерисовки) формы объектов или показа последовательных изображений с фазами движения.

Векторная графика – это изображения, созданные при помощи математических описаний элементарных геометрических объектов.

Кодировка – таблица, где каждому символу сопоставляется последовательность нулей и единиц (битов).

Поток документа – это воспроизведение текста, написанного на западных языках, слева направо и сверху вниз, и обычная схема компоновки текста традиционных HTML-документов. Для восточных языков направление потока может меняться.

Тег – имя элемента, заключенное в угловые скобки < >. Теги бывают открывающимися и закрывающимися.

ЦЕРН (фр. *CERN* – Conseil Européen pour la Recherche Nucléaire) – Европейский совет по ядерным исследованиям.